# COMP30191
# The Theory of Games and Game Models

Andrea Schalk
A.Schalk@cs.man.ac.uk
School of Computer Science
University of Manchester

August 7, 2008

## About this course

This is an introduction into the theory of games and the use of games to model a variety of situations. While it is directed at third year computer science students it is also suitable for students of mathematics, and possibly of physics. It contains some proofs but does not require advanced knowledge of mathematics. The course goes beyond what is classically understood as *game theory* (the basics of which are covered in Sections 1 and 2) by making connections with computer programs for games such as Chess, and using games to model scenarios in sociology or biology. This course has been taught in the past as CS3191 and CS3192.

### What this course is about

Games have been used with great success to describe a variety of situations where one or more entities referred to as *players* interact with each other according to various rules. Because the concept is so broad, it is very flexible and that is the reason why applications range from the social sciences and economics to biology and mathematics or computer science (games correspond to proofs in logic, to statements regarding the 'fairness' of concurrent systems, they are used to give a semantics for programs and to establish the bisimulation property for processes). As such the theory of games has proved to be particularly fruitful for areas which are notoriously inaccessible to other methods of mathematical analysis. There is no set of equations which describes the goings-on of the stock-market (or if there is, it's far too complicated to be easily discoverable). Single transactions, however, can be described using (fairly simple) games, and from these components a bigger picture can be assembled. This is a rather different paradigm from the one which seeks to identify forces that can be viewed as the variables of an equation. Games have also been successfully studied as models of conflict, for example in biology as well as in sociology (animals or plants competing for resources or mating partners). In particular in the early days of the theory of games a lot of work was funded by the military.

When playing games it is assumed that there is some sort of punishment/reward system in place, so that some outcomes of a game are better for a player than others. This is typically described by assigning numbers to these outcomes (one for each player), and it is assumed that each player wishes to maximise his number. This is what is meant when it is stipulated that *all players are assumed to behave rationally.* Games are then analysed in order to find the actions a given player should take to achieve this aim.

This is what is referred to as a *game theoretic* analysis, but it should be pointed out that there are other ways of looking at the behaviour of players. Sociologists, psychologists and political scientists, for example, are more likely to be interested what people actually *do* when playing various games, not in what they *should* be doing to maximize their gains. The only way of finding out about people's behaviour is to run experiments and watch, which is a very different activity from the one this course engages in.

To give a practical example, assume you are given a coin and, when observing it being thrown, you notice that it shows heads about 75% of the time, and tails the remaining 25%. When asked to bet on such a coin, a player's chances are maximized by betting on heads *every single time.* It turns out, however, that people typically bet on heads 75% of the time only!

Economists, on the other hand, often are interested in maximizing gains under the assumption that everybody else behaves 'as usual', which may lead to different results than if one assumes that all players play to maximize their gains. Provided the 'typical' behaviour is known, such an analysis can be carried out with game-theoretic means.

Depending on the size of the game in question, this analysis will take different forms: Games which are small enough allow a complete analysis, while games which consist of a great many different positions (such as Chess or Go) can not be handled in that way.

In this course we examine games of different sizes and appropriate tools for analysing them, as well as a number of applications.

## Organization

This course has a webpage at

`http://www.cs.man.ac.uk/~schalk/3192/index.html`.

where all information not contained in the notes can be found.

These notes form the foundation for the course, and students are expected to *work through them in their own time.* To help them with this task there will be one **question sessions** per week which serve two purposes,

- to give students the opportunity to ask questions regarding the material they were set to read in the past fortnight;
- to give the lecturer the opportunity to introduce the more technical concepts before setting the reading for the coming fortnight.

Part of the notes is a collection of **exercises** that aim to prepare students for solving the 'problem solving' part of the exam. There are unassessed as well as assessed exercises, and the latter form the **assessed course work**. The unassessed exercises come in a part (a) and a part (b)—the purpose of the second part is to provide additional examples for practice during the revision period.

In weeks A student learning is supported by **examples classes**. In these the students get help with solving the exercises, including the assessed course work. These examples classes take place in the lecture slots with the lecturer and demonstrators present to ensure an instructor/student ration of about 1 to 10. In the last week of term the assessed course work is marked.

Marking takes place as follows: Students present their workings of the assessed exercises and a demonstrator or lecturer asks them to explain these workings. If the students can do so satisfactorily, and if their workings do not contain more than minor errors, then they will get full marks for an attempted exercise. Exercises where students *cannot give a coherent explanation of what they have done* will result in *zero marks* for that exercise. The purpose of these measures is to avoid students merely copying solutions.

The timetable for scheduled activities is as follows:

| When | What | Where | Who |
|---|---|---|---|
| Week 1 | Introduction | Mo 4.00pm in 1.1 | All |
| Weeks 2-5, 7-12 | Question sessions | Mo 4.00pm in 1.1 | All |
| Weeks A (3, 5, 8, 10) | Examples classes | We 9.00am in 2.19 | Surnames A–L |
| | | We 10.00am in 2.19 | Surnames M–Z |
| Week 12 | Assessment | We 9.00am in 2.19 | Surnames A–L |
| | | We 10.00am in 2.19 | Surnames M–Z |

Set reading.

| Week | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Theme | Games | Strategies | Equilibria | Non 0-sum | Mixed strats | Solving |
| Pages | 1–14 | 14–22 | 23–33 | 34–42 | 42–48 | 48–59 |
| Week | 7 | 8 | 9 | 10 | 11 | 12 |
| Theme | Models | Evolution I | Evolution II | Medium games | Large games | Catch-up |
| Pages | 60–75 | 76–88 | 88–97 | 98–110 | 111–130 | |

The timetable for the examples classes is as follows:

| Examples Class | Week | Exercises | Assessed Exercises |
|---|---|---|---|
| 1 | 3 | 1–7 | 1(a), 1(b), 1(c), 2 |
| 2 | 5 | 7–13 | 1(b), 1(c), 3(a) |
| 3 | 8 | 13–18 | 1(d), 3(a), 3(b), 4 |
| 4 | 10 | 19–24 | 4, 5 |

The fifth examples class in week 12 is reserved for marking the assessed exercises.

The assessed course work provides 20% of the final mark for the course, with 80% coming from an **exam**. All the material in the notes is examinable, including the exercises. Past exam papers are available online, and links are provided at the course's webpage. On the same webpage feedback on previous years' exam performance can be found.

The reason for changing the format from the traditional lecture course is the experience I have gained over seven years of teaching this course. The material is fairly abstract, and to learn such material, an active mode of learning is required, where students try to solve exercises by themselves (rather than just 'consuming' what is being presented by somebody else). Or, as somebody else put it, mathematics is not a spectator sport. Watching me explain it all in lectures would only be a small first step towards understanding the material—my expectations are made much clearer in the new format. I used to have the more outgoing students asking for additional meetings in the second half of term, whereas the rest never received any help in a smaller group setting. Again the new format gives that chance to every student on the course.

I would appreciate the readers' help in order to eliminate any remaining errors and to improve the notes. If you spot something that seems wrong, or doubtful, or if you can think of a better way of explaining something then please let me know by sending email to `A.Schalk@cs.man.ac.uk`. I will keep a list of corrigenda available on the course's webpage.

I would like to thank David MacIntosh, Isaac Wilcox, Robert Isenberg, Roy Schestowitz, Andrew Burgess, Xue Qing Huang, Cassie Wong, Florian Stürmer, Mark Daniel, Steve Hughes, Kevin Patel, and Mousa Shaikh-Soltan from previous years' courses for helping me improve the course material.

### Literature

As far as I know, this course is unique in that I don't think any other computer science department has a comparable course unit. Hence there is no one text book which covers everything I lecture on. Within the text I give references for specific topics to allow you to read up on something using a source other than the notes, or for further reading if something should find your particular interest.

# Contents

# 1 Games and strategies

## 1.1 So what's a game?

In every-day language, 'game' is quite a common word which seems to apply to a variety of activities (a game of Chess, badminton, solitaire, Poker, quake), and if we consider the act of 'playing' as something that applies to a game, then we get an even more varied range (playing the guitar, the lottery, the stock market). The latter set certainly takes us far beyond what we will consider in this course. The members of the former set all have something in common: here 'game' applies to the interaction of entities, the *players*, according to predefined rules.[1]

For our purposes, we will restrict the notion further. We assume that at any given time, it is the turn of precisely one player who may choose among the available *moves* (which are given by the rules of the game).[2] This allows us to present each game via a tree which we refer to as the *game tree*: By this convention, it is one player's turn when the game starts. We use the root of the tree to represent the start of the game, and each valid move this player might make is represented by a branch from the root to another node which represents the new state. Each node should be labelled with the Player whose turn it is, and there has to be a way of mapping the branches to the moves of the game. We say that a position is *final* when the game is over once it has been reached, that is when there are no valid moves at all from that position. The final positions drawn in Figure 1 are those which have a comment regarding their outcome (one of '$X$ wins', '$O$ wins' and 'Draw'). This Figure should demonstrate that using game trees to describe games is fairly intuitive.

**Example 1.1 Noughts and Crosses.** Part of a game tree for Noughts and Crosses (also known as Tic-Tac-Toe) is given in Figure 1.

At first sight, the game tree in Example 1.1 has fewer opening moves than it should have. But do we really lose information by having just the three shown? The answer is no. There are nine opening moves: $X$ might move into the middle square, or he might move into one of the four corners, or into one of the four remaining fields. But for the purposes of the game *it does not make any difference* which corner is chosen, so we replace those four moves by just one, and similar for the remaining four moves. We say that we *make use of symmetry considerations* to cut down the game tree. This is commonly done to keep the size of the tree manageable.

It is also worth pointing out that a game tree will distinguish between positions that might be considered the same: There are several ways of getting to the position in the third line of Figure 1. Player $X$ might start with a move into the centre, or a corner, and similarly for Player $O$. Hence this position will come up several times in the game tree. This may seem inefficient since it seems to blow up the game tree unnecessarily, but it is the accepted way of analysing a game. If we allowed a 'game graph' (instead of a game tree) then it would be more difficult to keep track of other things. We might, for example want to represent a Chess position by the current position of all the pieces on the board. Then two positions which 'look' the same to an observer would be the same. However, even in Chess, that information is not sufficient. For example, we would still have to keep track of whose turn it is, and we

[1] Granted, in the case of solitaire we have only one player, so 'interaction' is not a particularly apt description, unless we allow for the possibility that that player might interact with him- or herself.

[2] This will still allow us to model situations where the players move simultaneously, although that treatment might appear slightly contrived. Nonetheless the advantages of this way of thinking outweigh the disadvantages.

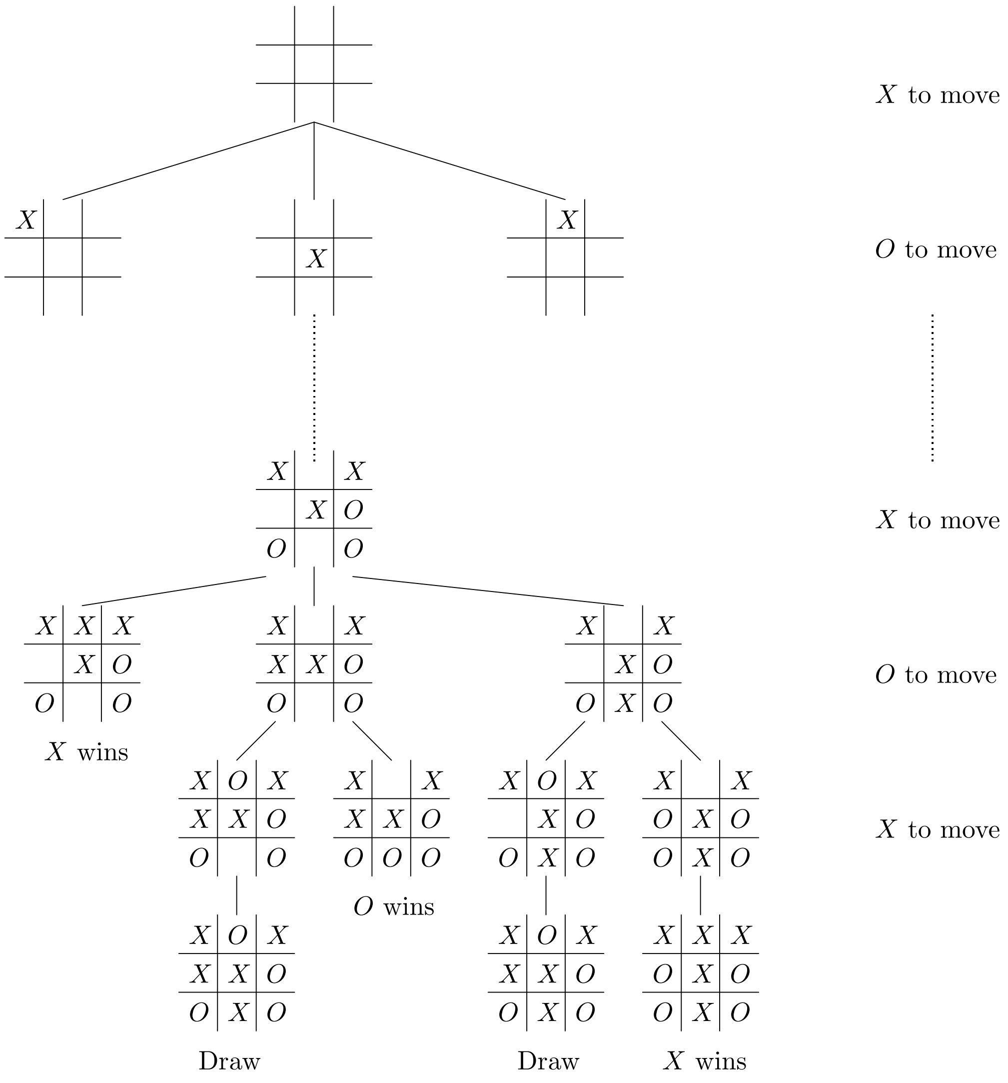



Figure 1: Part of a game tree for Noughts and Crosses

would have to know which of the two sides is still allowed to castle. Hence at least in Chess some information (beyond a picture of the board) is required to determine the valid moves in a given position.

With the game tree, every position (that is, node of the tree) comes with the entire history of moves that led to it. The reason for this is that in a tree there is precisely *one* route from the root to any given node, and in a game tree that allows us to read off the moves that led to the given position. As a consequence, when following moves from the start node (root), possibilities may divide but they can never reunite. In that sense, the game tree makes the *maximal number of distinctions* between positions. This allows us to consider a larger number of *strategies* for each player.

**Question 1** (a) Could you (in principle, don't mind the size) draw a game tree for Backgammon, or Snakes-and-Ladders? If not, why not?

(b) Could you draw a game tree for Paper-Stone-Scissors? If not, why not?

(c) Consider the following simple game between two players: Player 1 has a coin which he hides under his hand, having first decided whether it should show head or tail. Player 2 guesses which of these has been chosen. If she guesses correctly, Player 1 pays her 1 quid, otherwise she has to pay the same amount to him. Could you draw a game tree for this game? If not why not?

There are some features a game might have which cannot be presented straight-forwardly in such a game tree:

- **Chance.** There might be situations when moves depend on chance, for example the throwing of a die, or the drawing of a card. In that case, the control over which move will be made does not entirely rest with the player whose turn it is at the time. From time to time we will allow elements of chance.
- **Imperfect information.** The players may not know where exactly in the game tree they are (although they have to be able to tell which moves are valid at any given time!). This often occurs in card games (which also typically contain elements of chance), where one player does not know what cards the other players hold, or when the game allows for 'hidden' moves whose consequences are not immediately clear. For the time being we will concentrate on games of *perfect information*.
- **Simultaneous moves.** We will take care of those by turning these into moves under imperfect information.

We will treat these complications later; they can be incorporated into the formal framework we are about to present without great problems.

We say that a game is of **perfect information** if at any point, both players know precisely where in the game tree they are. In particular, each player knows which moves have occurred so far. We will only look at these games for a little while, and there are quite a few results in this course which *only* hold for these kinds of games.

**Definition 1** *A* **game** *is given by*

- *a finite set of players,*
- *a finite*[3] *game tree,*
- *for each node of the tree, a player whose turn it is in that position and*
- *for each final node and each player a pay-off function.*[4]

[3]In this course we will only consider games which are finite in the sense that there is no infinite path (How long would it take to play through such a game?), and that at every position, there are only finitely many moves a player might make. The reason for this latter restriction is that some knowledge of Analysis is required to examine games with infinitely many positions.

[4]It will take us until Section 1.4 to explain this requirement.

We can view a game tree as a representation of the decision process that has to be followed when playing a game. The positions where a given player is to move are the *decision points* for that player (who has to make a choice at those points). The game tree provides us with a convenient format for keeping track of those and their dependency on each other.

Often the games we consider will have just two players, these games are known as *two person games*. We will usually refer to them as Player 1 (who makes the first move) and Player 2, and to make it easier to talk about them we'll assume that Player 1 is male while Player 2 is female. (However, there are examples and exercises where the two players are given names, and sometimes the female player will move first in those.)

**Example 1.2 Chomp.** Consider the following game. Two players have a bar of chocolate with $m \times n$ squares. The square in the top left corner is known to be poisonous. The players play in turn, where the rules are as follows: A player chooses one of the (remaining) squares of chocolate. He then eats this together with all the pieces which are below and/or to the right of the chosen one. In other words, he removes the entire rectangle of chocolate whose top left corner is given by the piece he chose. (Obviously) the player who has to eat the poisonous piece loses.

Figure 2 shows a game tree for $2 \times 2$-Chomp.

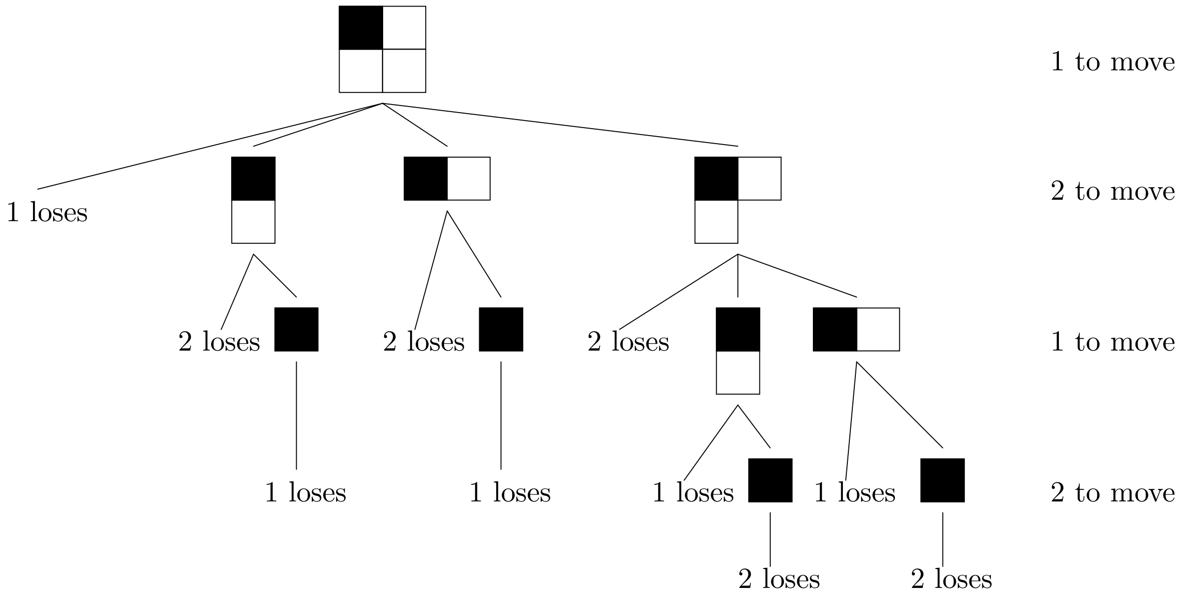



Figure 2: A game tree for $2 \times 2$-Chomp

**Exercise 1** (a) **Nim.** This is a game between two players who have a (finite) number of piles of matches in front of them. A valid move consists of choosing a pile and removing as many matches from that as the player chooses as long as it is at least one. The player who has to take the last match loses. (There is also a version where the player who takes the last match wins.) Draw a game tree for Nim with two piles of two matches each. This is known as $(2, 2)$-Nim. (If we had one pile of one match, two piles of two matches and one pile of three matches, it would be $(1, 2, 2, 3)$-Nim.)

(b) Draw a game tree for $2 \times 3$-Chomp.

**Question 2** Why are the games discussed so far so boring? Can you think of ways of making them more interesting?

Most of the examples of 'game' from above can be made to fit into this definition. In practice, however, we often describe games in ways other than by giving an explicit game tree. The most compelling reason for that is that for most interesting games, such a tree would be far too big to be of any practical use. For the game of Chess, for example, there are 20 opening moves for White (the eight pawns may each move one or two fields, and the knights have two possible moves each), and as many for Black's first move. Hence on the second level of the game tree we already have $20 \times 20 = 400$ positions (note how the possibilities are *multiplied* by each other). Therefore most game rules are specified in a way so as to allow the players to derive the valid moves in *any* given position. This makes for a much more compact description. This also shows that the game tree is a *theoretic device* which allows us to reason about a game, but which may not be of much use when playing the game.

A (complete) **play** of a game is one path through the game tree, starting at the root and finishing at a final node. The game tree makes it possible to read off all possible plays of a game.

**Question 3** How many plays are there for Noughts and Crosses? If you can't give the precise number, can you give an upper bound?

For this course, we will distinguish between small, medium, and large games, depending on the size of the game tree. These distinctions are somewhat fuzzy in that we do not set a definite border for these sizes. They are driven by practical considerations: Dealing with games in any of these classes requires different methods. Section 2 describes techniques appropriate for small games; later we will explore what works for larger games, but this part is not covered in the notes. The borders between these categories of games depend on the support we have for solving them; with ever faster machines with ever more memory, the class of truly large games has been steadily moving further out. Examples of these include Chess and Go. This introductory section continues with the promised treatment of elements of chance, and imperfect information.

### Chance

So how do we go about adding chance elements to our game? One of the accepted methods for doing so is to consider somebody called *Nature* who takes care of all the moves that involve an element of chance. (But Nature is not normally considered a *player* in the sense of Definition 1.) In the game tree, all we do is to add nodes

- where it is nobody's turn and
- where the branches from that node are labelled with the *probability* of the corresponding move occurring.

This does not just allow for the incorporation of chance devices, such as the throwing of coins and the rolling of dice, but also for situations with an otherwise uncertain outcome. In battle simulations, for example, it is often assumed that in certain situations (for example, defender *versus* aggressor), we have some idea of what is going to happen based on statistics (for example, in seven out of ten cases, defender will win). By force this is a somewhat crude way of modelling such things since it does not take into account the particular circumstances of a specific encounter (for example the personality and experience of those involved, the influence of geographical features, the quality of the defender's position (bunkers, suitable terrain, supplies, or the like)), but it still allows us to make a reasonable prediction regarding the outcome. A somewhat crude model is often better than none at all.[5]

**Example 1.3 Risk.** In the board game Risk players have 'armies' which can defend or conquer territory on a map (which forms the board). Assume a defender has one army left in some country. An attacker can choose (by placing his armies) how many he or she might want to attack with. We limit the choices here to attacking with one or two armies. Both players then roll as many dice as they have armies in the bout (here, one or two). In the case where two dice are rolled against one, only the bigger of the results of the throw of the two dice counts. If the defender's throw is at least as high as the attacker's then defender wins. In other words, for attacker to win his highest throw has to be higher than defender's. To keep the size of the game tree reasonable, we assume that instead of using ordinary dice we use ones which produce the numbers 1, 2 and 3 only, with equal probability.

**Exercise 2** (a) Draw a game tree where a player throws two dice one after the other. Assume that these dice show 1, 2, or 3 with equal probability. Use it to calculate the probability for each possible outcome and use them to explain Figure 3 (the subtree where $A$ rolls two dice). You may want to read on a bit if you are unsure how to deal with probabilities.

(b) Draw a tree for the game where two players get one card each out of a deck of three (consisting, say, of $J$, $Q$ and $K$). Count the number of different deals, and then the number where Player 1 has the higher card. If Player 2 wins in the case where she has the $Q$, or where she has the $K$ and Player 1 has the $J$, what is the probability that she wins the game?

The outcome of such a bout is shown in Figure 3. We say that the defender $D$ wins if he successfully defends his territory, and that the attacker $A$ wins if he invades the territory. The winner is marked for each final position of Figure 3.

So how much 'better' is it for the attacker to use two armies? For this we want to calculate the probability that $A$ will win in each case. How do we do that?

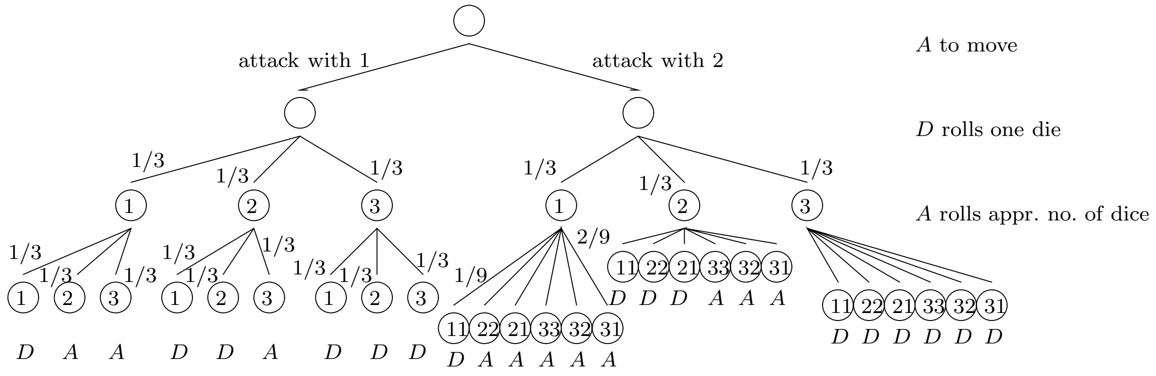
From Figure 3 we can see that if he attacks with one army, there are 3 final positions (out of 9) (corresponding to one play each) where $A$ wins. We have to *add up* those probabilities.

To calculate the probabilities for some final position, we have to *multiply* all probabilities mentioned along the path from the root that leads to it.

So the probability that $D$ throws a 1 while $A$ throws a 2 is $1/3 \times 1/3 = 1/9$. Similarly for the other two positions where $A$ wins (namely where $A$ throws a 3 while $D$ throws a 1 or a 2), so the probability that $A$ wins if he attacks with one army is

$$1/9 + 1/9 + 1/9 = 3/9 = 1/3 \approx 0.33.$$

[5] And Newton's theory of gravity is still good enough for many practical purposes, despite having been 'superseded' by the theory of relativity.




Probabilities for throwing two dice: 1/9 for each branch where the two numbers agree, 2/9 where they differ.

Figure 3: An excerpt from a game of Risk

Secondly we consider the case where $A$ attacks with two armies. Now we have eight cases (out of 18) where $A$ will win. The probabilities we have to add up are (from left to right) as follows.

$$1/27 + 2/27 + 1/27 + 2/27 + 2/27 + 1/27 + 2/27 + 2/27 = 13/27 \approx 0.48.$$

**Imperfect information**

Card games in particular often involve chance as well as imperfect information, because no player knows the cards the other players hold. This information has to be built into the game tree.

**Example 1.4 Paper-Stone-Scissors.** This is a two player game. At a command, both players hold out their right hand in one of three ways, indicating whether they have chosen paper, stone, or scissors. Paper beats stone which beats scissors which beats paper. You might have thought that the big problem of drawing a game tree for this game is the fact that both players move at the same time (and it is important that they do not know at the time which move the other player is about to make). But if we are allowed to mark imperfect information in our tree then we can deal with this. We let Player 1 move first, but demand that Player 2 be unable to tell which choice Player 1 has made. Figure 4 gives the corresponding game tree, where $P$ is for choice 'paper', $R$ is for choice 'stone' (think 'rock') and $S$ is for choice 'scissors'. The result is marked as 1 if Player 1 wins, 2 if Player 2 is successful and $D$ for a draw.

The grey-shaded area containing three nodes is called an *information set*—Player 2 only knows that the game has reached one of these nodes, but not which one.

**Question 4** Can you say anything about the nodes in the same information set? Is there a property they must all share?

Note that for nodes to be members of the same information set it *must be* the case that the moves from any of those nodes are precisely the same. In other words the moves which

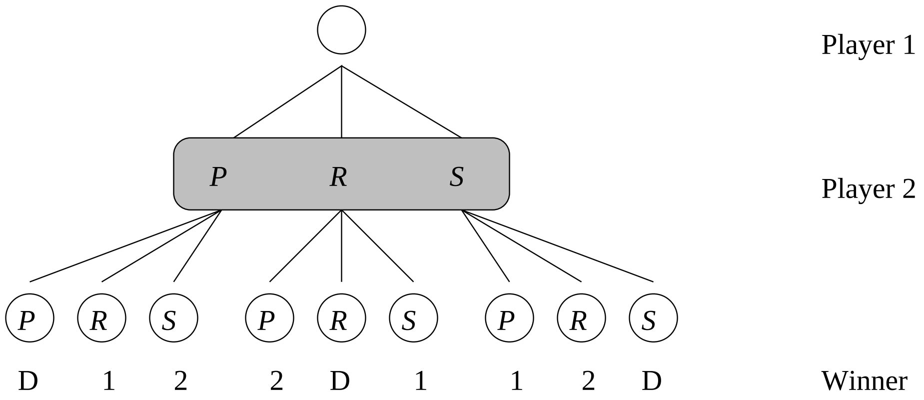



Figure 4: Paper-Stone-Scissors

are possible from any of those nodes are identical, and they are all moves for the same player. This is necessary so that the player whose turn it is at that point cannot find out whether or not a node really belongs to the information set by trying to play a move which is possible only for some of the nodes in the set.

Hence in addition to what is detailed in **Definition 1**, we allow the indication of groups of nodes, so called *information sets*, that one player cannot distinguish between. The nodes have to have the property that

- for all the nodes in an information set it is the same player's turn, and he is the one who cannot distinguish between them and
- the moves from one of the nodes in the information set are indistinguishable from the moves from any other such node.

**Exercise 3** (a) **Simplified Poker.** There are two players, each of whom has to pay one pound to enter a game (the *ante*). They then are dealt a hand of one card each from a deck containing three cards, labelled $J$, $Q$ and $K$. The players then have the choice between either betting one pound or passing. The game ends when

- either a player passes after the other has bet, in which case the better takes the money on the table (the *pot*),
- or there are two successive passes or bets, in which case the player with the higher card ($K$ beats $Q$ beats $J$) wins the pot.

The game tree for Simplified Poker is very large, and I don't expect you to draw the whole thing. Instead, draw a game tree for the deal. Then convince yourself that for the six possible deals, the game tree rooted there has the same shape. Draw this shape, and try to give pay-offs for all leaves. Finally, think about what the full game tree looks like—what are the information sets going to be?

(b) Draw a game tree for the game from Question 1 (c).

## 1.2 Strategies

When we play games, we usually have some sort of idea as to how we intend to go about it—people often talk about having a *strategy* for playing a game. This is, however, a fairly loose notion: Typically, it refers to a general plan without giving too much thought as to how exactly that plan should be carried out. For our purposes, a strategy is a much more specific notion. Leaving aside problems resulting from a large game tree, it is possible to do the following before the start of a game: For each position which might conceivably be reached where it is my turn, I choose a move that I will make if we get there.[6]

Figure 5 gives an example for such a strategy (for the first player) in the game of $2 \times 2$-Chomp. The strategy is given by the solid lines, the remainder of the original game tree is added in a 'paler' version for reference purposes.

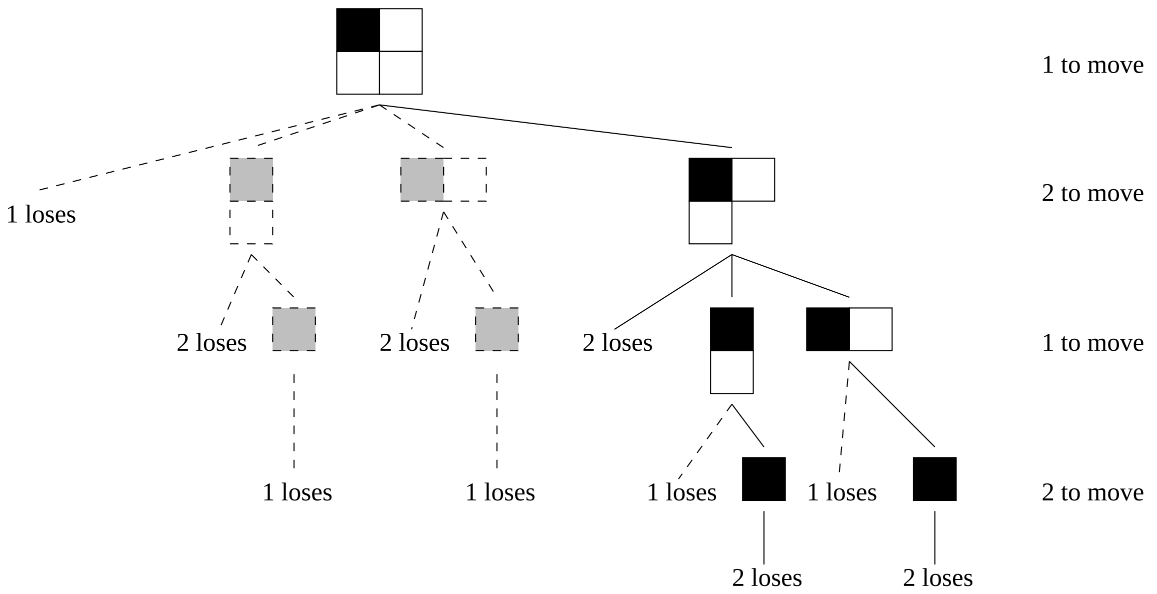



Figure 5: A strategy for $2 \times 2$-Chomp

Every play that follows the solid lines is a possible outcome when playing in accord with this strategy.

**Question 5** How many possible outcomes (final positions) does playing in accord with this strategy have? How many are advantageous to Player 1?

Note that because Player 1 has chosen to make the right-most move in the start position, he has ruled out that any of the positions following an alternative first move will ever be reached. As a consequence there is no need for the strategy to specify what should happen if a position in any of these subtrees will ever be reached, since this event cannot occur.

Closer inspection of the strategy shows that it is a part of the game tree with certain properties: Whenever a position is reachable based on the strategy's choices 'so far'

[6]Clearly not every position will be reached in the course of one play, unless the game is very boring indeed!

- if it is the chosen player's turn, precisely one of the available moves is chosen;
- if it is not the chosen player's turn, all available moves are chosen.

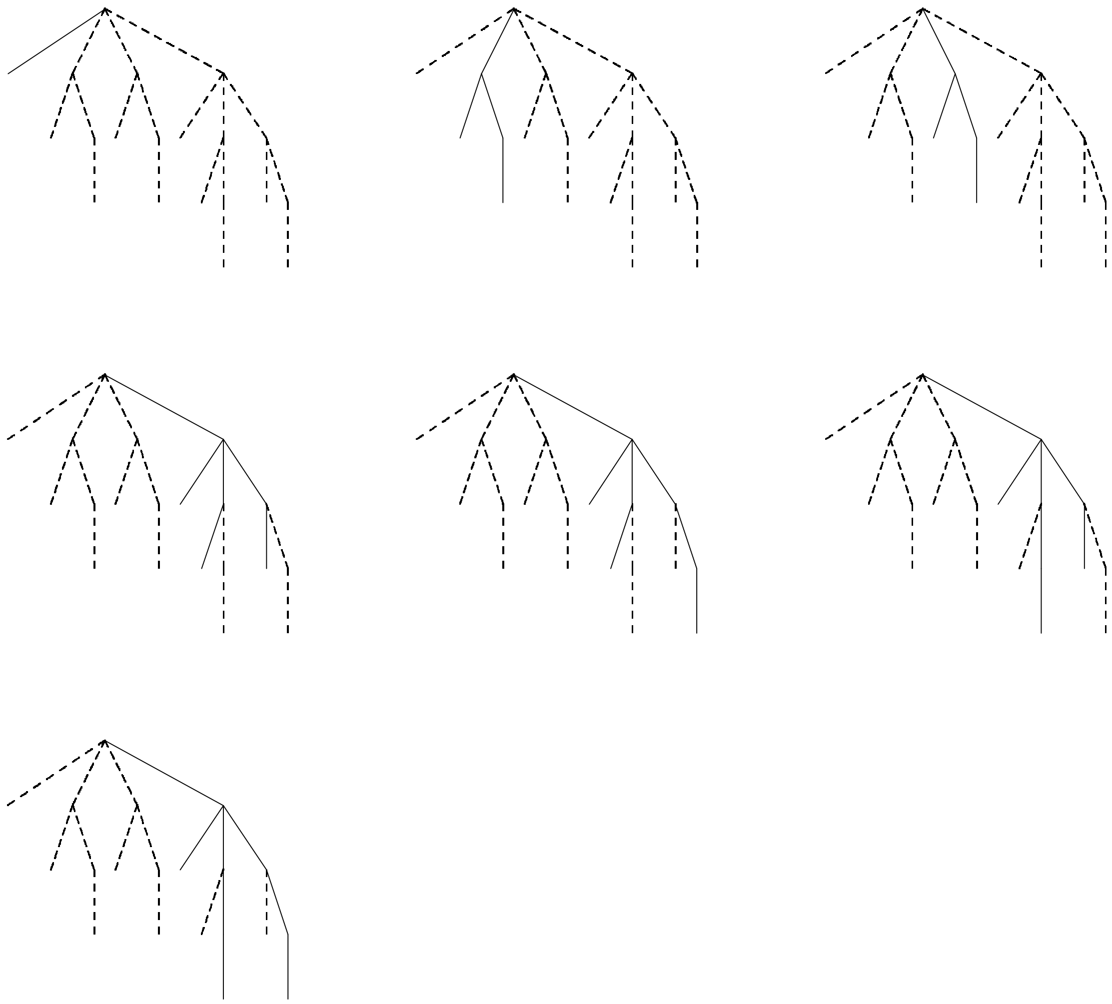


Figure 6: All the strategies in $2 \times 2$-Chomp for Player 1

If we follow these rules we can generate a strategy, making sure that we only make a decision when we have to (that is, we do not worry about unreachable positions). Mathematically, that means that the substructure of the game tree that we get when drawing a strategy is a tree again, namely a *subtree* of the game tree, with the same root. We can now define formally what a strategy is. Note that we demand that a choice be made for every position where it is the chosen player's turn; we do not allow bringing the game to a halt by refusing to continue.[7] If we want to give the player the option of resigning we should make that an explicit move in the game tree.

[7] In standard parlance our strategies are said to be *total*.

**Definition 2** *A* **strategy for Player** $X$ *is a subtree of the game tree with the following properties:*

- *The root of the game tree belongs to the strategy;*
- *whenever it is Player $X$'s turn at a node that belongs to the subtree,*
  - *exactly one of the available moves belongs to the subtree and*
  - *for all nodes in the same information set for Player $X$ where it is Player $X$'s term, the same move is chosen by the strategy;*
- *whenever it is not Player $X$'s turn at a node that belongs to the subtree, all of the available moves belong to the subtree.*

Note that as a consequence of using trees rather than graphs to give the rules of the game, we are allowing *more* strategies: We are allowed to take into account all the moves made so far, not merely the position reached on the board, say. This more generous notion can be justified by pointing out that the history that led to the current position might have given us an insight into the other players' ability in playing the game in question.[8] Figure 6 gives all the strategies for Player 1 in $2 \times 2$-Chomp. The game tree is now given in a stylized form only.

**Exercise 4** (a) How many strategies are there for Player 2 in $2 \times 2$-Chomp?

(b) How many strategies for Simplified Poker (see Exercise 3) are there for both players?

So what happens if we have a game which includes elements of chance? Actually, the definition we have just given will still work.

Let's have a closer look at a game of incomplete information, Example 1.4, Paper-Stone-Scissors. What the definition says is that there are only three strategies for Player 2—he (or she) is not allowed to try to take into account something he does not know, namely the first move made by Player 1. All valid strategies for Player 2 are given in Figure 7.

**Exercise 5** (a) Give all the strategies for $(2, 2)$-Nim (for both players). For this it is useful if your game tree takes symmetry into account to make it smaller!

(b) Give three different strategies for Simplified Poker (confer Exercise 3).

### Generating all strategies

Generating all the strategies for some player, say $X$, can be performed recursively as follows. When searching for the strategies for Player $X$ in a game tree $t$, we assume that we already know the strategies of the sub-games $t_1, \ldots, t_n$, which follow after the first move has been played (see Figure 8). At the same time we count the **number of strategies** $N_X(t)$ **for Player** $X$. However, this only works for games of *perfection information*!

- A game with a game tree of height zero has one strategy for each player ('do nothing').

[8] If we wanted to take away this source of information we could use *information sets* (see below) to deal with positions which 'look the same'.

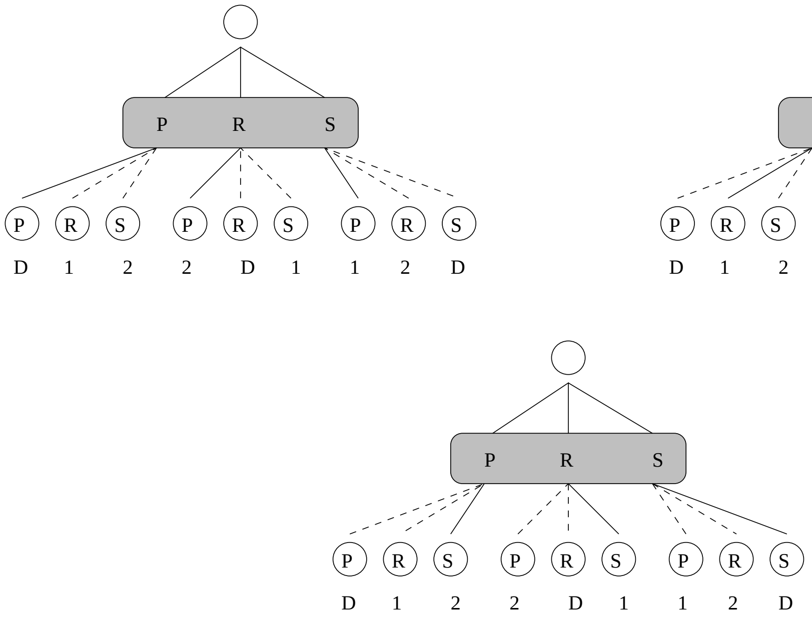



Figure 7: All the strategies for Player 2 in Paper-Stone-Scissors

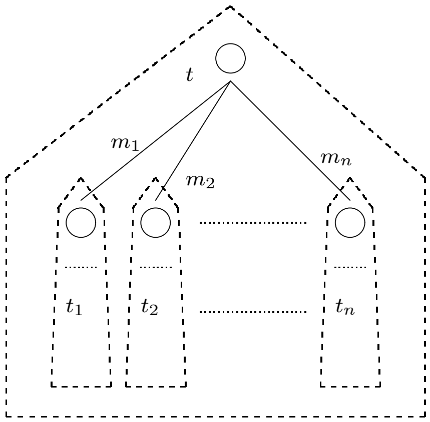



Figure 8: Counting strategies and immediate sub-games

- To find all the strategies for Player $X$ when the first move is Player $X$'s: Player $X$ has to choose one of the available first moves, $m_1$, $m_2$, $\ldots m_n$. Once that move, say $m_i$, has been played the game proceeds as per the game tree $t_i$. Hence every first move $m_i$, combined with some possible strategy for the corresponding sub-game $t_i$, gives a valid strategy for the game $t$. Therefore in this situation,

$$N_X(t) = N_X(t_1) + N_X(t_2) + \cdots + N_X(t_n).$$

- To find all the strategies for Player $X$ when the first move is not Player $X$'s, Player $X$ needs a reply for all the possible first moves in the game (which are somebody else's choice). So a strategy for the game $t$ for Player $X$ consists of picking a strategy for this player in each of the games $t_1$, $t_2$, …, $t_n$. All the combinations arising in this way are counted by

$$N_X(t) = N_X(t_1) \times N_X(t_2) \times \cdots \times N_X(t_n).$$

Think about how the above would have to be changed for games of *imperfect information*.

**Playing games *via* strategies**

Once a player has chosen a strategy, playing becomes a purely mechanical act: All he has to do from then on is to look up the chosen move whenever it is his turn and make it. Arguably, that makes playing a game a fairly boring activity, but we will see in a little while why this sometimes is a useful point of view. But leaving entertainment issues aside, why don't people typically do this? The answer is simple: For 'interesting' games, the game tree is typically too big to make it feasible to write down a strategy. And while it may be possible to describe the rules of the game (from which the game tree can be generated) in a compact form this is typically not the case for strategies. In Chess, for example, the rules will fit onto a page or two, but as we have seen the game tree has more than 400 positions after the first two moves. The strategies that could be described in a compact form mostly are of no practical value. So when playing a game like Chess the game tree unfolds move by move. The player typically only looks at the current position (thus removing irrelevant positions from consideration) and merely to a certain depth of the tree from that position (looking more than a few moves ahead is beyond most people's capacity). This means that moves are made based on fairly 'short term' considerations. Following the various choices to a final position is not feasible unless the game is almost over. Therefore players try to maximize short term gains, or use heuristics to aim for positions which they judge to be advantageous (an evaluation which is typically made based on experience). We will study these issues in more detail in the section on large games.

## 1.3 Games via strategies—matrix games

Once we have identified all the strategies for all the players of a game we can change the entire process of playing the game. Just let every player choose a strategy (independently from each other, that is, without knowing what everybody else has chosen), and then have them carry out their moves according to those. That means the entire course of play is determined once everybody has made a choice, which makes for a rather boring game. While no single player knows what the result will be, it is predetermined from the moment that all players have committed themselves. So why not leave out the process of playing altogether to jump to the inevitable conclusion?

Clearly this takes the fun out of playing games, and from what we have said above it should be obvious that large games cannot practically be treated in this way. Nonetheless it is a useful point of view to take for a game-theoretic analysis.

A simple example is Paper-Scissors-Stone. Each player has three strategies, which may be conveniently labelled by $P$, $R$ and $S$

If we list the strategies for Player 1 in a column, and those for Player 2 in a row we can fill in the result of playing a strategy for the one against a strategy for the other in the form of a table.

Determining the outcome when playing strategies against each other is done as follows. If no elements of chance are involved then simply follow the unique play (that is, path through the game tree) that the strategies under consideration (one for each player) have in common until a final position has been reached.

If elements of chance are involved then there may be several such plays. The probability for a given final position is calculated by multiplying the probabilities occurring along the path leading to it. However, there is no meaningful way of combining these results. That is why below we introduce the notion of a *pay-off function* mentioned above.

For Papers-Scissors-Stone we have recorded the result once again in term of who wins, $D$ stands for a draw.

| | | | 2 | |
|---|---|---|---|---|
| | | $P$ | $R$ | $S$ |
| | $P$ | $D$ | 1 | 2 |
| 1 | $R$ | 2 | $D$ | 1 |
| | $S$ | 1 | 2 | $D$ |

So once each player has chosen a strategy, we can read off the result from this table, or matrix, without bothering with going through the motion of playing the actual game. We have therefore given an *alternative description* of the original game—one which makes playing it rather boring. From the game-theoretic point of view, however it is totally irrelevant how exactly the result is reached, just what it is. Therefore this format strips off all information which is not germane to the analysis. We refer to a game presented in this form as a *matrix game*.

**Exercise 6** (a) Turn $(2,2)$-Nim into a matrix game.

(b) Turn the game from Question 1 (c) (and Exercise 3 (b)) into a matrix game.

If there are more than two players then we will not get a two-dimensional matrix. For each player we will have to add a dimension to keep track of all the possible outcomes.

**Question 6** How would you turn a three player version of Simplified Poker into a matrix game?

If the game in question contains elements of chance then it cannot be described in a matrix form unless the result can be recorded using a number. We will give an account of how to turn such a game into a matrix game in the next section.

The notion of a matrix game really only makes sense if the game is small enough for it to be described in this way. Nobody has a good idea even how many strategies White might have in a game of Chess or Go. While it is still true that considerations regarding matrix games apply to games where we have no such description, it really does not make a lot of sense to think in them that way. We will discuss matrix games in the section about small games.

## 1.4 The pay-off of playing a game

We finally turn to the last ingredient of our definition of 'game'. In particular when there are more than two players there is a valid question of what the entries for a matrix description of the game should be. Simply recording a winner might not sufficient. (What if there is more than one? Is it not worth recording who came second?) We will instead adopt the solution that each player attaches a number to a particular outcome which measures the player's preference (the higher the number, the 'better' the result). Doing this faithfully, however, is a far from trivial question.

It is related to another problem we will be concerned with, namely that of finding good ways of playing particular games—but what does 'good' mean? For some games, where there is a clear winner, that means winning. But there also are games where players score points, for example, in which case the aim might be to maximize the number of points, or maybe to maximize the difference between one's own points and those for everybody else. We therefore assume that our games come with a *pay-off function*: For each final position of the game (that is, a position from which no further moves are possible, which means that the game is over) a pay-off for each player is given. It is customary to assume that this pay-off can be any real number. (For many practical purposes that is far more generous than required.) The assumption then is that a high pay-off is desirable. For games where there is no natural pay-off, for example in Chess, we have to assign a pay-off function to turn the outcome into a number. Popular choices are to assign 1 to a win, 1/2 (for each player) to a draw, and 0 to a loss (for example during Chess championships). The Premiership, on the other hand, now functions with 3 points for a win, 1 for a draw, and none for a loss—and the sports enthusiasts among you will know that this has made a difference in the way teams play!

It is worth pointing out that for some examples it may be less than straight-forward how to assign a number to the outcome of a game. As indicated above, this choice may make a difference to any results an analysis may bring, and therefore such values should be chosen with care. We will say a bit more about this when we talk about game models.[9] The pay-off function provides us with a convenient entry for the matrix description of a game: Just fill in the pay-offs for the various players.

Here is a matrix version of Paper-Stone-Scissors under the assumption that a win is worth 1 (it doesn't really matter 1 of what), a loss is worth $-1$ and a draw 0. The matrix on the left is for Player 1, that on the right for Player 2.[10]

| | | | 2 | |
|---|---|---|---|---|
| | | $P$ | $R$ | $S$ |
| | $P$ | 0 | 1 | $-1$ |
| 1 | $R$ | $-1$ | 0 | 1 |
| | $S$ | 1 | $-1$ | 0 |

| | | | 2 | |
|---|---|---|---|---|
| | | $P$ | $R$ | $S$ |
| | $P$ | 0 | $-1$ | 1 |
| 1 | $R$ | 1 | 0 | $-1$ |
| | $S$ | $-1$ | 1 | 0 |

Something curious has happened here: if we add the entries in the matrix at each position we obtain a new $3 \times 3$ matrix all of whose entries are 0s. Games like that are special.

[9] In the jargon, we are trying to assign a utility function for each player to the final positions—this aims to find an accurate value (in the form of a real number) to assign to the outcome of a game which weighs *all* possible considerations of the player against each other (for example, the joy of beating $X$ *versus* any monetary payment made). This is almost impossible to achieve and it is not clear how to test any proposed utility function.

[10] Note that in a game with $n$ players we will need $n$ $n$-dimensional matrices to fully describe the game!

**Definition 3** *A game is a* **zero-sum game** *if for each final position the pay-offs for all players add up to 0.*

Such a game can be viewed as a *closed system*: Whether the numbers of the pay-off function stand for payments in money or points awarded, in a zero-sum game all the losses are somebody's gain, and *vice versa*.

Two person zero-sum games are particularly easy to describe: Given the pay-off matrix for one of the players that for the other is easily derivable: Just put a minus-sign in front of each of the entries (observing, of course, that $-0 = 0$ and that $- - r = r$). Hence whenever we describe a game using just one two-dimensional matrix, we are making the implicit assumption that this is a two player zero-sum game, where the payoffs are given for the player whose strategies are recorded in the first column of the matrix. (The other player's strategies are recorded in the first row of the matrix.) Sometimes the first row and column are left out entirely, in which case it is assumed that each player's strategies are simply numbered.

There are games which are not zero-sum: For example, if battles are modelled using games then losses might be the loss of troops. If that is the only part of the outcome that is recorded in the pay-off function then the game matrix will contain entries all of which are less than or equal to 0! Other examples are games played at casinos, where some of the money paid by the players goes to the casino.

If a game comes attached with a pay-off function for each player (as it should) then we can also put games with elements of chance into the matrix format. Such games differ from those we have considered so far in that even when each player has chosen a strategy the outcome is not uniquely determined (compare Risk, Example 1.3).

Calculating the pay-off when playing strategies against each other is done as follows. If no elements of chance are involved then simply follow the unique play (that is, path through the game tree) that the strategies under consideration (one for each player) have in common and read off the pay-off for each player from the resulting final position. If elements of chance are involved then there may be several such plays. The probability for a given final position is calculated by multiplying the probabilities occurring along the path leading to it. The pay-off for some player for a final position is then weighted with this probability and the expected pay-off is given by the sum of the weighted pay-offs for all the final positions that may occur.

**Example 1.5** Consider the following game between two players. Player 1 rolls a a three-faced die (compare Example 1.3). If he throws 1 he pays two units to Player 2. If he throws 2 or 3, Player 2 has a choice. She can either choose to pay one unit to Player 1 (she stops the game) or she can throw the die. If she repeats Player 1's throw, he has to pay her two units. Otherwise she pays him one unit. The game tree is given in Figure 9, with the pay-off being given for Player 1.

Player 1 has only one strategy (he never gets a choice) whereas Player 2 has four strategies (she can choose to throw the die or not, and is allowed to make that dependent on Player 1's throw). We can encode her strategies by saying what she will do when Player 1 throws a 2, and what she will do when Player 1 throws a 3, stop ($S$) or throw the die ($T$). So $S|T$ means that she will stop if he throws a 2, but throw if he throws a 3. The matrix will look something like this:

$$\begin{array}{c|cccc} & & & 2 & \\ & S|S & S|T & T|S & T|T \\ \hline 1 & & & & \end{array}$$

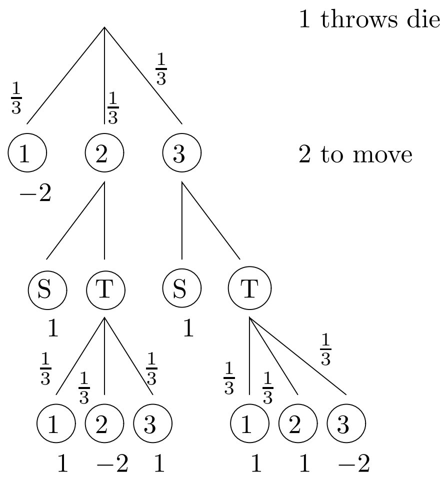


Figure 9: A game of dice

What are the expected pay-offs for the outcome of these strategies? We will first consider the case $S|S$. We calculate the expected pay-off as follows: For each of the possible outcomes of playing this strategy (combined with Player 1's only strategy), take the probability that it will occur and multiply it with the pay-off, then add all these up. Hence we get

$$(1/3 \times -2) + (1/3 \times 1) + (1/3 \times 1) = 0.$$

Now for the more interesting case of $S|T$:

$$(1/3 \times -2) + (1/3 \times 1) + (1/9 \times 1) + (1/9 \times 1) + (1/9 \times -2) = -3/9 = -1/3.$$

The case of $T|S$ is symmetric and therefore yields the same pay-off. Finally for the case $T|T$.

$$(1/3 \times -2) + (1/9 \times 1) + (1/9 \times 1) + (1/9 \times -2) + (1/9 \times 1) + (1/9 \times 1) + (1/9 \times -2) = -2/3.$$

The complete matrix looks like this:

| | 2 | | | |
|---|---|---|---|---|
| | $S\|S$ | $S\|T$ | $T\|S$ | $T\|T$ |
| 1 | 0 | $-1/3$ | $-1/3$ | $-2/3$ |

**Question 7** Which player would you rather be in the game from Example 1.5?

**Exercise 7** (a) Take the game tree where one player throws two dice in succession (see Exercise 2). Assume that the recorded outcome this time is the sum of the two thrown dice. For all numbers from 2 to 6, calculate how likely they are to occur. Then calculate the expected value of the sum.

(b) Take the game from Example 1.5, but change the pay-off if Player 2 decides to throw a die. If Player 1 and Player 2's throws add up to an odd number then Player 1 pays Player 2 one unit, otherwise she pays him one unit. Produce the matrix version of this game.

We say that a game is in **normal form** when it is given *via* a matrix.

### 1.5 Simple two person games

We are now ready to state our first result. If a player has a strategy which allows him to always win, no matter what the other player does, we call that strategy a **winning strategy**.

**Theorem 1.1** *Consider a game with two players, 1 and 2, of perfect information without chance, which can only have three different outcomes: Player 1 wins, Player 2 wins, or they draw. Then one of the following must be true.*

*(i) Player 1 has a winning strategy;*

*(ii) Player 2 has a winning strategy;*

*(iii) Player 1 and 2 both have strategies which ensure that they will not lose (which means that either side can enforce a draw).*

**Proof.** The proof proceeds by induction over the height of the game tree. The base case is given by a game of height 0, that is a game without moves. If this game is to fulfil the conditions regarding possible outcomes given in the theorem, it clearly has to fulfil one of the three stated cases. We can label the (only) node accordingly with a 1 if Player 1 wins, with a $-1$ if Player 2 wins and with a 0 if either side can enforce a draw.

Assume that the statement is true for all games of height at most $n$. Consider a game of height $n + 1$. This game can be considered as being constructed as follows: From the root, there are a number of moves (say $k$ many) leading to game trees of height at most $n$.

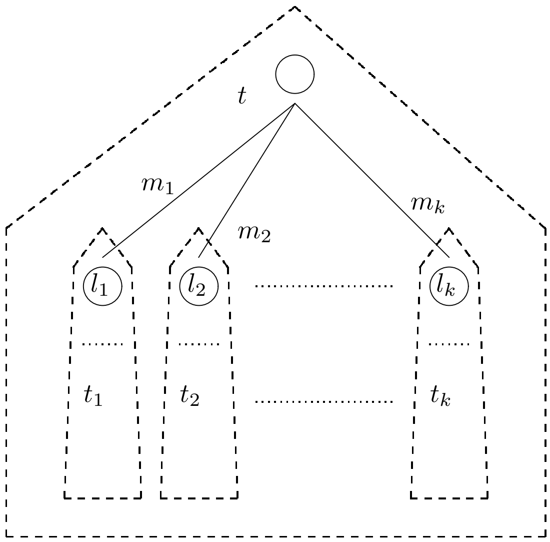



Figure 10: First moves and sub-games

By the induction hypothesis we can label the roots of these game trees with a number (say $l_i$ for tree $t_i$) as follows:

- it bears label $l_i = 1$ if Player 1 wins the game rooted there;
- it bears label $l_i = -1$ if Player 2 wins the game rooted there;

- it bears label $l_i = 0$ if the game rooted there is such that either side can enforce (at least) a draw.

Now if the first move of the game is made by Player 1, then there are the following cases to be considered:

- There is a child of the root labelled with 1, that is there is an $i$ in $\{1, 2, \dots, k\}$ such that $l_i = 1$. Then Player 1 can choose $m_i$ as his opening move, and combine it with the winning strategy for the game rooted at that child. This results in a winning strategy for the whole game and case (i) is met.

- None of the children of the root is labelled with 1, (that is $l_i \neq 1$ for all $1 \leq i \leq k$) but there is at least one $i$ with $l_i = 0$. Then by choosing $m_i$ as his first move, Player 1 can ensure that game $t_i$ is now played out where he can enforce a draw since $l_i = 0$. Hence Player 1 can enforce a draw in the overall game. To ensure that case (iii) is met we have to show that Player 2 also can enforce at least a draw. But all the games rooted at a child of the root of the overall game have label 0 or $-1$, so Player 2 can enforce at least a draw in all of them. Hence she can enforce at least a draw in the overall game.

- None of the children of the root is labelled with 1 or 0. That means that for all $1 \leq i \leq k$, $l_i = -1$ and Player 2 can enforce a win for all $t_i$. That means she has a winning strategy for the overall game, no matter which first move Player 1 chooses. Hence case (ii) is met.

The case where the first move of this game is made by Player 2 is symmetric to the one just discussed. □

A slightly more general statement (involving chance and allowing a larger variety of outcomes, which require the result be stated in a different language) was first made by Zermelo and later proved by John von Neumann. We will have more general results later on which subsume this one. Note that in order to find a winning strategy the entire game tree has to be searched if one is to follow the method given in the proof. Hence this method can only be applied to sufficiently small games.

Nonetheless, it means that games like Chess or Go are intrinsically boring in that one of those three statements has to be true for each of them. The games are so large, however, that we currently are nowhere near deciding which of the three cases applies, and so we still find it worthwhile to play them. Contrast this with the game of Noughts-and-Crosses, where the third case applies. Children typically discover this after a having played that game a few times and discard it as a pastime thereafter.

### Summary of Section 1

- Games can be represented using a *game tree*. Typically, a position in such a tree contains more information than just what would be shown on the board.

- Elements of *chance* are then modelled by having a player called Nature, with probabilities labelling such moves. *Imperfect information* is modelled by including in the game tree information about any nodes which cannot be distinguished by the player about to move.

- The *pay-off function* for a player assigns a value to each of the possible outcomes (final positions) possible in the game.

- A *strategy* for a player is a complete game plan for that player. It will choose a move for *every* situation in which the player might find himself.

- Small games have an alternative description *via* a *matrices* which show the pay-off for each player depending on the strategies chosen by all the players. Larger games have too many strategies for all of them to be listed. A game given in this way is known to be in *normal form.*

- In 2-player games of perfect information without chance either one of the players can force a win, or they can both force a draw.

**Sources for this section.** The material presented here is almost taken for granted in most texts on game theory and typically covered in a couple of pages. Examples have been taken from various publications which are listed in subsequent sections.

## 2 Small (non-cooperative) games

In this section we are concerned with solving what I call *small games*. I should like to stress that from a mathematical point of view, all the considerations to follow apply to *all* games satisfying the criteria we will be laying out (namely that of being *non-cooperative* and given in their *normal form*). But they are of not much use (beyond theoretical musings) unless the game can be brought into its matrix form. So when I speak of a 'small' game I mean one that is given by its matrix form (we will introduce a more suitable way of describing the matrix form of a game with more than two players). For the remainder of this section, whenever we speak of a game, we assume that that is how it is presented.

### 2.1 2-person zero-sum games: equilibria

Game theory is concerned with identifying ways of 'playing a game well', or even 'optimal play'. But what should that mean in practice? The pay-off function for each player gives us a way of deciding how well an individual did: The higher the pay-off, the better. Note that negative pay-offs mean that rather than receiving anything (whether it be points, money or something else), the player will have to pay something, so 'pay-off' can be a misleading term. We will in general not distinguish between the case where that number is positive and the one where it is negative.

**Remark.** It should be pointed out here that the following considerations will only apply if we assume that each player is aiming to get the best pay-off for himself. What we do *not* allow in this course is for the players to cooperate with each other, and then split the sum of their pay-offs. In other words we treat what are generally termed **non-cooperative** games.[11] This includes the sharing of information: no player is allowed to divulge information to another unless the rules explicitly allow it. While players are allowed to turn against one or more of the other players they may only do so if it is to their own advantage. Negotiations of the 'if you do this, I promise to do that' type are forbidden. A 2-person zero-sum game only makes sense as a non-cooperative game.

The generally adopted idea is as follows: Every player should play such that he (or she) will get the best pay-off he can ensure for the worst case.[12] One way of looking at this decision is to see it as an exercise in *damage limitation*: Even if the other players 'do their worst' as far as our player is concerned, they won't be able to push his pay-off below some threshold our player is aiming for (as a minimum).

**Question 8** Is this really the best thing a player can do? Might it not be worth taking a risk of receiving a lower pay-off if the worst comes to the worst if there's a chance of getting a much higher pay-off in return? Granted, the other players are supposed to be 'rational', but might they not have similar considerations?

If elements of chance are involved in the game then a 'guaranteed minimum pay-off' means an *expected pay-off*. That hides some pitfalls that you may not be immediately aware of.

[11] As might be expected there is also a notion of *cooperative game* in game theory. However, dealing with those is a lot more complicated than what we consider in this course.

[12] Recall that all our players are supposed to be *rational*.

**Question 9** Assume we are playing a game where we throw a coin, and one of us bets on heads while the other bets on tails. If you win, you have to pay me a million pounds, otherwise I pay you a million pounds. What is the expected pay-off in this game? Would you be willing to play a round?

This shows once again that pay-off functions have to be carefully chosen. If we merely use the monetary payment involved in the game in Question 9 and then just look at the *expected pay-off* it seems a harmless game to play—on average neither of us will lose (or win) anything. In practice, this doesn't cover all the considerations each of us would like to take into account before playing this game. One solution to this might be not to merely use the monetary payment as a pay-off, but rather make it clear that neither of us could afford to pay out that sort of money. If, for example, we set the pay-off for losing a million pound to $-100,000,000,000,000$, or the like, and kept 1,000,000 for the win of the million, then the 'expected result' would better reflect our real opinion.

What does the idea of a 'guaranteed minimal pay-off' mean in practice? We will study it using an example.

**Example 2.1 Camping holiday.** Let us assume there is a couple heading for a camping holiday in the American Rockies. They both love being out of doors, there is just one conflict that keeps cropping up. Amelia[13] appreciates being high up at night so as to enjoy cool air which means she will sleep a lot better. Scottie[13] on the other hand has a problem with the thin air and would prefer sleeping at a lower altitude, even if that means it's warm and muggy. In order to come to a decision they've decided upon the following: The area has four fire roads running from east to west and the same number from north to south. They have decided that they will camp near a crossing, with Scottie choosing a north-south road while Amelia decides on an east-west one, independently from each other. The height (in thousands of feet) of these crossings (with the roads numbered from east to west and north to south) is given by the following matrix.

| | | Scottie | | | |
|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 |
| | 1 | 7 | 2 | 5 | 1 |
| Amelia | 2 | 2 | 2 | 3 | 4 |
| | 3 | 5 | 3 | 4 | 4 |
| | 4 | 3 | 2 | 1 | 6 |

Let's consider the situation from Amelia's point of view. If she chooses Road 1 (with a potential nice and airy 7000 feet) then Scottie can push her down as far as 1000 feet by going for his Road 4. If she decides on her Road 2 then Scottie can decide between his Roads 1 and 2, and they'll still be sleeping in a hot and humid 2000 feet. In other words, she finds the *minimum* of each row to see what Scottie's worst response is for every one of her choices. We can summarize the result of her considerations in this table.

| Road No | min. height |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 1 |

[13] Yes, there are some very obscure references hidden in those names (one for each). A chocolate bar to anybody who can figure those out! Hint: Look back a few decades.

If she goes for Road 3 then they will sleep at a guaranteed 3000 feet, no matter what Scottie does. In all other cases he can push her further down. So she chooses the *maximum* of the entries in the new table by choosing her Road 3.

Let's now look at the situation from Scottie's point of view. If he chooses Road 1 then the worst that can happen to him is that Amelia goes for her Road 1, leaving him at a scary 7000 feet, the mere thought of which makes him feel somewhat nauseous. If he chooses Road 2, on the other hand, then at worst Amelia can push him up to 3000 feet. In order to calculate the worst that can happen to him, he looks for the *maximal* entry in each column. The result is summarized in the following table.

| Road No | max. height |
|---|---|
| 1 | 7 |
| 2 | 3 |
| 3 | 5 |
| 4 | 6 |

Now the best *he* can do is to choose the strategy which gives him the least of these numbers, guaranteeing that he will not have to sleep above 3000 feet. Hence he goes for his Road 2.

If both choose the strategy they have decided upon because of these considerations, they will indeed end up with the 'threshold' value identified, namely 3000 feet.

We can summarize this information in the original table of heights as follows.

| | | Scottie | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | min of row |
| | 1 | 7 | 2 | 5 | 1 | 1 |
| Amelia | 2 | 2 | 2 | 3 | 4 | 2 |
| | 3 | 5 | 3 | 4 | 4 | 3 |
| | 4 | 3 | 2 | 1 | 6 | 1 |
| | max of col. | 7 | 3 | 5 | 6 | 3\3 |

The situation is pictured in Figure 11. Here the dashed lines are the roads which lead from north to south and the solid lines are those running from east to west.[14]

The flag indicates where they will eventually camp.

**Question 10** What happens if Amelia changes her mind while Scottie sticks with his choice? What about if it is the other way round, that is, Amelia keeps her choice while Scottie changes his?

It is worth pointing out that from Amelia's point of view

- if she changes her mind, but Scottie doesn't, then the situation will worsen as far as she's concerned, that is they will camp at a lower site.

From Scottie's point of view, on the other hand, it is the case that

- if he changes his mind while Amelia doesn't then the situation will worsen as far as he's concerned, that is they will stay even higher up.

[14]Note that this 'landscape' shows the terrain in terms of the roads only.

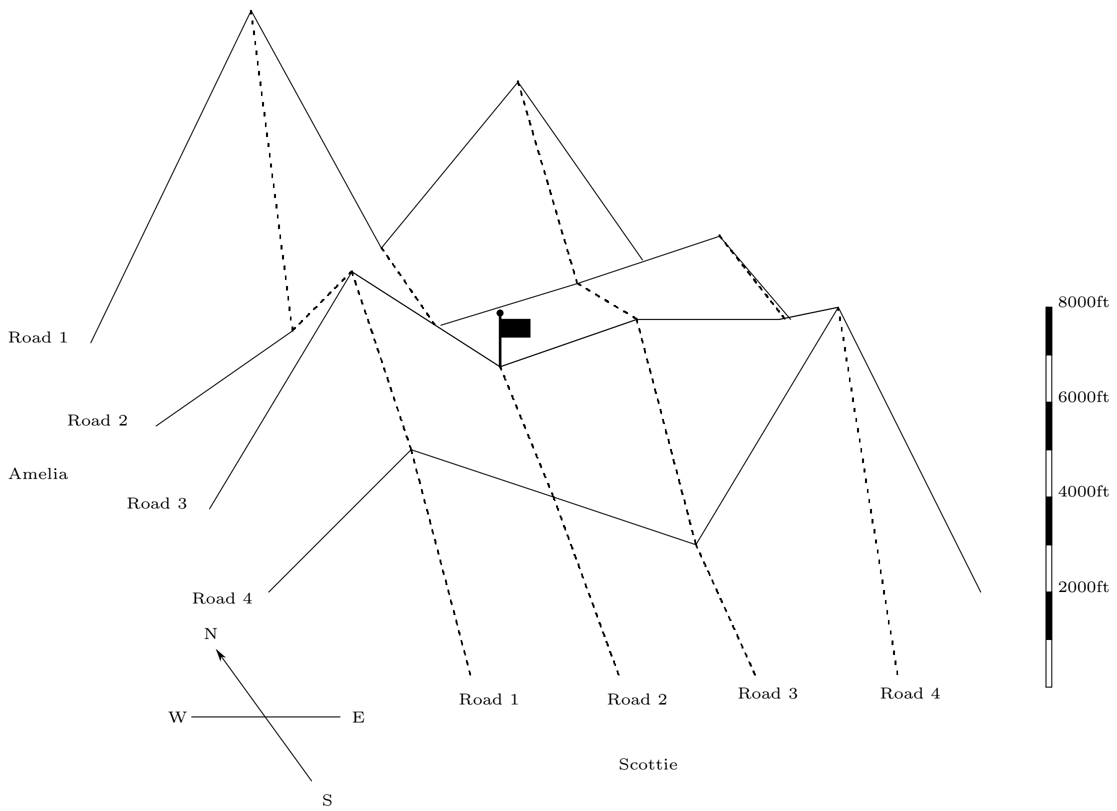



Figure 11: A landscape

In other words if we cut along Scottie's choice of roads (which corresponds to Amelia changing her mind while Scottie sticks to his choice) then the point they choose lies on the top of a hill (see Figure 12)—if she changes her mind, they will end up lower down. If, on the other hand, we cut along her choice (which corresponds to Scottie changing his mind) then their site lies in a valley (see Figure 13). Such points are known as **saddle points**.

However, more is true about these points: In fact, there is no point on Road 3 which is below the chosen one, nor is there a point above it on Road 2. This is a stronger condition since Road 3, for example, might dip into a valley, then go up, and then go very far down again—leaving a saddle point as before, but violating this new observation. We will formulate this idea mathematically in a moment.

Clearly it would not be very convenient always to have to draw a picture to find a saddle point, so how can we do it by just inspecting a matrix? We just mimic what Amelia and Scottie did to arrive at their choice of roads.

Let us assume that the matrix in question has elements $a_{i,j}$, where $i$ indicates the row and $j$ indicate the column.[15]

[15] If you don't feel happy thinking of matrices just think of arrays!

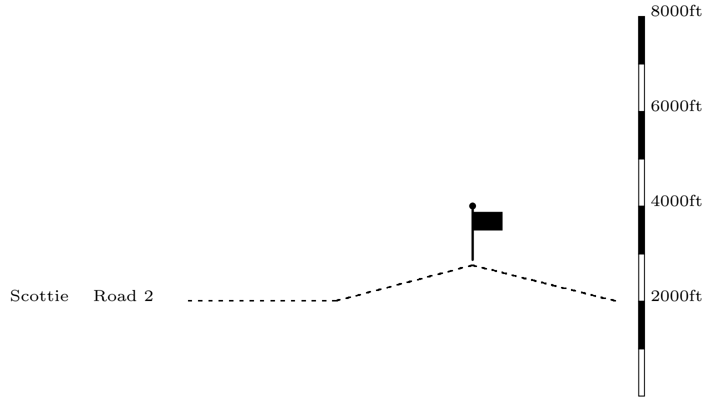



Figure 12: Road 2 (north-south), Scottie's choice

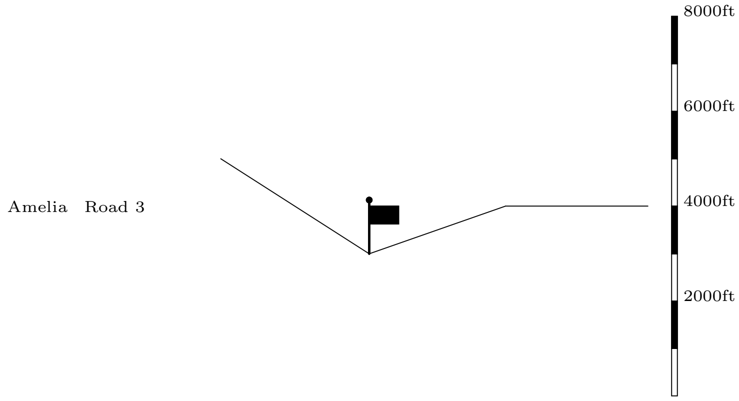



Figure 13: Road 3 (east-west), Amelia's choice

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

We say that this is a $(m \times n)$ **matrix**.[16]

Amelia's first step consisted of calculating, for a fixed row, that is for a fixed $i$, the *minimum* of all the $a_{i,j}$ where $j$ ranges over the number of columns (here $1 \leq j \leq 4$). That is, she determined, for each $1 \leq i \leq 4$,

$$\min_{1 \leq j \leq 4} a_{i,j}.$$

And then she calculated the largest one of those to make the corresponding road her choice, that is she computed

$$\max_{1 \leq i \leq 4} \min_{1 \leq j \leq 4} a_{i,j}.$$

[16]Other people give the number of columns before the number of rows, so this may be different from what you're used to.

Scottie, on the other hand, first computed, for a fixed column, that is for a fixed $j$, the maximum of the $a_{i,j}$, that is

$$\max_{1\leq i\leq 4} a_{i,j}.$$

Then he took the least of those and chose accordingly, that is he looked at

$$\min_{1\leq j\leq 4}\max_{1\leq i\leq 4} a_{i,j}.$$

**Exercise 8** For the zero-sum matrix games given below, calculate

$$\max_{1\leq i\leq m}\min_{1\leq j\leq n} a_{i,j} \qquad \text{and} \qquad \min_{1\leq j\leq n}\max_{1\leq i\leq m} a_{i,j}:$$

$$\text{(a)} \quad \begin{vmatrix} 4 & 3 & 1 & 1 \\ 3 & 2 & 2 & 2 \\ 4 & 4 & 2 & 2 \\ 3 & 3 & 1 & 2 \end{vmatrix} \qquad\qquad \text{(b)} \quad \begin{vmatrix} 2 & 3 & 4 & 1 \\ 4 & 2 & 3 & 2 \\ 1 & 2 & 3 & 2 \\ 3 & 1 & 2 & 3 \end{vmatrix}$$

**Exercise 9** This exercise isn't entirely straight-forward. The second part requires that you write out a little proof, and the main difficulty may well be to structure your ideas properly.

(a) Find an $(m \times n)$ matrix $(a_{i,j})$ for which the two values do not agree, that is such that

$$\max_{1\leq i\leq m}\min_{1\leq j\leq n} a_{i,j} \neq \min_{1\leq j\leq n}\max_{1\leq i\leq m} a_{i,j}.$$

(b) Show that for every $(m \times n)$ matrix $(a_{i,j})$

$$\max_{1\leq i\leq m}\min_{1\leq j\leq n} a_{i,j} \leq \min_{1\leq j\leq n}\max_{1\leq i\leq m} a_{i,j}.$$

Generalizing this idea we define the following.

**Definition 4** *Let $G$ be a zero-sum game of two players. Assume that Player 1 has strategies numbered 1, ..., $m$ and that Player 2 has strategies numbered 1, ..., $n$. Let the pay-off be given by the $m \times n$ matrix $(a_{i,j})$. We say that a pair of strategies, $i'$ for Player 1 and $j'$ for Player 2, give an* **equilibrium point** *of the game if it is the case that the entry $a_{i'j'}$ is maximal in its column and minimal in its row.*

In the camping holiday Example 2.1 the choice $(3, 2)$ is an equilibrium point. Why is it called that? The idea is that it describes a point of balance (hence the name). Let us look at why that should be so.

Amelia is the 'Player 1' in Example 2.1, and so she only controls which east-west road to take, that is, she gets to choose a row of the matrix. If she should change her mind to move away from the equilibrium point she faces the possibility that Scottie (Player 2) will stay with his choice, Road 2. But that means that the outcome will be worse for her—Figure 12 demonstrates that. If she changes her decision while Scottie sticks with his they will camp on some other intersection of Road 2 with another, and all those intersections are at a lower altitude.

Similarly if Scottie changes his mind while Amelia sticks with her Road 3 then they'll camp somewhere along the Road depicted in Figure 13, and that means that they'll camp at a higher altitude. Therefore both players have a reason to stick with their decision rather than change it, because any such change would be unilateral and thus likely lead to a worse outcome. We can think of an equilibrium as a point where both player's wishes are in balance with each other. Moving away from that upsets the balance, and the result will be a worse outcome *for the player who is responsible for the disturbance*. We can view this as thinking of a player as being punished for moving away from an equilibrium point.

**Exercise 10** Find the equilibria in the 2-person zero-sum games given by the following matrices:

$$\text{(a)} \quad \begin{vmatrix} 4 & 3 & 1 & 1 \\ 3 & 2 & 2 & 2 \\ 4 & 4 & 2 & 2 \\ 3 & 3 & 1 & 2 \end{vmatrix} \qquad\qquad \text{(b)} \quad \begin{vmatrix} 2 & -3 & 1 & -4 \\ 6 & -4 & 1 & -5 \\ 4 & 3 & 3 & 2 \\ 2 & -3 & 2 & -4 \end{vmatrix}$$

**Question 11** Can you find a 2-person zero-sum game which does not have any equilibrium points? We will treat the question of what to do with such games in Section 2.4.

Something curious has happened here. In our definition of an equilibrium point, we only demand that the corresponding matrix entry is the minimum of its row and the maximum of its column. Yet in the example, we had Amelia calculate

$$\max_{1\leq i\leq 4}\min_{1\leq j\leq 4} a_{i,j},$$

while Scottie calculated

$$\min_{1\leq j\leq 4}\max_{1\leq i\leq 4} a_{i,j}.$$

We know from Exercise 9 that the two do not have to agree. Was this just a coincidence in the example? The answer to that is given by the following result.

**Proposition 2.1** *Let $(i',j')$ be an equilibrium point for a 2-person zero-sum game with $m$ strategies for Player 1 (rows) and $n$ strategies for Player 2 (columns). Then it is the case that*

$$a_{i',j'} = \min_{1\leq j\leq n}\max_{1\leq i\leq m} a_{i,j} = \max_{1\leq i\leq m}\min_{1\leq j\leq n} a_{i,j}.$$

*If on the other hand*

$$\min_{1\leq j\leq n}\max_{1\leq i\leq m} a_{i,j} = \max_{1\leq i\leq m}\min_{1\leq j\leq n} a_{i,j},$$

*then the game has an equilibrium point.*

**Proof.** Let us first assume that the game has an equilibrium point. Since $a_{i',j'}$ is the maximum of its column, that is $a_{i',j'} = \max_{1\leq i\leq m} a_{i,j'}$, it is the case that

$$\begin{aligned} a_{i',j'} &= \max_{1\leq i\leq m} a_{i,j'} \\ &\geq \min_{1\leq j\leq n}\max_{1\leq i\leq m} a_{i,j}. \end{aligned}$$

Since it is also the case that $a_{i',j'}$ is the minimum of its row we can calculate

$$\begin{aligned} a_{i',j'} &= \min_{1\le j\le n} a_{i',j} \\ &\le \max_{1\le i\le m} \min_{1\le j\le n} a_{i,j}. \end{aligned}$$

Hence

$$\max_{1\le i\le m} \min_{1\le j\le n} a_{i,j} \ge a_{i',j'} \ge \min_{1\le j\le n} \max_{1\le i\le m} a_{i,j}.$$

Since we know from Exercise 9 that

$$\max_{1\le i\le m} \min_{1\le j\le n} a_{i,j} \le \min_{1\le j\le n} \max_{1\le i\le m} a_{i,j}$$

we are done with this direction.

Let us now assume that the equation

$$\min_{1\le j\le n} \max_{1\le i\le m} a_{i,j} = \max_{1\le i\le m} \min_{1\le j\le n} a_{i,j}$$

holds. We use $x$ for this number. We can now choose $i'$ in $\{1,\dots,m\}$ such that

$$\min_{1\le j\le n} a_{i',j} = x$$

and $j'$ in $\{1,\dots,n\}$ such that

$$\max_{1\le i\le m} a_{i,j'} = x.$$

We claim that $(i',j')$ is an equilibrium point.

We note that

$$x = \min_{1\le j\le n} a_{i',j} \le a_{i',j'} \le \max_{1\le i\le m} a_{i,j'} = x$$

and so all these numbers are equal. In particular $\min_{1\le j\le n} a_{i',j} = a_{i',j'}$ says that $a_{i',j'}$ is minimal in its row, and $\max_{1\le i\le m} a_{i,j'} = a_{i',j'}$ says that $a_{i',j'}$ is maximal in its column, so it is indeed an equilibrium point. □

Hence if a 2-person zero-sum game has more than one equilibrium point then the pay-offs for the players at each of those have to agree. That allows us to speak of the (unique) **value** of the game, namely the entry in the matrix defined by the equilibrium point(s).

If the game is not zero-sum then there is nothing sensible we can say about the expected pay-offs for both players, even if equilibrium points exist—the notion of a value of a game does not make any sense. An example appears on page 38. But even for zero-sum game, a value need not exist:

Consider the matrix

$$\begin{pmatrix} 5 & 1 \\ 3 & 4 \end{pmatrix}.$$

Then the considerations made by each player are as follows.

| | | | min of row |
|---|---|---|---|
| | 5 | 1 | 1 |
| | 3 | 4 | 3 |
| max of col. | 5 | 4 | 4\3 |

However, if we *can* find an equilibrium point then it doesn't matter *which* solution we find (in case there are several), because the outcome will be the same in each case! So if we are only interested in *one* solution to the game we can stop after we've found the first equilibrium point. While the others may add some variety (to our play, if nothing else), they do not change the outcome (the pay-off for either side) of the game in any way.

Another interesting fact regarding equilibria in 2-person zero-sum games is the following: In a game with an equilibrium point, even if we told the other player *in advance* that we were going to use the corresponding strategy, he could not use that additional information to his advantage: By sticking to that announced choice, we have ensured that we will receive (at least) the identified value of the game, and there's nothing our opponent can do about that!

We have already seen that some 2-person zero-sum games do not have any equilibrium points, for example Paper-Stone-Scissors, described in Example 1.4. But a large class of such game does indeed have equilibrium points. Before we state that result we first want to give a slightly different characterization for equilibrium points.

**Proposition 2.2** *A 2-person zero-sum game has an equilibrium point if and only if there exists a value $v \in \mathbb{R}$ such that*

- *$v$ is the highest value such that Player 1 can ensure a pay-off of at least $v$;*
- *$v$ is the smallest value such that Player 2 can ensure that she will not have to pay out more than $-v$.*

**Proof.** If the game has an equilibrium point then by Proposition 2.1 that is equal to $\max_{1 \le i \le m} \min_{1 \le j \le n} a_{i,j}$, which is the highest value such that Player 1 can ensure at least pay-off $v$. By the same Proposition this value is also equal to $\min_{1 \le j \le n} \max_{1 \le i \le m} a_{i,j}$, which results in the smallest pay-out, $-v$, that Player 2 can ensure.

If, on the other hand, there is a value $v$ satisfying the proposition then we can argue as follows. The highest value, $v$, that Player 1 can ensure as pay-off for himself is

$$\max_{1 \le i \le m} \min_{1 \le j \le n} a_{i,j}.$$

Player 2, on the other hand, can ensure that she does not have to pay out more than

$$\min_{1 \le j \le n} \max_{1 \le i \le m} a_{i,j},$$

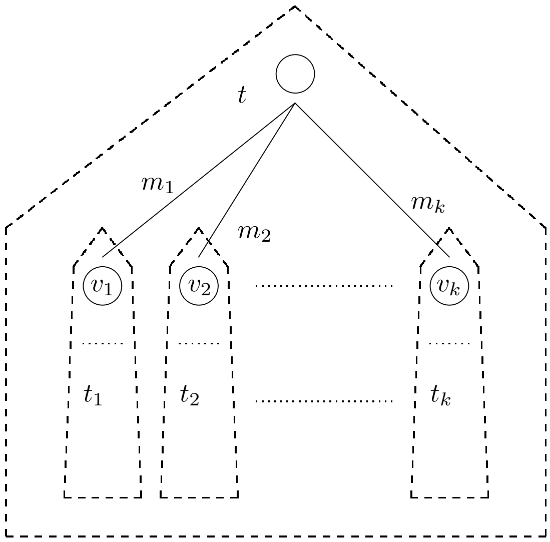
so this is the largest amount she may have to pay. But then

$$\min_{1 \le j \le n} \max_{1 \le i \le m} a_{i,j} = \max_{1 \le i \le m} \min_{1 \le j \le n} a_{i,j},$$

and by Proposition 2.1 this is sufficient to ensure the existence of an equilibrium point. □

**Proposition 2.3** *Every 2-person zero-sum game of perfect information has at least one equilibrium point.*

**Proof.** The proof of this result is similar to that of Theorem 1.1. A few changes have to be made, however. In that Theorem we assumed that the only outcomes were pay-offs 1, $-1$, or 0. Adapting this to arbitrary outcomes is easy: The induction hypothesis changes to the




assumption that each game of height at most $n$ has a value $v \in \mathbb{R}$, which we write down at its root.

The second adaptation is somewhat more complicated: In Theorem 1.1, we did not allow the game to include any elements of chance. So apart from nodes where it is Player 1's or Player 2's turn, we now have to include nodes where a random choice occurs.

Clearly every game of no moves has an equilibrium point and thus a value, and we assume that this is true for games whose game tree is of height at most $n$.

Now consider a game tree of height $n+1$. As argued in Theorem 1.1 we can view this game as starting at the root with each possible first move $m_i$ leading to a game $t_i$ of height at most $n$ which is subsequently played out. We assume that at the root of each of these sub-games its value is given. There are three cases to consider.

Let us first assume that the first move is made by Player 1. Then Player 1 can ensure that his pay-off is the maximum

$$v = \max_{1 \leq i \leq k} v_i$$

of the values $v_i$ of the sub-games reached after the first move. Player 2, on the other hand, can ensure that *her* pay-out is no worse than $-v$: No matter which first move Player 1 chooses, the worst case for Player 2 is that where she has to pay out $-v$. Hence $v$ is indeed the value of the overall game.

Let us now assume that the first move is made by Player 2. This case is almost the same as the one we have just discussed, the only difference being that values are given referring to Player 1's pay-off. Hence Player 2 will be looking for the *least*

$$v = \min_{1 \leq i \leq k} v_i$$

of the values labelling the sub-games. The argument that this $v$ is the value of the game is similar to the one before, only that the roles of Player 1 and Player 2 are reversed.

Finally we have to consider the case where the first move is a chance move. Then the highest pay-off Player 1 can hope for is the *expected pay-off*

$$\sum_{1 \le i \le k} q_i v_i,$$

which is calculated by taking the probability $q_i$ that a particular move $m_i$ occurs times the value $v_i$ of the subsequent game, and summing those up over all possible first moves. But that is precisely the least pay-out that Player 2 can expect. □

The technique employed in this proof will can also be used to say something about larger games.

The last observation which we wish to make about 2-player games of perfect information is that if we have found a Nash equilibrium we can consider the two strategies involved as being *best replies* to each other: For either player, changing the reply to the other side's choice results in a worse result. The situation is, of course, completely symmetric.

## 2.2 General non-cooperative games: equilibria

If we want to generalize the notion of equilibrium point to non zero-sum games of several players we have to change notation slightly. As we have seen in Section 1, we need quite a few ('multi-dimensional') matrices to describe such a game fully. If we refer to the pay-off via the elements of a matrix then a fairly large number of indices is required to fix the element we are talking about (one to indicate the player for which this is a pay-off and then one for each player to indicate which strategy we are talking about).

What we will do instead is to describe the game slightly differently.

**Definition 5** *A* **game in normal form** *is given by the following ingredients:*

- *A (finite) list of players, 1, ..., $l$;*
- *for each player a list of valid strategies for the player, numbered 1, ..., $n_j$ for Player $j$;*
- *for each player a pay-off function which maps the space of all strategies*

$$\prod_{1 \le i \le l} \{1, \dots, n_i\}$$

*to the real numbers, for Player $j$ that function is typically called $p_j$. So we know that*

$$p_j \colon \prod_{1 \le i \le l} \{1, \dots, n_i\} \longrightarrow \mathbb{R}.$$

A game in normal form is nothing but the generalization of the notion of a 'matrix game' to more than 2 players. In Section 1 we have seen that the name 'matrix game' is somewhat misleading if there are more than two players, and all we have done here is to choose a sensible representation. If, for all $1 \le j \le l$, Player $j$ chooses a strategy $i_j$ (in the space of his available strategies, that is the set $\{1, \dots, n_j\}$) then we can calculate the pay-off for each player by applying the appropriate function to the tuple $(i_1, \dots, i_l)$. That is, the pay-off for Player 1 in that situation is given by $p_1(i_1, \dots, i_l)$, that for Player 2 by $p_2(i_1, \dots, i_l)$, and so on.

**Question 12** For the 2-person zero-sum game in Example 2.1 (involving Scottie and Amelia), what is the space of all strategies, and how do you calculate the pay-off function for each player? Can you generalize this to any 2-person zero-sum game?

This formal definition looks scarier than it is. Here is a concrete example.

**Example 2.2** Consider the following three person game. Each player pays an ante of one. On a signal, all the players hold up one or two fingers. If the number of fingers held up is divisible by 3, Player 3 gets the pot. If the remainder when dividing is 1, Player 1 gets it, otherwise Player 2 is the lucky one.

**Question 13** Do you think that this game is likely to be 'fair', in the sense of giving all the players an even chance to win? Which player would you prefer to be?

Each player has two strategies: Holding up one finger or holding up two fingers. We number them as 1 and 2 (in that order). Hence the space of all strategies is

$$\begin{aligned}\prod_{1\leq j\leq 3}\{1,2\} &= \{1,2\}\times\{1,2\}\times\{1,2\}\\ &= \{(1,1,1),\ (1,1,2),\ (1,2,1),\ (1,2,2),\ (2,1,1),\ (2,1,2),\ (2,2,1),\ (2,2,2)\}.\end{aligned}$$

It has $2\times 2\times 2 = 8$ elements. The three pay-off functions are given as follows (this takes into account that the players have all paid the ante, that is they either lose one unit or they get 3, one of which they put in themselves, making for a win of 2):

| | $p_1$ | $p_2$ | $p_3$ |
|---|---|---|---|
| $(1,1,1)$ | $-1$ | $-1$ | $2$ |
| $(1,1,2)$ | $2$ | $-1$ | $-1$ |
| $(1,2,1)$ | $2$ | $-1$ | $-1$ |
| $(1,2,2)$ | $-1$ | $2$ | $-1$ |
| $(2,1,1)$ | $2$ | $-1$ | $-1$ |
| $(2,1,2)$ | $-1$ | $2$ | $-1$ |
| $(2,2,1)$ | $-1$ | $2$ | $-1$ |
| $(2,2,2)$ | $-1$ | $-1$ | $2$ |

Assume that Player 1 chooses to show 1 finger, that is his strategy 1, Player 2 and Player 3 show 2 fingers, that is their strategy 2. Then the pay-off for Player 2 is $p_2(1,2,2) = 2$. You will notice that there are three cases in which Player 1 wins, and also three cases in which Player 2 wins, but only two in which Player 3 is fortunate. Hence Player 3 seems to be at a disadvantage.

**Definition 6** *Let $G$ be a (non-cooperative) game in normal form with $l$ players. Then a choice of strategies for each player,*

$$(i'_1,\dots,i'_l)\in\prod_{1\leq j\leq l}\{1,\dots,n_j\}$$

*gives an* **equilibrium point** *for the game if it is the case that for all $1\leq j\leq l$ and for all $1\leq i\leq n_j$*

$$p_j(i'_1,\dots,i'_{j-1},i'_j,i'_{j+1},\dots,i'_l)\geq p_j(i'_1,\dots,i'_{j-1},i,i'_{j+1},\dots,i'_l).$$

*In other words: If Player $j$ changes away from his choice, strategy $i'_j$, then his pay-off can only decrease (or at best stay the same).*

Clearly the principle is just the same as that for zero-sum 2-person games: An equilibrium is a point where all the players have an incentive to stay with their choice of strategy, because they risk decreasing their pay-off if they unilaterally change their mind.

You can check that the the game on the prevoius page has two equilibrium points, $(1, 2, 1)$ and $(1, 2, 2)$. The first one is advantageous for Player 1 while the second works well for Player 2 (at their preferred choice they get 2 rather than having to pay 1), but it is Player 3 who gets to choose between those! And Player 3 has no incentive to go for one or the other, since his pay-off will be $-1$ in either case.

These equilibria are often referred to as *Nash equilibria* in the literature, after John Nash. He is a mathematician who won the Nobel prize (for economy) for his work in game theory in 1994, 45 years after his ground-breaking paper on the subject first appeared. Nash suffered from schizophrenia for decades, but recovered in the 1990s. If you want to find out more about him, `http://www-groups.dcs.st-and.ac.uk/~history/Mathematicians/Nash.html` gives a brief history. You are also encouraged to read the acclaimed biography *A Beautiful Mind*, by Sylvia Nasar (Simon & Schuster, 1998—there's also a paperback edition by Faber and Faber, 1999). Or, if you can't stomach a long book you can always watch the award-winning film of the same title.

Not all games have equilibria (at least not in the sense just defined), a counter-example is given by Paper-Stone-Scissors, Example 1.4. We will discuss how to define solutions for such games in Section 2.4.

**Exercise 11** Find the equilibria for the following matrix games. The first number in an entry gives the pay-off for the row player, the second number that for the column player.

$$\text{(a)} \quad \begin{vmatrix} (-10, 5) & (2, -2) \\ (1, -1) & (-1, 1) \end{vmatrix} \qquad\qquad \text{(b)} \quad \begin{vmatrix} (1, 2) & (0, 0) \\ (0, 0) & (2, 1) \end{vmatrix}$$

If we consider a non-cooperative game for two players which is *not* zero-sum then we lose the property that all equilibria are 'the same': In other words, equilibria do not have to yield the same pay-off, or value any longer.

Here is an example: In the game given by the matrix

$$\begin{vmatrix} (3, 2) & (0, 0) \\ (0, 0) & (2, 3) \end{vmatrix}$$

both choices, $(1, 1)$ and $(2, 2)$ form equilibrium points: From the row player's perspective, changing to strategy 2 while the column player sticks with strategy 1 reduces the pay-off, as does changing from strategy 2 to strategy 1 if the other player sticks with his strategy 2. But the two have *different* pay-offs: 3 for Player 1 in the case of $(1, 1)$ *versus* 2 in the case of $(2, 2)$. The situation for Player 2 is precisely the opposite. Hence Player 1 has reason to prefer the equilibrium point $(1, 1)$ while Player 2 would rather stick with $(2, 2)$. Hence Proposition 2.3 is no longer valid for non-zero sum games.

There are other problems with this idea which we will discuss in Section 2.3.

**Exercise 12** (a) Consider the following game for three players. Each player places a bet on the outcome (1 or 2) of a throw of a die without knowing what the others are betting. Then the die is thrown. If the number showing is odd we record the result as 1, otherwise as 2. A player gets a pay-off of ten points if he is the only one to bet on the correct result, if two of them do so they each get four points, and if all three are successful they get two points each. Describe the normal form of this game. Does it have equilibria?

(b) Consider the following game for three players. Player 1 announces whether he chooses left ($L$) or right ($R$), then Player 2 does the same, and lastly Player 3. The pay-off for each player is calculated as follows: If all players make the same choice, they each get 1 point if that choice is $L$, and they each lose 1 point if that choice is $R$. If two choose $R$ while one chooses $L$ then the two players choosing $R$ obtain 2 points each while the sole supporter of $L$ loses 2 points, and if two choose $L$ while only one chooses $R$ then the person choosing $R$ gets 3 points while the other two get nothing, but don't have to pay anything either.

How many strategies are there for each player in the game? Can you find a path in the game tree that leads to an equilibrium point pay-off? (It is possible to do so without writing out the normal form, although it might be helpful to draw a game tree first.) How many strategies lead to this pay-off, and how many equilibrium points exist?

## 2.3 Are equilibria really the answer?

With the exception of Question 8, we have accepted the idea of an equilibrium point as the solution to a game without much critical thought. After all, each player ensures a certain expected pay-off for himself below which the others cannot push him, and if a player unilaterally moves away from an equilibrium point he will be punished for it by risking a lower pay-off. But are equilibrium points really always the answer?

Consider the pay-off matrix given by

$$\begin{vmatrix} (-20,-20) & (15,-15) \\ (-15,15) & (10,10) \end{vmatrix}$$

It has two equilibrium points at $(1,2)$ and $(2,1)$, each of them being preferred by one of the players. What should they settle on? Clearly, once they have settled on one, each player risks the utterly undesirable outcome of $(-20,-20)$ when unilaterally moving away from the equilibrium point. The option $(2,2)$ is certainly a compromise of some sort, but how can the players get there if they are not allowed to communicate with each other? And wouldn't it be tempting for either of them to switch strategies to increase the pay-off from 10 to 15?

**Example 2.3 The Prisoner's Dilemma.** Two dark figures, Fred and Joe, have been caught by the police and are now being questioned—separately from each other. The police have the problem that they do not have firm proof against either—if they both keep mum then the best the police can expect is that they each get two years for petty crime, whereas it is suspected that they were involved in armed robbery. The police are therefore interested in bluffing them into making a confession, offering that if one of them turns crown witness he will get off scot-free while the other will face 10 years in prison. (It is not stressed that if they both confess they will each face 8 years, two years having been deducted due to the confession.)

Maybe somewhat surprisingly, the players in this game are the two prisoners. Each of them faces a choice: to confess or to stay quiet? It seems tempting at first not to say anything—after all, if Joe only does the same then Fred will get away with just two years. On the other hand, two years in gaol is a long time. If he talks he might walk away a free man. And, of course, can he really trust Joe to keep quiet? If Joe shops him he's looking at ten years while Joe is out. Surely it's better to at least have the confirmation that Joe is suffering as well for his treachery.

Here is the pay-off matrix (in years spent in prison, with a negative number to make it clear that 10 years in prison are worse than 2, and so to fit our interpretation of the pay-off functions) for the situation. The number on the left of each pair shows the pay-off for Fred, the number on the right that for Joe.

| | | Joe | |
|---|---|---|---|
| | | talk | don't talk |
| Fred | talk | $(-8, -8)$ | $(0, -10)$ |
| | don't talk | $(-10, 0)$ | $(-2, -2)$ |

This game has an equilibrium point at (talk, talk), since for both, Joe and Fred, the situation will get worse if they shift away from that strategy.

Hence from the game theory point of view, the 'solution' to this game is for each of them to talk and spend 8 years in prison. Clearly, this is not a particularly good outcome. If one takes their collective situation into account, it is very clear that what they should both do is to remain silent (much to the regret of the police!).

**Question 14** What would you do in a situation like Joe and Fred? You don't have to picture yourself as a prisoner to come into a similar dilemma. For example, assume somebody is offering goods for sale on the Internet and you're interested in buying. Neither of you wants to pay/send the goods first, so you decide you'll both send your contribution to the other party at the same time. Isn't it tempting to let the other guy send you the goods without paying? Would you change your mind if you wanted to do business with this person again (which amounts to playing the game again)?

There are, in fact, a large number of 'Prisoner's Dilemma type' situations, and we will meet more of those when we study game models, but these are not covered in the notes.

Here is another example. Douglas R. Hofstadter[17] once sent a postcard with the following text to twenty of his friends:[18]

'...Each of you is to give me a single letter: 'C' or 'D' standing for 'cooperate' or 'defect'. This will be used as your move in a Prisoner's Dilemma with *each* of the nineteen other players. The pay-off matrix I am using for the Prisoner's Dilemma is given in the diagram.

| | | Player B | |
|---|---|---|---|
| | | C | D |
| Player A | C | $(3, 3)$ | $(0, 5)$ |
| | D | $(5, 0)$ | $(1, 1)$ |

[17]He is the author of *Gödel, Escher, Bach*, a book that aims to expose the reader to a number of ultimately self-referential and apparently paradoxical systems taken from different areas: mathematics and logic (Gödel's Incompleteness Proof is one of the results explained), the arts (in the form of prints by M.C. Escher who delighted in inventing tilings and drawing ever-rising finite staircases) and music (that of J.S. Bach in particular).

[18]He summarized this in his column in the *Scientific American, June 1983.* These columns, with a few extra articles, can be found in his book *Metamagical Themas.*

Thus if everyone sends in 'C', everyone will get \$57, while if everyone sends in 'D', everyone will get \$19. You can't lose! And, of course, anyone who sends in a 'D' will get at least as much as everyone else will. If, for example, 11 people send in 'C' and 9 send in 'D', then the 11 C-ers will get \$3 apiece for each of the other C-ers, (making \$30), and zero for the D-ers. So C-ers will get \$30 each. The D-ers, by contrast, will pick up \$5 apiece for each of the C-ers, making \$55, and \$1 each for the other D-ers, making \$8, for a grand total of \$63.

...You are not aiming at maximizing the total number of dollars *Scientific American* shells out, only maximizing the number that come to *you*!

...I want all answers by telephone (call collect, please) the day you receive this letter.

It is to be understood (it *almost* goes without saying, but not quite) that you are not to try to get in touch with and consult with others who you guess have been asked to participate. In fact, please consult with no one at all. The purpose is to see what people will do on their own, in isolation. ...'

**Question 15** What would you do if you received such a letter? And why? How many people do you think chose 'C', how many 'D'?

Hofstadter had hoped for twenty 'C's.

**Question 16** Can you think of a way in which a clever person could have convinced himself that everybody should go for 'C'?

In fact, he got 14 'D's and 6 'C's, so the 'defectors' each received \$43 while the 'cooperators' had to make do with \$15 each.

**Question 17** What would game theory have to tell these people? What would have happened if they had all applied this theory? On the other hand, how do you think this sort of game would develop if it were played repeatedly, say once a week over a year?

**Exercise 13** Discuss the relative merits of the 'solutions' given by the equilibria for the following non-cooperative games. What if the pay-off is in pound sterling, and you are the player having to make a decision?

$$\text{(a)} \quad \begin{vmatrix} (4,-300) & (10,6) \\ (8,8) & (5,4) \end{vmatrix} \qquad\qquad \text{(b)} \quad \begin{vmatrix} (4,-300) & (10,6) \\ (12,8) & (5,4) \end{vmatrix}$$

The morale of this section so far is that equilibria only *guarantee* a solution (assuming both players are rational) in the case of a zero-sum 2-person game. There are reasons to consider them at least with some suspicion in other cases. However, there is no better solution for those cases. We will revisit some of these issues in the section on repeated games.

If you think that the above examples are artificial in nature, maybe the following two will be more to your taste. If nothing else they are taken from the area of computing.

**Example 2.4 TCP/IP Congestion Control Protocol.** All TCP implementations are required to support the algorithm known as *slow-start*. It works as follows:

- Start with a batch size of one packet;
- keep doubling the number of packets in a batch until a predefined threshold size is reached;

- from then on increase the batch size by one with each transmission;
- when congestion occurs (which is detected by a time-out occurring when waiting for an acknowledgement), reset the threshold value to half the current batch size and start over (with a batch size of one packet).

The idea of the algorithm is to adapt the batch size to what is currently supportable within the network, and it is fair since everybody follows the same rules, thus trying to split resources evenly between all users.

For an individual, however, it is tempting to employ a 'greedier' algorithm than the above slow start: For example jump straight back to the new threshold value (rather than starting with a batch size of 1). (It doesn't make sense to go for a bigger size since in all likelihood, that will just result in more time-outs.) But if everybody does that, the network will become very congested. This is a multi-player prisoner's dilemma-type situation: The community is best off if everybody exercises restraint. But a small number of people could get away with getting a better return than the average (and, indeed, the slow-start algorithm strategy does not lead to an equilibrium even when everybody employs it). Of course, this only works if only a very few people try to cheat.

**Example 2.5 ALOHA Network Algorithm.** An early, elegant (and very successful) algorithm for allocating a multiple-access channel (initially designed for ground-based radio broadcasting) works as follows:

- Send a data packet as soon as it becomes available for sending;
- if a collision occurs (because somebody else accessed the channel at the same time), wait a random amount of time and try again.

Using a random delay means that packets are unlikely to collide repeatedly. Again, the algorithm is fair because everybody uses the same algorithm, and thus on average has the same amount of waiting time.

Once more it is very tempting for the individual to try to get away with retransmitting a packet immediately—and if there's just one such individual, he's likely to get away with it. Again this is a multi-player prisoner's dilemma-type situation.

**Question 18** What happens if two players try to get away with immediate retransmissions?

While both the situations given in Examples 2.5 and 2.4 are games which are carried out repeatedly, it is impossible for the other players to tell whether any of the others is cheating. Hence any incentive to behave oneself that might exist due to social pressure is lost.

## 2.4 Mixed strategies and the Minimax Theorem

In Section 2.2 we have seen that with the exception of 2-person zero-sum games of perfect information, non-cooperative games do not have to have equilibrium points. It is natural to ask whether game theory is therefore unable to tell us anything about how to play a game such as, for example, Paper-Stone-Scissors (see Example 1.4). The answer is that it can help us, but we have to change our assumptions somewhat.

Games without equilibrium points have no obvious 'best' strategy in the sense of guaranteeing a certain threshold pay-off no matter what the other players do. Reconsider the example Paper-Stone-Scissors—clearly all three strategies for each of the players are equivalent to the others, and therefore there cannot be a 'best' one. So how should we play this game?

Game theory's answer to this problem is to assume that we are going to play the game under consideration *many times*. This appears somewhat paradoxical: if we can't even say how to play the game well once, why should we be any better at trying to play it a number of times?

The answer is that that allows us to come up with something other than *a single strategy* in our answer to the question of how best to play the game. Instead we can do something that sounds very complicated at first sight:

*We can assign a probability to each available strategy.*

**Definition 7** *A* **mixed strategy** *for a player consists of assigning a probability to each of the player's strategies so that these probabilities add up to one. If the player has strategies numbered* $\{1, \dots, m\}$ *then we represent a mixed strategy by an m-tuple*

$$(q_1, \dots, q_m)$$

*where* $q_i$ *is the probability that strategy* $i$ *will be employed, and such that* $q_1 + q_2 + \dots + q_m = 1$.

So how do we play according to a mixed strategy? Before the game starts, we employ a device which will give us a number from 1 to $m$, such that the result will be number $i$ with probability $q_i$. We then use the strategy thus decided upon. Clearly we still have the 'pure' strategies available to us: If we always want to use strategy $i$ we can do so by employing the mixed strategy $(0, \dots, 0, 1, 0, \dots 0)$, where the 1 occurs at the $i$th entry. We sometimes abbreviate this strategy as $i$, as we did when we were only considering 'pure' strategies.

Consider a game like Paper-Stone-Scissors. Assume you are playing it many times against your best friend. Clearly, if you decided to *always* play Paper then your friend would soon catch on and start to always play Scissors, thus winning every single time. Hence employing a pure strategy in this game is not a very good idea. What would be a good answer instead? Intuitively it is clear that all strategies are equivalent, and that if we remove one of them from our considerations we give the other player an advantage. In fact, *everything* that makes it easier for him to predict what we are going to do next will give him an advantage. Hence the intuitive solution to this game is that all three strategies should be employed with equal probability, meaning that the mixed strategy $(1/3, 1/3, 1/3)$ should come out as best.

This first of all raises the question of how we measure the performance of a mixed strategy, but that is easily solved: We just use the *expected pay-off* when playing according to this strategy. This makes it easy to assign a number to the outcome of playing a mixed strategy against the pure strategy of an opponent: Assume that the opponent has $n$ strategies, and that our strategy $i$ playing against his strategy $j$ gives a pay-off of $p_1(i, j)$. Then the mixed strategy $(q_1, \dots, q_m)$ employed against strategy $j$ will result in the expected pay-off

$$q_1 p_1(1, j) + q_2 p_1(2, j) + \dots + q_m p_1(m, j).$$

Let us consider our mixed strategy $(q_1, \dots, q_m)$ against his mixed strategy $(r_1, \dots, r_n)$, which will result in the expected pay-off for Player 1 (which we again refer to as $p_1$):

$$\begin{aligned} p_1((q_1,\dots,q_m),(r_1,\dots,r_n)) = & \quad q_1r_1p_1(1,1)+q_1r_2p_1(1,2)+\dots+q_1r_np_1(1,n)\\ & + \quad q_2r_1p_1(2,1)+q_2r_2p_1(2,2)+\dots+q_2r_np_1(2,n)\\ & + \quad \cdots\\ & + \quad q_mr_1p_1(m,1)+q_mr_2p_1(m,2)+\dots+q_mr_np_1(m,n)\\ = & \quad \sum_{i=1}^{m}\sum_{j=1}^{n} q_ir_jp_1(i,j). \end{aligned}$$

Let $S_1$ be the set of all mixed strategies for Player 1, and similarly $S_2$ for Player 2. Note that in general, these are *infinite* sets. Then the highest guaranteed expected pay-off that Player 1 can hope for (compare Amelia's considerations in Section 2.1) is

$$\max_{s\in S_1}\min_{t\in S_2} p_1(s,t).$$

Similarly, the best Player 2 can hope for (compare Scottie's thoughts in Section 2.1) is

$$\max_{t\in S_2}\min_{s\in S_1} p_2(s,t),$$

which for a 2-person zero-sum game is the same as paying the following amount to Player 1

$$\min_{t\in S_2}\max_{s\in S_1} p_1(s,t).$$

For pure strategies, these values did not have to coincide, but that becomes different when we consider mixed strategies. Proposition 2.6 tells us what we can say about pay-offs for equilibrium points in 2-person zero-sum games.

The concept of a mixed strategy and the corresponding pay-off functions for the players can be applied to any game given in its normal form. This gives rise to a more general definition of equilibrium point—for mixed strategies. there is no reason to restrict this to just 2 players and we give the more general form straight away.

**Definition 8** *Let $G$ be a non-cooperative game in normal form. A tuple of mixed strategies, one for each player, $(s_1,\dots s_l)$ is an* **equilibrium point** *of $G$ if for all $1 \le j \le l$ and all elements $s$ of the set of mixed strategies for Player $j$ it is the case that*

$$p_j(s_1,\dots,s_{j-1},\mathbf{s}_j,s_{j+1},\dots,s_l) \ge p_j(s_1,\dots,s_{j-1},\mathbf{s},s_{j+1},\dots,s_l).$$

Again it is the case that each player is being punished if he unilaterally moves away from an equilibrium point—his pay-off can only decrease, at best stay as it is.

It seems difficult at first sight to determine that a given tuple of mixed strategies is an equilibrium point for a game, but fortunately there is a proposition which tells us that we do not, in fact, have to match it against all the mixed strategies for all the players. Doing that with the pure ones suffices.

**Proposition 2.4** *A tuple of mixed strategies $(s_1,\dots s_l)$ is an equilibrium point for a non-cooperative game if and only if for all $1 \le j \le l$ and all* pure *strategies $k \in \{1,2,\dots,n_j\}$ for Player $j$ it is the case that*

$$p_j(s_1,\dots,s_{j-1},s_j,s_{j+1},\dots,s_l) \ge p_j(s_1,\dots,s_{j-1},k,s_{j+1},\dots,s_l).$$

**Proof.** Clearly this is a necessary condition for being an equilibrium point—the inequality is just a special case for the one given in the definition.

It is also sufficient. Let $s = (q_1, q_2, \ldots, q_{n_j})$ be a mixed strategy for Player $j$. Then it is the case that

$$\begin{aligned} p_j(s_1, \ldots, s_{j-1}, s, s_{j+1}, \ldots, s_l) &= \sum_{k=1}^{n_j} q_k p_j(s_1, \ldots, s_{j-1}, k, s_{j+1}, \ldots, s_l) \\ &\le \sum_{k=1}^{n_j} q_k p_j(s_1, \ldots, s_{j-1}, s_j, s_{j+1}, \ldots, s_l) \\ &= p_j(s_1, \ldots, s_{j-1}, s_j, s_{j+1}, \ldots, s_l) \sum_{k=1}^{n_j} q_k \\ &= p_j(s_1, \ldots, s_{j-1}, s_j, s_{j+1}, \ldots, s_l) \end{aligned}$$

as required. □

So we can *check* whether or not a given strategy tuple gives an equilibrium point. We return to our example of Paper-Stone-Scissors. We will use the pay-off matrix given on Page 20. We have claimed that the mixed strategies $(1/3, 1/3, 1/3)$ and $(1/3, 1/3, 1/3)$[19] together define an equilibrium point. We first calculate the expected pay-off which is

$$\begin{aligned} & p_1((1/3, 1/3, 1/3), (1/3, 1/3, 1/3)) = \\ & \quad (1/9 \times 0 + 1/9 \times 1 + 1/9 \times (-1)) + (1/9 \times (-1) + 1/9 \times 0 + 1/9 \times 1) \\ & \quad +(1/9 \times 1 + 1/9 \times (-1) + 1/9 \times 0) \\ &= 3 \times (1/9 \times 1) + 3 \times (1/9 \times (-1)) \\ &= 0. \end{aligned}$$

We then compare this mixed strategy to each of the pure strategies for Player 1.

$$p_1(1, (1/3, 1/3, 1/3)) = 0 \times 1/3 + 1 \times 1/3 - 1 \times 1/3 = 0.$$

Because all the strategies are equivalent, $p_1(2, (1/3, 1/3, 1/3))$ and $p_1(3, (1/3, 1/3, 1/3))$ evaluate to the same value. For symmetry reasons this argument also applies to comparing this mixed strategy to Player 2's pure strategies. By Proposition 2.4, we have indeed found an equilibrium point.

**Exercise 14** (a) Show that the game with the pay-off matrix given below has the mixed strategy equilibrium $((1/2, 0, 0, 1/2), (1/4, 1/4, 1/2))$.

$$\begin{vmatrix} -3 & -3 & 2 \\ -1 & 3 & -2 \\ 3 & -1 & -2 \\ 2 & 2 & -3 \end{vmatrix}$$

[19]Since the game is entirely symmetric it should come as no surprise that the solution is symmetric as well.

(b) Consider the following game. Alice has an Ace and a Queen, while Bob has a King and a Joker. It is assumed that the Ace beats the King which beats the Queen, whereas the Joker is somewhat special. Both players pay an ante of one pound into the pot. Then they select a card, each from his or her hand, which they reveal simultaneously. If Bob selects the King then the highest card chosen wins the pot and the game ends. If Bob chooses the Joker and Alice the Queen they split the pot and the game ends. If Bob chooses the Joker and Alice the Ace then Alice may either resign (so that Bob gets the pot) or demand a replay. If a replay occurs they each pay another pound into the pot and they play again, only this time Alice does not get the chance to demand a replay (so Bob gets the pot if he chooses the Joker and Alice the Ace).

Draw a game tree for this game and then bring it into matrix form. If you have your strategies in the same order as I do then you can show that an equilibrium is given by Alice's mixed strategy $(0, 1/8, 1/4, 5/8)$ and Bob's mixed strategy $(1/4, 1/2, 1/4)$. (If this doesn't come out as an equilibrium point for you then it's probably because you numbered your strategies differently. Don't worry about it!)

The introduction of mixed strategies solves the problem of non-existince of equilibrium points.

**Theorem 2.5 (Nash)** *Every non-cooperative game has at least one mixed strategy equilibrium point.*

**Proof.** This proof requires too much mathematical background to be appropriate for this course. □

**Remark.** This theorem becomes wrong if we want to cover infinite games as well.

It is also worthwhile to point out that the proof of Nash's Theorem does not lead to an algorithm for finding mixed strategy equilibrium points. In fact, no such algorithm is currently known for games with 3 or more players.

Once again it is the case that having found all equilibrium points lead to the same pay-off in the case of zero-sum 2-person games.

**Proposition 2.6** *For a 2-person zero-sum game all equilibrium points, whether they consist of pure or mixed strategies, lead to the same pay-off, which we call the* value *of the game.*

**Proof.** One possible proof of this result is similar to that of Proposition 2.1, just using the definition of the pay-off function for mixed strategies. There is an alternative proof which we present here. Let $(s, t)$ and $(s', t')$ be equilibrium points for the game under consideration. (If there are any pure strategies involved we still think of them as represented as mixed strategies, with a probability of 0 being given to all but the chosen strategy.) These lead to pay-offs $p(s, t)$ and $p(s', t')$ respectively for Player 1. By the definition of an equilibrium point, we know that if one player changes away from an equilibrium point, his or her pay-off can only decrease. Hence

$$p(s, t) \geq p(s', t) \geq p(s', t') \geq p(s, t') \geq p(s, t).$$

The first inequality is Player 1 changing away from $(s, t)$, the second is Player 2 changing away from $(s', t')$, the third Player 1 changing away from $(s', t')$, and the last Player 2 changing away from $(s, t)$. But the above chain of inequalities means that all these numbers have to be equal, in particular $p(s, t) = p(s', t')$. □

There are a few more results for mixed strategy equilibrium points for 2-person zero-sum games which we will briefly touch on. If $(s, t)$ is an equilibrium point for a matrix game then we say that $s$ is **optimal** for Player 1 and that $t$ is **optimal** for Player 2. It turns out that if $s$ and $s'$ are both optimal for Player 1 then we can combine them to give another optimal strategy for Player 1, and, in fact, there are infinitely many ways of doing so: We merely have to decide on a weight $0 \leq \lambda \leq 1$ to assign to $s$, and then we assign the weight $(1 - \lambda)$ to $s'$. We obtain the new optimal strategy by taking an entry in $s$, multiply it by $\lambda$ and adding it to $(1 - \lambda)$ times the corresponding entry of $s'$.

Let $s$ and $t$ be optimal strategies for Player 1 and 2 respectively. Let $i$ be a pure strategy for Player 1. If

$$p(i, t) < p(s, t)$$

then $i$ is no part of *any* optimal strategy for Player 1. In other words, for all optimal strategies for Player 1, the $i$th component is 0.

## 2.5 Finding equilibria in 2-person zero-sum games

As we have said before there is no algorithm which will solve games of 3 and more players. For more restricted games, however, solutions to this problem are known. In this section we restrict ourselves to 2-person zero-sum games in their matrix form. For this section we will therefore assume that all games considered are of this kind.

One possibility is to reformulate the problem of finding an equilibrium point in mixed strategies for such a game into one of maximizing (or minimizing) a linear function subject to some constraints. This fits into a discipline which used to be known as *linear programming*. The *simplex algorithm* delivers a solution to such problems. It does not require much mathematical background, merely a modest knowledge of matrices and their algebra. Nonetheless we will not cover it here.

An intuitive method for dealing with this problem (again for 2-person zero-sum games in matrix form) is the following. It is sometimes possible to inspect a matrix game and decide that a particular strategy is not at all desirable—because it is outperformed by other strategies. Consider the game given by the following matrix.

$$\begin{vmatrix} 1 & -1 & 2 \\ -1 & 1 & 3 \\ -3 & -2 & 4 \end{vmatrix}$$

Player 2 wishes to minimize her pay-out to Player 1, and from her point of view her strategy 3 is particularly undesirable: If she compares it point by point with her strategy 1 she notices that if Player 1 chooses his strategy 1 then she will only have to pay 1 if she chooses her strategy 1 as opposed to 2 if she selects her strategy 3. If Player 1 goes for his strategy 2, the values become a win of 1 *versus* a pay-out of 3, and for Player 1's strategy 3 a win of 3 compared to a pay-out of 4. From Player 2's point of view it is very clear: Her strategy 3 is utterly useless.

**Definition 9** *We say that a strategy $i$ for Player 1* **dominates** *another such strategy $i'$ for the same player if it is the case that for all strategies $1 \leq j \leq n$ for Player 2*

$$a_{i,j} \geq a_{i',j}.$$

*We say that a strategy $j$ for Player 2* **dominates** *another such strategy $j'$ for the same player if it is the case that for all strategies $1 \leq i \leq m$ for Player 1*

$$a_{i,j} \leq a_{i,j'}.$$

If this is the case then we may remove such strategies from consideration and make the game under consideration smaller and thus simpler.[20] By the above considerations, we remove strategy 3 for Player 2 from consideration, leading to the game matrix

$$\begin{vmatrix} 1 & -1 \\ -1 & 1 \\ -3 & -2 \end{vmatrix}$$

This new game allows the consideration that Player 1's strategy 3 is dominated by both, strategies 1 and 2, so out it goes, leaving us with

$$\begin{vmatrix} 1 & -1 \\ -1 & 1 \end{vmatrix}$$

This is a symmetric matrix game allowing for the particularly simple solution: the mixed strategies $(1/2, 1/2)$, $(1/2, 1/2)$ define an equilibrium point. If we wish to formulate this solution in terms of the original game, all we have to do is to turn those into mixed strategies $(1/2, 1/2, 0)$ and $(1/2, 1/2, 0)$.

If a game does possess an equilibrium point of pure strategies then it can be reduced by removing strategies dominated by others until only one entry is left: the value of the game.[21]

**Exercise 15** Reduce the games given by the matrices below via dominance consideration. If you can solve them, do so!

$$\text{(a)} \quad \begin{vmatrix} 2 & 4 & 0 & -2 \\ 4 & 8 & 2 & 6 \\ -2 & 0 & 4 & 2 \\ -4 & -2 & -2 & 0 \end{vmatrix} \qquad \text{(b)} \quad \begin{vmatrix} 2 & -3 & 1 & -4 \\ 6 & -4 & 1 & -5 \\ 4 & 3 & 3 & 2 \\ 2 & -3 & 2 & -4 \end{vmatrix}$$

This process allows us to reduce the problem of finding a solution (that is an equilibrium point) to one of a smaller size, but it is slightly limited in its application because it is unnecessarily restrictive. There is no reason to restrict dominance to pure strategies.

One can define a notion of dominance of mixed strategies which will be a straightforward generalization of Definition 9. However, our only application of this will be to remove pure strategies from consideration and thus reducing the normal form of the game to a smaller one. Therefore we restrict ourselves to defining merely what we need to do this.

[20] We may be losing some solutions by applying this technique. This is because we allow the removal of strategies which are 'no better' than remaining ones—but the removed strategy might be just as good as the optimal strategy that remains. However, if we are only interested in finding *one* solution then this is of no consequence. If we wanted to find *all* equilibrium points then we would have to make sure that there is a $>$ ($<$) wherever a $\geq$ ($\leq$) appears in Definition 9.

[21] This also tells us whether a given matrix corresponds to a 2-person game of perfect information: It does so precisely if it can be reduced to a $1 \times 1$ matrix.

**Definition 10** *A pure strategy $i$ for Player 1 is* **dominated** *by a mixed strategy $(q_1, \ldots, q_m)$ for the same player if for all pure strategies $j$ of Player 2 it is the case that the expected pay-off for strategy $i$ played against strategy $j$ is less than or equal to the pay-off for strategy $(q_1, \ldots, q_m)$ against strategy $j$. In other words, for all $1 \leq j \leq n$:*

$$a_{i,j} \leq q_1 a_{1,j} + q_2 a_{2,j} + \cdots + q_m a_{m,j}.$$

*The notion of domination for strategies for Player 2 is defined in the obvious dual way.*

Consider the following game as an example.

$$\begin{vmatrix} -1 & 2 \\ 2 & -1 \\ 0 & 0 \end{vmatrix}$$

No (pure) strategy for Player 1 is dominated by any other. Similarly for Player 2's two strategies.

Now we're faced with the problem of finding out whether we can remove a strategy for either player by using the idea of mixed dominance. Who should we start with, Player 1 or Player 2? As a rule of thumb, if one player has more strategies than the other, he or she is a good target, and so we will start with Player 1.

So which of Player 1's three strategies might be dominated by a mix of the other two? Let's consider his strategy 1. Its first component is $-1$, which is smaller than both 2 and 0, so it will be smaller than

$$2\lambda + 0(1 - \lambda)$$

for all $0 \leq \lambda \leq 1$. It is worth here to pause and note the following: If $0 \leq \lambda \leq 1$, and $a \leq b$ are two real numbers, then

$$a \leq \lambda a + (1 - \lambda)b \leq b$$

in other words, $\lambda a + (1 - \lambda)b$ *is always between* $a$ *and* $b$. The second component of Player 1's strategy 1 is 2, which cannot possibly be below a 'mixture' of $-1$ and 0. Hence Player 1's strategy 1 is not dominated by a mixture of the other two.

If we make the corresponding considerations for his strategy 2 we come to a similar result, which leaves his strategy 3. Both its entries are 0, which is between $-1$ and 2, making it a valid candidate for elimination. Note that we can't be *sure* yet that we really can eliminate it—for that we need to find *one* $0 \leq \lambda \leq 1$ which satisfies *all* of the following inequalities:

$$\begin{aligned} 0 &\leq -\lambda + 2(1 - \lambda) = 2 - 3\lambda \\ 0 &\leq 2\lambda - (1 - \lambda) = 3\lambda - 1 \end{aligned}$$

The former is equivalent to

$$\lambda \leq \frac{2}{3}$$

and the latter to

$$\lambda \geq \frac{1}{3},$$

so $\lambda = 1/3$ will do the job. Hence Player 1's strategy 3 is dominated by his mixed strategy $(1/3, 2/3, 0)$.

Of course, removing strategies which are dominated by other (mixed) strategies only makes sense if the dominated strategy does not contribute to the mixed strategy. In other words, if $i$ is dominated by $(q_1, \dots, q_m)$ then we are only allowed to remove it if $q_i = 0$. We do, of course, need a result that tells us that this is a safe thing to do.

**Proposition 2.7** *Let $G'$ be a game that results from the 2-person zero-sum game $G$ by removing a strategy $i$ for Player 1 which is dominated by some mixed strategy $(q_1, \dots, q_m)$ with $q_i = 0$. If $(q'_1, \dots, q'_{i-1}, q'_{i+1}, \dots, q'_m)$ is an optimal strategy for Player 1 in the game $G'$ then $(q'_1, , \dots, q'_{i-1}, 0, q'_{i+1}, \dots, q'_m)$ is an optimal strategy for Player 1 in the game $G$.*

*An analogous result holds for Player 2.*

The proof of this result is lengthy but not too difficult—it consists of making calculations regarding pay-off functions. Since it is of no particular interest we omit it here.

Finally we will discuss how to solve games with 2 strategies for each player *graphically*. Consider the game given by the matrix from page 33, which has no pure strategy equilibrium points.

$$\begin{vmatrix} 5 & 1 \\ 3 & 4 \end{vmatrix}$$

If Player 2 plays her strategy 1 then by using a mixed strategy, Player 1 can achieve any pay-off from 5 to 3. This is demonstrated in Figure 14. This figure shows the graph of $y = -2x + 5$. (Note that $x$ gives the proportion of strategy 2!)

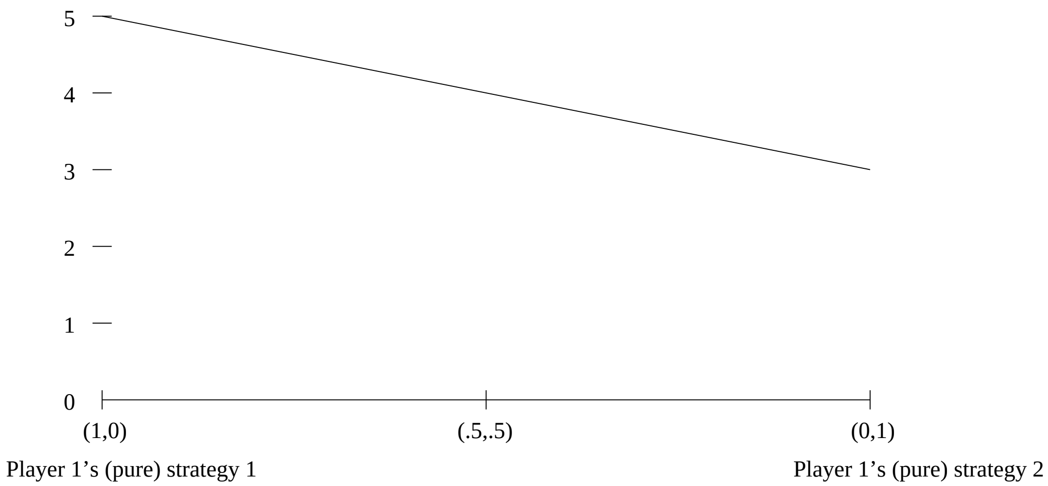



Figure 14: Player 1's pay-off against Player 2's strategy 1

If, on the other hand, Player 2 plays her strategy 2 then Player 1's pay-off will vary from 1 to 4 with a mixed strategy, as pictured in Figure 15. The line pictured in this figure is given by the function $y = 3x + 1$. (Again, $x$ gives the proportion of strategy 2 in the mix.)

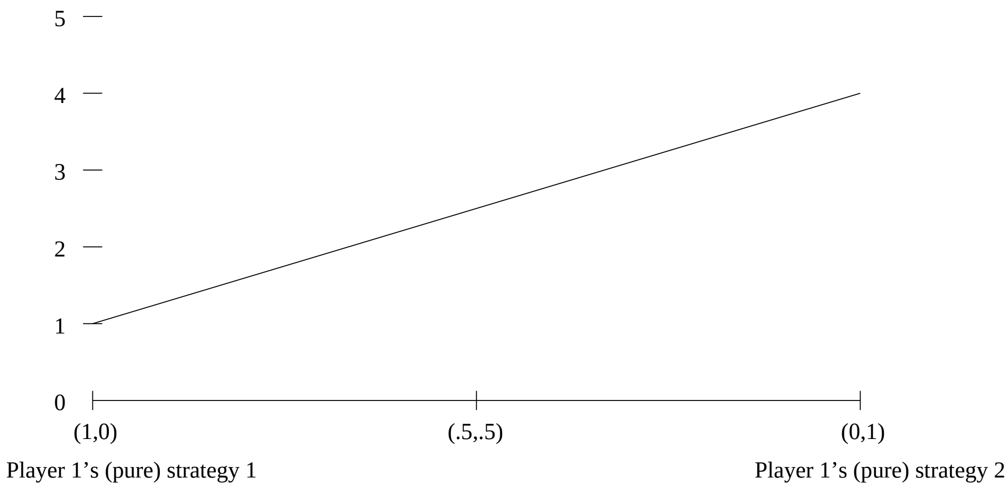



Figure 15: Player 1's pay-off against Player 2's strategy 2

By superimposing these two images we get some idea of the *minimum pay-off* that Player 1 can expect when using a mixed strategy, see Figure 16. Here the dashed lines are the ones from Figures 14 and 15, and the solid line gives the minimum of the two, which represents the expected minimum pay-off for Player 1.

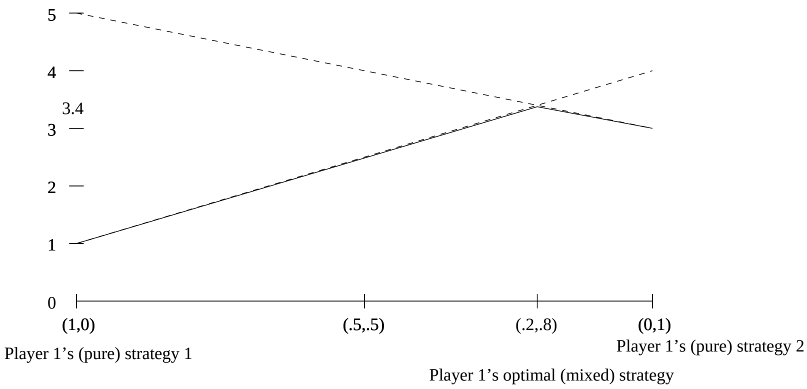



Figure 16: Player 1's minimum pay-off and his optimal strategy

Note that in the way we represent the pure strategies in the graph, with strategy 1 $(1, 0)$ on the left and strategy 2 $(0, 1)$ on the right, it is the *second* coordinate which goes from 0 to 1, and that is the one that we use to describe the various lines via equations. Hence $x$ below always gives the proportion of the *second* strategy in the mix.

Since Player 1 is attempting to *maximize* this minimum expected pay-off, his solution is to find the mixed strategy which corresponds to the maximum on that solid line in Figure 16. In order to find it, we have to find the intersection of the two lines in the picture,

$$y = -2x + 5 \qquad \text{and} \qquad y = 3x + 1.$$

This can be done, for example, by equating the two right hand sides

$$-2x + 5 = 3x + 1,$$

and isolating $x$:

$$5x = 4 \qquad \text{and so} \qquad x = 4/5.$$

Note that $x$ gives the proportion of strategy 2, and that of strategy 1 is obtained by deducting $x$ from 1. Hence Player 1's optimal strategy is $(1/5, 4/5)$.

Similarly we can summarize the situation for Player 2, which is done in Figure 17. Player 2, of course, is concerned with minimizing the maximum of her expected pay-offs.

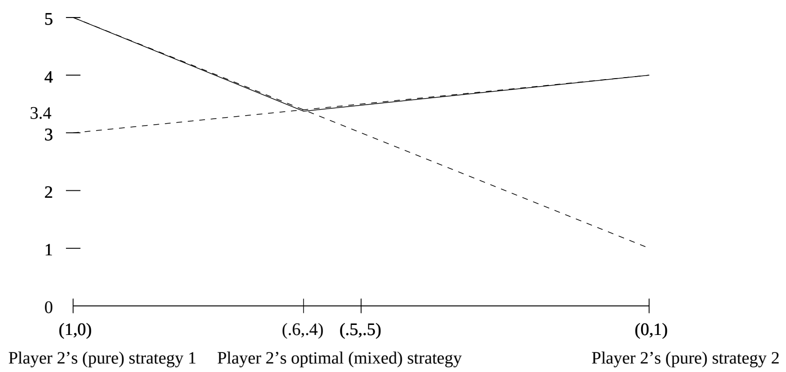



Figure 17: Player 2's maximum pay-off and her optimal strategy

The optimal strategy for Player 2 is $(3/5, 2/5)$. It is also possible to read off the *value* of the game from Figures 16 and 17: It is given by the pay-off when playing according to the optimal strategies (the $y$-value of the intersection of the two lines) and is 3.4 in this example.

**Exercise 16** Solve the games given by the following matrices.

$$\text{(a)} \quad \begin{vmatrix} 16 & 14 & 6 & 11 \\ -14 & 4 & -10 & -8 \\ 0 & -2 & 12 & -6 \\ 22 & -12 & 6 & 10 \end{vmatrix} \qquad \text{(b)} \quad \begin{vmatrix} 0 & 3 & 6 & 5 \\ 15 & 10 & 8 & 9 \\ 10 & 15 & 11 & 7 \\ 5 & 9 & 4 & 2 \end{vmatrix}$$

**Exercise 17** Reduce the following matrices to the size of $2 \times 2$ using dominance arguments. Note that these are less straight-forward than (a) and (b) in Exercise 6 although they are smaller.

$$\text{(a)} \quad \begin{vmatrix} 2 & 1 & 0 \\ 2 & 0 & 3 \\ -1 & 3 & -3 \end{vmatrix} \qquad \text{(b)} \quad \begin{vmatrix} 6 & 0 & 3 \\ 8 & -2 & 3 \\ 4 & 6 & 5 \end{vmatrix}$$

If you didn't manage to solve the games in the previous exercise, try again now!

**Exercise 18** Reduce the following matrices to the size of $2 \times 2$ using dominance arguments. These matrices are tougher than those in the previous exercise!

$$\text{(a)} \quad \begin{vmatrix} 0 & 1 & 2 & 3 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \end{vmatrix} \qquad\qquad \text{(b)} \quad \begin{vmatrix} 0 & 2 & 2 & 3 \\ 3 & 2 & 1 & 0 \\ 2 & 0 & 3 & 1 \end{vmatrix}$$

## 2.6 An extended example: Simplified Poker

This section is for the most part taken from A.J. Jones book on game theory (see the list of sources), and she attributes the study to H.W. Kuhn.[22]

Poker as it is actually played (it does not particularly matter which variant we are talking about) is much too big for a detailed analysis. We radically simplify it by fixing the number of players at 2, and by assuming there are only 3 cards (which each player being dealt a hand of 1). While this would not be a very interesting game to play in practice, it turns out that many issues of playing 'real' Poker appear when analysing this simple game, for example, that of bluffing, or folding with a hand which appears quite good. The game is as described in Example 3. We repeat the rules here for convenience.

Each of the two players has to pay one unit to enter a game (the *ante*). They then are dealt a hand of one card each from a deck containing three cards, labelled $J$, $Q$ and $K$. The players then have the choice between either betting one unit or passing. The game ends when

- either a player passes after the other has bet, in which case the better takes the money on the table (the *pot*),
- or there are two successive passes or bets, in which case the player with the higher card ($K$ beats $Q$ beats $J$) wins the pot.

There are six possible deals (Player 1 might get one of three cards, and Player 2 will get one of the two remaining cards, making $3 \times 2 = 6$ possibilities). For each of these deals, the subsequent game tree is given in Figure 18, and we briefly explain the notation used there. The players' options in each case are to bet ($B$) or to pass ($P$). The result is expressed by giving the winner (1 is Player 1, 2 is Player 2 and $H$ is the holder of the higher card) and the amount he or she will win (where we have deducted the player's own contribution to the pay-off), that is 1:1 means that Player 1 wins 1 unit.

If we consider the available strategies for Player 1 we find that he has to make a decision whether to pass or to bet, depending on his card, for round 1 and a potential round 2. We record his strategies in tables, where for example

| $J$ | $Q$ | $K$ |
|---|---|---|
| $PB$ | $PP$ | $B$ |

means that if Player 1 has the Jack ($J$) he will pass in the first round and bet in the second (if they get that far), if he has the Queen ($Q$) he will always pass, and if he has the King ($K$), he will bet in the first round. But we do not really need a table to gather all the information: After all, the order $J$, $Q$, $K$ is constant. Hence we can use a triple $(PB, PP, B)$ to denote the

[22] The full article is: H.W. Kuhn, **A simplified two-person poker.** In: *Contributions to the Theory of Games*, I, Ann. Math. Studies No. 24, 105–116, 1950.

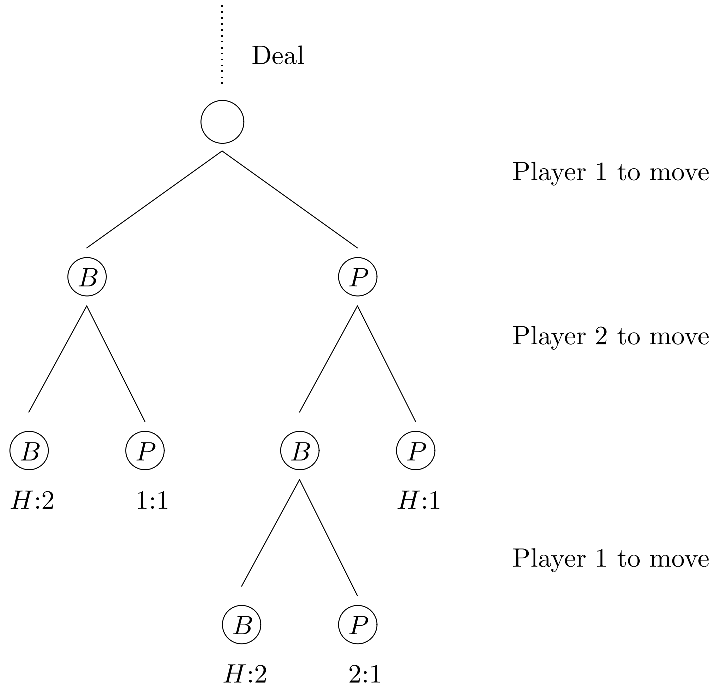



Figure 18: Simplified Poker

same information. Since Player 1 has three choices, $PP$, $PB$, and $B$, for each of the three cards he might possibly be dealt, he has $3 \times 3 \times 3 = 3^3 = 27$ (pure) strategies.

Now for Player 2. Her strategies may depend on her own card (one of three) *as well as Player 1's move in the first round* (one of two). For each of her potential cards, there are four strategies (giving an answer for the two first moves Player 1 might have made), giving $4^3$ strategies altogether. Again we encode these as triples which tell her how to make a choice based on her card. However, her potential moves depend on Player 1's first move as well, and therefore we need a different encoding. We use the four options $P|P$, $P|B$, $B|P$, and $B|B$, where the first component says what to do if Player 1 bets, and the second what to do if hs passes. So $P|B$, for example means to pass if Player 1 bets and to bet if Player 1 passes.

We wish to solve this game, but at the moment it is still somewhat large. Hence we will first think about which strategies will obviously be unsuccessful before trying to list all of them. Clearly, when confronted with a bet, a player should bet if he or she has the $K$, and pass if he or she has the $J$: the former guarantees a win, and the latter will result in a loss of 1 rather than 2 units. (If we assembled the full matrix for the game we could see that the purportedly inferior strategies are dominated by the corresponding ones claimed to be superior, but that matrix is rather large.)

Let us go through this argument in more detail. We start with Player 1. If Player 1 has the $J$, and the course of play so far has been $P, B$ then if Player 1 decides to bet, he will lose 2 units rather than just the 1 he will lose if he passes in that situation. Hence we remove all strategies which have $PB$ in their first component (which describes what to do if one's card is the $J$). If Player 1 has the $K$, on the other hand, then if the course of play so far has been $P, B$ he should definitely bet, guaranteeing a win of 2 rather than just 1 unit. Therefore we remove all strategies which have $PP$ in their last component (which describes what to do if one's card is the $K$). That leaves us with $2 \times 3 \times 2 = 12$ strategies for Player 1.

Now we apply similar considerations to Player 2. If she is faced with a bet, that is the course of play so far has been $B$, and she holds the $J$, then she will lose 1 unit if she passes, and 2 if she bets, so passing is the better option. Therefore we remove all strategies that have $B|P$ or $B|B$ in their first component. If Player 2 has the $K$, on the other hand, she should *never* pass, that would only decrease her potential winnings. (Note that these considerations do *not* apply if Player 1 has the $K$!) When faced with a course of play so far of $P$, then if she passes too she will win 1 unit, whereas she might win 2 if she were to bet. When faced with a course of play so far of $B$ then by betting in return she will win 2 rather than lose 1. Hence we insist that all her strategies have $B|B$ as their last component. That leaves $2 \times 4 \times 1 = 8$ strategies for her.

Now that we have removed some strategies from consideration we can use this information to get rid of more: We may use the fact that the strategies we have removed already are out of consideration. If Player 1 holds the $Q$, he should pass in the first round and only bet if Player 2 bets in her first go: If Player 2 has the $K$ she will definitely bet no matter what (as we have just ruled out all other strategies). Hence Player 1 will lose 2 if he bets in the first round, and the same amount if he instead first passes and then bets in the second round. If Player 2 has the $J$, on the other hand, then she will pass when confronted with a bet (we have ruled out all her other strategies in the previous paragraph). This gives the following possibilities when comparing the strategies $B$ and $PB$ under the assumption that Player 1 has the $Q$.

<table>
<tr><td colspan="6">P1 has the $Q$</td></tr>
<tr><td>P1's strategy</td><td colspan="2">$B$</td><td colspan="3">$PB$</td></tr>
<tr><td>P1's first move</td><td colspan="2">$B$</td><td colspan="3">$P$</td></tr>
<tr><td>P2's card</td><td>$J$</td><td>$K$</td><td colspan="2">$J$</td><td>$K$</td></tr>
<tr><td>P2's move</td><td>$P$</td><td>$B$</td><td>$B$</td><td>$P$</td><td>$B$</td></tr>
<tr><td>P1's snd move</td><td></td><td></td><td>$B$</td><td></td><td>$B$</td></tr>
<tr><td>pay-off for P1</td><td>1</td><td>$-2$</td><td>2</td><td>1</td><td>$-2$</td></tr>
</table>

If we compare the possible outcomes of playing $B$ *versus* playing $PB$ when Player 1 holds the $Q$, we find that he can only improve his situation when choosing $PB$. The only difference occurs when Player 2 holds the $J$: she might be tempted into betting when facing a pass (in which case there is a chance of Player 1 winning 2 units rather than 1) but she certainly won't do so when faced with a bet. Hence we remove all the strategies for Player 1 which have a $B$ in the second component. That leaves $2 \times 2 \times 2 = 8$ strategies for Player 1.

Finally we look at the case of Player 2 holding the $Q$ and facing a pass. We are only considering the middle component of Player 2's strategies here (assuming she does indeed have the $Q$), which is independent from the others. Hence it must be the case that Player 1 holds either the $J$ or the $K$, and we only have to consider those components of Player 1's strategies. This leaves us with four strategies for Player 1 (combining $PP$ or $B$ for the first component with $PB$ or $B$ for the last component), and the four possibilities for the middle component of Player 2's strategies: $P|P$, $B|P$, $P|B$, and $B|B$.

To fill in the resulting table we have to work out how a strategy for Player 2 with the given middle component would do against a strategy for Player 1 with the given first and third component. Since either case occurs with the same probability and we are only interested in dominance we do not have to take probability factors into account.

So if a strategy for Player 1 of the form $(PP, \,, PB)$ faces a strategy for Player 2 of the form $(\,, P|P, \,)$ then, under the assumption that she has the $Q$ and he has either the $J$ or the $K$, there are two cases which are equally likely.

Player 1 has the $J$: The course of play will be $P, P$ and the pay-off will be $-1$ for Player 1.

Player 1 has the $K$: The course of play will be $P, P$ and the pay-off will be 1 for Player 1.

Hence we can summarize this case by saying that the expected pay-off will be 0. Treating all the other cases in the same way we obtain the following full table.

| | Second component | | | |
|---|---|---|---|---|
| | $P\|P$ | $B\|P$ | $P\|B$ | $B\|B$ |
| $(PP, \,, PB)$ | 0 | 0 | 1 | 1 |
| $(PP, \,, B)$ | 0 | 1 | 0 | 1 |
| $(B, \,, PB)$ | 2 | $-1$ | 3 | 0 |
| $(B, \,, B)$ | 2 | 0 | 2 | 0 |

We notice that strategy 3 for Player 2 is dominated by strategy 1 and that strategy 4 for the same player is dominated by strategy 2. This means that having $P|B$ or $B|B$ in the second component is something that Player 2 should avoid.[23]

We can argue the same case without looking at a matrix. Assume Player 2 holds the $Q$ and play so far consists of a pass $P$ by Player 1. If she bets then if Player 1 holds the $J$ he will pass (because we have excluded his other strategies) and she will win 1, and if he holds the $K$ he will bet (again because we have excluded his other strategies) and she will lose 2. If, on the other hand, she passes then she will once again win 1 if Player 1 holds the $J$, but only lose 1 if he holds the $K$.

Hence we we strike all her strategies which have $P|B$ or $B|B$ in the second component. That leaves $2 \times 2 \times 1 = 4$ strategies for Player 2 ($P|P$ or $B|P$ in the first component, the same for the second, and $B|B$ in the third).

Discarding all these strategies makes it feasible to look at the full matrix of the remaining strategies for this game. To calculate the pay-off function for playing, say, $(PP, PP, PB)$ against $(P|P, P|P, B|B)$, we have to calculate the expected pay-off. For this we must consider all possible deals. We give these as pairs, with the first component the card of Player 1, and the second component that of Player 2. Each of these deals occurs with the probability 1/6.

$$(PP, PP, PB) \qquad \textit{versus} \qquad (P|P, P|P, B|B):$$

**$(\mathbf{J}, \mathbf{Q})$:** The moves played are $P, P$ and Player 1 gets $-1$.

**$(\mathbf{J}, \mathbf{K})$:** The moves played are $P, B, P$ and Player 1 gets $-1$.

**$(\mathbf{Q}, \mathbf{J})$:** The moves played are $P, P$ and Player 1 gets 1.

[23] It should be pointed out that we could argue as we did only because all strategies consist of independent components.

**$(\mathbf{Q},\mathbf{K})$:** The moves played are $P, B, P$ and Player 1 gets $-1$.

**$(\mathbf{K},\mathbf{J})$:** The moves played are $P, P$ and Player 1 gets 1.

**$(\mathbf{K},\mathbf{Q})$:** The moves played are $P, P$ and Player 1 gets 1.

Hence the expected pay-off when playing $(PP, PP, PB)$ against $(P|P, P|P, B|B)$ is

$$(1/6 \times (-1)) + (1/6 \times (-1)) + (1/6 \times 1) + (1/6 \times (-1)) + (1/6 \times 1) + (1/6 \times 1) = 0.$$

We give the full matrix, but to make it easier to compare the entries, we multiply all of them by 6. So the true game matrix is 1/6 times the one given below.

| | $(P\|P, P\|P, B\|B)$ | $(P\|P, B\|P, B\|B)$ | $(P\|B, P\|P, B\|B)$ | $(P\|B, B\|P, B\|B)$ |
|---|---|---|---|---|
| $(PP, PP, PB)$ | $0$ | $0$ | $-1$ | $-1$ |
| $(PP, PP, B)$ | $0$ | $1$ | $-2$ | $-1$ |
| $(PP, PB, PB)$ | $-1$ | $-1$ | $1$ | $1$ |
| $(PP, PB, B)$ | $-1$ | $0$ | $0$ | $1$ |
| $(B, PP, PB)$ | $1$ | $-2$ | $0$ | $-3$ |
| $(B, PP, B)$ | $1$ | $-1$ | $-1$ | $-3$ |
| $(B, PB, PB)$ | $0$ | $-3$ | $2$ | $-1$ |
| $(B, PB, B)$ | $0$ | $-2$ | $1$ | $-1$ |

One can easily verify that the following 12 mixed strategies are optimal for Player 1 for this matrix (and thus for Simplified Poker):[24]

$$\begin{array}{llllllll}
2/3 & (PP,PP,PB) & + & 1/3 & (PP,PB,PB) & & & \\
1/3 & (PP,PP,PB) & + & 1/2 & (PP,PB,B) & + & 1/6 & (B,PP,PB) \\
5/9 & (PP,PP,PB) & + & 1/3 & (PP,PB,B) & + & 1/9 & (B,PB,PB) \\
1/2 & (PP,PP,PB) & + & 1/3 & (PP,PB,B) & + & 1/6 & (B,PB,B) \\
2/5 & (PP,PP,B) & + & 7/15 & (PP,PB,PB) & + & 2/15 & (B,PP,PB) \\
1/3 & (PP,PP,B) & + & 1/2 & (PP,PB,PB) & + & 1/6 & (B,PP,B) \\
1/2 & (PP,PP,B) & + & 1/3 & (PP,PB,PB) & + & 1/6 & (B,PB,PB) \\
4/9 & (PP,PP,B) & + & 1/3 & (PP,PB,PB) & + & 2/9 & (B,PB,B) \\
1/6 & (PP,PP,B) & + & 7/12 & (PP,PB,B) & + & 1/4 & (B,PP,PB) \\
5/12 & (PP,PP,B) & + & 1/3 & (PP,PB,B) & + & 1/4 & (B,PB,PB) \\
1/3 & (PP,PP,B) & + & 1/3 & (PP,PB,B) & + & 1/3 & (B,PB,PB) \\
2/3 & (PP,PB,B) & + & 1/3 & (B,PP,B) & & &
\end{array}$$

Player 2 has far fewer optimal strategies, namely just the following two

$$\begin{array}{llllllll}
1/3 & (P|P,P|P,B|B) & + & 1/3 & (P|P,B|P,B|B) & + & 1/3 & (P|B,P|P,B|B) \\
2/3 & (P|P,P|P,B|B) & + & 1/3 & (P|B,B|P,B|B). & & &
\end{array}$$

[24] In, fact, these are *all* Player 1's optimal strategies for this game, something we cannot be sure of when deleting dominated strategies.

There are ways of categorizing these strategies using *behavioural parameters*, but we will not look at those here. The interested reader is encouraged to look up the treatment in Jones's version or to go back to the original article.

It is worth pointing out that two practices employed by experienced Poker players play a role in Player 1's arsenal of optimal strategies, namely *bluffing* and *underbidding*.

Bluffing means betting with a $J$ (a card which is bound to lose if a comparison is forced) and underbidding refers to passing (at least initially) with a $K$ (a card that is bound to win under any circumstances). Almost all Player 1's optimal strategies involve *both* those. Similarly, all of Player 2's optimal strategies involve bluffing. Bluffing is not really an option for her, due to the specific rules of our game.[25]

When looking at ordinary Poker it is commonly observed that the average player will

- not bluff often enough—people generally have the feeling that they should be able to win a 'show-down' before betting on their cards;
- not underbid often enough. This case is slightly more complicated. In ordinary Poker (as opposed to our 'baby' version here), underbidding is usually more varied. When people have a reasonably good hand, they will almost inevitably bet fairly highly on it, assuming erroneously that their having an above-average hand must mean that nobody else can have one (let alone one which beats theirs). While our example is too simple to show this effect, we get at least its shadow.

The value of Simplified Poker is $-1/18$, so Player 1 will lose on average. In this simplified version of the game having to take the initiative is a disadvantage. In other such variants the opposite is true. The lesson of this section is that game theory can indeed help us to improve our game at the kinds of games people play as a pastime. If the actual game is too large to analyse, looking at simplified versions can produce useful guidelines.

**Exercise 19** Alice and Bob play the following form of simplified Poker. There are three cards, $J$, $Q$ and $K$, which are ranked as in the above example. Each player puts an ante of one pound into the pot, and Alice is then dealt a card face down. She looks at it and announces 'high' or 'low'. To go 'high' costs her 2 pounds paid into the pot, to go 'low' just 1. Next Bob is dealt one of the remaining cards face down. He looks at it and then has the option to 'fold' or 'see'. If he folds the pot goes to Alice. If he wants to see he first has to match Alice's bet. If Alice bet 'high' the pot goes to the holder of the higher, if she bet 'low' it goes to the holder of the lower card.

Draw the game tree for this game, indicating the information sets. Convince yourself that Alice has 8 (pure) strategies and that Bob has 64. Discard as many of these strategies you can by arguing that there are better alternatives. You should be able to get the game down to 2 strategies for Alice and 4 for Bob. Find the matrix for the reduced game and solve it.[26]

[25] If one assumes that both, the ante as well as the amount players can place on a bet, are real numbers, then one can play with these parameters and see their effect on the optimal strategies. Some of these variants rule out bluffing. Again, see Jones for a discussion of this.

[26] This exercise isn't easy—certainly bigger than anything I would consider as an exam question. But good practice in reasoning about strategies!

### Summary of Section 2

- The solution to a game is given by *equilibrium points*.
- For zero-sum games equilibrium points make sense, but for other games they are doubtful.
- In order to guarantee the existence of an equilibrium point we may have to switch to *mixed strategies*, where strategies are chosen with certain probabilities.
- Equilibrium points have the property that if any one player moves away from them unilaterally, his pay-off can only get worse.
- Every (non-cooperative) game has at least one equilibrium point (of mixed strategies).
- In 2-person zero-sum games, all equilibrium points lead to the same pay-off, the *value of the game*. The value is the least pay-off that Player 1 can guarantee for himself, while Player 2 can ensure that it is the highest amount she may have to pay to Player 1. In such a game, it makes sense to talk of equilibrium strategies as *optimal* for the respective player. If the game is one of perfect information, then an equilibrium point consisting of pure strategies exist.
- In practice, we can make a game matrix smaller by removing *dominated strategy*, and there is an easy way of solving $(2 \times 2)$ matrices.

### Sources for this section


Antonia J. Jones. **Game Theory: Mathematical models of conflict.** *Horwood Publishing*, Chichester, 2000.

Melvin Dresher. **Games of Strategy: Theory and Applications.** *Prentice-Hall International, Inc.* 1961.

J. D. Williams. **The Compleat Strategyst.** *McGraw-Hill Book Company, Inc.* 1954.

J. F. Nash. **Non-cooperative games.** In: *Annals of Math.*, 54, 286–295, 1951.

J. von Neumann and O. Morgenstern. **Theory of Games and Economic Behaviour.** *Princeton University Press*, 1947.

M.J. Osborne and A. Rubinstein. **A Course in Game Theory.** *MIT Press*, 1994.

D.R. Hofstadter, *Dilemmas for Superrational Thinkers, Leading up to a Luring Lottery.* In: **Metamagical Themas.** *Basic Books*, 1986.

F. Moller, private communication.

# 3 Game models

This section is about using (small) games as models for various situation. We will concentrate on the Prisoner's Dilemma, and variants thereof, because it has become the best studied game of all.[27]

## 3.1 The Prisoner's Dilemma revisited

When we discussed weaknesses of the notion of equilibrium point in Section 2.3 we encountered the game known as the Prisoner's Dilemma, Example 2.3. It has a pay-off matrix which looks like this:

| | | Joe | |
|---|---|---|---|
| | | defect | cooperate |
| Fred | defect | $(-8,-8)$ | $(0,-10)$ |
| | cooperate | $(-10,0)$ | $(-2,-2)$ |

We have changed the names for the actions here to comply with those most commonly found in the literature. Here 'cooperate' means 'cooperate with the other player' (by remaining silent rather than shopping him), *not* 'cooperating with the police'. Therefore 'defect' means 'defect from your partner in crime'.

We argued at the time that $(1,1)$ is the only equilibrium point of the game, leading to a pay-off of $-8$ for each player. Clearly they would both be better off if they chose their strategy 2, leading to a pay-off of $-2$ for each of them. We used this to argue that searching for equilibrium points may not always be the best answer when looking for good strategies. Nonetheless there is still one use for equilibrium points: When the game is played repeatedly, and both parties have used the equilibrium strategies in the previous round, there is an incentive of staying with it: If either side leaves the equilibrium unilaterally it will be punished by an even worse pay-off of $-10$.[28]

Even when we try to use other arguments in our endeavour of finding a 'good' strategy we are thwarted: If we look at the notion of one strategy dominating another (see Definition 9), we find that the 'defect' strategy dominates the 'cooperate' strategy: Let us look at this from Fred's point of view (although the situation is, of course, symmetric). There are two cases, either Joe will talk or he won't. If Joe talks, then if Fred talks as well he will get a pay-off of $-8$ as compared to $-10$ if he doesn't. If Joe does not talk, then if Fred talks he will get a pay-off of 0 as opposed to $-2$ if he does not. Hence the 'defect' strategy clearly outperforms the 'cooperate' strategy according to these criteria—in any given situation, it performs strictly better. And even if Fred knew in advance what Joe is going to do, his best choice would be to defect.

## 3.2 Generalizing the game

Here is a slightly different story to go with this type of situation. Assume that two people wish to exchange goods, and that for some reason this has to happen in secret, without the two people meeting each other.[29] They each promise to deposit their trade goods at a certain

[27] One of the reasons for this is that it is so useful when it comes to modelling every-day dilemmas, and another is that games with well-behaved solutions are not *that* interesting.

[28] This is, of course, what the notion of equilibrium point is all about.

[29] You might think of secret agents exchanging information, for example.

time and each at a separate place. The plan says that they then both move on to the other location and pick up what was left there. The whole situation might repeat itself a week later, in which case both parties are likely to take into account what happened in the previous week. We study the repeated game below.

We find that it does not matter which numbers occur in the pay-off matrix as long as they satisfy certain conditions. We start by assuming that the situation is *symmetric*, that is

- if both players cooperate they get the same pay-off;
- if both players defect they get the same pay-off;
- if Player 1 cooperates and Player 2 defects then Player 1 gets the same pay-off as Player 2 obtains when their roles are reversed (that is, Player 1 defects and Player 2 cooperates) and *vice versa*.

Hence rather than taking a matrix whose entries consist of pairs can use a matrix with single entries to describe the situation.[30] The following gives the pay-off for Player 1.

| | | Player 2 | |
|---|---|---|---|
| | | defect | cooperate |
| Player 1 | defect | $P$ | $T$ |
| | cooperate | $S$ | $R$ |

Symmetry means that the pay-off for Player 2 can be described by the transposed matrix

| | | Player 2 | |
|---|---|---|---|
| | | defect | cooperate |
| Player 1 | defect | $P$ | $S$ |
| | cooperate | $T$ | $R$ |

The numbers are typically read as

$P$: the **P**unishment for mutual defection;

$T$: the **T**emptation to defect on a cooperating fellow player;

$S$: the '**S**ucker's pay-off' for cooperating while the other defects;

$R$: the **R**eward for mutual cooperation.

Now if we assume that

$$T > R > P > S$$

then the 'defect' strategy strictly dominates the 'cooperate' strategy, and $(1, 1)$ is still the only equilibrium point. Further, mutual cooperation is preferable to mutual defection (let alone the sucker's pay-off). In other words, the actual numbers are irrelevant to the considerations made in the context of 'Prisoner's Dilemma-type situations' as long as they satisfy the given conditions. From now on we will assume that we have a game matrix with the four entries $P$, $T$, $S$ and $R$ satisfying these conditions. We will refer to these four letters without reiterating what they mean.

[30]But do not confuse this with a zero-sum game, the way the pay-off for Player 2 is calculated is *different* in this situation.

### 3.3 Repeated games

This section is about exploring what happens if the same game is played repeatedly. Clearly this is only of interest if the game in question is not one with an 'obvious' solution: For such a game we would expect repeated versions to behave just like single instances without any connection between the different rounds. So what if we take the Prisoner's Dilemma? Does the prospect of cooperation in future rounds make any difference?

Let us start simple and assume the game is played twice. Then each player has a choice between 'defect' and 'cooperate' for the first games, but when the second game begins they are allowed to make their decision dependent on what happened in the previous game. This game is pictured in Figure 19 where we have chosen the same pay-off matrix as in our original version of the game[31] in Example 2.3.

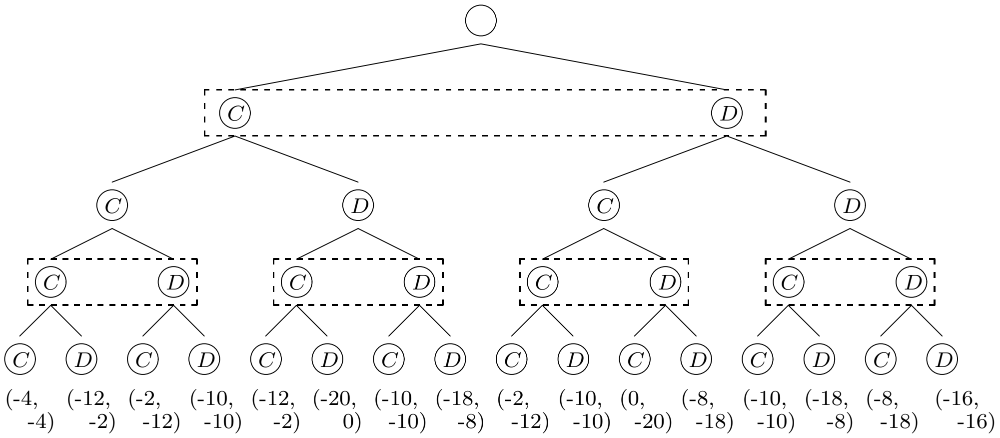



Figure 19: Two-rounds Prisoner's Dilemma

We can encode strategies as follows. Let $D$ stand for 'defect', $C$ for 'cooperate'. Then $D(C|D)$, for example, is to mean 'Defect in the first round; if the other player cooperated in the first round, cooperate in the second, otherwise defect'. We can then express the new game as a matrix game:

| | $C(C\|C)$ | $C(C\|D)$ | $C(D\|C)$ | $C(D\|D)$ | $D(C\|C)$ | $D(C\|D)$ | $D(D\|C)$ | $D(D\|D)$ |
|---|---|---|---|---|---|---|---|---|
| $C(C\|C)$ | (-4,-4) | (-4,-4) | (-12,-2) | (-12,-2) | (-12,-2) | (-12,-2) | (-20,0) | (-20,0) |
| $C(C\|D)$ | (-4,-4) | (-4,-4) | (-12,-2) | (-12,-2) | (-10,-10) | (-10,-10) | (-18,-8) | (-18,-8) |
| $C(D\|C)$ | (-2,-12) | (-2,-12) | (-10,-10) | (-10,-10) | (-12,-2) | (-12,-2) | (-20,0) | (-20,0) |
| $C(D\|D)$ | (-2,-12) | (-2,-12) | (-10,-10) | (-10,-10) | (-10,-10) | (-10,-10) | (-18,-8) | (-18,-8) |
| $D(C\|C)$ | (-2,-12) | (-10,-10) | (-2,-12) | (-10,-10) | (-10,-10) | (-18,-8) | (-10,-10) | (-18,-8) |
| $D(C\|D)$ | (-2,-12) | (-10,-10) | (-10,-10) | (-10,-10) | (-8,-18) | (-16,-16) | (-8,-18) | (-16,-16) |
| $D(D\|C)$ | (0,-20) | (-8,-18) | (0,-20) | (-8,-18) | (-10,-10) | (-18,-8) | (-10,-10) | (-18,-8) |
| $D(D\|D)$ | (0,-20) | (-8,-18) | (0,-20) | (-8,-18) | (-8,-18) | (-16,-16) | (-8,-18) | (-16,-16) |

Notice how the number of strategy increases quite rapidly, although we are only playing two rounds!

**Question 19** How does the number of strategies increase when the number of rounds played is 5, 10, 100?

[31] For this purpose it is probably a good idea to change the story, for example, make the entries in the matrix 'fines payable in hundreds of pounds' rather than 'years in prison'.

Much to our disappointment we find that the only equilibrium point is $(8, 8)$ (that is, both players choose to always defect). This leads to a pay-off of $-16$ *versus* one of $-4$ if they both cooperated. In other words, increasing the difference in pay-off by playing several rounds of the game does not make defection any less of an equilibrium point.

**Exercise 20** (a) Show that the repeated Prisoner's Dilemma game of 6 rounds has at least two equilibrium points. (Hint: Player 2 plays the same strategy for both these equilibrium points, which can be described as follows: On the first five rounds, defect. On the last round, if the other player cooperated five times, cooperate, otherwise defect again. Second hint: Player 1's two strategies which lead to an equilibrium point lead to the same play when paired with this strategy for Player 1.)

(b) Can you use your considerations in (a) to show that there are at least 4 equilibrium points in this game?

As this exercise shows, in repeated games we can get somewhat artificial equilibrium points: The 'always defect' strategy paired with itself, and some other pairs of strategies which lead to a play of mutual defection. They arise because decision points which never arise in play against another equilibrium strategy allow 'irrelevant' choices. We can tighten our definition to deal with that.

**Definition 11** *Let a a* **sub-game** *of a given game be obtained by choosing any position in the game tree and considering it as the root of a game tree, namely the one given by the part of the original tree which is below it.*

*A* **sub-game equilibrium point** *in a $j$-person game consists of a $j$-tuple of strategies, one for each player, such that this tuple gives an equilibrium point for each sub-game of the game, even for sub-games that wouldn't be reached when playing these strategies.*

The reason that this rules out strategies like the one we have just described is that if we now consider the sub-games rooted at the point where Player 2 has defected five times in a row while Player 1 has cooperated for the same number of rounds, then we're left with one round of Prisoner's Dilemma. The strategy we have just described chooses to cooperate at this point, and that is *not* the equilibrium choice.

**Proposition 3.1** *The only sub-game equilibrium point in a game of Prisoner's Dilemma with finitely many repetitions consists of choosing the 'always defect' strategy for each player.*

Here is an argument which gives a reason for this: The only difference between playing the game once or twice is that in the first round, we know that we will play again. But that means that *the second round* is just like ordinary Prisoner's Dilemma, that is, the equilibrium point is $(D, D)$. But if we know what will happen in the second, or last, round then the first round becomes like a one-off game (because the decisions the players make in the first round will not effect the outcome of the second round), so both players should defect in the first round too to reach the equilibrium point. This is something of a 'backwards induction' argument. We have ruled out all other strategies that show this behaviour against the 'always defect' strategy by asking that we get an equilibrium point for *all* sub-games (compare $(6, 6)$ in the matrix on page 62).

Clearly it does not make any difference whether we play the game once, twice, five or five hundred times: The equilibrium point will always be given by the 'always defect' strategy for both players, because we can employ the same argument, and it will be the only sub-game equilibrium point. We will study ways of ruling out this argument in Section 3.6.

So despite the fact that the *difference* in pay-off has become a lot bigger, when looking for stability (that is, balance) both players will still choose a strategy which will hurt them both. It is as if the fear of the threat available to the other person (namely to defect) is so big that, rather than risk cooperating and possibly having one's good will abused, it is more appealing to just defect oneself.

This is not how people behave in general, as has been shown in experiments. The world would certainly look a very different place if everybody followed this maxim (and presumably only the ruthless would survive). We have seen an example on page 40 of real people playing a multi-player Prisoner's Dilemma where not everybody defected.[32]

If we assume that the game is played repeatedly, then in principle it might be the case that when both players alternate defection and cooperation out of step with each other (that is, Player 1 defects whenever Player 2 cooperates and Player 1 cooperates whenever Player 2 defects) they might be better off than when both players always cooperate. In order to avoid that they can agree to 'take turns' in this way, or that they profit by reaching such a series accidentally, we assume from now on that

$$(T + S)/2 < R.$$

As we outlined above, in the repeated game there are many more strategies for both players, since in the $(n + 1)$st game they may take the history of all previous $n$ games into account if they so wish. Since there are $4^n$ different such histories that adds up to a lot of strategies!

It is easy to come up with a few strategies for a repeated Prisoner's Dilemma: One might, for example, always defect, going with the equilibrium point. (We call this strategy AlwaysD.) Or one might always cooperate, hoping the other player will do the same for the mutual benefit of both. (We call this strategy AlwaysC.) Or one could decide randomly what to do in each round. Or ...

The Prisoner's Dilemma became well-used as a model for interaction in the political and social sciences as well as in economics in the 1970s, its simplicity making it relatively easy to apply and develop. Ironically, however, there were no accounts on what it took to play well in the repeated version of the game. The theory only covered the unsatisfying equilibrium solution, but what strategies would be successful in the real world?

[32]When aiming for equilibrium points we have assumed that both players are *rational.*, where that means that they are both aiming towards maximizing their gain under the assumption that the other side has a similar target. It seems that in practice, people are not as rational as that, at least in games where the number of repetition is fairly high—they only tend to start defecting in the last few rounds, if ever.

### 3.4 A computer tournament

In 1979 the political scientist Robert Axelrod decided he wanted to know more about successful strategies for playing repeated Prisoner's Dilemma type games *in practice*. His interest in this particular game originated from the question how, in a world where being selfish seems to give individuals advantages, cooperation could ever have developed. He had the obvious(?) idea that it would be helpful if he had lots of people thinking about this, and cleverly attracted them into taking part in a computer tournament. The following pay-off matrix was used:

| | defect | cooperate |
|---|---|---|
| defect | 1 | 5 |
| cooperate | 0 | 3 |

He invited game theorists from a number of different academic fields such as economics, psychology, political science, mathematics and sociology and invited them to send him a computer program to play repeated Prisoner's Dilemma in a round robin tournament. At the same time he gave them the results of a preliminary tournament played with fairly simple strategies. In the proposed contest each program would play each of the other programs, as well as a clone of itself, for 200 rounds. Whichever program had the most points at the end would win. In order to rule out pseudo-effects caused by statistical fluctuations[33], he ran the entire tournament five times.

There was no limitation regarding what people could try. They were allowed random choices, and they were allowed to remember the entire history of the bout against the same fellow player so far (but not the history of another run of the tournament, or the history of play against another player). None of the strategies were penalized for being slow.

He had fourteen submissions of programs from 4 to 77 lines (of Basic). He himself introduced another entry, the strategy which plays totally randomly, by 'tossing a coin'[34] before each game. It is referred to as RANDOM.

Note that here strategies are *competing with each other for points*, which is a different point of view from the one employed when considering equilibria. Now it is important how a strategy does when compared to others, whereas before we only worried about what would happen if one player unilaterally moved away from an equilibrium point.

**Question 20** What would you have submitted? Do you think your strategy might have won?

It is clear that given the conditions imposed by Axelrod there is no 'best' strategy, that is one that will win such a tournament against all comers. Instead which strategy is best depends on the *population*, that is the other strategies present and their numbers.[35]

To everybody's surprise, the shortest program won. It had been submitted by the psychologist and philosopher Anatol Rapoport. It looks quite simple:

[33] The players, that is the programmers, were allowed to make random moves, hence the need for doing this.
[34] Well, the computing equivalent of that.
[35] People criticized Axelrod's work using the argument that he had not paid sufficient attention to the effect of the initial population, let alone conducted a systematic analysis of same. But Axelrod pioneered this approach which has become hugely influential, opening up new vistas. It seems natural that later investigations should be required to develop this idea to its full potential.

- On the first move, cooperate;
- then do whatever the other player did last.

This strategy is known as TitForTat, because it cooperates until the other player defects, and then it retaliates by defecting in the next round. What happens subsequently depends on the other player: If the other player goes back to cooperating, so does TitForTat. It is a very simple idea, but actually more subtle than you might think at first. TitForTat scored 504 points on average over the 200 rounds it played against the other strategies. If over 200 games both strategies always cooperate, they can each get 600 points. If both strategies always defect they each get 200 points. What might come as a surprise, however, is that TitForTat will *never* outscore a single one of its opponents in their bout of 200 games: if both strategies keep cooperating, they will both score 600 points. If the other strategy defects first then the best TitForTat can hope for is eventually to make up the disadvantage of five points caused thus! After all, it will only defect in reply to another defection. If it is fortunate, then the other side will cooperate, and it will recoup this loss. If the other side defects as well, they both get the same low pay-off.

Let us look at some of the strategies which did worse than TitForTat. Joss (named after its creator), for example: It behaves just the same as TitForTat but every so often it throws in a random defection and afterwards it goes back to cooperating. When TitForTat and Joss play against each other the following effect occurs: They both cooperate, racking up points, until Joss throws in one of its random defections. In the move after, it goes back to cooperating, but TitForTat will now copy the defection. In turn, Joss defects in the following round when TitForTat has gone back to cooperating. In other words, the two programs are tied together in lockstep, and the payoff they get for each set of two games is 5, whereas it would be 6 if they both cooperated (see the table below). But, worse can happen when Joss decides that it should throw in another random defection. Here is an example of such a match.

| TitForTat | Joss |
|---|---|
| $C$ | $C$ |
| $\vdots$ | $\vdots$ |
| $C$ | $C$ |
| $C$ | $D$ |
| $D$ | $C$ |
| $C$ | $D$ |
| $D$ | $C$ |
| $\vdots$ | $\vdots$ |
| $C$ | $D$ |
| $D$ | $D$ |
| $D$ | $D$ |
| $D$ | $D$ |
| ... | ... |

| TitForTat | Joss |
|---|---|
| 3 | 3 |
| $\vdots$ | $\vdots$ |
| 3 | 3 |
| 0 | 5 |
| 5 | 0 |
| 0 | 5 |
| 5 | 0 |
| $\vdots$ | $\vdots$ |
| 0 | 5 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| ... | ... |

At this point, both programs will defect forever, and their pay-off goes down to just one point per game. These two strategies did fairly poorly against each other. Note that, because of that second random defection, JOSS (at 230 points on average when paired with TITFORTAT) actually outscores TITFORTAT (at 225 points). Their respective final positions in the league table after the tournament therefore has to be explained by looking at how they did against other programs. It turns out that play as outlined above occurred not infrequently between JOSS and its opponents, so that ultimately a strategy like JOSS punishes itself.

We might reveal here that, in fact, all the participants in the tournament knew that TITFORTAT is a useful strategy for this game (it won the preliminary tournament they all were informed about), and therefore many used variants of it. What is surprising, however, is that none of these sophisticated constructs (which were supposedly improvements) did as well as the original!

Another fairly sophisticated submission is known as DOWNING, after its proponent. The psychologists Robert Downing was also the person to submit it as a tournament entry. It attempts to find out, in the course of play, how its playing partner reacts to defections, and then defects as often as it thinks it can get away with. At the start, DOWNING assumes that the probability for the other player to cooperate after itself has defected is 1/2, and the same for the probability that the other player will cooperate after itself has cooperated. These two probabilities are then adjusted in the course of play, and used to determine DOWNING's own behaviour: It tries to play such that the long term gain will be maximized, under the assumption that it is modelling the other player correctly. If the two probabilities are similar then DOWNING will tend to defect, having decided that it makes no difference to the other side's behaviour. If the other player instead tends to retaliate after a defection, but cooperates when DOWNING cooperates, it will decide that it is more prudent to cooperate. Due to the way the two probabilities are set at the start (when DOWNING has no real information about its playing partner) it typically starts with defecting twice. In the tournament that led to other programs punishing it accordingly. As a consequence the program only came 10th in the tournament, gaining only 390 points on average against other entries.

Not unsurprisingly RANDOM came last, so at least all the programs, into which much thought must have gone, were better than making random choices (but not all outperformed it by much!). A strategy called GRUDGE, which cooperates until the other side defects and then defects forever, came 7th.[36]

After the tournament had finished Axelrod spent some time analysing the results. First of all he noticed the following:

- All of the eight top scoring entries had one property in common: They were never the first to defect, with the possible exception of the last few rounds. Axelrod called such strategies **nice**.[37]

Interestingly there was a fairly big gap between these eight and the next seven: The top eight programs all scored more than 470 points on average, whereas the next best strategy only just managed to top 400 points. Notice that when two nice programs meet each other then they will score a 'perfect' 600![38]

[36] This strategy is also known as TRIGGER or GRIM in the literature.

[37] This does not, of course, mean that nice strategies never defect.

[38] Strictly speaking, a perfect result would be the 1000 points to be won over 200 games by the ALWAYSD strategy against ALWAYSC, but that does not seem very realistic.

A number of entries tried to get away with the occasional unprovoked defection, just like JOSS, presumably hoping for forgiving playing partners. But what they failed to consider was that this kind of behaviour might have considerable repercussions over the course of play, see the table above—because a defection started by *them* might be mirrored by the other player in the next round, and then that would provoke their own program to defect again. They did not appreciate this secondary (or maybe even tertiary) 'echo'-effect. So the other strategies present turned out to be less forgiving than expected, and the programs were typically constructed such that the 'punishment' would last for much longer than anticipated.

- Strategies that did well were **forgiving** in that they did not hold a grudge once they had retaliated after a defection by the other side.

Axelrod discovered that other strategies, had they been present at the tournament, would have won. One example of this is TITFORTWOTATS, a strategy that will cooperate unless the other player has defected twice in a row in which case it will defect (but just once). Ironically, this strategy was given to all contestants as a sample program which they simply could have copied! The good performance of this strategy against the assembled field seems to indicate that the entries in the tournament were too keen on retaliating, where being slightly more forgiving might have been advantageous. Another winner, had it been present, would have been a revised DOWNING which starts with cooperation rather than defection. We call the resulting strategy RDOWNING.

All in all it seems surprising that the assembled brain power could not find anything better than the deceptively simple TITFORTAT. The bad performance of some of the submitted program suggests that quite a few of the contestants totally misread the situation in which their program would likely find itself. They were trying to get away with defections far too early in the game without anticipating the fallout, and they typically were not forgiving enough, resulting in long runs of mutual defection (or at least the pairing of a cooperation with a defection), compare the example of JOSS against TITFORTAT given above. A random cooperation might have moved them out of that sort of rut.

### 3.5 A second computer tournament

Armed with these results Axelrod decided that he wanted to take it another level up: By holding a second tournament where all the contestants would know about the outcome of the first one he hoped he would get a new generation of strategies which would be more sophisticated and more successful than the first.

This time Axelrod did not invite all the participants personally; while he did so for a number of people who had done some work on games, including all the participants of the first tournament, he advertised in a number of computer magazines to attract people interested in programming. He hoped that this last group would be interested enough to spend a lot of time on fine-tuning their strategies. To all who entered he sent a detailed and thorough analysis of the first event, including his thoughts on 'nice' and 'forgiving' strategies. He also included a description of the strategies that 'would have won'. The rules were exactly as they were for that first tournament.

The second tournament was a somewhat bigger affair, with 62 entrants from six countries. TITFORTAT was once again submitted by Rapoport[39], TITFORTWOTATS was an entry by

[39]Maybe surprisingly he was the only one to do so!

the British biologist John Maynard Smith[40], and two contestants decided to offer RDowning. Once again, Axelrod added Random to the mix.

**Question 21** What kind of program would you have submitted to the second tournament, knowing what you know now?

It took a few hours to calculate all the results. When he had them, Axelrod was stunned: TitForTat had won again![41]. Despite the fact that all the contestants had known how well it might do, none of them could come up with something that was better. The strategies which would have done so well in the first event, TitForTwoTats and RDowning, ended up in 24th and 38th/40th position. It seems that TitForTat's main advantage is that it gets on well with a large variety of strategies, while more sophisticated programs could not match that.

Axelrod looked at the results also to see how well his observations from the first round worked.

- **Niceness.** All but one of the top fifteen entries were nice (and the sole one which wasn't came eighth). All but one of the bottom fifteen entries were not nice. In fact, almost two-thirds of the entries were nice, so many people had taken this lesson from the first round on board. However, a good many people apparently could not conceive of the fact that offering cooperation might work better than trying to get away with defection in an attempt to exploit weaknesses wherever possible, and none of them did very well.
- **Forgiveness.** Almost all the entries were somewhat forgiving, so this was no longer a useful criterion to distinguish between strategies.

Axelrod came up with another criterion to separate the sheep from the goats. In particular among the top fifteen nice strategies their success seemed to rely mostly on one new condition.

- In order not to be exploited as *too* nice and forgiving, a strategy has to hit back if the other side defects 'without reason'. Axelrod called such strategies **retaliatory**. There are two parameters that go into this criterion: How *promptly* a strategy hits back and how *reliably* it does so. However, bearing grudges is not a good property to have—programs which punished defections for too long did poorly overall, due to their violating the forgiveness condition.

[40]He played a big role in the founding of Evolutionary Game Theory, a subject we discuss below.

[41]It might surprise you to note that nobody had the idea to submit a TitForTat strategy which defects on the very last round—since the number of rounds was known to all contestants in advance, this would have been a perfectly sensible change to apply

Axelrod identified two strategies which seem to test just how retaliatory and forgiving another strategy is, and then try to maximize their pay-offs under assumptions derived from these observations. The first one, which he named TESTER, is looking for softies to exploit. It defects on the first move to see how the other side reacts. If the other strategy retaliates, it 'apologizes' by cooperating in the second game and playing as TITFORTAT from then on. If the other side does not respond promptly, it cooperates on the second and third move, but then defects every other round. It did very well against strategies which forgave too easily, or needed more provocation to retaliate, such as TITFORTWOTATS. It also exploited strategies similar to DOWNING: Because TESTER cooperates just over half of the time, these strategies calculate that it pays off to keep cooperating—and are exploited for that. However there were not enough suckers for it to do well in the overall tournament: It only came 46th.

Another strategy looking for possibilities to exploit others is one that Axelrod dubbed 'Tranquilizer', short TRANQ. It works by first establishing a relation of mutual cooperation (assuming the other player allows that), and then cautiously tests whether it can get away with some defections. It is careful, however, never to defect twice in a row, nor more than at a quarter of all moves—it does not push its luck. It also defects in reply to the other side defecting. Again, this fairly sophisticated program, based on reasonable sounding arguments, did poorly overall—it only came twenty-seventh (so it at least did considerably better than TESTER).

In order to do well against programs like TESTER and TRANQ, a program must retaliate as soon as the other side defects without provocation and so try to convince the other player that it will not tolerate this sort of behaviour. In other words, it has to be *retaliatory*, the property identified above. TESTER and TRANQ, in turn, were not nice, and paid for that by doing poorly, mostly due to a breakdown in trust that followed from their trying to get away with defections. With all their sophistication in sniffing out strategies which were too forgiving they only improved on TITFORTAT's score against about a third of the field, doing worse than TITFORTAT against the remaining two thirds.

TITFORTAT has all three properties identified as advantageous: It is nice (it never defects first), forgiving (it is prepared to try cooperation again if the other side shows willingness) and retaliatory (it will copy defections made by the other side in the next round).

The poor result of strategies such as RDOWNING and TITFORTWOTATS shows just how important the *environment* is to the performance of a strategy. As we have argued before, there is no unique best strategy for all circumstances; it depends on the overall population what will do well. If nothing else TITFORTAT seems to be very successful in an environment where most others are forgiving, and a good many are nice.

Another surprising lesson is that it seems remarkably difficult to improve on TITFORTAT: A number of the strategies in the second tournament were variations of TITFORTAT designed, for example, to give up on cooperating with RANDOM and other very uncooperative playing partners.

In analysing the results of the second tournament further, Axelrod discovered that he could divide the strategies present into one of five categories. Moreover, there were 'typical' representatives of these categories present in the following sense. Let the representatives of the five categories be $S_1$, $S_2$, …, $S_5$ and let $P(T, S_i)$ be the pay-off of the strategy $T$ playing against the strategy $S_i$. Now consider the formula

$$120 + .202P(T, S_1) + .198P(T, S_2) + .11P(T, S_3) + .072P(T, S_4) + .086P(T, S_5).$$

This can be used to predict the score of a strategy $T$ in the tournament to an amazing degree of accuracy (typically a difference of fewer than 10 points out of about three to four hundred). The strategy $S_4$ is TESTER while $S_5$ is TRANQ. The strategies $S_1$, $S_2$ and $S_3$ are nice. Among these, $S_2$, called REVISEDSTATETRANSITION by Axelrod, keeps a model of the other player as a one-step Markov process.[42] Given the history of the game so far, it estimates the probability that the other player will defect based on the result of the previous round. When the other strategy appears to defect randomly, REVISEDSTATETRANSITION quickly decides that it does not pay to cooperate after the other side defected, and sometimes even decides that its best course of action is to defect always. Again this proved fatal for some programs—they did not score highly when paired with REVISEDSTATETRANSITION.

The discovery of these categories gave Axelrod the idea that he could use the large $63 \times 63$ matrix of outcomes of this tournament to run hypothetical tournaments, by using weights to simulate a certain percentage of certain strategies being present. It turns out that TITFORTAT would have won five out of the six tournaments he thus constructed, indicating that TITFORTAT is what he calls **robust**: It does well in a large variety of environments. The reason it does so is that it offers a behaviour which allows it as well as its playing partner to do well, provided that the latter is prepared to cooperate. While TITFORTAT is friendly in the sense of trying to elicit cooperation, it is impossible to exploit, since it is retaliatory. Hence it pays to cooperate with it, and that in turn works for TITFORTAT. It is also a strategy that is easy to recognize. Since it never outscores another strategy (see above for the reasons) this shows that in order to do well in a non-zero sum world one does *not* have to do better than the others on an individual basis.[43]

It should also be pointed out that the equilibrium strategy, ALWAYSD would have done very poorly in either tournament with a pay-off of not much above 200 against most other strategies. Strategies which cannot adapt to their playing partner do poorly for the most part, because the other side eventually gives up trying to elicit cooperation. In the second tournament, RANDOM was the only such strategy, but the others which came near the bottom were so inscrutable that they *appeared* to be unresponsive to their competitors.

In another tournament, held by D.R Hofstadter and Marek Lugowski, three ALWAYSD strategies submitted came in at the very bottom out of 53 competitors, followed closely by a few versions of RANDOM. However, in this tournament three variations on TITFORTAT did better than the original—all these switched to ALWAYSD if the other player was too uncooperative. Ultimately it remains true that the environment decides over which strategy will do best.

Meanwhile a competition has been held to celebrate the 20th anniversary of Axelrod's original, organized by somebody at the University of Nottingham in 2004.[44] There were 223 entries this time, and another competition was planned for 2005. A group from the University of Southampton submitted 60 programs which were designed to recognize each other, at which point one of them would turn into ALWAYSD while the other behaved like ALWAYSC. On meeting a program outside of the group they would behave like ALWAYSD to spoil the other's

[42] This is a mathematical description of a system which, given its current state, will move to other states with certain probabilities.

[43] This seems to be a lesson we find difficult to learn. Axelrod reports that when he divides his students into pairs and lets them play a number of rounds of Prisoner's Dilemma against each other with the aim of maximizing their overall score, they will typically lose sight of this target and concentrate on the immediately obvious comparison with their partner's score.

[44] See the literature list for this section for a url.

result. The top three entries of the tournament were submitted by Southampton, but a lot of the bottom entries were theirs too. The organizer hadn't expected this kind of thing to happen, but decided that it was impossible to rule out collusion between different entrants. The Southampton team was interested in the question of how many colluders it would take to win the tournament, and they think now that they had rather more than would have been necessary. They think they would have won with 20 entries. The head of the team speculated that they would be able to win a territorial evolutionary tournament with this idea as well, but no evidence has been provided for that that I know of.

But how realistic is it to assume that the environment is *constant*? After all, we wish to use this kind of game to explain why people behave as they do when faced with certain decisions. Isn't it realistic to assume that unsuccessful strategies will vanish, and the more successful ones multiply? If the players are intelligent beings they might *learn* from their mistakes and switch to better strategies. Or they might observe others doing better and *imitate* their behaviour. Or a manager of a business might be removed from his job because of poor results, in the hope that the successor (often *selected* based on previous success) will do better. The three mechanisms of learning, imitation and selection all might be at work and result in unsuccessful strategies vanishing while others proliferate. We will meet these thoughts again in the section on *Games and Evolution*.

### 3.6 Infinitely and indefinitely repeated versions

Something that Axelrod did not consider in the two tournaments he conducted was the question of whether, when modelling real life situations, it really makes sense to assume that proponents of different strategies will meet each other a fixed number of times. This knowledge allows the 'backwards induction' argument from page 63 which leads to unconditional defection being the only sub-game equilibrium point, which seems not very realistic for most real-world situations.[45]

One possibility is to look at **infinite** repetitions of the game. That would keep the 'last game' below the horizon of the players forever, and thus this kind of induction argument could never get started. This carries with it its own set of problems.

- **Realism.** Is it realistic to use a model like this? After all, nobody will ever play an infinite number of Prisoner's Dilemma games. This can be countered by considering that it is not really necessary to make the number of games infinite; it is sufficient if this number is large enough. For example it is safe to assume that no person will live more than 1000 years, and that they could not play the game more often than once per second. Hence a safe assumption would be that $10^{12}$ is a safe upper bound for the number of games a living being could play. That gives a finite upper bound for the number of games that could occur.[46] But this moves us into the realms of philosophy.

[45] Certainly it seems to be the case that in *practice*, when it comes to the repeated game people do not choose the 'always defect' strategy.

[46] Some people claim that this would still make the backwards-induction argument possible, but I don't agree. What do you think?

- **Calculating pay-offs.** Clearly in an infinite game of repeated Prisoner's Dilemma, no matter what decisions each side takes, the pay-off will be infinite as long as all entries in the pay-off matrix are positive numbers. This problem can be solved by limiting ourselves to strategies which have finite descriptions, because then it can be shown that eventually, play will enter into cycles that will be repeated forever. By counting the length of the cycle and the pay-off for a player we can calculate an *average payoff* for each player for each game that belongs to the cycle. That gives a good enough measure to make comparisons.

A more popular model is instead to make the number of repetitions *indefinite*. Rather than assuming that after every game there will be another until a certain number has been reached, we do the following. After each game, conduct a chance experiment, for example by creating a random number between 0 and 1. If that value is below some threshold value, say $w$, then play another round, otherwise stop. That means that the expected pay-off for the AlwaysC strategy played against itself is

$$R + wR + w^2R + w^3R + \cdots = \frac{R}{1-w}.$$

The closer $w$ is to 1 the higher the probability that there will be another game, and the higher the difference between the expected pay-off from mutual defection *versus* that from mutual cooperation. The higher $w$ the higher the potential worth of mutual cooperation. People sometimes call this 'the shadow of the future' to capture the idea that current behaviour is likely to influence future gains.

There is another way of reading this, which seems to be preferred by economists: Rather than read $w$ as a probability that there will be another game they tend to view it as a *discount factor*, the idea being that a pay-off in the future is worth less than one now.

If $w$ is very close to 0 then the situation becomes similar to a one-shot Prisoner's Dilemma. Hence this one parameter allows a variety of situations to be captured.

We discuss this version of the repeated Prisoner's Dilemma in Section 4. Let us note here, however, the following result.

**Proposition 3.2** *For each of the following properties there is at least one sub-game equilibrium point for the indefinitely repeated Prisoner's Dilemma game satisfying it. In the play resulting from employing these strategies against each other it is the case that*

- *they both cooperate throughout;*
- *they both defect throughout;*
- *they both defect in the first round and cooperate ever thereafter;*
- *they both cooperate in every odd round and defect in every even one;*
- *Player 1 cooperates in even rounds and defects in odd rounds while Player 2 defects in even rounds and cooperates in odd ones.*

**Proof.** A proof of this is beyond our scope. □

This shows that indefinitely repeated Prisoner's Dilemma has a huge number of sub-game equilibrium points, which means that it is very hard to predict just what will happen in this game. Also note that TitForTat is *not* a sub-game equilibrium point, but it is an equilibrium point in this game!

### 3.7 Prisoner's Dilemma-type situations in real life

We have already seen examples of Prisoner's Dilemma-type situations in the various sections above, but we will look at a few more here.

We are typically talking about situations with two or more players[47] where each individual has the choice to cooperate or to defect. Mutual cooperation is advantageous, but people are yet better off if they defect while others cooperate. Arguably everybody would be better off if all people paid their taxes, because then no money would have to be spent on dealing with fraud. Yet the individual has much to gain from getting away with cheating.[48] Insurance fraud is another example—everybody's premium has to go up to cover for that, and the honest participants have to pay for the defectors. On motorways, some people will try to pass a queue of cars at an exit to cut into it late on so that they won't have to wait quite as long. Airlines try to seat their passengers by seat row to make it more comfortable for everybody. Yet many people defect by refusing to wait their turn (and as a reward they get to fill up the lockers with their luggage). The world would be a much nicer place if pollution were to be reduced, but for each country it is much cheaper to let the *others* work at decreasing their emissions. Or the inhabitants of a lakeside would prefer if the water wasn't polluted, and maybe an individual dump of rubbish doesn't do much to the water quality, but if everybody does it the consequences are noticeable.

With the exception of the last two examples it is the case that the individuals engaged in unsocial behaviour do not hurt one (or several) other individuals who might recognize them and then retaliate. There is not much real pressure (beyond the force of the law) to behave. (Thankfully the vast majority of the population does not try to defect.) Axelrod's tournament, on the other hand, made sure that individuals played with each other on a one-on-one basis, and that there were sufficiently many rounds to reward or punish the other side's behaviour. So repetition and recognition are important in order for cooperation to develop.

Villages tend to be friendlier places than cities, people typically leave higher tips in their local restaurants, business partners with a long mutual history do not try to squeeze the last bit out of their contract; instead they try to make sure that their part of the cooperation works smoothly. It is generally accepted among social scientists that the reason for this behaviour is precisely the fact that future encounters are very likely, and that cooperation will therefore pay off in the long run.

Finally here is a concrete example for cooperation arising in a repeated Prisoner's Dilemma situation. In the trenches in World War I, a phenomenon called 'live-and-let-live' occurred, where both sides refused to take shots at individuals on the other side of the line. Typically the same small units faced each other across one to four hundred yards of no man's land for months at a time. Clearly when members of the other side exposed themselves, there was at least short term gain to be made from shooting (and potentially killing) them because that meant they would not be available for the enemy's next attack. This is the temptation pay-off. Such short-term cease-fires may have started when both sides received their rations each night after dark and were not inclined to engage in any other activity at that particular time (including warfare). The enemy waslikely to retaliate if one disturbed his evening meal,

[47]Some people have discussed one-person versions of the game, but I do not find the results very convincing.

[48]And, ironically, most people seem to think that they only hurt some evil entity known as the Inland Revenue.

threatening punishment, whereas the reward for mutual cooperation was that both sides could eat in relative peace. Other situations where this might have occurred were bouts of bad weather, where no major offensive action could take place. Headquarters frowned upon such behaviour, but throughout the course of the war were unable to eliminate it. The most famous example is probably given by various Christmas celebrations, where again both sides were disinclined to fire, making life much easier for each other by doing so. Sometimes this 'cooperation' between enemy troop carried through to the actual fighting part, when raids (ordered by headquarters) were made without hurting any enemies, or artillery fire was directed in such a predictable way that the other side could evacuate the endangered spots. Typically, this behaviour was reciprocated. We note that both, repetition and recognition (due to the fact that the same small units faced each other over long periods of time) were present in this example. We claim that they are vital, for cooperation to evolve.

### Summary of Section 3

- The *generalized Prisoner's Dilemma* has four pay-offs, $T > R > P > S$ with $R > (S+T)/2$. Its only equilibrium point is given by the strategy which always defects (for both players), and this is also the dominant strategy.

- If the game is *repeated* a fixed number of times, then the strategy which always defects still gives an equilibrium point, but no longer the only one. It is, however, the only *sub-game equilibrium point*, that is it gives an equilibrium point in every sub-game.

- In two computer tournaments playing this game, the strategy known as *TitForTat* won twice. Successful strategies had properties known as *niceness*, *forgiveness* and they were *retaliatory*.

- More realistic versions of this game are *indefinitely repeated*, where after each round a chance experiments determines whether a further round will be played. This will occur with a fixed probability.

### Sources for this Section

As indicated in the text there is now a proliferation of articles concerning themselves with Prisoner's Dilemma type situations. I have mostly used secondary sources.

R. Axelrod. **The Evolution of Cooperation.** *Basic Books, Inc.* 1984.

D.R. Hofstadter. *The Prisoner's Dilemma and the Evolution of Cooperation.* In: **Metamagical Themas.** *Basic Books*, 1986.

The entry on the Prisoner's Dilemma in the **Stanford Encyclopaedia of Philosophy**, (mirrored) at `http://www.seop.leeds.ac.uk/entries/prisoner-dilemma/`.

On the 20th anniversary '*The Iterated Prisoner's Dilemma Competition* there's comprehensive coverage at `http://www.prisoners-dilemma.com` with further links.

B. Brembs. *Chaos, cheating and Cooperation: potential solutions in the Prisoner's Dilemma*, in: **Oikos** 76, pp. 14–24 or at `http://www.brembs.net/papers/ipd.pdf`.

M.J. Osborne and A. Rubinstein. **A Course in Game Theory.** *MIT Press*, 1994.

# 4 Games and evolution

We have indicated above that scenarios such as those of the Prisoner's Dilemma type are not exhausted by looking at how a population of strategies does against each other. The next logical step is to assume that success will breed success, and that strategies which do not perform well will vanish eventually. In this section we consider models for this kind of situation.

As we have seen the space of all strategies in the indefinite Prisoner's Dilemma is huge. So far it has not been fully explored. Even if when limiting one's considerations to strategies which are nice, retaliatory and forgiving, there is much which currently is not known. How many strategies satisfy these properties? Just how successful are they?

If the probability for playing another round is sufficiently large then cooperating pays off when compared with defecting, and we do know that two TitForTat strategies form a Nash equilibrium pair for the indefinitely repeated Prisoner's Dilemma game, but then, so do two AlwaysD strategies. In fact there are a lot of Nash equilibria for this game. Notice that TitForTat does *not* form a sub-game equilibrium when paired with itself, and so is not a privileged solution.

Despite the fact that so much is unknown about this game we will take it as our basis for looking at evolution. It is worth pointing out here that games such as the ones introduced in this and the previous section are used to model computational systems, for example multi-agent ones. However, looking at examples of this in any detail is beyond our scope.

## 4.1 An ecological tournament

Assume there is a population of animals which meet each other from time to time, and at each such meeting can decide either to cooperate with the other or to refuse that. If such an animal can recognize individuals it has already interacted with then a series of meetings between the same individuals form an indefinitely repeated Prisoner's Dilemma-type game. Then a tournament between a number of such individuals (where each strategy is likely to be employed by a number of entities) can simulate one generation of these animals. In the next round, their success in the previous one should influence how many animals support each strategy. We can view these individuals taking part in the next tournament (at least in an abstract way) as the offspring of those present in the previous one. If we start with Axelrod's second tournament, but assume that each strategy is supported by a number of individuals rather than being just one entry, then in the next generation the number of individuals using Random is likely to be lower, whereas there presumably will be more individuals employing TitForTat. For this scenario to be sensible it is not that important which mechanism leads to this change in numbers: Be it that the individuals learn from past mistakes, that they imitate other, more successful individuals, be it that there is some sort of pre-selection before a player is allowed into the game, or be it that there is biological selection at work where successful individuals are likely to be able to bring up more offspring than those who are struggling to survive.

To put this into a formula one might make the number of representatives of each strategy in the next round proportional to the strategy's average score in the previous round(s) (although if biological selection is employed, the number of representatives in the previous round should probably also contribute to this). This is a mechanism which ensures the survival of the fittest, that is, the most successful. Using the scores given by the second tournament as a measure for how one strategy would score when paired with another, Axelrod ran such a simulation. He observed that what happened first of all was that the lowest ranking eleven entries were down to half their initial number by the fifth generation, while the mid-ranking ones roughly held their own and the top-ranking ones started to grow in number. By the fiftieth generation the bottom third of the strategies had all but disappeared and most in the middle third had started to shrink in number, while those in the top third continued to grow. We present a sample version of such a tournament in a slightly different environment in Section 4.4.

Note that at each stage, the population changes, resulting in an environment that fluctuates from round to round. In order to be successful a strategy therefore has to perform well when paired with other successful strategies. A good example of a strategy which does not manage to do this is the only non-nice strategy among the top third (it came eighth) in the second tournament, dubbed HARRINGTON after its creator. It is a variant of TITFORTAT. It checks whether it is matched against the RANDOM strategy, and it has a way of getting out of alternating defections (compare the problems encountered by the strategy JOSS, page 66). It always defects on the 37th move, and thereafter makes random defections with increasing probability if the other strategy does not retaliate. In the first 200 or so generations it did well, together with the other successful strategies in the tournament such as TITFORTAT. Its problems only began at that point: It relied on there being strategies which it could exploit, but the exploitable strategies had all but vanished by then. That was when HARRINGTON could no longer keep up with the other successful strategies, all of which were nice. By the thousandth generation it had become almost extinct. Arguably it had become a victim of its own success, having eradicated all its potential victims. This 'ecological' tournament was also won by TITFORTAT—at the end of the game it had a fifteen percent share of the entire population, thus having increased its numbers fifteen-fold.

## 4.2 Invaders and collective stability

Intrigued by the results of the ecological tournament Axelrod became interested in exploring evolution using this technique. How would a population consisting entirely of TITFORTAT strategies fare? Could such a world be invaded by some rival? And, most importantly, if we assume that originally individuals tend to be selfish (in particular when that leads to the obvious immediate gain of the 'defect' strategy in a Prisoner's Dilemma-type game), how can cooperation ever evolve? After all, there could not be an institution imposing good behaviour on everybody (like today's states do to some extent). What would tempt individuals away from the equilibrium point strategy, in particular when seeing that they do at least as well as any rivals by using it?

From now on we will assume that the game in question is the *indefinitely repeated Prisoner's Dilemma* as described on page 72. We assume an underlying matrix with pay-offs $R$, $T$, $P$ and $S$ as discussed on page 61.

### No simple answers

Our first result for that game should come as no surprise, and we have been assuming that implicitly in Section 3.

**Proposition 4.1** *In an indefinitely repeated game of Prisoner's Dilemma with two players there is no one strategy that is the best response to all other strategies, provided that $w$ is large enough.*

**Proof.** Assume there was a best strategy and that $w > 0$. We first assume that that strategy cooperates on the first move. If this strategy is played against the AlwaysD strategy, then the best it can do after cooperating on the first move is to defect forever, since that means choosing the pay-off $P$ over $S$ and since we know that $P > S$. But then the best pay-off it can get against AlwaysD is

$$S + wP + w^2P + \cdots = S + \frac{wP}{1-w}.$$

Our strategy is outperformed by AlwaysD, which when playing against itself can manage the higher pay-off of

$$P + wP + w^2P + \cdots = P + \frac{wP}{1-w}.$$

Since we have found a better strategy for this situation, the one we started with can't have been the best overall.

Now assume that our best strategy defects on the first move. But that means that when playing against the Grudge strategy it will get a pay-off of $T$ for the first move, but forever thereafter, Grudge will defect. Hence our strategy (assuming it will defect from the second move onwards when playing against Grudge since that is the best it can do under the circumstances) will get pay-off at most

$$T + wP + w^2P + \cdots = T + \frac{wP}{1-w}.$$

The AlwaysC strategy (or indeed Grudge itself), on the other hand, when playing against Grudge can expect a pay-off of

$$R + wR + w^2R + \cdots = R + \frac{wR}{1-w}.$$

Now the latter is bigger than the former provided that

$$R + \frac{wR}{1-w} > T + \frac{wP}{1-w},$$

that is when

$$R > T(1-w) + wP$$

which is the case if and only if

$$w > \frac{T-R}{T-P}.$$

Hence if $w$ is larger than this threshold value then we have once again found a strategy which performs better (against Grudge) than our best strategy. Therefore such a best strategy cannot exist. □

### Collectively stable strategies

One of the original approaches to this kind of question comes from John Maynard Smith, which proposes that in a population of individuals all of which employ the same strategy, a single mutant might occur following some other decision procedure.[49] Clearly such a mutant

[49] It is generally accepted in biology that mutations play an important role in evolution by introducing some variety which did not previously exist. Successful mutants eventually increase in numbers.

would be quite successful if its strategy outperformed the 'native' one, that is, if it scored better against that native strategy than that strategy scored against itself. In other words, an individual using strategy $B$ can invade a population consisting entirely of individuals using strategy $A$ if and only if the expected pay-off of $B$ playing against $A$ is larger than that of $A$ playing against itself.[50] It is important to note here that we assume that *every individual competes with every other individual.* In Section 4.4 we look at systems where players only play against (and learn from) their neighbours.

**Definition 12** *Let $P(A, B)$ be the pay-off that a strategy $A$ receives when playing indefinitely repeated Prisoner's Dilemma against a strategy $B$. We say that a strategy $B$ can* **invade** *a native strategy $A$ if it gets a higher pay-off when playing against $A$ then $A$ gets when playing against itself, that is if*

$$P(B, A) > P(A, A).$$

*We say that a strategy is* **collectively stable** *if it cannot be invaded by any other strategy.*[51]

If we assume that mutations do occur, and that they are all equally likely it seems safe to assume that in order for a population employing one single strategy to maintain itself indefinitely this one strategy has to be collectively stable, because that guarantees that such a population will be able to ward off all invaders (or 'newcomers'). But the notion of being collectively stable is really independent from just how an invasion (that is, change) might occur as long as it is safe to assume that attempted invasions will be carried out by individuals rather than groups. We turn to the more complex question below.

We start our investigation by considering a population consisting entirely of AlwaysD strategies. After all, this strategy gives the sole sub-game equilibrium point in the ordinary repeated game.

**Proposition 4.2** *The* AlwaysD *strategy is collectively stable for all $w$.*

**Exercise 21** (a) Prove Proposition 4.2.

(b) Give circumstances (pay-offs $R$, $T$, $S$ and $P$ as well as $w$) under which the Grudge strategy is not collectively stable.

We next consider a population consisting solely of TitForTat strategies to see whether we can explain the success of this strategy in the tournament. What kind of strategy might be capable of invading such a group? Clearly in order to score better against TitForTat than TitForTat scores against itself the strategy in question has to defect at some point (or the entire series of interactions will consist entirely of mutual cooperation, which is the same outcome as TitForTat playing against itself). For defecting while TitForTat is still cooperating, the invader will get pay-off $T$ *versus* pay-off $S$ scored by TitForTat on that move. On the next move, TitForTat will retaliate by defecting itself. Hence the possible pay-off for the invader in that next round is either $P$ if it defects again or $S$ if it should

[50] We will be somewhat sloppy and often replace the lengthy 'an individual employing strategy $X$' by 'strategy $X$', or just '$X$'.

[51] Note that this is fairly restrictive, in that if strategy $B$ performs as well (but no better) against strategy $A$ as $A$ against itself it could still spread in the population, a phenomenon biologists call 'drift'.

cooperate. But in the latter case they're both level again. But if $w$ is small enough then chances are that that second game, allowing TitForTat to catch up, may not happen.

We begin a proper investigation by showing that the strategy AlwaysD (which we know to be fairly successful) cannot invade a population of TitForTat strategies in the case where $w$ is large enough. By the definition of 'invading' we have to calculate the expected pay-off of AlwaysD playing against TitForTat *versus* TitForTat playing against itself. The former is

$$T + \frac{w}{1-w}P,$$

whereas the latter is

$$\frac{R}{1-w}.$$

The latter is greater than or equal to the former (meaning TitForTat can fight off the invasion) if and only if

$$\frac{R}{1-w} \geq T + \frac{w}{1-w}P$$

which is the case if and only if

$$R \geq T - wT + wP,$$

which is equivalent to

$$w \geq \frac{T-R}{T-P}.$$

**Proposition 4.3** *The strategy* TitForTat *is collectively stable provided that the parameter $w$ is greater than or equal to the maximum of $(T-R)/(T-P)$ and $(T-R)/(R-S)$.*

**Proof.** Again we assume that $w > 0$ to start with. Let us consider what happens when an arbitrary strategy interacts with TitForTat. If that other strategy is nice, then it cannot possibly do better than TitForTat against itself. Hence such a strategy will have to defect at some point, say in round $n$. From that point, the following history can develop:

- The strategy may defect forever. But in that case we may discard the first $(n-1)$ rounds in which it gets the same pay-off as TitForTat and treat it as the AlwaysD strategy which we have shown cannot invade provided that $w \geq (T-R)/(T-P)$.
- The strategy may defect $k \geq 0$ times thereafter, and then cooperate.[52] Given the definition of TitForTat that means that from round $n$ to round $n+k+1$, the strategy will accumulate a pay-off of

  $$w^n(T + wP + w^2P + \cdots + w^kP + w^{k+1}S),$$

  and thereafter it is in the same situation as before (that is, TitForTat will cooperate on the next move, and the cycle repeats).[53] Now TitForTat's pay-off when playing against itself over these $n+k+1$ rounds is

  $$w^n(R + wR + w^2R + \cdots w^kR + w^{k+1}R).$$

[52] What follows is not the proof found in Axelrod's book. He claims that it is clear that one only has to consider the 'defect forever' and the $k = 0$ case. Since I couldn't see why that is true, I decided to come up with my own proof instead.

[53] It is instructive to note that *unless* at some point it accepts the Sucker's pay-off $S$ by cooperating while TitForTat defects, both strategies will defect forever, and we are in the first case.

We want to show that the latter is as least as large as the former, that is that (after dividing by $w^n$)

$$\begin{aligned}&T + wP + w^2P + \cdots + w^kP + w^{k+1}S\\ &\quad\leq \quad R + wR + w^2R + \cdots w^kR + w^{k+1}R.\end{aligned}$$

This is equivalent to

$$T - R \leq w(R - P) + \cdots + w^k(R - P) + w^{k+1}(R - S).$$

We proceed by induction over $k$. The base case occurs when $k = 0$. But in that case the above inequality is

$$T - R \leq w(R - S)$$

which is true by the condition on $w$. The induction step, that is the proof that the statement for $k$ implies the statement for $k + 1$, works as follows. By the condition on $w$ we know that

$$T - R \leq w(T - P) = w(T - R + R - P) = w(R - P) + w(T - R).$$

But if the inequality holds for $k$ then we have

$$\begin{aligned}&w(R - P) + w(T - R)\\ &\leq \quad w(R - P) + w(w(R - P) + \cdots + w^k(R - P) + w^{k+1}(R - S))\\ &= \quad w(R - P) + w^2(R - P) + \cdots w^{k+1}(R - P) + w^{k+2}(R - S)\end{aligned}$$

which is precisely our inequality for $k + 1$. □

Hence if the probability of 'meeting again' is sufficiently large, then even a population of nice strategies such as TitForTat can protect itself against invaders. Note that no communal pressure of any sort is required to ensure this. There is another proposition in Axelrod's book stating that for any nice strategy to be collectively stable it *has* to be the case that $w$ is large enough, so TitForTat's reliance in this is shared by all other nice strategies. Also note that AlwaysD does *not* require any condition on $w$ to hold for it to be collectively stable.

We can do even better than just picking out strategies and finding out whether or not they are collectively stable: Axelrod characterized all collectively stable strategies.

**Theorem 4.4** *A strategy $A$ is collectively stable if and only if it defects when the $B$'s cumulative score so far is too great. Specifically, it defects on move $n+1$ if the score of $B$ up to game $n$ exceeds that of $A$ playing $n$ games against itself minus*

$$w^{n-1}(T + \frac{w}{1-w}P).$$

**Proof.** We do not provide a proof here; a source is given below. □

This theorem tells us that there is quite a bit of flexibility in the concept of being collectively stable, in that a strategy has a lot of points where it can choose either to defect or to cooperate provided the opponent's score so far isn't too big. This explains why so many strategies turn out to have this property. Because nice strategies do best when it comes to achieving high scores they are the most flexible when it comes to reacting to the other side's defections. However, they have to be provoked into hitting back by the first defection of the other side:

**Proposition 4.5** *For a nice strategy to be collectively stable it must react in some way to the very first defection of its playing partner.*

**Proof.** If a nice strategy does not react at all to a defection on, say, move $n$ then the strategy which defects on move $n$ and cooperates in every remaining round will exceed its pay-off by $w^{n-1}(T - S)$, and thus can invade. □

Note that this proposition does not say that the nice strategy must hit back immediately—there may be cases where it can afford to wait a move. However, TitForTwoTats does not satisfy this criterion and therefore is *not* collectively stable. We see here why Axelrod's notion of being retaliatory is required for strategies to be successful. While the advantages of being nice and being forgiving might be more immediately clear this shows that strategies which cannot be provoked into retaliation, and fairly sharply so, will have no chance of pervading. Having explored the notion of invasion by individuals we find that, maybe depressingly, AlwaysD is collectively stable in all our models, that is for all $w$. Since the starting point of this whole investigation was the question of whether cooperation could ever involve in a world where self-interest is of advantage, does that mean we have failed?

## 4.3 Invasion by clusters

As explained above, AlwaysD is immune against an invasion by a lone nice strategy. The reason for that is, of course, that the nice strategy does not have anybody else with whom to cooperate (and thus to get a higher score than its AlwaysD neighbours). But who says that such newcomers have to arrive by themselves?

Assume we have a Prisoner's Dilemma game with the pay-off matrix used by Axelrod, see page 65. Further assume that $w = .9$, that means that on average, two individuals will meet each other 10 times. Then the expected pay-off for two individuals employing the AlwaysD strategy will be 10 points (since they will always both defect). A single TitForTat strategy introduced in such a population will only ever be paired with AlwaysD strategies, and it will get a pay-off of 9 for every such encounter (the AlwaysD strategy with which it has played will get 14 rather than 10, because it can exploit TitForTat on the first move). But two TitForTat strategies paired with each other manage a rather more impressive pay-off of 30. This suggests that if there are just sufficiently many other TitForTat strategies to meet then TitForTat will be able to invade a population of AlwaysD strategies—not by individuals, but by groups.

What does 'sufficiently many' mean here? If every strategy is paired with every other strategy we can calculate a strategy's expected pay-off in one generation. Let us make the additional assumption that in our population, there is a proportion $p$ of TitForTat strategies, leaving a proportion of $(1-p)$ AlwaysD strategies. Then the expected pay-off after one generation for a TitForTat strategy is

$$30p + 9(1 - p),$$

whereas for a AlwaysD strategy, it is

$$14p + 10(1 - p).$$

The former is bigger than the latter if and only if

$$16p - (1 - p) > 0$$

which is the case if and only if

$$p > 1/17 \approx .0588.$$

In other words, as soon as just 6%[54] of all members of the population are TitForTat it pays to be nice![55]

But we don't have to have that many invaders even, provided that there is a notion of *location* present, and individuals only meet their neighbours. For example, one can place strategies within a grid and then assume that each will only meet its eight (or possibly four) neighbours.

We call such a system, in which each individual only interacts with its neighbours, a **territorial**[56] one. In a territorial system each individual gets a score for each round which is the average of its scores resulting from interactions with its neighbours. In the next generation an individual takes on the strategy employed by its most successful neighbour.[57] So we assume for the time being that interaction is among neighbours, and individuals learn from those neighbours.[58]

We have just seen that for TitForTat to survive, it is merely required that 6% of all its interactions are with other proponents of the TitForTat strategy—but that means it is sufficient if just *one* of its neighbours is another TitForTat![59] Hence in a territorial system a small cluster of TitForTat strategies can invade an AlwaysD population. Because they do three times as well with each other as AlwaysD does with itself, they do not have to meet each other very often to be successful. If $w$ is very high, for example $w = .99654$ then for TitForTat to survive it is sufficient that one of a thousand of its interactions is with another nice strategy. Hence a population of AlwaysD strategies is not impregnable. Also note that in such a world with a small percentage of invaders every member of the original population will have almost all of its interactions with another such member, so from their point of view, the invaders are not particularly relevant.[60]

[54] Axelrod compares the average score of TitForTat with of AlwaysD playing against itself and thus obtains a slightly lower required percentage.

[55] Note that because TitForTat does well with itself it will spread with increasing rapidity, whereas AlwaysD is a poor invader when it becomes too frequent—it does not do well when paired with copies of itself.

[56] It seems that in the social sciences, the term **spatial system** has become the accepted term.

[57] If there is more than one 'must successful' such strategy, one of these is chosen at random.

[58] Some researchers suggest that one might also want to consider systems where interaction is only with neighbours, but learning occurs globally, or that while interaction might occur globally, learning might only occur among neighbours.

[59] Of course, if the probability for meeting an individual again is lower then .9 then this required minimal percentage will go up.

[60] Further note that once the invaders grow in numbers, their interactions with each other will become more frequent even if they do not exclusively interact with their neighbours.

Of course if the only way of introducing such new ideas is via mutations then one might wonder whether clusters really are realistic. But if such a mutant has offspring, then chances are that those offspring live closely enough together to interact with each other, so that might be sufficient for a successful invasion in a territorial system. When the next generation is determined by one of the other mechanisms suggested above, there is no problem at all with the assumption that there may be more than one potential invader at a time.

A measure for how effective a strategy is when it comes to invading a population of AlwaysD is the proportion of interactions required with itself, $p$. Axelrod calls a strategy **maximally discriminating** if it tries cooperating at least once with every other strategy, will never cooperate more than once with AlwaysD, and will always cooperate with itself.

**Proposition 4.6** *Assume that $p$ is the smallest proportion for which there is a strategy which can successfully invade a population of* AlwaysD*. Then any strategy for which $p$ is sufficient to guarantee success is maximally discriminating.*

**Proof.** For a proof see Axelrod's book, The Evolution of Cooperation. □

Obviously TitForTat is maximally discriminating, so there is no strategy which is more efficient when it comes to invading a population of AlwaysD.

And while we have seen that for AlwaysD strategy is vulnerable to invasion by clusters (although it cannot be threatened by individuals), nice strategies enjoy better protection.

**Proposition 4.7** *Assume that in our model, any cluster of invaders is small enough that the majority of the native population will interact only with itself. Then if a nice strategy is collectively stable it cannot be invaded by a cluster of individuals.*

**Proof.** Let $A$ be a nice strategy which is collectively stable, and let $B$ be a potential invader. Assume that the proportion of $B$s among the $A$s is $p$. Then the expected pay-off in one generation for $B$ is

$$pP(B,B) + (1-p)P(B,A),$$

whereas that for $A$ is

$$P(A,A)$$

since we have stipulated that most $A$ strategies only interact with themselves. Hence the expected pay-off for $B$ is larger than that for $A$ if and only if

$$pP(B,B) + (1-p)P(B,A) - P(A,A) > 0.$$

We know that $P(A,A) = R/(1-w)$, and that this is as good as any strategy can possibly do against itself. Therefore $P(B,B) \leq P(A,A)$. So the best case for $B$ is $P(B,B) = P(A,A)$. We set $P(B,B) = P(A,A)$ in the above inequality and obtain

$$(p-1)P(A,A) + (1-p)P(B,A) = (1-p)(P(B,A) - P(A,A)) > 0,$$

which is equivalent to

$$P(B,A) > P(A,A),$$

which is precisely the condition which has to be satisfied for an individual of $B$ to be able to invade a population of $A$s. □

In other words, cooperation can develop in a world of egoism and take over that world, as long as it is not just single individuals that appear. For such a 'take-over' it is necessary that the probability $w$ for interacting with the same individual as well as that of one invader meeting another $p$ being high enough. Once they have taken over, nice strategies are fairly resistant against counter-invasions.[61]

## 4.4 Territorial systems

Let us have another look at territorial systems. Since the mechanism for determining the individuals of the next generation is quite different from the original we cannot assume that our results from Section 4.2 will carry over to these new systems.

We say that a strategy can **territorially invade** a population consisting of another strategy if, eventually, *every* location in the territory holds an individual employing the new strategy. We say that a strategy is **territorially stable** if it cannot be territorially invaded. It turns out that quite a few of our conclusions for the evolutionary systems where each individual interacts with each other individual carry over to the territorial system. In particular we have the following result.

**Proposition 4.8** *If a strategy is collectively stable then it is territorially stable.*

**Proof.** A strategy can only survive in the territorial system if there is an individual in the next generation which carries it on. But that will only be the case if it is more successful against the native strategy than that strategy is against itself, which is precisely the condition for it being able to invade a population consisting entirely of that native strategy. □

Invasions in a territorial system can follow some very intricate patterns. Figure 20 shows such a system where $w = .3$ (that is, there are not very many interactions with the same individual), $T = 56$, $R = 29$, $P = 6$ and $S = 0$. Each strategy had four neighbours. The simulation started with a single AlwaysD invader in the centre of a system of TitForTat.[62]

[61] It should be pointed out that a strategy which is immune against invasion by any one strategy might still be invaded by a mixture of two or more strategies.

[62] Given the regularity of the outcome it seems that Axelrod used the pay-off over the expected number of rounds ($1/(1 - .3) = 1.4286$) rather than a proper random parameter $w$.

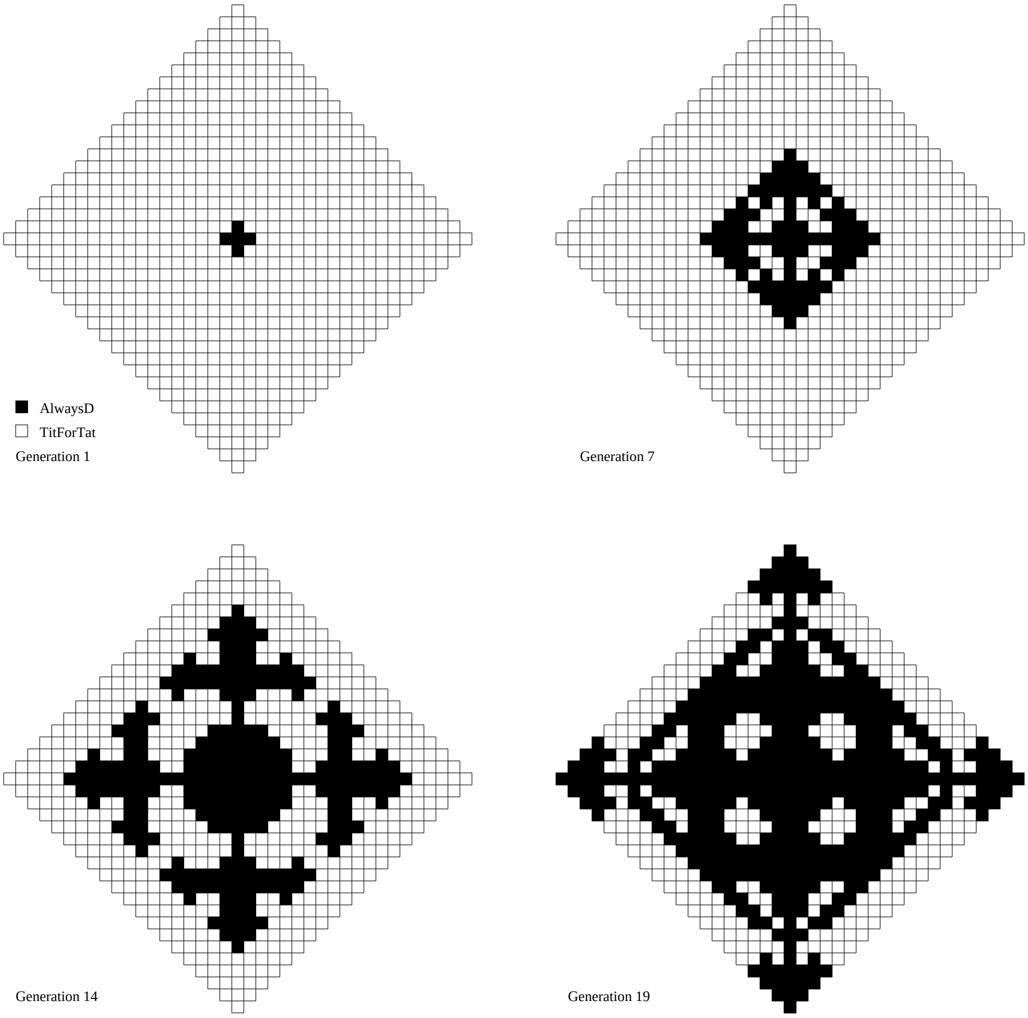



Figure 20: A territorial system

Axelrod also decided that it would be interesting to run such a territorial system with the original population being given by strategies present for his second tournament. He picked four strategies of each of the submitted ones, making for a territory consisting of $18 \times 14$ fields. Each strategy was supposed to have four neighbours, with the territory 'wrapped around' at the edges to make sure that this was also true for fields at the borders.[63] The initial placement of strategies was *random* and to make sure that this did not unduly influence the outcome he ran this tournament 10 times. After eleven to twenty-four generations, the situation stabilized, and no more change took place, since by then all surviving strategies were nice.[64] That was when his simulation stopped. Figure 21 gives a typical outcome.

[63]Topologically, the resulting structure is a *torus*, also known as (the surface of) a doughnut.

[64]Now this is proof that Axelrod indeed used *expected pay-offs* rather than simulating a behaviour where $w$ was implemented as a random variable.

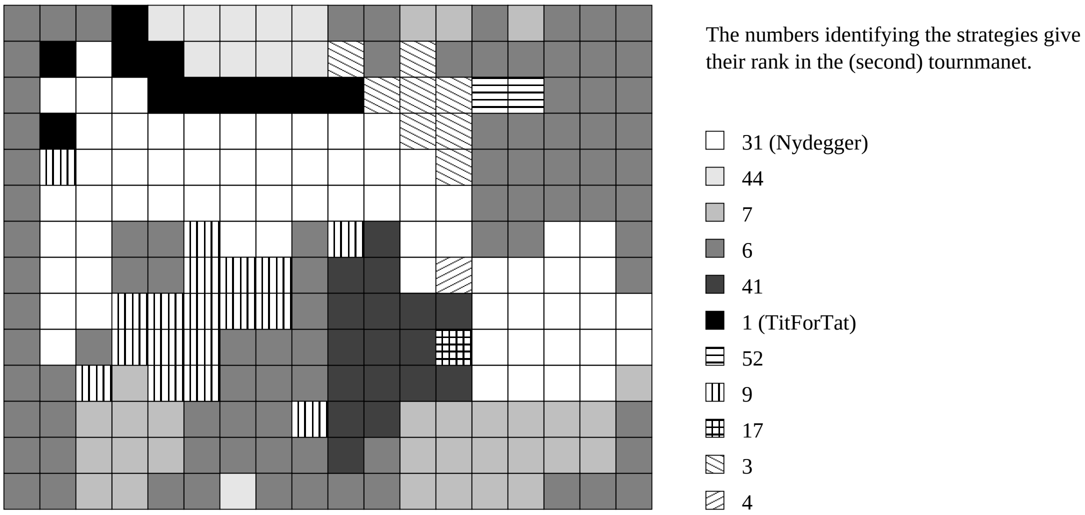



Figure 21: A territorial evolutionary tournament

The following are immediately remarkable:

- Not all surviving strategies did well in the tournament;
- not all strategies that did well in the tournament survive;
- most surviving strategies form clusters;
- all surviving rules are nice.[65]

Running this simulation a number of times, with different starting constellations, Axelrod found that a few rules did particularly well in this tournament. TitForTat, for example, on average managed to increase its population from the initial four to seventeen. A strategy Axelrod calls Nydegger, after the person who submitted it, did extremely well, *despite* the fact that it only finished 31st in the tournament proper: On average, it had 40 copies.

At first sight this seems very surprising. But before we look at Axelrod's explanation, let us investigate what this tournament actually tests: Having only nice strategies play against each other is boring in the sense that they will all get the same score on average. So in order for the tournament to have some spice, there need to be non-nice strategies to differentiate between the others. So arguably this set-up tests which rule does best when it comes to *exploiting* the non-nice strategies. Once those are all gone, the current constellation is perpetuated forever.

Nydegger is a complicated strategy which makes a number of case-distinctions based on the previous three rounds to decide what to do next. It is nice, and so never defects first. However, when the other side does so Nydegger sometimes gets the other strategy to 'apologize' by cooperating while Nydegger defects. This occurs with five of the twenty-five non-nice rules in the tournament, which was not enough to make a considerable difference then (in particular because Nydegger tended to get into trouble with the remaining non-nice rules). But in the territorial system, this changes: Whenever a copy of Nydegger is

[65] This is why there will be no more change once all other strategies have been eliminated, at least in Axelrod's set-up which does not include random events.

surrounded by nice rules as well as, say, one of those five rules which become apologetic in its presence, it will do considerably better than all of its neighbours, and probably even better than all of its neighbours' neighbours. Thus it converts a number of strategies to its own ideas.

People have argued that this set-up is not very realistic when it comes to describing real-world phenomena: The strategies present at the start are a somewhat eclectic mix, there is no notion of 'misunderstanding' or 'error'[66], and no mutations are present to change the environment. Nonetheless it certainly provided a new approach with interesting results, even if it does not give the last word on evolution.

A lesson one might take away from this is that in a system where individuals tend to imitate their successful neighbours, it really pays to be outstandingly successful under at least some circumstances (because that generates converts), even if one's average performance is below that of the average of the entire population.

## 4.5 Beyond Axelrod

### Even more strategies

As time has passed since Axelrod's original publications people have spent a lot of time on exploring these games.

One investigation suggests the use of 'Pavlovian' strategies which learn as follows: Let $n$ be a natural number. Then the strategy $n$-Pavlov, $P_n$, will cooperate and defect with certain probabilities.

The probability that it repeats its current actions will

- increase by $1/n$ if it received pay-off $R$;[67]
- decrease by $2/n$ if it received pay-off $S$;[67]
- decrease by $1/n$ if it received pay-off $P$;[67]
- increase by $2/n$ if it received pay-off $T$.[67]

If the probability that it currently cooperates is $p$ then it will cooperate in the next round with probability

- $p + 1/n$ if it received pay-off $R$;[67]
- $p - 2/n$ if it received pay-off $S$;[67]
- $p + 1/n$ if it received pay-off $P$;[67]
- $p - 2/n$ if it received pay-off $T$.[67]

Note that 0-Pavlov is just a form of the Random strategy, and that 1-Pavlov will

- cooperate in the next round if both parties chose the same move in the current round;
- defect in the next round if both parties chose different moves in the current round.

Typically it does not matter too much which initial probability is chosen, as long as sufficiently many games are being played.

These strategies can be viewed as *learning* from the experience they make. Because the the result of the previous round is treated as a stimulus, this form of learning fits into Skinner's

[66] These have been explored to some extent now but these ideas lead us too far afield. We merely note that TitForTat is vulnerable against such misunderstandings; its pay-off will go down if it mistakenly assumes the other side defected.

[67] Here $+$ and $-$ is meant to be bounded in that the result is at least 0 and at most 1; that is, if $p = .9$ and $n = 3$ then $p + 1/n = .9 + 1/3 = 1.\overline{3}$, but the probability $p$ is adjusted to 1.

operant conditioning model for learning.[68] This is also a model which is deemed realistic when it comes to describing animal learning. When paired with a responsive strategy, the various PAVLOV strategies eventually reach a state where they cooperate almost exclusively. They typically outperform TITFORTAT against versions of the RANDOM strategy, provided the probability for cooperation is at least 1/2. It can, however, take such a strategy a fairly long time to learn to cooperate when paired with another Pavlovian strategy or TITFORTAT.

**Probabilistic models**

Above we criticized Axelrod's set-up as not being very realistic. Since his pioneering work, other people have considered slightly different scenarios.

Let us begin by running a thought-experiment to discover what happens in the presence of mutation. In a world which consists largely, or even exclusively, of TITFORTAT or other nice strategies a mutation leading to an ALWAYSC strategy would survive without problem. But the presence of such mutants in turn might allow strategies to regain a foothold which exploit such generosity, such as ALWAYSD. Hence it seems that more realistic models will lead to *cycles* the population goes through.

Inspired by Axelrod's results, Nowak and Sigmund decided they were going to try this kind of tournament which would be more suited to modelling populations from a biological point of view. They agreed that for their purposes it would be sufficient if they only allowed strategies observing a certain pattern; they named these *reactive strategies*. Such a strategy has three parameters, $p$, $q$ and $r$, all of which are probabilities. A strategy $R(r, p, q)$ is defined as follows. It will

- cooperate on the first move with probability $r$;
- cooperate with probability $p$ if the other player cooperated in the previous round;
- cooperate with probability $q$ if the other player defected in the previous round.

Then the ALWAYSD strategy is nothing but $R(0, 0, 0)$, and TITFORTAT is $R(1, 1, 0)$, whereas ALWAYSC is $R(1, 1, 1)$. There is also a generous version of TITFORTAT, known as GENTITFORTAT: It has $r = p = 1$, but rather than cooperating with probability 0 when the other side has defected last, it will cooperate with probability

$$\min\{1 - \frac{T-R}{R-S}, \frac{R-P}{T-P}\}.$$

But Nowak and Sigmund decided that, in order to model error, they would not allow any strategies where $p$ and $q$ were equal to 0 or 1: Their idea is that no being is that perfect. The effect of the initial move (decided by $r$) is not significant if there are sufficiently many rounds, so we will not mention it further.

They seeded their population with a random selection of these strategies. (Note that there are infinitely many possibilities.) They found that for most of these, the strategies that do best are those closest to ALWAYSD, that is, strategies for which $p$ and $q$ are close to 0. However, if there is at least one TITFORTAT-like strategy in the initial population then everything changes: At the start, this strategies (and its copies) struggles to survive. But

[68] And, in fact, biologists have suggested that these strategies should have been named after him, since the model of learning employed is not really the classical conditioning one pioneered by Pavlov.

inevitably, those strategies where both $p$ and $q$ are relatively large (for $q$ being large, these are the 'suckers') are reduced in numbers, and that is when the tide turns and TitForTat-like strategies start growing in number at the cost of the AlwaysD strategies. But once the exploiters have gone, it is GenTitForTat which takes over, and then evolution stops. Nowak and Sigmund concluded that while TitForTat is vital for cooperation to evolve, persistent patterns of cooperation in the real world are more likely to be due to GenTitForTat.

They then ran a second series of simulations, with a wider class of strategies. They decided to allow four random values to describe a strategy, $p_1$, $p_2$, $p_3$, and $p_4$ so that it would be possible to take the strategy's own last move into account and not just the other player's. A strategy $S(p_1, p_2, p_3, p_4)$ will cooperate on the next move with

- probability $p_1$ if in the current round, both players cooperated;
- probability $p_2$ if in the current round, it cooperated while the other side defected;
- probability $p_3$ if in the current round, it defected while the other side cooperated;
- probability $p_4$ if in the current round, both sides defected.

Now TitForTat can be represented as $S(1,0,1,0)$. There was an initial population of strategies all playing $S(.5,.5,.5,.5)$, and every 100 generations a small number of randomly chosen mutants was introduced. They used the proportional evolutionary model rather than the territorial one. After 10 million generations, 90% of all simulations had reached a state of steady mutual cooperation. But in only 8.3% of these was the dominating strategy TitForTat or GenTitForTat; in the remaining ones it was strategies close to $S(1,0,0,1)$ which flourished. But this is precisely the strategy 1-Pavlov! This strategy had been disparagingly called 'simpleton' by Rapoport and others: It cooperates with AlwaysD on every other move, and against TitForTat it can be locked into a sequence where it receives repeating pay-offs of $T$, $P$, $S$. Nowak and Sigmund argued that the reason for this strategy doing so well is that it makes it *harder* for strategies like AlwaysD to gain a foothold. Strategies like this do best in an environment that contains generous strategies, such as AlwaysC and GenTitForTat. But 1-Pavlov makes it harder for these strategies to develop, and that in turn means that in populations where it is present, the circumstances that allow AlwaysD to invade don't usually develop. A rational explanation for this kind of behaviour is provided by the observation that this strategy stays with its previous decision if it received the higher of the two pay-offs available (that is $T$ and $R$), and in the remaining cases changes its mind in the next move.

**Models based on finite state machines**

Other researchers did not like the idea of there being so much randomness involved in these situations, and they decided instead to explore simulations where all strategies are represented by finite state machines. Figure 22 shows TitForTat in their setup.

This machine has two states, one in which it will cooperate on the next move, which is labelled $C$ (which is also the start state) and one where it will defect on the next move, which is labelled $D$. The labels along the arrows stand for the other side's actions in the current round. In other words, if the machine has been cooperating, and the other side has cooperated, it will keep cooperating.

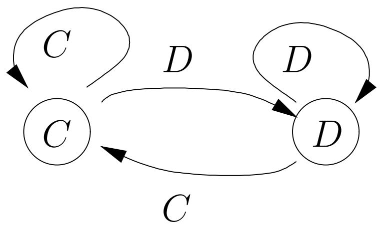



Figure 22: TitForTat as a finite state machine

Linster conducted a tournament where he used all strategies which can be expressed as such automata with two states (it would be possible to allow a longer history to be used). However, there are several ways of encoding the AlwaysC and AlwaysD strategies using two states, and he made sure to only include one copy of each. He thus ended up with 22 strategies. He ran a number of evolutionary tournaments. Sometimes he allowed truly random mutation to occur, sometimes only between machines which were sufficiently related. Sometimes mutations were assumed to be very rare events, sometimes he thought of mutants as an invasion force and allowed as much as 1% of the original population to be replaced by mutants.

In his tournaments, no single strategy ever ended up dominating a population in the way it had occurred with Nowak and Sigmund's. The strategy that generally did very well by comprising over 50% of most populations translates into $S(1,0,0,0)$ (with cooperation being its first move). It is the Grudge strategy which is described above. It does not do well in the randomized world even with itself, because once a random defection occurs it will defect forever. Other strategies that did well, if not as well as Grudge, were TitForTat, 1-Pavlov, AlwaysC and the initially cooperative version of $S(0,1,1,0)$. His results suggest that there may be stable mixes of strategies (rather than stable populations dominated by just one strategy) and that there may be stable *cycles* that a population might go through.[69]

Since these results the notion of 'evolutionary stability' has been studied in some detail, and a number of different definitions exist for this concept. Only recently have researchers begun to study the relationship between these, and to investigate what properties a strategy has to have to satisfy any of them. This is a complex area of current research and we have gone as far as we can in the course of these notes.[70] It is also not clear at this point whether these mechanisms can be used to describe cooperation currently existing in nature, but it is certainly the most convincing model found so far.

[69] If one introduces 'noise' as a method of error into Nowak and Sigmund's simulations then one does indeed obtain populations where the proportions of some strategies will 'oscillate', while others vanish entirely. TitForTat is the only strategy which can obtain high numbers, but whenever that is the case, it will be outperformed by more generous strategies which in turn are invaded by parasitic strategies. That allows AlwaysD and Grudge to gain a foothold, which in turn are ousted by TitForTat.

[70] One of the problems is that it is very much dependent on the original population which strategies will emerge as successful in a given simulation, and the results are not stable when additional factors are introduced, such as noise (or error), payment for complexity of employed strategies, changing pay-offs, and the like.

### The use of simulations

Among social scientists there is considerable debate about the merit of the kinds of situations that we have discussed. However, in situations which are too complex to yield easily to a purely theoretical approach this seems the only method which provides some insight. Once some general behaviour patterns are known it is sometimes possible to make a rigorous argument explaining them.

Some tests have been made to help judge the outcome of simulations. In particular an evolutionary set-up has been used in finite round games, where the participants knew how many rounds there were going to be. This finite set-up also makes it possible to have *all* available strategies present at the start, and thus avoiding that the deck is being primed for some specific outcome. Such simulations favour TitForTat after a few round, which is then overtaken by a variant which defects in the last round. That, in turn, is replaced by a strategy which defects one round earlier, and ultimately it is AlwaysD which dominates the population. Since this is the expected outcome due to the fact that AlwaysD provides the unique Nash equilibrium, this is encouraging. There certainly is no reason to ban simulations from the collection of available methods. People using them just have to be aware of how their initial set-up might influence the result (and thus be careful about it), and to which extent their findings have to be taken with a grain of salt.

### Towards greater Realism

Since Axelrod's original results, simulations have been used to investigate a number of different scenarios, for example one where a strategy may decide to try to find a new partner. In that kind of world, versions of the AlwaysD strategy try to find other individuals to exploit when their current partner has become wise to their ways. How successful such a behaviour is depends on how large the population is and on how difficult it is to find another individual who is willing to give cooperation a go. Other possibilities allow for individuals to observe the behaviour of others and then behave accordingly—if they saw the other defect such a strategy would defect itself when partnered with the offender. However, such an observant strategy only out-scores TitForTat if $w$ is relatively small, which is an environment where strategies which tend to defect do well. Yet other simulations have been conducted in which the probability of meeting again, $w$, varies, or even ones where the pay-offs $T$, $R$, $P$ and $S$ are subject to change during the simulation. Finally there is some recent work using *genetic algorithms* and other methods originating in the field of machine learning to model strategies which can learn, and to see whether that leads to yet different dominating strategies. When using any form of machine learning in an attempt to find successful strategies, agents are typically presented as finite state automata in the way described above. This raises a number of new problems, in particular about how much *memory* agents should be allowed, that is, how many moves they are allowed to remember to base their decisions on.[71]

## 4.6 More biological games

The Prisoner's Dilemma is far from the only game used in biology to model various situations. A typical example, often used to model fights (for example among males for females) is the 'Hawk-Dove' game.

[71] Some authors introduce costs for complexity, giving even more parameters to play with.

For many species fights among individuals only rarely end with one of the fighters being seriously wounded or even dead. An example are stags fighting for a group of females. They start with a prolonged roaring match, followed by a parallel walk, followed by a direct contest of strength where the two interlock antlers and push against each other (always assuming one of the contestants, usually the intruder, does not quit by retreating first). Why does not one of the stags attack the other during the 'parallel walk' phase, where the flank of the opponent makes an enticing target? Such an aggressive stag might well have advantages if all other stags would retreat under such an assault.

To explain this and a number of similar phenomena, consider a game where there are two strategies, the HAWK and the DOVE strategy.[72] The DOVE strategy will pretend that it is willing to fight, but when the situation gets serious it will retreat. The HAWK, on the other hand, will keep fighting until either it is too severely injured to continue or until the opponent retreats.

What might the pay-offs be of one such strategy playing another? Let us assume that they are fighting for some 'gain in fitness' (a better territory, food, females—all factors in the quest to pass on one's genes) $G$. If a HAWK meets a DOVE then the DOVE will run away, leaving the HAWK with a pay-off of $G$. If HAWK meets HAWK, then a serious fight will ensue. Let us say that on average that will reduce the loser's fitness by $C$ (due to injury, maybe even death). Assuming either HAWK has a .5 chance of winning, the pay-off is $(G-C)/2$ for each of them. It is typically assumed that $C$ is bigger than $G$, making $G-C$ negative. If two DOVES meet each other they may pretend to fight ('display', that is a way of contesting which does not cause injuries to either party) for a long time, which we assume comes at a cost of $L$. So the winner gets $G-L$, and the loser $-L$. If again each side has a .5 chance of winning, the expected pay-off is $(G-2L)/2$. Hence this game can be described by the following matrix giving the pay-off for Player 1.[73]

| | HAWK | DOVE |
|---|---|---|
| HAWK | $(G-C)/2$ | $G$ |
| DOVE | $0$ | $(G-2L)/2$ |

It is assumed that $L$ is much smaller than $C$. The fewer HAWKs there are the better the chance of meeting a DOVE, and the better HAWKs do on average. Let us consider a specific example. Let $G=50$, $C=100$ and $L=10$ (points). This is the resulting pay-off matrix.

| | HAWK | DOVE |
|---|---|---|
| HAWK | $-25$ | 50 |
| DOVE | 0 | 15 |

In a population consisting entirely of DOVEs, on average the score from a contest is 15, which looks decent. Now assume a mutant HAWK turns up. Then in every contest, that HAWK will meet a DOVE, always gaining 50 points. This is much better than a DOVE manages, and therefore the HAWK genes will spread quite rapidly, leading to an increase in the number of HAWKs.

[72] These names have nothing to do with the behaviour of the animal they are named after, but fit common perceptions. Apparently, doves are fairly aggressive against each other.

[73] This is another example of a *symmetric game*, where the pay-off of Player 2 is given by the transpose of the matrix for Player 1, compare the Prisoner's Dilemma, page 61.

But if the HAWKS become too successful, they will be their own downfall: In a population consisting entirely of HAWKs the average pay-off from a contest is $-25$! A single DOVE in such a population is at an advantage: While it loses all its fights, it at least gets an average pay-off of 0 as opposed to $-25$. This would lead to an increase of the number of DOVEs.

But does the population have to oscillate between the two extremes? Is there no stable population? In a population with a proportion of $p$ DOVEs and $(1-p)$ HAWKs, the average pay-off of one contest for a DOVE is

$$p\frac{G-2L}{2},$$

and that for a HAWK

$$pG+(1-p)\frac{G-C}{2}.$$

In our example, the former is $15p$, and the latter

$$50p-25(1-p)=75p-25.$$

In a balanced population, neither is at an advantage, that is, the two average pay-offs are equal. This happens precisely when

$$15p=75p-25$$

which is true if and only if $p=5/12$. A population with a proportion of DOVEs to HAWKS of 5 to 7 is stable, and the average pay-off for an individual of the population (no matter whether HAWK or DOVE) is $75/12=6.25$. Note that if everybody agreed to be a DOVE, there would be a much higher pay-off per contest for the individual, and thus for the entire population! But, as we have seen, such a population wouldn't be stable.[74]

Note that a mixed population is *not* the only way of reaching a stable population. We could interpret the game differently, namely as one where the pure strategies are the HAWK and DOVE strategy, but where each contestant picks a *mixed* strategy for himself.

We claim that the balanced population also gives us the clue to finding an equilibrium point[75]: In other words, $((7/12,5/12),(7/12,5/12))$ is an equilibrium point for the game. We prove this by applying Proposition 2.4. We calculate the following pay-offs for Player 2 when playing against $(7/12,5/12)$:

| | Player 2's strategy | | |
|---|---|---|---|
| | $(7/12,5/12)$ | $(1,0)$ | $(0,1)$ |
| Player 2's pay-off against $(7/12,5/12)$ | 6.25 | 6.25 | 6.25 |

Since his pay-off does not increase (compared to his playing the claimed equilibrium point strategy $(7/12,5/12)$) we have verified the required inequalities for Player 2. But the game is symmetric, so if Player 1 changes to a pure strategy instead his pay-off will be just the same. Hence we have indeed verified that $((7/12,5/12),(7/12,5/12))$ is an equilibrium point.

[74]The population with the highest average pay-off would be one consisting of 1/6 HAWKs and 5/6 DOVEs, leading to an average pay-off per contest of 50/3.

[75]If there are more than two pure strategies then this correspondence between mixed strategy and pure strategy stability is no longer true. There are games which have a stable mixed strategy population but the corresponding pure strategy one is not stable, and *vice versa*.

It turns out that this is also the *only* equilibrium point (which we do not prove here). As a consequence the only stable population is everybody adopting the mixed strategy $(7/12, 5/12)$: if an intruder playing some other strategy enters the population his pay-off when playing against $(7/12, 5/12)$ cannot by higher than that strategy playing against itself; if it were then members of the population would be better off by switching to the intruder's strategy, but by the definition of equilibrium point this cannot be the case.

And here we get the connection with equilibrium points. Since this game is symmetric, an optimal strategy for Player 1 is also optimal for Player 2. Hence we are looking for a strategy which is a best response to itself. When solving the equilibrium point equation we find that it is *precisely* the equation we solved above. In other words, $((7/12, 5/12), (7/12, 5/12))$ is the sole equilibrium point for this game. Clearly in a population consisting entirely of such optimal strategies, every invader will do worse against these than they do against themselves, and therefore such a population cannot be invaded. However, if there are more than two strategies around (and contests are on a one-on-one basis) then this changes. Also among biologists the idea that an invader would have to *outperform* the resident strategy to succeed is not accepted, so they do not consider the equilibrium point as a truly stable situation: Strategies which perform *as well* against the resident strategy as that strategy does against itself might still spread.

**Exercise 22** (a) Assume that the game is played with the gain for winning a fight being 40, losing a fight costs 120 and the cost of a display is 10. Calculate a stable population for this game.

(b) What is a stable population in the general game, using the first matrix on page 93? What happens if we cannot assume that $G < C$?

This game can be more interesting by adding more strategies to the mix. There is, for example RETALIATOR: It starts by behaving similar to a DOVE, but when attacked (by a HAWK, for example), it retaliates. Hence it behaves like a HAWK when paired with a HAWK, and like a DOVE when paired with a DOVE. The pay-off matrix thus derived is given below.

| | HAWK | DOVE | RETALIATOR |
|---|---|---|---|
| HAWK | $(G-C)/2$ | $G$ | $(G-C)/2$ |
| DOVE | $0$ | $(G-2L)/2$ | $(G-2L)/2$ |
| RETALIATOR | $(G-C)/2$ | $(G-2L)/2$ | $(G-2L)/2$ |

If $L = 0$ in this game, then the only stable population is a mixture of HAWKS and DOVES, without any RETALIATORS. If we add a fourth strategy, BULLY, which behaves like a HAWK until it is seriously attacked (by a HAWK, for example) in which case it turns into a DOVE, then there is *no* stable population at all, but the system oscillates.

For the above matrix RETALIATOR and DOVE are indistinguishable in the absence of a HAWK. A suggestion to remedy this is to assume that when paired with a DOVE, there is a slight chance that RETALIATOR may find out that escalating the fight will win it. It then seems only fair to assume that a HAWK has an advantage when paired with a RETALIATOR since it will escalate first. An adjusted matrix for the three strategy game with $L = 0$ might look somewhat like this:

| | HAWK | DOVE | RETALIATOR |
|---|---|---|---|
| HAWK | $(G-C)/2$ | $G$ | $(G-C+E)/2$ |
| DOVE | $0$ | $G/2$ | $(G-E)/2$ |
| RETALIATOR | $(G-C-E)/2$ | $(G+E)/2$ | $G/2$ |

This game has two stable populations, one consisting entirely of RETALIATORs and one consisting of a mixture of HAWKs and DOVEs. We will not work any of these out in detail; they are just meant to give an idea of the variety of situations that are possible with this setup.

There are other strategies one might add to this game, and there are different games that describe slightly different situations. In particular when the potential gain $G$ is small, contests often become *asymmetric*: The two contestants do not fight on equal grounds, for example because one is an intruder and the other on home territory.[76] In such fights there typically is a considerable advantage for the home side. This seems sensible, because the home side knows the territory in question, and there are good reasons for striving to be a resident.[77] This makes fights a lot shorter, and thus less costly, and gives a 'natural' solution, namely a stable population.

There are many more biological models than we can cover here. Biologists have found applications for bi-matrix games (which defy analysis via equilibrium points to some extent), they consider games with a continuum of strategies[78] and they have found systems evolving towards stable populations. In these systems, an equilibrium point can act as an attractor (the system will inexorably move towards it, unless it started too far away), as a deflector (the equilibrium point is so delicate that the population will develop away from it), and there can be systems which are so unstable that the best we can say is that the oscillate between certain states. There are also models for populations of different species interacting with each other, for example in a predator-prey relation. The references given below provide some pointers towards literature covering these situations.

## Summary of Section 4

- The indefinitely repeated Prisoner's Dilemma can be used to model *evolution* of traits. There is no single best strategy for this game if the probability $w$ of another round being played is large enough.

- The relevant question then becomes which strategies are *collectively stable*, that is, which are safe from invasions. Examples of such strategies are ALWAYSD (always), and TITFORTAT (if $w$ is large enough). Nice strategies have to react to the first defection of a playing partner to be collectively stable, and one can define a rule of when a collectively stable strategy will have to defect.

- Invasion becomes a more likely proposition for nice strategies if they invade in small clusters, but nice collectively stable strategies are safe against such invasions. In many ways, TITFORTAT is as successful a strategy as it can be in such a world.

[76] A certain kind of butterfly, for example, seeks out sunny spots in the hope of being joined by a female. If the spot is already occupied, the intruder gives up very quickly.

[77] Although there's a type of Mexican social spider which, when disturbed tries to find a new hiding place. If it darts into a crevice occupied by another spider the occupant will leave and seek a new place for itself.

[78] Like the 'War of Attrition'.

- We can model the idea of localized interaction in *territorial system.*
- There are a number of models which go beyond Axelrod by introducing noise, simple learning based on ideas such as probabilistic strategies or finite state machines.
- There are other games such as the *Hawk-Dove* game that are used in biology to explain the point of balance of stable populations.

### Sources for this section

R. Axelrod. **The Evolution of Cooperation.** *Basic Books, Inc.* 1984.

The entry on the Prisoner's Dilemma in the **Stanford Encyclopaedia of Philosophy**, (mirrored) at `http://www.seop.leeds.ac.uk/entries/prisoner-dilemma/`.

R. Dawkins. **The Selfish Gene.** *Oxford University Press*, 2nd edition, 1989.

B. Brembs. *Chaos, cheating and Cooperation: potential solutions in the Prisoner's Dilemma*, in: **Oikos** 76, pp. 14–24 or at `http://www.brembs.net/papers/ipd.pdf`.

D.R. Hofstadter. *The Prisoner's Dilemma and the Evolution of Cooperation.* In: **Metamagical Themas.** *Basic Books*, 1986.

For a proof of Theorem 4.4, see: R. Axelrod. *The Emergence of Cooperation Among Egoists.* In: **American Political Science Review** 75, pp. 306–318.

For a general account of games used in biology, see: K. Sigmund. **Games of Life.** *Penguin*, 1993.

For a fairly mathematical treatment of game models used in biology, with an analysis of the family of strategies $R(y, p, q)$, see J. Hofbauer and K. Sigmund. **Evolutionary Games and Population Dynamics.** *Cambridge University Press*, 1998.

For another treatment not unlike the previous one by the original creator of this branch of biology, see: J. Maynard Smith. **Evolution and the Theory of Games.** *Cambridge University Press*, 1982.

For a survey on simulations that have been conducted, in particular in connection with learning, see Robert Hoffmann's PhD thesis, available from his home-page at Nottingham, `http://www.nottingham.ac.uk/~lizrh2/hoffmann.html`.

There is an annotated bibliography for the Prisoner's Dilemma (from 1994) available at `http://pscs.physics.lsa.umich.edu/RESEARCH/Evol_of_Coop_Bibliography.html`. It does not include recent results.

# 5 Medium games

This section is concerned with games which are too big for it to be feasible to find all the strategies. If we cannot list all the strategies then we cannot transfer a game into its normal form. As the title indicates we do not, however, give recipes for games of arbitrary size. There is a further section on even larger games. The limitations of these techniques and the size of game to which they can be applied should become clear as you read this section.

## 5.1 The algorithmic point of view

So far we have not given any serious thought to tackling games which are beyond the fairly small. While the principles developed in Section 2 apply to all games, we argued even then that they are not feasible for games of a larger size. All the examples we have seen so far are, in fact, really small so that they could be treated without computer support. So what if we want to use machines?

Let us consider the various tasks that arise when we follow the method given in Section 2. Given a game tree we

- generate all the strategies for all the players;
- calculate the pay-offs when playing the various strategies against each other;
- find equilibrium points.

The first two tasks come with algorithms for performing them, and these were described in Section 1, on pages 16 and 21 respectively.

The last task identified above, namely that of calculating equilibrium points, is problematic if there are more than two players involved, and for that case, no algorithm for finding equilibrium points is known. Still, at least there is an algorithm for making the matrix smaller by removing strategies which are dominated by others. This is as far as we can get with using computer support for the method described in Section 2. That section contains all the necessary information to turn these algorithms into programs.

We might try to apply what we have learned so far while abandoning the notion of equilibrium points in order to be able to enlist computer support. We can, for example, calculate, for each player, his best choice at every stage when assuming that the other players will do their worst (from his point of view).

But what is the difference between doing that and finding an equilibrium point? In an equilibrium point *all* the players have reason to stick with their current strategy, and all have reason to be content with the situation in which they find themselves. The whole system is 'in balance' with itself and thus stable. If all players opt for their best choice under the assumption that the other players always do their worst, they fail to take into account that the other players are *not* trying to minimize somebody else's profit, but are rather interested in maximizing their own. A player may therefore go down a route he considers the safest based on the assumption that the worst case might happen. But going down another may lead to a higher pay-off for him, because the other players are interested in maximizing *their* pay-off rather than in minimizing his.

One might consider carrying out an analysis which chooses a strategy for each player according to these criteria (best outcome in the worst case scenario), and then think about whether any of the players, given that knowledge, might not safely diverge from this strategy to gain a better pay-off. Alas, this would lead to an iterative process *that need not terminate.* As a decision procedure, that is pretty poor. It should be noted that even if there are only two players then this problem still persists. This way of thinking only makes sense if *what's good for the first player is automatically bad for the second player, and* vice versa. Hence this sort of method only works convincingly for *2-person zero-sum games.* Examples for the different cases appear below.

## 5.2 Beyond small games

Using computers makes it possible to tackle games for which finding all the strategies would take far too long. In order to think about this we need to think about how various numbers grow with the size of the game tree.

**Question 22** How does the number of positions grow with the size of the game tree? How does the number of strategies (added up over all players) grow with the size of the game tree?

If some game tree has at least two choices at each decision point[79] then the number of positions of a tree of height $m$ is at least $2^{m+1} - 1$. In other words the number of positions is *exponential* in the height of the game tree.

Counting strategies is more difficult, because their number depends on how many decision points there are for each player. If all the decisions are made by the same player then the number of strategies in a game tree of height $m$ is the same as the number of final positions, that is $2^m$. But this is not very realistic, and if the decision points are distributed evenly then the number grows much more quickly.

Here is a table counting the number of positions where we assume that

- the game tree is a complete binary tree, that is, at every decision point there are precisely two choices and
- there are two players who move strictly alternating, starting with Player 1.

| height | no pos. | strats for P1 | strats for P2 | all strats |
|---:|---:|---:|---:|---:|
| 0 | 1 | 1 | 1 | 2 |
| 1 | 3 | 2 | 1 | 3 |
| 2 | 7 | 2 | 4 | 6 |
| 3 | 15 | 8 | 4 | 12 |
| 4 | 31 | 8 | 64 | 72 |
| 5 | 63 | 128 | 64 | 192 |
| 6 | 127 | 128 | 16384 | 16512 |
| 7 | 255 | 32768 | 16384 | 49152 |
| 8 | 511 | 32768 | 1073741824 | 1073774592 |

**Exercise 23 (Optional)** (a) Take a complete binary tree of height 3. How many decision points does it have? Try different ways of assigning those to two players and count the number of strategies for each.

(b) In the table above, how can one calculate the number of strategies for each player from the previous entries?

[79] It would be sufficient for that to be the case at *most* decision points.

Because of the exponential growth, generating all strategies is expensive. With a large game tree there are also problems with storing strategies efficiently—they will take up a good deal of space if stored as subtrees of the game tree.

A possible solution might be a way of finding 'good' strategies without having to list *all* strategies. This would vastly increase the number of games that we can cope with.

We have been tacitly assuming that the game tree is somehow stored in the computer, a clearly necessary requirement. Tree data structures (as introduced in other modules) can be used to store the entire game tree. But in many cases this is neither practical nor required. For a program to work out all the possible positions, and how they are related via moves, it merely needs enough information to *generate* the tree.

This brings us back to some considerations from the very beginning of the course: It can be quite easy to describe a game (Chess, or Go) but the resulting game tree can be huge. Hence it would be ideal if we could describe an algorithm for finding good strategies which only requires *local* information at any given time. This is another feature that makes the algorithms we are about to introduce so appealing.

### 5.3 The minimax algorithm

The algorithm described below should remind you of the proofs of Theorem 1.1 and Proposition 2.3. What we did there, in fact, was to describe an algorithm for finding the *value* of each sub-game for a game, and once we had assigned values to each of the immediate sub-games, we could calculate the value of the overall game. The motivation we are going to use here looks slightly different at first sight, but it turns out to be the same thing.

For a game of perfect information we define the **value of a position $p$ for Player $X$**, $v_X(p)$, to be the pay-off that he can guarantee to achieve (at a minimum) from that position.[80] is The fact that the player can guarantee this means that he will have to be able to do so even if the other players do their worst, so it does agree with the philosophy outlined above. It also means that the player must have a *plan* for achieving his target, in other words, he has a *strategy* for doing so. There is only one slight problem with this idea: It does not say how to cope with elements of chance. In such a case we stick with the idea that it is the *expected pay-off* which matters. It does not really make sense to assume that 'chance (or nature) will do its worst' as far as our player is concerned, because the nature of chance is precisely that over sufficiently many experiments the average pay-off will be close to the the expected one.[81] This does assume, however, that when we assigned values to outcomes we did it in such a way that they faithfully record what we think of them (compare the discussion on page 27).

**Question 23** Why do we insist on having a game of perfect information here?

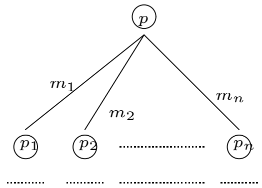



Figure 23: The value of a position

[80]For a 2-person zero-sum game the *value of the game* as defined in Section 2 is the *value of the root position for Player 1.* The *value of the root position for Player 2* is the *negative* of that number.

[81]One could come up with a related notion which does indeed always consider the worst case, but I am not aware of any good applications for this.

**Proposition 5.1** *Let $p$ be a position with $p_1$, $p_2$, …, $p_m$ the positions which can be reached from $p$ with one move (see Figure 23). Then the value for Player $X$ of $p$, $v_X(p)$, is*

$$v_X(p) = \begin{cases} p_X(p) & \textit{p is final} \\ \max_{1\le i\le m} v_X(p_i) & \textit{it is X's turn at p} \\ \min_{1\le i\le m} v_X(p_i) & \textit{it is some other player's turn at p} \\ q_1 v_X(p_1) + q_2 v_X(p_2) + \cdots + q_m v_X(p_m) & \textit{a chance move occurs at p; the} \\ & \textit{probability of } p_i \textit{ being chosen is } q_i \end{cases}$$

**Proof.** This is obvious from the definition. If it is Player $X$'s turn at $p$ then he can choose the position with the maximum value and from there use whatever strategy to achieve that value. If it is some other player's turn he cannot do better than guarantee the value that he might get in the worst case, which is the minimum of the values that can be reached from $p$ with one move. If it is a chance move then the minimal expected pay-off from $p$ is $q_1 v_X(p_1) + q_2 v_X(p_2) + \cdots + q_m v_X(p_m)$. □

The values of the root (for all players) tell us the minimal expected pay-off for each player. But this value may be different for different players, and the final positions each player is aiming for need not be the same. We are unable to assign a *single* value to a game unless we are talking about a 2-person zero-sum game. People often refer to positions where it is $X$'s turn as *max positions* and positions where it is not $X$'s turn as *min positions*, because that is the operation used to calculate its value. Most people do *not* include chance moves as a viable option for such games.

What we have described here is a *recursive algorithm*: To find the value of a position, we need to know the values of all its immediate successors, and to calculate those we need to know the values of *their* immediate successors, and so forth, until we reach a final position, where we can read off the value from the pay-off function. When it comes to computing the value of the root of game tree it is tempting to apply a *bottom-up* approach, that is, to start at the final positions and work one's way up from there. This approach is demonstrated in Figure 24.

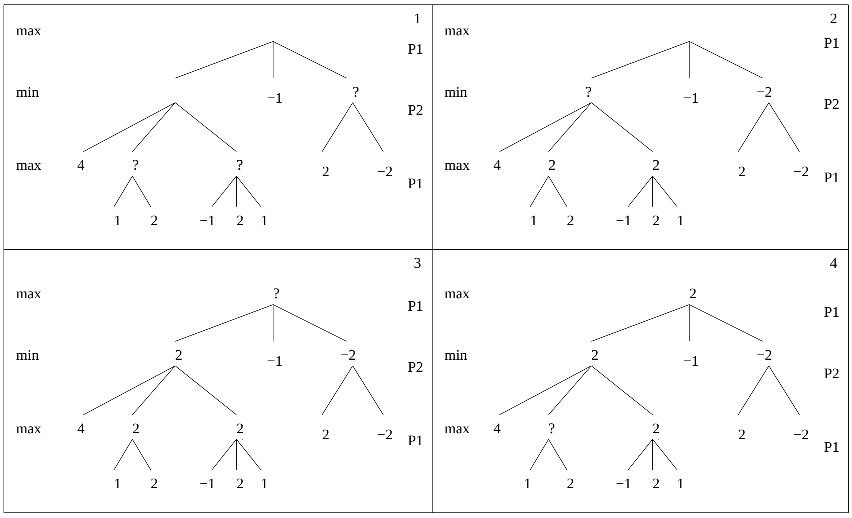



Figure 24: A bottom-up approach to determining the value

But if you had to program this algorithm then starting at the final positions and working from there is not at all convenient—it requires a lot of unnecessary traversal of the tree, with many positions having to be kept in memory at the same time (or generated repeatedly). Instead this is typically programmed using a *depth-first search* algorithm. An example of calculating $v_1$ (the value for Player 1) for the root of the game tree (and thus for all the positions) is given in Figures 25 and 26.

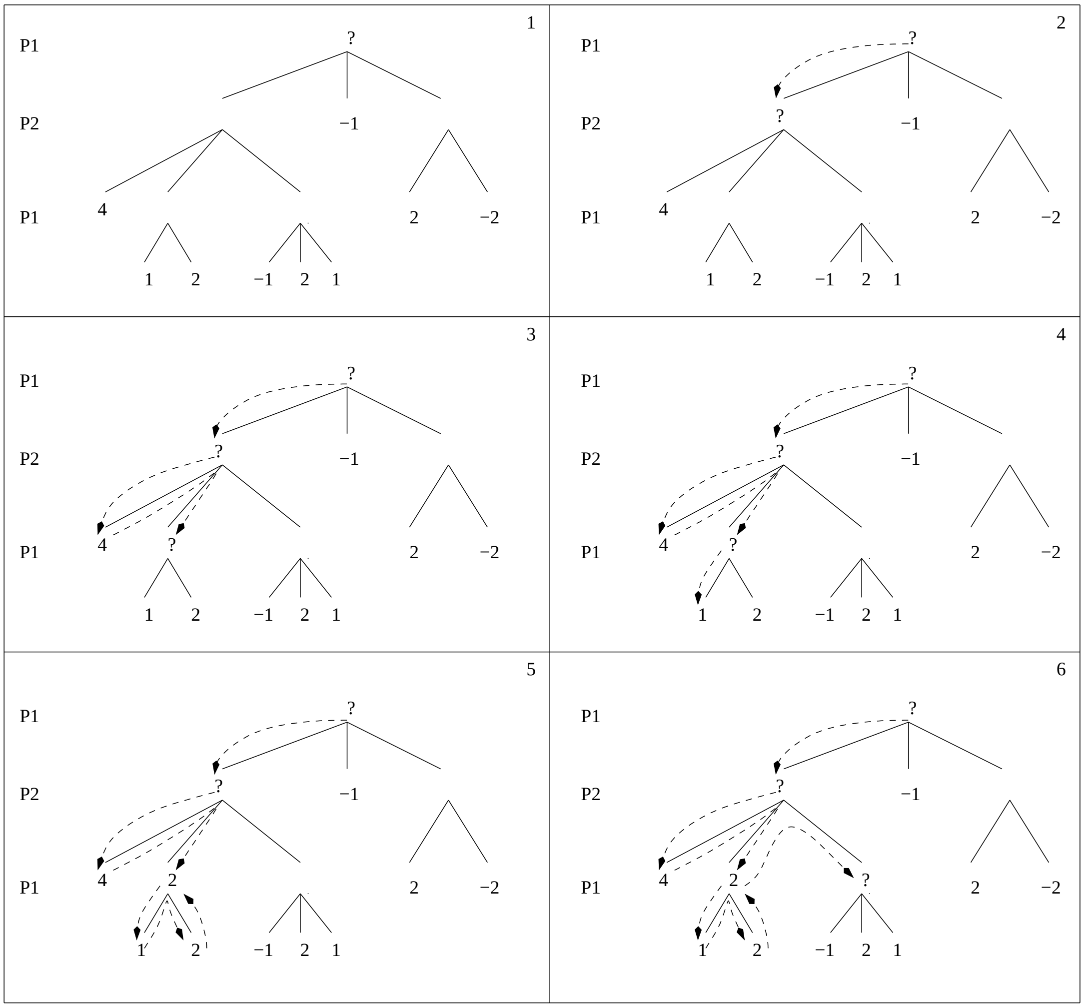


Figure 25: A depth-first algorithm to determine the value of the root

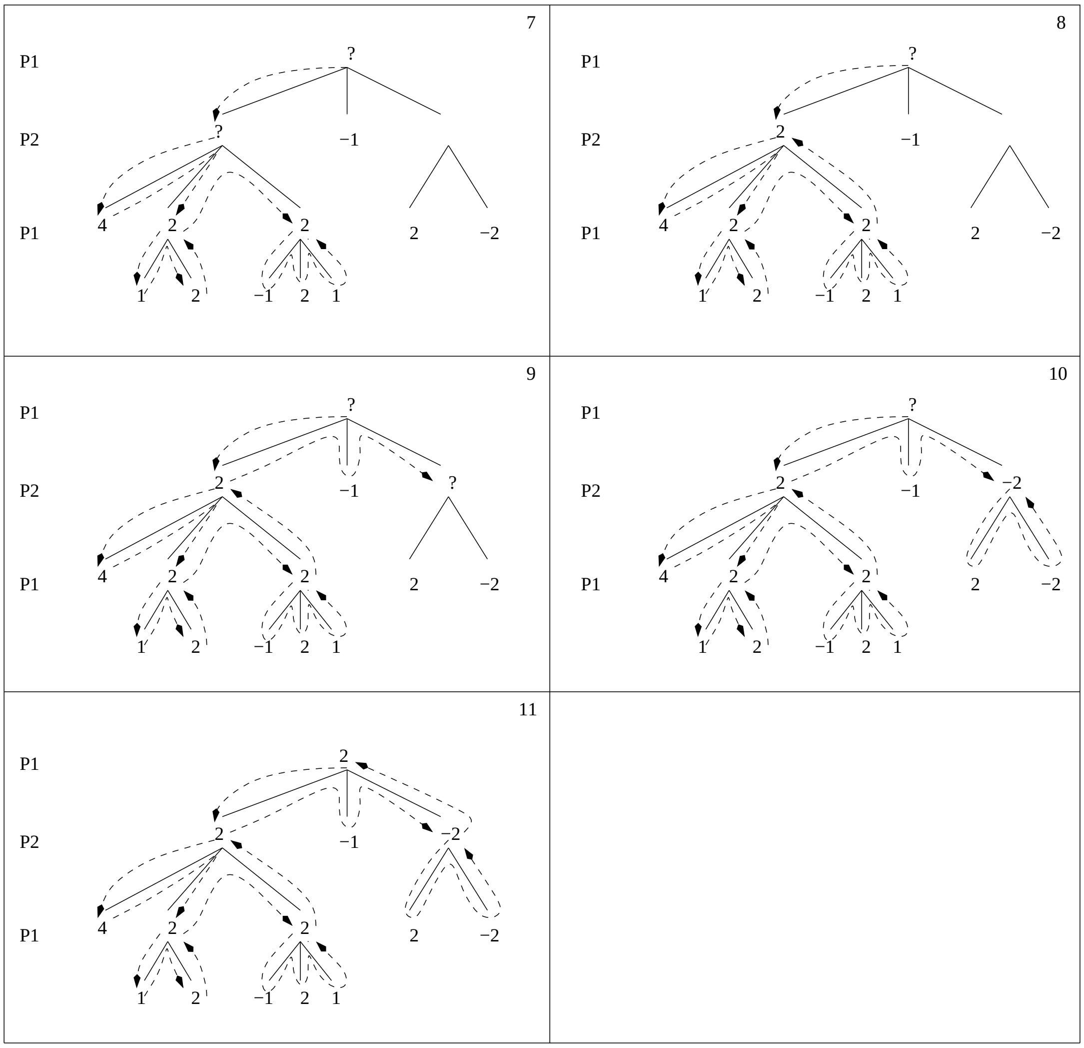



Figure 26: A depth-first algorithm to determine the value of the root

This procedure is called the **minimax algorithm**, because minima and maxima (over the values of successor positions) are taken as it proceeds. Figures 25 and 26 assume that from each node, we can only move to its children and its parent. If the chosen data structure knows about siblings then it is not necessary to return to the parent between visiting two siblings, but this speed is gained at some cost to the space needed to store the currently required part of the game tree. It is possible, along the way, to record the decision which leads to the identified value for each position for a player, thus defining a strategy. In the case of a 2-person zero-sum game these strategies define an equilibrium point for the game, and the value of the root for Player 1 calculated in this way is the value of a game as defined on page 33.

**Proposition 5.2** *In a 2-person zero-sum game of perfect information the values of the root for both players are the negative of each other, and the strategies for both players which guarantee this value give an equilibrium point for the game.*

**Proof.** Compare the definition of the value of a position with the proof of Proposition 2.3, and it becomes obvious that for the games under consideration the value for Player 1 of the root is the same as the value of the game. The strategy guaranteeing the pay-off of the value for Player 1 is given in the proof of Proposition 2.3 and follows precisely the minimax algorithm. The situation for Player 2 is similar. □

Hence we have identified a way of finding an equilibrium point for a 2-person zero-sum game of perfect information which does *not* require that all the strategies be identified first. However, if we are interested in *all* equilibrium points then this algorithm needs some adjustment (we need to record all the decisions that lead to the identified value). In the next section we will discuss ways of speeding up the minimax algorithm (so that we can deal with even larger games). If the game in question is not zero-sum, but still only has 2 players, then the minimax algorithm will do the following:

- For Player 1, it will find the largest expected pay-off that he can guarantee for himself.
- For Player 2, it will find the largest expected pay-off that she can guarantee for herself.
- If both players use the strategy found by the minimax algorithm they may not arrive at *either* of these pay-offs—both players might be better off than they thought!

**Exercise 24** (a) Carry out the minimax algorithm for each player in the game pictured in Figure 27. What are the values for the root of the game, which are the corresponding strategies, and what happens if they play these strategies against each other?

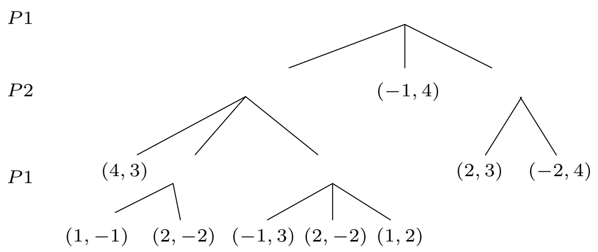



Figure 27: A non-zero sum game

(b) Apply the minimax algorithm to the game from Exercise 12 (b).

It has to be pointed out here that the minimax algorithm does *not* apply to games which are of imperfect information. This should come as no surprise: While all the positions in the same information set must have the same immediate moves, 'down the line' the game trees will become different, and in particular they will lead to different pay-offs. A typical example are card games: A player only knows his own hand, and his moves are the same initially no matter what cards the other players hold. But eventually the pay-off will depend on everybody's hand, so there cannot be a good decision procedure which makes do without that information (the game trees for card games are typically such that the potential difference in pay-offs for positions in the same information set is huge). Compare Simplified Poker, Section 2.6.

## 5.4 Alpha-beta pruning

The minimax algorithm allows us to find equilibrium points for games without having to generate all the strategies. At first sight, it appears to do the least possible amount of work: After all, by traversing the tree it visits each position only the minimal number of times required. But, in fact, it can be improved upon. In the small games typically drawn in this notes the difference may seem trifling, but with larger games (which can easily be held in a computer) the difference can be enormous. And, in fact, we will meet this technique again in the section on large games.

The idea behind alpha-beta pruning is to use the values calculated for the sub-games *so far* to make decision on whether or not it is worthwhile to look at other sub-games. Consider the situation in Figure 28, where some of the values for positions have already been calculated, and only part of the tree has been given (the local structure).

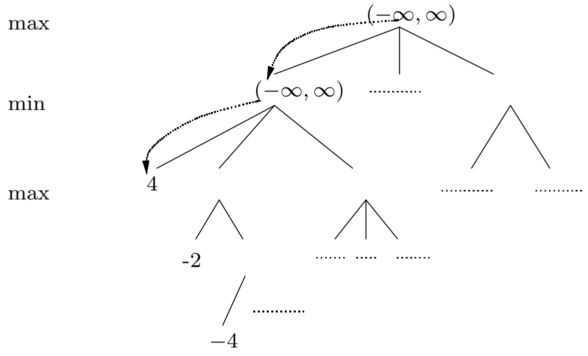



Figure 28: A minimax calculation

At the root, we have as yet no idea at all where the real value will lie. All we know is that it is a real number, so we know it is somewhere in $(-\infty, \infty)$. In order to find out its value we need to search the children of the root. As we pass down to them we pass on what we know already in terms of the possible values. So far, that is *nothing*.

Then we reach a tree in the leaf for which we can just read off the value. Once we know this we know that the parent of that leaf, being a min node, will have a value of *at most* the value of this child, 4. So when we return to this parent we update the available information. We descend to the next child, passing along that we are only interested in values in $(-\infty, 4]$, since higher values will not have any bearing on the parent node.

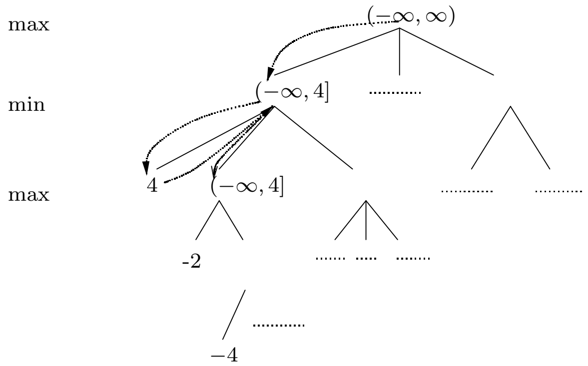



Figure 29: A minimax calculation—relevant ranges

Passing down to the first child we find another leaf. Its value is returned to the parent and it gives us a *lower bound* for that parent's value which is a max node.

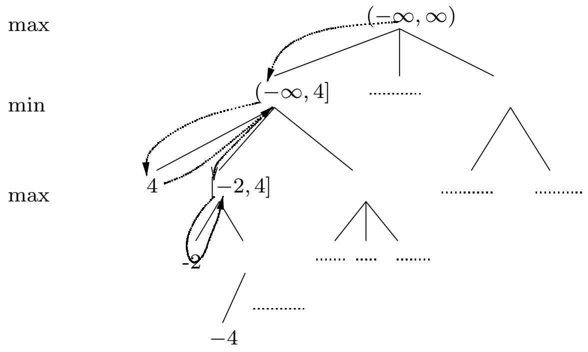



Figure 30: A minimax calculation—relevant ranges

We visit the sibling of the node with value $-2$, again passing along the range of values we are interested in. We find a value *outside* of that range with $-4$.

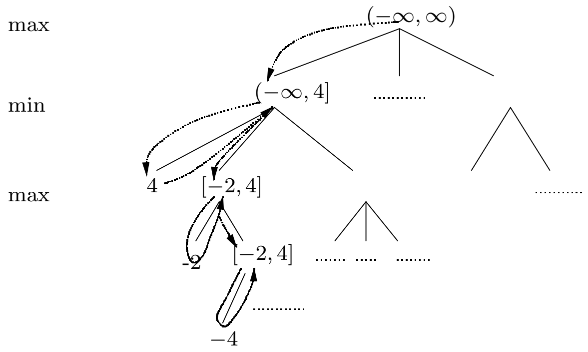



Figure 31: A minimax calculation—relevant ranges

This means that we *can stop searching this part of the tree*! The reason for this is that we now know that the value of the sibling of the node with value $-2$ is at most $-4$ since it is a min node, but since the parent of these two nodes is a max node, the max player will definitely not choose this branch, so there's no need to work out its precise value.

When we return to the parent of the node with value $-2$ we *know* that its value is $-2$ too, and we report that value upwards to *its* parent. There we note that we can update our range of values for this node—since it is a min node its value will be at most $-2$.

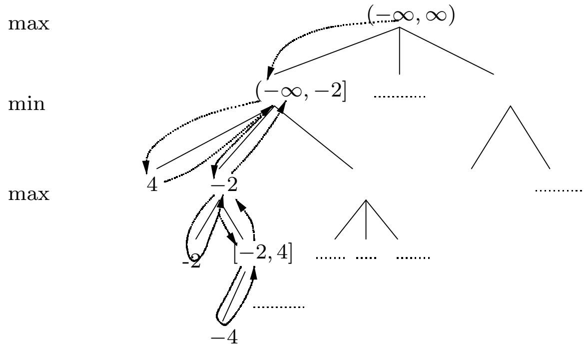



Figure 32: A minimax calculation—relevant ranges

We move to the next child of the first child of the root. Let us assume that we eventually establish a value of 1 as indicated in Figure 33. Then we can once more stop searching the tree further down there. The fact that 1 is not in our range of interest is equivalent with the fact that it will not be of relevance to the value of nodes higher above it. This establishes a value of $-2$ for the first child of the root.

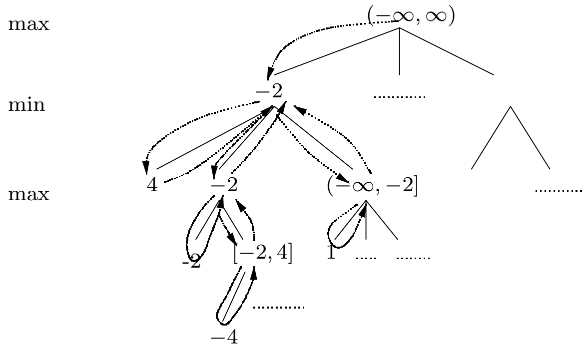



Figure 33: A minimax calculation—relevant ranges

Note that in order to find this value we *did not have to search the entire tree rooted there*! By passing down the relevant ranges as we recursively move through the tree in a depth-first manner, we may cut off the search at various points. That is what alpha-beta search is all about. We continue with our example.

Returning to the root for the first time we can finally give a smaller possible range. The value we will obtain for it is *at least* $-2$ since the max player can ensure that by moving to the first child. Let us assume that further search eventually returns a value of 1 for the second child of the root.

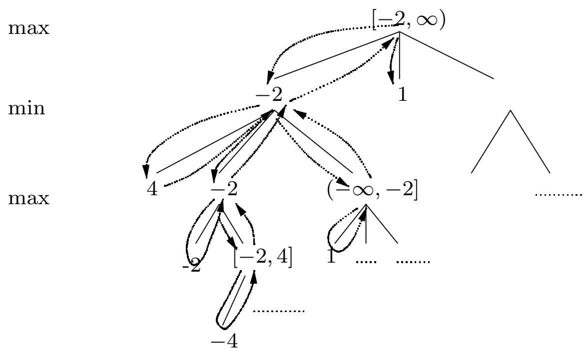



Figure 34: A minimax calculation—relevant ranges

Again this value is reported to the root and leads to an update of the relevant range—we now know that the value of the root is at least 1, which the max player can ensure by moving to the second child of the root.

We descend to the third child of the root, again passing along the relevant information.

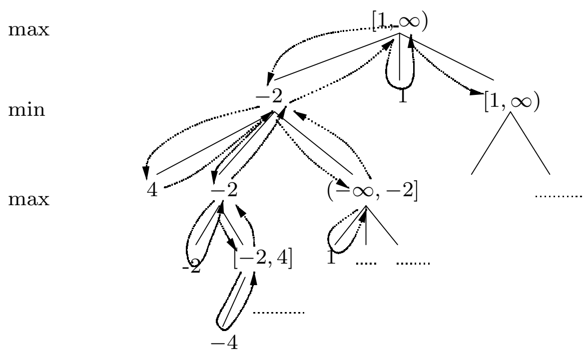



Figure 35: A minimax calculation—relevant ranges

If eventually we find a value of $-1$ for the first child as in Figure 36, it is again outside of the relevant range, and once more we may stop our search without traversing the tree further. We now know that the value of the root is 1.

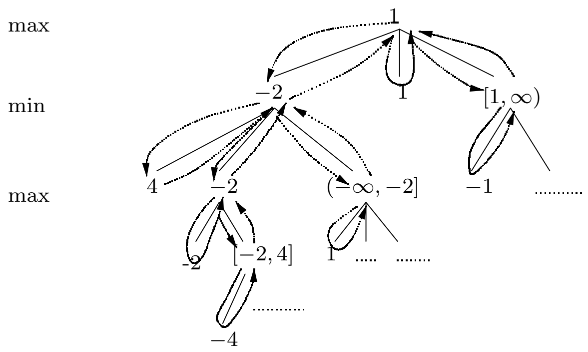



Figure 36: A minimax calculation—relevant ranges

Let us summarize how we did this. On each recursive call of the procedure that determines the value, two parameters (the 'alpha' and 'beta' in 'alpha-beta pruning', or 'alpha-beta search') are passed along to indicate the relevant range of values. A the start, these are set to $-\infty$ and $\infty$ respectively. So, at any stage we are at a particular node, and have a range from $\alpha$ to $\beta$ for its potential value. We get a value $v$ which is reported back from one of the children. We then do the following.

| Value $v$ is | Node is of type | |
|---|---|---|
| | max | min |
| below or equal to $\alpha$ | ignore $v$ and move on to next child | value of current node is irrelevant; return to parent |
| between $\alpha$ and $\beta$ | set $\alpha$ to $v$ and move on to next child | set $\beta$ to $v$ and move on to next child |
| above or equal to $\beta$ | value of current node is irrelevant; return to parent | ignore $v$ and move on to next child |

It should be clear where the 'pruning' in alpha-beta pruning comes from: By finding that we do not have to explore a particular subtree we effectively cut it off from the tree that we will have to traverse to find the value of the root. In other words, we 'prune' the tree under consideration by cutting off irrelevant parts.

We note that alpha-beta pruning is particularly effective if the *first* child investigated is the best move for the Player who makes it:

The final value of a node is the value of that particular child which corresponds to the best move for the Player whose turn it is. Hence by investigating that child first we guarantee that we will never get a *better* value coming from another child, which means that *all* the other children will allow some pruning!

When carrying out alpha-beta pruning it is therefore advantageous to try promising moves first. How to find 'promising' moves is a science in itself which clearly has to be adapted to the game under consideration. We will come back to this in the section on large games. When carrying out alpha-beta pruning by hand it seems tempting first to look at children with small subsequent subtrees, but when you program this algorithm you may have no way of telling which children those are. With some games, such as Nim and Chomp, it is clear that by reducing the number of available matches/remaining chocolate squares as far as possible in one move we limit the remaining subtree to be explored, but that is typically the exception rather than the rule. Clearly, when playing Chess, for example, it is normally irrelevant even which move leads to a small subtree unless the end is in sight.

**Exercise 25** Find a winning strategy in the following games using alpha-beta pruning. Try to do so without first creating the game tree, and make use of symmetry whenever you can. Which player can force a win? For the player who isn't so lucky, can you find a 'good' strategy that allows him to win if the other player makes mistakes? Can you find a way of generalizing your strategy to larger Chomp/Nim games?

(a) $2 \times 3$-Chomp;

(b) $(3, 3, 3)$-Nim.

### Summary of Section 5

- *Medium-sized games* are too large to determine all strategies, but small enough that a program might traverse the entire game tree depth-first in a reasonable amount of time.
- For such games we can calculate a *value* for each player which is the minimum pay-off the player can secure for himself.
- For 2-person zero-sum game this value agrees with the one introduced in Section 2.
- An algorithm which finds the value of the root is known as *minimax*, it performs a depth-first traversal of the tree.
- This can be improved upon by *alpha-beta pruning*, where information regarding relevant values is passed down in the search, so that parts of the tree can be discarded as irrelevant.

### Sources for this section

Most of the material in this section is well known, although this presentation is original. I did have a look, however, to see how other people present it, and found:

David Eppstein's notes on *Strategy and board game programming* at the University of California at Irvine, `http://www.ics.uci.edu/~eppstein/180a/`.

A.N. Walker's notes on his course in *Game Theory* (with a part on alpha-beta pruning) at the University of Nottingham, `http://www.maths.nott.ac.uk/personal/anw/G13GAM/`.

Pui Yee Chan, Hiu Yin Choi and Zhifeng Xiao's webpage on *Game trees and alpha beta search* at McGill University, `http://www.cs.mcgill.ca/~cs251/OldCourses/1997/topic11/`.

# 6 Large games

When we talk about large games we mean games which are too big to be *solved* using the methods outlined in Section 5. In other words, they are so big that even an implementation of the minimax algorithm with alpha-beta pruning cannot cover the entire game tree. For this section we will concentrate exclusively on 2-person zero-sum games of complete information. (There are programs playing games of incomplete information involving several players, for example for Bridge, but the methods involved in those would take us too far afield.)

## 6.1 Writing game-playing programs

We will first assemble the various tasks involved when writing game-playing programs and then discuss them in some detail. Examples of the kinds of games that we are interested here are Chess, Go, Othello (also known as Reversi), Hex, Go-Moku, Connect-4, Checkers, and Backgammon. Note, however, that we will not make many comments about chance elements.

As we have established in Section 5, for large games it is not really feasible to have a complete game tree stored in the computer. Instead, we generate it as required. This means that we have to implement a way of **representing the current position** and **generating the legal moves for a position.**

All successful programs for the kind of games we are interested in involve some version of a minimax algorithm with alpha-beta pruning. Since we are interested here in finding a good next move, for the remainder of this section we refer to this as **alpha-beta search**.

But how can we employ this algorithm in situations where it is not feasible to work one's way to the final positions of the game in order to calculate values? The answer is that the program will have to include some sort of mechanism to judge the value of a position *without searching the game tree below it.* We speak of having an **evaluation function** which assigns a score to a given position based on various criteria which depend on the game in question. This is a tough job: Imagine you had to look at a Chess board and say somehow how good a position this is! Typically there is *no one right answer* to the question—what is required of a good evaluation function is that it gives a *decent approximation*, and in particular that it can *compare one position to another*. To put it bluntly, an evaluation function is *guessing* the true value of a position, and it is the job of the programmer to make it the *most educated guess possible.* In particular if the evaluation function assigns a higher value to some position than to another then it ought to be the case that it is easier to win from the former than from the latter!

What the program then does is to apply some variant of alpha-beta search to find out which next move ultimately leads to a position with a high score. The deeper the program can search the game tree to do this, the better it will play. But searching the game tree to a high depth is very expensive, it will slow the program down. There is a direct pay-off between speed and depth of search. Hence many important techniques developed for game playing programs consist of deciding to which depth to search (this might be variable), which moves to search first, and how to improve the evaluation function. We discuss some of these techniques below, but it should be pointed out here that particular tricks often are dependent on the game under consideration. We also explore some of the traps such a program might fall into. To see a graphical presentation of the way a particular Chess programme searches among positions, go to `http://turbulence.org/spotlight/thinking/chess.html`.

## 6.2 Representing positions and moves

Speed is the one thing that is at the bottom of every decision made when designing a game playing program. Hence even the internal presentation of positions and moves is important: Speeding up the program's ability to examine moves makes the whole search process faster, and since the program will spend a lot of time doing that, getting this right can make a real difference. Game programmers quite often use clever encodings which allow bit-wise operations to make their programs faster.

Let us consider the example of Chess. Obviously the current position on the board will have to be represented in the program. But there also has to be a way of denoting whose turn it is, whether a player can still castle, and whether a capture *en passant* is possible. Worse, there are rules about *repeating previous positions* in Chess (which will lead to a draw), so the program has to have a way of remembering those! Clearly, whichever format is used to represent a board position, saving all those would be expensive, and searching them a nightmare.

Hence Chess programs typically use a large hash table to keep track of positions that have occurred in play. But we can make that even better: We can make further use of the hash table for positions to make sure we don't search the same position twice. As in many other games, in Chess the same position may arise from different sequences of moves. Clearly it would be inefficient to search again from the same position. We might also have previously searched it, if to a lower depth. Using a hash table we can keep track of which positions have been searched, to which depth, and what the best move discovered is. It is not worth hashing positions which have only been searched to a very low depth. A hash function frequently used consists of assigning to each pair, consisting of a piece and a field on the board, a large random number. The idea is that this number encodes the fact that the corresponding piece occupies the corresponding field. Then one sums up the appropriate numbers for the given position to obtain the hash key.[82]

Something else that should be easily possible in the program is to *undo* moves. This is not so much in case a (human) opponent wishes to cheat by reconsidering a move made, but because in the course of investigating possible moves the program has to pretend it is making moves, evaluate the resulting positions and then it has to undo them and explore other moves. That means that the program will have to remember where a given piece came from, and which, if any, piece was captured by the moves.

A fairly obvious presentation of the game board is as an $8 \times 8$ *array*, with each element of the array containing the code for one (or none) of the pieces. To generate valid moves, a program then has to loop over the array to pick up one piece after the other. The moves of knights are easy in the sense that all it takes for a given move to be valid is that the field where the piece ends is not already occupied by a figure of the same colour—that's just one look-up operation. For a king, not only does the field it goes to have to be vacant of own pieces, it also must not be a field any of the enemy pieces may go to in one move, so the program also has to figure out the valid moves for the opponent. For the other pieces, rook, bishop, queen and pawn, the program has to make sure that all the fields *on the way* to the new one have to be empty, generating many more look-up operations.[83] Finally the program has to ensure that the move would end on a valid field and not go beyond the borders.

[82] Some sort of checksum process is then applied to make sure later whether one has found 'the right' position upon look-up in the hash table.

[83] At least the program should be clever enough to remember fields previously looked up when trying to move a bishop, rook or queen one field beyond the one previously tried.

When a move has been made the position is updated by changing the entries in source and target fields. Even with a simple figure like a pawn, four moves have to be taken care of: move one field forward, move two fields forward, capture on the left and capture on the right (including the possibility of capture *en passant*).

An alternative to this presentation is to give each square of the board a number (a single byte), where the high four bits decode the row and the low four bits the column, leading to a table like this:

| | | a | b | c | d | e | f | g | h | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | low bits |
| 8 | 0111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | |
| 7 | 0110 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | |
| 6 | 0101 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | |
| 5 | 0100 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | |
| 4 | 0011 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | |
| 3 | 0010 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | |
| 2 | 0001 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | |
| 1 | 0000 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
| | high bits | | | | | | | | | |

To move one field to the left or right, just subtract or add one. To move up a row, add 16, to move down a row, subtract 16. The whole board is then represented as an array with 128 entries, only 64 of which correspond to actual fields on the Chess board. At first sight, this is puzzling—why use this particular presentation?[84] The answer is that checking whether a number describes a valid field is very easy: It does if the number $i$ satisfies $i\&0x88 == 0$ (where & is a bitwise operation and $0x88$ is the hexadecimal representation of the number 136). It is the reason why this presentation is sometimes referred to as $0x88$. It provides a sufficient speed-up over our first one that it is implemented in a number of Chess playing programs.

Another popular representation uses *bitboards*. Rather than use an array where entries stand for a square on the board and hold the type of piece on that square, the approach here is, for each piece, to hold a presentation of the board indicating where to find such a piece.

The advantage of this is that, given a particular piece, for each square of the board we require only one bit to state whether or not that piece can be found on that square. That means that for every piece we can store the required information in a 64-bit word (or two 32-bit words). Then many operations can be carried out using bit-wise Boolean operations, speeding them up considerably. Let us consider an example.

In the board position given in Figure 37, the following bitboard describes the position of the white pawns.

$$\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

[84] My source claims that performing a look-up operation in a one-dimensional array is a tad faster than in a two-dimensional one, but I'm not sure about that.

Figure 37: A Chess position

The bitboard representing all the fields occupied by black pieces is then merely the bit-wise 'or' of all the bitboards for the black pieces. Bit-wise operations are extremely fast, which can be used to good effect here. Similarly we can compute a bitboard for all the occupied positions, or just those occupied by a white piece. A move of a piece by one row consists of shifting the corresponding bitboard by 8. If a bitboard of empty fields is required, a bit-wise negation of the bitboard of occupied fields will suffice—this allows a quick test of whether the intended move is valid. If required, *all* the legal moves of pawns by one field can be stored in a bitboard, and similarly for all legal moves of pawns by two fields (carrying out a bit-wise 'and' operation with the board which contains '1's on the fourth row and '0's everywhere else makes sure only the pawns which are allowed to move two fields will be considered—and constant bitboards can be prepared at compile time to be available in a library). Pawn captures are equally quick to calculate (shifting the bitboard by 7 or 9 and bit-wise 'and'ing it with the bitboard for pieces of the opposite colour).

The code required when using bitboards is more complicated than that for using arrays. However, it has the following advantages:

- bit-wise operations are fast;
- bitboards required more than once only have to be computed once (compare that with checking whether a field is occupied in the array representation);
- several moves can be generated at the same time.

A disadvantages is that it is more complicated to turn a bitboard of possible moves into a list of such moves. Many tricks are used to make such operations (for example finding the non-zero bits in a bitboard) fast.

Undoing moves should be done by having a stack of moves made with sufficient information to undo them—using a stack of positions is a lot slower.

In many ways, generating moves (and representing the board internally) is the easiest of the tasks we have identified. Once it has been implemented, it can typically be left alone, and no changes will be required down the line when sufficient attention has been paid in the design process. However, some forms of alpha-beta search ideally want the potentially 'best' moves to be generated first—implementing this requires a lot more thought!

## 6.3 Evaluation functions

Whereas we are mostly concerned with speed when it comes to the internal representation of moves and position, finding a good evaluation function is a matter of implementing *knowledge* about the game in question.

The basic idea for this is fairly simple. Since we cannot expect to work our way through the entire game tree for Chess, or Go, we have to have a way of turning positions into values *without further searching the game tree*. That is the purpose of the evaluation function. It provides us with a provisional value for a position, which by force has to be crude. After all, it is a static process which does not take into account how the game might develop from a given position. Instead, it is an attempt to assign a value by merely looking at *what is currently on the board*. If this part goes wrong, the alpha-beta search will decide on making moves which may not lead to good positions, which can then be exploited by the opponent. There is no way of guaranteeing one has 'the right' evaluation function (such a thing certainly does not have to be unique), and a big part of writing game playing programs consists of watching the program play and fine-tuning the evaluation function accordingly. Apart from a few common sense ideas, evaluation functions are therefore mostly based on *heuristics*.

There is one aspect regarding evaluation functions which concerns *speed*: Evaluating a position can be quite a complicated process, with various aspects of the position requiring scoring. Therefore calculating such an estimated value for each position separately will inevitably repeat some of the work done previously, and hence be fairly slow.

One might think then that it would be useful if only parts of the evaluation would have to be recalculated for every move. One might hope, for example, that the evaluation function $e$ applied to a position $p$ can be expressed as a sum of the components $e_s$, where $s$ ranges over the various pieces, and $e_s$ is dependent on the current placement of $s$, that is

$$e(p) = e_{s_1}(s_1\text{'s place in } p) + \cdots + e_{s_n}(s_n\text{'s place in } p),$$

where $s_1, \ldots, s_n$ are the pieces involved. This can be adjusted usefully by using different *weights* for the different $e_{s_i}$.

Then we can calculate the estimated value of a new position reached when making a particular move by subtracting the piece's contribution in the old position and adding the corresponding component in its new position. Or we can score a move of a piece $s$ by measuring the improvement made in this way:

$$\mathrm{s}core(move) = e_s(s\text{'s new field}) - e_s(s\text{'s old field}).$$

However, for a game like Chess that does not lead to very good results as the desirability of a position also depends on the interaction of the pieces. One might, for example, score a pawn's position more highly if it is on a field protecting a castled king.

Another problem with this kind of evaluation function is that it does not take into account, for example, whether the piece in question is protected against capture by another piece, or whether in case of a capture a recapture would make the exchange worthwhile. An alternative to insisting on the evaluation function being capable of judging all this is instead to decide to search the game tree below such a critical position. Then the various available moves can be checked out more easily than with a strictly static technique.

Here are some aspects that might be relevant to evaluating a position. Just how important they are will vary from game to game. To judge a position it is typically important to do these evaluations for *both* players—having many pieces on the board does not give White any advantage if Black is about to checkmate him!

- **Material.** In Chess, that would be the number of pieces, where each piece gets its own value, in Go, it would be a count of pieces on the board, and similarly in, say, Othello. This is not equally useful in all games, however: In Othello, for example, it is not really the number of pieces in one's own colour that is important, but whether one holds specific fields, for example corner positions. Quite often the player with the better position will have *fewer* pieces on the board. There are other games where the number of pieces may be irrelevant.

- **Space.** In some games it is possible to divide the board into areas of influence, where a given player controls a number of fields. This is particularly relevant for Go. In Chess, one can count the number of fields threatened by one player for this purpose, and in Othello the number of pieces which cannot be taken by the opponent (a connected group of pieces surrounding a corner). One could just calculate the size of these regions, or attach some sort of weight to them if not all fields are equally important.

- **Mobility.** Having many different moves available can be an advantage in a game, Othello being a particular example. For Chess there is some doubt as to whether or not this is a useful measure—some people have tried and discarded it while others have retained the principle.

- **Tempo.** In games such as Go there is a question of which player has the *initiative*, that is the ability to make moves which advance his own agenda (as opposed to having to make defensive moves whose main purpose is damage limitation). Often having the initiative means that in reply, the other player has to make a defensive move to avoid worse, leaving the initiative with the original player. In other games, some sort of parity argument works: there are positions which lead to a win for the player whose turn it is (or sometimes for the player whose turn it is not), and that often merely depends on numbers easy to evaluate (in Nim and Connect-4 these arise quite often).

- **Threats.** Can one of the players capture (or threaten to capture) a piece? In Connect-4, or Go-Moku, does one of the players have a number of pieces lined up already? In Othello, is a player threatening to take a corner?

- **Shape.** This is really about various pieces on the board relating to each other. In Chess, for example, a line of pawns advancing is much stronger than, say, pawns sharing a column. In Go, shape is about 'territory to be'—a few well-placed stones outline a territory which the player can defend when threatened.[85]Judging shape can be very

[85]This is an over-simplification, really, but then Go is the game which to date most stubbornly defies programmers.

difficult, and typically shape is formed by a number of moves being made, where every one such move improves the position only incrementally, but where the resulting position can be a lot stronger. Shape is also typically a *long-term target*. An evaluation function partially based on shape will have to be based on something other than the simple addition of piece-based evaluation functions we discussed above.

- **Known Patterns.** In many games there are patterns which come up over and over again. This is particularly true for Go, where there are many libraries of sequences of moves concerning a small area (such a sequence is known as a *joseki*, where players following such an established line can maintain balance). In Chess, a bishop capturing a pawn on the border is often trapped. In Othello, it is sometimes advantageous to sacrifice one of the corners if one can then force ownership of another corner. It might be worthwhile to program such things explicitly in order to avoid making a bad move, or to follow the moves from a library if certain constellations are reached. What is typically difficult is to reliably recognize positions where such patterns should be applied, and to adjust the moves identified to the current situation.

The above criteria result in a variety of components that might make up the actual evaluation function. Typically these components are weighted and then added up, where the weights are determined based on heuristics. The reason for this is that summation is a simple process for combining numbers, which is fairly fast. There certainly is a question of whether one could not assign probabilities (for the other player to choose various moves, for example) to help with defining weights, but game programmers typically do not use such an analysis.

Here are a few issues that can be used to fine-tune an evaluation function.

- **Deducing constraints.** In games such as Chess, every piece is given a material value. Clearly a rook, say, is more powerful than a pawn, and the material value should reflect that. By analysing typical games, it can be possible to deduce constraints that these values should satisfy. Chess players know, for example, that it is usually advantageous to exchange a rook for a two pawns and a bishop, or two pawns and a knight, but a disadvantage if there is only one pawn involved. Hence the weight of a rook should be below that of two pawns and a bishop, but above that of one pawn and a bishop. That drastically reduces the numbers one might have to try.

- **Hand tweaking.** This is what happens most often in practice. Programmers watch their implementation play and then try to judge which parameters should be changed, and how. They perform the change and watch again. This produces reasonable results fairly quickly, but requires that the programmer knows enough about the game to analyse what is going wrong.

- **Optimization techniques.** Rather than use human judgement to tweak any parameters involved, one can use general optimization techniques. One example for these is 'hill climbing': Small changes are made to the parameters, and changes are only kept if they improve the performance. This requires some sort of measure to judge performance, for example the percentage of won games against some opponent. This tends to be slow and risks being stuck in positions where each small change makes the performance worse, but where a big change might bring huge gains (such situations are known as 'local optima'). This algorithm can be modified by randomly sticking with some changes which do not improve performance in the hope of avoiding this problem. The 'randomness' is controlled by some probabilities which typically should start out fairly high and then become smaller as a good value is approached. This adjusted method is slower than the original, but can get good values.

- **Learning.** When the first game playing programs were written it was envisaged that machine-based learning would be the most important aspect in developing them. This faith in Artificial Intelligence has not proved appropriate in practice. All world-class game-playing programs use other principles foremost. Examples of approaches based on learning involve genetic algorithms and neural networks. Both are in practice rather slow methods, and their main advantage is that they do not require much 'human intelligence' in the form of knowledge relevant to the game in question. The reason they can be very slow is that the number of test games required is typically very high (commercial game programmers who have worked with these approaches tried about 3000 matches to allow the program to learn about the game, and that was not sufficient to perform better than hand tweaking). Another problem is that if the program plays against an opponent that is too good it will lose all the time and never start learning.

Almost all these methods require some sort of measure for the performance of the evaluation function which results from a particular choice of parameters. One way of doing so is to run the program on a large suit of test positions which come, for example, from high-quality human games, and see whether the program can follow the winner's actions. (This is actually useful to just test whether one's program makes sense at all, for example by trying it on 'checkmate in two' kind of positions.) This method is typically combined with letting the program play a large number of matches against a known opponent, such as another program, or even a version of itself which has different weights, so that the two can be compared to each other. The problem with the latter is that playing what is, for the most part, the same program against itself will often lead to the same lines being explored over and over. To avoid this one might want to start the program(s) from positions a few moves into a game.

## 6.4 Alpha-beta search

As outlined above, a game playing program will apply the minimax algorithm making use of alpha-beta pruning. Rather than explore the game tree all the way down to the final positions it will stop at a pre-programmed depth. It will there use the value given by the evaluation function as an estimate for the real value, and otherwise use the algorithm as described in Section 5.4. You can find a nice illustration of the number of moves a program might search by going to `http://turbulence.org/spotlight/thinking/chess.html` (you must have Java enabled in your browser for this to work).

**Iterative deepening.** One of the problems with searching to a pre-defined depth is that time constraints may mean that not all moves can be explored. Also when applying alpha-beta search, the *order* in which moves are searched becomes vital—the better the first move, the more pruning can occur. That is why many programs first carry out shallow searches, deepening the level one by one. This sounds inefficient, but shallow searches are quite cheap when it comes to time spent on them. Compared with an exhaustive search to a higher depth, it does not really amount to much. This technique is often combined with others which make use of the information gained from the shallow search. But, if nothing else, we can use this information to decide the order in which we consider moves in the alpha-beta search, and it ensures that we've got a decent move to make if we should run out of time while searching to a greater depth. And if we use a hash table as outlined above to keep track of positions already searched then any position we encounter on the board will already have been searched to some depth before we made our previous move, so that sort of information should already be on hand, so that we don't have to start from scratch.

**Modified alpha-beta search.** When we do ordinary alpha-beta search as described in Section 5.4 we have no preconceived idea what the value of the root of the tree might be. As the search reports back a value for the child of the current position we get

- successively increasing lower bounds for the value if the current node is a max node, this value is usually called $\alpha$ (the 'alpha' from 'alpha-beta');
- successively decreasing upper bounds for the value if the current node is a min node, this value is typically called $\beta$ the 'beta' from 'alpha-beta').

As we descend into the tree we keep track of the current values of $\alpha$ and $\beta$ by passing them down and updating them as appropriate.

- If the current node is a max node we only consider moves which lead to a value of at least $\alpha$, because we know that we have found a move guaranteeing it, and thus are only interested in finding better ones. If we find a move with a better value we adjust $\alpha$ accordingly.

  If we find a value of above $\beta$ then we have discovered a part of the tree that is irrelevant for our search, and we return to the parent without adjusting $\alpha$ or $\beta$.

- If the current node is a min node we only consider moves which lead to a value of at most $\beta$, because we know that we have found a move limiting us to this, and thus are only interested in finding better moves from the opposing player's point of view. If we find a move with a lower value we adjust $\beta$ accordingly.

  If we find a value of below $\alpha$ we know that we have found a value which is irrelevant for our search, and we return to the parent without adjusting $\alpha$ or $\beta$.

It is worth mentioning at least that there is no need to program an 'alpha-beta for max nodes' and an 'alpha-beta for min nodes': By using the negative of existing values and exchanging $\alpha$ and $\beta$ when moving down one level we can ensure that the same code works for both kinds of nodes.

Iteratively deepening search provides us with a provisional value, say $v$, for a position we want to search to a higher depth now. One can then *pretend* that one already has an upper and a lower bound for the possible score. We thus use a range from $\alpha$ to $\beta$ with

$$\alpha \leq v \leq \beta$$

to achieve further pruning as follows. Carry out the alpha-beta search algorithm to the required depth, but on a max node

- only consider moves which lead to a value at least $\alpha$ (this allows more pruning)—this is as before, only then $\alpha$ was a value we could guarantee as a minimum;
- if you find a value $w$ above $\beta$, stop the search and report $w$ back.

On a min node

- only consider moves which lead to a value of at most $\beta$ (again, this allows more pruning)—again this is as before, only then $\beta$ was a value we could guarantee as being the maximum achievable;
- if you find a value $w$ below $\alpha$, stop the search and report $w$ back.

Whereas before we could be sure that values above $\beta$ (respective below $\alpha$) resulted from descending into irrelevant parts of the tree this is no longer true with our guessed parameters, so we have to keep track of this.

The following cases may then arise

- The search returns a value in the given range from $\alpha$ to $\beta$. This will be the correct value.
- The search returns a value $w$ larger than $\beta$. That means that our preliminary value was too pessimistic. We have to adjust our preliminary value to $w$, and might consider allowing a larger range. This is known as 'failing high'.
- The search returns a value $w$ below $\alpha$. That means that our preliminary value was overly optimistic. Again we have to start over with the adjusted preliminary value $w$ in the place of $v$, and again we may want to allow a larger range. This is known as 'failing low'.

This technique is known as 'aspiration search'. In the very best case (where the best move is explored first, and the considered range always contains the correct value) the total size of the tree searched is reduced to $(\sqrt{b})^d$, where $b$ is the branching factor of the tree and $d$ is the depth. That means that using this algorithm, one can search *twice* as deeply in the same time (at least in the best case). This explains why such variants of alpha-beta pruning are employed in almost all game playing programs.

When combining this alpha-beta search algorithm with the hashing of positions as described on page 112 one has to be careful to store enough information for the hash table to be really useful.[86] It is typically a good idea to store the best approximation of a value so far, together with upper (alpha) and lower (beta) bounds, which may be useful when that position is revisited.

**Move ordering.** As stated above, in order for the alpha-beta search algorithm to perform at its best, that is to prune as often as possible, it is vital that the good moves are explored first. Of course, the whole point of alpha-beta search is to find good moves, but we can still use some of the clues we gain along the way to speed the process up as far as possible. For one, if we have searched a position before, even at a lower depth, we have some idea of which moves lead to good positions. (This kind of information can come either from a hash table or from employing an iteratively deepening search.) Secondly, we may have some ideas about which moves are typically good for the game in question (for example capturing moves in Chess), and lastly we may have found a good move in a similar position (a sibling in the game tree) which may still be valid. Using these criteria, one should sort the available moves by expected quality, and then do the search in that order. And quite often (when pruning can be applied) it is good enough to just order the first few moves because the others may never be explored. Hence it makes sense to apply a sorting algorithm like SelectionSort or HeapSort which deliver the sorted items one by one. An obvious choice for a move to search first are known as 'killer moves'—moves which literally end the game. In Chess these are captures of big pieces (in particular by small ones), checks and promotions.

[86] In fact, one of my sources claims that the proper interaction between the two is not easy to achieve, and many bugs occur in this area.

When a good such sorting principle is in place and the best move is fairly often explored first, it can pay off to *reduce* the range for alpha-beta search on siblings of this expected best move. If this search fails, a normal search can still be applied. Since rather more positions are pruned when doing this, it can speed up the program considerably. This technique is known as 'PVS', short for 'principal variation search', because everything is compared against this principal variation (the first move searched).

**Winning positions which don't lead to wins.** One of the problems with alpha-beta search is a situation which seems almost paradox. When programmed naively, some winning positions may not lead to a win! The reason for this is not that 'the algorithm is wrong' but that it may need some encouragement to *force progress*. The reason for that is this: Say we give each winning position (that is, one we know we can win from) a value of 1000. When looking for the next move to make this will ensure that from a winning position we will always move to another winning position. That sounds quite good, but it's not good enough: It does not ensure that the move we have made leads to an actual win. Figure 38 gives a situation where this idea might fail.

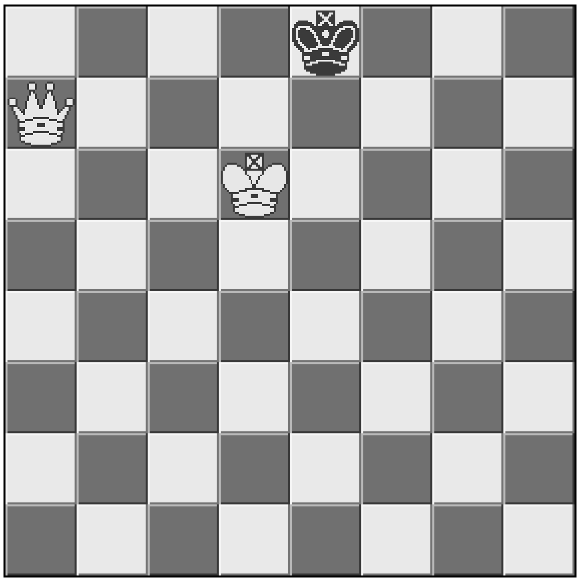


Figure 38: Another Chess position

If White moves the king to $e6$ (one field to the right) then he is still in a winning position, with Black's only valid moves being to $d8$ and $f8$. Let's assume Black moves to $d8$. Then moving the king back to $d6$ again gives White a winning position. But if Black now moves back to $e8$, we are back where we started and our program might go into a loop. This will lead to a draw since there are rules about repeating the same position.

We can avoid falling into this trap by assigning a slightly adjusted value to a winning position, say 1000 minus the number of moves required to get to the win. Then alpha-beta search will indeed ensure that the program wins when it finds itself in a winning position.[87]

[87] Clearly there are some programs, such as Othello, where this is not required since this game ends after at most 60 moves.

**The horizon effect.** One way of formulating the problem explained in the previous item is that while a win can be forced, it stays forever below the horizon (and the program is happy with moving to a position from where it can (still) win, because we have not told it otherwise). There is also the opposite effect.

A program as we have defined it so far is quite happy with bad moves *as long as the consequences lie below the current search horizon.* Clearly, there is no way the program can know about these consequences. We do get an effect when something 'bad' is about to happen, say the opponent might capture one of the program's pieces. Then the program will often try to avoid this capture (which might be inevitable) and thus will play a sequence of pointless moves to keep the event so long that it moves below the horizon, and so effectively can't be seen by the program.

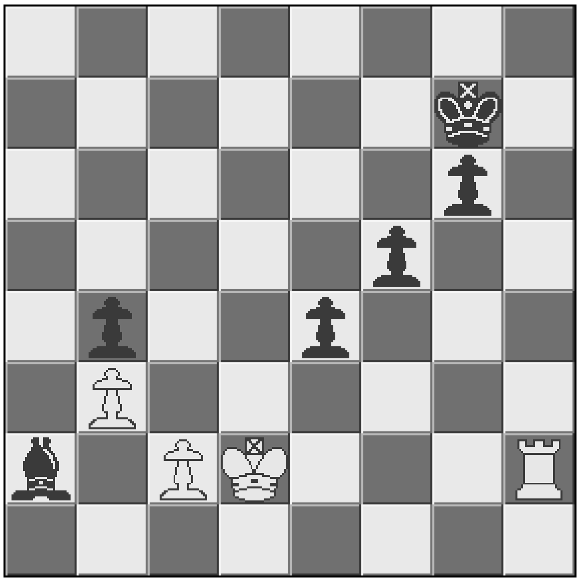


Figure 39: Yet another Chess position

In the situation depicted in Figure 39 the black bishop is trapped by the white pawns. Eventually it will be taken, for example by the white rook moving from h2 to h1, a1 and finally a2. This would therefore occur after six moves. Assume the program playing Black searches up to six moves. An accepted 'good' line for Black in this situation is to trade off the bishop against a pawn by capturing the pawn on b3 and being taken by the pawn on c2. The three connected pawns then might be good enough to win or draw against the rook. A program searching six moves will, however, typically move the black pawn e4 forward to e3, checking the king, a move which requires White to move the king (possibly capturing the attacking pawn). That delays the taking of the bishop long enough so that the program can't see it anymore, and it thinks the bishop is safe. A program might thus throw away all its pawns to delay what is inevitable—setting itself up for a loss as a result.

There are several ways of trying to overcome this problem. One might try adding knowledge to the program so that it can detect when one of its pieces is trapped. This is typically rather difficult to achieve. Another way is to try and increase the overall depth of search in the hope that pushing back the horizon will make this sort of situation less likely. But probably the best tactics to employ is to instead selectively search deeper in situations like this, where a piece (like the pawn here) is being sacrificed. We discuss this as the next item.

**Selective extension.** Many game playing programs today do not search to a given depth no matter what the position is. Instead, they are selective about that and search to a greater depth whenever

- there is reason to believe that the current value for a position is inaccurate or
- when the current line of play is particularly important.

A popular choice is to look at the deepest positions to be evaluated (that is, after reaching the current search horizon) and extend all lines which start by moves which are likely to change the evaluation of the position considerably. In Chess such moves are capturing moves (and that is true for other games such as Checkers). This is also known as 'quiescent search'. Alternatives are to extend the depth of search whenever the line of play under consideration contains a capturing move (this would be good enough to work for the example in Figure 39), or maybe a check. Clearly this has to be used in a limited fashion or the tree searched might expand hugely, possibly even infinitely, so some sort of termination mechanism has to be implemented. But at least this avoids making moves towards a position where, say, the opponent can capture our queen but our evaluation function thought that our pieces were placed well enough to give us an advantage over the opponent!

Many programs (for example Deep Blue, see the subsequent section) apply an extended search to moves which they identify as the 'principal line', because they start with a move that is much better (on current knowledge) than its siblings, thus trying to ensure that when they choose this move no ugly surprises lurk below the horizon. There are also tricks when instead of increasing the depth of some lines, the depth is decreased on 'obviously bad' lines.

One amusing phenomenon in computers playing games is that every now and then they spot lines, due to carrying out an exhaustive search, which are so complicated that human opponents are unlikely to see them. One program, playing a Grandmaster, suddenly seemed to offer a rook for capture for no reason that the assembled experts could discern. After the game was over they made the machine go back to that position and asked it what would happen if it had made the move judged 'obviously better' by the audience. The machine pointed out an intricate mate which it was trying to avoid. Arguably, it would have been better off leaving the rook alone and just hoping that the opponent wouldn't see the mate!

Another phenomenon that comes up with machines playing games is that if there is a mistake in the program, for example it has a weakness if a particular line or position comes up, then this weakness can be explored over and over again, because most programs are incapable of learning. Many tournaments between various programs seemed to be more about who could discover whose built-in faults, rather than whose program genuinely played best!

## 6.5 The history of Chess programs

Of all the game playing programs those playing Chess have had the most time and man power invested in them, and as a result they are the most sophisticated. Just how sophisticated up-to-date programs currently are is described below. In the remainder of this section, when we talk about 'level of search', or the like, we typically count a move of the program followed by a move of the opponent as one level. Chess players speak of a 'ply' when they mean what we call a move—for them a move consists of a move by White followed by one by Black.

In 1950 Claude Shannon (probably known best for his contributions to information theory) described principles for a program that could play Chess. He suggested that each position should have a value (or score), to be calculated from the number of the various pieces (that is our 'material' criterion), each with an appropriate weight (so that a pawn is worth less than a bishop, say), their mobility and with special values for good 'pawn formations'. The program was then to search the game tree, and Shannon suggested two alternatives:

- The program might search to a given depth, the same everywhere; and he called that the 'fixed depth' method.
- The program might search to a variable depth depending on the 'type' of a position. Thus it should decide that if a move was 'obviously bad' there was no point in searching the game tree below it. There would have to be some notion of what he called 'stability' to decide at which step to stop. He called this the 'variable depth' method.

The program was then to apply a depth-first minimax algorithm to adjust the value of some given position. As discussed above it clearly is not realistic to wait for a program to calculate the *actual* value of some position, hence the need to fix a depth for the search in some way. His outline is still the basis for most game playing programs, although employing alpha-beta pruning is typically applied to increase speed. Shannon advertised this idea for one to find some application for computers, but also to gain insights into playing games, and thus into making intelligent decisions.

In 1951, Alan Turing[88] created the first algorithm for computer chess here in Manchester. Turing had worked at Bletchley Park during the war and where he was a central figure in the breaking of the German Enigma codes; computing machines were used there in order to try many different combinations quickly, thus helping with the decoding of messages. In the thirties he wrote a paper introducing the first formal notion of 'computability' with the help of *Turing machines* which were named after him. He originally came to Manchester to help build the first computer here, the 'Baby', but was actually a member of the Maths Department. His Chess algorithm was not very good, searching to a low depth and with a very crude evaluation function. It was meant to be carried out by hand, and he did, in fact, play games employing this algorithm—making Manchester the place where the first Chess playing program was designed and 'executed'.

[88]See A. Hodges, Alan Turing: The Enigma, Vintage, 1992, or visit `http://www.turing.org.uk/turing/`.

Soon after that the first 'real' Chess programs appeared, and in 1966 the first match between a Soviet and a US American program took place, the former winning 3 to 1. The rules for the match gave a timeout of 1 hour for every 20 moves, where each player was allowed to bring 'saved time' forward into the next 20 moves. That meant that searches had to be cut off, and typically a program would allow 3 minutes per move, that is, the same time for all moves (without trying to take into account how complicated the current situation was). The decision of which moves to explore first (and thus which moves might not be explored at all) was fairly random.

In 1974 the first world computer Chess championships took place, which were repeated every three years thereafter. By the late eighties, the following improvements had been made.

- In order to avoid exploring the same position twice programs employed hash tables to remember which positions had already been searched, and what value had been discovered for them. This technique is known as employing *transposition tables*. However, programs typically failed to update this value if in the course of the game a different one was found.

- Most programs had *opening libraries* so that at the beginning, programs would just follow 'approved opening lines' (and thus not make catastrophic moves early on).[89]

- Programs employed hash tables to store 'tricky positions' for future play, thus implementing some sort of 'learning'.

- Rather than doing a pure depth-first search for the next move, many programs switched to 'iteratively deepening search'. A shallow search is carried out for *all* possible moves, and the result might be used to decide in which order to explore these moves to the desired depth. No pruning should be carried out at this point, since moves which might look bad if one only looks one step ahead (such as sacrifices) might become very good moves if the game tree is explored to a higher depth. Many of the early programs fell into this trap of avoiding to lose pieces at all cost (since such moves were classified as 'obviously bad'), whereas they are an integral part of playing Chess.

- Instead of searching to a given depth for all positions attempts were made to make this more dynamic, depending on whether the current situation is 'tricky' or straightforward. In practice this is achieved by setting time-limits (low for 'simple' positions, high for 'tricky' ones), and then carrying out complete searches of iterative depths (first all moves on level 1, then all moves on level 2, and so on) until time runs out. Then the best move found during that time is made.

Some of the techniques learned over time were useful for other areas of computing, for example the 'iteratively deepening search' technique is successfully applied in automated theorem proving. In fact, one of the reasons for putting that much effort into writing Chess playing programs was that the lessons learned would be applicable to other situations where a space had to be searched.

[89] It was much later that *endgame libraries* were used for the first time. When both those are employed the really interesting part of a Chess program is how it does in the middle game—and that's also the most interesting part of a game between human players.

In the seventies, when Artificial Intelligence was assumed to be 'just around the corner' (at least by some people), Chess playing programs were taken to provide a prime example of what might be done using machine learning, and a lot of research went into that. The only contribution to Chess programming made by this approach was its use in solving various end games in the late seventies and early eighties. The real power of Chess programs consists in the speed with which they can search the game tree. While doing this cleverly with good evaluation functions requires some real knowledge, everything else is raw computing power!

The first 'serious' man-computer match occurred in 1978—man won. In the late eighties, AT&T developed the first 'chess circuitry' which lead to a program (called Deep Thought[90]) which dominated the computer Chess scene for years. As a result of its special circuits it could generate moves very quickly and so search more positions in a given time frame than other programs.

A further development in the eighties was the inclusion of entire opening databases into programs, thus making whole books of openings available to the computer. Similarly endgame databases were developed, and all five piece endgames were solved in that period. Again, the computer's raw power allowed the finding of solutions which had eluded people so far. As a result, the official Chess rules were changed—the previous version did not allow for some of the checkmates to be played out (due to a restriction in the number of moves the players were allowed to make without the capture of a piece or a 'check' occurring).

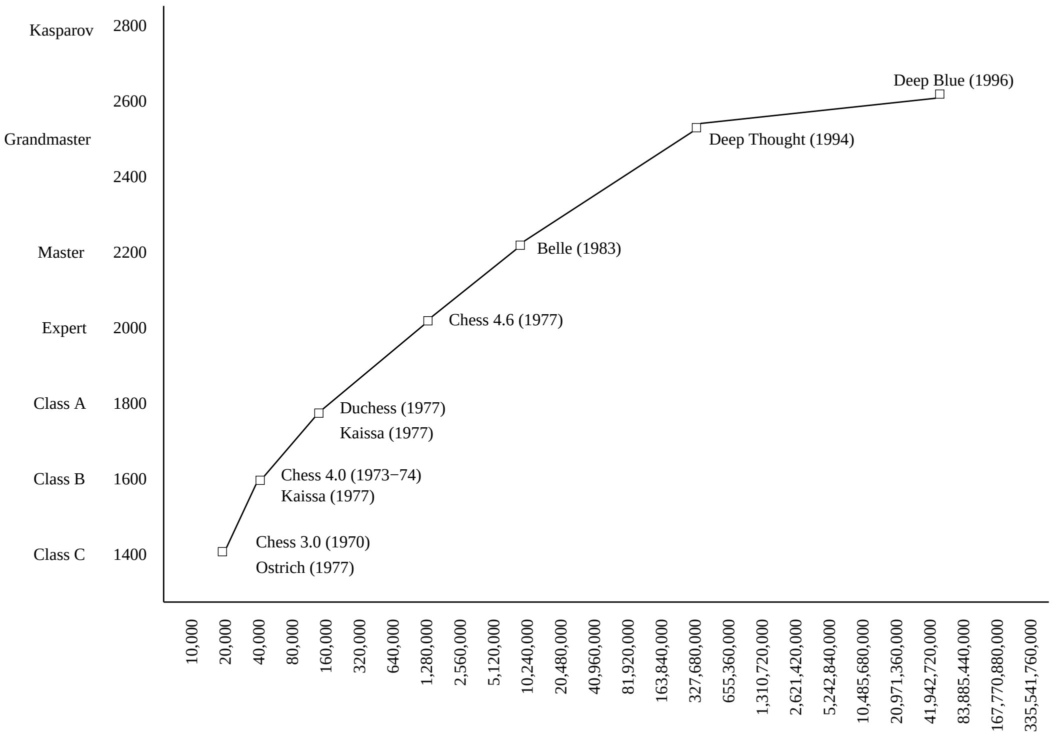



Figure 40: The level of play of programs

[90] Anybody want to guess more obscure references?

To illustrate the development of Chess playing programs, Figure 40[91] shows the number of positions examined in three minutes by a program *versus* its rating according to the Chess rating system. For a computer, this is a 'brute force' chart connecting playing ability and computing power. Note the logarithmic scale along the horizontal axis! It certainly isn't the case that humans look at anything like as many positions as machines. They have a much better way of weeding out 'obviously bad moves', and research has shown that very good players do recognize patterns of some form which allow them to only consider a few candidates for the next move to make. We do not currently have any abstract way of describing such patterns, and Chess players are not really aware of *how* they do what they do. Understanding this would certainly do a lot to advance a number of different subjects.

Another interesting question regarding Figure 40 is that of where exactly 'perfect play' would come in. Clearly machines have improved a lot over time, but how far off is it? And does the curve suggested really approach it, or does it have a lower limit?

Another issue worth examining is how much difference it makes to increase the search depth, that is the number of moves considered before making a decision. Figure 41 shows the results of playing the same program (Belle) against itself, using different depths to which alpha-beta search is carried out. The results are shown as 'won games out of 20', and are based on two experiments carried out by Thompson, the creator of the Belle program, in the late 70s.

| | 3 | 4 | 5 | 6 | 7 | 8 | rating |
|---|---|---|---|---|---|---|---|
| 3 | | 4 | | | | | 1091 |
| 4 | 16 | | 5.5 | | | | 1332 |
| 5 | | 14.5 | | 4.5 | | | 1500 |
| 6 | | | 15.5 | | 2.5 | | 1714 |
| 7 | | | | 17.5 | | 3.5 | 2052 |
| 8 | | | | | 16.5 | | 2320 |

| | 4 | 5 | 6 | 7 | 8 | 9 | rating |
|---|---|---|---|---|---|---|---|
| 4 | | 5 | .5 | 0 | 0 | 0 | 1235 |
| 5 | 15 | | 3.5 | 3 | .5 | 0 | 1570 |
| 6 | 19.5 | 16.5 | | 4 | 1.5 | 1.5 | 1826 |
| 7 | 20 | 17 | 16 | | 5 | 4 | 2031 |
| 8 | 20 | 19.5 | 18.5 | 15 | | 5.5 | 2208 |
| 9 | 20 | 20 | 18.5 | 16 | 14.5 | | 2328 |

Figure 41: Two experiments studying the effect of increasing depth

Figure 41 shows quite convincingly what a big difference searching three or four levels more can make—that of totally outclassing the opponent. The one constraining factor for searching to greater and greater depth is time, and computer programs have made great strides not only because a lot of time has been spent on making them better but because today's machines are so much faster and thus allow searching to greater depth. Of course, every bit of time gained by being 'clever' (for example, early on abandoning search on hopeless moves) can then be spent on going even deeper on promising moves.

[91] All tables and figures in the remainder of this section are taken from Newborn's book, see sources.

The other interesting observation to be made regarding Figure 41 is the relation between depth of search and the program's rating. While Figure 40 seems to suggest a linear connection between depth of search (logarithmic in number of positions searched means roughly linear in depth) and rating until about 2500, Thompson's experiments suggest that linearity only applies to level 2000. His data appears somewhat more convincing since no other improvements than depth of search have been made in his case, whereas Figure 40 compares very different programs with each other. One might tentatively state that the improvement beyond that which might be expected from adding depth to the search must result from other improvements made. After all, 25 years separate the early programs in Figure 40 from Deep Blue.

An alternative way of measuring improvement when increasing the search of depth is to compare how often the more thorough (and more expensive) search actually pays off. The following table attempts to measure that by looking at the number of times searching to a higher depth resulted in a different move being picked.

| level | percentage of moves picked different from predecessor | approximate rating |
|---:|:---:|:---:|
| 4 | 33.1 | 1300 |
| 5 | 33.1 | 1570 |
| 6 | 27.7 | 1796 |
| 7 | 29.5 | 2037 |
| 8 | 26.0 | 2249 |
| 9 | 22.6 | 2433 |
| 10 | 17.7 | 2577 |
| 11 | 18.1 | 2725 |

Hence different moves are indeed picked quite frequently when searching to a higher depth, which presumably explains the results in Figure 41. The table also shows that the return of this is diminishing with increasing depth.

The late eighties saw the first time that Grandmasters[92] were beaten by programs, and in tournaments rather than in display matches. The first program to be rated Grandmaster was Deep Thought. But playing against the world champion in 1989 it was defeated in only 41 moves. In 1993 it managed to beat the youngest Grandmaster ever, Judit Polgar (still a top twenty Chess player).

The main development from the late eighties onwards is the development of more specialized hardware to speed up the generation of moves, with considerable use of parallel machines. Programs that have been build to play at the highest level are therefore increasingly hardware dependent and cannot run on other machines. Computers participate increasingly in tournaments, in particular in the US (for example state championships). The following table gives some idea of the hardware and computing power thrown at this problem over time.

[92] Bent Larsen, beaten by Deep Thought.

| Name | Year | Description |
|---|---|---|
| Ostrich | 1981 | 5-processor Data General system |
| Ostrich | 1982 | 8-processor Data General system |
| Cray Blitz | 1983 | 2-processor Cray XMP |
| Cray Blitz | 1984 | 4-processor Cray XMP |
| Sun Phoenix | 1986 | Network of 20 VAXs and Suns |
| Chess Challenger | 1986 | 20 8086 microprocessors |
| Waycool[93] | 1986 | 64-processor N/Cube system |
| Waycool | 1988 | 256-processor N/Cube system |
| Deep Thought | 1989 | 3 2-processor VLSI chess circuits |
| Star Tech | 1993 | 512-processor Connection Machine |
| Star Socrates | 1995 | 1,824-processor Intel Paragon |
| Zugzwang | 1995 | 96-processor GC-Powerplus distributed system (based on the PowerPC) |
| Deep Blue | 1996 | 32-processor IBM RS/6000 SP with 6 VLSI chess circuits per processor |

Until the early nineties, writing Chess programs on this level was entirely an academic effort, and commercial programs available for sale typically played on a much weaker level.[94]

It was only then that IBM entered the scene and created Deep Blue, probably the best-known Chess program, based on Deep Thought. In 1996 a 6-game match was arranged between it and the reigning world champion, Gary Kasparov. Deep Blue won the first game, but lost the match 2 to 4. In 1997 a rematch occurred which was won by the machine 3.5 to 2.5. Kasparov made a mistake in the deciding match, leading to his loss of the series. You may therefore be surprised to find Deep Blue ranking below Kasparov in Figure 40. The reason for this is that Deep Blue was very much fine-tuned to play against this one opponent. It had whole books on lines which Kasparov liked to play, and others on lines which Kasparov was known to avoid (in the hope that he did not know them well enough to play that strongly when forced into them by Deep Blue). Arguably the human player learns from playing against the machine and can then exploit its weaknesses. However, Deep Blue had access to hundreds of games that Kasparov had played, whereas the creators of the program were very reluctant to let him have access to games it had played prior to the match.

In summary the history of Chess programs shows that currently, game programming is not really about mimicking the way human beings reason and make decisions. Instead it became a case study in applying the speed at which computers can carry out instructions to searching a given space. In particular it has shown us something about the relation between greater depth of search and reaching better results. As far as other games are concerned: In 1982 a program called IAGO was assessed as playing Othello (also known as Reversi) at world championship level, but didn't take part in any tournaments. In 1994 a program called Chinook became world Checkers champion, but the reigning world champion had to forfeit the match due to illness. Go playing programs currently are many levels below even good amateurs, let alone professionals. The people who created Chinook have now solved the game using the program: Either side can enforce a draw.

[93] This serves to show that this expression goes quite some way back!

[94] Presumably it would be harder to sell a program which requires its own hardware, and expensive such a thing would be too.

### Summary of Section 6

- Three tasks have to be solved when writing a game-playing program: Designing an internal board representation and generating valid moves, designing an evaluation function and implementing (some variant of) alpha-beta search.
- All considerations are overshadowed by the *need for speed*.
- Board representations should make the generation of moves, doing and undoing them fast.
- Evaluation functions require knowledge about the game in question. They are an attempt to assign a value to a board position from just what is on the board, without further descending into the game tree.
- Alpha-beta search is concerned with assigning a value to a position by searching the game tree below it and eventually applying the evaluation function. Searching to greater depth will result in a better program, so any gain in speed goes into searching to a greater depth. There are many tricks one might try to employ in order to concentrate on searching the relevant parts of the game tree; in particular ordering moves to search the most promising ones first.
- Most effort so far has gone into creating Chess-playing programs. They have profited from faster hardware, and many improvements have been made which are very Chess-specific: better heuristics, opening and endgame libraries, and the like.

### Sources for this section

The material in this section has been compiled from the following.

David Eppstein's notes on *Strategy and board game programming* at the University of California at Irvine, `http://www1.ics.uci.edu/~eppstein/180a/w99.html`.

A.N. Walker's notes for his course on *Game Theory* at the University of Nottingham, available at `http://www.maths.nott.ac.uk/personal/anw/G13GAM/`.

M. Newborn. **Kasparov versus Deep Blue: computer chess comes of age.** *Springer*, 1997.

The *Scientific American*'s account of the second match between Kasparov and Deep Blue at `http://www.sciam.com/explorations/042197chess/`.

IBM's account of the same match at `http://www.chess.ibm.com/`.

D. Levy and M. Newborn. **How Computers Play Chess.** *Computer Science Press*, 1991.

The Chinook webpage (about checkers) at `http://www.cs.ualberta.ca/~chinook/`. This includes a link to the paper on the solving checkers.

## Assessed Exercises

**Ass. Exercise 1** Alice and Bob play the following card game. Each pays one pound into the pot. Alice draws a card and looks at it without showing it to Bob.

She now declares whether she goes 'low' or 'high'. In the latter case she has to pay two more pounds into the pot.

Bob can now resign, or demand to see. In the latter case he must match Alice's contribution to the pot.

If Bob resigns, Alice gets the pot. If he demands to see then he gets the pot if Alice has a red card, otherwise she gets it.

(a) Draw a game tree for this game. You may assume that the probability of drawing a red card, and that of drawing a black card, are both 1/2. (2 marks)

(b) Describe all strategies for both players. (2 marks)

(c) Give the matrix form of the game. (2 marks)

(d) Find an equilibrium point and the value for the game. (2 marks)

**Ass. Exercise 2** Bob and Alice each pay one pound into the pot. They have a deck of 3 cards, consisting of a 2, a 3 and a 4. Alice draws a card, and then Bob draws a card. They both look at their card without showing it to the other player. Alice can either resign, in which case the pot goes to Bob, or she can bet and put another pound into the pot. In his turn Bob can now resign, in which case the pot goes to Alice, or he can bet, in which case he has to put a pound into the pot too. If they have both bet then they both turn over their card and the holder of the higher one gets the pot. Describe the strategies for both players in this game. Hint: You don't want to draw a full game tree for this! (2 marks)

**Ass. Exercise 3** Find as many equilibrium points as you can for the following games, and give their values.

(a) $\begin{vmatrix} 1 & 1 & 1 \\ 1 & 3 & -1 \\ 1 & 0 & 2 \end{vmatrix}$ (2 marks) (b) $\begin{vmatrix} -1 & 0 & 1 & 2 \\ 2 & 1 & 0 & -1 \\ 0 & 1 & 0 & 1 \end{vmatrix}$ (2 marks)

**Ass. Exercise 4** Prove that the AlwaysD strategy cannot be invaded in an indefinitely repeated Prisoner's Dilemma game. (2 marks)

**Ass. Exercise 5** In a population there are two kinds of individuals, Lions and Lambs. Whenever two individuals meet, 40 points are at stake. When two Lions meet they fight each other until one of them is seriously injured, and while the winner gets all the points the loser has to pay 120 points to get well again. If a Lion meets a Lamb then the Lion takes all the points without a contest. When two Lambs meet each other they debate endlessly until one side gives up. The loss of time means they are fined 10 points each. You may assume that the probability of winning against an individual of the same kind is 1/2.

(a) Give a matrix that describes the game and argue that a population consisting solely of Lions isn't stable, nor is one consisting solely of Lambs. (2 marks)

(b) Give an example for a population that is stable. (2 marks)