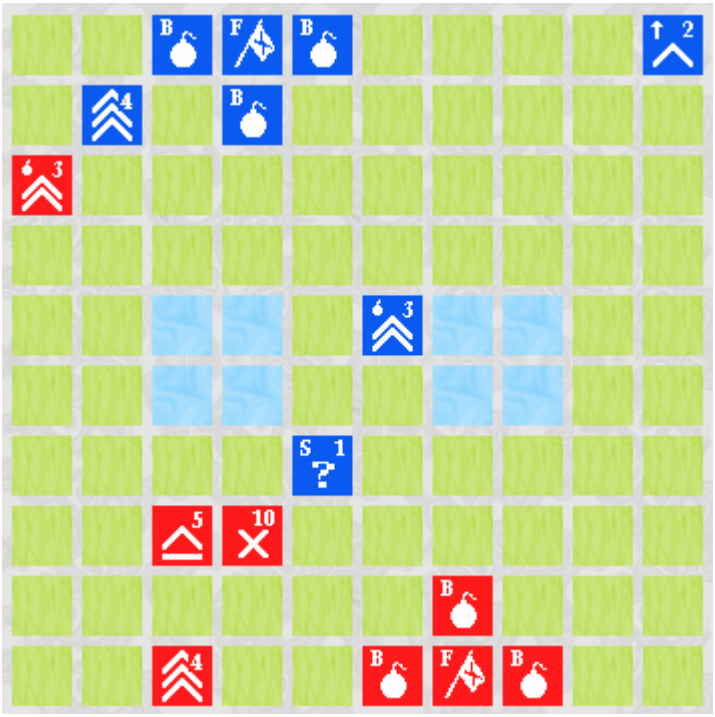
Invincible

A Stratego Bot



Vincent de Boer

November 2007

TU Delft
Technische Universiteit Delft

Invincible

A Stratego Bot

by
Vincent de Boer

A thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

Presented at

Delft University of Technology,
Faculty of Electrical Engineering,
Mathematics and Computer Science,
Man-Machine Interaction Group.

November 2007

Man-Machine Interaction Group

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands
E-mail: vdeboer_nl@yahoo.com
Student number: 1012975

Graduation Committee

drs. dr. L.J.M. Rothkrantz
dr. ir. C.A.P.G. van der Mast
dr. ir. R. Bidarra
ir. Pascal Wiggers

Abstract

Stratego is a game that has proven to be a very difficult game for computers to master. The incomplete information introduces the need to bluff and anticipate actions of the opponent which can't be solved by simple computations. Also the number of possibilities for each move and the search depth required for basic game play are so large that brute force searching techniques give no satisfactory level for even beginning Stratego players.

When posing the question whether computers can show intelligent behavior, one can not only look at domains in which the computer is proven to be strong. If computers can simulate human intelligence, then computers should be able to play the game of Stratego. Since this seems not to be the case with the existing techniques, it is an interesting field to test new ideas.

In this thesis I will describe the construction of a Stratego Bot called "Invincible" which uses original approaches for creating setups, making guesses about the ranks of unknown pieces and reasoning about the best move in a given situation.

Setups are created by a combination of statistical information of human-created setups and heuristic evaluation of the results to select the best of a number of semi-random setups. The number of setups out of which the best is selected can be used as a variable to make setups stronger or more unpredictable.

The predictions about the opponent's pieces are represented by a two-dimensional probabilities array where the columns are the different pieces and the rows the different ranks. Each piece is given a probability that it has each of the different ranks, where a known piece has probability 1 to have its real rank and 0 for all the other ranks. The sum of the probabilities of a piece is always kept 1 and the sum of the probabilities of a rank is always kept equal to the number of pieces of that rank still in the game. After each update to the probabilities the table is returned to normal form by iterating normalization over the columns and then over the rows until a target accuracy is reached.

Reasoning about the next move is split up over different types of plans, where each plan gives a value to each possible move. The move with the highest total value is selected as the best move. Plans can for example be the capture of a known opponent's piece or the defense of the own flag. Plans with a higher importance should give higher values to moves so that if there is a move they consider important this will dominate the values given by the lesser important plans. If neither of the more important plans can find a definite move, or they find several equal moves then the lower values given by lesser important plans decide the outcome. These plans are computationally light, which is possible because the problem is greatly simplified. Winning the game is a difficult problem, whereas capturing a piece with 2 other pieces on a 10x10 board is not.

These techniques result in a Stratego Bot that moves very fast at a decent level. It's level is limited by the amount of information it is given and not by the AI techniques used, but it is already clearly better than average players.

Preface

This thesis concludes a long and difficult, but also a very interesting and fun period in which I was allowed to fool around with ideas that might or might not give some result. It was a difficult journey, where the upcoming hills at first often seemed insurmountable, but when approaching the summit it then seemed like no problem could be too large. Unfortunately, the summit itself just reveals the other hills behind it and the inevitable valleys in between. Somewhere in the middle I lost sight of both where I started and where I still had to go, but there were always friends around to put me back on the right path. At the moment I write this, the few rocks in front of me seem to be insignificant compared to what has already been passed and I can look back with satisfaction to what has been achieved.

I could not have fulfilled this journey without the help of my adviser Leon, the everlasting support of my wife Lena, the wordless encouragement of my little daughter Masja, the more eloquent encouragement of my parents and the help and advise of my friends and lab colleagues whom I was allowed to bore with my problems and even agreed to play Stratego to test my program. I owe them a lot of thanks and hope to have been a source of inspiration to them as they have been to me.

Vincent de Boer

Table of contents

Abstract 5
Preface 7
Table of contents 9
1. Introduction 13
  1.1 The Game of stratego 13
  1.2 Practical use 13
  1.3 Why Stratego? 15
  1.4 Existing programs 16
  1.5 Personal experience 17
  1.6 Division in sub problems 17
  1.7 Goal 19
2. Related Work 21
  2.1 Min-max algorithm 21
    2.1.1 Introduction 21
    2.1.2 The Algorithm 21
    2.1.3 Optimizations 22
    2.1.4 Successful applications 23
    2.1.5 Min-Max for Stratego 24
  2.2 Dijkstra algorithm 26
    2.2.1 Introduction 26
    2.2.2 The algorithm 26
    2.2.3 Dijkstra on the Stratego board 27
3. Basic concepts 31
  3.1 The value of pieces 31
    3.1.1 Problem definition 31
    3.1.2 Implementation 33
  3.2 The value of information 34
    3.2.1 Problem definition 34
    3.2.2 Implementation 35
  3.3 The game state 36
  3.4 Strategic positions 36
    3.4.1 Lanes 36
    3.4.2 High piece defense 37
4. Creating a setup 39
  4.1 Importance of the setup 39
  4.2 Setup theory 39
    4.2.1 Flag placement 39
    4.2.2 Bomb placement 40
    4.2.3 High pieces 41
    4.2.4 Other pieces 41
    4.2.5 Considerations for Invincible 42
  4.3 Implementation 42
    4.3.1 The database 42
    4.3.2 Using the data 43

4.3.3 Testing the semi-random setups 44
4.3.4 Heuristic evaluation 45
4.3.5 Testing the heuristic evaluation 48
5. Move Planning 51
5.1 Introduction 51
5.1.1 Advantage: “Unlimited” search depth 52
5.1.2 Advantage: Possibilities to bluff 52
5.1.3 Advantage: Possibilities to coordinate attacks with multiple pieces. 52
5.1.4 Advantage: Focus computation on interesting situations 52
5.1.5 Disadvantage: “Invincible” knows only what it is told 53
5.1.6 Disadvantage: Comparable information is calculated multiple times 53
5.1.7 Disadvantage: Game specific knowledge needed 53
5.2 Algorithm 53
5.2.1 Plans as black boxes 53
5.2.2 The inside of plans 55
5.3 Plans used 57
5.4 Tactical plans 58
5.4.1 Immediate captures 58
5.4.2 Tactical defense 59
5.4.3 Tactical attack 62
5.4.4 Defense against gamblers 63
5.4.5 Defend marshal 64
5.5 Strategic plans 64
5.5.1 Strategic attack 64
5.5.2 Strategic defense 64
5.5.3 Trap pieces 65
5.6 General plans 66
5.6.1 Randomness 66
5.6.2 Move penalty 66
5.7 Other plans 67
6. Guessing ranks 69
6.1 Introduction 69
6.2 The model 69
6.3 Heuristics 73
6.3.1 Starting position 73
6.3.2 Missed opportunities 74
6.3.3 Behavior towards known pieces 74
7. Implementation 77
7.1 The user interface 77
7.1.1 Requirements 77
7.1.2 Class diagram 79
7.1.3 Class descriptions 80
7.1.4 Sequence diagrams 82
7.2 The artificial intelligence 84
7.2.1 Class diagram 84
7.2.2 Class description 85

7.2.3 Sequence diagrams ... 88
7.2.4 Setup representation ... 89
7.2.5 SetupCreatorV1 ... 90
8. Testing ... 93
8.1 Introduction ... 93
8.2 Test methods ... 93
8.3 Playing human opponents ... 95
8.3.1 Game 1: Mohannad defeats Invincible ... 95
8.3.2 Game 2: Invincible defeats Raymond ... 95
8.3.3 Game 3: Invincible defeats Mehdi ... 95
8.3.4 Game 4: Invincible defeats Lorena ... 96
8.3.5 Game 5: Sjoerd defeats Invincible ... 96
8.3.6 Summary ... 96
9. Conclusions & Recommendations ... 97
9.1 Extended summary ... 97
9.1.1 User interface ... 97
9.1.2 Setup creation ... 97
9.1.3 Guessing ranks ... 97
9.1.4 Move selection ... 98
9.1.5 Testing ... 99
9.1.6 Skill level ... 99
9.2 Conclusions ... 100
9.3 Recommendations ... 100
9.3.1 Adding more knowledge ... 100
9.3.2 Opponent modeling ... 101
9.3.3 Other applications ... 101
Bibliography ... 103
Appendix A: Stratego ... 105
A1 The game ... 105
How to set up the game ... 105
Game play ... 106
Rules for movement ... 106
Rules for attack ... 106
Rules of rank ... 106
Strategy hints ... 107
Winning the game ... 107
A2 History ... 107
A3 Repetition rules ... 109
The two-squares rule ... 110
Implementation of the two-square rule ... 111
Game play effects of the 2-square rule ... 111
The more-square rule ... 112
Implementation of the more-square rule ... 113
Game play effects of the more-square rule ... 114
Appendix B: The game client ... 117
The pieces ... 117

Saving setups 118
Saving games 118
Replaying games 118
Free setup 119
The menus 119
Appendix C – testgames 121
Game 1: Mohannad vs. Invincible (1-0) – Mohannad wins in 313 moves 121
Game 2: Raymond vs. Invincible (0-1) – Invincible wins in 494 moves 123
Game 3: Mehdi vs. Invincible (0-1) – Invincible wins in 316 moves 125
Game 4: Lorena vs. Invincible (0-1) – Invincible wins in 328 moves 127
Game 5: Sjoerd vs. Invincible (1-0) – Sjoerd wins in 442 moves 128
Appendix D – Test setups 131

1. Introduction

1.1 The Game of stratego

Stratego is a game where at the start the players have incomplete information. The players each have an opposing army and the goal of the game is to conquer the flag of the opponent. The exact place of the different pieces is at the start only known to the player who owns them. The players move in turns a piece to an empty square or a square that contains an enemy piece. In the latter case, a battle takes place. The piece with the lower rank is removed from the board and the winner has to reveal his identity. The further the game progresses, the more is known about the pieces remaining on the board. The amount of hidden information only decreases during the game until in the end the game becomes deterministic rather than probabilistic.
A basic overview of the rules is included in appendix A, though some parts of the report that deal with strategies might be easier understood by people who already know the game.

Figure 1.1 – Stratego.

1.2 Practical use

When it comes to games, and in particular elements of games, like artificial intelligence, one might wonder what the practical use of it is and how one can justify spending research or development time on games. Games are after all just a pastime practiced by kids who are neglecting school.

That previous statement is of course not completely honest towards games. Games are indeed a major source of entertainment, but for far more people than just a few kids with too much time on their hands. There are yearly billions or dollars earned in the gaming industry, making games economically interesting.

But games are not only recreational. There are many useful aspects about playing games. Not only do they entertain and relax the mind and so help fighting stress, they are also an innocent outlet for competitive drift, teach social skills and can serve a variety of other educational purposes. Where a book might be more efficient in transferring knowledge, children enthusiastically spend far more time on games than they would on books.

Someone once wrote in a “how do you know if you played too much” list on an internet forum about a particular game: “If you get an A on topography without studying” which is probably true because players of this game look at world maps possibly hundreds of times a day, which is far more often than they would while studying topography.

This aspect of games leads to an increasing interest in so called “serious games” which are meant mainly for education or training rather than purely entertainment. They are however still games and have the competitive and entertaining quality to keep people highly interested in spending time on them.

Examples can be found in the military where DARPA[10], or first-person shooters like Unreal tournament are used for military training, for learning languages (Tactical Language & Culture Training System[11]), medical purposes (Simendo, Re-Mission) and management training (Intel IT Management game)

Figure 1.2 – Unreal Tournament.

Finally, games have for many years been a test bed for artificial intelligence researchers to get inspiration and test new ideas. For the last decades the main interest has gone to Chess. Recently however, a computer program called “Deep Blue” was able to beat the human world champion Kasparov in a series of games. Deep Blue uses a combination of opening books, endgame databases and brute-force search on special powerful hardware. One can argue whether this still falls under AI rather than under simulation so recently interest has been shifting to other games like Go where the number of possible games greatly exceeds the number of possible chess games or to games like poker, where human aspects like bluffing and anticipating are more important than cold computation. These games provide a new challenge for AI researchers that can’t be solved with the techniques that were successful for chess.

Games are relatively simple compared to real-life problems, but their competitive nature challenges researchers to invent very good solutions that can be relatively easily compared to other solutions by letting them play this game against each other. Solutions found for games can then be used for similar sub-problems in the real world.

1.3 Why Stratego?

A lot of techniques are developed to play games like chess where the computer can investigate all the different possibilities a number of moves ahead. Some games like Tic-tac-toe, Connect-4, Go-moku or Awari are even simple enough to be completely solved[4] by the computer, meaning that for all situations it is known which move is best based on evaluation of all the possible continuations till the end of the game. Even if a game can not be solved, like chess, the techniques that simulate all the possible move sequences till a certain depth can be used to make very strong computer opponents for these games.

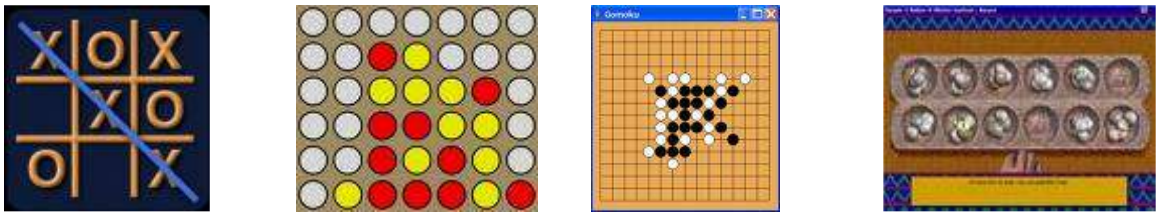


Figure 1.3 - Tic-tac-toe, Connect-four, Go-moku and Awari.

However, there remain groups of games for which the existing techniques do not provide a solution. This means that real-life problems that require the same aspects of reason probably also have no good solution yet. Stratego is one of these games that remain a challenge for the computer to compete in at the level with human players.

It is a game of incomplete information and therefore requires “human” skills like bluffing, and analyzing the thoughts of the opponent. It combines this with tactical skills and strategic maneuvers. The latter is especially difficult for computers because the effect of strategic moves is usually long-term and thus difficult to see within a reasonable search space. Players not only try to conceal information from someone else while trying to look into their mind, they also need to use the gathered information by making a better tactical or strategic plan. It is often hard to find the “best” move, because it depends not only on the situation on the board but also on what the player and, more importantly, his opponent think to know about the situation. Despite its appearance, there is almost no randomness involved. There is uncertainty in the outcome of every battle, but with each loss the uncertainty of the opponent’s pieces decreases giving an advantage in that respect. Especially at top level, the better player nearly always wins.

Many real-life problems are not solvable, so the developments of AI techniques that aim to solve a game are of limited use. In Stratego, the aim can’t be computationally perfect play since incomplete information makes this impossible; the aim is instead to find a good action in an uncertain environment. Especially the aspect of partial uncertainty is

something that occurs in many every-day problems. One of the key elements of Stratego is knowledge management. How to extract information that someone else knows and doesn't want to tell by examining the actions that he takes, but also how to use this information to gain maximal profit without revealing too much own secret information. The players must be alert to mistakes of the opponent and exploit them when they are made. Taking calculated risks and having backup plans in case they go wrong. Many of these properties are also required of managers and investors in real-life situations.

1.4 Existing programs

Invincible is not the first Stratego Bot, the oldest program that I know of is Probe[21][28], which was created in 1983 and required a supercomputer to run. This program has recently been improved and is now most likely the strongest available program. It is based on a form of exhaustive search, but also includes pathfinding algorithms for deeper searches. Probe is shareware and can be downloaded[28] for private use. It can also be found playing online at Metaforge Webstratego[12].

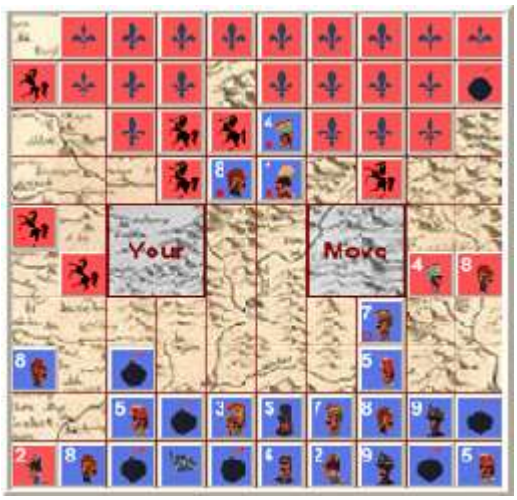


Figure 1.4 – Probe.

The first program that was commercially released was Accolade Stratego, developed by Ken McLeod in 1991. It was not very strong, but for a long time considered the strongest commercial program. This program is now freeware and can also be downloaded.

Figure 1.5 – Hasbro Stratego.

Other commercial programs include Hasbro Stratego, a game released by the company holding copyright on Stratego in the United States. It has very detailed graphics and animated battles but again not a very strong AI.

A more recent program is Sean O'Connor's "The General" which supports many different personalities for the artificial opponent. Jumbo, the dutch game publisher with copyright on Stratego has also taken an interest in computer games of Stratego and recently released a version to be played on mobile phone, and is currently working on a version for the Nintendo.

Other programs include:

- Raimonds Rudmanis "Reveal Your Rank!"
  Can be downloaded at http://www.yellowgames.com/
- Sven Jug's "Master of the Flag"
  Can be played online at http://www.jayoogee.com/masteroftheflag/
- Karl Stengaard's "Perspecto"[15]
  Made as a Master thesis project at Lund University. It beat the commercial program "The General" but unfortunately the program itself is lost.
- Agent-based Stratego by Mohannad Ismail[29]
  A bachelor project at Delft University where the goal was primarily to make an agent based system
- Erik ten Vergert's Aaaweb
  Can be played online at http://www.eriktenvergert.nl/stratego/Stratego.html
- Browserchess Stratego
  Can be played online at http://www.browserchess.net/stratego/
  At the higher level it cheats though.

I have tested all these programs, but none of them really know what they are doing. Probe is by far the strongest of them all, but even this program ranks among the worst players at the online site where it plays.

1.5 Personal experience

In this thesis I will often draw on my own experience as a Stratego player.

I started playing Stratego in may 1999, on an online site called Metaforge[12]. I liked the game, played it often and soon became one of the better players on this site. I then got in contact with someone who told me about real life Stratego tournaments that are being organized regularly in the Netherlands by SBN[13]. The level here was far higher than it was on internet and I got beaten in most of my games, but I learned a lot from these beatings and won the 4th tournament I played, despite being the lowest ranked participant. From then on I gradually got better, winning several tournaments in 2000 and the Dutch Championships in 2001. I played approximately 70 tournaments in 2000-2005 and was the only player in this period who never ended a tournament lower than the 5th place. I reached the first position in the international Stratego ranking[14] in 2002 and am still holding it today. I "retired" from playing online in 2002 while holding nr 1 positions on both large Stratego sites. My top year in live tournaments was 2003, in which I won all but one of my tournaments (where was unbeaten but still ended 2nd by a minimal margin). In that year I won the dutch championships for the second time and later won the World Championships for the first time. After that I started playing less. I prolonged my world title in 2004, but didn't win another tournament till august 2007 when I was working on this report and went to the World Championships to refresh my knowledge and won the world title for the 3rd time.

1.6 Division in sub problems

The game of Stratego can be divided in 3 major parts.

1. Creating a starting setup
2. Guessing the ranks of unknown pieces
3. Computing and executing moves

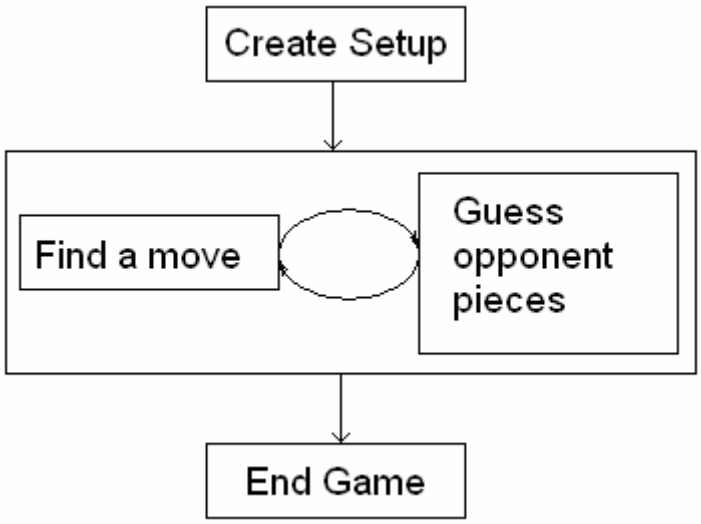


Figure 1.4 The three major subsystems of Stratego.

Each of these requires a different kind of logic. They will be the sub-parts of my program and I will treat these problems separately in the following chapters.

For the setup module, a statistical approach is used. Information from thousands of human-created setups is used in order to generate semi-random setups for the Stratego-bot.

For guessing the ranks of pieces, a probability matrix is kept holding the probability for each rank for each piece. The numbers are updated after each move of the opponent, using heuristics to subtract as much information as possibly from the opponent's actions. At the same time, the algorithm keeps an eye on what the opponent is likely to be able to conclude from its own past moves. The matrix is kept in normalized format by iterating the normalizing process over two dimensions until a target accuracy is reached.

Move planning is done not by looking at a move and examining what it can lead to but by looking at possible goals and checking how well each possible move contributes to that specific goal. This is a much simpler problem that allows "Invincible" to see opportunities that would be well beyond the search horizon of any exhaustive search algorithm. This is possible because most of the moves in Stratego are not inter-related. They might be performed in any order and always have the same result, and often sequences of moves occur that end up back in more or less the same situation. A plan to capture a certain piece might be interrupted by moving another piece out of danger, but this doesn't necessarily change the initial plan; it often only delays it. The moves that

were calculated to be necessary for capturing the piece do not have to take into account possible move-sequences on the other side of the board if they clearly don't interfere.

1.7 Goal

The goal of this thesis is to build a basis of a Stratego-bot that can compete with the best existing programs and have the potential to get to a much higher level so that it can challenge experienced human players. Unlike its predecessors for Stratego, "Invincible" will not make use of the established techniques from game AI, but develop a new technique more suitable for a game that requires dealing with unknown information and deep searches but where different actions do not always directly influence each other.

At the end of the project, there should be a working prototype of the program and a user interface that lets human players play against the AI-bot.

2. Related Work

2.1 Min-max algorithm

2.1.1 Introduction

The min-max algorithm[5] and its optimizations can be argued to be the most popular technique in (board-) game AI's. For many games it allows the programmer to create a decent computer opponent while having little or no knowledge of the game itself. It effectively performs an exhaustive search over the game tree. Because this algorithm is exponential in the number of moves (or plies: a move by one player) the number of moves one can look ahead is limited. The more different moves a player can choose from a given position, the higher the branching factor of the tree and the less deep the algorithm can search before running out of time. Generally, the game is far from over at the point the search must be stopped, so an evaluation function is used to give a value for the board position at that time. This is an estimation of how the game will evolve from that point on. This is the only place where knowledge of the game is required from the programmer, but in many games a simple check on the remaining pieces or temporary score can give a reasonable approximation.

2.1.2 The Algorithm

The min-max algorithm is a depth-first search over the game-tree until a specified depth. In short, it searches moves that lead to the best reachable position when one player makes optimal moves leading to the highest value and the other makes optimal moves leading to the lowest value, where the value of a position is either a high or low value for a win/loss situation or a heuristic value defined by an evaluation function taking the board position as input. The min-max algorithm got his name because the players are called min and max, where min tries to find the minimal evaluation value it can reach and max the maximal value. At each node where min has to make a move, the minimal evaluation value of its children is the value of this node. When max has to make a move, he returns the maximal value of the children.

This is illustrated in figure 2.1. The values on the second layer are found by taking the maximum values of their children and the top value is found by taking the minimal value of its children. This tree returns only the best reachable value, and not the path that needs to be taken to reach it. Since the move that leads to the best score is only relevant in the first step a min-max tree can be made after each possible move in the current board position and compare the values to find which move led to the highest value.

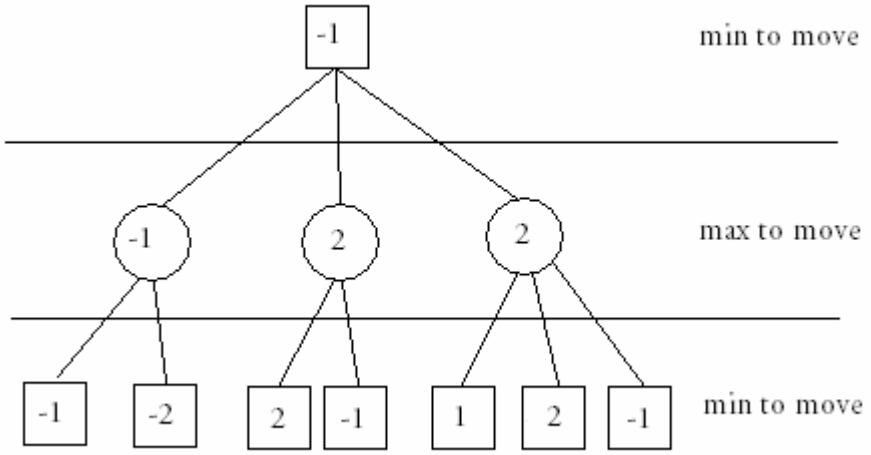


Figure 2.1 – Min-max search tree.

2.1.3 Optimizations

A lot of research has been done on ways to perform more efficient searches in this tree without losing correctness. The most notable improvement is the alpha-beta pruning[6]. Because the algorithm alternates between taking the minimum and maximum of different values, the previous results create a window of useful future values. If a minimum value is searched and a 2 is already found; only values lower than 2 can change the result. Similarly if a maximum is searched and already have a -1 only values higher than -1 are interesting. These requirements from all previous levels can be combined so if on level 1 a value lower than 2 is needed to change anything, and on level 2 a value higher than -1 then only values between -1 and 2 on the third level can change the eventual result. If this is again a minimum search and a -2 is found the value of the node will never be higher than that and thus never fall in the interesting window between -1 and 2. The search can then be stopped without losing correctness.

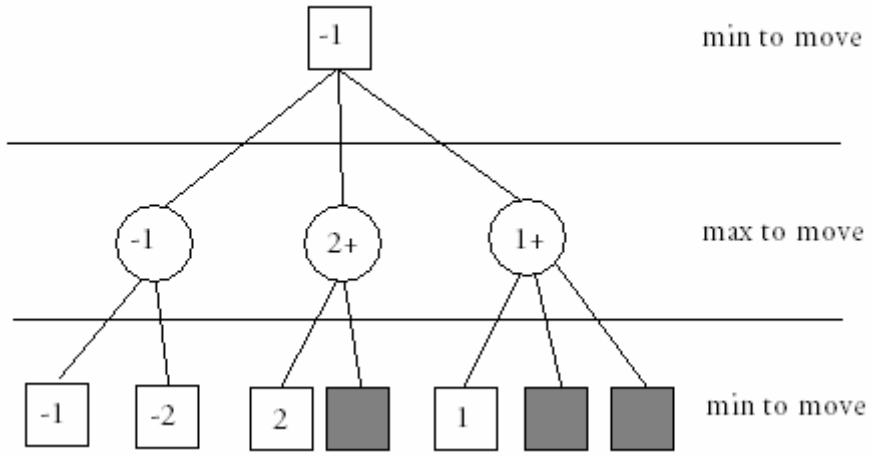


Figure 2.2 – Alpha-beta pruning.

Figure 2.2 shows the same tree as in figure 2.1. If the search goes from right to left it can safely skip the grey nodes and the whole tree that comes after them. Whatever values are given to those nodes can't possibly affect the value of the root.

The exact value of the node marked with "2+" isn't known but it is known that it is at least 2. When looking for the minimum of -1, a number larger than 2 and some unknown numbers, the number larger than 2 can be discarded so it's not necessary to find more information about this node.

Rather than searching all values, only values between alpha (lower bound) and beta (upper bound) are searched. Alpha and beta are updated based on previous results and any search path whose result is certain to fall outside the alpha-beta window is terminated. In the example in figure 2.2 three of the seven leaves can be ignored. Since in problem with a larger search depth each of these leaves is actually a whole sub tree then cutting off 3 out of 7 of these sub trees is a very significant improvement.

Unfortunately it isn't always that great. If in the tree from figure 2.1 the algorithm searches from right to left rather than from left to right then it cannot ignore a single leaf. For optimal performance the child nodes must be ordered based on how promising they are and evaluated in that order. The best situation is when the best child nodes are evaluated first, the worst (equal to normal min-max) happens when the algorithm starts with the worst child nodes each time.

2.1.4 Successful applications

Most of the artificial intelligence programs that can compete at very high level with human players are based on this algorithm. Most notable examples are chess [1] and Othello [2] where the best programs are better than the best humans. The advantage of this method is that relatively little experience with the game is required from the programmer, making it an ideal method to quickly create a decent AI-bot for lesser known games. The success of the method however largely depends on the type of game.

Othello and Chess are:

- Games of complete information
- Games in which it is relatively complicated for human players to look ahead very far
- Games with a decent evaluation function for a board position.

Complete information is important to build a search tree. In games of incomplete information there will be many points in the tree where it is not known how to continue without knowing some piece of hidden information. It is possible to overcome this by making calculated guesses about the hidden information but this sacrifice some certainty of the correctness of the answer. What is even more important is that these games generally cannot be treated purely mathematically. Even if the optimal move is found this is not guaranteed to be the best move since the optimal move is usually based on information that is hidden from the opponent and should preferably stay hidden as long as

possible. If a player consistently makes “optimal” moves he might tell his opponent a lot of things he doesn’t want him to know. Also, such approach ignores bluffing: making sub-optimal moves on purpose in order to let the opponent make a bigger mistake. Games of this type are rarely won by a direct approach. Even the best poker player won’t beat a beginner if he is playing with open cards.
It is also important for this method to work that the search depth needed to compete with human players is relatively small. Human players generally play by selecting a few interesting sequences and investigate them as deep as they can, with only a few variations to the most likely responses. This means they need to do a lot fewer calculations than the computer who explores all possible directions. The more the situation can change after a single move, the harder it is for human players to imagine the new possibilities, while for the computer this is relatively easy. In Go, where pieces put on the board never move, even average human players can think a lot further than the computer can possibly explore.

Finally it’s important that a simple evaluation function can be made for the value of a given board position. Since it’s usually not possible to calculate the game all the way till the end it is necessary to have a reliable measure to say whether one board position is better than another. It doesn’t have to mean that a higher value always means a bigger chance to win, because if such a function existed the game would be solved by a one move look ahead to see which reachable position has the highest value. It should however be a good indication whether a position is favorable or not. With a poor evaluation function the AI-bot might pick moves leading to a poor position if the evaluation function says it’s a very good one. Making an evaluation function is generally easier in games where material advantage is important than in games that are mostly positional (for example Go).

Most existing Stratego programs are based on the min-max algorithm. Even though this may quickly give a superficial player, I will argue that it can’t possibly lead to a program that can compete with experienced human players.

2.1.5 Min-Max for Stratego

Even though Stratego has incomplete information and it would be tolerably hard to make a decent evaluation function, I will explain that the real reason why this algorithm won’t work for Stratego is the search depth.

A simple example where a miner is 10 squares away from the opponent’s flag requires a search depth of 20 plies. The branching factor of Stratego can be very high. A piece standing on the middle of the board can do 4 moves, and a scout can do even up to 18 moves. It is therefore unlikely that a depth of 20 plies can be reached, but it gets even worse because of repetitive situations. If the opponent can threaten a piece that the player can’t afford to lose he will have to spend a move on moving it away from the danger. Because of the repetition rules the other player will eventually have to stop threatening his pieces, but even in simple situations he is allowed to spend 5 moves on it for every

single other move. This effectively multiplies the needed search depth by 5. Two of these situations would multiply it by 10 and worse situations are imaginable.

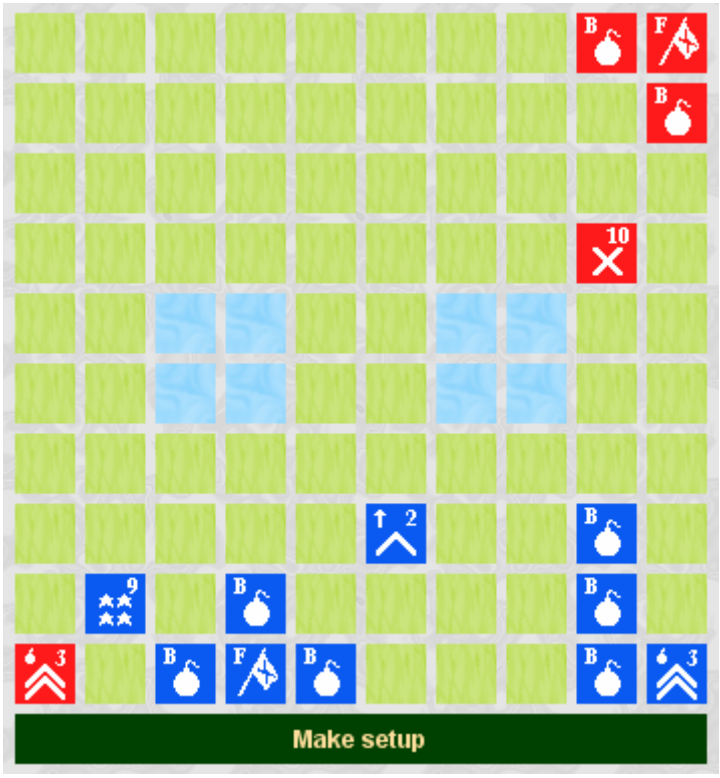


Figure 2.3 – Red wins in at most 100 moves: how?

Take for example the situation in figure 2.3: red is to move. If all the pieces are known this is an easy win for red. He can trap the blue miner and then use his marshal to help his miner get past the blue general to the flag. Without using heuristic values one can calculate that blue does not stand a chance and red will capture the blue flag, regardless of the actions of blue. The shortest path to do this however takes 28 moves (56 plies). That is if blue moves only his scout and moves his general down one step when the marshal comes close, after which the miner has to move up around the marshal and general to the flag.

If blue however moves his general down earlier, red must move his miner away since it is needed to reach the flag later. So the miner must move up, after which the general moves up to. Now the miner should go back down instead of north because this leads to a repetition of "only" 5 moves; going up again leads to a (much) longer chase. After 5 moves blue can't move up again and has to move another piece so he moves his scout. The red marshal takes one more step towards his goal and after this the blue general again continues threatening the red miner. This multiplies the number of moves needed by 6 until the marshal is close enough to the general to limit his movement. In this case, the first 18 moves are multiplied by 5, after which 10 more moves are needed to reach the flag, making a total of 100 moves (200 plies). Considering the exponential complexity of the min-max algorithm in the number of plies it seems very unlikely that a computer using this algorithm will ever be able to state in this position that red wins this game.

Apparently, the search depth that human players can reach in Stratego is a lot higher than a computer can achieve. This is a big difference from chess, where the computer can generally think further ahead and is less likely to overlook an important possible path than human players. In this situation of complete information there is even no loss of correctness. Even though there may be a miscalculation in the number of moves needed, or a path may be overlooked that allows a quicker win, but it is certain that red will win, given that neither side makes a mistake.

How is it possible that a human can "out-calculate" the computer in Stratego when he can't do so in chess? The reason for this seems to be that in Stratego people generally think in plans. They don't think about each move separately, but think in goals that must be achieved. In the situation above there are just 3 goals. Capture the miner, chase the general from his blocking position and finally go with the miner to the flag. The importance of the first plan can be assured by counting that if this plan is omitted the blue miner reaches the flag long before the red miner can. Therefore capturing the blue miner should be a priority. How to do this is also simple. There is only 1 piece that can capture him, so that piece will have to move there. It's useless to consider what happens when the red marshal moves upwards since that obviously won't contribute to a winning plan. The exact path that is taken is often not important either. It is easy to count that the miner can't escape his trap before the marshal blocks the exit so the miner can be captured. Even though blue can do a lot of moves, for this calculation only the moves of the miner need to be taken into account. Any move done by the scout can be safely ignored (which is the large majority of blue's possible moves!) Also the sequence of the blue general chasing the red miner can be ignored since it doesn't change the situation. It's just a sequence of forced moves resulting in essentially the same board position that was there before.

Rather than taking a move and examining what this move can possibly lead to, human players generally take a board position, decide on a plan and then find the move needed to do to execute that plan.

2.2 Dijkstra algorithm

2.2.1 Introduction

The Dijkstra algorithm[7] is an efficient way to find the shortest path from any given point to any other in a directed graph with non-negative distances between nodes.

2.2.2 The algorithm

The algorithm takes each time the node with the shortest distance to the source (beginning with the source itself) that has not been evaluated yet. It then updates all the neighbors of this node if they can be reached with a lower cost than they could before.

As soon as the destination node is found as having the shortest distance to the source the algorithm terminates. Note that it can not stop the first time the destination node is reached by another node since it may be possible to reach it by a lower cost in one of the next iterations.

Because all vertexes have a non-negative weight, the node that is closest to the source of all unevaluated nodes can never be reached for a lower cost by another route. Because this distance is final, it can be used to update it's neighbors with the correct cheapest cost to that node from the source passing that node.

```
For each node n
        distances[n] := ∞
        visited[n] := false

Distances[source] := 0;

While visited[destination]=false
        min_dist := ∞
        min_node := -1
        for each node n
                if visited[n] = false and distances[n] < min_dist
                        min_dist = distances[n]
                        min_node = n

        if min_dist = ∞
                break

        visited[min_node] = true

        for each node n
                distances[n] = min(distances[n], min_dist+vertex[min_node][n]

return distances[destination]
```

Figure 2.4 – The Dijkstra algorithm.

2.2.3 Dijkstra on the Stratego board

The Stratego board can be seen as a directed graph where each field is a node, with vertexes of equal weight between every horizontal and vertical neighbor.

The algorithm is simplified because all vertexes have equal weights. This means the distances can be updated in "runs" where in each run fields that are connected to fields reachable in n steps but have not been reached in previous steps are given distance n+1.

This also makes it easy to take moving enemies into account that may or may not block the path of a piece. If in each run the distances from all enemies on the board are updated first and the distances from all friends on the board second then it is easy to see whether the enemy can reach block the path of the friendly piece.

It is also possible to define obstacles as inaccessible fields, such as enemy bombs or own pieces. Usually both colors have different fields that block them. For example a low known friendly piece would block his friends, but not a high opponent piece.

Figure 2.5 gives in pseudo code the algorithm that calculates the distance from any piece defined as friend to any other reachable field on the board taking into account that pieces defined as enemy must be avoided.

```
for each field f
        if enemyAt(f) enemies[f] := 0
        else enemies[f] := ∞

        if friendAt(f) friends[f] := 0
        else friends[f] := ∞

        if isObstacle(f) obstacles[f] := true
        else obstacles[f] := false

updated := true
run := 0

while updated
        updated = false
        run++
        for each field f
                if enemies[f] = run and not obstacles[f]
                        updated := true
                        for each neighbor n
                                enemies[n] := min(run+1,enemies[n])

        for each field f
                if friends[f] = run and enemies[f] > run and not obstacles[f]
                        updated := true
                        for each neighbor n
                                friends[n] := min(run+1,friends[n])
```

Figure 2.5 – adapted Dijkstra for multiple sources and targets with equal distances between nodes.

This is a simplification of the real situation because a single enemy is considered to be on all the squares it could possible reach, which is a safe method when the enemy is

determined to stop a single attacker from reaching his goal. When there are multiple attacking pieces however, they often can't be stopped by a single enemy so this method would erroneously state that the goal can't be reached.

The computational gain however is huge. Each field is evaluated a constant number of times so the order of this algorithm is O(n). In the case of Stratego, n is actually also a constant since there are only 100 fields on the board, it is independent on the number of moves that are needed to complete this plan!

Even though this method has its limitations, I found it can solve a variety of problems that can occur during a Stratego game. Some examples are:

- Detection of an open path from a miner to a possible flag.
- Calculating the distance of an unbeatable piece to any unmovable target (may be a clump of moving pieces that may theoretically move but are likely to provide a capture possibility).
- Detection of the reachable area of a given or of all enemy pieces.
- Detection whether the own flag can be reached.

More complicated attacks like 2 miners approaching a flag with a single defender can not be so easily seen. To detect this possibility in reasonable time one could use the heuristic that 2 attackers cannot be stopped by 1 defender, nor can 3 attackers be stopped by 2 defenders if the target is not in the corner. The attackers should be as far apart as possible (approaching from different sides). This would require more programming, but would not add to the time complexity of the algorithm.

3. Basic concepts

This chapter will discuss some basic concepts about the game Stratego. This is a part of my own knowledge of the game that I used to design some elements of “Invincible”. I will discuss some background of the problems and the solutions that I implemented.

3.1 The value of pieces

3.1.1 Problem definition

It is often useful to be able to give a numeric value to a piece, for example to calculate the profit of attacking a piece, or for determining whether preparing an attack on one piece is better than on another piece. When attacking an unknown piece “Invincible” can then take expected profit by taking a weighted sum of all the possible outcomes and their probabilities. He multiplies all the values of the pieces he can capture by the probability that the unknown piece has this rank and subtract his own rank multiplied by the probability that he will lose the attack. It is not only necessary that a “better” piece has a higher value, but also that a much better piece has a much higher value so that the small probability of the loss of an important piece weighs stronger than the large probability of the capture of an unimportant one.

The problem of giving a value to pieces is a very difficult one in Stratego. In general it is of course true that a higher rank has a higher value, but what really matters is the abilities they have. Higher pieces generally have more abilities than lower pieces because they can capture more pieces, but there are also pieces with special abilities like the spy who can capture the marshal or the miner who can capture bombs. Also the movements a piece can make are important, a bomb that can beat all pieces clearly deserves a penalty in value because of its immobility and the scout’s speed should also be reflected in the piece’s value. It is hard to give an objective formula how the value of a special ability relates to the value of having a higher rank.

There is another issue: A Colonel (rank 8) is more valuable than a Major (rank 7) because the Colonel can capture Majors and the Major can not. If the opponent however doesn’t have either Majors or Colonels then the Colonel has no advantage over the Major and has an equal value. Also, a General is much more valuable than a sergeant because at the start the general can capture 17 pieces that the sergeant can not. At the start of the game it would be very profitable to get the enemy general in return for 2 sergeants. Figure 3.1 however shows an endgame position where blue has a general against red’s two sergeant and blue should be very happy to get a draw. If the blue general is known and the other pieces are not, which is not a very unrealistic situation in an endgame then red has an almost certain win.

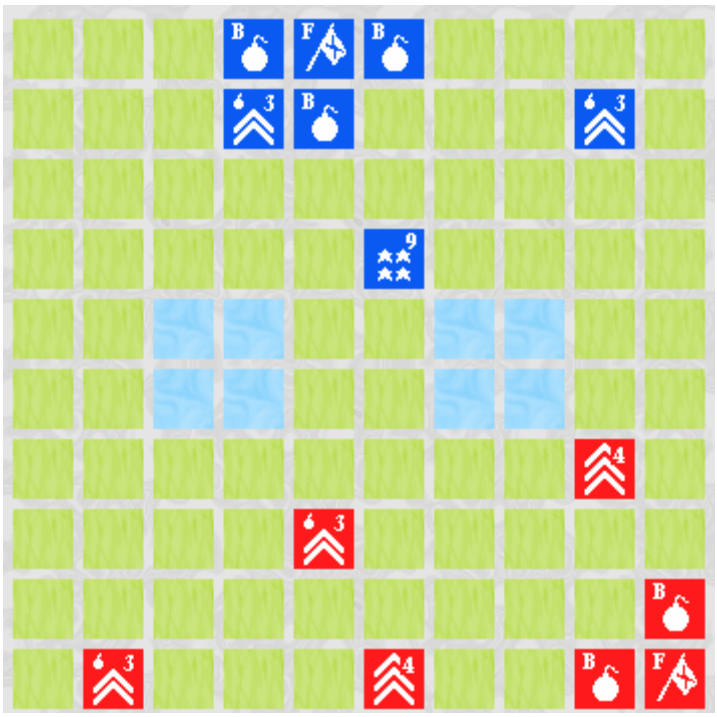


Figure 3.1 – blue should be happy with a draw.

This shows that pieces don't have absolute values, but one that changes depending on the pieces of the opponent. It also shows that pieces of equal rank don't have the same value, because it is possible to get from the starting position with a red general and two blue sergeants removed to the endgame position in figure 3.1 by only removing equal pairs of pieces from the board, yet somehow the material advantage shifted from blue to red in the meantime.

This seems strange, but can be explained by the loss of ability on the general's part. If he had captured only a single piece higher than sergeant blue would already have a won position, but with all these pieces removed, blue no longer has this possibility.

Figure 3.2 shows a more obvious example in which pieces of equal strength have an unequal value, and it also shows the importance for a player to recognize this. If both colonels are removed from this position then red wins because his two majors can trap blue's major and blue will not have any pieces left. This means that the exchange of colonels is favorable for red and unfavorable for blue, or in other words, the blue colonel is worth more than the red colonel. Not recognizing this might result in a result-changing mistake.

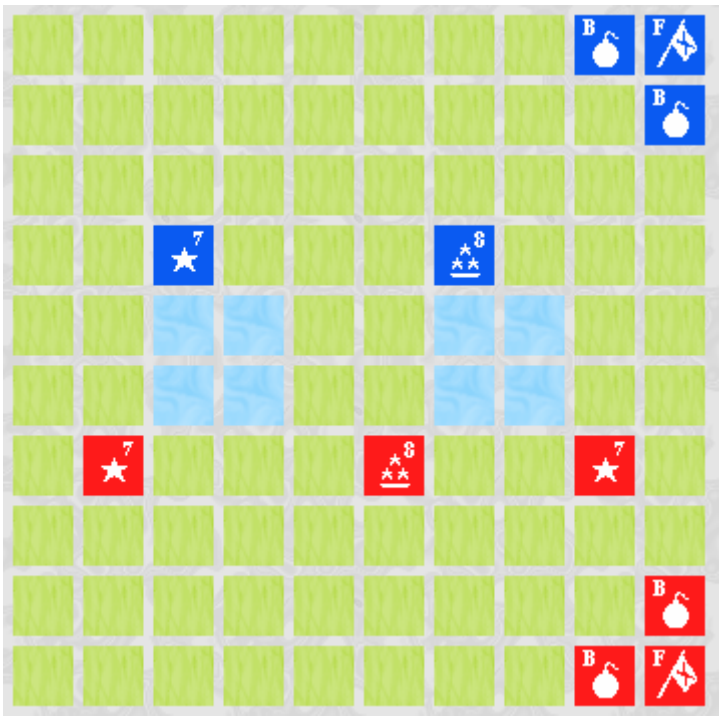


Figure 3.2 – The blue colonel (8) is worth more than the red colonel (8).

A final consideration is that special abilities become more valuable when they are rarer. The less scouts or miners "Invincible" has, the more careful he should be with his remaining ones because when he loses this ability completely it makes his opponent's defense a lot easier.

3.1.2 Implementation

The piece values are recalculated after every move. As a general rule, a piece is only worth more than a lower piece if the opponent has a piece that can be captured by this piece, but not by the lower piece. The base values of the pieces are therefore decided by iterating over the existing ranks, starting from the lowest, and multiplying the value for the next rank by a constant factor if both players have at least one piece of the previous rank or the new one. This is given in pseudo code in figure 3.3. The rank_difference_factor is a heuristic value, set by trail and error on 1.45.

```
Basevalue[1] = 0.02

for( rank = 2 (scout)..9 (marshal) )
   if( (nr_red[rank-1]+nr_red[rank]>0 ) AND  (nr_blue[rank-1]+nr_blue[rank]>0 ) )
       basevalue[rank] = basevalue[rank-1]*rank_difference_factor
   else
       basevalue[rank] = basevalue[rank-1]
```

Figure 3.3- Pseudocode for determining the base value of a rank.

The base value of a rank is the same for both colors, but as explained in chapter 3.1.1 this is not always correct. As a rule, I used that of equal ranks, the piece that can capture more opponent pieces has a higher value. This makes sense, because removing a high piece from the game increases the value of the pieces lower than the captured piece. If two equal pieces are removed then the side with more pieces lower than the removed pieces profits most by this exchange. The pieces of this rank of the color that has less lower pieces have their value increased by another constant factor, which should be lower than the rank_difference_factor because this doesn't make the piece more valuable than a piece of higher rank. I chose this factor at 1.15.

The ranks are then scaled so that the highest piece always has value 1.0.

There are a few pieces that have additional rules:

- The flag is given a value higher than one to signify its importance
- The spy is given half the value of the marshal
- The marshal's value is multiplied by 0.8 if the opponent still has a spy
- If there are less than 3 miners or scouts their value is multiplied by [4-nr_left]
- Bombs are given a value of 0.5 (half the value of the strongest piece)
- Each piece gets an added value of 1/[nr_own_pieces] to increase the value of pieces when there are less pieces left.

The value of the flag is made dependent on the expected winning chances; in particular the value is higher of the color that is ahead in pieces, increasing with an increasing lead. This represents an increasing importance of defense in games that "Invincible" is expecting to win and an increasing importance of guessing for the flag in games where "Invincible" is expecting to lose. This value is typically around 3-5 for the side that is ahead and 1 for the side that is not.

3.2 The value of information

3.2.1 Problem definition

Attacking a piece that is very likely much a higher rank than one's own piece is a very common move in Stratego games. A naïve artificial intelligence approach might find that a stupid move, because he is likely to lose a piece which is in principle a bad thing. Yet every Stratego player will be very happy with the result when his scout loses to the opponent's previously unknown marshal and thus finds its location.

Apparently, information has a value. Furthermore, it has a value that can be compared to the value of a piece.

The higher the rank of the piece that loses the attack, the less happy that player will be with the result, and the higher the rank of the winning piece, the happier he will be. Of course this "happiness" must be translated to numbers. The negative effect of losing a higher piece is already caused by the loss of a higher piece, so it's only necessary to add a

positive value for discovering the identity of an unknown piece that is relative to the value of the unknown piece. I say “value” and not “rank” on purpose, because in the same way that the values of a piece of a certain rank differs throughout the game, the importance of finding its location also varies.

If one of the highest pieces on the board becomes known, this reveals that all other pieces don’t have this highest rank and thus can be attacked safely by the second-highest rank. This also makes it safer to attack on another part of the board, because it is known to the opponent that the unknown pieces there are not higher than his second highest piece; likewise, defending gets harder when there are fewer opportunities to bluff that a low piece is actually high enough to stop the attacking pieces.

If one of the middle pieces becomes known this gives a different kind of disadvantage as it does not directly influence the ability to bluff, but because these pieces are still too valuable to lose they must be defended, which limits the attacking possibilities. If the high pieces are revealed while attacking with them then these known middle pieces will become a profitable target for the opponent.

Finally, one of the low pieces may become known. This has the least importance, because low pieces can be sacrificed. However, once a low piece is known, the probability that its sacrifice will reveal a high piece becomes much lower. It is far more likely that the opponent will use a piece that is just one rank higher to take this piece, giving away as little information as possible. This disadvantage is especially low against human players because they generally don’t remember these pieces for a long time.

3.2.2 Implementation

To keep things simple, I decided to use a constant factor for the value of discovering the identity of a piece. The profit of losing a piece to a higher piece is always given by reveal_penalty*piece_value, where piece_value is the value of the revealed piece. If the winning piece was already known then the profit is of course 0 because no new information was found by this attack.

Initially, this constant value is set at 33%, which is again a heuristic value that gives good results in the games “Invincible” played. One problem with this approach is that in the end of the game Invincible was often unwilling to reveal his last pieces, which made him more passive than he should have been.

The reason is that in the end there are fewer pieces and those that are still unknown are a lot easier to guess. It therefore makes sense to reveal them easier and get some material profit. The value of keeping Invincible’s pieces unknown is then lower. This is implemented by letting the reveal penalty decrease when the opponent has less moving pieces left. Specifically: when the opponent has exactly or less than 15 moving pieces, the reveal penalty is 2%*[moving pieces].

3.3 The game state

An important aspect in decision-making in Stratego is the state of the game. Depending on whether a player has a strong, possibly winning, position or a likely loss he may take different decisions in the same situation. Human players on tournaments often look at the captured pieces to determine whether they have a good position or not.

It might for example be unwise to try a bluff when the game can easily be won by playing careful. The opponent has little to lose when calling a bluff because he is losing anyway. The same bluff in a lost position however might fool an opponent and turn the game around.

Also, hitting unmoved pieces with high ranks is generally a bad idea since they can be bombs and losing a high piece greatly reduces the chances of winning. In a losing position however, doing this might be the only chance to win and it would be a mistake not to try.

This is in fact a state machine where depending on the state, different strategies are followed. For example when a player is far behind in pieces he might do risky attacks, while if he is comfortably ahead he would rather play safe and pay more attention to defense against such attacks. Also, having the highest piece, but less lower pieces requires a different way of playing than having more pieces but not the highest piece.

To keep this simple I implemented the game state simply as a value that represents the relative strength of both armies. A value higher than 1 means a good position for the computer player and a value lower than 1 means the opponent is ahead.

This value is calculated by taking the weighted sum of the values of all “Invincible”’s pieces and dividing that number by the weighted sum of his opponent’s pieces.

The weights are 2 for the highest piece each color has left after all the equal ranks are removed and 1 for all the other pieces.

3.4 Strategic positions

3.4.1 Lanes

The goal of some strategies is not directly to capture pieces but rather to occupy important positions on the board. Because of the lakes in the middle of the board, all attacks must go through one of the three passages between the lakes, also called “lanes”. This is the easiest place to defend because a single piece can block it. Even if the opponent has higher pieces it helps to occupy such position because it can stop miners from reaching the other side. It takes two higher pieces to remove a piece that is determined to occupy a lane. In most situations elsewhere on the board it is possible to

get past a single defender with a miner and one high piece, but not when the defender is on one of the 4 squares forming a lane.

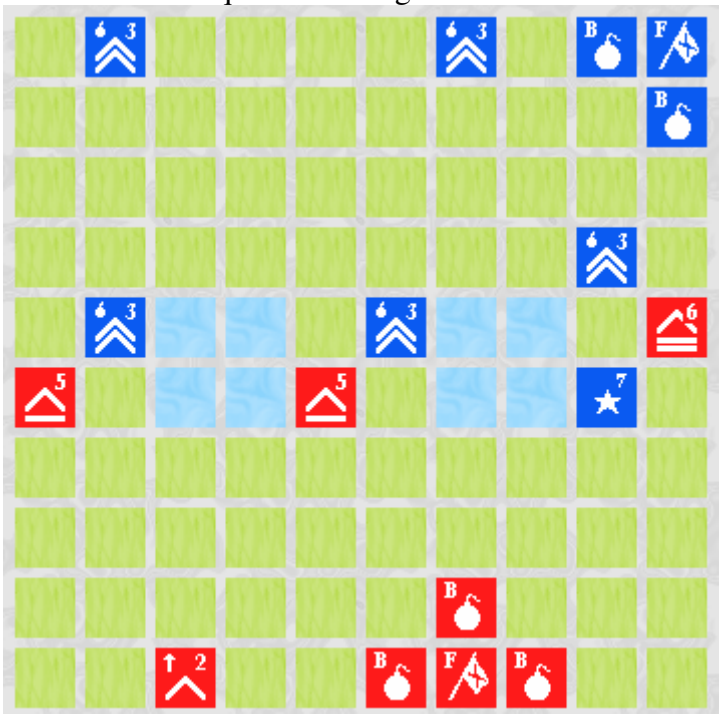


Figure 3.4 – Red is outnumbered and uses the lanes to defend to a draw.

In figure 3.4 red is seriously outnumbered and would lose the game if not for the presence of the lakes on the board. If red keeps his pieces on the four squares between the lakes then blue should not be able to get his miners to the flag, making this a draw.

In less extreme cases, keeping a high piece defending the lane in front of the flag can greatly reduce the attacking options of the opponent because he will have to take a much longer route to approach from the side, probably passing more unmoved and unknown defending pieces.

In a situation where one player has the four highest pieces on the board, but the opponent greatly outnumbers him in lower pieces, the lanes are also the key to achieve victory. The winning strategy is then to put a high piece in all of the three lanes and use the fourth to trap those pieces that get too close to one of the defenders so that it can be captured without letting another pass by.

3.4.2 High piece defense

When a player has only a single highest piece against multiple lower, then defending at the lanes may not be effective enough. The opponent can then come by another lane and move freely towards the flag. If the flag is bombed in, then the opponent usually needs at least two pieces near the flag to form a danger. One miner to take a bomb and a piece that is high enough to beat the unmoved pieces protecting the bombs.

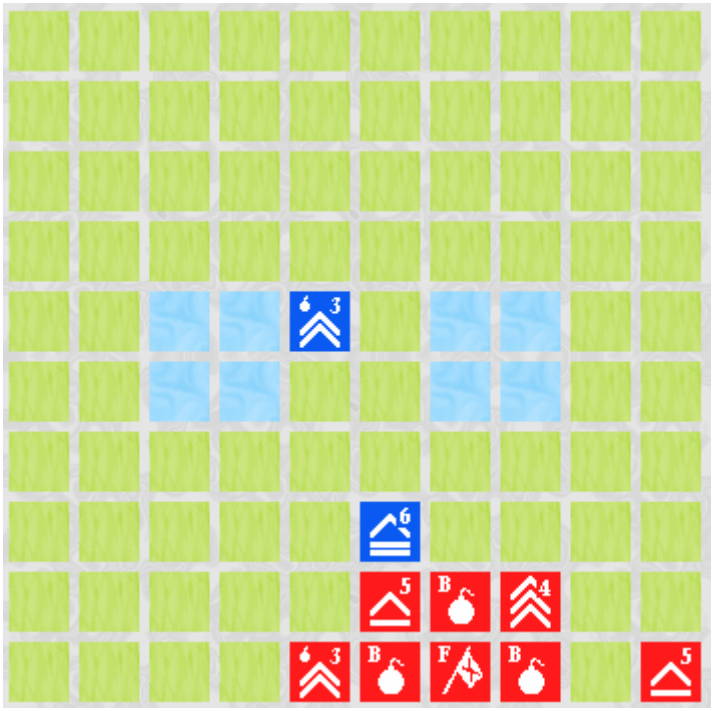


Figure 3.5 – Reds pieces are all unknown, but blue's miner should not be allowed to guess for the bomb in front of the flag.

A situation like in figure 3.5 is a (part of a) very common endgame position. Red didn't move any of the shown pieces so any of them may be a bomb. Guessing with the blue captain would be risky, but attacking the bomb in front of the flag with the miner is a win-win position for blue. If it is the bomb he is virtually certain of the flag and can always guess it when it becomes necessary to risk. If it is not a bomb but a piece higher than miner then this piece is trapped with no way to escape from the blue captain. Allowing a position like this is very dangerous for red. If red still has a piece that is higher than a captain he would be advised to bring it closer to solve this situation.

In general, keeping a high piece behind to defend against attacks from multiple pieces is a good idea when being ahead and losing the flag is the only way to lose the game. The important fields are usually those on the third line in front of or one field next to the flag. If there is an invincible piece on these fields it is very hard to plan a successful attack on the flag. It's not impossible, but significantly harder than if the high piece would be somewhere else.

4. Creating a setup

This chapter will first give some basic theory about making Stratego setups and explain how the problem of creating a setup is solved for “Invincible”. The goal is to give the reader understanding of the type of problems which are faced when creating a Stratego setup.

4.1 Importance of the setup

At the beginning of the game, the players can choose how they arrange their pieces on the 40 squares that are assigned to them. A good setup is vital for a successful Stratego player; however an unpredictable setup is even more important. For this reason there can’t exist an optimal setup; if there were any it would be obvious or at least become quickly known and therefore be a very bad setup. There must be a trade-off between having pieces on the right places and being hard to read for the opponent.

The problem of creating setups could easily be solved by adding a list of manually created setups and letting “Invincible” pick one of these at but this would very soon become predictable. It is therefore important that the computer can generate its own setups. They do not have to be perfect, as long as they are decent and unpredictable.

4.2 Setup theory

I will start with explaining a few guidelines about what is important when making a Stratego setup. There are many heuristics human players think about when they are making a setup. When a human player makes a setup, an important factor is the strategy he wants to follow in the game against that particular opponent. The same setup may be the key to victory with one playing style against a certain opponent but be the cause for losing the game when using it with a different style. When the opponent is unknown and the player is comfortable with different playing styles, this can be ignored so only general heuristics are discussed here.

4.2.1 Flag placement

One of the first questions is what to do with the flag. After all this is the most important piece in the game and its defense can’t be ignored. The safest place to put it is on the last row surrounded by 3 bombs or in the corner surrounded by 2 bombs. The corner is much more difficult to defend because defending pieces can come only from 1 side and can thus be blocked by a single high piece from the opponent. In the middle defenders can come from two sides and if the flag is behind one of the lakes the attacking miner needs to make a longer path around the lake to get there. The flag on one of the 4 squares behind the lakes on the last row surrounded by 3 bombs is therefore by far the most popular place for the flag. The opponent needs to get an important lead in pieces before he can attempt to attack the flag in this position.

This is of course very predictable. It is a good defensive position at the cost of basically telling the opponent where to go, so this should only be done when the opponent is not expected to get a lead in pieces.

The alternative is to put it in an unexpected place. This is very strong in close endgames because the opponent is more likely to lose some pieces trying to find the flag. Because a different position requires a (sometimes very big) lead in pieces to properly defend it may also cost a player the game when he has a comfortable lead but not big enough to defend the flag in such position. The opponent may decide he has nothing to lose and luckily stumble upon the flag. It is best not to do this against weaker opponents. Firstly because such players usually fall behind in pieces, making it hard for them to reach the flag in a safe position, and secondly because the power of surprise works best on experienced players and not so well on players who don't know what to expect anyway.

4.2.2 Bomb placement

Bombs are the best defense in Stratego. The high pieces are often needed to attack which allows the opponent to do the same. The more of bombs are known the more profitable it is to take risks by attacking unmoved pieces once the highest pieces are found.

Since bombs can't be moved they can also be a huge hindrance to the player who owns them when they stand in his way.

Both of these are reasons to put bombs more in the back of the setup. First of all this gives the player the front 2 rows to maneuver, which is needed to let 2 pieces pass each other behind the lake and secondly since they are needed mainly later in the game when the high pieces are revealed. Many of the front pieces will have moved away by then, making bombs there rather obvious because they haven't moved yet.

The biggest reason not to put all bombs on the last 2 rows is the predictability. If a player knows his opponent always puts all his bombs in the back then he can take a very high piece (e.g. general) and just hit all the pieces on the first 2 rows once he discovers the enemy marshal. This is risky because the game is usually lost if a bomb is hit but it can be done it against too predictable opponents.

A common strategy is to have one side of the board with a lot of bombs and unmoved low pieces and defend the middle and the other side of the board with high pieces. Attacking the side with the bombs and low pieces is generally very costly and time consuming while it gives little profit so it doesn't need to be defended. It allows the player to focus his defense on only two of the three entrances between the lakes.

This is good for the side that has the lead and bad for the one who is behind in pieces so it is unwise to block one side completely. If the player who blocked one side of the board gets behind in pieces he wants to be able to open it up and use it as a third attack route.

As with everything concerning setups, it is best to vary with this. Even the best bomb structure quickly becomes useless if the opponent recognizes it.

4.2.3 High pieces

The next thing to think about is the high pieces. They are often needed relatively early in the game so they must be in easily available places. A player still wants to keep their exact location unknown till he can make a profit by revealing them, which means they shouldn't be too exposed to attacks from scouts. This is why they are usually put on the first or second row behind the lakes or on the second or third row behind a lane.

A marshal in the back of the field may work very well as a bluff, but it's only a bluff when it comes unexpected so that can't be done too often.

4.2.4 Other pieces

A good setup has a plan for every single piece, so not even the least valuable piece is put at random. Most games start with testing each other out by sending low pieces so some low ranking pieces are needed in the front to use as cannon fodder and to protect the high pieces from being found by the opponent's initial attacks. In this phase of the game the aim is to gather as much information as possible while giving away as little as possible.

After these initial encounters, this information is used to get a material advantage. The high pieces are usually revealed to do so, so the character of the game changes. In the beginning there is no piece that doesn't have an unknown enemy that can beat him, but once the high pieces start getting known the next highest pieces know where their enemy is and can safely attack elsewhere. As long as the game is balanced, this usually results in a lot of trade-offs when equal pieces attack each other and are both removed from the game. This in turn leads to the "invulnerable" ranks getting lower and lower because they higher ranks are removed from the game. In this phase a player should never attack with a piece slightly lower than the highest unknown ranks because the risk to lose it to the highest ranks is very big, while the chance that the opponent lets this piece capture something even lower is very small. A good strategy here is to move a bluff piece to pretend it is a high rank, but since these pieces tend to get captured a lot whether the bluff works or not it's best not to use an important piece, like the second highest rank, for this purpose.

The result of this is that the middle pieces are nearly always used in descending order. The majors are used before the captains and the captains are needed before the lieutenants. It is important to think about this when making a setup. A major should not be placed behind a captain because that means the captain will have to be moved in a stage of the game where a major is needed and when the last piece a player would like to move is a captain as this makes it a target for the opponent's majors.

4.2.5 Considerations for Invincible

One consideration to make is that the computer, unlike humans, doesn't suffer from "moods" that make him unable to resist the urge to attack with a defensive setup. A good computer player should be able both to be a strong defender and a good attacker and should be able to detect what suits best to the current board position and thus to the setup. For the sake of unpredictability it is also a lot better to make the playing style dependent on the setup rather than the setup dependent on the favorite playing style.

4.3 Implementation

The base of my setup generator is a semi-random generator based purely on statistics. I used nearly 20,000 real setups to collect a probability distribution of the different ranks over the 40 starting squares. I also collected stats of how often 2 ranks appear next to, in front of or behind each other. I will explain how this is done in chapter 4.3.2.
These statistics contain not only good setups, but also a lot of mediocre or bad setups. Still, a lot of the above principles seem to be so basic that they can still be recognized in these general statistics.

4.3.1 The database

The database used to collect this data is downloaded from Gravon[2]. This is an internet site where among other games Stratego can be played against players from all over the world. All the games are stored and publicly available for download. The games contain no names of the players and are published in random order several months after they were played. Only the month in which the game was played is known. This is done to prevent the database from being a way to get information on the style about a specific player.

There are players of widely different strengths playing on this site and since the database is anonymous there is no way to tell whether a setup was made by a weak or by a strong player. It is possible to use only the setup from the winning player of each game but that would leave out a lot of good setups of which the player lost to a stronger opponent while it includes bad setups from games where two bad players played against each other.

The main goal of analyzing this data is to find general rules that apply to all Stratego setups and not to pick specific good setups and use them completely. I make the assumption that even bad players use general setup principles while the thing that distinguishes really good setups is the specific combination of all the pieces which is lost anyway when an average over thousands of setups is taken.

I therefore decided to simply use all the setups from the database and not try to make a distinction between good and bad players.

4.3.2 Using the data

Starting with the flag, then bombs, marshal, general, spy and then the lower pieces I now one by one place the pieces on the board according to the probability distribution found earlier. Whenever a piece is put on the board, the probabilities of adjacent fields are multiplied with the relative frequency of the ranks being adjacent to the rank that was just put on the board.

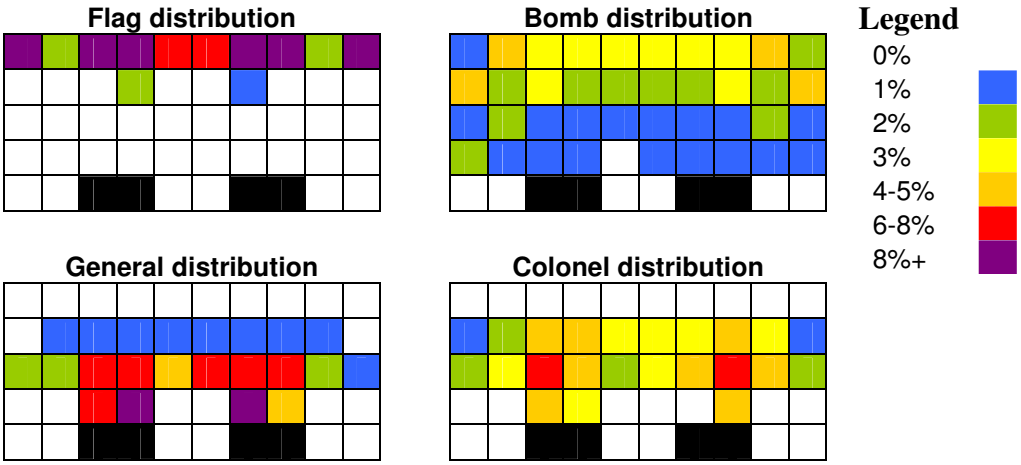


Figure 4.1 – The probability distribution of a selection of ranks.

Figure 4.1 shows the some examples of these probability distributions. It shows that the flag is nearly always on the last row. In 66% of the setups, the flag was on any of the 8 purple squares. Bombs can be expected anywhere on the field, but with a very low probability on the front 2 rows. The squares next to the back corners have the highest probability. Possibly to protect a corner flag, or to fake a flag position by bombing in a sergeant (both are common strategies). Not surprisingly, the higher pieces are usually more to the front while the lower pieces appear more often in the back.

These statistics assume however that the position of a piece doesn't depend on the other pieces at all. This is assumption would be very hard to defend. In practice, there are probably all kind of relations between the positions of pieces and in a well thought over setup the position of each piece may depend on all the others. To model this, one would need to store a relation for each field with each other field for each rank with each other rank, resulting in 40x40x12x12= 230.400 numbers. Even though it would in principle be possible to store that many numbers, they would have little statistical significance if they were collected from a mere 20.000 setups. I therefore chose to simplify the model and assume that the probabilities are only dependent on the pieces directly next to them. It seemed obvious to discriminate between the different directions because a flag for example would very often appear behind a bomb but rarely before one.

I counted the number of times a rank appears next to another rank and divided this by the number of times this was expected to happen. This calculation results in a number larger than 1 for pieces that are often put together and a number smaller than 1 for pieces that appear next to each other less than the average number of times.

I chose not to take more complicated relations into account like bomb patterns or the combination of pieces on a side used for an attack because the number of possibilities for suchlike patterns is endless and it's not feasible to collect statistical data for this. The next-to relations result in only 3 numbers for each combination of two ranks, so 3x12x12 = 431 numbers.

Table 4.1 – The next-to relation of a selection of ranks.

	M	G	C	Sg	Sp	F	B
Marshal	0	1.8	1.8	0.6	1.3	0.2	0.4
General	1.7	0	1.2	0.7	2.5	0.1	0.5
Colonel	1.6	1	0.2	0.5	2.8	0.3	0.8
Sergeant	0.8	0.9	0.8	0.6	0.6	0.9	1.8
Spy	1.1	2.3	2.7	0.4	0	0.3	0.9
Flag	0.4	0.2	0.5	1.1	0.5	0	5.6
Bomb	0.4	0.5	0.8	1.3	0.9	3.4	0.6

Table 4.1 shows a selection of the table of ranks appearing next to each other. A lot of the middle pieces have numbers close to 1 because they are often placed without much consideration, but the ranks in this table show some interesting (though expected) deviations. It shows that bombs and flags have a very strong relation, and the spy and general or colonel have too. Also the highest pieces seem often to appear next to each other. Sergeants have far less extreme relations which suggests they are placed with less care than the more important pieces. They do appear rather frequently next to bombs though, which makes sense since they are the lowest pieces that can kill the only enemies of bombs, making them a very tough defensive combination.
Even though the next-to relation is symmetrical, the table is not, as might be expected, symmetrical on the main diagonal. This is because the expected number of occurrences is not symmetrical. One reason is that there is a different number of the involved ranks and another that one rank may have had more neighbors than the other in the investigated setups.

4.3.3 Testing the semi-random setups

The semi-random generator creates setups that are a lot better than purely-random generators and may occasionally result in a more than decent setup. Mostly however there will still be some obvious flaws that are sure to cause trouble somewhere during the game. To test these setups I printed 12 setups generated by my program and took them to the World Championships. I asked the opinion of several expert players on them and there were only one or two of them that they would possibly use on a tournament. Mostly the reasons for discarding the others were easy, like important pieces being trapped by bombs or a very weakly protected flag.

I played two practice games with my two favorite setups out of these 12. Sometimes the unpredictability of the setup helped me, but at other times some minor flaw showed itself.

I won both games, but didn't consider the setups good enough to dare use them for any of my official games.

Even though this method still generates a lot of very bad setups, the result of two decent setups in 12 is still a lot better than for purely random setups where there probably isn't more than 1 setup of that quality among a million. Also, because human experts could usually distinguish the worst setups by one or two simple aspects there is hope that the computer can do this too.

4.3.4 Heuristic evaluation

The problem now comes down to generating a number of setups that is large enough to have at least 1 good setup among them and then to find the best of these using a heuristic function. The smaller the number of setups the more likely it is that neither of them is acceptable, the larger the number the more predictable the result becomes. It does have to be more than 12 because the previous test set wasn't large enough to guarantee a very good setup. I picked the arbitrary number of 50 setups for.

I used the following heuristic functions (H1 till H9) to give a value to a setup:

H1 - Distance to freedom

One value that is very important for many of these heuristics is the "distance to freedom" or the number of pieces that have to be moved before this piece can be moved. Pieces on the first row and not behind a lake have a value of 0; pieces behind them have a value of 1 and so on. Because bombs and the flag can't be moved they function as obstacles so a piece has to move around them to get out.

H2 - Pieces surrounded by bombs

Pieces that are completely surrounded by bombs can't be used unless the opponent is kind enough to free them. Because a player can't count on this courtesy this often means that these pieces can't be used at all. Pieces that are trapped can be recognized because they have an infinite distance to freedom. Any piece except the sergeant, lieutenant or of course the flag that is trapped by bombs gives a value of -2. Pieces partially surrounded give a value of -1 when their distance to freedom is larger than 5 (without obstacles no piece has a distance larger than 4). Note that bombs are also included in this rule, even though they can't move even if they have the liberty. This is because a bomb that is completely trapped by other bombs is rarely going to be useful. It can be a powerful bluff, but it's usually just bad. The lieutenant and sergeant are excluded from this rule to have some variety, though even putting these pieces between bombs is usually not very good.

H3 - Flag Bombed in

A flag between bombs makes the defense significantly stronger. This might become very predictable so it's good to have it in the open sometimes, but not often. I therefore give a bombed in flag a +5 value.

H4 -Flag defense

Another thing that often went wrong was a complete lack of strong pieces on the side of the flag. This means that if he attacks on that side with even a low piece the flag will be completely exposed. Based on the side where the flag stands "Invincible" checks which pieces stand on fields that defend that side.

Table 4.2 – Defensive fields depending on the flag side.

Left	Left	Left	Middle			Middle	Right	Right	Right
Left	Left	Left	Middle	Middle	Middle	Middle	Right	Right	Right

If the flag is for example on the left side of the board it's preferable to have some high pieces on any of the fields marked "left" in table 4.2
The marshal gives a value of +3, if this piece isn't there then the general gives a value of +2. A major or colonel also gives an additional point. There shouldn't be too many weak pieces here, so if there are more than 2 scouts, this situation is worth -2 points and more than one sergeant, miner or lieutenant gives -1.

H5 - Pieces blocked by the spy

The spy is a valuable, but at the same time very vulnerable piece. Usually it isn't moved before the marshals are removed from the game, which may happen only very late in the game. Moving it earlier often arouses suspicion because it moved but is never used. Using a not previously moved captain in this case pretty much gives away that the previously moved pieces are not captains, so the longer a moved piece remains unused the more suspicious it becomes.

A player doesn't want to arouse any interest in his spy so he wants it to be in a place where he doesn't have to move it to free an important piece. Any piece higher or equal to a major that stands next to the spy but has a higher distance to freedom might be blocked by the spy and gives a 1 point penalty to the setup.

H6 - Pieces blocked by a slightly lower piece

Pieces in the back are usually used in decreasing order of rank, the higher pieces first. When for example majors are needed because they are the highest pieces in the game, then the captains are the best targets for the opponent and therefore most vulnerable when they are moved and revealed not to be bombs. If a captain has to be moved to free a major this is always going to be a problem.

Each piece that stands next to another but is one or two ranks lower and has a lower distance to freedom therefore gives 1 point penalty. Pieces with a distance to freedom of 0 or 1 are excluded because they are the pieces that are to be sacrificed for information at the start and are not likely to block anything in the middle and endgame.

H7 - Miners on the front row

Miners are almost useless early in the game but become very valuable in the end. Any piece becomes more valuable in the end when there are less higher pieces, but the miner also wins in value because there will be relatively more bombs among the opponents pieces. It's therefore a waste to use them in the opening phase of the game. Apparantly people do this because the statistical function gives a non-zero probability of a miner being on one of the front row fields, but the database also contained many bad setups. Each miner on the front row gives 1 point penalty.

H8 - Bomb protection

Another thing that often went wrong was a flag that was bombed in with only scouts around it. This means that a lucky miner can just walk through everything to the flag. Obviously, that's a bad thing. People often mix bombs with low pieces like sergeants or lieutenants. A high piece risks attacking a bomb and miners can be quickly disabled when they take one of the bombs. Defending bombs with higher pieces leaves them vulnerable when these higher pieces are needed elsewhere, which usually happens earlier in the game than the moment the last sergeants and lieutenants are needed.

Any bomb that has at least one sergeant or lieutenant next to it gives +2 points. Each scout or spy that stands next to a bomb gives -1 point. If the bomb stands next to the flag its defense becomes especially important so if such bomb has no sergeant or lieutenant nearby this gives an additional -3 points. It's better not to have bombs near the flag than to have weakly protected bombs.

H9 - Starting pieces

The pieces with a distance to freedom of 0 or 1 are those that will be used in the opening. It's good if they are varied and don't contain too many vulnerable pieces. If there are only scouts then this becomes a problem when the opponent opens with sergeants or lieutenants. If there are 3 majors there then it will be a very uncomfortable position when they all become known. A mix of low pieces to scout and higher pieces to capture the opponent's early attackers usually works best.

- More than 2 bombs gives -1 for each extra bomb
  too many bombs block the attack paths and this probably means they will be found too quickly.
- More than 1 miner gives -1 for each extra miner
  miners are needed later.
- Both colonels give -1 point
  two colonels usually can't do more than 1.

- The flag gives -10 points
  flag in front row is possible but it requires a different playing style and much knowledge of the opponent, neither of which the computer player has.
- More than 2 majors or colonels give -2
  hard to defend when they become known.
- More than 3 sergeants, lieutenants or captains give -2
  not diverse enough.
- More than 5 scouts give -1
  not diverse enough.

Total evaluation value

The total value of the setup is now simply the sum of all the above heuristics. The setup with the highest evaluation function is used for the game.

4.3.5 Testing the heuristic evaluation

To test whether this evaluation function actually gives higher values to "good" setups than to fully random setups I evaluated several of my own setups from real life tournaments. I chose some of the setups that gave me the best results. The setups I selected can be found in appendix D.

In order of my personal preference the 6 setups get the scores: 18, 18, 20, 22, 19, 16.

My preference is of course a subjective measure, so it's not surprising that my first two setups don't get the highest value. The last one is an old setup (about 7 years old) which I would no longer use on tournaments while the others are more recent.

Next I compared the results of four different methods to generate setups

1. Fully random (each piece individually placed with equal chances on each empty field)
2. Best of 50 random
3. Best of 500 random
4. Semi random (the method described in chapter 4.3.2)
5. Best of 20 semi random (the method described in chapter 4.3.4)
6. Best of 50 semi random (as 3, the value actually used for Invincible)

Table 4.3 shows the average, best and worst scores of a set of 200 automatically generated setups. One would expect each next method to give better results than all the previous, with fully random giving the worst result and best 50 the best.

Table 4.3 – Results of the scores of 200 setups created by the above mentioned generating methods.

		"worst"	average	"best"	time
1	Fully random	-30	-3	11	0 ms
2	Best of 50 random	5	9	16	31 ms

3	Best of 1000 random	12	14	18	544 ms
4	Semi-random	-8	5	17	1 ms
5	Best of 20 semi-random	9	15	22	41 ms
6	**Best of 50 semi-random***	**12**	**17**	**24**	**99 ms**

**the method used for Invincible*

Note that the absolute values have no meaning, only the relative values are important with high scores being defined as better.

Method 2 and 3 are naturally improvements on method 1, as this is inherent to the way they are generated (picking the best of n method 1-setups) and the same is true about 5 and 6 compared to 4 but their values are added as a reference to which values can be reached by automatic setup creation.

Best of 50 random gives a result that is on average better than a single semi-random setup, so it might seem I could have saved myself the trouble of making the semi-random setup creation process, but the best setup found by the semi-random method is better than the best of 50 random setups so it is a better starting point for the best-of method. Taking the best 20x50 random setups already gives a lower average than taking 20 semi-random setups while the latter takes far less time.

My own setups score a bit higher than Invincible, but these are my most decently built setups; I often use more strange setups to confuse opponents so my own average would probably be lower than 18 and most likely comparable to method 4, or slightly worse.

5. Move Planning

5.1 Introduction

Move planning in Stratego is significantly different from classical AI examples like chess. In chess, all moves are inter-related and because of this complexity, human players only make plans a couple of moves deep. In Stratego, pieces are a lot slower, the board is larger and a lot of opponents move combinations do not affect a player's plans at all. Some may delay a plan by making some other moves more urgent, but not effectively changing them. This difference is shown in chapter 2.1.5

Human Stratego players think in plans, not in moves. Stratego is not a game ever to be solved, both because of the immense number of moves and game states compared to solved games but also, because of the uncertainty because of the incomplete information. The stress is therefore not to find the best move, but rather to find a good move. The traditional approach is to find a good move by finding a move that leads to a good position in the next limited number of moves. This is difficult for Stratego because many strategic moves have only effect after a number of plies that is far beyond the horizon of any exhaustive search technique. In my approach, a move is good when it contributes to a good plan without neglecting urgent other plans (like moving a piece out of danger or protecting the flag). Another move may be better when it contributes to a better plan.

The basis of "Invincible" is a collection of plans that give a value to every possible move. The move with the highest total contribution to all the plans is selected. These plans evaluate only a single ply, so no deeper searches are made. The algorithm will be further explained in chapter 5.2. Because this collection can never be complete it is impossible to claim that the best plan in every situation is among them and therefore it can't be expected that the best move is always found. It is however possible to find a good move, and the better plans are added the better moves can be found find with them. Because plans have a well-defined goal, the complexity of finding how well a move contributes to a plan is usually constant, or at most dependent on the (constant) size of the board. This is a huge improvement over the exponential complexity of the min-max algorithm. Since by far most of the moves that are investigated by min-max are consumed by sequences that don't contribute to a plan that significantly changes the board position this huge improvement in complexity is not necessarily an equally huge loss in correctness.

Advantages:

1. "Unlimited" search depth (5.1.1).
2. Possibilities to bluff (5.1.2).
3. Possibilities to coordinate attacks with multiple pieces. (5.1.3).
4. Focus computation on interesting situations. (5.1.4).

Disadvantages:

1. "Invincible" knows only what it's told. (5.1.5).
2. Comparable information is calculated multiple times. (5.1.6).
3. Game specific knowledge needed (5.1.7).

These advantages and disadvantages will be discussed in the following sections.

5.1.1 Advantage: “Unlimited” search depth

The search depth is unlimited because the depth is defined by the useful depth to detect and complete a plan. Because of the directional rather than exhaustive search this depth can easily be reached. To see whether a miner can reach the flag, one only needs to find a path from one point to another on a 10x10 board which can be seen as a graph search on a graph with 100 nodes. This is a relatively small search problem that can easily be performed thousands of times without noticeable delay on modern hardware. As a comparison, min-max for 20 plies with an average branching factor of 10 is a search problem over 100.000.000.000.000.000.000 nodes and this is not an unreasonable average scenario in Stratego: 20 plies is often not even enough for simple plans.

5.1.2 Advantage: Possibilities to bluff

Bluffing can be implemented by defining what makes moves good for bluffing. Usually this means performing a normal plan with the wrong pieces. If “Invincible” can find a piece of which the opponent may believe it is the right piece for a certain plan and he uses it for that plan, then the opponents belief that this piece has a rank that in reality it doesn’t have is strengthened. This behavior is easy to implement when moves are chosen based on plans but very hard/impossible to include in a system that finds all the effects of a limited sequence of moves.

5.1.3 Advantage: Possibilities to coordinate attacks with multiple pieces.

There are not many things in Stratego that a single piece can do, but plans that require multiple pieces generally also require many more moves. An exhaustive search of 12 plies allows to find plans that require 1 piece to move 6 fields, or 2 pieces to move 3 fields. That means both pieces must be already very close to where they are needed before a combination that requires these pieces can be found. If this is defined as a plan, then this plan can find the location of the required pieces and guide them where they have to be also as a search over a 10x10 graph.

5.1.4 Advantage: Focus computation on interesting situations

The computational is used more specifically for computing interesting aspects of a board position rather than for computing all the possible move sequences, out of which most are not going to be interesting.

5.1.5 Disadvantage: “Invincible” knows only what it is told

Because there is no exhaustive search, everything that is not formulated as a plan will be completely ignored. If there is no plan telling “Invincible” to move a known low piece away when a known higher piece is approaching it then it will not do so. There is however only a limited number of useful things to pay attention to on short-term planning, and since the long-term plans can’t be seen by exhaustive search anyway this is only a disadvantage in terms of programming time and not in playing strength.

5.1.6 Disadvantage: Comparable information is calculated multiple times

Many plans use similar information, but can’t use each others results if the calculation is split in independent sub-problems. This means that sometimes the same calculation is done multiple times. This is of course inefficient, but because time isn’t really an issue this doesn’t prove to be a big problem.

5.1.7 Disadvantage: Game specific knowledge needed

This is of course the real catch. So much of the search can be ignored only because of knowledge specific to the domain (in this case the game Stratego). A human expert in the domain will have to decide which plans should be added. In my case this is simplified because I myself spent 6 years of my life playing Stratego rather seriously and have 2 world championships titles and holding the nr 1 ranking position uninterrupted since 2001 to justify drawing on my own experience for this. This could of course also be done by using someone else’s expert knowledge. For testing and evaluation I will use the opinion of other Stratego experts. In either case, a lot of the development time will go into formalizing human experience into computer-readable algorithms.

5.2 Algorithm

In this chapter I will discuss the algorithm for selecting moves in detail. First the plans are introduced as black boxes in section 5.2.1 and in section 5.2.2 the inner working of a typical plan is explained.

5.2.1 Plans as black boxes

In my implementation, a Plan is specified as an entity that can give a value to every legal move in a legal position based on a clear goal. This goal must be a relatively small sub problem of the game. This splits the complicated problem in several smaller that should be easier to solve. Because there will be many plans, the complexity of each plan must be low, in the order of the number of fields or the number of pieces, both of which can be

seen as constants for the game of Stratego as neither value can be very high. This more specifically excludes the use of an exhaustive search over all possibly move sequences.

```
public interface Plan
{
	/* gives a value to the given possible move
	 * The range of the value should reflect the importance of the plan, with more
	 * important plans giving higher values
	 * This method is called once per move for each of the possible following moves
	 */
	public double moveValue(Move m);

	/* updates the plan after a move is actually executed
	 * This method is called once per actual move
	 */
	public void recalc();

	/* determines whether the plan is useful on the given board position
	 * if the plan is not active it should return values of 0, though moveValue is still
	 * called in case the move activates the plan.
	 */
	public boolean isActive();

	/* gives a name to the plan: only for display purposes */
	public String name();
}
```

Figure 5.1 – The interface Plan.

Figure 5.2 shows how the final value of a move is determined. The move is given to each of the existing plans for evaluation and the values are added up to give the final value of that move. The move that got the best total evaluation score is selected as the best move and executed.

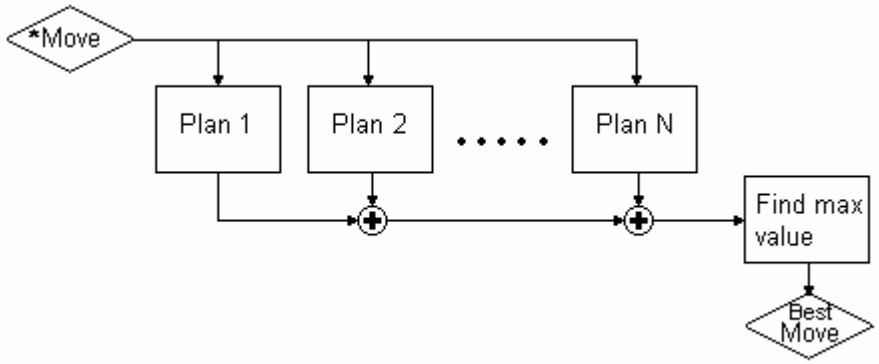


Figure 5.2 – Finding the best move.

All the possible plans are generated at the start of a new game, so there is no procedure to add or remove plans during the game. Plans that lose their function after a certain event (the capture or loss of a piece for example) or plans that require a certain event (discovery of a certain piece) to become useful can be in sleeping mode, in which case they give the value 0 for every possible move.

At the beginning of the process of finding the best move in a given situation, the method recalc is called for all plans. This allows the plans to make calculations depending on the starting board position. If a plan is for example gives positive values for moving a miner closer to a possible flag, then the starting distance is the same for all the possible moves and can be calculated only once before the possible moves are being evaluated.

The plans do not communicate between each other, and as such have no knowledge of the values given to the current move by other plans. The values given by plans should therefore reflect the real importance of the move from the point of view of that plan. Important plans should give higher values to moves that contribute to these plans than unimportant plans. Moves that neither contribute nor counter a plan should receive the value of 0. This ensures that if an important plan gives a non-zero value for a move this will most likely dominate the outcome of the sum for all plans, while if no important plan finds a useful move, the lesser important plans will determine the outcome. It is also possible for a plan to give negative values for moves that should certainly not be made, for example when the plan responsible for defending the flag finds that the flag is adequately defended he might consider most moves neutral, except those that weaken the defense. The latter will then be given a high negative value (because defending the flag is very important) so that these moves will never be selected even if they support a lesser important plan. All the other moves are equal for this plan and will receive a value of 0, so that the selection from these moves will be made by other plans that can distinguish them.

The selection is based not on plans but on the total value, so while the important plans can’t find relevant moves the lesser important plans dominate the outcome, but as soon as an urgent situation is detected the higher values of an important plan that has become relevant will dominate the outcome and in this way temporarily take control of the selected moves.

5.2.2 The inside of plans

To find the value for a move, plans are given access to all the useful information about the current board situation. In particular, they can see:

- The current board, with the position of all the pieces.
- The graveyard, containing the captured pieces with their rank and color.
- The dynamically determine values of the pieces on the board.
- The history of all moves previously executed.

They do not know whether other plans are active in this situation, or even whether or how many other plans there are. Because of this, it is important that the values given by plans are realistic in relation to all other plans in all imaginable situations. This seems like a very big problem for the programmer but there are more or less fixed values to which the plans can be related: the piece values.

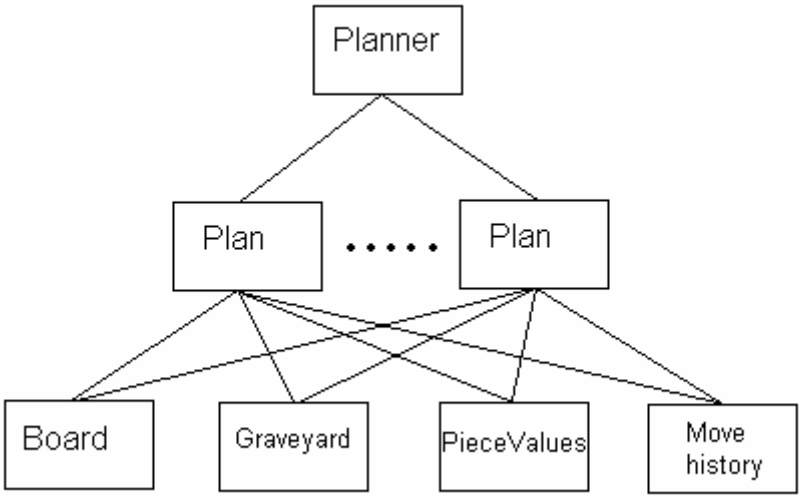


Figure 5.3 – The class structure of plans.

Plans that are directly related to the capture or the loss of a piece can be given values relative to the value of this piece; the relative factor should then be decided by the success chance of the plan if the evaluated move would be executed. The more likely it is that the move will actually result in the capture of an opponent's piece; the higher percentage of the piece's value should be given as value to the move. For example, directly attacking a piece will always result in a battle and should have the expected average outcome as value, while approaching a piece that may be attacked should give only a percentage of the expected outcome of the battle because it is not sure that the battle will take place (the opponent may move away). If it is easy for the opponent to move away then this might even be a very low percentage. If the opponent seems trapped it might be a high percentage.

Plans that do not directly relate to the capture of pieces might be for example strategic plans or plans dealing with defending the flag. These can be compared to the other plans and given a relative value according to their relative importance. Flag defense plans are more important than the loss of any piece and should be given a higher value so that the loss of a piece is only prevented if this can be done without a large chance of losing the own flag. Strategic plans are generally less important than plans that give direct results because their goal and chance of success is often very vague. These plans should be given low values. They can however be compared with other strategic plans and given higher values than obviously inferior strategic plans.

With piece values of approximately 10 to 100, strategic plans are generally given values of 2-5 while flag defense plans get values of 200-300. These values can be positive if something can be gained or negative if something can be lost.

The plans execute only a single move and then look whether the move contributed to the goal, made it more difficult to reach the goal or didn't affect it at all. The plans should never execute more plies than the one they are given to evaluate. They do know the battle logic and know how pieces can move so that they can calculate possible paths for pieces from one location to another.

5.3 Plans used

A careful selection of these plans is needed. Too few plans will results in obvious blunders but unfortunately it is not so that more plans are always better; if 2 plans both give a positive score for the actual capturing move of a piece then this move will get a score that is twice as high as it should have been. In general there shouldn't be 2 plans that award a score to the same goal. There can be 1 plan that brings a piece in position and another that awards a value to the actual capture, but then the first plan should give a score of 0 to the capture itself.

Plans should be added in order of importance. Basic plans that give positive values to moves that move pieces out of danger when they are threatened by the opponent or that capture pieces when a profitable opportunity is detected should always be added. Other plans that for example formulate attack strategies are of lesser importance. There is no need adding such plan when the program can't even capture a piece that moves directly towards it.

Different selections of advanced plans and/or different implementations of basic plans can be easily used to create programs of variable playing strength or with a different playing style.

To prevent overlapping responsibilities it is useful to define different types of plans. The plans can be grouped in 3 groups with different tasks. There are tactical plans that define tactical movement for encounters, strategic plans that regulate the large movements of pieces over the board and general plans that implement general heuristics about the game.

Tactical plans should detect situations in which pieces are already close to each other. They should not bother about how the pieces get to such situations. These plans are generally hard to program, but easy to value. Hard to program because they are exact, there is usually only 1 good move in such position, but easy to value because they often lead to capturing or saving a piece and can thus be related to the value of the piece involved. Problems like these could possibly be solved by exhaustive search on a local position of the board, but as shown later, even tactical situations may arise when the pieces are still many moves away from a confrontation.

Strategic plans should move pieces to positions where they may be useful. They shouldn't bother about what happens when the pieces arrive to their destination. This type of plans is usually easier to code, because only the general direction is important and the order in which the moves are done is not. However it is much harder to give them a proper value. Because they have no tactical value they do not necessarily imply a specific result. Their task is to create situations in which a profit can be made and are thus one step further away from the real profit than the tactical plans. They should generally have lower values than tactical plans and their value should also depend on the amount of profit that can possibly be gained from the situation they try to achieve.
Giving them values relative to the profit that can be gained is important to give priority to strategies with potential over strategies that can at best lead to the capture of a low ranking piece. Giving them values lower than the tactical plans is important to make sure the tactical plans dominate the values as soon as they recognize the situation.

General plans should award even lower values. They are more like guidelines that take effect only when no other plans give a decision. They may decide among equal moves but shouldn't disturb when good moves based on a specific plan are found.

5.4 Tactical plans

A lot of the tactics in Stratego are caused by the two-square rule. Because a piece can be moved back and forth only a limited number of times between the same two squares regardless of the situation it is in, there are countless situations in which a piece can be lost because of inattentiveness to this rule. Likewise, a player can use this rule to his advantage against careless opponents. Readers that are unaware of the effects of this rule are advised to read appendix A2 on the two-square rule before reading this chapter.

5.4.1 Immediate captures

Name	ImmediateCapturePlan
Value	Value of a piece
Description	This plan returns the "expected" profit of an attack on an enemy piece. If the piece is not known it is of a pessimistic nature, weighing possible negative results with a higher value than positive results. It also takes into account a penalty for revealing the rank of the capturing piece, so taking a scout with the unknown marshal is considered a negative result. It does not take into account where a piece will end up if he wins the attack.
Initialized	One per game (1).

This is one of the most basic plans “Invincible” contains. It only gives non-zero values to moves in which an opponent’s piece is attacked and should be the only plan that gives a value to such moves. For each of the ranks the enemy piece can have it calculates the profit of attacking it, given the rank of the attacking piece and multiplies this with the probability of the piece having this rank.

The profit can either be the value of that piece minus the value of revealing the own piece when the attack is won, the value of revealing the enemy minus the value of the own piece when the attack is lost or the sum of those when the pieces have equal rank and are both lost. The value of revealing a piece that was previously revealed is always zero.

It is possible that an attack that is certainly won still has a negative value because an important piece will be revealed by this attack. Likewise it’s possible that a near certainly lost attack is profitable if it can reveal a high rank for a low price. For more details about this read chapter 3 on the value of pieces and the value of information.

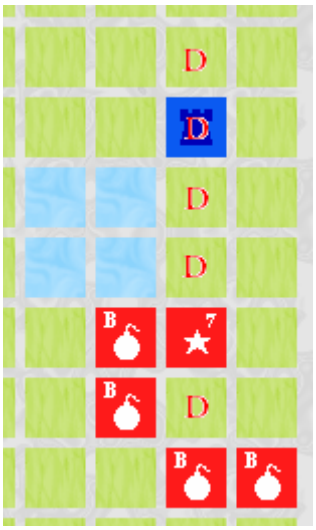
This plans sole responsibility is capturing moves, so if it decides an attack is not profitable it will not tell “Invincible” to move away from it, it only gives a negative value for the attacking move.

5.4.2 Tactical defense

Name	TacticalDefensePlan
Value	Value of a piece, decreasing over distance
Description	This plan looks after the safety of “Invincible”’s own pieces. It gives a negative value to moving next to dangerous pieces but it also looks ahead to dangerous situations from single-piece attacks. Given no interference from other plans it can ensure that a piece is never lost on the two-square rule, in some situations reacting when the attacking piece is still 11 squares away. It is only active for pieces that have already moved or are known.
Initialized	Once for every friendly piece that can move, except marshal (32).

The safety of “Invincible”’s own weak pieces depends not only on seeing threats when they are directly next to his piece. Because of the two-square rule it can often be too late then to rescue a piece that could have been saved if the situation was detected earlier.

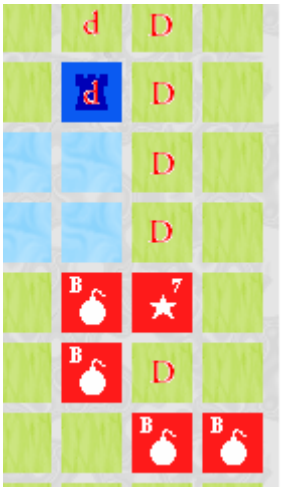
This plan calculates for every piece in its care which enemy pieces are potentially dangerous. These are marked as enemies and their locations are compared with the piece to be defended. There are 3 basic dangerous situations that are recognized by this module:



The biggest danger comes from pieces on the same row or column. In the situation in figure 5.4 the dangerous fields for the red major (7) are marked with the red "D". If any of its potential enemies are on any of these squares then he experiences a threat from that enemy. Moves that keep this piece in suchlike position will be given a negative value. In this position, moving up, down or not at all will not change the situation so these moves all receive a penalty. Moving to the right will move the blue dangerous zone away from the blue piece and will thus be the best choice according to this plan.

Figure 5.4 – The red 7 is threatened by the blue piece.

Of course if the blue piece is really hunting the red major he will try to stay in its dangerous zone by also moving to the right. The only response for red is then to move back to the left, after which blue will do the same. Now there occurs a repetition of moves over 2 squares and the piece that started this repetition will also be the first to end it. In this situation that means either the loss of the red major or its safety from the blue piece. Apparently, the threat of the blue piece might have been detected too late because the red piece still may be lost. The resulting move is still good whether the repetition is won or lost, but this check alone is not enough to ensure the safety of pieces.



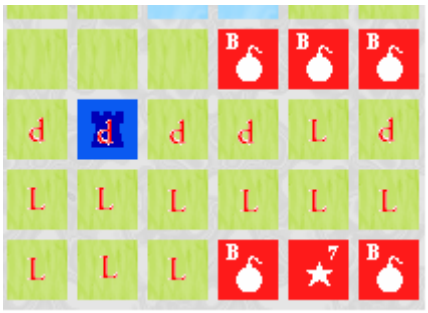
If the possible enemy came from the right, then the red piece can move back in the direction where the blue piece came from. If however he came from the left then red still has to avoid the threat by going to the right, but now he starts the repetition and is already lost.

This means that the squares marked with "d" are also dangerous. Action should be taken as soon as a blue possible enemy moves to one of these squares since red can't wait for the blue piece to move to a square marked "D". In general, squares on a row or column directly next to the piece to be defended are dangerous if the piece can't move to this column or row.

Figure 5.5 – The red 7 is threatened by the blue piece.

If an enemy is on a square from which it can go to a field from which it can see a lower piece in a straight line and this can't be answered by a symmetric move back to the row or column this enemy came from then the piece also experiences a threat from this enemy. If in figure 5.5 the blue piece is also on a dangerous field and the red piece should react by moving to the right to move the blue piece out of its dangerous zone. This is a move that many beginners would easily overlook and that most likely won't be found by

exhaustive search but that is absolutely necessary in this position if the red major is not to be lost.



The last situation is when the piece to be defended has a forced move to make to have a chance to be rescued. In figure 5.6 the red major can't stay in his trapped position because it will result in him being captured when the blue piece comes closer. If the enemy is on any of the squares marked with "L" then the red major is already lost (provided it can't pretend to be a bomb).

Figure 5.6 – The red 7 is threatened by the blue piece.

Therefore every field from which an enemy could move to a field marked with L is also dangerous. This increased dangerous zone happens only when the piece to be defended has only one adjacent square it can move to. If in the situation in figure 5.6 the red major moves up it gets more than one empty adjacent square and gets a normal dangerous zone around him. The blue piece is then not in the dangerous zone any more because if he moves to the same row red can answer by going to the row blue just came from. In figure 5.6, moving the red major one square up is the necessary move to ensure its safety.

Note that the presence of higher friends can block the route of the possible enemy, in which case the danger is relieved. Which pieces can act as higher friends can be different for every potential enemy so every field on the path between an enemy and its prey must be checked for possibly friends of the prey.

Note also that I do not check whether there are actually only 2 fields to move to and whether the passage away from the potential enemy is blocked. If the repetition can go over more than 2 fields, then this move will be found as a solution to move out of danger, if there is really no other move then a move away from the potential enemy will be selected because the danger from an enemy is considered slightly less when the enemy is further away. Even though there might be a passage since this is never checked this should always be the last resort. When this is done, the piece stays on the same row or column as the enemy and will always lose the repetition later when it tries to get to off. After such move, the only option is to keep running until there is enough space to get a repetition over more than 2 fields.

Even though this might not lead to the loss of a piece, it is in principle a bad thing if the opponent can force a piece to a move to a completely different location. If the piece needs to be moved there it can be done without forcing, if it shouldn't then it is not preferable to be forced to do this . This is therefore not only important to prevent losing pieces but also to ensure that they can keep the positions chosen for them.

All this assumes that if the enemy can't be blocked, moving the piece away from danger is the only option there is. This naturally means that using this plan to defend the flag or bombs is senseless because they can't move anyway. It also means it is not necessary to

bother about pieces that still can pretend to be a bomb. If such piece would be moved at the first hint of a possible threat a lot of information would be given away. Firstly, it would tell it is not a bomb and the secondly that it is lower than the approaching piece. In this case it's much better to go on pretending that it is a bomb. It is also not entirely accurate to use this plan for the defense of scouts since it assumes that the piece to be defended can only move 1 step at a time but is used for this purpose anyway to prevent revealing their identity by defending them differently.

Lastly, this plan is not used for the marshal since the only enemy of the marshal is of a different nature than normal enemies. Normal enemies win both in defense and in attack, but the spy only wins when he attacks. Therefore the defense of the marshal is also a bit different.

5.4.3 Tactical attack

Name	TacticalAttackPlan
Value	Value of a piece, decreasing over distance
Description	This plan is the reverse of the TacticalDefencePlan. It gives positive values to moves that correspond to favorable approach lines in order to capture pieces on the two-square rule. It is initialized on an enemy piece and looks out whether this piece can be captured. Possible pieces it can use to attack are pieces that are higher than all the ranks this piece still could be or pieces that could have such a rank (bluff)
Initialized	Once for each enemy piece (40).

The tactical attack plan uses the same techniques that were discussed for the tactical defense the other way around in order to catch the opponent on a mistake. It tries to move pieces into dangerous zones of pieces that are or are likely to be a good prey for the attacking piece.

Unfortunately, because of the incomplete information it can't be an exact opposite. Now the rank of the attackers is known and the rank of the defenders can be uncertain. Also, if in one case it would be wise to avoid moving next to an unknown piece it is not reasonable to expect the opponent to be equally careful. The plans are therefore alike, but not the same.

5.4.4 Defense against gamblers

Name	LottoDefensePlan
Value	Fraction of value of a piece
Description	It is possible that enemy pieces blindly hit unknown unmoved pieces. Apparently the opponent thinks he can afford losing this piece and it's likely that he will go on after a successful capture. This leaves unmoved pieces directly next to such a piece in bigger danger than if this piece would be a careful piece. This plan checks for each piece whether it borders such berserking piece that could capture him. This gives a higher priority to capturing such pieces when they have the possibility to make profitable captures and might make "Invincible" decide to capture when he would normally prefer to keep the rank of a higher bordering piece secret.
Initialized	Once for every friendly piece (40).

The term Lotto Stratego is most commonly used to describe the action of hitting unknown unmoved pieces with high ranks hoping that they are not bombs but valuable pieces to get an easy lead. Unless a player has a very good intuition, a very predictable opponent or a lot of luck this generally leads to losing the high piece and often the game if this is done too often, but if applied within limits it can be the key to victory. The key to doing this is to only use pieces that can be lost without losing the game. It is only true lotto when the piece used for attacking is too valuable to lose, though this of course often leads to arguments about whether or not the piece is important or not. In this context I use the term for any attack on an unknown unmoved piece even if it is with a miner or a clearly useless piece. This lets me tag pieces the opponent considers fodder. If such piece wins an attack it is more likely that this piece will hit more unmoved pieces than can be expected of a piece that didn't "lotto" before.

The tactical defense plan assumes that a piece is safe as long as it can pretend to be a bomb, so it does not see threats created by a "lotto-ing" piece. When such piece is bordering several slightly lower pieces then taking out this lotto-ing piece should have a higher priority than normal even if it means revealing a bit too much information on this capture. If there is nothing profitable he can capture then it would be better not to reveal information by capturing.

If no capture of the lotto piece is possible "Invincible" might have to consider moving a previously unmoved piece out of danger, even though that reveals both that the piece is not a bomb and that it is afraid of the attacker. This would be a bad move if the opponent is not likely to risk an attack, but might be necessary when it is likely that he will.

This plan therefore gives a penalty for every unknown unmoved piece bordering a previously "lotto-ing" enemy that should preferably not be attacked by this piece. This can be either because it has a lower rank (a lotto piece is always already known) or because a piece is still unknown and too valuable to be revealed by the lotto piece.

This penalty is always lower than would be awarded by the tactical defense plan when the piece already moved or is already known. In the latter case it is much more likely that the piece will be attacked than in the case of the lotto piece that might decide to stop "lotto-ing" or go on in another direction.

5.4.5 Defend marshal

Name	DefMarshalFromSpyPlan
Value	Value of marshal
Description	Defending the marshal is different from defending any other piece because the marshal can only be captured when the opponent attacks with his spy while the other pieces can be lost both when attacking and when attacking themselves. Therefore the way to keep the marshal in safety is a little different from other pieces.
Initialized	Only for the marshal (1).

Attacking the marshal is not as easy as attacking any other piece, since the only piece that can capture him is the spy. It is therefore not necessary to do all the checks that are done for other moving pieces in the tactical defense plan. There should be some checks though to prevent the marshal from foolishly stepping next to a possible spy.

This plan only checks if after the move, the marshal is next to a piece that could possibly be the enemy spy and if so awards a negative value depending on the likelihood of it being the spy and the current value of the marshal.

5.5 Strategic plans

Strategic plans represent long-term strategies. They often have no clear result in mind, but work towards improving the board position in a general way.

5.5.1 Strategic attack

Name	StrategicAttackPlan
Value	Low
Description	Gives a low positive value for a move that brings an important piece towards a good strategic attacking position.
Initialized	One for each game (1).

This plan is responsible for bringing "Invincible"'s pieces in good strategic positions.

5.5.2 Strategic defense

Name	StrategicDefensePlan
Value	Low
Description	Gives a low positive value for a move that brings an important piece towards a good strategic defensive position. Also checks whether vulnerable pieces aren't too close to each other in places reachable by enemy pieces.
Initialized	One for each game (1).

This plan is responsible for bringing pieces to good defensive positions. Where the tactical defense plan only looks at single pieces, this plan looks for combinations of pieces, trying to prevent situations in which the opponent can make moves that attack multiple pieces at the same time.

5.5.3 Trap pieces

Name	TrapPiecePlan
Value	Fraction of value of piece.
Description	Coordinates the cooperation between multiple pieces in order to capture an enemy piece, viewed from the perspective of the enemy piece. It first limits the mobility of the piece by looking how many squares it can reach in less moves than one of his enemies and minimizing this. When no move can be found that makes this area smaller, a second piece is brought nearer. When it is close enough it assumes that the TacticalAttackPlan and ImmediateCapturePlan take over for the actual capture.
Initialized	Once per enemy piece (40).

Trapping pieces is a strategic maneuver that requires a combination of two pieces to capture one enemy. This plan only attempts to trap pieces for which there are two pieces that are, or could likely be, pieces that can profitably capture the enemy piece.

An unknown marshal is not a good choice to capture a low sergeant and a known colonel is obviously not a good choice to trap a (higher) general.

Trapping a piece is a two step plan, first its freedom must be limited since there is no use in moving 2 pieces to the left side of the board if the target can still escape and move to the right side of the board. Secondly, when the freedom can not be further limited with a single piece, a second piece must be brought nearer. Once the 2 pieces are close enough this plan assumes that a tactical plan completes the job. This happens automatically because the values given by the tactical plan would be of a higher order than the values for moves suggested by the strategic plan.

The trapping of a piece is always calculated from the point of view of the piece that is attempted to be captured. The freedom of this piece is defined by the number of squares it can reach sooner than any of his enemies.

The pieces that Invincible uses to trap are not allowed to move next to pieces that could be even higher than they are but in this calculation I assume that higher pieces don't move because otherwise the possibilities are endless since it is theoretically also necessary to check for enemies of enemies of enemies and so on. This would practically result in an exhaustive search, which was proven unfeasible.

A move that decreases the freedom of a piece that can be trapped is given a positive score, and furthermore a move that brings the second-closest enemy closer to the piece to be trapped is also given a positive score. The second value is always lower than the first for the same plan so that the priority is first to decrease the freedom and then to approach with the second piece. Also, the higher the likely rank of the piece that is to be trapped the higher the values for moves that contribute to trapping them.

5.6 General plans

It proved that several plans were needed that don't really depend on a specific situation. They use no positional information and thus can't be grouped as a tactical or a strategic plan. These plans fall in the last category of general plans.

5.6.1 Randomness

Name	RandomPlan
Value	Very low
Description	Each move is given a low random value to make a random selection between equal moves.
Initialized	One for each game (1).

In the beginning all moves are equal. Invincible knows nothing about the opponent so there is no real reason to find any of the 6 opening moves better than another. In this case "Invincible" will make a random selection from all the equal moves. This can easily be achieved by adding a very low random value to every move. By setting this value between 0 and 1 all moves with less than 1 difference in total value are considered more or less equal and can all be selected by the move planning module.

5.6.2 Move penalty

Name	MovePenaltyPlan
Value	Low
Description	Gives a penalty to giving away information by moving (for example moving a piece that could have been a bomb before it was moved, or moving an unknown scout more than 1 square and thus revealing its identity). The more moved pieces "Invincible" has on the board, the higher the penalty for moving another.
Initialized	One for each game (1).

Sometimes simply moving a piece gives away information. If before the move it could have been a bomb, then moving this piece equals telling the opponent that this is not a bomb. If the piece is unknown and it is moved more than one step the opponent can make the conclusion that it is a scout, so this move reveals the identity of the unknown piece and should be recognized as such.

This penalty should be low, because it is impossible to play well by moving around just a single piece until it dies. Moving a lot of pieces however seriously limits the ability to attack because those pieces that can no longer play bomb have to be defended. The penalty for moving an unmoved unknown piece therefore increases when there are more pieces on the board that have already moved. This seems even more logical since there are more moves that can be done which don't give away new information to the opponent.

It effectively limits the number of pieces that are active at the same time while not limiting "Invincible" to playing with just a single piece.
Of course, moves that attack an enemy piece should not be given this penalty since their identity would be revealed anyway and the penalty for this is already taken into account by another plan.

5.7 Other plans

There are more plans that are not described in this chapter, but they work similar to those that are described. In the current version of Invincible there are 18 different types of plans and 173 instances of plans (some plans are instantiated for all 40 friendly or opponent pieces, or for a selection of them).

6. Guessing ranks

6.1 Introduction

An important aspect of the game Stratego is the uncertainty of the ranks of the opponent's pieces. At the start of the game each player has 40 pieces of 12 different ranks. Which piece has which rank is hidden from the opponent. When a piece ends up in the same square with an opponent piece a battle takes place and the highest rank wins. The rank of the winning piece is revealed to the opponent and the losing piece is removed from the board and returned to the box where it is visible to both players so that at any time they can check which pieces are still left on the board. When pieces of equal rank meet, they are both removed from the board.

In Stratego, information is very important. Having more knowledge about the enemy army often compensates for having fewer pieces. Initially there are no invulnerable pieces and thus every attack can fail. The players have to find out where the opponent's high pieces are in order to plan their attacks. Also it helps to know vulnerable targets that can be attacked.

A naive approach would be to assume that only the pieces revealed in a battle are known and all others have equal probabilities to all remaining unknown ranks. Pieces can revealed by attacking them with expandable low pieces but when a player does that he wants to maximize his profit by only attacking pieces that he suspects to be interesting. Also, if a player is afraid of all unknown pieces his opponent will be able to guide all his moves and dominate the game, but if he doesn't fear any he will lose a lot of pieces. A human player can usually estimate which pieces might be dangerous and which probably are not, so there must be a way to teach this to the computer.

6.2 The model

Programs using exhaustive search are usually limited to making a single guess about the most likely rank of a piece. This is however often insufficient to model the knowledge that might be collected during the game. If a bluff is called by moving a general next to an unknown approaching piece and it isn't captured, this doesn't reveal the rank of the unknown piece, but this piece can be marked as probably not being the marshal. There must be a way to store that information so that if the same piece is involved in a situation later in the game somewhere else on the board this information can be used.

I chose to use a probabilistic model where pieces have 12 probabilities: the probability for each of the 12 different ranks that this piece has this rank. Since they are probabilities they all must be between 0 and 1 and the total must always equal 1. Once the piece is revealed the probability for his real rank is set to 1 and for all others to 0. Heuristics are used to gather information from every move the opponent makes. This is represented by updating the probabilities of all pieces. The same heuristics are also applied to "Invincible"'s own pieces based on the moves "Invincible" makes to be able to guess

what the opponent can probably guess about his ranks. Even though “Invincible” knows what rank his own pieces has he tries to derive what his opponent would think about them, not using his own secret knowledge. This is important because if a piece has behaved very obviously it shouldn't be considered unknown. Also if a piece has behaved like a different rank before it can be used very well for bluffing by continuing as if this piece has the credible other rank. The same bluff would not be believed if it is unlikely the bluffing piece has this rank.

Because the algorithm must be applicable to both sides, the heuristics can use neither side’s hidden information. Only the common knowledge that is known from previous moves can be used. This makes sense because if piece X runs away from a piece of which his opponent knows it is a captain it doesn’t mean that X is afraid of a captain. Only if both players know the rank of this piece such conclusion can be made.

This model represents an interesting normalizing problem. As the numbers represent probabilities they are only meaningful if the probabilities for a single piece add up to 1 and if the total probability of a rank on the entire board adds up to the number of pieces of that rank still on the board. A piece can't have a 50% probability for 12 different ranks, nor can each piece have a 40% probability of being the flag when it's known there is only 1 flag among for example 20 pieces. Normalizing in 1 direction is easy enough, but when the probabilities are right for the number of ranks on the board and the pieces are normalized, this invalidates the probabilities over the board.

However since at each step some volume is taken from the high areas and distributed over the low areas the changes that are made get smaller after each iteration until a target-accuracy is reached. This means that the algorithm always converges to a model in which the probabilities are approximately normalized. It is therefore freely possible to update one probability on a piece and later normalize the whole board to calculate the effect this has on other pieces. If piece A has a higher probability for rank R that means the probability of piece A having any other rank necessarily decreases. It also means that the probability of any other piece having rank R decreases, even if there are multiple pieces of that rank. Reasoning one step further, it also means that the probability of a piece other than piece A having a rank other than rank R increases. Once a piece has only 1 non-zero probability left it will stay 1 and the other probabilities 0, if only x pieces have a non-zero probability for a certain rank and there are x pieces of that rank left they all have a probability of 1 to have that rank and thus 0 for any other rank. If there are no unknown pieces left of a certain rank, all probabilities for this rank lower than 1 should be 0. All these effects follow automatically from the chosen model so no special rules are needed to ensure logical conclusions from the available unknown ranks.

The pseudo code for normalizing the matrix is as follows:

```
updated := true
while updated
        updated := false

        for each rank r
                sum := 0
                for each piece p
                        sum += p.prob(r)

                if sum > 1+e or sum < 1-e
                        updated := true
                        for each piece p
                                p.prob(r) /= sum

        for each piece p
                sum := 0
                for each rank r
                        sum += p.prob(r)

                if sum > 1+e or sum < 1-e
                        updated := true
                        for each rank r
                                p.prob(r) /= sum
```

Figure 6.1 – pseudocode for normalizing the probability matrix.

Example

Suppose there are 3 pieces that can have either one of 3 possible ranks. At the start of this example their probabilities are given by:

	Piece 1	Piece 2	Piece 3	Total
Marshal	0.3	0.4	0.3	1.0
Captain	0.6	0.2	0.2	1.0
Miner	0.1	0.4	0.5	1.0
Total	1.0	1.0	1.0	

Each piece has a total probability of 1 and the probability of each rank appearing on the board is also 1. Now this player makes a move that gives arouses a strong suspicion that piece 2 is the marshal. The heuristics say that it multiplies the probability of piece 2 being the marshal by 2.

	Piece 1	Piece 2	Piece 3	Total
Marshal	0.3	0.8	0.3	1.4

Captain	0.6	0.2	0.2	1.0
Miner	0.1	0.4	0.5	1.0
Total	1.0	1.4	1.0	

This naturally invalidates the probabilities of piece 2 and those of the marshal on the board so the matrix effects of this change must be distributed over the whole matrix.

First, the probabilities of piece 2 are normalized. Each element in the column belonging to piece 2 is divided by the total volume of that column (1.4)

	Piece 1	Piece 2	Piece 3	Total
Marshal	0.3	0.57	0.3	1.17
Captain	0.6	0.14	0.2	0.94
Miner	0.1	0.29	0.5	0.89
Total	1.0	1.0	1.0	

This looks even worse, now all the horizontal totals are wrong. All totals have an error less than 40% though, which was the worst error in the previous table. The next step is to normalize the rows. Each element is divided by the total of the row it is in.

	Piece 1	Piece 2	Piece 3	Total
Marshal	0.26	0.48	0.26	1.0
Captain	0.64	0.15	0.21	1.0
Miner	0.11	0.33	0.56	1.0
Total	1.01	0.96	1.03	

The totals of the columns are now not equal to 1, but again the errors are smaller than after the previous step. One more normalization over the rows results in

	Piece 1	Piece 2	Piece 3	Total
Marshal	0.26	0.5	0.25	1.01
Captain	0.63	0.16	0.20	0.99
Miner	0.11	0.34	0.55	1.0
Total	1.0	1.0	1.0	

Now all errors are less than 1% which is a nice target accuracy for a manual calculation. Comparing this table with the starting situation:

old	Piece 1	Piece 2	Piece 3	Total
Marshal	0.3	0.4	0.3	1.0
Captain	0.6	0.2	0.2	1.0
Miner	0.1	0.4	0.5	1.0
Total	1.0	1.0	1.0	

new	Piece 1	Piece 2	Piece 3	Total
Marshal	0.26	0.5	0.25	1.01

Captain	0.63	0.16	0.20	0.99
Miner	0.11	0.34	0.55	1.0
Total	1.0	1.0	1.0	

We see that the probability of piece 2 is increased only from 0.4 to 0.5, while we initially multiplied it by 2. This is normal, because I only added more volume in one field after which the total volume had to be decreased again. The eventual increase in one field must come from an equal decrease in other fields. Eventual effects are therefore never as big as the initial change. This is something to take into account when defining the heuristics.

We also see that the probabilities on the same row and column as the entry that was increased all decreased, and that all other entries increased. These are logical conclusions from the change to a single field, as explained before.

6.3 Heuristics

6.3.1 Starting position

A very important heuristic in guessing the ranks of opponent's pieces is to look at the starting position of his pieces. The spy is often on the side where the general was discovered, and a back row piece is rarely the marshal.

A player using completely random setups will not win many games, so it is usually safe to assume the opponent thought about where he put his pieces initially. This means his flag is probably in an easily defendable place in the back of the field, often surrounded by bombs. His high pieces are usually more near the front and his spy is often near the general. Sergeants are often found near bombs to stop miners. These are only some of the many heuristics people use to make a good setup, and that therefore can be used to analyze the likely ranks of unknown pieces.

I have already collected setup statistics for the setup creation module (section 4.3.2), and these same statistics are used to guess the ranks of the opponent's pieces.

The statitics show clearly that pieces on different positions in the setups have very different rank probabilities. Most of the figures aren't very surprising for an experienced player, these are numbers he knows from experience. "Invincible" can now easily use these numbers as initial probabilities. Of course these values are very unreliable since they are averages over thousands of games. The most likely setup using these values may never have been played at all. They are only initial guesses that must be confirmed by later events.

The second statistic I collected from the setup data was the probability of ranks being put next to each other. I distinguished between a rank standing in front of another, behind another or next to it. Right or left seemed to be symmetrical, but behind or in front of certainly not: there often is a bomb in front of the flag, but rarely behind it, the spy often stands behind the general, but almost never in front of it.

Since the initial ranks are unknown this information can't be used at the start, but Invincible keeps track of which pieces were standing next to each other in the initial setup. Every time an enemy piece is revealed this also tells something about its neighbors. Once the general is discovered, the pieces that started next to or behind it become automatically suspected of being a spy. When a bomb is discovered somewhere the piece behind it gets a higher probability of being the flag.

This is done by multiplying the probabilities for each rank for its initial neighbors with the relative likelihood of the pieces being neighbors: number of times they were neighbors divided by the expected number of times.

6.3.2 Missed opportunities

An obvious give-away of a piece's rank is if he had a certain chance to capture a known piece but didn't do it. If a known general is trapped by 2 unknown pieces, one of them probably being the marshal and the other a bluff he will be forced to choose. If "Invincible" guesses right and move next to the bluff and he doesn't capture him but runs away, the conclusion should be made that that piece is lower than a general and remember that for possible future situations in which the same piece is involved.

It is important to add here that conclusions can only be drawn from unforced moves. If the opponent has to bring his own general in safety he wouldn't capture a major in that move even though he could. It would then be a mistake to think that he couldn't.

6.3.3 Behavior towards known pieces

As said before, the initial probabilities are necessarily unreliable since they do not use any information about this particular game. To make a better prediction, the moves of the opponent have to be analyzed. One way to do this is to look at how they behave towards the pieces that are known to him. If a piece moves towards a known general, the probability of it being the marshal becomes larger. He may be bluffing, or just accidentally moving there so the probability should never become 1, but the action does justify increasing the probability.

Generally speaking, a piece moving away from a known piece is expected to fear it and a piece moving towards it is expected to want a battle with that piece.

To calculate this I calculate for every field on the board for every rank of the opponent how "happy" this rank would feel on that field. This is based on the profit (or loss) of that rank against any known piece and the distance to it. The further away, the less important its effect, but there always is some effect. In addition, the type of approach is also taken into account. An attacking piece likes to be in one line with his target (because of the two-square rule) whereas a evasive piece would prefer to approach over adjacent lines, the first having a much higher chance to lead to an encounter.

This may seem like a rather bold assumption, since the opponent may be bluffing and he certainly doesn't take all these relations into account especially on longer distances. However it's very hard for a player to consistently bluff. Pieces that move without system will alternate between having certain probabilities increased and decreased and thus not be affected as much as pieces that consistently move further away or closer to pieces of certain ranks. If the general is on the same side as the opponent’s revealed marshal it can’t be useful on that side and if he wants to use it for attack he will necessarily have to move away from the marshal. The result of this is that all pieces fleeing from the side where a marshal is known become possible suspects for the opponent general.

7. Implementation

This chapter will discuss technical details of the implementation.

7.1 The user interface

This section focuses on the design and implementation of the user interface that allows human players to play Stratego against "Invincible". This is a technical description of the system and not a user manual. The manual can be found in appendix B.

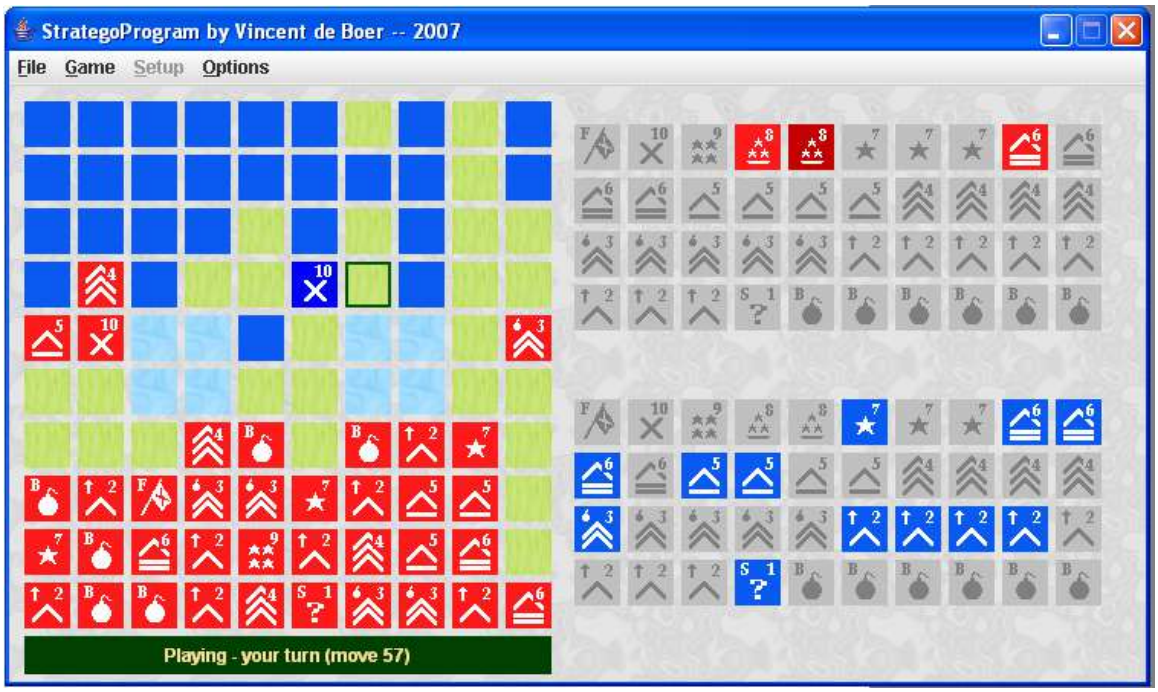


Figure 7.1 – Screenshot of the game client (manual in appendix B).

7.1.1 Requirements

After looking at other Stratego programs, and combining my experience with them with that of other Stratego players, I came to the following list of requirements for the user interface:

- It should be possible to see the board and execute moves against "Invincible"
- The last move of the player and the opponent should be clearly visible
- The captured pieces should be clearly visible at any point in the game
- The rank of a piece that won a battle should remain visible during the move of the player, but not longer than one move.
- The player should be able to save and load setups

For development and testing, the following extra requirements were made:

- It should be possible to save finished games and replay them later
- It should be possible to play with all opponent pieces visible
- It should be possible to play with a custom starting position

All these requirements have been met. Figure 7.1 gives a screenshot of the user interface. On the left there is the board and on the right the captured pieces. The last move is indicated by giving the piece that was moved a darker color and drawing a border around the starting field of the move. If a piece was captured, it shows up in dark color in the box on the right, pieces captured in earlier moves are shown in light color. Setups can be saved and loaded through the setup menu.

The options menu allows the tester to reveal all Invincible's pieces, or during setup phase select that he wants to make a custom starting position, which allows him to put both his and "Invincible"'s pieces on the board and not be limited only to the normal setup area.

Games can be saved in the file menu after they are completed. When a saved game is loaded, a forward and a back button allow the evaluator to do and undo moves of the completed game.

7.1.2 Class diagram

The class diagram of the game client has been displayed in figure 7.2.

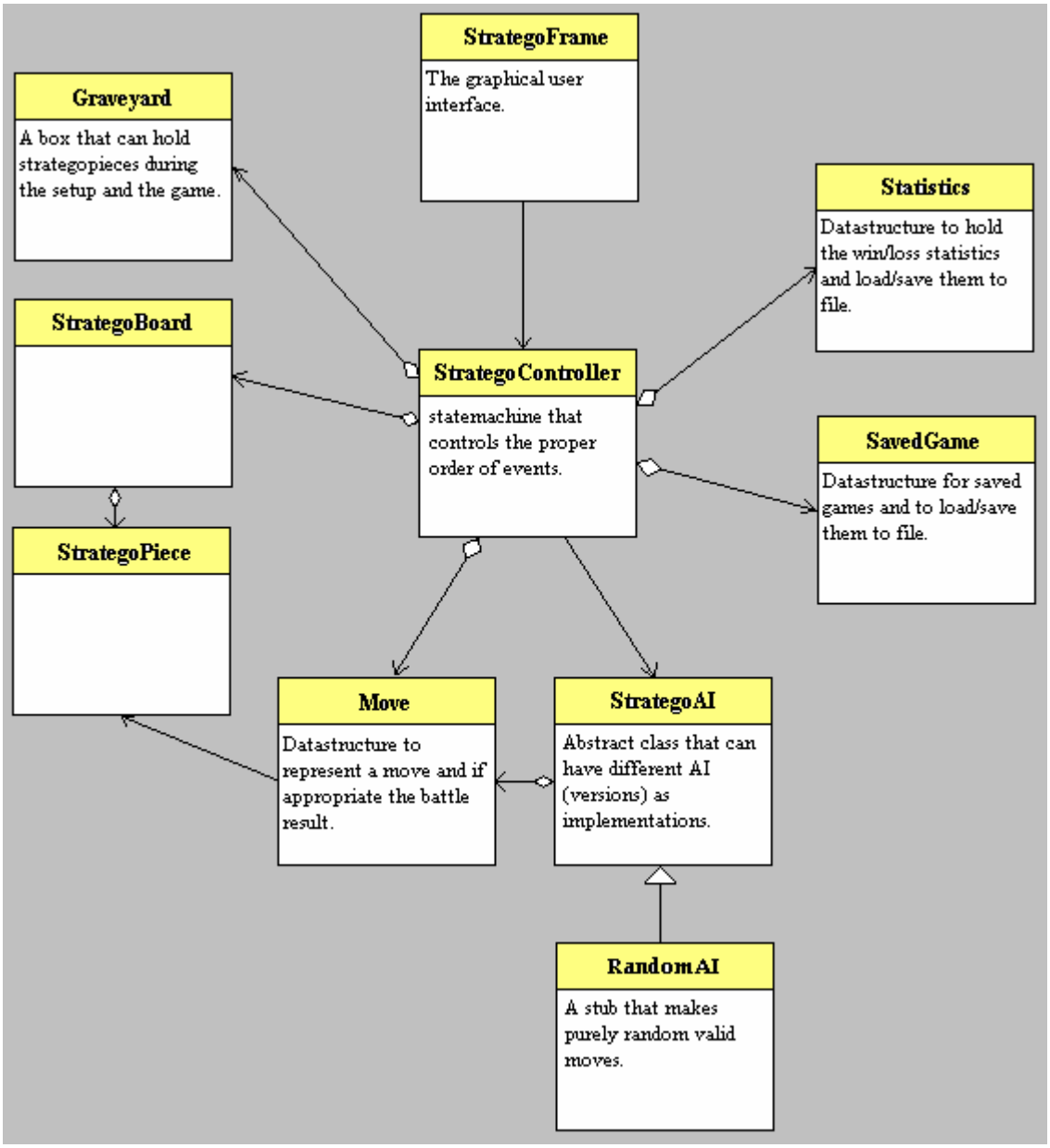


Figure 7.2 – Class diagram of the game client.

The StrategoFrame class deals with everything that is shown on the screen and handles user events. The StrategoController contains the functional code. All events from the user are passed to the controller, which executes them. The state of the controller also defines the behavior of the user interface. The abstract StrategoAI class contains the code to determine the legal moves, which is also used by the StrategoController class.

7.1.3 Class descriptions

This is an overview of the classes. Only the important features are described. Get and set methods, methods to save and load data from files or give string representations of items are not mentioned.

Class:	**StrategoController**
Description:	This class controls the behaviour of the program. It contains a state machine with 4 states: SETUP, PLAYING, GAMEOVER and REPLAY. In the setup state the player can make his setup. This is the only state in which the setup menu is enabled. In the playing state, the players alternately make moves. After the player makes a move this move is given to "Invincible" and he is asked for his move. The result of the move is then returned to "Invincible" so that he knows the result of a possible battle. This goes on until either side wins the game or a technical or declared draw occurs. When the game is decided the controller goes to the gameover state. No more moves can be done, but the game can be saved now so that it can be reviewed later. From this state the program can be closed without resulting in a loss, a new game can be started, or a game can be loaded for review. When a game is loaded, the controller goes to the review state. In the review state no moves can be made, but the loaded game can be played move by move forward or backward. From this state a new game can be started or another review game can be loaded.

Class:	**StrategoFrame**
Description:	This class takes care of the interaction with the user. It creates the graphical user interface and handles user actions. All requests from the user and then sent to the StrategoController to be processed.

Class:	**StrategoAI**
Description:	This is an abstract class that contains some standard methods for the AI-bot. The StrategoAI has no information about the enemy pieces. The only information it gets from the controller are the enemy moves and the results of battles.
Abstract methods	**public abstract Move getMove()** *This method should return the next move of the AI-bot, and gives the opponents previous move. If the move was a battle, the opponents rank*

	is also included in the move **public abstract void opponentMove(Move oppMove);** *This method is called to submit the opponents move to the AI-bot.* **public abstract void myMoveResult(Move myMove);** *This is the callback method that is used to return the result of the AI-bot's move back to the bot. If the move was a battle, the opponent rank is included in the move.* **public abstract String getSetup();** *Gives a setup for the AI-bot. Setups are represented as 40-character Strings.* **public abstract boolean acceptDraw();** *This method is called when the user offers a draw. Iff true is returned, a draw is agreed on and the game ends.* **public abstract void setBoard(String b);** *This method is used to submit a user defined board position that the AI-bot should start with.*

Class:	**Move**
Description:	Internal representation of a move. Used to pass to the AI-bot and store in history lists.
Fields	fromX: *x-coordinate of origin* fromY: y-coordinate of origin toX: *x-coordinate of destination* toY: *y-coordinate of destination* isBattle: *true if a battle took place in this move* result: *WIN,LOSS or TIE ->the result of the attack* defender: *StrategoPiece on the destination field before the move* attacker: *StrategoPiece on the origin field before the move*

Class:	**StrategoBoard**
Description:	Internal representation of the stratego board.
Methods	isSetupArea(color) isOccupied(y,x) isAccessibleField(y,x): *returns true if x,y is a moveable field on the board*
Fields	lakes: *10x10 array locating the lakes on the board* board: *10x10 array holding the pieces on the board*

Class:	**StrategoPiece**
Description:	Internal representation of a stratego piece.
Methods	getSpeed(): *returns the number of squares this piece can move*

Fields	rank: *intvalue of the rank of the piece. The higher the value, the stronger the piece.* color: *RED or BLUE* hasMoved: *not currently used* isKnown: *not currently used*

Class:	**Graveyard**
Description:	This is the Stratego box, used to store pieces that are not on the board. During setup, all pieces start here and can be put on the board, and during the game, the captured pieces go to back here.
Fields	dead: *a 2x40 array to hold both sets of pieces* rank_positions: *a 40 element array with information about which piece belongs on each position of the dead array.*

Class:	**SavedGame**
Description:	A datastructure for saved games. Makes it easier to load and save games.
Fields	initBoard: *the initial board position* history: the list of moves result: the result of the game

Class:	**Statistics**
Description:	A class to load and store the win/loss statistics. They are stored in a file to keep the statistics every time the program is started.
Fields	wins ties losses

7.1.4 Sequence diagrams

In figure 7.3, a sequence diagram is given for the usecase where the user makes a move. The move is passed from the GUI to the StrategoController, which executes the move on its own board and passes it to the AI-bot, which also executes the move. The AI-bot then returns its own move, which is passed then shown in the GUI.

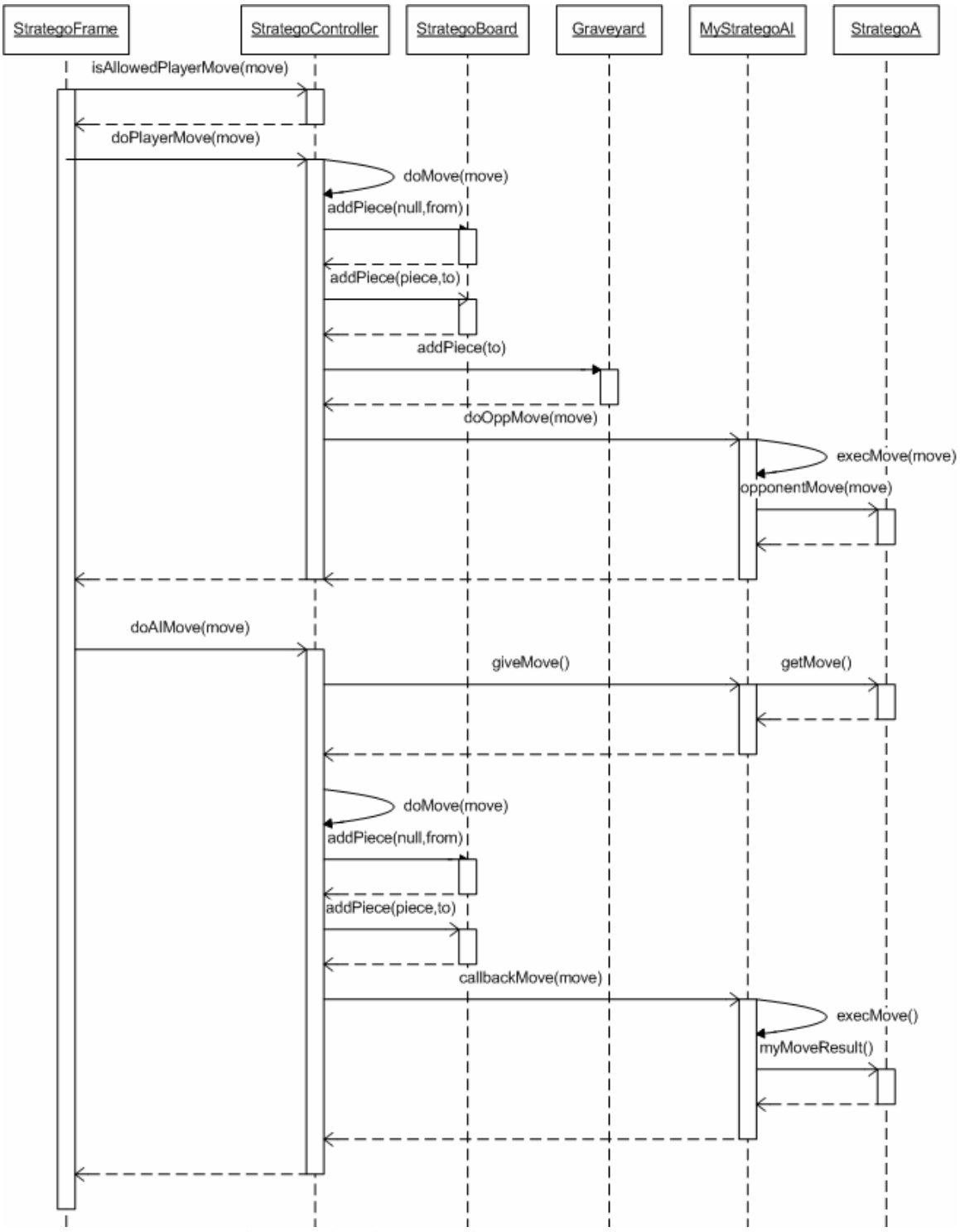


Figure 7.3 – Diagram for the case when the user makes a move.

7.2 The artificial intelligence

This section focuses on the design and implementation of the Artificial Intelligence bot "Invincible".

7.2.1 Class diagram

In figure 7.4, the class diagram of Invincible is given.

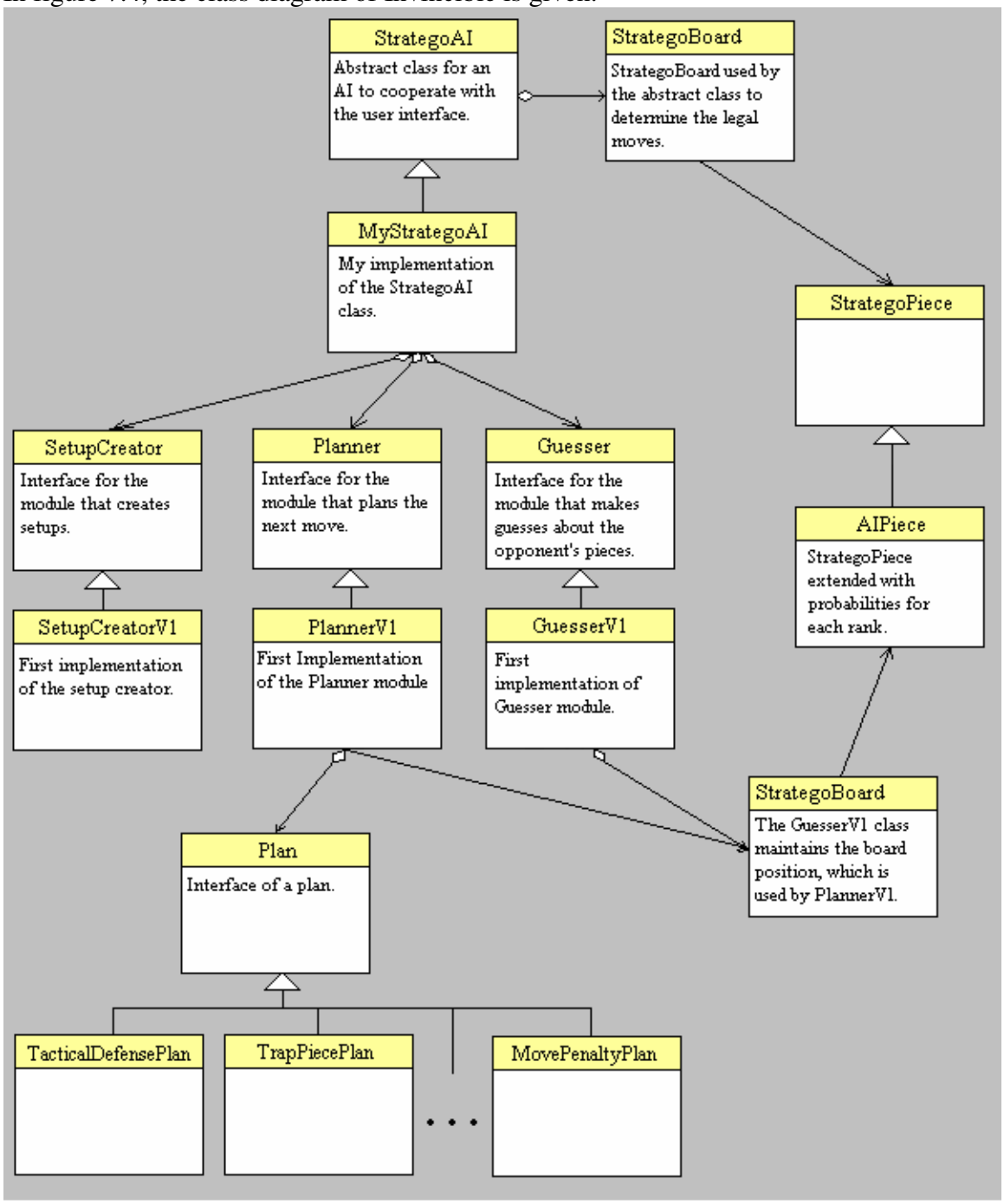


Figure 7.4 – Class diagram of the Stratego Bot.

The main class, MyStrategoAI is an implementation of the abstract StrategoAI class, supplied by the user interface. It consists further of the three sub-parts for making setups, guessing opponent pieces and reasoning about moves.

7.2.2 Class description

This section gives class descriptions, with important methods and fields. Get and set methods are omitted and so are small helper methods.

Class:	**MyStrategoAI**
Description:	The main class of my AI bot. Implementation of the abstract class StrategoAI from the user interface.
Methods:	**public Move getMove()** *//abstract method in StrategoAI* **public void opponentMove(Move oppMove)** *//abstract method in StrategoAI* **publicvoid myMoveResult(Move myMove)** *//abstract method in StrategoAI* **public String getSetup()** *//abstract method in StrategoAI* **public boolean acceptDraw()** *//abstract method in StrategoAI*
Fields:	**private StrategoBoard board;** *//own representation of the board, shared by all modules* **private Guesser guesser;** *//module that guesses pieces* **private SetupCreator setupcreator;** *//module to create setups* **private Planner planner;** *//module that decides the next move*

Interface:	**SetupCreator**
Description:	Interface for classes that can create Stratego setups. This makes it easy to make different implementations and plug them in to the bot without changing a lot of code.
Methods:	**public String createSetup(int color);** *//returns a setup in String format*

Interface:	**Guesser**
Description:	Interface for the Guesser module. This makes it easy to make different implementations and plug them in to the bot without changing a lot of code.
Methods:	**public void doMove(Move m);** *//updates the probabilities after this move* **public double[] getProbabilities(int x, int y);** *//returns the probabilities for the piece on position (x,y)*

Interface:	**Planner**
Description:	Interface for the Planner module. This makes it easy to make different implementations and plug them in to the bot without changing a lot of code.
Methods:	**public void checkPlans();** *//recalculates all plans. Is called once at the beginning of every move* **public Move getBestMove(Vector possibleMoves);** *//returns the move this class considers best*

Class:	**SetupCreatorV1**
Description:	Implemenation of the SetupCreator interface
Methods:	**public String createSetup(int color);**
Fields:	**private double[][] setup_prob;** *//setup_prob[field][rank] is the probability that field field contains a piece of rank rank. Field is a number between 0 and 39 that represents one of the 40 setups fields* **private double[][][] neighbour_prob;** *//neighbour_prob[rank1][NEXTTO][rank2] is the probability that rank1 appears next to rank2* **public static final int BEFORE = 0;** *//used as index for neighbour_prob* **public static final int NEXTTO = 1;** *//used as index for neighbour_prob* **public static final int BEHIND = 2;** *//used as index for neighbour_prob*

Class:	**GuesserV1**
Description:	Implemenation of the Guesser interface. This class updates the expected probabilities of each piece for each rank. It also updates the shared instance of StrategoBoard that is used by the Planner implementation. The initial probabilities are the setup-probabilities as used by the setup creator class.
Methods:	**public void doMove(Move m);** *//updates the probabilities after this move* **public double[] getProbabilities(int x, int y);** *//returns the probabilities for the piece on position (x,y)*
Fields:	**private StrategoBoard board;** *//shared board* **private Graveyard graveyard;** *//shared graveyard, containing the pieces that are not on the board* **private Move lastmove=null;** *//stores the previous move* **private double[][][] influencematrix;** *//used to calculate the influence of known enemy pieces on all fields to see how the moving piece behaves towards them* **private double[][] piecevalues;** *//contains the values of the different ranks for both players* **private double[][] setup_prob;** *//setup_prob[field][rank] is the*

	probability that field field contains a piece of rank rank. Field is a number between 0 and 39 that represents one of the 40 setups fields **private double[][][] neighbour_prob;** *//neighbour_prob[rank1][NEXTTO][rank2] is the probability that rank1 appears next to rank2* **public static final int BEFORE = 0;** *//used as index for neighbour_prob* **public static final int NEXTTO = 1;** *//used as index for neighbour_prob* **public static final int BEHIND = 2;** *//used as index for neighbour_prob*

Class:	**PlannerV1**
Description:	Implemenation of the Planner interface. It keeps a collection of "Plans" that give a value to each of the legal moves in this position. The move with the highest total value is selected.
Methods:	**public void checkPlans();** *//recalculates all plans. Is called once at the beginning of every move* **public Move getBestMove(Vector possibleMoves);** *//returns the move this class considers best*
Fields:	**private int myColor;** *//the color of the player for which this planner needs to select a move.* **private StrategoBoard board;** *//the shared board* **private Guesser guesser;** *//the guesser. This is not really because the planner can access the information supplied by the guesser through the board* **private Graveyard graveyard;** *//the graveyard* **private Vector history;** *//the previous moves made in this game* **private Vector<Plan> plans = new Vector<Plan>();** *//the collection of plans*

Interface:	**Plan**
Description:	Interface of a plan. The planner keeps a collection of classes implementing this interface and uses them to determine the next move.
Methods:	**public double moveValue(Move m);** *//awards a value to Move m* **public void recalc();** *//recalculates the plan based on the current board position. This method is called once for each board position, rather than once for each position for each possible move like moveValue* **public boolean isActive();** *//returns true iff the plan is active. If it is not active, moveValue is still called for every possible move because the move may activate the plan. This method is mainly to filter interesting plans when showing a (debug) overview of the reasoning of the planne.* **public String name();** *//returns a name for this plan. Also used only for giving overviews*

7.2.3 Sequence diagrams

Figure 7.5 gives the sequence diagram for the case when Invincible is given his opponent's move and asked for his own move.

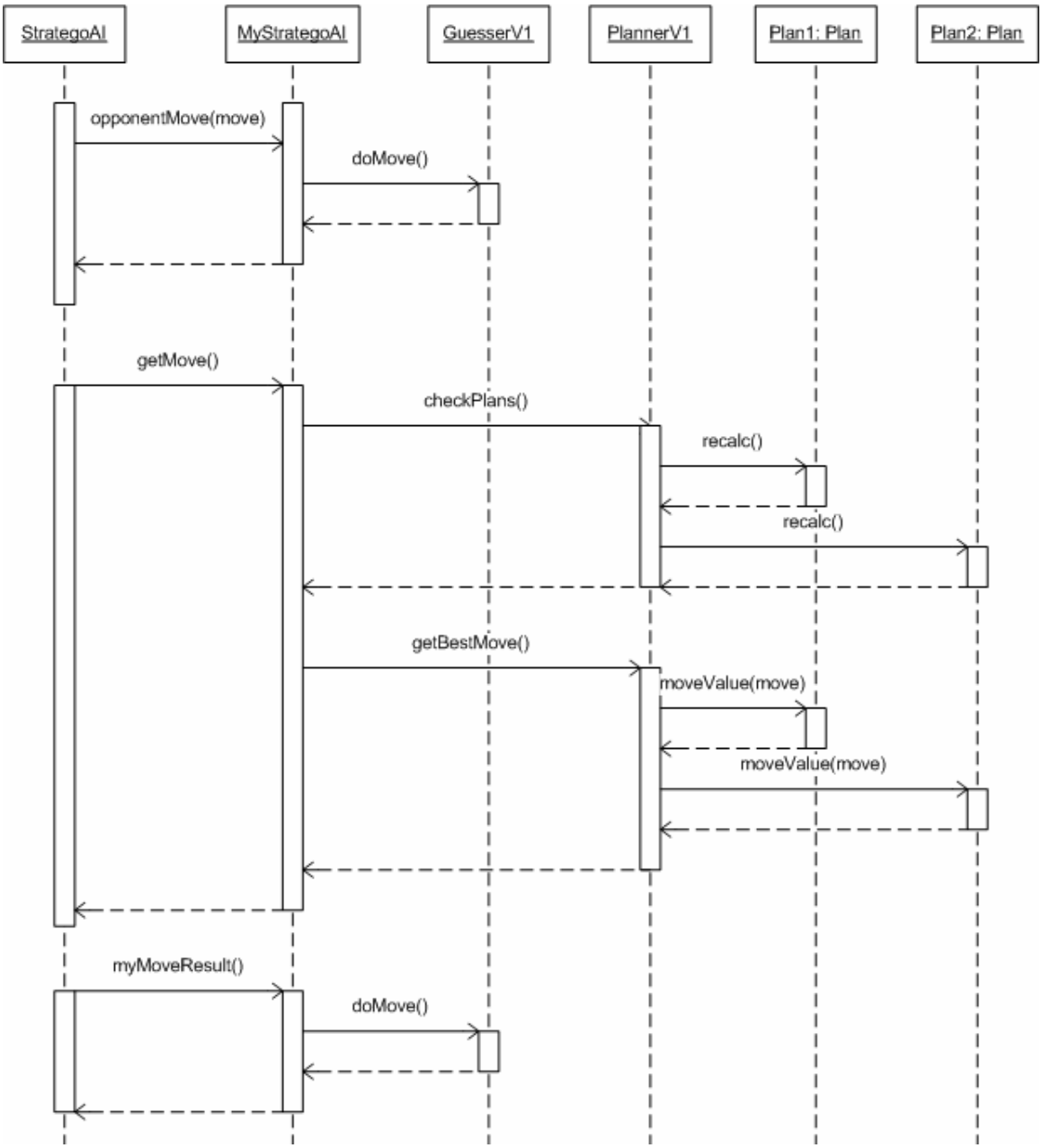


Figure 7.5 - The sequence diagram for determining and processing a move.

Figure 7.6 gives the usecase for starting a game. Invincible is asked for a setup and then given his opponent's setup. There is no explicit start of the game, Invincible assumes the game has started when he is asked for a move.

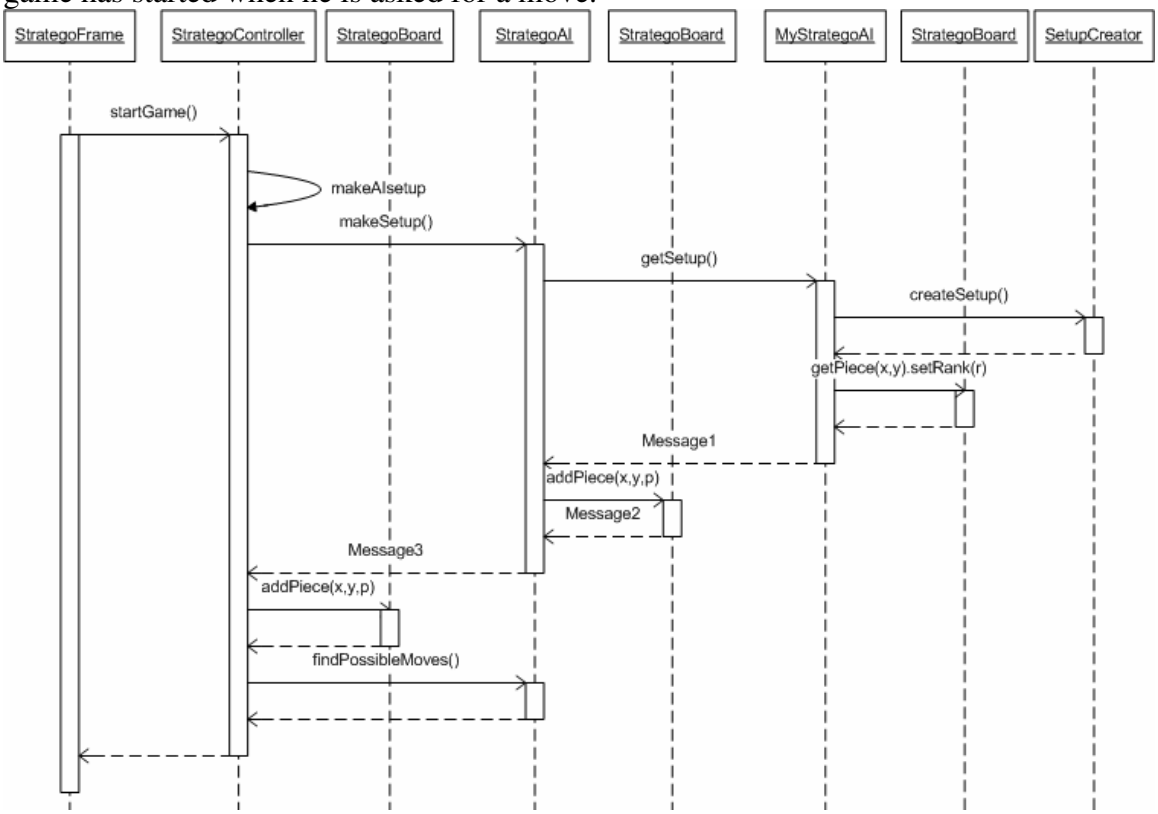


Figure 7.6 - Sequence diagram for starting a game.

7.2.4 Setup representation

For passing setups to the game controller and saving them for re-use, setups are represented as 40-character Strings. Each character represents a single piece. Depending on the side, pieces are put on the board in the following order:

Table 7.1 – String indexes of setup positions.

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
39	38	37	36	35	34	33	32	31	30
29	28	27	26	25	24	23	22	21	20
19	18	17	16	15	14	13	12	11	10
9	8	7	6	5	4	3	2	1	0

The piece characters are taken from the gravon[2] database. They use the StraDoS[3] notation, which is the only standard for annotating Stratego positions.

A: empty field	
B: red bomb	N: blue bomb
C: red spy	O: blue spy
D: red scout	P: blue scout
E: red miner	Q: blue miner
F: red sergeant	R: blue sergeant
G: red lieutenant	S: blue lieutenant
H: red captain	T: blue captain
I: red major	U: blue major
J: red colonel	V: blue colonel
K: red general	W: blue general
L: red marshal	X: blue marshal
M: red flag	Y: blue flag

The distinction between red and blue pieces isn't really necessary for setups when the color of the player who supplies the setup is known, nor is the empty field needed when 40 pieces have to be distributed over 40 squares, but I still chose to use the standard rather than inventing something new. The color distinction and the empty field can become useful for defining partial setups as test cases for "Invincible".

7.2.5 SetupCreatorV1

The setup creator uses statistical data about human-made setups to generate semi-random setups. First of all it uses the probability that a certain rank appears on a certain position and secondly it uses the probability that a certain rank appears as neighbor of another rank. It makes 3 distinctions in the neighbor relation namely: "next to", "in front of" and "behind". The pseudo code for generating pseudo-random setups from these data is as follows:

```
for each piece p with rank r
        rand := random double between 0 and 1
        for each empty square s
                if(prob[s][r]>rand)
                        put piece p on square s
                        for each neighbour n next to s
                                for each rank r2
                                        prob[n][r2] *= probability that r2 is next to r
                        n := square behind s
                        for each rank r2
                                prob[n][r2] *= probability that r2 is behind r
                        n := square in front of s
                        for each rank r2
                                prob[n][r2] *= probability that r2 is in front of r
                else
                        rand -= prob[s][r]
        normalize probs
```

Figure 7.7 – Pseudo code for generating pseudo-random setups.

Because the probability matrix is updated after placing a piece it makes a difference in which order the pieces are put on the board. To have the biggest effect of the neighbor relation statistics the ranks that have the biggest impact on their neighbors should be put on the board first. If they are placed later, a lot of their neighboring fields may already contain a piece so the update of the probabilities of that field has no effect. If the ranks whose relations to neighboring fields deviate the most from 1.0 are placed first and those whose relations are closest to 1.0 are placed last then the expectation of updates to fields which already contain a piece will be minimal.

8. Testing

8.1 Introduction

Testing a knowledge-based system like Invincible is a complicated thing to do. Even though the world in which it has to take decisions is greatly simplified because it is a game with relatively simple rules, the number of different situations is still vast. The best move in a situation depends not only on the positions of the pieces and the information available to the program, but also on what Invincible assumes about the information he doesn't yet have. A move that seems strange may be caused by wrong reasoning or by wrong assumptions about the hidden information. The latter may again occur because of wrong reasoning or because of the skill of the opponent. Human players also frequently make mistakes against bluffing opponents. Furthermore, whether a specific move is good or not can only be judged by a human expert. It is possible to measure performance by playing games, but unless the opponent is of a known, constant strength which is also comparable to the level of the program, even this measure is not very reliable. And if such opponent is found, the win/loss record will not give hints about how to improve the program.

8.2 Test methods

The bulk of the testing is done by playing games against the program or by watching other people play games. When the program makes a weak move I try to establish the reason and find a pattern in the most common mistakes so that this can be improved by adding a new plan to solve the situations the program can't yet handle, or by fixing a bug in one of the existing plans.

To find the cause of mistakes it is necessary to know what the program knows or guesses about the enemy pieces at that moment, and to know which plans caused to erroneous move to be selected.

The expectations about all pieces can be checked during the game in the AI debug frame shown in figure 8.1. It is possible to select a rank and color of a piece and it will show with colors the expectation that each piece of that color has this rank. The pieces that are known to be that rank have a probability of 100% and black color. The pieces that are known to be a different rank have a probability of 0% and are thus colored white.

On the right it shows the values of all ranks at that point in the game. Since this is determined dynamically depending on the pieces left, and the values of pieces for a large part decide which actions are worth making it is useful to be able to see these.

The number below the ranks is the game state. If this is larger than 1 then the program considers himself to be ahead in pieces. This is another value that determines the behavior of the program.

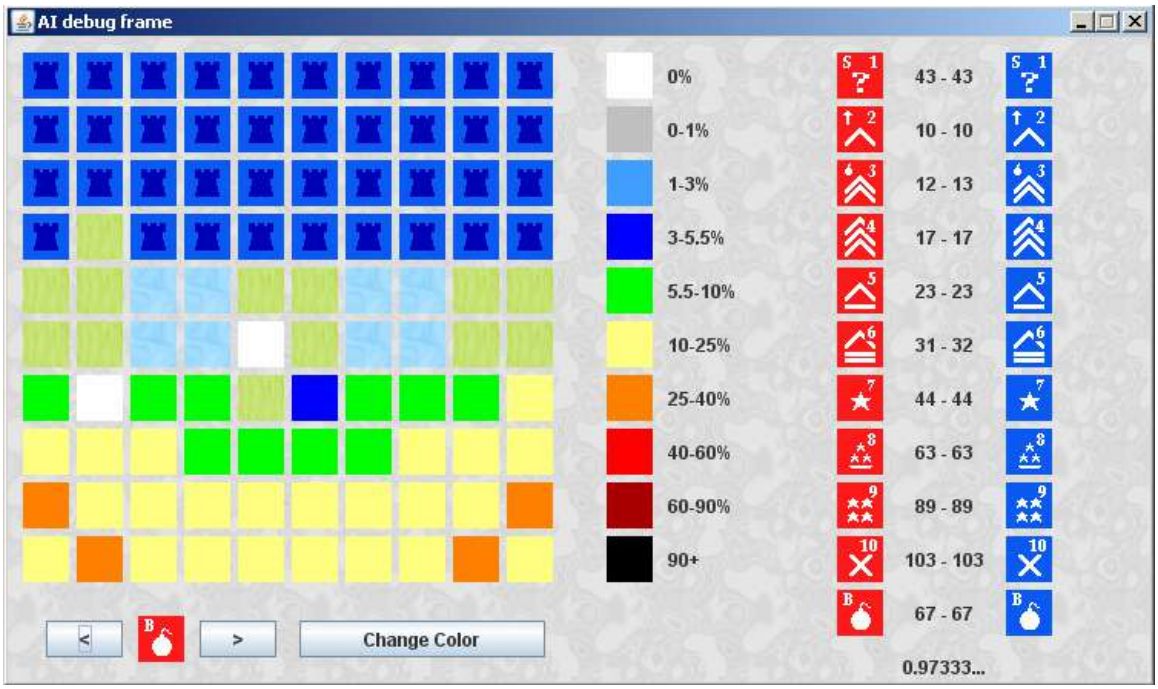


Figure 8.1 – a window into the programs "mind".

To find the values of each of the separate plans for each of the possible moves, a table is printed, where each row is a move and each column represents a plan. To make the table easier to read, only the active plans are showed. In table 8.1 a selection from such table at the start of the game is given. There are more plans but for readability they are removed in this table.

Table 8.1 – The values of the active plans contributing to the selected move.

Move	Rand	MPen	dMfS	ICap	SDef	Intel	SAtt	Se.bS	Se.bX
Move from A4 to A5	0.291	-11	0	0	0	0	0.562	0	0
Move from B4 to B5	0.299	-1	0	0	0	0	0.547	0	0
Move from B4 to B6	0.23	-4.14	0	0	0	0	0.547	0	0
Move from B4 to B7	0.893	0	0	0.667	0	0	0	0	0
Move from E4 to E5	0.586	-1	0	0	0	0	0.657	0	0
Move from F4 to F5	0.135	-1	0	0	0	0	0.613	0	0
Move from I4 to I5	0.544	-1	0	0	0	0	0.547	0	0
Move from I4 to I6	0.577	-4.14	0	0	0	0	0.547	0	0
Move from I4 to I7	0.801	0	0	0.612	0	0	0	0	0
Move from J4 to J5	0.714	-1	0	0	0	0	0.547	0	0
Move from J4 to J6	0.113	-4.14	0	0	0	0	0.547	0	0
Move from J4 to J7	0.624	0	0	0.902	0	0	0	0	0
Selected Move from B4 to B7 (1.5615150943916754)									

When the program makes a weak move it is easy to see which plan(s) contributed most to giving this move a high value and they can be adapted to give a more appropriate value in a situation like the one where it went wrong.

When a new plan is added that should find the solution in specific situations it is possible to make a custom starting position where the user can select the positions of all the pieces on the board and is not restricted to putting them in the normal setup areas. This makes it possible to quickly create an interesting situation to see how the program responds in this situation.

It is also possible to replay saved games and scroll back and forth through the moves to analyze games that have already been played.

8.3 Playing human opponents

Eventually, the strength of the program can only be tested by seeing how it performs in real games. Invincible played a number of fellow master students. There is a move-by-move analysis of these games in appendix C. These are players with varying experience with Stratego.

8.3.1 Game 1: Mohannad defeats Invincible

In this game, Invincible still has no idea how to find high ranks if the opponent doesn't attack with them and this costs him the game. Mohannad keeps his marshal and spy behind and Invincible has no idea how to attack. He therefore gives Mohannad all the time to roll up his defenses and get the flag without ever posing a serious threat.

8.3.2 Game 2: Invincible defeats Raymond

Invincible plays much more aggressively now. He knows how to collect information and is therefore able to launch some attacks. He still misses a few positions where he could have gotten an easy advantage, but is leading throughout the game. He does make it unnecessary exciting towards the end because he fails to withdraw his highest pieces for defense when there is nothing left to win in the attack. He lets Raymond get dangerously close to getting the flag, but eventually manages to get himself out of danger.

8.3.3 Game 3: Invincible defeats Mehdi

This game started very even, but halfway through the game Mehdi started to take risks. They eventually gave him a lead, but it cost him a few high pieces. Invincible used his lead well and won the game without making the defensive mistakes which he made in game 2.

8.3.4 Game 4: Invincible defeats Lorena

Lorena was another aggressive opponent. Invincible was lucky to have his pieces at the right places and got an early lead. He never gave away this lead, and was strong both in defending against Lorena's attacks and attacking her in return.

8.3.5 Game 5: Sjoerd defeats Invincible

Sjoerd is the most experienced Stratego player from all the opponents; He used an unexpected setup with bombs blocking two of the three lanes between the lakes and all his high pieces on the last side. This is something Invincible isn't taught to recognize and which he handled wrong. Invincible immediately got a lot of pieces behind, but in return got Sjoerd's marshal. He later also got a colonel so he had the highest two pieces. The marshal was blocking Sjoerd's only exit so it looked very good for Invincible. He didn't know how to finish this however as Sjoerd still had a spy and he kept his colonel on the other side to defend against an attack which couldn't happen because Sjoerd had blocked these paths with bombs. This stayed so for a long time with both players moving back and forth, but eventually Sjoerd managed to get a miner past the marshal, with the help of an unknown spy and get to the flag. Especially this last feat was something Invincible didn't have to allow but he considered himself to be down in pieces because Sjoerd had about 10 low pieces in return for only the marshal and the colonel. Since Sjoerd still had a spy this was a game he could win, if not for his strange setup. Invincible might have considered himself to be on the losing side so he didn't bother defending the flag too hard.

8.3.6 Summary

Invincible held his own against a varied set of opponents, winning 3 of 5 games. He lacks the ability to detect the weakness of unusual approaches like the one Sjoerd used in the fifth game. This is a weakness of non-learning expert based systems that can't be overcome. It can never be told everything so there will always be situations that it can't handle correctly. On a positive note, he did have winning chances in all but the first game and converted these chances to wins in three games, which is not a bad score for a computer in such a human game as Stratego.

9. Conclusions & Recommendations

9.1 Extended summary

This chapter starts with a summary about the results of the different sub-parts discussed in this thesis.

9.1.1 User interface

To test my program and let human players play against Invincible, I created a simple user interface that allows the user to play a game against the computer. There are options to create a manual setup and possibly store it or let the computer generate one. The player can also use the generated setup as basis and make some changes before confirming.

Finished games can be saved and replayed later for analysis and it's possible to make a custom position and play starting from that position. This was particularly useful for testing specific situations.

9.1.2 Setup creation

Unlike any other Stratego program, Invincible creates his own setups piece by piece without using any pre-defined structures. He can do this by using statistical data collected from human-made setups and using this to build his own semi-random setups. To improve the quality of these setups he is given a number of heuristic rules to give values to such semi-random setup. He then makes several setups and picks the one with the highest heuristic value.

When creating a setup the trade-off between putting everything in the right places and being unpredictable has to be made. It often depends on the opponent whether a setup should be more secure or more unpredictable. Because Invincible doesn't use opponent modeling it can not take this into account, but otherwise the setups he creates are pretty good. They are not much different from human-created setups.

Human players are also given the option to play with an automatically generated setup and usually agree to the first semi-random setup they get. There are some strange exceptions. They could be removed by increasing the number of setups created from which the best is selected but that would make them more predictable. Having an occasional strange element in setups is therefore not necessarily a bad thing.

9.1.3 Guessing ranks

The program uses probabilities per piece per rank. For each piece it keeps a list of probabilities that the piece has that rank. The probabilities for each piece always add up to one and the probabilities that pieces have rank "a" always adds up to the number of

pieces of this rank that the opponent still has (the number of each rank the opponent still has is always known).

Because of the normalization, this architecture automatically distributes changes to one piece over all the other pieces. This takes care of inference like that when a piece is revealed to be a marshal, all other pieces have a 0% probability to be the marshal (since there is only one marshal), but also that this makes it more likely that this piece has any of the other ranks. If all but one rank is found, then the remaining unknown pieces automatically get a 100% probability to have this rank.

This means that heuristics only have to focus about changes to one piece, which greatly reduces the number of rules needed to give a reasonable estimation of the enemy pieces. There are only several general rules that update the pieces after each move, but the conclusions made this way are often remarkably accurate. If a player doesn't bluff then the program usually knows which pieces the player is moving. If he does bluff then he makes mistakes, but so would human players. He frequently does guess right, hitting some unknown pieces and then retreating from the real high rank.

9.1.4 Move selection

The move selection is done using a new approach, where a set of plans each give a value to all the available moves which are then added to give a total value for each move. The move with the highest total value is selected. The plans are simple sub-problems of the relatively complex game, like capturing a known piece or preventing pieces from getting captured. These plans can be solved by path finding and profit formulas rather than by making a tree of all the possible moves until a certain search depth. This greatly simplifies the computational complexity of the game while it still allows directional searches for with clearly defined aim. The drawback is that something which is not specifically searched for will not be detected.

In spite of it often being hard to make general plans out of knowledge that are specific enough not to overlap yet general enough to be useful, I did not encounter a problem that could not be solved by this architecture. Even tactical close-distance encounters of one or two known pieces, the one thing that brute-force search could possible solve perfectly, could be solved by looking at relative positions of pieces rather than by looking at all the possible moves they could make. More complex things like bluffing or trapping a distant piece with two of own pieces could also easily be solved by defining them as plans telling whether a move contributes to this plan or not. More vague strategic notions like retreating high pieces when the program is ahead and should defend from desperate attacks can be easily formulated as a plan while it's something that can't realistically be seen by brute-force search.

Some plans turned out to be cancelling each other out, for example when one plan encouraged "Invincible" to bluff by moving a low unknown piece next to a higher known enemy but another plan discouraged him to move a low piece next to a higher one. This was usually caused by plans being applied to more situations than they should. Making

the plans more specific always solved these issues. In the above example, there should be no penalty for moving next to a higher piece when there is a reasonable chance that the opponent will believe the low piece is even higher than the known piece it moves next to. This was often pretty easy to see when a new plan was cancelled out by an old plan because it is easy to make a situation in which the new plan should find the right move, but if a new plan cancels out one of the existing plans this is more easily overlooked.

Another problem was to find appropriate relative values for all the plans. If a value is too high then the program might pursue this plan while ignoring more urgent things causing him to make stupid mistakes. If the value is too low it might not act on this plan when it should but rather follow an inferior plan. Some of these things can be captured in formulas, like it being better to move away a colonel from danger than to capture a captain. It is not so easy to calculate how this relates to the penalty of moving a piece for the first time and thus revealing it is not a bomb or the value of trying to capture a piece when the attack is not guaranteed to succeed (as is often the case because of incomplete information). Most of these values are heuristic and tweaked by looking how they relate in different testing situations.

9.1.5 Testing

Testing the system turned out to be difficult. The more plans were added, the less clear it was why certain moves were made in certain positions. Also each new plan could disrupt the working of any of the existing plans.

Because of the incomplete information, a realistic test situation is hard to create. A situation is defined not only by the position of the pieces, but also by knowledge each player has about the opponents pieces and by what they can suspect about the pieces based on the previous moves they made. It is further complicated because of the dynamic values of pieces, so setting up just a part of the board and leaving out the bulk of the pieces affects the values of the remaining pieces and thus altering the situation.

It is possible to quickly create a simplified test situation to see whether the plan is in principle working as it should, but this gives no guarantees for how the plan works in a real game. Most mistakes found in manually setup positions were caused by the guesses of piece ranks or the values of the remaining pieces not being realistic. Fortunately, if there was a clear bug then it usually would be found this way, so this way of testing was still useful.

The final effect of a new plan can only be found by playing a real game, which made testing rather time-consuming.

9.1.6 Skill level

The level of the program seems promising. It can beat average human players without taking risks, and though it still falls short to experienced human players, there is still much room for improvement by adding more knowledge to the system. The current

implementation is by far not the limit of what can be done with the chosen architecture, but making it significantly stronger would require a lot more time.

9.2 Conclusions

A working prototype of Invincible has been created and tested that plays at a level that is well above that of a beginning human player. The new algorithms used are a big reason for this success.

Invincible makes his own setups, which are good enough to be used by human players. It is the first program ever that can generate decent Stratego setups. This makes the player more unpredictable than its rivals which use hand-made setups.

The probabilistic representation of piece ranks makes the process of guessing opponents ranks and managing and using this information very easy. Only three general heuristics are used to evaluate the moves of the opponent, which gives surprisingly accurate guesses of the opponent ranks. This could be improved by adding more heuristics, but in none of the tests Invincible made unreasonable guesses about piece ranks. He is often wrong or indecisive about pieces, but so are human players. He is also often sure of pieces that I personally would not have suspected, which shows that the perfect memory and computing power have its merits in something as vague as bluffing.

The plan based reasoning about the best move also turned out to be a very convenient way to formulate knowledge. Even tactical close-distance encounters could be implemented in this system with a strength that is equal to what can be done with exhaustive search and a depth in plies that is much larger. Strategic long-term notions that are impossible to implement by exhaustive search were very easily formulated as plans. The knowledge is stored in only 17 different plans, which makes it much easier to maintain than a rule-based system that might need thousands of rules to reach this level.

To reach a higher level, Invincible will need more plans. There is certainly room for this. The current implementation is by no means the limit of what can be reached by these algorithms but it will take time to make a considerable improvement. Because of the relatively low amount of plans and the strict division in responsibilities it should be relatively easy to increase the knowledge base, though the programmer should pay careful attention not to let plans overlap.

9.3 Recommendations

In this chapter I will give a few recommendations for future work.

9.3.1 Adding more knowledge

To improve the level of Invincible, more knowledge should be added in the form of new plans to cover situations in which Invincible currently makes mistakes. It is particularly weak in defending against aggressive players so it could use some sort of plan trying to trick aggressive attackers in making wrong guesses. There are many ways to do this and catching this in a generally useful plan might be difficult, but it should certainly be possible. Another aspect on which it could be improved is to attack against a partially unknown defense early in the game. He can do this, but usually not very effective. This is difficult for human players as well, but it should be possible to fit it in Invincible's architecture.

9.3.2 Opponent modeling

No program can reach a really high level in Stratego without using opponent modeling. Since human players do remember things about the style and setups of the computer he gets better results with every next game. It is vital for the computer program to learn something about the player's style and adapt to that to stay competitive over a number of games. Things that could be changed depending on the opponent's style could include the setups (solid setups against aggressive gamblers and surprising setups against careful players) or the value given to bluffing moves (bluff more against people who often believe bluffs and bluff less against people who often call bluffs). It would also be possible to store information about the setups the player uses, but this might make the advantage of the computers perfect memory too large if the player accidentally uses the same setup twice.

9.3.3 Other applications

The plan-based architecture can be used for different problems that are too complex for exact computation. It supplies a simple framework for defining sub problems. It holds the middle between exact computation and a rule-based system, having less and more complicated "rules" than the latter. The plans allow relatively simple calculations of one aspect of the whole problem, dividing the complex problem into as many simple problems as necessary.
In games, this might be used for the Asian game of "Go" which has complete information, but an immense search space, having hundreds of possible moves in one ply. Human players in go use a lot of patterns and local directional searches in combination with a general strategy on the whole board. This would very well fit into the architectural structure that I made for selecting moves, but would again require a lot of heuristic knowledge from go-experts.

It also might be used for computer strategy games, where plans are often long-term and only weakly inter-related.

Bibliography

[1] Ed's Stratego site: http://www.edcollins.com/stratego/
[2] Gravon – gamers paradise: http://www.gravon.net/
[3] StraDoS2: Stratego Documentation System:
http://www.gravon.de/english/stratego/strados2.php
[4] M.Heule, R.J.M. Rothkrantz. *Solving games. Dependence of applicable solving procedures.* Science of computer Programming No 67,p105-124 2007
[5] Min-max algorithm: http://en.wikipedia.org/wiki/Minimax
[6] Alpha-beta pruning: http://en.wikipedia.org/wiki/Alpha_beta_pruning
[7] Dijkstra algorithm: http://en.wikipedia.org/wiki/Dijkstra_algorithm
[8] Deep Blue Chess: http://en.wikipedia.org/wiki/IBM_Deep_Blue
[9] Othello AI: http://en.wikipedia.org/wiki/Reversi
[10] Darpa: http://www.darpa.mil/
[11] Tactical Language & Culture Training System: http://www.tacticallanguage.com/
[12] Metaforge Webstratego: http://www.metaforge.net
[13] Stratego Bond Nederland (SBN): http://www.strategobond.nl
[14] International Stratego Rating: http://www.kleier.net/rating/index.php5
[15] Karl Stengaard, *Utveckling av minimax-baserad agent för strategispelet Stratego.* 2006 Lund University, Sweden
[16] Jörg Bewersdorff, *Luck, Logic and White Lies,* 2005
[17] Jean-Paul van Waveren, Léon J. M. Rothkrantz: *Artificial Player for Quake III Arena. Int. J. Intell.* Games & Simulation 1(1). 2002.
Jean-Paul van Waveren, Léon J. M. Rothkrantz: *Automated path and route finding through arbitrary complex polygonal worlds, robotics and autonomous systems.* Robotics Journal vol 54, p 442-452, 2006
[18] John W. Romein, Henri E. Bal, *Solving Awari with Parallel Retrograde Analysis* 2003, Vrije Universiteit Amsterdam.
[19] James Edward Davis and Graham Kendall, *An Investigation, using Co-Evolution, to Evolve an Awari Player*, University of Nottingham, United Kingdom. 2005.
[20] Mohammed Daoud, Nawwaf Khannal, Ali Haidar and Julius Popoola, *Ayo, the Awari Player, or How Better Representation Trumps Deeper Search*
Concordia University, Montreal, Canada.
[21] 2007 Computer Stratego World Championships:
http://www.strategousa.org/wiki/index.php/2007_Computer_Stratego_World_Championship
[22] Stefan J. Johansson, *On using Multiagent Systems in Playing Board Games*
[23] Caspar Treijtel. *Multi-agent stratego*. Master's thesis, Delft University of Technology, 2000.
[24] J. Schaeffer. *Solving checkers: First result.* International Computer Games Association (ICGA), 2005.
[25] Jeffrey Hargrave, Flora Basinger and Mark Zarqawi, *Investigation of Expert Systems*
[26] Adam Conner-Simons *What games can humans still win?*
http://www.gelfmagazine.com/gelflog/archives/what_games_can_humans_still_win.php
[27] David M. Bourg; Glenn Seeman, *AI for Game Developers.*

ISBN: 0-596-00555-5. 2004
[28] Stratego AI-bot “Probe”: http://www.imersatz.com/probe/index.htm
[29] Mohannad Ismael *Multi-Agent Stratego*
University of Rotterdam, Delft University 2004

Appendix A: Stratego

A1 The game

The rules of the game are taken from
http://files.boardgamegeek.com/viewfile.php3?fileid=14131

The object of the game is to capture your opponent's flag.
Each army consists of:
--- These moveable pieces, (#) = indicates quantity
10 Marshal (1)
9 General (1)
8 Colonel (2)
7 Major (3)
6 Captain (4)
5 Lieutenant (4)
4 Sergeant (4)
3 Miner (5)
2 Scout (8)
1 Spy (1)
Note: Higher number indicates higher rank
--- These immobile pieces
Bomb (6)
Flag (1)

How to set up the game

1. Place the game board between you and your opponent with the name Stratego facing each of you.
2. Hide a red piece in one hand and a blue piece in the other. Your opponent chooses a hand and plays with the color army the selected piece designates. The other color army is yours.
3. Set up your armies using the strategy hints and rules for movement and attacking that are discussed below.
4. Place your pieces on the game board with the notched end up. The printed side faces you so your opponent cannot see the rank of your pieces. Your opponent does the same.
5. Only one piece can occupy a square. Place them anywhere in the last four rows on your half of the game board. The two middle rows are left unoccupied at the start of the game.

Game play

You and your opponent alternate turns. The red player moves first.
On your turn you can do one of the following:
Move -- one of your playing pieces to an open adjacent space.
Or Attack -- one of your opponents playing pieces.

Rules for movement

1. Pieces move one square at a time, forward, backward or sideways. (Exception: see Special Scout Privilege, Rule 6).
2. Pieces cannot move diagonally. They cannot jump over another piece. They cannot move onto a square already occupied by another piece (unless when they are attacking).
3. Pieces cannot jump over or move onto the two areas in the center of the game board that are indicated by the dotted lines.
4. A piece cannot move back and forth between the same two squares in three consecutive turns.
5. Only one piece can be moved on a turn.
6. Special Scout Privilege: A Scout can move any number of open squares forward, backward, or sideways. But remember, this movement will let your opponent know the value of that piece. You may wish to move your Scouts one space at a time to confuse your opponent. Scouts are the only pieces allowed to move and attack on the same turn. See “Rules for Attack”, below.

Remember, the Bomb and Flag pieces cannot be moved and must remain on the squares where they were originally placed throughout the game.

Rules for attack

1. Attack Position: When a red and a blue piece occupy adjacent spaces either back to back, side to side, or face to face, they are in a position to attack.
2. How to Attack: To attack on your turn, take your attacking piece and lightly tap your opponent's piece. Then, declare the rank of your attacking piece. Your opponent then declares the rank of his/her defending piece.
3. The piece with the lower rank is captured and removed from the board. If your piece (the attacking piece) is the remaining and winning piece, it moves into the space formerly occupied by the defending piece. If the remaining and winning piece is the defending piece, it stays on the square it was in when it was attacked.
4. When pieces of the same rank battle, both pieces are removed from the game.
5. Attacking is always optional.

Rules of rank

1. A Marshal (Number 10) outranks a General (Number 9) and any other lower-ranking piece. A General (Number 9) outranks a Colonel (Number 8) and any lower-ranking piece. A Colonel (Number 8) outranks a Major (Number 7) and so on down to the Spy which is the lowest-ranking piece. Refer to the illustration of moveable pieces on page 2 for remaining ranks.
2. Special Miner Privilege. When any piece (except a Miner - ranked 3) strikes a Bomb, that piece is lost and removed from the board. When a Miner strikes a Bomb, the Bomb is defused and removed from the game board. The Miner then moves into the Bomb's space on the board. Bombs remain on the same square throughout the game unless they are defused. Bombs cannot attack or move.
3. Special Spy Privilege. If any piece attacks the spy, it is captured and removed from the board. But the Spy has a unique attacking privilege. It is the only piece that can outrank a Marshal providing the Spy attacks the Marshal first. If the Marshal attacks first then the Spy is removed.

Strategy hints

1. Place Bombs around the Flag to protect it. But place a Bomb or two elsewhere to confuse your opponent.
2. Put a few high-ranking pieces in the front line, but be careful! If you lose them early in the game you're in a weak position.
3. Scouts should be in the front lines to help you discover the strength of opposing pieces.
4. Place some Miners in the rear for the end of the game, where they will be needed to defuse Bombs.

Winning the game

The first player to attack an opponent's Flag captures it and wins the game.
If all of your moveable pieces have been removed and you cannot move or attack on a turn, you must give up and declare your opponent the winner.

A2 History

This text was taken literally from wikipedia: http://en.wikipedia.org/wiki/Stratego

The origins of *Stratego* can be traced back to traditional Chinese board game "Jungle" also known as "Game of the Fighting Animals" (*Dou Shou Qi*) or "Animal Chess". The game Jungle also has pieces (but of animals rather than soldiers) with different ranks and pieces with higher rank capture the pieces with lower rank. The board, with two lakes in the middle, is also remarkably similar to that in *Stratego*. The major differences between the two games is that in Jungle, the pieces are not hidden from the opponent, and the initial setup is fixed.

A modern, more elaborate, Chinese game known as *Land Battle Chess* (*Te Zhi Lu Zhan Qi*) or *Army Chess* (*Lu Zhan Jun Qi*) is a descendant of Jungle, and a cousin of *Stratego* - the initial setup is not fixed, one's opponent's pieces are hidden, and the basic game play is similar (differences include "missile" pieces and a Chinese Chess style board layout with railroads and defensive "camps"; a third player is also typically used as a neutral referee to decide battles between pieces without revealing their identities). An expanded version of the *Land Battle Chess* game also exists - this adds naval and aircraft pieces and is known as *Sea-Land-Air Battle Chess* (*Hai Lu Kong Zhan Qi*).

In its present form *Stratego* appeared in Europe before World War I as a game called ***L'attaque***. Thierry Depaulis writes on "Ed's *Stratego* Site":[1]

"It was in fact designed by a lady, Mademoiselle Hermance Edan, who filed a patent for a 'jeu de bataille avec pieces mobiles sur damier' (a battle game with mobile pieces on a gameboard) on 11-26-1908. The patent was released by the French Patent Office in 1909 (patent #396.795). Hermance Edan had given no name to her game but a French manufacturer named "Au Jeu Retrouvé" was selling the game as L'Attaque *as early as 1910... "*

Depaulis further notes that the 1910 version divided the armies into red and blue colors. The rules of *L'attaque* were basically the same as the game we know as *Stratego*. It featured standing cardboard rectangular pieces, color printed with soldiers who wore contemporary (to 1900), not Napoleonic uniforms.

The modern game, with its Napoleonic imagery, was originally published in the Netherlands by Jumbo, and was licensed by the Milton Bradley Company for American distribution, and first published in the United States in 1961 (although it was trademarked in 1960). The Jumbo Company continues to release European editions, including a three- and four-player version, and a new **Cannon** piece (which jumps two squares to capture any piece, but loses to any attack against it). It also included some alternate rules such as **Barrage** (a quicker two-player game with fewer pieces) and **Reserves** (reinforcements in the three- and four-player games). The four-player version appeared in America in the 1990s.

Other themed variants appeared first in North America: a *Star Wars* version, a *The Lord of the Rings* variant, and a "Legends" variant with fantasy pieces arguably inspired by *Magic: The Gathering*. The Legends variant added more rules and complexity, giving the players choices of pieces with special attributes, collectible "armies" from more than a hundred individual pieces offered in six sets, and varied boards with terrain features.

Pieces were originally made of printed cardboard. After World War II, painted wood pieces became standard, but starting in the late 1960s all versions had plastic pieces. The change from wood to plastic was not made so much for economy, but because the wooden pieces tended to fall over and the plastic pieces could be designed not to. European versions introduced cylindrical castle-shaped pieces that proved to be popular.

American variants later introduced new rectangular pieces with a more stable base and colorful stickers, not images directly imprinted on the plastic.

The game is particularly popular in the Netherlands, Germany and Belgium, where regular national and world championships are organized. The international *Stratego* scene has, in recent years, been dominated by players from the Netherlands.

European versions of the game show the Marshal rank with the numerically-highest number (10), while American versions give the Marshal the lowest number (1) to show the highest value (i.e. it is the #1 or most powerful tile). Recent American versions of the game which adopted the European system caused considerable complaint among American players who grew up in the 1960s and 1970s. This may have been a factor in the release of a *Nostalgic* edition, in a wooden box, reproducing the *Classic* edition of the early 1970s.

Electronic Stratego was published by Milton Bradley in 1982. It has features that make many aspects of the game strikingly different from those of classic *Stratego*. Each type of playing piece in *Electronic Stratego* has a unique series of bumps on its bottom that are read by the game's battery-operated touch-sensitive "board". When attacking another piece a player hits his Strike button, presses his own piece and then the piece he is targeting: the game either rewards a successful attack or punishes a failed strike with an appropriate bit of music. In this way the players never know for certain the rank of the piece that wins the attack, only whether the attack wins, fails, or ties. Instead of choosing to move a piece, a player can opt to "probe" an opposing piece by hitting the Probe button and pressing down on the enemy piece: the game then beeps out a rough approximation of the strength of that piece. There are no bomb pieces: bombs are set using pegs placed on a touch-sensitive "peg board" that is closed from view prior to the start of the game. Hence, it is possible for a player to have his own piece occupying a square on which a bomb has been placed. If an opposing piece lands on the seemingly-empty square, the game plays the sound of an explosion and that piece is removed from play. As in classic *Stratego*, only a Miner can remove a bomb from play. A player who successfully captures the opposing Flag is rewarded with a triumphant bit of music from the *1812 Overture*.

A3 Repetition rules

In Stratego, there are 2 rules to prevent endless repetition of moves. Their exact definition is quite complicated, which is probably the reason most existing programs have either no notion of them, or have not implemented them completely or incorrectly. It has to be said that for leisurely play these rules have no real significance since you normally don’t get into an endless repetition if you play just for fun. And even in real-life official games, they are only guidelines about when approximately you are supposed to end the chase. Since the games are not annotated it’s impossible to say the exact moment when a repetition of moves becomes illegal.

There are several reasons why I find it necessary to implement the official rules in "Invincible" and in the game client. First of all, it's impossible to play a serious game without repetition rules at all. If both sides are determined to win, there are countless situations in Stratego that can lead to a stalemate if neither is forced to do a different move. Where the human player might decide to do a sub-optimal move when he gets bored with the situation, the computer would never do that if not restricted. Also, since it's impossible to argue with the computer, I find it important to make sure the computer is always right in which moves are allowed. In possible future Stratego AI competitions, it will be important that both AI-bots use the same set of rules so they must agree in all the details of these repetition rules as well.

The two-squares rule

The first of these rules is the two-square rule, also called 5-moves rule, though I prefer to use the first term because the essence of this rule is that the repetition of moves is over 2 squares and the actual number of moves is not important (in tournaments they are hardly ever counted).

Taken from the official Stratego rules [1] this reads literally

10 Five Moves on Two Squares: Two-Squares Rule

10.1

> *It is not allowed to move a piece more than 5 times non-stop between the same two squares, regardless of what the opponent is doing. It does not matter whether a piece is moving and thereby attacking an opponent's piece, or just moving to an empty square.*

10.2

> *When a scout is involved in the Two Squares Rule, a scout is considered to start on the starting position of his move plus all the squares he steps over, and he ends on the final position of his move plus all the squares he steps over.*

10.1 seems easy enough, though of all the repetition rules this has the biggest impact on the game, which I will explain later. Most other Stratego sites and AI-bots have implemented this rule, but none of the AI-bots I've come across seemed to know its consequence to the game. This rule is very common and restricts moves several times in each game. It often results in the capture of a piece, not only because of mistakes of the opponent, but also because of tactical combinations of moves making use of this rule.

10.2 makes this rule a little more complicated to program, but is very important when trying to capture a scout or defending your flag against scouts. Without this rule (only taking 10.1 literally) you need about 6 pieces to capture a scout, with this rule it can be done with 4. There are not so many situations in which this rule really makes a difference; only in very close endgames it's important to understand this rule, so I will not go into detail here. To understand the rule, you must look at the lines between the

squares, if a line is crossed 5 times on row by the same piece then this piece is not allowed to cross it in the next move.

Implementation of the two-square rule

Because of the 10.2 clause this is not a simple check on the last 5 moves to see if they went back and forth between the same 2 squares. The problem can be reduced to finding the overlapping squares in the last 5 moves. If all 5 moves started, ended or passed over at least 2 of the same squares, then the current move is not allowed to start end or pass over these 2 or more squares. This can be implemented by finding the maximum of the lowest y-coordinate of the last 5 moves and the current move, and the minimum of the highest y-coordinates if they all started and ended in the same row. If the minimal highest y-coordinate is higher than the maximal lowest y-coordinate there are at least 2 squares overlapping in these moves so the current move is not allowed.
If they all started and ended in the same column, the same check has to be applied to the x-coordinates. If the moves don't all take place in the same row or column, the move can't break this rule.

If the last 5 moves were not all done by the same piece, then this rule can never be broken. It is important to check this, since in theory it's possible to cross the same 2 squares with 2 scouts and then move back with the last one. This would cross the same 2 squares for the 6th time on row, while the move is allowed.

Game play effects of the 2-square rule

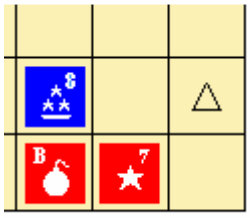
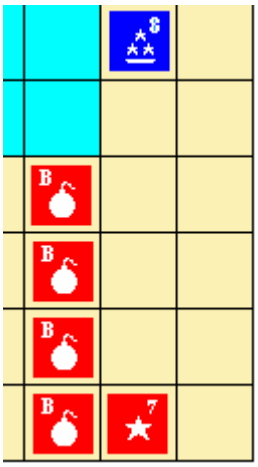


Illustration of the 2-square rule
If the blue colonel moves to the right, there inevitably follow 2 repetitions over 2 squares. The red major will try to evade on the first row, and the blue colonel follows on the 2nd row. Since the red major must do the first repetitive move he also has to end it first and thus will be lost!

Figure A1 – The major (7) lost?

If however the blue colonel would start in this situation (after moves elsewhere on the board) on the square marked with a triangle and would from there move to the left, the colonel would start the repetitive moves earlier than the major and won't succeed in the capture. These are two seemingly alike situations, but with a very different result. Even though in the first situation the blue colonel seems to initiate the sequence, he is not the one that starts the repetition, since he doesn't start on one of the squares between which the moves are repeated.



The key is that the first piece that crosses the line between the 2 squares of the repetition is at a disadvantage
The attacker should approach from above or (in this situation) from the left, and the defender can anticipate by moving to the right before the colonel moves into his column, to make sure he can move to the attackers column as soon as the attacker moves to his column. As soon as both pieces are in the same column for a move and there is no escape from the two squares the piece has to move back and forth between, the piece is lost. In the situation in figure 2, the red piece is already lost. If he moves to the right, the attacker does the same. After 5 times, red must do another move, and blue can move down, eventually capturing the red piece.

Figure A2 – The major (7) lost?

Since a single mistake with this rule can easily cost you an important piece and decide the game, it is important that the artificial intelligence has a good understanding of this rule and its consequences.

The more-square rule

The second repetition rule is the more-square rule. This rule is a lot more vague and impossible to apply precisely in a real non-annotated game so it's usually known as "You can't endlessly chase enemy pieces without giving him at least one free move in which he doesn't have to run away from you". A seemingly innocent rule that makes the attacker stop a chase that doesn't lead to the capture of a piece. The rule can't force a player to let his piece be captured like the two-square rule could since it's always the attacking player that has to stop the repetitive moves. Like the two-square rule though, this rule also has its side-effects that are of no less consequence to the result of the game, though its importance becomes clear only in close endgames, and is thus less often decisive than the two-square rule, but when it is, it always decides the result of the game.

The less common occurrence and the vague nature of this rule are most likely the reason that of all the Stratego rules this one is least commonly known. Even experienced tournament players do not always know precisely its consequences to the endgame. Only one Stratego site that I know of implements this rule [2], and no Stratego AI-bot that I came across does. The reason for this is most likely that the commonly known vague version of the rule is not clear enough to be implemented (actually it would take an extraordinary effort from a player to be able to tell when exactly a move becomes illegal during a live game, so this is not expected of players, only that the attacker "eventually" stops) and the official description is very formal and can be hard to read if it's not explained to you.

From the ISF game rules [1]:
11 Repetition of Threatening Moves: More-Squares Rule
11.1

> *It is not allowed to continuously chase one or more pieces of the opponent endlessly. The continuous chaser may not play a chasing move again more which would lead to a position on the*

board which has already taken place.

11.2

Exception: chasing moves back to the square where the chasing piece came from in the directly preceding turn are always allowed as long as this does not violate the Two-Squares Rule (Five-Moves-on-Two-Squares).

11.3

Definitions:

- *continuous chase: the same player is non-stop threatening one or more pieces of his opponent that is/are evading th threatening moves.*
- *chasing move: a move in a continuous chase that threatens an opponent's piece that was evading during the continuous chase.*

Hereby

- *a/to move: a/to move plus attacking or a/to move to an empty square.*
- *to threaten: to move a piece next (before, behind or besides) a piece of the opponent.*
- *to evade: to move a piece away promptly after it has been threatened.*

In short that means you are not allowed to do a move that would lead to a situation on the board that has already taken place during an uninterrupted sequence of threatening moves. After a single non-threatening move, all history is thrown away and all moves are allowed again.

Implementation of the more-square rule

To implement this rule, I keep 2 lists of hashed board positions. After every move of red, the board position goes into list A, and after every move of blue, the board position goes into list B. If a blue move does not start on a square next to the previous red move, the red move is not considered a chasing move, so reds list (A) is emptied. A similar check is applied to see when list B must be emptied.

Any move to that would lead to a board position that is already in the list is not allowed.

The exception that you can move back to the square you came from must be taken into account separately. A move going back to the field this piece came from in the previous move can always be allowed by this rule (and then possibly forbidden by in the other check you do for the two-square rule).

A simplification may be that a capturing move can never be part of a repetitive sequence, so you can omit this check on moves that attack an enemy piece.

One thing where I found the rule unclear is whether a board situation is the same if all the pieces are in the same place and have the same ranks, but different pieces of the same rank have exchanged places.

Because of the secretive nature of the game, it must be possible however to tell which moves are allowed without knowing the ranks of the opponents pieces, so the only 2 options would be to treat all pieces equally and thus consider a board situation the same when there are pieces of the same color on the same fields, or to treat them all differently and consider a board position the same only if each individual piece is in the same

location. The first seems clearly wrong because if a marshal and scout have exchanged places there are different possibilities and it's thus a different position so I interpret this rule to read that every individual pieces must be on the same field before a board position is considered the same. To implement this, all pieces are given a unique id that is used in the hash function for the board. The hash function simply puts all these codes for the 100 squares in sequence, resulting in a String that is unique for every different board position.

Game play effects of the more-square rule

Though it seems that this rule can't affect the result of the game, since it can't lead to the unnecessary capture of a piece, there is a side-effect. In close endgames it's possible to create a situation where the move you have to do to defend your flag is not allowed by this rule, forcing you to do another and give your opponent free access to your flag.

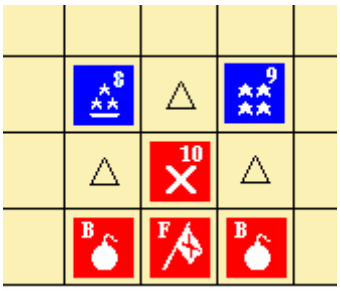


Figure A3 – Blue wins.

Figure 3 shows a situation in which the more-square rule decides the result of the game. Suppose that the red marshal (10) is the only piece red has left and blue has no miners left to capture the red bombs. It's reds turn to move. Normally, the marshal would move back and forth between any of the squares marked with a triangle. As soon as the 2-square rule would forbid that, red can move back and forth between any of the other squares.

This can go on forever, without the marshal ever moving too far from the flag to allow blue to capture it. However, if the blue pieces stay in these positions and only move to one of the triangle fields when the marshal comes next to them on the other triangle field, then each move of red is seen as a threatening move! Even though the blue pieces stay there on purpose and might be seen as the attacking pieces, and red can't capture either of them because then the other will go to the flag. What counts is the action of red moving next to a blue piece, and blue to move this piece away in the following move.
Since this situation is thus by the definition a continuous chase, red will be forced eventually to make a move that doesn't end next to a blue piece. In the situation above that means red will instantly lose because he has no valid moves, and if red would be on one of the triangles he would be forced to move outward, letting blue capture his flag.

Note that if red would have another piece, for example a scout, this would not work for blue, since red would not be forced to move the marshal.

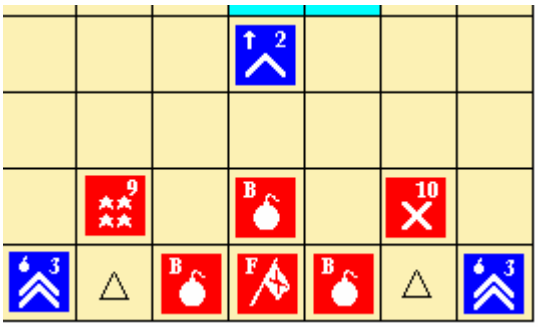


Figure A4 – Blue wins.

This rule also applies when more chasing and running pieces are involved. See figure 4: Without other pieces, the red marshal and general can't defend the flag against 2 miners and a scout (the scout is to make moves, since blue would lose if his miners were forced to move in this position). Red can't defend endlessly by going back and forth to the marked fields.

There are many different situations imaginable in which one player can force the other player out of a defensive position using this rule. The effect of this rule on the game is that as soon as you have so few pieces left that you can be forced to move an important piece you lose, regardless of how high the pieces you have left are.

If you have the 2 highest pieces on the board, but the opponent has more pieces, you have a losing endgame. With 3 highest pieces it can go either way depending on the situation, only if you have the 4 highest pieces it's enough to guarantee the win.

Appendix B: The game client

In order to be able to show and test the artificial intelligence, a user interface is needed that allows people to play a game against the computer. I therefore started with making a simple program that allows you to play games, save setups and replay previously saved games to see where, on hindsight, the mistakes were made. This also makes it easy to let other Stratego players test the artificial intelligence and send me the save files so that I can analyze the game later.

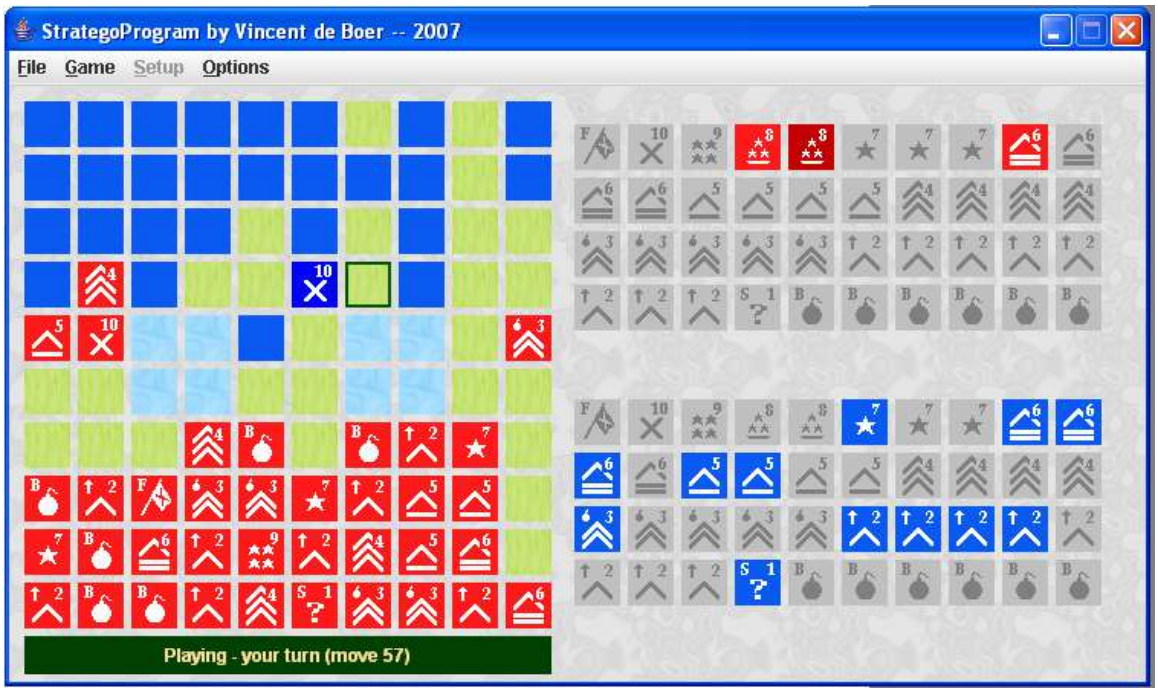


Figure B1 – Screenshot of the game client.

On the left is the game board, you can click on pieces and then on the square you want to move to to make moves. Captured pieces go to the right of the screen. The grey pieces are still in the game, and the colored pieces are already captured. During setup phase, all the pieces are initially there and can be moved to the board by clicking on them and on a field on the board. To speed up the setup process, the next highest piece is automatically selected when you place a piece on the board.

The pieces

To avoid any possible copyright issues I decided to make my own set of images for the pieces. The idea was to make them easily distinguishable symbols with as few "useless" details as possible.

	Marshal	The cross is often associated with the marshal in different versions of Stratego. In annotations, the letter X is usually used to refer to the marshal.
	General	The 4-star rank ensign used for the general in the classic Stratego

		pieces.
	Colonel	The 3-star rank ensign used for the colonels in the classic Stratego pieces.
	Major	The 1-star rank ensign used for the majors in the classic Stratego pieces.
	Captain	
	Lieutenant	
	Sergeant	
	Miner	The bomb icon is added to show that this is a special piece that can capture bombs.
	Scout	The arrow icon is added to show that this is a special piece that can move faster than normal pieces.
	Spy	The spy can capture the marshal, so also has a sign in the upper left corner to show it's a special piece.
	Bomb	Bomb is special because it can't move, but beats every attacker except the miner.
	Flag	The flag can't move and if lost you lose the game.

Saving setups

To save a setup you should click "save" in the setup menu after completing your whole setup and before starting the game. It is then automatically saved on the first empty slot (you have 6 slots for setups) and called setup #n. To delete a setup, you must load it and then select delete from the setup menu.

Saving games

You can only save finished games. To save a game, go to the file menu and click "save". The game is automatically saved as "[result] in [n] moves". Only 10 games can be saved, after this the oldest saved game is automatically deleted. To manually delete a savegame you must load it and then select delete from the file menu. You may want to do this to prevent an older game to be auto-deleted.

Replaying games

To replay a game you must first save them. You can load a saved game by selecting it in the file menu. You then get a "forward" and "backward" button under the board with which you can scroll through the moves. You can select "open pieces" from the options menu to see the ranks of the computer so you can see where all his pieces were standing during the game.

Free setup

If you select free setup from the options menu you can manually insert any position on the board. You can (and have to) setup both the blue and the red pieces. It is not necessary to put all the pieces on the board and it's not necessary to put them only in their setup area. When you have finished setting up, you can start the game by selecting "start" from the game menu. Free setup games don't count for your win/loss statistics and they can be played with open pieces (which is normally only available during replay of a game).

The menus

File

- New game – starts a new game against "Invincible".
- Save – saves a finished game for replay later.
- Delete – deletes the loaded savegame
- List of saved games – The last 10 saved games can be loaded for replay, optionally with open pieces. Moves can be played and taken back to see what went good or bad in the analysed game.
- Quit – quits the program.

Game

- Start – starts the game. Only possible when all your pieces are setup.
- Offer draw – offers a draw to "Invincible". If accepted, the game ends in a tie.
- Surrender – resigns from the game. This results in a loss.

Setup

- Random setup – makes a random setup
- Clear setup – removes all the pieces already on the board and puts them back in the box
- Save setup – saves the current setup. This is only possible when all pieces are placed and the game hasn't started yet. The game is saved on the first empty of the 6 available slots. If there are no empty slots, the setup can't be saved.
- Delete setup – deletes the currently loaded setup. To delete a setup it must first be loaded.
- List of setups – clicking on any of the setup slots loads the setup saved in that slot. It can be modified or deleted before the game starts.

Options

- Statistics – shows your win/loss record against "Invincible".
- Free setup – allows the user to put any position on the board to start the game with. If the game is in this mode, the win/loss record is not affected.

- Open pieces – only available during replay or free setup games. Shows the ranks of all enemy pieces.

Appendix C – testgames

Game 1: Mohannad vs. Invincible (1-0) – Mohannad wins in 313 moves

- Move 3-4: Invincible moved his marshal early and Mohannad attacked it with a scout. This is bad for Invincible because playing with a known marshal is very hard. Given his inability to dominate the game aggressively, he cannot afford to let his high pieces get known so easily.
- Move 16-22: Invincible advances his marshal and Mohannad moves his spy in defense. Invincible is careful and doesn't come near the spy.
- Move 25-29: Invincible takes a few unmoved pieces in an attack with a lieutenant. The attack ends when he attacks Mohannad's lieutenant.
- Move 31: Invincible scouts Mohannad's unmoved general. This is good, because with a known marshal an unknown enemy general is very dangerous. The game is still approximately even.
- Move 32-57: there's a skirmish between Mohannad's known captain and Invincible's unknown major. There is no confrontation between the pieces because Mohannad suspects it is a major and approaches with his general. The captain hits an unmoved captain from Invincible and the majors are later exchanged too.
- Move 59: Invincible scouts Mohannad's colonel. Invincible is getting more information, while the pieces are still approximately even. Invincible's problem is that he doesn't know the marshal and spy so none of his high pieces dares to attack.
- Move 73-78: Mohannad takes a few unmoved pieces in an attack with a lieutenant. The attack ends when the lieutenant hits a bomb.
- Move 87-91: Mohannad takes a few unmoved pieces in an attack with a sergeant. The attack ends when the sergeant hits another sergeant. Invincible remains passive because he sees no good attack.
- Move 103: Invincible scouts a backrow miner.
- Move 103-117: Both Mohannad and Invincible stay passive, neither eager to attack.
- Move 118-136: Invincible had more patience, so Mohannad started an attack with a major attacking Invincible's back lines. This works very well, the major takes two captains and a few more pieces but eventually hits a bomb. This makes Mohannad suspect where Invincible's flag is. Mohannad is now behind a major, but this is more than compensated in the lower ranks. Also Invincible still has too little information to attack.
- Move 145-177: Mohannad attacks with a colonel and miner trying to get to the flag. Invincible knows the colonel and approaches it with 2 pieces. He succeeds in taking it with his general. Things look a bit better for Invincible now, but he still has dangerously few low pieces and Mohannad still suspects his flag.

- Move 180-213: Mohannad suspects the spy and tries to capture it with a scout. Invincible defends well with the general and marshal and Mohannad gives up, scouting a major instead.
- Move 214-236: Mohannad attacks with his unknown marshal and known general. Invincible defends with known marshal and known general. Invincible is carefull of the unknown piece, not attacking it with his general, but not giving him a chance to scout the spy either. It ends when Mohannad attacks the general with his marshal and loses his marshal. This is a fair trade, but now Invincible knows Mohannad's highest piece (the general) so he is finally able to attack.
- Move 249-260: Mohannad attacks on the flag side with a major. Invincible's defenses are too thin there because the marshal was drawn to the fight in the middle of the board. He pushes back the remaining defender and hits an unmoved piece in front of the flag, weakening this side even further. He then hits a bomb in front of the suspected flag, making it even more likely that the flag is actually there. Invincible is leading by a colonel and two majors, but has few low pieces and a weak flag side. He doesn't consider himself up enough to close his defense. He underestimates both his lead and how obvious his flag has become.
- Move 260-274: Invincible instead attacks with a marshal and colonel, pushing the general back in defense. He takes a strong position in the centre. His path to the flag-side is blocked by the spy and general though, so his flag defenses are dangerously weak.
- Move 279: Invincible takes the miner that he scouted in move 102 with his colonel.
- Move 288-303: Invincible moves his second colonel to attack too and traps a moved piece in the centre. Mohannad moves with a miner to the suspected flag. He takes the bomb and Invincible hits his miner with his own.
- Move 303-313: Both Invincible and Mohannad attack. Invincible advances his spy to use it for more information; Mohannad advances a sergeant to the flag. Invincible makes no move to defend, still considering his lead insufficient and preferring attack.
- Move 314: Mohannad captures the flag and wins the game.

Summary Mohannad vs. Invincible (1-0):

Invincible played a very careful game, but put too little pressure on his opponent. His first mistake was to move his marshal too early. He didn't really have anything he could do with it so it only gave away important information. He also doesn't really know how to find the information he needs and this limits his ability to attack. He gave Mohannad too much freedom to dominate the game and this was not in his advantage. His second mistake was that he didn't recognize he had to defend the flag. He only gives much value to defense when he has a very clear lead, but should also do so when his flag is vulnerable in games that are even or where he has a small lead.
To improve, the program needs to get a plan to find the location of the highest enemy piece and the spy and the strategic defense plan must be activated when the flag becomes vulnerable. He also should not move his marshal as early as he did this game.

Game 2: Raymond vs. Invincible (0-1) – Invincible wins in 494 moves

- Move 1-22: Both players attack with a couple of low pieces to test out the opponent. Both lose some and by move 22 Raymond knows a bomb and Invincible knows a captain.
- Move 23-24: Invincible captures the known captain with his major, and loses his major to a marshal. This is not a bad exchange for either player because Raymond get's a small material lead while Invincible gets to know the marshals location.
- Move 25-43: Raymond moves a couple of low pieces to defensive positions and Invincible moves his unknown general forward through the center. He gets know for a scout and blocked by the marshal.
- Move 44-59: The marshal let the general pass and this costs Raymond a sergeant and miner that he moved earlier. He also had to move his spy out in the center to get it out of the way.
- Move 50: Raymonds captures an attacking captain with his major on the right side of the board.
- Move 63: The generals get exchanged. This stops Invincible's attack, but Raymonds general was not known yet, so Invincible gains more from this swap. The game is still about even, with Raymond leading by a major, and Invincible being ahead two low pieces and having more information.
- Move 64-68: Raymond attacks with his known major and takes 2 scouts. Even though these are low ranking pieces it didn't seem necessary to sacrifice both to a known major.
- Move 68-77: Invincible pushes the major back with an unknown piece and captures it, revealing the piece to be a colonel.
- Move 93-98: Invincible does a wild attack with a captain, takes a lieutenant and then loses to a colonel.
- Move 108-109: Raymond takes a known front-row bomb with a miner and loses his miner to a lieutenant.
- Move 110-120: Both Raymond and Invincible attack on opposite sides. Raymond loses his captain to a major after taking the known lieutenant and Invincible takes an unknown bomb with a miner (!) and then get's captured by Raymond's last unknown colonel.
- Move 123: Two known colonels get exchanged.
- Move 124-139: Invincible does a very weak bluffing attack on the left with a scout on Raymond's colonel. The colonel is not impressed and the scout retreats (unknown) when Raymond moves his marshal to the left as well.
- Move 140-145: With the marshal out of the center, Invincible takes his chance to attack there with his unknown colonel. Raymond meets him with a major, but Invincible thinks this is a low piece and avoids confrontation. He takes a captain instead.
- Move 147: Invincible is given the chance to move his colonel in between a major and the spy, both unknown, but he knows both can be captured by a colonel but he lets the moment pass.

- Move 153: Invincible hits an unknown major with his known major.
- Move 158-174: Raymond attacks 3 of Invincible's pieces in the back of the field with scouts. He hits another scout a bomb and unexpectedly also the spy. Invincible still has a slight advantage in the game, it's even in high pieces but Invincible is leading both in small pieces and in information. The only difficulty might be the loss of his spy.
- Move 193-227: Raymond attacks on the right with a major, lieutenant and miner. It leads to nothing because an unknown piece approaches in defense. Invincible also unexpectedly approaches with his marshal though the center and captures the unknown major. Now Invincible has a serious lead in pieces.
- Move 240-285: Invincible is indecisive now. He leaves his highest two pieces in an attacking position on the right side, but there is hardly anything left there to attack. This gives Raymond a lot of options on the rest of the board. Invincible should have moved his marshal to a more active position.
- Move 290-333: Raymond takes advantage by attacking with a miner and colonel to the suspected flag. Invincible defends with too many pieces and Raymond misses a few chances to capture some with his colonel. Invincible eventually manages to capture the miner and remove the danger.
- Move 353: Invincible attacked with a lieutenant and captures a sergeant.
- Move 369: Invincible finally decides to simplify the game by exchanging colonels. This is a big step towards victory because he is leading in all ranks lower than colonel.
- Move 400-427: Invincible attacks with a major, taking a sergeant and following two pieces that are running towards Raymonds marshal. He misses all chances to catch up and capture one of them. This is again the same mistake that he previously made where he didn't see the opportunity to step between 2 pieces that are known to be lower.
- Move 428-434: Raymonds gets trapped which costs him his second-last miner and his spy. He does manage to drive the major on a bomb. This is no problem for invincible because he has enough pieces.
- Move 449: Invincible takes a guess with his lieutenant and finds a bomb.
- Move 455: The marshals are exchanged. This means an easy victory for Invincible who can now easily trap the remaining pieces.
- Move 455-495: Invincible does this fairly efficient without giving Raymond a chance to get one of his last pieces near the flag. Invincible wins.

Summary Raymond vs. Invincible (0-1):

After a few changes to the program, Invincible now played a lot better. He played much more active, punishing most mistakes of his opponent. He did several times make the mistake were that he doesn't recognize the situation where he can move a high piece between two lower pieces and capture one of them. It should be easy to teach him to recognize this. He also played too passively with his known marshal, leaving it in a useless position. This may be harder to improve because it's not easy to define what exactly he should have done instead.

Game 3: Mehdi vs. Invincible (0-1) – Invincible wins in 316 moves

- Move 1-12: Both players scout a few front-row pieces.
- Move 13: Invincible takes a known captain with a colonel.
- Move 17: Invincible "accidentally" takes Mehdi's spy with his scout. This is a lucky start for Invincible.
- Move 18-27: Invincible attacks with a lieutenant, taking a scout and finding a major.
- Move 28-37: Invincible attacks with a captain, taking a scout and finding a bomb. Mehdi moves forward with his known major.
- Move 38-63: Mehdi moves a lot of pieces forward, putting pressure on Invincible. Invincible awaits the attack.
- Move 68-75: Invincible attacks with a captain. He is revealed by a scout and retreats because a known colonel is defending on that side.
- Move 75-84: Mehdi continues moving forward with several pieces, finding a colonel with his lieutenant.
- Move 85-113: The known colonel wreaks havoc among Mehdi's attacking pieces. He takes the known major, a lieutenant and a captain. He retreats when the (unknown) marshal comes his way.
- Move 114-122: The marshal pushes the colonel back to his own side. Invincible then leaves it next to an unknown piece bluffing that this is his spy. Mehdi is unafraid and takes the colonel with his marshal. This makes the game approximately even, with Mehdi leading by a colonel and Invincible having an extra major and captain. Invincible has more information though, having taken the spy and now knowing the marshal. Also Mehdi moved more pieces.
- Move 136: Mehdi takes a scout with his general. Invincible now knows both the marshal and the general.
- Move 140: Mehdi takes a risk with his colonel by taking an unmoved piece deep in Invincible's setup. It turns out to be a sergeant, and neither the general nor the marshal seems to be nearby to retaliate.
- Move 140-147: Invincible attacks with a known captain, taking unmoved pieces. He gets a lieutenant, miner and then hits another captain. A very good result for such attack.
- Move 140-164: Mehdi attacks with a miner and his marshal. The miner gets captured by a major.
- Move 170-178: Mehdi takes more risks with his colonel, taking a lieutenant, captain and finally running into a bomb.
- Move 184: Mehdi's attacking marshal takes a known major that got trapped on the back row. In pieces it is almost even now, with Invincible having three more miners and Mehdi three more sergeants. Still, it looks good for Invincible because he has far more information. His marshal and general are still both unknown.
- Move 189: Mehdi's attacking general gets captured by Invincible's marshal.
- Move 190-196: With the general captured and the marshal out of the way, Invincible uses his known colonel to attack. He goes for a trapped (unknown)

major. Mehdi can only avoid its capture by exchanging his unknown colonel for Mehdi's known one.

- Move 197-209: Invincible now uses his marshal for the same purpose and captures the major.
- Move 214: Mehdi attacks a bomb with his marshal
- Move 215: Invincible attacks a bomb with his marshal (!) This is a strange move and even stranger timing, but Invincible can afford to lose his marshal in this position and the bomb was not a very likely one. Invincible still has a comfortable lead. His general is the highest piece on the board and it's still unknown. He also has one more major, which is the second highest rank on the board and Mehdi has only 1 miner left. This is a situation Invincible should win without having to be very creative or taking risks.
- Move 220-234: Both Invincible and Mehdi use a major to attack in the centre. They walk past each other, with Invincible taking a sergeant.
- Move 235: Mehdi risks too much with his major and gets captured by the general. Mehdi's highest piece is now a captain, while Invincible has a general and two majors left.
- Move 236-266: Invincible attacks with general and major, taking all pieces that Mehdi moved.
- Move 270-278: Mehdi attacks with a miner, takes a front row bomb and gets exchanged by a miner of Invincible. This seems a bit too good for Mehdi, given the advantage of Invincible, but it wasn't dangerous either. The bomb wasn't important and the exchange of Invincible's first miner against Mehdi's last miner was good too.
- Move 278-314: Mehdi get's a last chance to attack with his captain but hits a bomb. He could not have reached the flag. It turned out Invincible had one major trapped by bombs, so he was ahead a little less than it seemed. Still he took no unreasonable risks and secured the win fairly efficient.

Summary Mehdi vs. Invincible (0-1):

Mehdi played risky and unexpected, having his marshal on the last row in the initial setup. This is something Invincible won't expect, but he didn't fall for it either when this piece approached his colonel. He safely retreated instead. Mehdi initially got some advantage when he started taking unmoved pieces but it eventually cost him his high pieces after which Invincible secured his win without giving Mehdi a chance to take his flag. In this game Invincible could probably have won quicker Mehdi moved a lot of pieces which he couldn't defend given that his spy was killed early and his marshal and general were known. Invincible didn't attack because he preferred to keep his own marshal and general unknown a bit longer. Eventually this paid off because he took two important pieces with them, which is probably more than he would have gotten if he attacked earlier.

Game 4: Lorena vs. Invincible (0-1) – Invincible wins in 328 moves

- Move 1-40: Lorena get's the better deal in the opening confrontations. After 40 moves she is leading with three low pieces.
- Move 54: Lorena attacks with a captain and finds a bomb
- Move 57: Lorena moved her marshal forward, which immediately is attacked by a scout.
- Move 58-82: Lorena continues attacking with her marshal in search of the flag. She captures a few low pieces but then hits a bomb.
- Move 84-89: Lorena attacks with her general. Because the general is unknown she actually chases Invincible's marshal away. Invincible thinks the general is a low piece and doesn't want to reveal the rank of his marshal by attacking. It still turns bad for Lorena when she takes a lieutenant next to Invincible's marshal. Invincible, who now finds out the rank of the attacking piece, captures it with his marshal. Invincible is now leading by a marshal and general. Even though this is certainly not a safe win against an aggressive player, it's something Invincible should be able to convert in a win.
- Move 90-100: Lorena attacks with a captain and attacks a colonel. Invincible could have attacked the captain the move before, but he doesn't. He knows that Lorena has no pieces higher than a colonel, but she still has two unknown colonels and while leading a general and marshal it is better to try and capture these than exchange them with his own colonel.
- Move 101-111: Invincible attacks with an unknown piece. It is a scout, deliberately looking for Lorena's spy, it never moved, but was in a place where people often put their spy. Invincible is right and captures the spy.
- Move 120-128: Lorena attacks with a colonel but hits a bomb
- Move 130-147: Invincible attacks with his known colonel and clear up a few pieces that had previously been moved. He also traps one in the center, using his marshal and colonel. Lorena attacked with another captain and again hit a colonel. Invincible again didn't want to attack the captain himself with the colonel but rather waited for Lorena to attack something.
- Move 150-180: Lorena attacks with her last colonel. She is attacked by the spy, which was now useless so Invincible used it for getting information. She goes on attacking and eventually hits a bomb. Invincible continues using his marshal and colonel to trap pieces that moved.
- Move 181-328: Invincible now has a huge lead and can't lose. He spends the rest of the game trapping and capturing pieces with his marshal and colonel, while keeping the rest of his high pieces in defense to wait for attacking pieces. Lorena does manage to capture a few low pieces, but can never create a dangerous situation. Invincible had too much to do and sometimes moves back and forth, it might have been possible to finish this game faster, but the way Invincible did it was safe and eventually effective.
- Move 329: Lorena's last piece is a scout, which can move fast, but Invincible used his last scout to unexpectedly attack it from a distance and win the game.

Summary Lorena vs. Invincible (0-1):

Lorena played very aggressively, but Invincible had his defending pieces in the right places. The game was therefore decided relatively quickly, but it was interesting to see the techniques Invincible used to finish a won game. He kept enough pieces in defense to prevent Lorena from getting to the flag in a last attempt, leaving her only the side with bombs to attack on. He used two high pieces to trap the pieces she moved so that she would have to move other pieces, eventually causing her to lose all her pieces. He never attempted to get the flag, but in this situation that wasn't necessary. It is often easier just to capture all moving pieces than to try and find the flag. Invincible knows a few more tricks to force a won game, but capturing moved pieces with two high ranks was in this situation the easiest and most effective.

Game 5: Sjoerd vs. Invincible (1-0) – Sjoerd wins in 442 moves

- Move 1-13: Sjoerd blocked two of the passages with bombs so he could only play from the remaining side where he had his marshal and general in the front. He made a wild attack with his marshal and lost it to the spy.
- Move 14-34: Sjoerd next attacks with his general and takes a lot of unmoved pieces, eventually hitting the unknown and unmoved general of Invincible. Invincible now has both Sjoerd's marshal and general, but only those two pieces while he lost his own general and 9 more pieces. This would be in Sjoerd's advantage, but now his bombs could turn against him if Invincible would realize they are there. Invincible scouted only two of them and makes no attempt to find out anything about the other two front-row pieces.
- Move 48: Sjoerd loses a captain to a front row bomb. Both his marshal and general just missed this bomb, so a captain is a small price for finding it.
- Move 64-65: Sjoerd takes a major with his colonel and finds the marshal.
- Move 66-83: Invincible moves his marshal to block Sjoerd's only entrance. This should be a won position for Invincible, in spite of the fact that he is down so many small pieces because two his pieces are enough to block Sjoerd's only way out and attack. The only problem is that Sjoerd still has a spy.
- Move 99: A miner got through and gets captured by a colonel when he gets too close to the flag.
- Move 100-123: Invincible attacks with his known colonel and chases away the spy. Sjoerd is forced to exchange it for his last (unknown) colonel.
- Move 124-150: Sjoerd moves three pieces including the spy and a major forward. Invincible blocks him with his marshal but is unable to attack because of the unknown spy. If he would get his last unknown colonel he could kill all these pieces and win the game, but he keeps it behind.
- Move 150-320: In the next 170 moves both Sjoerd and Invincible move back and forth. This is understandable from Sjoerd's side as he has a problem getting out, but Invincible should have taken action to secure his win.

- Move 320-370: Sjoerd gets through with his major and hits a few pieces and finds a bomb. This again cost Invincible low pieces, but by now he has enough high pieces to win.
- Move 370-420: Sjoerd again moves back and forth and so does Invincible, when all he has to do is reveal his unknown colonel.
- Move 430-440: Sjoerd got a miner through and is allowed to get to the known bomb, even though Invincible has a couple of high pieces just one move away. He seems to take it for granted that the approaching piece is not a miner and gives him a free path to the flag. This is a huge mistake.
- Move 442: Sjoerd captures the flag.

Summary Sjoerd vs. Invincible (1-0)

This game nicely demonstrated a few weaknesses Invincible still has. He is presented an unusual, but nevertheless for human players easy, win and is unable to convert it. He then blunders and lets Sjoerd capture the flag. In response to the blunder I lowered the minimal chance for a piece to be a miner before Invincible bothers to defend. The other mistake was that Invincible was unwilling to reveal his colonel even though he could have put so much pressure with it that he would almost certainly win. This is logical from his point of view because his formulas tell him that he shouldn't reveal it just for the first piece he could capture with it. They don't take this specific situation into account. Invincible also never gets tired of moving back and forth when he has no profitable plan. This works more often against than in favor of him so maybe this should be solved by giving him an artificial impatience, letting him do less favorable attacks when nothing is happening while he should be winning.

Appendix D – Test setups

In this appendix you can find my own manually created setups that I used to test the heuristic evaluation function of my setup creator. These setups are not used by "Invincible" but are used as a measure of good setups. Some of these setups I have used countless of times on real life tournaments, others are newly created.

Setup 1 – score: 18

This is one of my most "famous" setups, which is a dubious quality for a Stratego setup. I've not used any other setup as often on tournaments and several other players have successfully used this setup as well. I lost track of who already knows it so I can't use it any more. It was once used against me in a practice game by someone who didn't know it was my setup. It is a bit predictable, but all pieces are at hand when you need them and it works well for both attack and defense.

Setup 2 – score: 18

This is another setup I have apparently used too often on tournaments. I used it far less than the previous because it's outspoken aggressive and not very suitable for careful slow games, but it's less predictable and works better against stronger players. It has a funny story attached to it because I used this setup on the last tournament against a player who somehow had this exact setup written down and memorized (I don't even write my setups down myself) and used it twice in that same tournament himself. On that one occasion I decided to swap the miner in the lower right corner with the bomb next to it. When he hit that bomb with a high piece he explained why he did that and to my amazement could tell the rank of every one of my 25 remaining pieces.

Setup 3 – score: 20

Another setup I used often. Harder It's harder to defend the flag in the corner, but otherwise works very well.

Setup 4 – score: 22

This is a setup I didn't use often, but which I still considered a good all-round setup.

Setup 5 – score: 19

The last one is a new invention. The flag on this position is something I often use against stronger opponents, but the rest of the pieces I usually change every time in such games.

Setup 6 – score: 16

This is an old setup that I invented on my first world championships and which turned my losing streak then. It's rather weak defensively though, it worked well when setups of this style were still unexpected but later it got too weak.