2DI66 - Advanced Simulation

Assignment 1: Tic-tac-toe

17th February 2020

Group 10

M. Berkhout 0953813
N.J.M. Segers 0958992
V.F.E. Zijlstra 0940456
A.M. Raju 1306162

Table of Contents

1 Introduction

In this report, we present our findings upon the usage of a number of strategies used in the game TicTacToe, based on randomness. To achieve this goal, we create a simulation environment for the game TicTacToe that enables us to apply a number of strategies for each player, whilst also allowing us to predict the probabilities of winning based on an initial setup.

To compute an estimation of the probabilities of success of the various strategies used in the report, stochastic simulation is used. Stochastic simulation calculates the probabilities of the end-results by performing a large number of simulations, and it is particularly well-suited to games like TicTacToe where there are limited ending categories (P1 winning, P2 winning or a draw).

The rest of the report is described as follows. Section 2 describes the simulation environment and describes how the simulation is run. It also presents pseudocode about the randomness-based strategies used in the simulation environment. Section 3 presents the findings based on these approaches. Multiple subsections are devoted towards special cases - a completely random strategy used from the start, best first moves to make when using a completely random strategy, and also the results of two other strategies based on a recursive greedy choice based randomness strategy. In addition, three specific games are considered, and the probabilities of winning for each valid move are presented. Finally, Chapter 4 concludes our report and suggests some recommendations towards some other strategies that might provide better results.

2 Simulation Description

The simulation description consists of two parts. In the first part the simulation set-up is discussed. In this part all decision variables are discussed. Afterwards, the algorithm for creating the possible solution outcomes and the main idea behind the simulation is discussed. In the second part of this chapter, the simulation algorithms are discussed.

2.1 Simulation set-up

Before the simulation can run some variables have to be determined:

- amountOfRuns: The amount of runs to be used by the simulation. This value should be high enough to ensure that the outcomes are correct.
- n: The size of the board, given by the amount of vertical/horizontal rows of the board. The value has to be between 3 (for a 3x3 board) and 5 (for a 5x5 board).
- strategy1, strategy2: The strategy of both players 1 and 2. The strategy can be either Random, SimWin or SimNotLose.
- startPlace: The desired start place for player 1 (beginning player), in which the first player must start.
- p1Taken, p2Taken: The places that are already taken by a player.

Based on these values, all possible win options are generated with the function described in Algorithm 1. It is important to note that the positions on the board are tracked by integer values: The position on the top-left of the board is seen as 0, the top-middle position as 1, top-right as 3 and so on until the bottom-right position with value 8. An illustration is given in XXX.
ADD PICTURE

Algorithm 1 Generating all possible win options of a board

```
1:  function GENERATEWINOPTIONS
2:      possibleOptions ← list(0, 1, .., n^2)
3:      winOptions ← list of all options to win
4:      diagonalOne ← first diagonal of the board
5:      diagonalTwo ← second diagonal of the board
6:      for i in range(0, n) do
7:          winOptions add list of every value starting from(i * n) until value ((i + 1) * n) from possibleOptions
8:          winOptions add list of every n-th value starting by value i from possibleOptions
9:          diagonalOne add value i*n+1
10:         diagonalTwo add value (i+1)*n-(i+1)
11:     winOptions add diagonalOne
12:     winOptions add diagonalTwo
13: return winOptions
```

2.2 Running the simulation

After initializing all the variables, the simulation can be run. An overview of the steps taken to run the final simulation for a random player game is listed in Algorithm 2. All the input variables of this function are listed in the Simulation set-up section. The games are played in rounds of the algorithm, in which the playGame function is called. This function is listed in Algorithm 3.
In the playGame function (Algorithm 3), a list of all free spots is firstly generated. After which a first predetermined spot will be chosen (if given). Next, the game will be played with random

picks until there are no more spots to take or until there is a winner. The result of the game is sent back to the simulation Algorithm.

2.2.1 Integrating Simwin and SimNotLose

As an extension, the SimWin and SimLose algorithms are integrated in to the simulation. First off, the decision of choosing the best possible position based on the game strategy is given in Algorithm 4. This extension still uses the playGame function listed in Algorithm 3 to determine the chances of having a win, lose or draft per possible move (given that the following moves will be random). Next to this, an extra game should also be set-up to play the "end-game" in which all moves are based on the strategy of the player. This is listed in Algorithm 5. It is notable that this Algorithm looks similar to the standard playGame Algorithm. The biggest difference is that the player can now also pick a strategy and that the amountOfRuns variable is required in order to determine the amount of runs for every pick. This game returns the player that will win. A simple adjustment of the Algorithm 2 in which the starting variables will be extended and the playGame function is replaced by the PlayStrategyGame function makes it possible to also simulate the PlayStrategyGame.
The simulation Algorithm will then use this result to track the outcomes. After playing the predetermined amount of rounds, the results are computed.

2.2.2 Final code remark

In the section above the simulation of gaining the chances of winning/losing based on one strategy was discussed. In order to obtain a matrix of all possible results, this simulation has to be run multiple times. In the simulation file code it is also possible to run the complete results section by running the functions called "Question 1" till "Question 4".

Algorithm 2 Simulation environment for random strategy

```
1:  function SIMULATEGAMES(n, amountOfRuns, winOptions, startPlace, p1Taken, p2Taken)
2:      p1Wins ← 0
3:      p2Wins ← 0
4:      drafts ← 0
5:      for i in (0,1,..,amountOfRuns) do
6:          Game ← playGame(n, startPlace, p1Taken, p2Taken, "p1")
7:          if game == p1 then
8:              p1Wins ← p1Wins + 1
9:          else if game == p2 then
10:             p2Wins ← p2Wins + 1
11:         else
12:             drafts ← drafts + 1
13:     p1Probability ← p1Wins \ amountOfRuns
14:     p2Probability ← p2Wins \ amountOfRuns
15:     draftProbability ← drafts \ amountOfRuns
16: return p1Probability, p2Probability, draftProbability
```

Algorithm 3 Play a game of tic tac toe

```
1: function playGame(n, startPlace = None, p1Taken = empty(), p2Taken = empty(), turn = "p1")
2:     possibleOptions ← list(0, 1, .., n^2) - p1Taken - p2Taken
3:     if startPlace != None then
4:         possibleOptions remove startPlace
5:         if turn == "p1" then
6:             p1Taken add startPlace
7:             turn ← p2
8:             for winOption in winOptions do
9:                 if winOption values are in p1Taken then return p1
10:        else
11:            p2Taken add startPlace
12:            turn ← p1
13:            for winOption in winOptions do
14:                if winOption values are in p2Taken then return p2
15:    maxPicks ← length of possibleOptions
16:    for i in range(0, 1, , maxPicks) do
17:        if turn == p1 then
18:            playerPick ← a random pick
19:            p1Taken ← p1Taken + playerPick
20:            possibleOptions ← possibleOptions - playerPick
21:            turn ← p1
22:        else
23:            playerPick ← a random pick
24:            p2Taken ← p2Taken + playerPick
25:            possibleOptions ← possibleOptions - playerPick
26:            turn ← p1
27:        for winOption in winOptions do
28:            if winOption values are in p1Taken then return p1
29:            if winOption values are in p2Taken then return p2
    return draft
```

Algorithm 4 Move prediction for Sim strategies

```
1: function simStrategy(strategy, possibleOptions, turn, taken1, taken2, amountOfRuns)
2:     for move in possibleOptions do
3:         p1Wins ← 0
4:         p2Wins ← 0
5:         drafts ← 0
6:         for i in (0,1,..,amountOfRuns) do
7:             Game ← playGame(n, startPlace, p1Taken, p2Taken, "p1")
8:             if game == p1 then
9:                 p1Wins ← p1Wins + 1
10:            else if game == p2 then
11:                p2Wins ← p2Wins + 1
12:            else
13:                drafts ← drafts + 1
14:        if turn == p1 then
15:            if strategy == "SimWin" then
16:                possibleMoves[move] ← p1Wins/amountOfRuns
17:            if strategy == "SimNotLose" then
18:                possibleMoves[move] ← (p1Wins+drafts) /amountOfRuns
19:        if turn == p2 then
20:            if strategy == "SimWin" then
21:                possibleMoves[move] ← p2Wins/amountOfRuns
22:            if strategy == "SimNotLose" then
23:                possibleMoves[move] ← (p2Wins+drafts) /amountOfRuns
    return Key of highest value in possibleMoves
```

Algorithm 5 Adjust play of a game of tic tac toe, based on player strategy

```
1: function playStrategyGame(amountOfRuns, winOptions, strategy1, strategy2, turn =
   "p1", p1Taken = empty(), p2Taken = empty())
2:     n ← 3
3:     possibleOptions ← list(0, 1, .., n^2) - p1Taken - p2Taken
4:     for winOption in winOptions do
5:         if winOption values are in p1Taken then return p1
6:         if winOption values are in p2Taken then return p2
7:     maxPicks ← length of possibleOptions
8:     for i in range(0, 1, .., maxPicks) do
9:         if turn == p1 then
10:            if strategy1 == Random then
11:                playerPick ← a random pick
12:            else
13:                playerPick ← simStrategy(strategy1, possibleOptions, "p1", p1Taken, p2Taken, amountOfRuns)
14:            possibleOptions ← possibleOptions - playerPick
15:            p1Taken ← p1Taken + playerPick
16:            turn ← p2
17:        else
18:            if strategy2 == Random then
19:                playerPick ← a random pick
20:            else
21:                playerPick ← simStrategy(strategy2, possibleOptions, "p2", p1Taken, p2Taken, amountOfRuns)
22:            possibleOptions ← possibleOptions - playerPick
23:            p2Taken ← p2Taken + playerPick
24:        for winOption in winOptions do
25:            if winOption values are in p1Taken then return p1
26:            if winOption values are in p2Taken then return p2
       return draft
```

3 Presentation of Results

3.1 Random strategy

The tic-tac-toe game was simulated to determine the probabilities that a game would end in a win for player 1, a win for player 2 or a draw, when all possible moves are considered and a box is picked uniformly at random. To obtain values for these probabilities, algorithm 1 was used: Play a game of tic-tac-toe. The game was simulated for n=3, n=4 and n=5.

3.1.1 Random strategy when n=3

When n=3, the game is played on a playing board with nine possible first picks for player 1 and eight possible winning options. To achieve accurate results, the game is simulated 1000 times. This yields the probabilities as presented in Table 3.1.

As can be seen in Table 3.1 the game ended in a draw with probability 0.450. This means that in nearly half of the games played, there was no winner. Considering the random strategy this is reasonable. Comparing the probability of winning of player 1 and player 2, player 1 had the highest probability of winning, namely 0.361 (win player 1) compared to 0.189 (win player 2). Important to note is that player 1 always starts the game, meaning player 1 can, in case there is no winner after eight boxes have been filled, pick the last (extra) box, after which the game can still end in a win for player 1. Therefore, it is reasonable that player 1 has a higher probability of winning than player 2.

Table 3.1: Winning probabilities when n=3

Probability win player 1	**Probability win player 2**	**Draw**
0.361	0.189	0.450

3.1.2 Random strategy when n=4

Increasing the value of n to n=4, ensures sixteen possible first marks for player 1. For n=4 there are ten possible winning options. Again, the game is simulated 1000 times. The probabilities of a game ending in a win for player 1, a win for player 2 and draw are presented in Table 3.2.

Table 3.2 shows that the probability of a game ending in a draw increased to 0.723 when playing the tic-tac-toe game with n=4. This is reasonable as the number of boxes in line to win becomes four instead of two. Besides the number of boxes considered increased to sixteen (compared to nine when n=3), while the number of winning options did not increase proportionally (ten for n=4, eight for n=3). However, in contrast to n=3 the difference between the probabilities of player 1 winning and player 2 winning is not as large as for n=3 (0.153 for player 1 and 0.124 for player 2), since the number of turns is equal in some of the simulated games (i.e. no winner after the 15th mark by player 1, ensures player 2 can still win the game by playing the last mark). The probability of player 1 winning the game is slightly larger than the probability of player 2 winning the game, as player 1 starts each game.

Table 3.2: Winning probabilities when n=4

Probability win player 1	**Probability win player 2**	**Draw**
0.153	0.124	0.723

3.1.3 Random strategy when n=5

In case n=5, the number of first picks increases to 25 and the number of winning options increases to twelve. The game is simulated 1000 times to obtain more accurate results. The winning probabilities are presented in Table 3.3.

From Table 3.3 one can see that, again, the probability of a draw increased, compared to n=3 and n=4. The same reasoning applies for n=5 as explained for n=4: the number of possible picks increases to 25, which happens disproportionally to the increase in the number of winning options, which increases to twelve. Adding to that, the number of boxes in line required to win the game increases to five. Consequently, the probabilities of one of both players winning the game decreases. Still, player 1 has a greater probability of winning than player 2 ($0.081 > 0.049$), because player 1 starts the game and, since there are 25 boxes, in some cases (if no winner after the 24th turn) player 1 ends the game with an "extra" turn which can still result in a win for player 1.

Table 3.3: Winning probabilities when n=5

Probability win player 1	**Probability win player 2**	**Draw**
0.081	0.049	0.870

3.2 Random strategy with fixed first mark

Section 3.1 showed the simulation results of a game ending in a win for player 1, player 2 or a draw. This simulation model was extended to test whether the placement of the first mark of the game had an influence on the probability distribution. Symmetry is exploited when retrieving these results: the number of different starting points was determined for n=3, n=4 and n=5. The first mark in the game is set, after which the game was open and played uniformly at random. For each fixed starting point, the probabilities were determined of a game ending in a win for player 1, a win for player 2 or a draw.

3.2.1 Random strategy with fixed first mark when n=3

For n=3, the number first marks to be tested equals three due to symmetry. Table 3.4 shows the probabilities of a game ending in a win for player 1, a win for player 2 or draw based on 1000 simulation runs.

From Table 3.4 it can be concluded that placing the first mark in the center box of the playing grid provided player 1 with the highest probability of winning, compared to starting in any other box (probability 0.570 compared to 0.497 and 0.406). This can be explained as the center box appears in four of the eight winning options, whilst starting point 0 appears in three of the winning options and starting point 1 in only two of the winning options. Player 2 has the largest probability of winning when player 1 places the first mark in the most unfavorable box (i.e. with smallest winning probability of player 1).

Table 3.4: Winning probabilities when n=3 and fixed first mark

First mark	**Probability win player 1**	**Probability win player 2**	**Draw**
0	0.497	0.156	0.347
1	0.406	0.159	0.435
4	0.570	0.126	0.304

3.2.2 Random strategy with fixed first mark when n=4

When n=4, the number first marks to be tested also equals three due to symmetry. Table 3.5 shows the winning probabilities based on 1000 simulation runs.

The table shows that player 1 has the highest probability of winning when placing the first mark in one of the corners (probability = 0.189). However, there is still a high probability that the game ends in a draw.

Table 3.5: Winning probabilities when n=4 and fixed first mark

First mark	Probability win player 1	Probability win player 2	Draw
0	0.189	0.112	0.699
1	0.150	0.127	0.723
5	0.169	0.094	0.737

3.2.3 Random strategy with fixed first mark when n=5

In case n=5, there are six different starting points that need to be tested due to symmetry. Again 1000 simulation runs are used. The results are presented in Table 3.6.

While Table 3.6 shows that for most of the starting points the probabilities of winning for player 1 are somewhat equal, except for starting in the center box, which shows a slightly higher probability of winning for player 1. However, as the number of boxes in one line required to win now equals five, the probability of a game ending in draw is very large.

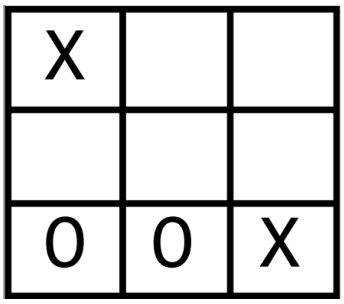
Table 3.6: Winning probabilities when n=5 and fixed first mark

First mark	Probability of win for player 1	Probability of win for player 2	Draw
0	0.095	0.054	0.851
5	0.096	0.059	0.845
6	0.092	0.044	0.864
10	0.091	0.045	0.864
11	0.094	0.055	0.851
12	0.131	0.032	0.837

3.3 Random Strategy from given game state

Given the 3 different scenarios of games, we identify the player whose move it is, and estimate the probabilities of winning or drawing if the random strategy were to be used.

3.3.1 Random strategy in Game 1 with n = 3

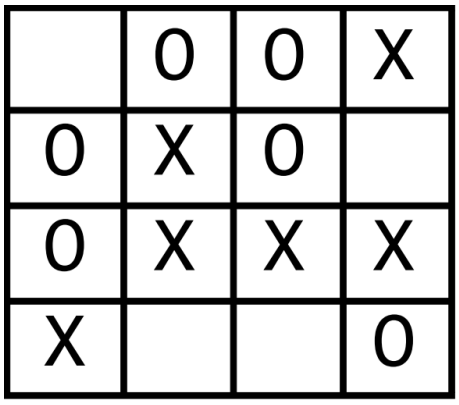


From the number of moves already done, we can see that it is Player 1's turn. Using this as the starting point of moves done so far, we get the following results when simulating a random strategy with a 100[illegible]ials.

Table 3.7: Winning probabilities from given positions in Game 1

Move	Probability of win for player 1	Probability of win for player 2	Draw
(0,1)	0.824	0.176	0.0
(0,2)	0.929	0.071	0.0
(1,0)	0.664	0.336	0.0
(1,1)	1.0	0.0	0.0
(1,2)	0.752	0.248	0.0

3.3.2 Random strategy in Game 2 with n = 4

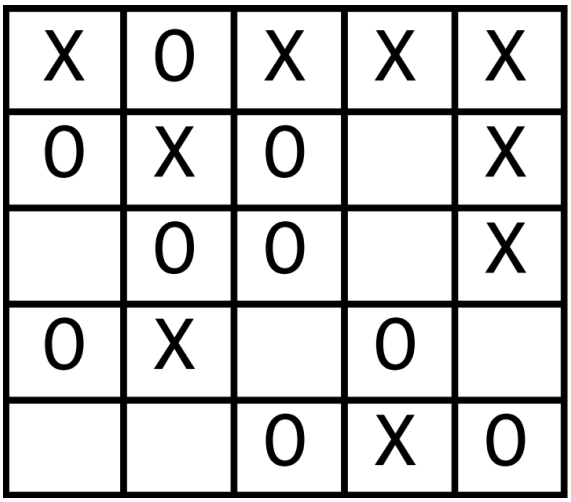


From the number of moves already done (even), we can see that it is Player 1's turn in this game too. Using this as the starting point of moves done so far, we simulate a random strategy with a 1000 trials. It is obvious that there are no possible moves that would lead to a winning situation for any player and this is represented in the table below.

Table 3.8: Winning probabilities from given positions in Game 2

Move	Probability of win for player 1	Probability of win for player 2	Draw
(0,0)	0.0	0.0	1.0
(1,3)	0.0	0.0	1.0
(3,1)	0.0	0.0	1.0
(3,2)	0.0	0.0	1.0

3.3.3 Random strategy in Game 3 with n = 5



From the number of moves already done (even), we can see that it is Player 1's turn in this game too. Using this as the starting point of moves done so far, we simulate a random strategy with a 1000 trials. It is obvious that there are no possible moves that would lead to a winning situation for any player and this is represented in the table below.

Table 3.9: Winning probabilities from given positions in Game 2

Move	Probability of win for player 1	Probability of win for player 2	Draw
(1,3)	0.0	0.0	1.0
(2,0)	0.0	0.0	1.0
(2,3)	0.0	0.0	1.0
(3,2)	0.0	0.0	1.0
(3,4)	0.0	0.0	1.0
(4,0)	0.0	0.0	1.0
(4,1)	0.0	0.0	1.0

3.4 Results 4

The outcome of a TicTacToe game can be predicted when the strategy of the players is known. In this situation the strategies consist of a random strategy, the SimWin strategy and the SimNotLose strategy. The SimWin and SimNotLose strategies choose the best move by choosing the

move that results in the highest probability of winning or not losing respectively. These probabilities are determined by simulating the remaining moves of the game with random placement for both players. Simulating 1000 games gives a good indication of the best move that gives the highest probability of the desired outcome.

The probability of actually winning, losing or ending in draft can be obtained by applying the same strategies 100 times. Doing so results in a 3x3 matrix with in each cell the probabilities of a win for player1, a win for player 2 or a draft respectively.

P(p1,p2,draft)	**Random**	**SimWin**	**SimNotLose**
Random	(0.63,0.27,0.10)	(0.06,0.90,0.04)	(0.0,0.8,0.2)
SimWin	(0.99,0.0,0.01)	(1.0,0.0,0.0)	(0.0,0.01,0.99)
SimNotLose	(0.99,0.0,0.01)	(1.0,0.0,0.0)	(0.0,0.0,1.0)

Table 3.10: Probability table, Strategy1 vertically, Strategy2 horizontally

References:
Lecture notes???

4 Conclusion

All in all, the simulations have provided some interesting conclusions using the different strategies. At first, player 1 always has a larger probability of winning than player 2, as player 1 always starts the game. However, important to note is that the difference in these winning probabilities is larger when n is an odd number (i.e. n=3 and n=4). This occurs as player 1 has an extra turn compared to player 2, when there is no winner in the second to last round, providing player 1 with an extra opportunity to win. Adding to that, the probability of either of the players winning decreases as n increases. The number of marks to be placed increases when n increases, which happens disproportionate to the number of combinations that results in a win. As such, for a larger value of n the probability of the probability of a of a game ending in draw is larger.

4.1 Recommendations& Future research

- describe Best strategy being player 1
- describe Best strategy being player 2
- future: what happens if Simwin or simnotlose is simulated by applying simwin or simnotlose in finding the best move - confidence intervals or exact calculation of probabilities

5 Distribution of work

The project started with a group of three team members: Mick, Veronique and Niek. The first part of the simulation, which consisted of everything except the extra strategies, was made together with those three team members. This was done to ensure consistency. After making this part of the simulation Abhishek joined the group, however as the simulation was already almost-done it was decided that it would be better if someone of the initial group would finish it. In the end, Niek has finished the new algorithms. While writing the report, the division of tasks was as follows:

- Introduction: Abhishek
- Simulation Description: Mick
- Presentation of results:
  - Question 1: Veronique
  - Question 2: Veronique
  - Question 3: Abhishek
  - Question 4: Niek
- Conclusion: Open
- Division of Work: Mick

After writing the report, the report was checked by all team members.
Although the division of work was not split completely equally (due to the fact that Abhishek joined the group later on), we assume that this can be corrected in the next assignment.