jamk.fi

Developing and Optimizing Artificial Intelligence in Zero-sum Games

Matti Järvensivu

Bachelor’s thesis
May 2018
Technology, communication and transport
Information and Communication Technology
Degree Programme in Software Engineering

jamk.fi

Description

Author(s) Järvensivu, Matti	Type of publication Bachelor's thesis	Date May 2018
		Language of publication: English
	Number of pages 31	Permission for web publication: x
Title of publication **Developing and Optimizing Artificial Intelligence in Zero-sum Games**		
Degree Programme Software Engineering		
Supervisor(s) Salmikangas, Esa Huotari, Jouni		
Assigned by		

Abstract

Artificial Intelligence makes actions to complete a specific goal and it is important that it makes an optimal choice. Game theory provides a mathematical framework that helps to build an optimal Artificial Intelligence. Minimax algorithm is the solution to zero-sum games, and it will iterate through every possible position in a game making the optimal choice.

The goal was to research common concepts of game theory and develop an unbeatable Artificial Intelligence for a game of tic-tac-toe with the tools provided by game theory. The developed Artificial Intelligence was to be optimized with alpha-beta pruning to decrease the amount of computations. The developed Artificial Intelligence was tested by putting the optimal Artificial Intelligence play against two different simpler Artificial Intelligences. The game was developed with Unity3D game engine and C# programming language.

The goal to make unbeatable Artificial Intelligence for tic-tac-toe was successful. The game theoretical Artificial Intelligence won 17 games against a simple Artificial Intelligence and 13 ended in a draw out of 30 games. The optimization with alpha-beta pruning improved the performance significantly, reducing the computations from 549945 to 16771 on Artificial Intellligence's first move. After the improvements to the alpha-beta pruning, the amount of computations decreased to 7273.

Game theory provides a great framework for development of different Artificial Intelligences. Minimax is an easy solution to zero-sum games and it is easy to implement, It can be optimized with few easy methods. The solution is great because it can be implemented into any other zero-sum game.

Keywords/tags
Game theory, Zero-sum game, Artificial intelligence, Unity3D, Minimax, Alpha-beta pruning

Miscellaneous

jamk.fi

Kuvailulehti

Tekijä(t) Järvensivu, Matti	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä Toukokuu 2018
	Sivumäärä 31	Julkaisun kieli Englanti
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Tekoälyn kehitys ja optimointi nollasummapeleissä

Tutkinto-ohjelma
Insinööri (AMK), ohjelmistotekniikan tutkinto-ohjelma

Työn ohjaaja(t)
Esa Salmikangas
Jouni Huotari

Toimeksiantaja(t)

Tiivistelmä

Tekoäly tekee toimintoja saavuttakseen tietyn tavoitteen, ja on tärkeää, että se tekee optimaalisen valinnan. Peliteoria tarjoaa matemaattisen viitekehyksen, joka auttaa kehittämään optimaalisen tekoälyn. Minimax-algoritmi on ratkaisu nollasummapeliin, ja se iteroi läpi jokaisen pelitilanteen pelissä tehden optimaalisen valinnan.

Tavoiteena oli tutkia yleisimpiä peliteorian konsepteja ja kehittää voittamaton tekoäly ristinolla-peliin peliteorian antamilla työkaluilla. Kehitettyä tekoälyä optimoitiin alpha-beta karsintamenetelmällä, jotta saadaan laskettua komputaatioiden määrää. Kehitettyä tekoälyä testattiin laittamalla se pelaamaan kahta yksinkertaisempaa tekoälyä vastaan. Peli kehitetettiin Unity3D-pelimoottorilla ja ohjelmointikielenä käytettiin C# ohjelmointikieltä.

Tavoite luoda voittamaton tekoäly ristinollaan onnistui. Peliteoreettinen tekoäly voitti 17 peliä ja pelasi 13 peliä tasan 30 pelistä. Optimointi alpha-beta-karsintamenelmällä nosti tekoälyn suorituskykyä huomattavasti laskien komputaatioden määrää 549945:stä 16771:een ensimmäisellä vuorolla, ja kun alpha-beta-karsintaa paranneltiin komputaatioden määrä laski 7273:een.

Peliteoria tarjoaa loistavan viitekehyksen tekoälyjen kehitykseen. Minimax on ratkaisu nollasummapeliin, ja sitä on helppo implementoida ja sitä voi optimoida muutamalla helpolla tavalla. Ratkaisu on hyvä, sillä sitä voi hyödyntää myös muissa nollasummapeleissä.

Avainsanat (asiasanat)

Peliteoria, Nollasummapeli, Tekoäly, Unity3D, Minimax, Alpha-beta pruning

Muut tiedot

Contents

Terminology **3**

1. Introduction **4**

1.1 Introduction to project 4

1.2 Goals, motivation and progression of thesis 5

2. Game theory **6**

2.1 Concept of game theory 6

2.2 Presenting games 7

2.3 Strategies 8

2.4 Different forms of games 10

2.5 Prisoner's Dilemma 12

2.6 Nash Equilibrium 13

3. Artificial Intelligence **15**

3.1 General 15

3.2 Use of artificial intelligence 16

4. Modelling AI with game theory **17**

4.1 Game theoretical modelling and solutions 17

4.2 Optimizing minimax algorithm 18

5. Zero-sum tic-tac-toe game **20**

5.1 Goal 20

5.2 Implementation 20

5.3 Testing tic-tac-toe AI 26

6. Conclusions **28**

References **30**

Figures

Figure 1. Rock, paper scissors matrix 7
Figure 2. Rock-paper-scissors game tree 8
Figure 3 Eliminating dominated strategies 9
Figure 4. Eliminating dominated strategies 2 10
Figure 5. Eliminating dominated strategies 3 10
Figure 6. Prisoner's dilemma payoff matrix 12
Figure 7 Nash equilibrium matrix 13
Figure 8. Nash equilibrium search 14
Figure 9. Nash equilibrium search 2 14
Figure 10. Minimax algorithm 18
Figure 11 Alpha-beta pruning branch elimination 19
Figure 12. Tic-tac-toe user interface 21

Tables

Table 1. Minimax efficiency 27
Table 2. Minimax with alpha-beta pruning efficiency 27
Table 3. Minimax algorithm efficiency with alpha-beta pruning and better move ordering 28

Terminology

Actions

Action in game theory are all the possible choices from which the player can choose.

AI

Artificial Intelligence.

Common Knowledge

Common knowledge in game theory is something that all rational players know for certain, and everyone knows that all the other players know that same fact.

Payoff

In any game payoffs represent the motivations of players. Payoff can e.g. be a profit from a certain outcome. Larger value is commonly the best outcome to the player.

Player

Player is a rational decision maker in the game.

Preferences

Preferences describe how the player ranks all possible outcomes from the most desired to least desired.

Outcomes

Outcomes are the possible consequences that can result from the actions.

Unity

Unity is an all-purpose game engine developed by Unity Technologies. Unity game engine supports multiple platforms.

1. Introduction

1.1 Introduction to project

Use of AI has become an important part of day-to-day life and AI can be seen everywhere. When developing AI, the goal is to make the AI make an optimal choice to the given tasks. To achieve that goal, it is important to research mathematical models, concepts and algorithms. In economics, game theory is used for modeling competitive behaviors of firms and in political science, it is used to model actions of voters, politicians, etc (Tadelis 2013, Preface xi). However how can it be used to develop an Artificial Intelligence? Game theory is all about modeling interactions with strategic agents, so the game theoretical modeling seems to be great framework to start with when trying to achieve the goal of making an AI that always makes the optimal choice to its given task. When an AI always makes an optional choice, the AI should never lose the game; at least not a game of a simple zero-sum game. First, in the thesis there will be research of what kind of tools game theoretical modelling gives and what sort of game AI can be developed with the given tools.

In the project, the plan is to research the basics of game theory and to develop a tic-tac-toe game with an Artificial Intelligence that always makes the optimal choice using the concepts and solutions that game theory provides. The plan is also to research and improve computation time of the developed AI with the possible algorithms available. After the AI is complete, the plan is to compare the game theoretical AI to a simpler AI that only analyzes the current state of the game and makes a move with that information. The comparison takes place by making the AI'-s play against each other in the game of tic-tac-toe. If the game theoretical AI always plays the most optimal choice, it should never lose the game. In the end, the AI is reviewed, and it will be seen how it played the game. It will be tested if the game theoretical AI is unbeatable. Also, the performance of the AI is tested along the way of the optimization process.

1.2 Goals, motivation and progression of thesis

The goal for developing an Artificial Intelligence in the game of tic-tac-toe is that it chooses the optimal choice from the given parameters with the mathematical framework that the game theory provides. The AI will be a perfect solver of the game, which means that AI will go through every possible outcome of the game and choose the best possible move of all the possible choices. The goal is to make an Artificial Intelligence also as efficient and fast as possible. The goal is also to help the reader to understand basics of game theory, and how that mathematical framework can be used to develop and optimize Artificial Intelligences into zero-sum games particularly.

The personal motivation for the thesis is to understand the basics of game theory and how it can be implemented in the development of an AI. The whole process of mathematically developing a logical AI is a subject that does not get much attention in school; hence, the goal is to learn and model a logical AI that uses game theoretical solutions. Even if the end project is a simple game, the process of developing the optimal AI with the game theoretical framework is an important subject to learn and understand in future larger projects involving logical decision-making.

The game will be made with Unity3D game-engine because the game itself is not the focus of the thesis but the development of the Artificial Intelligence that the game uses. Unity game-engine will make the development of the game easier and faster. Unity game engine is popular and easy to approach so the platform is good to everyone to come into and build an Artificial Intelligence to their games.

In the thesis, the first plan is to research the concept of game theory and the mathematical framework that it provides including what different forms of games there are, how to present games and what strategies mean. Additionally, Nash equilibrium is introduced. After that the plan is to go through the concept of Artificial Intelligence and what way it has been implemented in zero-sum games.

After the groundwork is done, the plan is to research the ways game theory can be implemented to develop one’s own artificial intelligence for a simple zero-sum game, and how it can be optimized in a few simple methods.

The last part of the thesis is to test the developed game and analyze the developed AI. The testing consists of two parts. First, the testing of the Artificial Intelligence is performance based, where the amount of the computations and computation time is compared to all stages of optimization. At first there will not be any optimization, however the AI still makes the optimal choice according to the zero-sum-game solution. The first optimization applies the alpha-beta pruning search algorithm to decrease the amount of computations. After that, the alpha-beta pruning is improved by calculating the best moves first. The second part of the testing is to test if the optimal AI by making it play game against an AI that makes moves based on the current state of the game and does not go recursively through all possible moves. How well the optimal AI works is also tested manually to see if the AI never loses.

2. Game theory

2.1 Concept of game theory

Game theory provides a mathematical framework that describes situations where rational decision makers are cooperating or in conflict. The rational decision makers strive to make optimal actions that maximizes their payoff. Game theory has been applied into multiple different fields, such as economics, political campaigning, jury voting, auctions, psychology and biology (Tadelis 2013, Preface xi). In computer science the game theory has been used in systems with multiple entities, networking, e-commerce and in the development of Artificial Intelligence. (Shoham 2008, 75.)

The first and earliest example of formal game theoretical analysis is Cournot’s duopoly by Antoine Cournot in 1838 which is a model showing a situation where two firm compete deciding independently and simultaneously on how much output they will produce (Tadelis 2013, 49). In 1921, the mathematician Emile Borel suggested a formal theory of games. In 1944, John von Neumann and Oskar Morgenstern published a book “Theory of games and economic behavior” where they found a consistent solution for two-player zero-sum games that resulted in the game theory becoming a field of its own. In 1950, mathematician John Nash developed a solution concept, which is now called Nash equilibrium that was applicable to a finite number of players and not only to two-player games. (von Stengel & Turocy 2001, 4.)

Game theory is a mathematical framework, and it opens many possibilities to be implemented into the realm of programming. Basically, game theory can be implemented into anything that involves strategic interaction and decision making.

2.2 Presenting games

Different games and their payoffs can be presented in different ways. Some most common ways to present games payoffs are a matrix or a game tree. How the game's payoffs are the most useful to display, varies much depending on what kind of game is played.

In a matrix form, the game’s payoffs are presented in a form of a matrix, and it is most useful in simultaneous two player games. Figure 1 shows the payoffs of the rock-paper-scissors game.

		Player 2		
		R	P	S
	R	0, 0	-1, 1	1, -1
Player 1	P	1, -1	0, 0	-1, 1
	S	-1, 1	1, -1	0, 0

Figure 1. Rock, paper scissors matrix

The players choose from the matrix either R, P or S, and the outcome of the game is the path which they cross. The numbers in the boxes are the payoffs. The first number is the player one’s payoff, and the second number is the second player’s payoff. Payoffs in rock-paper-scissors game are 0 being a tie, 1 a win and -1 if a player loses.

Game tree is represented in a form of a tree and it is best suited for representing a sequential game; however, it can also be used in simultaneous game. In a simultaneous game, the second player does not know the choice the first player made; so the second player does not know in which path in a game tree the player is. In a sequential game, the second player might know the path he is in, depending on the game. Figure 2 is the game tree presentation of the rock-paper-scissors game.

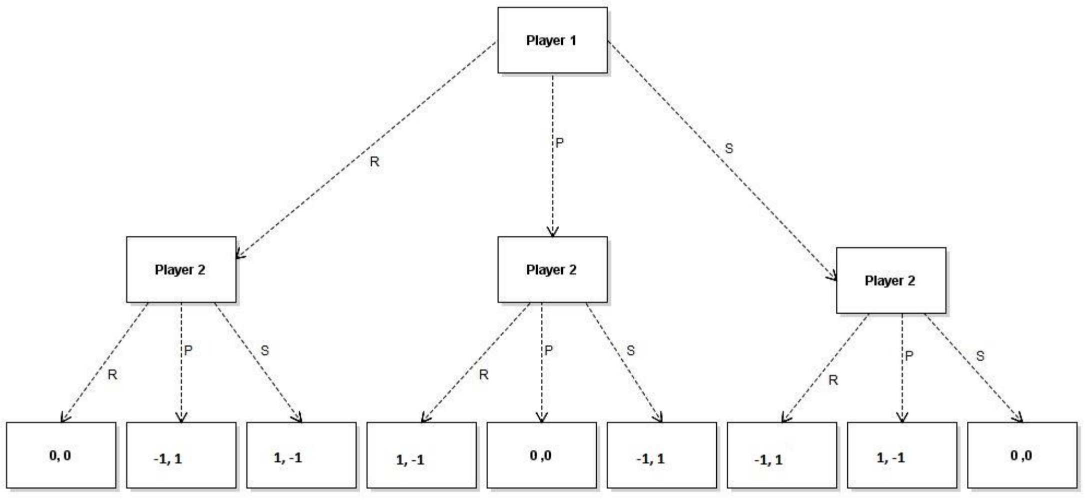


Figure 2. Rock-paper-scissors game tree

2.3 Strategies

In game theory, a strategy is defined as a plan of action to accomplish a specific goal. When talking about pure strategies it is when a player chooses a certain plan of action. Alternatively, a player can choose actions randomly. Mixed strategy is the probability distribution over the player's pure strategies. (Tadelis 2013, 46, 103.)

Different strategies can be categorized into dominating and dominated strategies. When a strategy is dominating, the strategy gives at least as good an outcome as the other strategies. Dominating strategy can either strictly dominate or weakly dominate. In strictly dominating strategy, the strategy always gives the best outcome, no matter what the other players choose. Weakly dominating strategy is when there is at least one set of opponents' actions, for which the strategy gives a better outcome than other strategies, and all the other sets of opponents' actions give the same payoff. Dominated strategy is when a strategy never gives a better outcome than other strategies regardless what another player chooses. Dominated strategy can be strictly dominated and weakly dominated in the same way as the dominating strategies. A strategy can also be intransitive when all strategies are neither dominating nor dominated. (Tadelis 2013, 60-61.)

There is a common technique for solving what strategy a player will use to play optimally by iteratively removing strictly dominated strategies. This is called iterated elimination of strictly dominated strategies or IESDS. The strategy that survives IESDS is called iterated-elimination equilibrium. For an example of how to eliminate strictly dominated strategies, one can picture a payoff matrix as presented in Figure 3. (Tadelis 2013, 64-65.)

		Player 2		
		D	E	F
	A	7, 7	3, 2	6, 8
Player 1	B	2, 4	8, 3	3, 6
	C	3, 1	2, 2	4, 4

Figure 3 Eliminating dominated strategies

As can be seen in Figure 3, player 2 has strategy E, which is dominated because there is always a better outcome in other strategies, no matter what the opponent chooses. Hence, when eliminating strictly dominated strategies it can be removed from the matrix.

		Player 2	
		D	F
	A	7, 7	6, 8
Player 1	B	2, 4	3, 6
	C	3, 1	4, 4

Figure 4. Eliminating dominated strategies 2

Now, in Figure 4 of the updated matrix, in which it can be seen that for the player 1 both B and C strategies are strictly dominated by the strategy A; hence both can also be removed from the payoff matrix. The result (figure 5) is the following trivial game, which returns a unique prediction (A, F) which results in a payoff of (6, 8).

		Player 2	
		D	F
Player 1	A	7, 7	6, 8

Figure 5. Eliminating dominated strategies 3

2.4 Different forms of games

Different situations where individuals are in a strategic interaction are called games. They are categorized into different forms of games based on the features of the game. In game theory, there are sequential and simultaneous games. In a simultaneous game, the players make their choice simultaneously and in a sequential game the players choice is turn based. In a sequential game, it is important to state that the player knows what the previous player chose when they are making their decisions. (Brocas, Carrillo & Sachdreva, 2016.)

When players do not have common knowledge about the game being played, the game is called game of incomplete information. If some players do not know what actions the other players chose; however, he has common knowledge of the game, the game is called game of imperfect information. All simultaneous games are games of imperfect information. (Tadelis 2013 241-242,136.)

Normal form games

In a normal form game there is a finite set of players, a set of actions for each player, a collection of sets of pure strategies and a set of payoff functions for each player that give a payoff value for each combination of actions. (Tadelis 2013, 46.)

Zero-sum games

In zero-sum games in game theory, player's gains and losses are exactly balanced, meaning that a player cannot benefit from a choice without reducing another player's position, hence their payoffs sum to zero. (Tadelis 2013, 101.)

For example, the game of tic-tac-toe is a zero-sum game. Both players have the same gains and losses, and by making a move, it always benefits the player and reduces the opponent's position. If there are two rational players that play against each other and always choose the most optimal choice, the game will always be a tie. In the sample project, the plan is to focus on these games.

Multistage games

A multistage game is a finite number of independent normal form simultaneous-move games of complete information. The Outcome of the stage is common knowledge amongst all players. After the game, another game is followed until the multistage game ends. Total payoff is evaluated from the outcomes of the games sequences. Repeated game a special case multistage game in which the same stage-game is being played sequentially at every stage. (Tadelis 2013, 176, 190.)

2.5 Prisoner's Dilemma

The most common example in game theory is perhaps the Prisoner's Dilemma. It is a static game of complete information with two players, who have commited a serious crime that would be a 5-year prison sentence; however, the prosecutor has only evidence for a minor crime that is a 2-year jail sentence. The prosecutor puts the players in different rooms and offers them a deal that gives them the chance to confess (C) and as a result, the partner in crime gets a 5-year sentence and the other player gets only a 1-year sentence. However, if both players take the deal, both get a 4-year sentence. Players can also stay quiet (Q) and not take the deal and as a result, both players get the original 2-year sentence if both parties decide to stay quiet.

It is reasonable to assume that less time in prison is better, so the payoffs for player 1 can be set as follows:

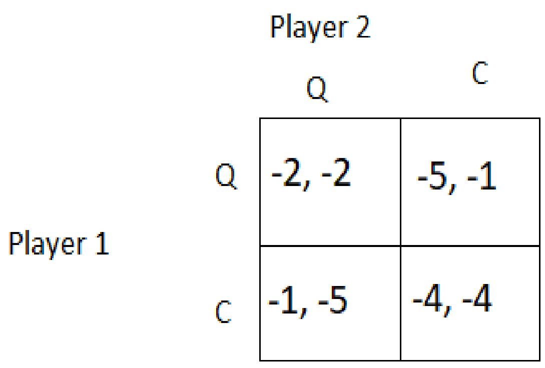
Both stay quiet (Q, Q) = -2

Both confess (C, C) = -4

Player 1 confess, partner stays quiet (C, Q) = -1

Player 1 stays quiet and partner confesses (Q, C) = -5

Payoffs can be represented in a form of a matrix:



		Player 2	
		Q	C
Player 1	Q	-2, -2	-5, -1
	C	-1, -5	-4, -4

Figure 6. Prisoner's dilemma payoff matrix

Figure 6 illustrates that confessing is a strictly dominant strategy because no matter what the opponent chooses, it gives the player a better outcome. Thus, by staying quiet always gives the worst outcome, no matter what the opponent chooses as can be seen in the matrix -1 > -2 and -4 > -5.

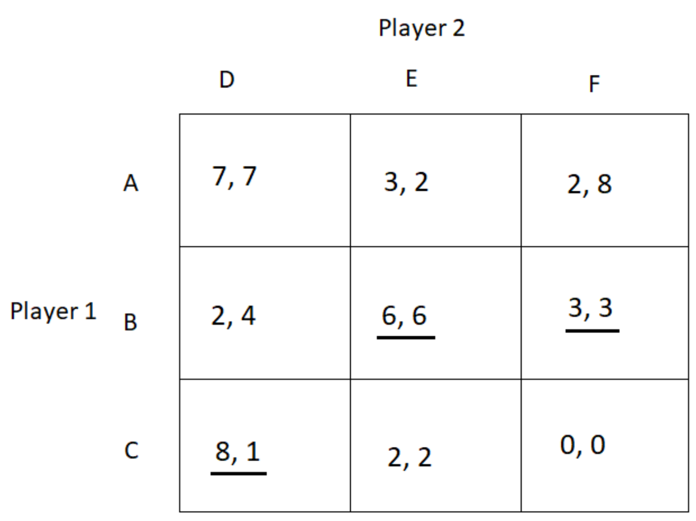
2.6 Nash Equilibrium

Nash equilibrium is a concept by John Nash who won a Nobel Prize in economics for this achievement. Nash equilibrium is an optimal outcome of a game where no player has no benefit from changing only their own chosen strategy after considering the opponent's choice, assuming all other players keep their strategy unchanged. The requirements of Nash equilibrium are that each player is playing the best response to their belief and that believes of the other players are correct. If players are rational, the first requirement is met because a rational player always plays the best response to their belief. There is a simple method to find pure-strategy Nash equilibrium in a matrix game if at least one exists. Payoffs of the game are as follows in a matrix form. (Tadelis 2013 80-82.)

			Player 2	
		D	E	F
	A	7, 7	3, 2	2, 8
Player 1	B	2, 4	6, 6	3, 3
	C	8, 1	2, 2	0, 0

Figure 7 Nash equilibrium matrix

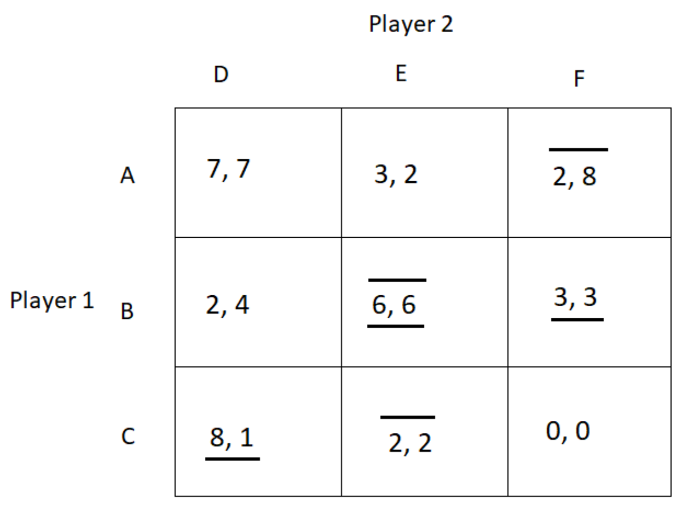
The matrix demonstrates that no strategy is dominated; however, there still is a Nash equilibrium. First, the best payoffs of the player 1 need to be found in the strategies of player 2. For example, if player 2 is playing D, the best response of player 1 is C. So, the best responses are (C, D), (B, E) and (B, F).



		Player 2		
		D	E	F
	A	7, 7	3, 2	2, 8
Player 1	B	2, 4	$\underline{6, 6}$	$\underline{3, 3}$
	C	$\underline{8, 1}$	2, 2	0, 0

Figure 8. Nash equilibrium search

After that the best payoffs of player 2 need to be found in the strategies of player 1 which are (C, E), (B, E) and (A, F).



		Player 2		
		D	E	F
	A	7, 7	3, 2	$\overline{2, 8}$
Player 1	B	2, 4	$\overline{\underline{6, 6}}$	$\underline{3, 3}$
	C	$\underline{8, 1}$	$\overline{2, 2}$	0, 0

Figure 9. Nash equilibrium search 2

If a strategy is chosen in both groups, the strategy is Nash equilibrium. This example shows that (B, E) is the unique pure-strategy Nash equilibrium.

In the prisoner’s dilemma it can be said that the game has a unique Nash equilibrium (Confess, Confess) because if the player is playing dominant strategy he is playing the best response - no matter what the opponent chooses.

3. Artificial Intelligence

3.1 General

Artificial intelligence or AI is a machine taking actions to maximize its chances to complete a goal given to the machine. AI can, for example, mimic human actions of learning and problem solving. Another important concept in the realm of artificial intelligence is machine learning. While AI is a broader concept of machines completing specific goals, machine learning is an application of AI based on the concept of giving a machine data and letting it learn from it by self. When teaching computers to think, the neural network is an important concept where computer systems are designed to classify information like a human brain. (Marr 2016.)

The term Artificial Intelligence was first used by John MacCarthy in 1956 in an academic conference; however, the concept has been visited earlier. In 1945 Vannevar Bush gave a seminar "As we may think" where he proposed a system that amplifies people's knowledge. In 1950, English mathematician Alan Turing, who is considered to be father of modern computer science wrote a paper that centered on the idea that machines could simulate human behavior such as play games like chess. Alan Turing posed the question "Can machines think" which came to be known as a Turing test. Turing test is passed if a human cannot tell if it is interacting with a human or with a machine. (McGuire & Smith 2006, 4-5.)

Artificial intelligence has improved a great deal in the last 20 years, and the dangers of AI have become a current issue. The Co-founder of Tesla and SpaceX, Elon Musk stated on August 2017 on twitter that “If you're not concerned about AI safety, you should be. Vastly more risk than North Korea.” Mainly because of that reason, Elon Musk with Sam Altman created a non-profit AI research company OpenAI that focuses on building a safe Artificial General Intelligence or AGI. AGI is intelligence of a machine that can make any intellectual task that a human can. AGI can also be referred to as a Strong AI. OpenAi said in a statement in 2015 that “AI systems today have impressive but narrow capabilities. It seems that we’ll keep whittling away at their constraints, and in the extreme case they will reach human performance on virtually every intellectual task. It’s hard to fathom how much human-level AI could benefit society, and it’s equally hard to imagine how much it could damage society if built or used incorrectly.” (OpenAI 2015.)

3.2 Use of artificial intelligence

AI and machine learning are nowadays widely used and have many different applications in today's society. For example, PayPal uses machine learning against money laundering, cybersecurity company Deep instinct against malware, Amazon optimizes inventory and product recommendations and Google's DeepMind AI improves cooling of data centers. Voice recognition programs such as Siri, Alexa and Google assistant are most noticeable due to their popularity. A study by the Stanford computer scientist James Landay and his colleagues found that speech recognition is approximately three times faster than typing and its error rate is 4.9%. Another popular implementation of machine learning is image recognition that can be seen, for example, in iPhone’s face recognition and in Facebook: when one takes a photo, it prompts one to tag one’s friends to those photos. Image recognition is even replacing some ID cards in some corporations. (Brynjolfsson & Mcafee, 2017.)

In video games Ai is used to represent the behavior of the non-player characters(NPC). Video games are supposed to be enjoyable, so the optimal AI is not implemented to the extent that the game is unbeatable. Gary Eastwood wrote in an article in January 2017 called "How video game AI is changing the world" that "AI developer are now trying to create AIs that can actually think, learn and develop their own personalities". (Eastwood 2017.)

in 1996, a Chess computer called Deep Blue beat the reigning world champion in a chess game for the first time. In 2011, IBM Watson question-answering computer beat the best players Brad Rutter and Ken Jennings in the American game show Jeopardy. In 2013, Watson's commercial application would be used in utilization management decisions in lung cancer treatments in New York City. Watson showed that it made accurate decisions 90% of the time, and 90% of the nurses in the field trusts its guidance. (Upbin 2013.)

In 2016, Google developed an artificial intelligence AlphaGo that beat a professional player in Go which has been said to be a very difficult game for an Ai because of the amount of different choices the game has. The experts said that it would not be possible in 10 years. Most recently, in December 2016 Ai called DeepStack beat 11 professional poker players in head-up no-limit hold'em poker.(Biggins 2017.)

Google has also developed an Ai called DeepMind that has learned to play 49 Atari games and beat them. This shows how much computer learning has developed and illustrates all the possibilities it holds in the future. (Saxena 2015.)

4. Modelling AI with game theory

4.1 Game theoretical modelling and solutions

In game theory, there are different games to various situations. The question that needs to be answered is what theory to implement in which decision problem. John Von Neumann and Oskar Morgenstern in the book “Theory of games and economic behavior” in 1953 created a solution called minimax-solution, which is a commonly accepted solution to two-player zero-sum games. In zero-sum game, the goal of mini-max solutions is to minimize the opponent’s maximum payoff. Since the game is a zero-sum game, the moves that reduces the opponent’s position the most is also the most beneficial to the player. By minimizing the opponent’s maximum payoff, the player is also minimizing his maximum loss. In zero-sum games, the solution that minimax algorithm returns is Nash equilibrium. (O’Neill 1987.)

Most of the common strategic games are zero-sum games. There are multiple zero-sum games e.g. tic-tac-toe, connect four, chess and Go that can benefit from the minimax solution when creating an optimal Ai. The Solution used when making an AI to a two-player zero-sum game is a minimax algorithm. A minimax algorithm is a re-cursive algorithm that plays the game ahead till it finds a terminal state of the game. The terminal state ending with players winning it results a positive value, and if an opponent wins, the value will be negative. If the game will end with a tie, the value will be a zero. On the player's turn it will prefer the maximum value, and on the opponent's turn it will prefer the minimum value. AI can be made to prefer a quicker win over long term wins by adding the value of depth in the game tree. Whenever an algorithm goes deeper into to the stack, the end node value is subtracted by one on maximizer's turn and added one on the minimizer's turn. Figure 10 describes the minimax algorithm.

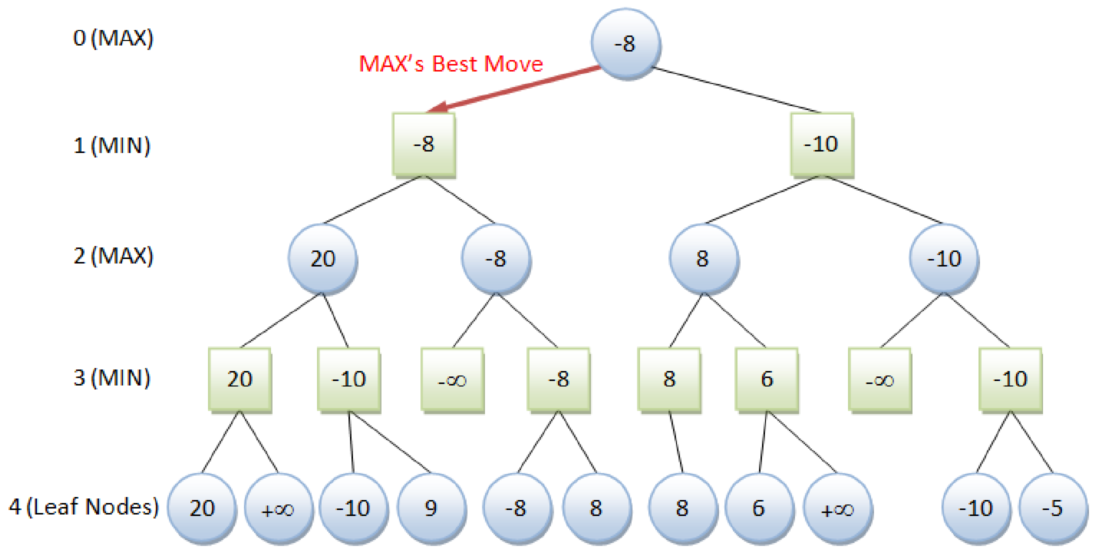


Figure 10. Minimax algorithm (How minimax algorithm works)

4.2 Optimizing minimax algorithm

Even when the minimax algorithm works as its own when implemented, it can consume a lot of processing power specially in games where there are a lot of possible moves to make in the game. There are few easy methods how to improve the number of nodes the minimax algorithm has to go iterate through the game tree.

Alpha-beta pruning searches the algorithm to help to decrease the number of nodes that the minimax algorithm iterates. By implementing alpha-beta pruning, minimax algorithm stops evaluating moves in a branch where a move is found that is worse than a previous move because whatever value is in that branch, it will return a worse value than the previous move. When using alpha-beta pruning it returns the same value as minimax algorithm; however, it does not iterate to unnecessary branches or leaf nodes. This is a great way to reduce the processing time and improve performance. It is most commonly used in two-player zero-sum games such as tic-tac-toe, Go and chess etc. Alpha-beta pruning can be implemented into trees of any depth.

Alpha-beta pruning gets its name from the two-back-up parameter values alpha and beta. Both values are set to the worst possible value in the beginning. Alpha is set to the lowest possible value and beta to the largest possible value. When minimax is minimizing the alpha value, the changes according to the returned terminal node value and the maximizer change the beta value respectively. When the alpha value is larger or equal to beta, it is not necessary to iterate those branches because it will make the parent node worse. Figure 11 describes the game tree branch elimination of alpha-beta pruning.

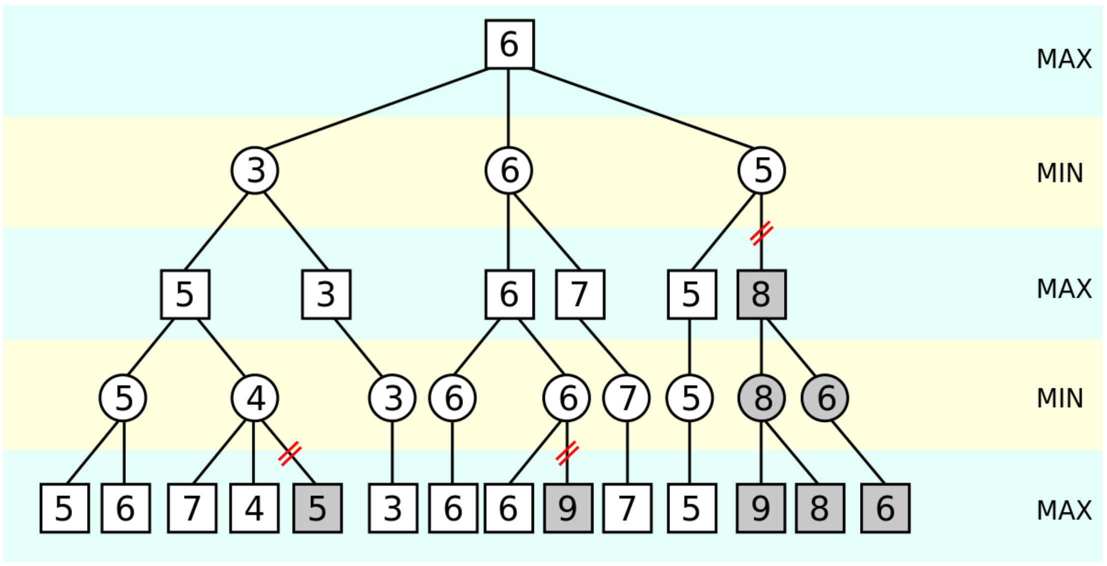


Figure 11 Alpha-beta pruning branch elimination (Example of alpha-beta pruning)

Alpha-beta pruning can be improved by ordering the moves in the way that the algorithm iterates through the best moves first, so the alpha-beta pruning can eliminate most of the branches.

5. Zero-sum tic-tac-toe game

5.1 Goal

The goal in the sample project was to develop an unbeatable tic-tac-toe game. Because the tic-tac-toe is a zero-sum game the solution to the game is the minimax algorithm. Another goal was to optimize the algorithm with alpha-beta pruning and better move ordering and test the AI's performance while only using the minimax and then while the AI is optimized with alpha-beta pruning and finally fully optimized with both alpha-beta pruning and better move ordering.

The goal is also to test two other simpler AI's to play against the AI that uses the minimax algorithm. The simple AI's prioritizes certain actions in different situations. AI’s first priority is that if it can win, it will do so. The second priority is that if the opponent has two straight the AI prevents that line. The third priority is to make randomly two straight rows and as the final priority, put either to the center tile or if that is taken then to a corner tile and if those are taken, it will put one randomly. In the other simple AI, the AI will not prioritize corner or the center tile, so if the AI will not be able to win, stop a win, make two straight, it will make a random move from all the free tiles. Those two AIs will play against the minimax AI 30 games, and it can be observed how many games the AI using minimax algorithm wins.

Tic-tac toe game will be made with Unity3D game-engine to help the development process. C# programming language was used for programming the game logic and the AIs.

5.2 Implementation

In the game of tic-tac-toe, there is a 3x3 grid where a player can put his/her mark, which is either X or O. In Unity 3D, the grid is easy to do with UI button elements. When the mark is made, the buttons ‘text is changed to X or O depending on what mark the player has. The buttons are put into array where they can easily be handled.

The program itself consists of classes called gamecontroller and gridpace. Gridspace controls all buttons and their actions. When the button is clicked, the action sets the button text as the current turn value and sets the button to be non-clickable. The user interface of the tic-tac-toe is illustrated in Figure 12.

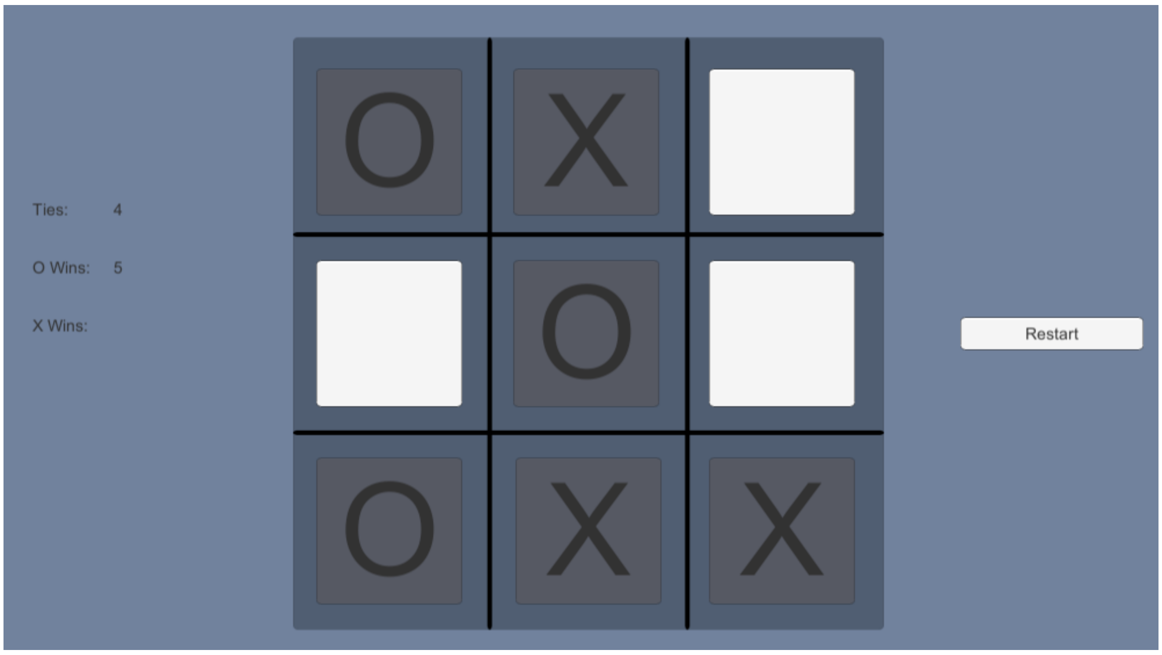
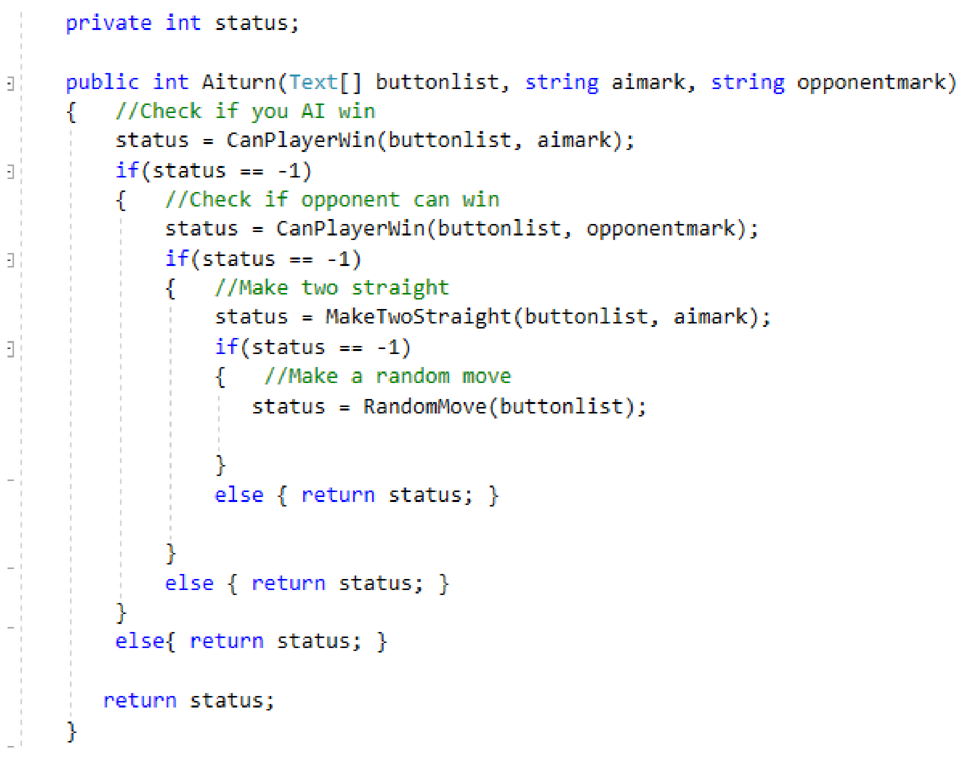


Figure 12. Tic-tac-toe user interface

The gamecontroller class holds all the board's buttons, all AI instances and functions that check if the game ends, execute AI's turns, and changes' turns. The actual Ais are in their own classes.

There are two AI classes smartAI and simpleAI. SimpleAI class has 4 function: Aiturn, CanPlayerWin, MakeTwoStraight and RandomMove. Aiturn prioritizes the best action. Status variable is -1 if the action requirement was not met. When the status is other than -1, the status will be the button array index. The following code snippet illustrates the Aiturn function of the simpleAI that prioritizes the best actions.

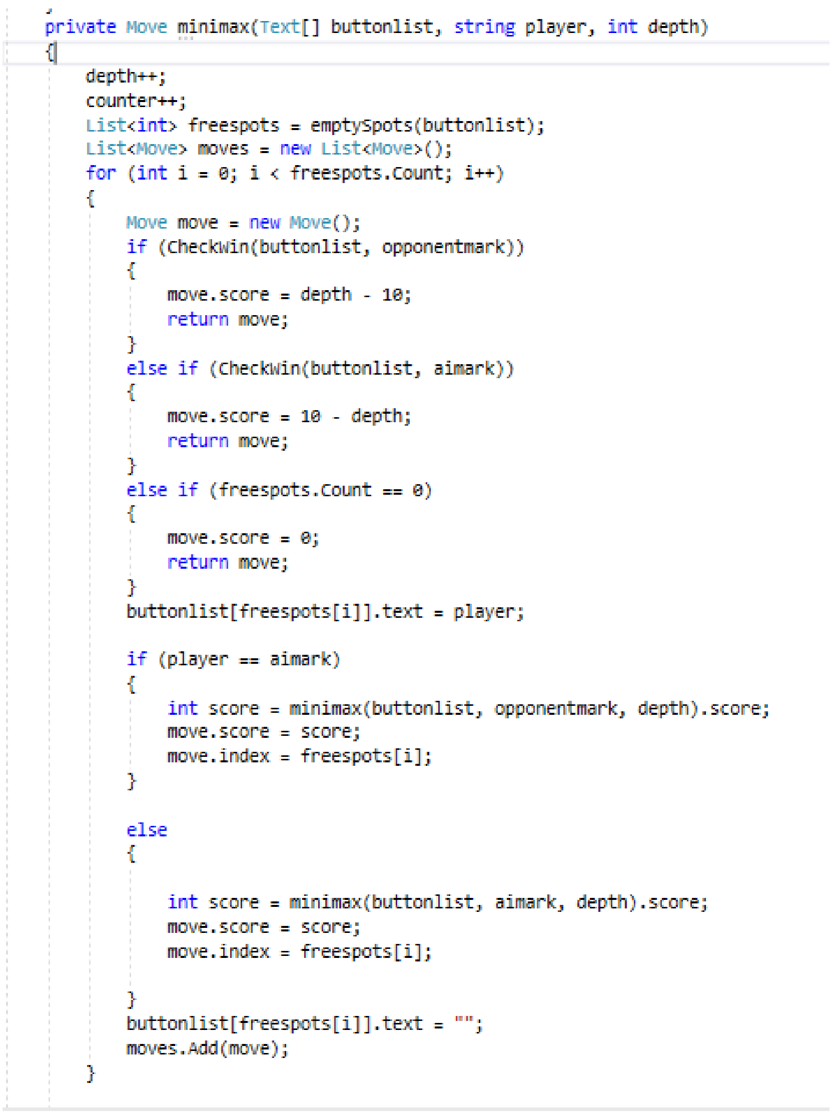


```
private int status;

public int Aiturn(Text[] buttonlist, string aimark, string opponentmark)
{   //Check if you AI win
    status = CanPlayerWin(buttonlist, aimark);
    if(status == -1)
    {   //Check if opponent can win
        status = CanPlayerWin(buttonlist, opponentmark);
        if(status == -1)
        {   //Make two straight
            status = MakeTwoStraight(buttonlist, aimark);
            if(status == -1)
            {   //Make a random move
               status = RandomMove(buttonlist);

            }
            else { return status; }

        }
        else { return status; }
    }
    else{ return status; }

   return status;
}
```

CanPlayerWin function checks horizontal, vertical, and diagonal rows and checks if the same player has two marks in those rows, and if the final spot in the row is empty. If those conditions are met it will return the index of the move. MakeTwoStraight function checks the horizontal, vertical and diagonal rows in the same way as the CanPlayerWin does, however, it will check if two of the tiles are free. If there is more than one possible choice, it will choose one randomly amongst the possible moves. RandomMove chooses a free tile randomly in the other, simpleAI and in the other, it will prefer the middle tile and corner tiles before choosing the move randomly.

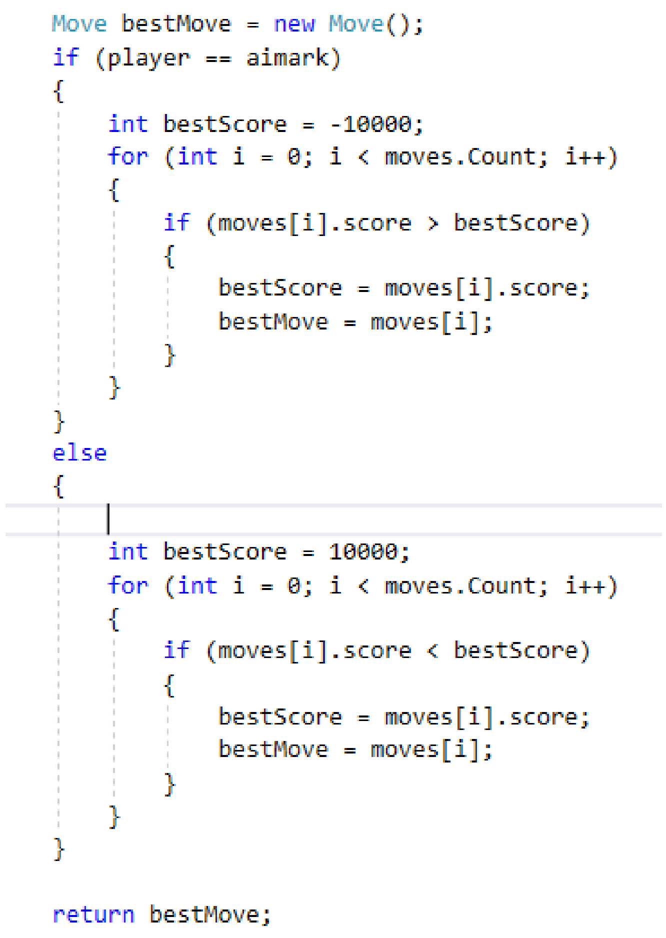
SmartAI is the AI uses the minmax algorithm. SmartAI class hold another class Move that holds two integers called index and score. The score value holds the minimax value of the move, and the index is the value in which row the move was made. The functions in the SmartAI class are findBestMove, Minimax, emptyspots, and Checkwin. FindBestMove function basicly just calls minimax function and is used to observe and test what the minimax function returns. The gamecontroller calls this function. Emptyspots function is pretty self explanatory it gets all the free spots from the gameboard. Checkwin takes the mark O or X as parameter and checks if that player has won.

The minimax function is the actual recursive minimax algorithm that goes through the game and gives the best move to be played. First, the minimax algorithm searches all available spots and starts looping through them. First, it checks if a player has won the game and inserts the Move class instance score as depth - 10 or 10 - depth depending on who won, and if there are no more free spots and it is a tie, it return the value of zero. By using the depth value, the AI will prefer a quicker win over a longer win. After that the minimax function makes a move to a free spot and depending on whose turn it is, it makes a recurisive call to the function. The move that the recursive call returns will save the score to the Move's score and puts it is index value as the loop index. After that, it will undo the move and inserts the move to a list. the code snippet below illustrates the recursive part of the minimax algorithm.



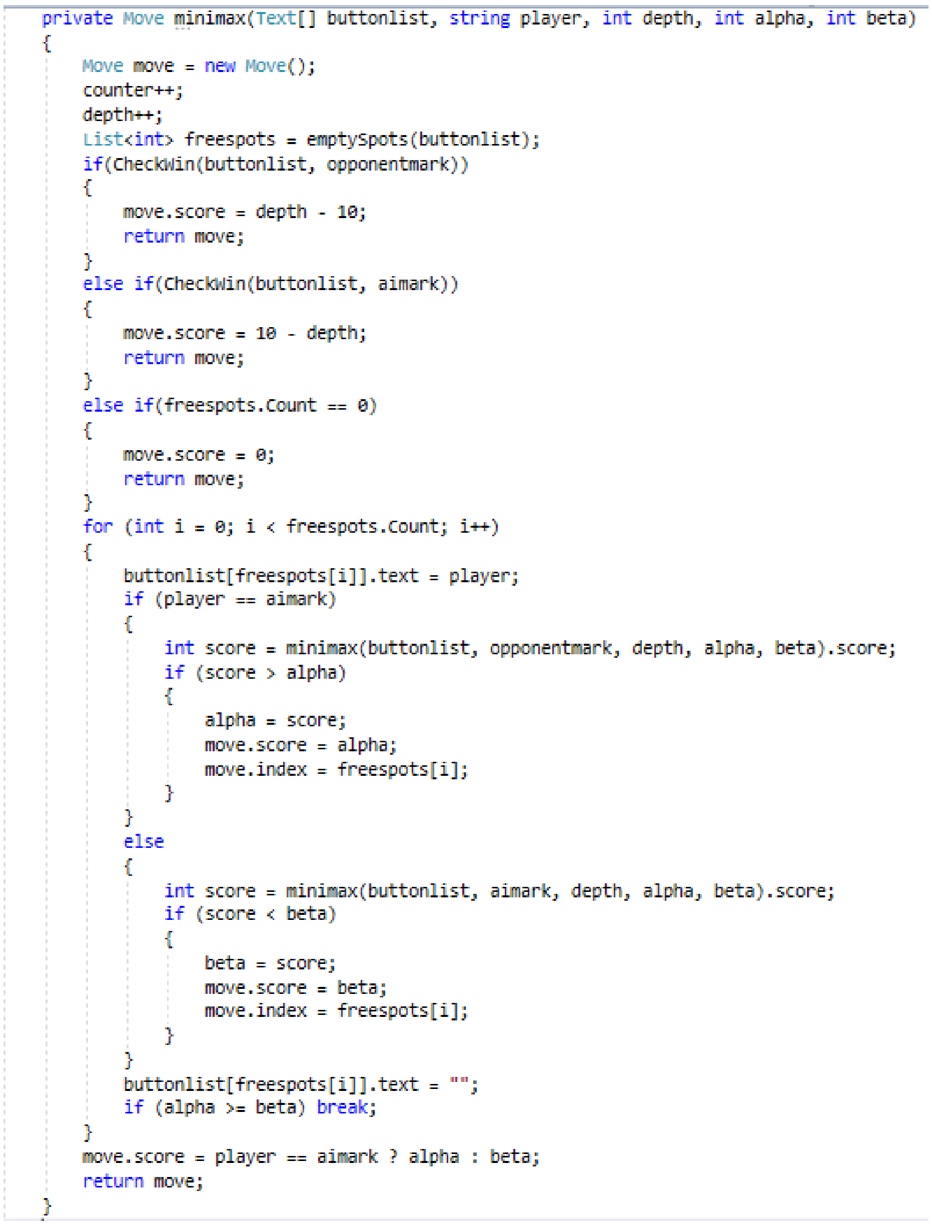
```
private Move minimax(Text[] buttonlist, string player, int depth)
{
    depth++;
    counter++;
    List<int> freespots = emptySpots(buttonlist);
    List<Move> moves = new List<Move>();
    for (int i = 0; i < freespots.Count; i++)
    {
        Move move = new Move();
        if (CheckWin(buttonlist, opponentmark))
        {
            move.score = depth - 10;
            return move;
        }
        else if (CheckWin(buttonlist, aimark))
        {
            move.score = 10 - depth;
            return move;
        }
        else if (freespots.Count == 0)
        {
            move.score = 0;
            return move;
        }
        buttonlist[freespots[i]].text = player;

        if (player == aimark)
        {
            int score = minimax(buttonlist, opponentmark, depth).score;
            move.score = score;
            move.index = freespots[i];
        }

        else
        {

            int score = minimax(buttonlist, aimark, depth).score;
            move.score = score;
            move.index = freespots[i];

        }
        buttonlist[freespots[i]].text = "";
        moves.Add(move);
    }
```

After all the moves are in the list, the algorithm finds the best move among them by looking whose turn it currently is and depending on if the player is minimazer or maximizer, it sets the starting value to high or low. In the end, it returns the move with the best score. Code snippet below searches the best move from the possible moves.



```
Move bestMove = new Move();
if (player == aimark)
{
    int bestScore = -10000;
    for (int i = 0; i < moves.Count; i++)
    {
        if (moves[i].score > bestScore)
        {
            bestScore = moves[i].score;
            bestMove = moves[i];
        }
    }
}
else
{

    int bestScore = 10000;
    for (int i = 0; i < moves.Count; i++)
    {
        if (moves[i].score < bestScore)
        {
            bestScore = moves[i].score;
            bestMove = moves[i];
        }
    }
}

return bestMove;
```

When implementing the alpha-beta pruning to the minimax algorithm it works pretty much the same way with a few differences. In alpha beta pruning two values are added, alpha and beta. Both players start with the worst possible score, so alpha is the minimum score that the maximizing player is using, and beta is the maximum score that the minimaxing player is using respectively. When the minimazing player's beta score becomes less that the maximizer's alpha value, the maximizer does not need to go any deeper in that game tree brach because no matter what the value will be in that branch, it will make the overall value worst. So when alpha >= beta, the recursive algorithm will break and move on to the next branch. The code snippet below illustrates the working minimax algorithm with alpha-beta pruning.



```
private Move minimax(Text[] buttonlist, string player, int depth, int alpha, int beta)
{
    Move move = new Move();
    counter++;
    depth++;
    List<int> freespots = emptySpots(buttonlist);
    if(CheckWin(buttonlist, opponentmark))
    {
        move.score = depth - 10;
        return move;
    }
    else if(CheckWin(buttonlist, aimark))
    {
        move.score = 10 - depth;
        return move;
    }
    else if(freespots.Count == 0)
    {
        move.score = 0;
        return move;
    }
    for (int i = 0; i < freespots.Count; i++)
    {
        buttonlist[freespots[i]].text = player;
        if (player == aimark)
        {
            int score = minimax(buttonlist, opponentmark, depth, alpha, beta).score;
            if (score > alpha)
            {
                alpha = score;
                move.score = alpha;
                move.index = freespots[i];
            }
        }
        else
        {
            int score = minimax(buttonlist, aimark, depth, alpha, beta).score;
            if (score < beta)
            {
                beta = score;
                move.score = beta;
                move.index = freespots[i];
            }
        }
        buttonlist[freespots[i]].text = "";
        if (alpha >= beta) break;
    }
    move.score = player == aimark ? alpha : beta;
    return move;
}
```

Optimizing with better move ordering is an easy way to make the alpha beta pruning work optimally with fewer computations. When minimax goes through the best moves first, the alpha beta pruning cuts more branches because they are more likely to be worse than the first move. In the game of tic-tac-toe, the best move usually is the middle tile because one can make most straight lines. The second best move is to the corner tiles where the player can make three different straight lines. The worst position is with the side tiles where the player can only make two straight lines. In the code, all that needs to be done is to order the free spots into an order that prioritizes the center and corner tiles by putting them first in the list of possible moves.

Before going through the minimax algorithm to find the Nash equilibria, one can search for straight win because that winning move is a dominating strategy. Alternatively if AI cannot win but the opponent will win on the next turn, the blocking move is a dominating strategy. It is implemented in code snippet below, where those checks are made before the minimax algorithm.

```
Move bestmove = new Move();
bestmove.index = CanPlayerWin(buttonlist, aimark);
if (bestmove.index != -1)
{
    return bestmove.index;
}
else
{
    bestmove.index = CanPlayerWin(buttonlist, opponentmark);
    if (bestmove.index != -1) return bestmove.index;
}
bestmove = minimax(buttonlist, aimark, 0, int.MinValue + 1, int.MaxValue - 1);
```

5.3 Testing tic-tac-toe AI

The goal to achieve an unbeatable AI was succesful. The games personally played by the writer always ended with a tie or victory for the AI. The game of tic-tac toe is a simple game, and it can easily be seen if after a certain move it is possible to win the game or not.

To test how good the game theoretical AI actually is, it is neccessary to also play against other AI's. This can be automated by making the game theoretical AI to play against different simpler AI's multiple times and keep track on the scores. Finally, the AI played against the other game theoretical AI.

The game theoretical AI played against two different simpler AI's against the AI that plays the game prioritizing the current game state in order of winning, preventing losing, making two straight and finally putting the mark randomly. Against that AI the minimax AI played 30 games, winning 17 games and playing a tie 13 games. Against the AI priorizises center and corner tiles before making random move, it played all the 30 games with a tie with the game theoretical AI. When the game theoretical AI played against the same AI all the games ended with a tie.

The start of the game needs the most of computation power so the most important part to research was to find out how many computations were needed and how long the start took. There are moves at the start that the player makes that are harder for the minimax to respond. The moves are categorisized into easy, medium and hard.

The Start of the game takes the most computations depending on where the player puts his/her first mark. Table 1 describes the amounts of computations and how much time they took in minimax algorithm without any optimization.

Table 1. Minimax efficiency

	Easy	**Medium**	**Hard**	**AI's first move**
Computations	55504	59704	63904	549945
Time(ms)	253 - 260	272 - 285	270 - 300	2757

The rest of the game's computations depends on how the game is going; However in the middle game there are 898 – 1404 computations taking 8 to 10ms. In the end, there are only 42- 60 computations and it takes 4ms to compute.

When the alpha-beta pruning is the implemented, the computations in the start decrease significantly. Alpha-beta pruning works as its best when the player puts his/her mark on the first tiles. The computations after the first move and AI's first move with alpha-beta pruning are illustrated in Table 2.

Table 2. Minimax with alpha-beta pruning efficiency

	Easy	**Medium**	**Hard**	**AI's first move**
Computations	2459	3957	5631	16771
Time(ms)	14	20	30	73

At halfway of the game, the amount of computations is between 119-525, and it takes around 5ms. In the end of the game, the computations are between 19-42 and it takes 4ms.

With better move ordering, the minimax works even better decreasing the number of computations. The computations after the first move when better move ordering is implemented are illustrated in Table 3.

Table 3. Minimax algorithm efficiency with alpha-beta pruning and better move ordering

	Easy	**Medium**	**Hard**	**Ai's first move**
Computations	1607	1998	2174	7273
Time(ms)	15	16	17	45

6. Conclusions

Game theory is a very wide mathematical framework so it has great potential in a field of computer science, specially when developing different AI's to make an optimal choice. When developing AI to zero-sum games, finding the Nash equlibrium is quite straightfoward with the minimax algorithm. Minimax algorithm is simple and easy to understand, and improving the amount of computations it has to make is easy with the alpha-beta pruning search algorithm, which improves the game tree search dramatically.

Tic-tac-toe is a simple game and making an AI that works as well as the game theoritical AI can be done without recursively going through the possible moves of the game. The main point is the fact that this solution to zero-sum game was interesting because the logic can be implemented to basically any other zero-sum game. All that is different in those games is how moves are made and how the game is won.

The goal to make an AI for tic-tac-toe that is unbeatable was succesful. The game theoretical AI won all the games played against the other simpler AI’s. Personally I was unable to beat the AI either. After playing couple of games against the game theoretical AI a pattern can be noticed, that makes developing an AI that does not use the the minimax solution as unbeatable quite easy. Great part of developing an AI that uses minimax algorithm is that the same logic can be used to develop AI’s to other zero-sum games.

The alpha-beta pruning algortihm ended up being highly effective. When AI made the first move, the amount of computations were 3% of the original minimax algorithm. After the move ordering was implemented the amount of computations were 1.3% of the original amount of computations. Minimax algorithm was very effective and easy to optimize to the point that the computition time was low enough to game to be playable.

Zero-sum games can be improved even further by using bitboards and trasposition tables. When bitboards are implemented in the game, player’s positions are stored as a series of one’s and zero’s. Zero denoting empty position and one denoting player’s position. Checking if the game ended or making an move, is done by bitwise actions. When the minimax algortihm iterates through the game tree and checks when the game ends along the way, it is the best to do with minimal processing power. When the AI has to handle nothing but bitboard, the computation time improves significantly. (Herzberg 2017.)

When minimax algorithm iterates through the game, it ends up analyzing same position multiple different ways. Solution to that is using Transposition table, which is mainly a hash table with fixed storage size. Trasposition table caches previous computations to avoid computing the same move twice. By caching previous move the AI trades compution time against memory space. Unfortunately all the positions cannot be stored, so frequent positions or positions that take more time to compute are great positions to save in to the cache. Saving the most recent postions is also a great strategy, because resently explored nodes are close to the current position, which increases the probabilty to find same move from the cache. (Pons 2017) In the case of the tic-tac-toe, the further optimization is not necessery, because of the small size of the game.

References

Biggins, M. 2017. It's All Fun and Games until AI wins them all. Accessed 3 May 2018. Retrieved from https://hackernoon.com/its-all-fun-games-until-ai-wins-them-all-d5d657b064be

Brynjolfsson, E & McAfee, A. 2017. The business of artificial intelligence. Harvard business review. Accessed on 3 May 2018. Retrieved from https://hbr.org/cover-story/2017/07/the-business-of-artificial-intelligence

Figure 13. How minimax algorithm works. Accessed on 2 May 2018. Retrieved from https://www.ntu.edu.sg/home/ehchua/programming/java/images/GameTTT_mini-max.png

Figure 11. Example of alpha-beta pruning. Accessed on 2 May 2018. Retrieved from https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning#/media/File:AB_pruning.svg

Herzberg, D. 2017. Bitboards and Connect Four. Accessed on 8 May 2018. Retrieved from https://github.com/denkspuren/BitboardC4/blob/master/BitboardDesign.md

McGuire, B & Smith, C. 2006. The History of Artificial Intelligence. History of Computing. University of Washington. Accessed on May 4 2018. Retrieved from https://courses.cs.washington.edu/courses/csep590/06au/projects/history-ai.pdf

Marr, B. 2016. What is the difference between artificial intelligence and machine learning? Forbes, 6 December 2016. Accessed on 3 May 2018. Retrieved from https://www.forbes.com/sites/bernardmarr/2016/12/06/what-is-the-difference-between-artificial-intelligence-and-machine-learning/#7601f5012742

O'Neill, B. 1987 Nonmetric test of the minimax theory of two-person zerosum games. *Proceedings of the National Academy of Sciences*, *84*, 2106-2109.

OpenAI. 2015. Accessed on 2 May 2018, Retrieved from https://blog.openai.com/introducing-openai/

Pons, P. 2017. Solving Connect 4: How to build a perfect AI. Accessed on 4 February 2018. Retrieved from http://blog.gamesolver.org/solving-connect-four/

Saxena, S. 2015 AI masters 49 Atari 2600 games without instructions. Accessed on 3 May 2018. Retrieved from https://arstechnica.com/science/2015/02/ai-masters-49-atari-2600-games-without-instructions/

Tadelis, S. 2013. Game theory introduction. 1th. ed. Princeton University Press.

Unity3D. 2018. Website. Accessed on 10 May 2018. Retrieved from https://en.wikipedia.org/wiki/Unity_(game_engine)

Upbin 2013 IMB's Watson Gets its First Piece Of Business in Healthcare. Forbes Accessed on 7 May 2018. Retrieved from https://www.forbes.com/sites/bruceupbin/2013/02/08/ibms-watson-gets-its-first-piece-of-business-in-healthcare/#6faf6f865402 von Stengel, B & L. Turocy, T. 2001. CDAM Research Report. Game theory. Accessed 7 May 2018. Retrieved from http://www.cdam.lse.ac.uk/Reports/Files/cdam-2001-09.pdf

Shoham, Y. 2008. Computer science and Game Theory. *Communications of the ACM, 51,* 72-79.