POLITECNICO DI TORINO

MASTER's Degree in COMPUTER ENGINEERING

MASTER's Degree Thesis

SMaILE game: application of search and learning algorithm within combinatorial game theory

Supervisors

Prof. GIACOMO COMO

Candidate

DAVIDE BATTAGLIA

DECEMBER 2022

Abstract

The SMaILE-App is a novel educational mobile application aiming at making Artificial Intelligence mechanisms and applications accessible to a broad audience of teenagers and non-experts. It introduces fundamental AI principles through an engaging gamified learning environment. The SMaILE-App has been developed based on the educational principles of “learning by doing” and “playful learning”. The development of the SMaILE-App is now at its large stages and the application is currently being tested with a randomized control trial study in Piedmontese schools to determine its actual impact on the developments of the students’ abilities.

The contribution made in this thesis work was to develop a tool necessary to carry out two activities within the SMaILEApp application related to the game theory and the reinforcement learning topic. Specifically, comparing two distinct machine learning models to determine which is best for use in this specific educational purpose. These artificial intelligence models are trained to play the tile-placing board game OKPLAY, which is analogous to the well-known game Four-In-A-Row, and to produce customized AI, or "bots," to play in specific contexts. The first activity on Game Theory involves confronting a bot at OKPLAY to learn how to build a winning strategy based on the opponent’s moves. The second task on Machine Learning allows you to try to set the training parameters of Artificial Intelligence to collide with another bot having a different setting. This activity teaches the importance of the training phase to make it perform a specific task to the fullest.

To do so, two different Machine Learning models have been explored, created and tested. The first model extracts knowledge from an initial dataset using Generative Adversarial Networks. The process relies on contrasting the dataset’s original values with the false ones produced by the neural network. The second model uses Reinforcement Learning, which gets knowledge by playing with itself. It does not need other information outside the condition of victory and the subsequent calculation of the movement. The second one ended up being the best option for the SMaILE educational purpose as it allowed the generation of different bots by changing more training parameters and other initial settings.

Acknowledgements

Table of Contents

List of Tables

List of Figures

Acronyms

AI
Artificial Intelligence

MCTS
Monte Carlo Tree Search

RL
Reinforcement Learning

BCELOSS
Binary Cross-Entropy Loss

MSE
Mean Square Error

CSP
Constraint Satisfaction Problem

UCB
Upper Confidence Bound

UCT
Upper Confidence bound for Trees

GAN
Generative Adversarial Network

SMaILE
Simple Methods for Artificial Intelligence Learning and Education

ICT

Information and Communication Tecnologies

Chapter 1

Introduction

1.1 The SMaILE Project

This thesis presents an application to match the educational purpose of the Simple Method for Artificial Intelligence Learning and Education (SMaILE) project. To achieve this goal, SMaILE adopts an educational methodology that aims to maximise the children's involvement by using game theory, gamification codes, an online portal, and a compelling new app, the SMaILEApp, through which to learn and use AI.

1.2 The SMaILE-App and its games

The SMaILEApp is an educational game that aims at introducing AI to young learners attending lower and upper secondary schools. It aims to engage them in entertaining, hands-on learning activities concerning AI to stimulate their creativity and reinforce their sense of agency. The players experience a specific AI area by becoming themselves the AI, then looking at what the AI does, the formalization of the concept is introduced. At the same time, since AI is based on mathematics, spatial ability and computational thinking, the users should also strengthen their competencies in those related fields. Developing such skills in young people is likely to significantly impact their future development and society [1].

The story of the app is related to the city of the future. The players are the majors that need to create and populate a sustainable city. For example, the city building activity is connected to the CSP constraints, and each activity/service in the town is related to an AI concept. The SMaILEApp city comprises 7 areas, each one associated with a micro-game to introduce a learning outcome. Among them are the park and the station where the player is asked to play against a bot to OKPLAY game. In the station, the user trains a bot to compete against other bots.

This work describes the creation of the system behind the park and the station. The park aims to introduce adversarial search strategies, while the topic of the station is Machine Learning (ML). It has been decided to have the exact game mechanism inside both activities to simplify the user experience. More precisely, the OKPLAY game served as a model.

That is somewhat comparable to the well-known game Four-In-A-Row. In the park, users should play against a bot to win by looking for the winning strategy. In the station, users should set some training parameters in order to build a bot that will play against an adversarial bot. Then, from the development point of view, creating a bot with the essential gaming skills was necessary to play against a human in the OKPLAY game.

1.3 Thesis contribution

In this thesis, Two distinct models have been studied in parallel to decide which ML model to use to implement the OKPLAY AI. This result would allow us to apply AI to investigate more complex combinatorial games to solve them and learn more about winning methods.

1.4 Thesis Structure

Below is the organizational structure of this thesis, with a description of the various chapters and topics.

- The second chapter, SMaILEApp, explains the concept underlying the application created to carry out the SMaILE project. The activity of building your own city is the application's core, through which you can access the following tasks. In the end, there will be an overview of its design and implementation.
- In the third chapter Combinational Games, combinatorial games and the theory and classification needed to study the OKPLAY game will be introduced. To have the theoretical foundation to comprehend combinatorial games, it starts explaining the fundamental concepts of game theory. Suddenly, the theorems that allow drawing the main conclusions about the game will be exposed. The theory of combinatorial games allows understanding mathematical concepts and the basics to face the many challenges and restrictions that arise when trying to solve them.

  The objectives and the various games, that make up the SMaILEApp focusing on the activities that use AI to play OKPLAY, will be deepen. Next, the rules of OKPLAY and a comparison with other similar tile placement games, such as the well-known Tic-Tac-Toe and Four-in-a-row, are exposed.

- The fourth chapter OKPLAY Game Study is an application of the contents previously studied in the previous chapter on OKPLAY. The theorems above illustrate the existence of a winning strategy from the start of the game. There will be demonstrated why it is impractical to explore the graph of all feasible moves and why an alternative approach is required to be able to play automatically.

  Following, the difficulties in clearly defining this strategy by studying the reduced cases of the original game are shown. Finally, a rough estimation of the possible placements of the game defines its complexity.

- The fifth chapter AI Background provides all the AI literature used in the next chapter to study the two ML models, GAN and RL. The first basic concept used in model development is the MCTS algorithm due to its efficient and fast export capability.

  Then, the explanation of the RL and its general operation show the complexity and the phases that make up the ALPHAGO ZERO model. Next, the discussion about the GAN and its fundamentals explains how the PIX2PIX model's foundations and how the MAGNUSGAN model employed them.

- The sixth chapter AI Models describes the structure, operation, and results of the two models studied: OKPLAYGAN AND ALPHAOKPLAY. OKPLAYGAN is the model based on the GAN network based on the PIX2PIX model, while ALPHAOKPLAY is the model derived from ALPHAGO ZERO. For each model, various studies modify the input data to adapt it to different contexts of OKPLAY.

- The seventh chapter of Educational Aspects describes how the gamification aspects of tasks allow better learning and is an approach that is becoming increasingly widespread for its effectiveness. Next, the RL model is adapted to the SMaILEApp in the park and station games. Then, how these tasks have an educational impact on students are treated. The final comment shows the research status of the SMaILE project.

- In the end, a conclusion about the studied models' results and their possible future uses indicate how they have an educational impact within the SMaILEApp.

Chapter 2

SMaILEApp

This chapter explains the concept underlying the application created to carry out the SMaILE project. The activity of building your own city is the application's core, through which you can access the following tasks. In the end, there will be an overview of its design and implementation. The study conducted for this thesis is a component of the SMaILE (Simple Methods for Artificial Intelligence Learning and Education) project, which aims to enhance the educational system in light of the usage of AI by raising students' knowledge of its potential applications.

Figure 2.1: SMaILE Project Logo[2]

2.1 Application Concept

The SMaILEApp, a structured application to conduct numerous gaming activities to improve the boys' knowledge and applicability in game theory and ML fields, has been created to help achieve the project purpose. The concept used in the application is "Learning by doing", which is attempting to carry out the task after

providing a brief explanation of the game's fundamental rules to solve the issue at hand without first having a detailed and impractical interpretation of it. After achieving the activity's goal, the idea is formalized in a brief notion to challenge the users to increase their awareness of the new perspective to face future tasks.

When compared to passively listening to long explanations and trying to reproduce what was understood, frequently incompletely, this approach is more effective because it allows for more hands-on experience. The result is that the person becomes more interested and motivated to carry out the tasks, which makes them more likely to be curious about how things went and to desire to improve their performance.

These AI are trained to play the board game OKPLAY, somewhat similar to the popular game Four-In-A-Row, and to generate specific AI, called bots, to play under particular conditions.

2.2 Story

The user will assume the role of mayor and try to increase the city's sustainability and liveability levels. The futuristic smart city is the cornerstone of SMaILEApp. The user will need to participate in several mini-games based on Game Theory and Machine Learning to get these results. The user will decide how to build the city, and this will be the initial mini-game and the structural component of the entire metropolis.

Figure 2.2: An illustration of an artificially generated smart city

In this CSP, or constraint satisfaction problem, the game called City Building, the player's first goal is to arrange the tiles to achieve the best possible sustainability score. To win, players must arrange hexagonal tiles on a 2D map optimally. The

user starts the game with three cards: a school, a house, and a park. Each one has a color: blue, red, and green, and the numbers are 2, 15, and 5. The weighted average of the points determined by the proximity of particular tiles to other tiles is the calculation of the scores, which will be updated automatically during the placement:

1. The value of a home next to a school is ten points.
2. A residence located one away from a school is worth three points.
3. Fifteen points are awarded for a home next to a park.
4. A residence located one away from a school is worth five points.

The user will then need to make an effort to develop numerous rules to apply for an automated solution that automatically constructs the city. Mathematical expressions between the different cell types can be used as rules to direct the solver on what to do first, such as trying to place as many housing tiles as you can at a distance of one from the school. Once the player follows the rules, the city will be constructed in this style, although the player can update and modernize it when he wants.

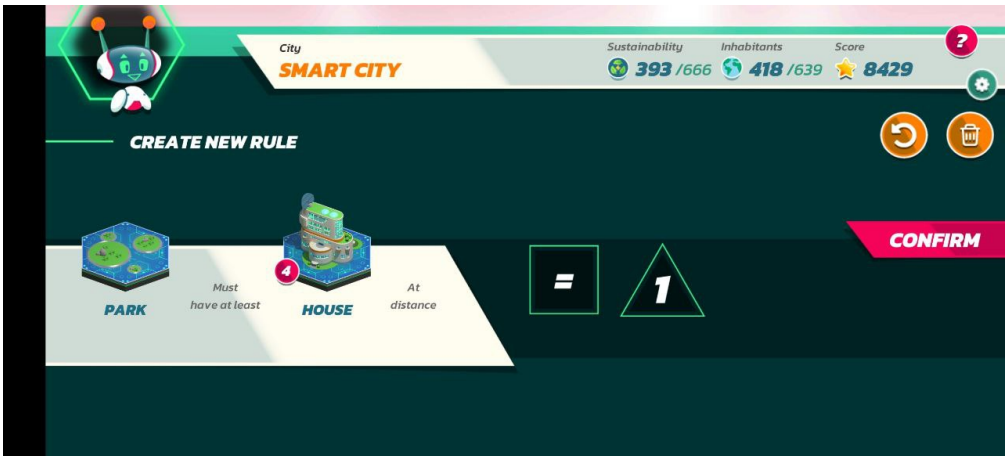


Figure 2.3: An illustration of an artificially generated smart city

The user can now move between the different cells on the city map in 3D and select a cell to access the associated activity or mini-game displayed in a 2D view. After completing challenges at the lower level, users can raise their rating for the city's sustainability and population. The aggregate ranking increases with the new score. Each mini-game has a concept of game theory or machine learning to teach the user. Increasing your overall score will unlock new tiles that contain fresh mini-games and challenges that highlight a characteristic of the cell they represent.

Sust. level	Type	Learning outcomes	Game Mechanics
0	City Building	Constraint satisfaction problem	Master plan creation for the autonomous city builder
1	Park	Adversarial search	Decide the land destination and maintain it
1	Hospital	Search	Move the patients on the right spot
1	City Hall	Theoretical contents	Analyse the plays and read about the theoretical concepts of each game
2	Metro station	Machine Learning	Multiplayer interaction with a trained avatar
2	Supermarket	Classification	Organize products on the shelves
3	Cinema	learning linguistic requirement	write an original movie plot to advertise the city
3	School	Reinforcement Learning	Teach to a school bus how to properly collect kids and reach the school

Table 2.1: List of actual city tile and relative activities

People can examine nearby cities developed by other users to compare their performance and the organization. An AI that appears as a humanoid robot will be present for all of the user's activities and serve as an assistant, accompanying and explaining any novelties that may occur and offering help and guidance when necessary at the user's option. The various mini-games are real-world issues to be solved more quickly and effectively by applying AI and game theory concepts.

2.3 Technology

The application is developed for smartphones and tablets to allow users interaction through touch screens. The application is divided into two sections: the graphics component and the backend. The MelaZeta company designed the graphics employing as base environment the multiplatform graphics engine Unity. The

SMaILE project's researcher developed the backend component, which includes the application's logic and the various AI employed. The city plan is initially presented in 2D to perform the phase of placing city tiles, then in 3D to enable users navigation inside the city. The numerous mini-games are all 2D and have unique Python frameworks, including those involving AI.

Utilizing the tool, you can save game and player statistics for future instructional use. The principles of Game Theory and Machine Learning are presented in a straightforward manner using some combinatorial games, some of which include AI as an opponent. It proved necessary to develop a bot with the game's basic skills to act as an opponent for the OKPLAY game. The construction of the bot involved the parallel study of two different models of ML to evaluate their performance in dealing with combinatorial games.

Chapter 3

Combinational Games

In this chapter, combinatorial games and the theory and classification needed to study the OKPLAY game will be introduced. To have the theoretical foundation to comprehend combinatorial games, it starts explaining the fundamental concepts of game theory. Suddenly, the theorems that allow drawing the main conclusions about the game will be exposed.

The theory of combinatorial games allows understanding mathematical concepts and the basics to face the many challenges and restrictions that arise when trying to solve them. The objectives and the various games, that make up the SMaILEApp focusing on the activities that use AI to play OKPLAY, will be deepen. Next, the rules of OKPLAY and a comparison with other similar tile placement games, such as the well-known Tic-Tac-Toe and Four-in-a-row, are exposed.

We will strive to preserve the same formalism and language for consistency. The literature we will discuss is primarily drawn from "Game Theory, Alive" [3], and "Combinatorial games: from theoretical solving to AI algorithms"[4].

3.1 Introduction to Combinatorial Game Theory

Let's begin by defining some pertinent game terms in a more formal manner: Position refers to the game's present state of play, including all relevant information, such as how the chess pieces are arranged on the board and who is taking turns at what time; The term "legal move" refers to a player's action of moving from one position to another while adhering to the game's rules. For instance, in chess, the tower can move in any position that is not occupied by other pieces either horizontally or vertically, and it is also permitted to use any free cells that are available to it;

A two-player game can be viewed as a collection of positions and a set of legal movements to change places. The execution of a legal move by a player in turn

constitutes the player's move. Some positions are referred to as terminals, which means that the game has reached the termination condition and one player has won the round or a draw has been declared. Here is how we define a combinatorial game: It is a two-player game with the following characteristics: only two players generally referred to as "Left" and "Right" play the game, taking turns alternately; There is no hidden information or ambiguous information in the game; each player is fully informed about the current state of the game and every move that might be made as a result;

The game is deterministic, meaning that there isn't any randomness-generating technique like using dice or coins; In every situation, the game's rules are set up to produce a winner; The game's final turn determines the winner since it results in the completion of the victory condition and prevents the adversary from taking another turn; The game's determinism assures that the decisions made by the player alone determine the moves that will be made, as opposed to being influenced by outside or other circumstances.

With full knowledge of the game, you may make judgments knowing exactly what the probable outcomes will be even for your subsequent movements. Even though some games do not adhere to this property, they are nonetheless treated as combinatorial games given the simplification that comes with it because the certainty that the game always comes to a conclusion ensures that the possible outcomes are the victory of Left, the victory of Right, and, by definition, never a draw.

Some games are referred to as "loopy" when players can repeatedly explore the same positions inside a single game at the risk of the game never ending, as in the case of chess, and this results in draw situations. Other specific examples of games that are similar to combinatorial games but differ from them include those in which the determination of a score, rather than the last move, determines the winner. These games, like Othello, address the issue of recurring positions.

The combination of these traits enables us to theorize that the player will always select the optimal move from the state of the game at hand, being able to forecast all subsequent movements, with the goal of securing their own victory. Combinatorial games can be seen as decision trees with nodes made up of game positions and offspring made up of all positions that can be reached by a single move from the current position. The tree begins with an initial situation and grows backward until it reaches the leaf nodes, which stand in for the game's final situations.

3.2 Theorems and Case Study Applications

Definition 1 *Combinatorial Game "A combinatorial game with a position set X is said to be progressively bounded if, for every starting position $x \in X$, there is a*

finite bound on the number of moves until the game terminates. Let B(x) be the maximum number of moves from x to a terminal position[3]."

In this instance, we will examine a particular type of combinatorial game known as a gradually bounded game, where B(x) is defined as the maximum number of movements that may be made from x to a terminal position when a collection of X positions is taken into account. The existence of a game's end is ensured by these combinatorial games that require a finite number of moves to complete.

Definition 2 *Impartial Game "An impartial combinatorial game has two players and a set of possible positions. To make a move is to take the game from one position to another. More formally, a move is an ordered pair of positions. A terminal position is one from which there are no legal moves. For every nonterminal position, there is a set of legal moves, the same for both players. Under normal play, the player who moves to a terminal position wins[3]."*

When discussing impartisan games, such as Nim, a key distinction between combinatorial games is whether winning positions and legal plays are attainable for both sides. As a result, players have a set of winning plays and positions that can be attained in numerous ways. Hex is just one example of how all other games are regarded as games.

It is feasible to think of the games as oriented graphs, where the nodes represent the game's positions and the oriented arcs represent the legal ways to move between further nodes. When there are cycles between the positions or if they may be reached from various beginning positions, this form is more useful. You can find patterns in games to route in a specific stream to achieve a particular outcome.

Definition 3 *Strategy "A strategy for a player is a function that assigns a legal move to each nonterminal position. A winning strategy from a position x is a strategy that, starting from x, is guaranteed to result in a win for that player in a finite number of steps[3]."*

We define strategy as a means of indicating to every non-terminal position which legal move must be successful in order to produce the desired outcome. In general, winning strategy refers to a set of steps leading from a generalised x position to a predetermined outcome for the current player. We provide a more formal definition of these ideas in the context of games using the sets N and P, where N stands for "next" and refers to the set of positions that ensure the victory of the first player, and P stands for "previous." Therefore, it is speculation in all cases where the second player has a winning guarantee.

Define:

- $\mathbf{N} = \{x \in \mathbb{N} : \text{the first (“next”) player can ensure a win}\}$

- $\mathbf{P} = \{x \in \mathbb{P} : \text{the second (“previous”) player can ensure a win}\}$

Let N_i (or P_i) represent the set of positions from which the player moving up (or down) may ensure victory in at most i moves (of either player). Take note that $P_0 \subseteq P_1 \subseteq P_2 \subseteq ...$, and $N_1 \subseteq N_2 \subseteq ...$ are all connected[3].

So:

$$\mathbf{N} = \bigcup_{i \geq 1} \mathbf{N}_i \text{ and } \mathbf{P} = \bigcup_{i \geq 1} \mathbf{P}_i$$

Where:

- $\mathbf{P_1} := \mathbf{P_0} :=$ {terminal positions }
- $\mathbf{N_{i+1}} :=$ { positions x for which there is a move leading to $\mathbf{P_i}$ }
- $\mathbf{P_{i+1}} :=$ { positions y such that each move leads to $\mathbf{N_i}$ }

The final positions that lead to success are contained inside, along with the intermediate locations that enable you to reach the final positions either directly or indirectly.

Theorem 1 *"In a progressively bounded impartial combinatorial game under normal play 2, all positions lie in $N \bigcup P$. Thus, from any initial position, one of the players has a winning strategy[3]."*

It allows us to state that, in a gradually constrained impartial combinatorial game, all possible positions are contained in the set denoted by N U P, guaranteeing that at least one player will have a winning strategy. This occurs because every move in the game belongs to either N or P, and since this also holds true for the initial position, it is possible to tell right away who has a winning strategy.

Definition 4 *Partisan Game "A combinatorial game in which the legal moves in some positions, or the sets of winning positions, differ for the two players, are called partisan.[3]."*

Theorem 2 *"In any progressively bounded combinatorial game with no ties allowed, one of the players has a winning strategy which depends only upon the current state of the game[3]."*

It assures us that at least one player has a winning strategy and depends solely on the state of the game, then by the current position of the game, and by the player who has to play the move in a progressively bounded combinatorial game with no probability of a tie. The number of movements required to win the game from a

specific position is now determined by the game's current state, $s = (x, i)$, where x stands for the position and $i \in \{\text{I, II}\}$ stands for the player who moves next. The greatest number of moves required to finish the game from the state is now denoted by $B(x, i)$. We now have the assurance that, in games of this nature, you can identify a player who can prevail from the outset.

Lemma 1 *"In a game with a finite position set, if the players cannot move to repeat a previous game state, then the game is progressively bounded[3]."*

It enables the return of a looping game to a gradually bounded combinatorial game with the restriction added not to repeat a previously turned move, and then the application of all category-related considerations. Let's use the game Nim as an example, which has been thoroughly researched for both its simplicity and the conclusions it enables. The game of Nim concludes when all of the pieces have been played and no player may take any more turns removing them. There are initially two or more stacks of pieces in defined numbers on the board.

Because both players can make identical actions and must achieve the same terminal locations to prevent the other player from receiving any pieces, this game is portrayed as being impartial. Specifically, in the scenario of two stacks, the winning tactic can be traced back to getting the opposing player to the point where both stacks have the same number of pieces so that, following the opposing player's move, they can spread the stacks and ensure that they are the last player to empty the last stack.

Unless the game starts with stacks that have the same amount of balls, in which case the starting position of the game is considered to belong to the N set, this approach is typically feasible by the first player who succeeds in balancing the stacks in his first round.

The objective of both players in the partisan game Hex is to connect their respective sides of the board with a chain of hexagons by turning a hexagon on the player board. Hex is played on a diamond-shaped board with hexagonal tiles, and each opposite side of the board is made up of a row of the same player's tiles.

Hex, as previously stated, is a partisan game since the terminal positions of the two players disagree, and a tie cannot be achieved because completely completing the board provides a link, or more precisely, a single complete connection, between two of the opposing walls. This game is not fully solved because there are several solutions depending on the size of the board, and the final winning strategies were only found up to a specific size because some are too complex to be formalised by humans.

3.3 Combinatorial Games in Practise

Although winning tactics for combinatorial games are theoretically obvious, it is not always straightforward to explain what these strategies could be. For instance, it is well known that the first player in the game of chess has a winning strategy from the start but has not yet been able to specify what it is due to the difficulty of exploring the graph of the game's legal movements.

A game's difficulty can be determined by determining the winning strategy from the outset. The following criteria make this search more difficult: The number of positions in the game is frequently determined by the size of the game grid, which determines how many positions are at worst to be explored; the branching factor, which is the average number of licit moves per position being a fixed value that tends, in general, to increase from the beginning of the game; if there are too many terminal positions, it is impossible to start looking for a backward strategy at the game's end; cycles make navigation difficult and make it necessary to keep track of previously visited areas in order to avoid repeating;

if the game is partisan because there are more paths to investigate because both players' endings are different and the tree to be examined is not symmetrical as in the case of partisan; According to the outcomes, there is a distinction in the study's gaming literature:

1. Strong solved: a winning strategy is known for every game's starting position;
2. Weakly solved: a winning strategy that assumes the opponent performs optimally is only known for the game's default starting position and is not extended to other conceivable positions;
3. Extremely Weakly Solved: A winning strategy has not yet been determined, although it is known which player has one;

We can say that concepts related to the mathematical formalisation of games have given us some certainty, such as the existence of an a priori winner in a game and a winning strategy, but this is insufficient to prove it, and we require more powerful methods than the straightforward exploration of the decision tree.

3.4 OKPLAY Game Introduction

OKPLAY will serve as our benchmark game for training our machine learning model. This is a placement game that can be played by two to four players; from this point on, we will only address the situation involving two players. The playing field of the game is defined by no bounds and is made up of a theoretically limitless grid of square tiles. The first Blue player and the second Red player both have 21

tiles. Each player takes turns placing one of his own tiles while keeping its side adjacent to that of another tile already on the playing field after the initial player places his first tile in the centre of the board. Making a row of five tiles in their

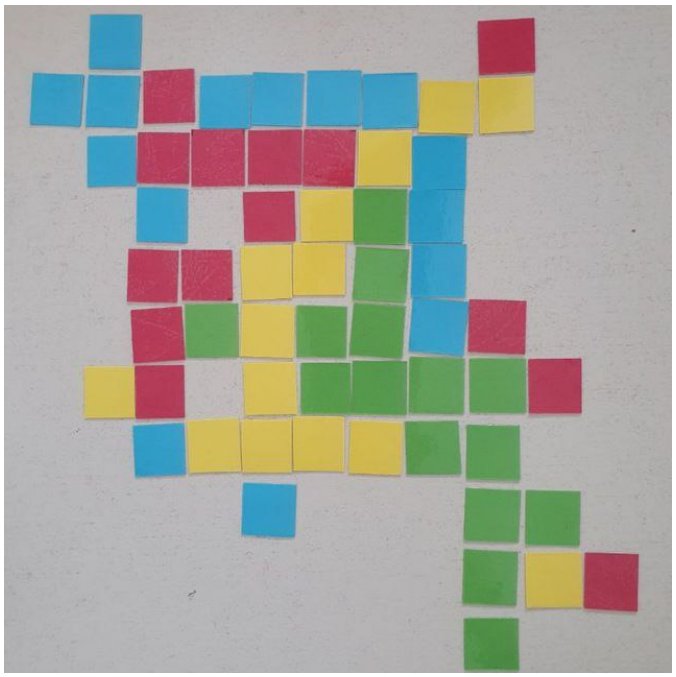


Figure 3.1: example of OKPLAY game placement

own colour that is either horizontal, vertical, or diagonal is the aim of both players. When both players have used up their 21 tiles, they must move one of the ones already on the board rather than adding more. The game can be seen as a series of two play stages. Players place tiles from their reserves in Step 1 and only take into account what the card placement entails. Phase 2 starts when both players have used all of their cards and must move the cards they've already positioned.

The final step significantly increases the game's complexity since, in addition to taking into account the tiles that have been set, the strategy must now take into consideration the empty cells. The game's objective and proximity restriction are comparable to those of the more well-known game Four-in-a-row, which is why the rules are extremely straightforward and there are some similarities. As a result, it has a high level of difficulty that can change significantly depending on the quantity of tiles, whether edges are present, or how many tiles are required to satisfy the victory condition.

The victor of a m, n, k-game is the one who first lines up k stones of their own colour in a row, whether it be horizontally, vertically, or diagonally. Two players take turns placing a stone of their colour on an m-by-n board in this abstract board

game. As a result, the 3,3,3-game for tic-tac-toe and the 15,15,5-game for free-style gomoku. On an m-by-n board, a m, n, k-game is also known as a k-in-a-row game.

3.5 Relations with Four-In-A-Row

All references to the game Four-in-a-Row have been taken from the paper "Solving Strong and Weak 4-in-a-Row"[5]. In the well-known game Four-in-a-row, Black and White players compete to be the first to line up four successive stones of the same colour in a straight line on a specific rectangular board. Horizontal, vertical, or diagonal lines might all be winners. By tradition, black starts. The strong positional game 4-in-a-Row is an example of a larger class of these games.

We concentrate on the positional game winning monotonicity property. It is widely accepted that if a game of "K-in-a-Row" is successful on one board, it will also succeed on any larger boards. This is considered a monotonous victory. As a result, any smaller board will also result in a draw in a k-in-a-row game if it draws on some board. The monotonicity property holds by definition for weak positional games. However, the monotonicity property for the robust variant of k-in-a-Row has never been properly established.On arbitrary graphs, when expanding the winning set might turn a winning game into a draw, it is unquestionably false. The Extra Set Paradox is the name given to this phenomenon.

In addition to demonstrating that the 4x9 board is the smallest 4xn board on which Black wins, we were able to replicate in a split second that the 5x6 board is actually the smallest board on which Black can win. Next, boards 5x5 (the smallest) and 4x7, a weak 4-in-a-row is already a black victory (the smallest 4xn). These victories are simpler since Black cannot be stopped by White using threats alone; instead, White can only do it by claiming at least one stone from each group. Since these victories are boring by definition, the weak 4-in-a-row is no longer an issue.

Additionally, we looked at the issue raised by the Paradox Extra Set, which shows that the wins in the strong version are not always monotonous (at least not tried so far, although they probably are, highlighted by all the available results). This means that even if a larger scoreboard includes a 5x6 or 4x9 scoreboard, we may not conclude that it is a black victory by solving the 5x6 and 4x9 scoreboards.

The solution for the 5x6 board where Black can only claim squares on a central board and White can claim squares on the entire board, the central board having two-sided edges. The findings indicated that despite. White having an advantage of three or more measures, Black still prevails in the game. As a result, black's triumph in games 5 and 6 is boring.

3.6 Relations with Tic-Tac-Toe

The paper "Tic-Tac-Toe on a Finite Plane"[6] contains all references to the game of tic-tac-toe.

Tic-tac-toe can be played by anyone. A player wins the game if she places n of her markers on a n x n board either horizontally, vertically, or diagonally before her opponent can do the same. What if there were more ways to win while maintaining the same game rules? For ease of use, a winning line will be referred to as any arrangement of n marks that results in a victory, whether or not it seems straight. In a zero-sum game, one player's loss results in the other player's gain (s). Tic-tac-toe is a two-player, zero-sum game that takes place on a 3 x 3 board.

Players take turns marking one open cell with an X or an O. We'll use the abbreviations X and O for the first player and assume that X always initiates motion. The first person to place three identical marks on a line wins. When a game is over and nobody has won, it is a draw. A game with perfect information is tic-tac-toe, in which every decision made by each player is known to the other player.

A strategy is an algorithm that determines a player's subsequent move based on the state of the board at the time. A winning tactic for X, for instance, is one that will undoubtedly result in victory for him. There is a best way to play traditional tic-tac-toe, but since each player can ensure that the other player cannot win, there is no winning strategy. It can be said that both players in this situation have a drawing strategy, or an algorithm that results in a draw.

The concept of rationality, which states that each player will make a decision that results in a state that has the greatest utility for that person, is a widely accepted assumption in game theory that both parties are competent and act appropriately. The definition of tic-tac-toe on an order-n plane is as follows.

Any area of the plane that hasn't been labelled yet is marked by X and O in turn.The winner of the game is the first person to collect every point on a line. If all points are used up and neither player has finished a line, the game is a draw. We will designate X's first move as Xi, his second as X2, and so on to indicate the order of play. O also has designated motions.

Theorem 3 *DRAW THEOREM FOR* Π_n *"O can force a draw on every projective plane of order n with n > 3[6]."*

Assume you are taking on the role of O on one of the countless planes for which there is a drawing technique. The algorithm described in the preceding section may ensure a draw, but in order to select the position with the greatest weight at each move, it necessitates calculations of Eulerian proportion.

Chapter 4

OKPLAY Game Study

This chapter is an application of the contents previously studied in the previous chapter on OKPLAY. The theorems above illustrate the existence of a winning strategy from the start of the game. There will be demonstrated why it is impractical to explore the graph of all feasible moves and why an alternative approach is required to be able to play automatically.

Following, the difficulties in clearly defining this strategy by studying the reduced cases of the original game are shown. Finally, a rough estimation of the possible placements of the game defines its complexity.

4.1 Game Theory on OKPLAY

The original game can be categorised as a combinatorial gradually bounded game theoretically since it is deterministic, has no random elements, and has comprehensive information that can anticipate all movements from a starting position in advance. Since the game continues after the change in rules and continues until a player forms a row of five tiles, we may be sure that the game will end eventually. There is also a draw condition.

It should be mentioned that OKPLAY might be viewed as a looping game because in phase 2 you could move the tiles back to a previously visited place, but you can prevent this by inserting the rule of not going back to previously examined positions. The genuine actions for both sides are the same, but the winning positions are different, making it a partisan game as well.

4.1.1 Theorems Basic

This game's definition allows us to think about applying the following theorems, which we know to be true: Theorem 2 : existence of a winning strategy for a player

depends on the state of game turn and position.

Lemma 1 : the game can apply the lemma because since the game can not get to a draw but only to the victory of a player and depends on the state of the game then all the positions are traceable to N or P and then there is the winning strategy from the beginning of the game for a player definitely we can use this theorem. We can derive that from the start of the game, the Blue has a winning strategy.

4.2 Case Studies

We will now discuss game variations that have the potential to undermine the criteria for the combinatorial game that we have just specified for OKPLAY but might not be helpful in some very specific circumstances due to other factors that we could take into account. For these reasons, as well as to be able to formalise it from the computer's perspective in a finite way, it has become important to examine it using certain constraints, such as gaming edges. Without them, the game would be considerably more intricate than its affiliate.

Now we will define some elements that we will use in order to draw some conclusions in more limited and simple cases. We define as L the side of the square game grid, W will be the number of tiles needed to meet the condition of victory and T the number of tiles available to each player.

For simplicity and to enable later models to pass the idea that the centre exists and is equivalent to the move of the first player, we will try to maintain the game grid with odd side so that there can be a centre. Now we will look at some cases of basic strategies or situations that lead to the victory of a player in reduced cases of the original game.

4.2.1 W is 2

When W is 2, the first player automatically wins and the winning strategy is obvious: any second move of the Blue will result in victory since the Red can only place the first tile on the four adjacent sides of the map, all of which are distinct moves and equal to one another. As a result, there will always be three sides adjacent to the first Blue card, whichever will be guaranteed to win. L and T must, of course, be bigger than 2.

This leads us to believe that the first participant in this situation is clearly at an advantage; It takes two rounds for the first player to clearly and unquestionably win.

4.2.2 W is 3

Let's now examine the scenario when W is 3; even in this circumstance, the first player can still quickly achieve a winning position in his fifth round. Additionally, in this scenario, the Red is powerless to do action to advance toward triumph or harm the Blue. The Blue's strategy is to be able to place two of its tiles next to each other while leaving the two ends of this row empty so that the Red can't occupy them. Even if the Red can occupy one of those two sides, the Blue will still have the option of placing his third card in the following round to create the row of three.

The opponent will not have a chance to react and prevent the Blue's triumph if the Blue places his tile as a second move out of alignment with the first tile of the Red. If this plan is ignored, there are other, harder-to-define ways to guarantee the Blue's win that need more turns, but they are avoidable and less clever because they have this obvious possibility.

In this instance, T needs to be at least three and L needs to be at least four. Since the Blue has an overwhelming early advantage over the Red in these instances, it is not necessary to overcome these win conditions in phase 2 of the game by moving the tiles.

4.2.3 W is 4

We must identify additional elements because the intricacy of the situation of W 4 has increased significantly. We start by categorising the purpose of your move into two groups: offensive, in which case you position your tile in an effort to go closer to fulfilling the victory condition, or defensive, in which case you block your opponent from winning.

Even though a move can be both offensive and defensive in the situation where it helps the player of turn and eliminates an opportunity for the opponent, these categories serve as the primary means of defining the turned move's goal. This first distinction is crucial because it may influence a player's decision to play an aggressive strategy to position himself for victory if he is in a tight game, especially if he thinks he has a significant starting advantage.

The opponent is obligated to pursue him as a result, which makes it nearly hard for him to play offensively and prevents him from focusing on winning. There is a winning strategy in the specific situation of OKPLAY with W 4 for the first player, which entails always playing offensive moves from the outset in order to force the opponent to block it. However, this is insufficient because the Blue always has the option to start another strategy in order to generate a position from which the opponent has no way to stop it.

The only advice given to the Blue is to avoid making plays that will result in the Red making two moves in a row that will ensure his victory. In this case, there

is more than one way to win; in fact, the Blue can guide her to a winning scenario in around 18 moves in response to the Red's moves.

The ability of the Red to make many moves that, despite having the same effect, cause quite distinct pathways and responses from the Blue is the reason why the strategy does not correspond with a single path or a few extremely similar ones. These placements, which we might refer to as "crossroads," are what make it so challenging to clearly define a strategy.

We clearly demonstrate various circumstances that put a player in a winning position or need a defensive move: If a player lines up three tiles with the two extremes vacant, that player will win without a doubt because no matter what move the other player makes to block one of the two extremes, there will always be another that will formally permit the row of four tiles. The only situation in which this does not represent a favourable position is when the adversary is in a position to start his row from 4.

In order to finish one of the rows, a player must set a horizontal or vertical row with a diagonal, both of which contain three incomplete tiles each, in addition to at least two more free cells between the two rows. Similar to the previous circumstance, this one is predicated on the idea that the opponent can only completely eliminate one opportunity of winning with a single action.

A player can create a 2 x 2 square, or two vertical rows, if there are at least three free cells, two of which are in a row that really creates at least one row of two that can be continued while the other row is blocked. Because the following move could result in 3 consecutive tiles, you must block the formation of 2 tiles in a row with a defensive move from at least one of the two sides in order to prevent your opponent from winning.

We can claim with assurance that the Blue has a win strategy that is assured from the start of the game, even in the W 4-version. When considering only the first phase of the game, the possibility of Blue winning in the fewest moves is practically impossible for the case W=5, considering how many turns are required for W to 4 to be able to force the opponent to have no chance to overturn the roles of those who chase their victory and those who prevent!

4.2.4 W is 5

Since phase 2 of the game must be introduced by shifting the tiles, the case W to 5 is extremely complex. This is true not just because we know that it is virtually impossible to win with confidence in 21 moves for each player. The goal of Phase 1's strategy is no longer to win by randomly putting the tiles; rather, it is to determine how the tiles should be arranged to benefit Phase 2's game.

In addition to the methods already mentioned, the objective can also be accomplished in detail by compelling the adversary to remove a tile, which would ensure

the other player's victory on the following move. As a result, it becomes impossible to define a strategy and formalise it in a way that is clear to everyone. It highlights the necessity to consider not just the effects of the card played, but also the effects of freeing a particular cell, as doing so could provide the opposition the chance to get the upper hand and get closer to winning the game.

Similar situations exist to specify a player's win position or to force the opposition to take a defensive action, such as in cases W to 4. The arrangement of 4 tiles in a row with 2 free ends and a cross of 2 rows, one diagonal and the other horizontal or vertical, both with at least 3 tiles in a row and at least 3 free ends of the various rows, are generally the most important requirements.

4.2.5 Positions Estimation

Let's attempt to make a broad approximation of the game's potential situations by overestimating them: We assume both lawful positions, such as simultaneous Blue and Red victories or multiple victories, and that all positions are still used even though they would be in terminal scenarios. We consider reasonable estimations for the W version up to 5.

In this manner, we will calculate the same position obtained from various paths at least several times for phase 1 of the game in order to obtain an overestimation because reductions are difficult to calculate when equivalent positions such as rotations, translations, mirrors, or combinations of these variations are taken into account. Every time the game begins, the Blue player will play a card in the middle, after which their opponent will have four movements to help them. Since they are identical because each is the rotation of the other, the Red's first move is actually 1 and not 4, as opposed to 4. The Blue will then have six different moves to make, and the number of moves will change depending on where he places the game's third tile.

In particular, if we maintained a line shape, the available moves would increase to 8, if we made an angle, they would decrease to 7, and if we formed a square in the following move, they would increase to 8. In particular, the situation of tiles arranged in a line as potential moves to insert them from the two sides of the line as well as the ends of the line itself, can be taken into consideration in order to generalise the cases. This configuration is limited by the size of the grid where we really learn the game and is of little interest from a play perspective.

Widening the line by placing it on the sides we can begin to notice that a kind of rectangle is formed that, even if it is perfectly regular, allows a number of moves slightly overestimated to 4 times the square root of the tile number. This estimation calculation can be used until all 42 game tiles have been used, at which point we would have a very large number of placements.

The manufacturer should take into account that the formula is only generalised

and that it is a number with difficult representation in order to acquire the overall estimate of the positions for phase 1 of the game.

$$children\_estimation(N) = 4 * \sqrt{N} \tag{4.1}$$

$$Phase1\_position\_estimation = \prod_{n=1}^{42} children\_estimation(n) \tag{4.2}$$

The estimate for the tile placements in phase 2 of the game is equivalent to performing repeated calculations of permutations while using a finished side L grid, of which 13 should be sufficient and in any case a very high value, while also taking into account 21 T tiles for each player, b and r, as elements and the difference between the players' tiles and the available grid cells as additional elements to be taken into account as empty cells in the permutation, v = L^2 - 2*T. L^2 clearly provides n, which is the total number of active cells. Phase 2 alone yields an enormous result that strongly supports the idea that all alternatives should be explored using a tree or graph structure that cannot be readily retained in memory.

$$Phase2\_position\_estimation = \frac{n!}{b! * r! * v!} \tag{4.3}$$

Placed tiles	possible positions
1	4
2	6
3	7/8
4	8/9/10
5	9/11/12

Table 4.1: Early estimates of the children's positions

Chapter 5

AI Background

5.1 AI models Introduction

This chapter provides all the AI literature used in the next chapter to study the two ML models, GAN and RL. The first basic concept used in model development is the MCTS algorithm due to its efficient and fast export capability. Then, the explanation of the RL and its general operation show the complexity and the phases that make up the ALPHAGO ZERO model.

Next, the discussion about the GAN and its fundamentals explains how the PIX2PIX model's foundations and how the MAGNUSGAN model employed them. Let's begin by giving a brief overview of the 2 models' guiding concepts. The first model is a structure built on two neural networks that compete against each other and is based on the GAN (Generative Adversarial Network) architecture. To be more explicit, we have a discriminative network that must be able to distinguish between elements that are true, that is, elements that are part of the dataset, and elements that are false, that is, elements that were formed by the generative network.

Both parties want to defeat the other, but the generation of false information is always better at doing so than the discriminative, which must be able to discern between the true data and the fake information supplied by the adversary. When these two work together, the Generative network produces fake data that is so convincing that it confuses the Discriminative when it tries to tell the difference between the real and the false. The Generator's fake values might be utilised to create new elements as though they were real.

We want the Generative network to make a smart move based on this concept and the current status of the game.

5.1.1 Game Settings

To make it simpler to read the models, we had to apply certain reductions to the original OKPLAY game. To provide the model a finite space to operate on, the initial alteration was to impose a strict grid on the game. In an endless grid, the game's state would be extremely difficult to express, and for the model to comprehend it, another finished representation of a different sort that was compatible with the initial models would still be needed. In order to validate that the models could grasp at least phase 1 of the game given the increased complexity of switching between game modes, phase 2 of the game was then deleted. The model might not have been able to grasp this complexity well, which would have made even phase 1 learning very difficult. A set of games with the objective of arranging 5 tiles in a row were instantly employed as a dataset due to time and resource constraints.

5.1.2 MCTS

The fundamental ideas of MCTS that we shall discuss were drawn from the paper, "Combinatorial games: from theoretical solving to AI algorithms" [4].

The Monte Carlo Tree Search (MCTS) is a heuristic search algorithm that enables you to browse decision trees without necessarily following all of the pathways that appear to be more promising than others. This technique runs random simulations and makes decisions based on the results of all the simulations, as opposed to using traditional static evaluation functions, which are frequently highly time-consuming to compute. The underlying idea is based on the Monte Carlo method, which involves performing a random sample from among the several workable solutions to deterministic problems that are extremely challenging to solve completely or using other techniques.

MCTS is the outcome of applying the Monte Carlo method to the MinMax algorithm to investigate decision trees using a logic of maximisation of the goal to be attained and minimization of the goal to be avoided, performing random simulations each time beginning from the point where a result is desired. Playing games that can be represented by trees is a breeze with this tool. The algorithm simulates randomly for a predetermined amount of times, creating an asymmetric tree, and then evaluates the best option in terms of the player's victory.

This makes it possible to use the algorithm without the requirement for highly advanced knowledge of the subject matter because all it needs to know is the alternatives that might exist at a specific point and whether it is in a final state or not.

MCTS Phases

The algorithm's 4 steps are as follows:

1. selection: a child node L is formed from the root node R, which represents the current state, starting from the root node R. L is positioned by performing an unexplored legal move from the preceding node.
2. expansion: from node L, one or more child nodes C are formed if it does not describe a win, loss, or a tie unambiguously;
3. simulation: starting at node C, a random game is played, in which players' actions are chosen at random up until the terminal node;
4. backpropagation: Counts are updated from node C and its ancestors to the root node R when the outcome is reached via backpropagation;

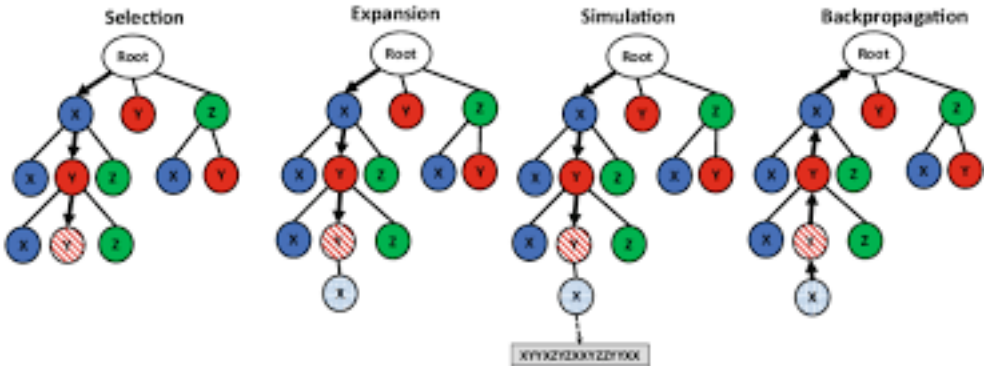


Figure 5.1: example of MCTS execution[7]

The counts of how many simulations have resulted in a success and the number of attempts made to pass from that particular node are the information that is retained and updated by the different nodes of the tree during the backpropagation phase. One of the benefits of this method is how quickly we can choose our next move by selecting it at random; this enables the single iteration to proceed quickly. The effective number of simulations depends on the content to which the algorithm is applied, particularly by the children that can be generated at various levels and the frequency of encountering terminal nodes in explorations. By performing many simulations, you can determine in a meaningful way a holistic understanding of the move.

UCT

To enhance this algorithm's performance or address the way it behaves at specific points, additional enhancements or factors can be added. The UCT (Upper

Confidence bound for Tree) is an enhancement to the MCTS that affects the choice of the node during the expansion phase by favouring the one that maximises a specific number of options for one of the two players. The UCB (Upper Confidence Bound) formula, which is implemented while taking into account the status of the single node at the time of decision, is one that is particularly intriguing.

The formula specifically has two key parts that control exploration and exploitation. The first component establishes the exploitation, which has a value that rises when there are more triumphs than there were in the simulations up until the point of execution. The second factor is exploration, that rises with the total number of iterations describes how much the node has been examined.

$$V(v_i) + C * \sqrt{\frac{\ln N}{n_i}} \tag{5.1}$$

In this case, a balance between these two criteria is created because the element that causes the bigger value reflects the approach the MCTS took in selecting the initial node. The C that multiplies the exploration factor and at its growth instructs the algorithm to offer more space to the little-explored nodes is the parameter that establishes this equilibrium.

The MCTS will favour the nodes that have previously achieved the greatest number of victories in particular if the value of exploitation prevails, as this will clarify whether the node is truly a good final option. If the exploration component wins out, the MCTS will attempt to prioritise nodes with a low win rate while allowing more room for nodes that may not have been well examined.

5.1.3 Reinforcement Learning

The justifications for Reinforcement Learning are taken from the article, "A Brief Survey of Deep Reinforcement Learning"[8].

The two main elements of the Reinforcement Learning process are the agent and the environment. The agent is the active component that interacts with the environment and decides how to change it through the action, based on how it is currently. After an interaction with the agent, the environment changes, and the agent analyses this change to decide what to do next after receiving a reward to evaluate the action a taken. The state s, a discrete representation, is how the agent evaluates the environment. The agent analyzes the status to create its own policies that specify how it will decide what to do next time.

The agent-generated policy π is the rule that should be used to decide which action to do in response to the state received. It can be conceptualised as a function that ties the state to the actions. All potential paths of action must be mapped in accordance with the state and the environment. The reward r is an evaluation of the new status that was achieved by the action on the previous state and provides

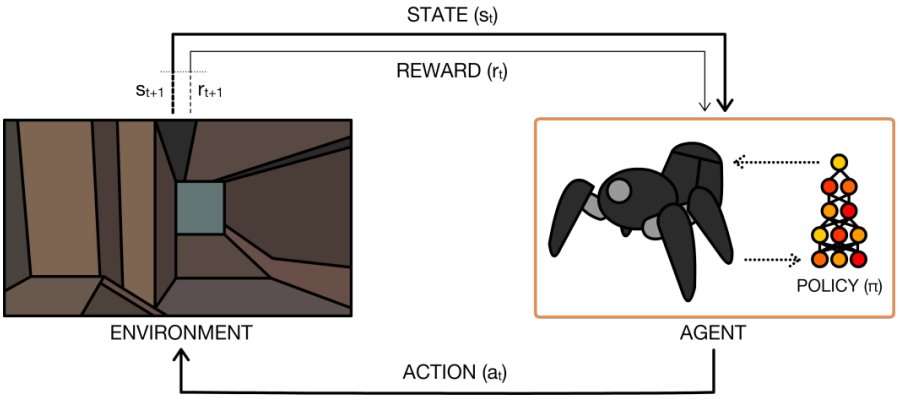


Figure 5.2: Illustration of Reinforcement learning the fundamentals

for an assessment of the action's decision. The agent's goal is to maximise the reward received from his various actions on the many states that it has engaged.

The fundamental idea behind this approach is trial-and-error, whereby the agent gradually learns how to interact with the environment through the reward, which is a judgment of the action he has taken on the environment changes. When we employ deep neural networks to approximate any of the value function, $v(s; v)$ or $q(s, a; q)$, policy $\pi(a|s; a)$ of reinforcement learning, we achieve deep reinforcement learning approaches for state transition function and reward function.

5.2 ALPHAGO ZERO

The ALPHAGO ZERO Model's functioning is based on the book, "Mastering the game of Go without human knowledge"[9]. The Google Deepmind model known as Alphago Zero has outperformed previous AI models and Go samples by a wide margin. The model performed exceptionally well on the Go variation on a 19x19 game grid, which is still regarded as an impossible level. Given the relatively lengthy procedures, training this model demands significant computer resources, such as dedicated graphics cards, as well as a lot of time.

5.2.1 State Representation

The model's state structure, which consists of 17 grids rather than a single grid that would reflect the game board, is an intriguing aspect. The composition gathers various data regarding not only the individual move but also the game in which it is involved in order to glean knowledge regarding the match's history.

Each player's tiles are represented by 1 and 0 additional zones that player did not use. There are eight grids for each player that show both the current configuration and the dispositions of the preceding seven positions. The first eight grids are those of the Black player, then the White player's last eight grids, at last there is a grid valued with all 0 or all 1 depending on which player is taking the turn, respectively Black White.

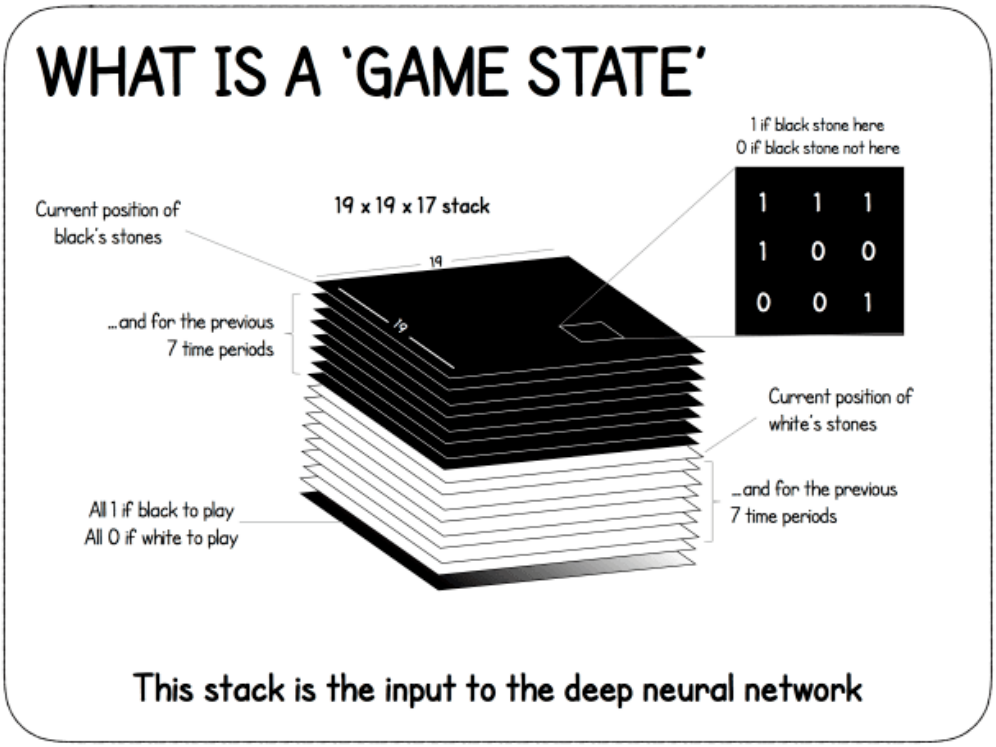


Figure 5.3: AlphaGo Zero State Rapresentation[9]

5.2.2 Network Structure

The massive neural network structure that is being used starts with a convolutional layer, followed by 40 residual layers, and ends with two distinct layers: one for evaluating the policy, or the probabilities that determine whether or not potential actions are meaningful, and one for estimating the state value, or the value that indicates how advantageous the present situation is for the current player. The convolutional block performs 256 3x3 filter calculations within it, does batch normalisation, and finishes with the activation ReLU function. Each residual block is made up of two successive convolutional blocks and a skip connection, which gives the option of adding the input received by the convolutional sequence to the

output of the sequence during training, according to network evaluation.

A convolutional block with only two 1x1 filters makes up the policy block, which is finished with fully connected layers with the same cardinality as the number of mapped actions, or 19x19. The state value block is made up of a single 1x1 filter convolutional block, 2 fully connected layers, a Relu function, and a Tanh activation function to bring the result into the [-1,1] range.

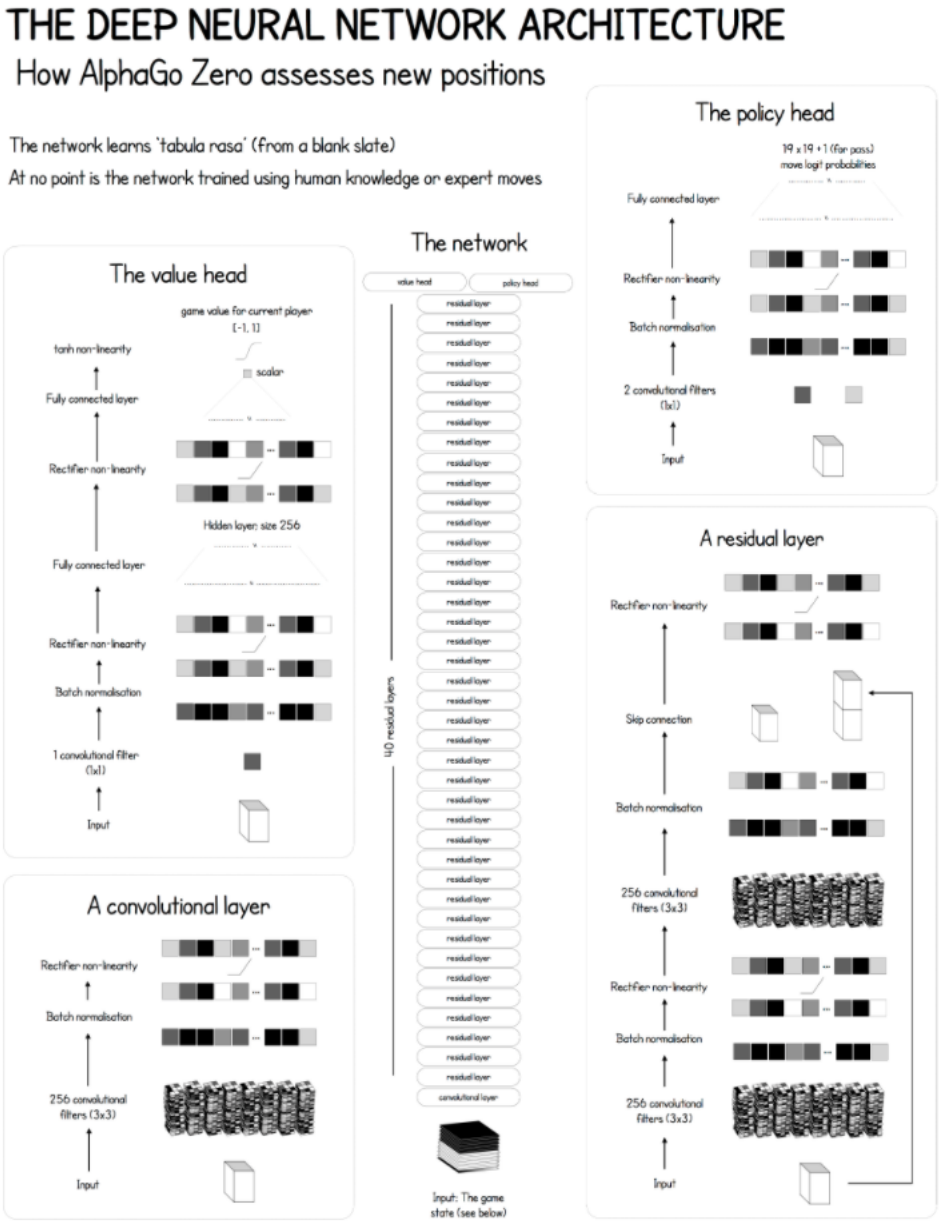


Figure 5.4: AlphaGo Zero Network Structure[8]

5.2.3 Training Phases

The following are the several steps for executing a model training iteration:

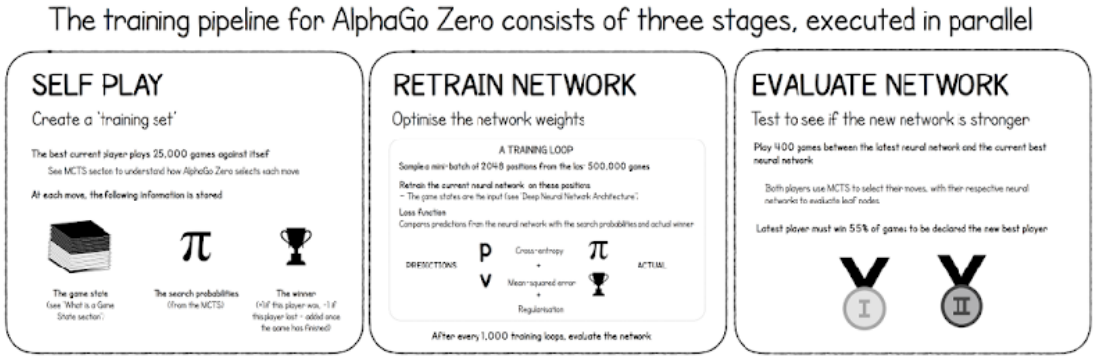


Figure 5.5: AlphaGo Zero Phases[9]

1. SELF-PLAY: The dataset for training the neural network on is built in the first phase by playing 25000 games between the best trained network up to that moment playing versus himself. For a total of 1600 simulations from the present state, each move is produced using the neural network and optimising the policy using the MCTS in order to establish, as previously indicated, an assessment of the move to occur to the current state.

   The end of this phase results in the creation of the dataset for the subsequent phase, which contains the current state of the game for each position played in this phase. The policy of the neural network and the outcomes of each position's match are established.

2. RETRAIN NET: In this phase, the neural network is trained for 100 epochs using mini-batches of 2048 positions from the final 500000 matches performed in the previous Self-Play phases. The neural network is evaluated using a Loss Function with two components.

   The first component results from the Mean-Squared Error between the state value calculated by the network and the final result of the game for each player in turn. The second component results from the Cross-Entropy function between the policy generated by the neural network and the policy generated by the MCTS.

3. EVALUATE NET: The newly trained network is compared to the previous one in the final phase by playing 400 games playing as the first and the second player alternately. If the new version outperforms the old one by a percentage

of at least 55%, the new net will be used for the next iteration; otherwise, the old network will be used again.

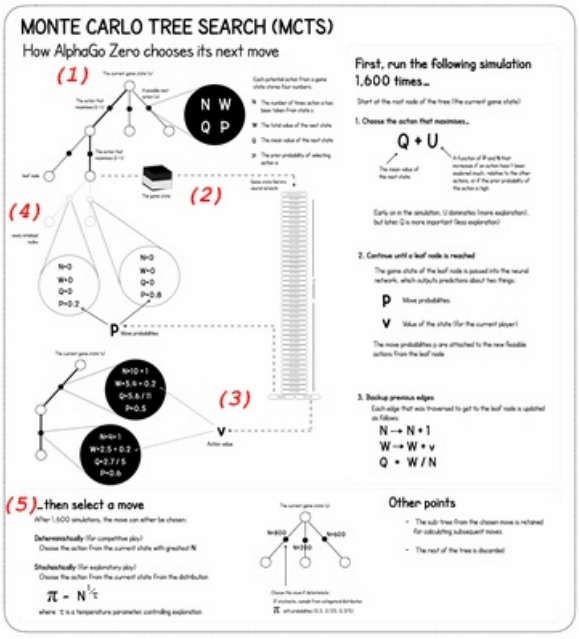


Figure 5.6: AlphaGo Zero MCTS[9]

5.3 Introduction GAN Architecture

The reference model utilised in our game is based on MAGNUSGAN, a GAN model that was trained using the games of the renowned chess player in order to be able to mimic his playing style.

5.3.1 GAN Introduction

The "Unsupervised Representation Learning with Deep Convolutional Generated Adversarial Networks"[10] study served as the source for the GAN Architecture and its guiding principles. The GAN architecture (Generative Adversarial Network) is a model of unsupervised learning, meaning that the dataset's elements lack labels that provide information about the data in order to allow the model to infer the data's identity and informations on its own. It should not be confused with the labelling of the data as real and false in order to distinguish between the two at the training level. The D (Discriminative) network and the G (Generative) network

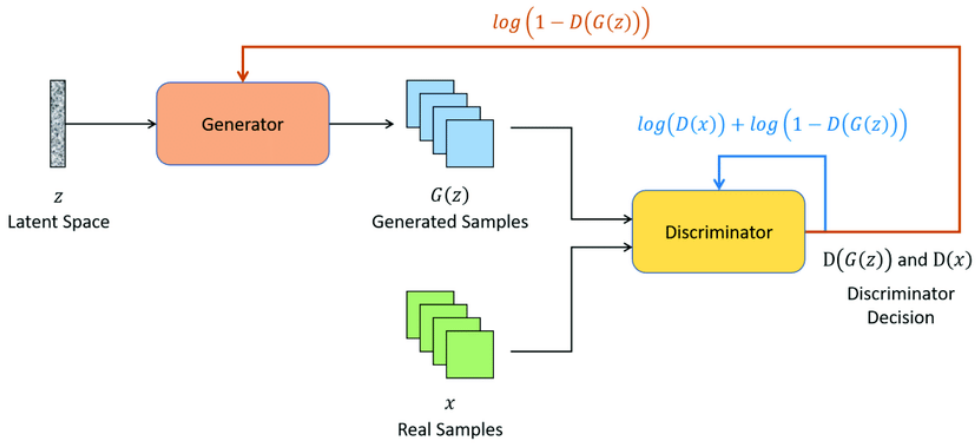


Figure 5.7: Example of GAN architecture[10]

work together during training to increase the G network's capability. The GAN model comprises of these two separate neural networks.

In general, this design has shown to be effective at producing fresh images from a signal of random noise or on a setting of arbitrary values. The G network is used to create new elements, or targets, from a starting sample, and the D network receives the original sample linked by the alternated targets from the true target, from dataset and the false target, formed by G. This requires the D network to determine which of the two is true or false. The sample might begin from a country or predetermined values depending on the scope.

The model is founded on the idea that the G network learns how to improve in accordance with the D network's capacity to identify its fakes, to comprehend what not to consider when improving, and to understand what to concentrate on in order to succeed. The D network gets better based on how well it can detect fakes from real things.

It goes without saying that having originals entails having an initial dataset of samples and their targets. These elements will allow the network D to distinguish the fakes from the original,at first, they will be completely random, and this will provide guidance for the G network on how to get better. since it can generate targets that are on par with the originals used during training by using the right input values.

5.3.2 Loss function

Maximizing the D network's error and decreasing the G network are the two goals of the GAN Loss function.

$$\mathcal{L}_{CGAN}(G, D) = \mathbb{E}_{x,y}\left[\log D(x, y)\right] + \mathbb{E}_{x,z}\left[\log\left(1 - D(x, G(x, z))\right)\right] \qquad (5.2)$$

The G network is minimised to bring the quality of the created fakes to parity with the originals by minimising as much as possible the disparity between the ability of the D network to recognise the fakes. The goal of the D network's maximising is to distinguish actual targets from G fakes by labelling them as such.

$$G^* = \arg\min_G \max_D \mathcal{L}_{CGAN}(G, D) + \lambda \mathcal{L}_{L1}(G) \tag{5.3}$$

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y}\left[||y - G(x, z)||_1\right] \tag{5.4}$$

The G network wants to trick the D network upto thinking that 50% of its objectives are fake in order to produce exceptional results.

Typically, a loss function is applied to both the G and D error components, such as the BCELoss (Binary Cross-Entropy), and additional regularisation can be added to enhance performance during training, such as the MSELoss (Mean Squared Error), which is added to the G loss to provide additional insight into how to improve their fake targets in comparison to real targets. In order to properly respond during training to the circumstances of the problem you are approaching, this component is multiplied by a hyperparameter λ.

5.3.3 Encoding-Decoding

The Encoder-Decoder structure is built on the idea of processing the input sample to create an image embedding, which is a concentrated version of the original data. The Encoder completes the first phase, which is then modified by the Decoder who attempts to readplify it in order to obtain a modified version of the initial structure.

5.4 PIX2PIX

The PIX2PIX Model's findings were obtained from the paper, "Image-Image Translation using Conditional Opposing Networks"[11]. The phrase "translating" an input image into a corresponding output image can be used to describe a variety of issues in image processing, computer graphics, and computer vision. A scene could be displayed as an RGB image, a gradient field, an edge map, a semantic label map, etc., just as a thought could be stated in a different language. We define automatic image-to-image translation, which is similar to automatic language translation, as the process of converting one feasible representation of a scene into another in the presence of enough training data.

Despite the fact that the goal is always the same, predict pixels from pixels, traditionally, each of these tasks has been approached with unique, specialised equipment.

5.4.1 Method

To train a mapping from a random noise vector z to an output image y, a generative model called a GAN is used. Conditional GANs, in contrast, learn a mapping from an observed picture x and a random noise vector z, to y, G: x, z y. An adversarially trained discriminator, D, which is trained to do as good as possible at spotting the generator's "fakes," is trained to produce outputs that cannot be distinguished from "real" images by the generator G.

5.4.2 Objective

The objective of a conditional GAN can be written as 5.2 where G attempts to reduce this objective while an adversarial D seeks to maximise it, i.e. 5.3. 5.2 allows us to verify the significance of conditioning the discriminator by comparing it to an unconditional form in which the discriminator does not witness x. It has been useful in the past to combine the GAN target with a more conventional loss, like L2 distance. The generator's aim is to not only deceive the discriminator but also to be close to the output of the ground truth in an L2 sense. The discriminator's job is left unaffected. As L1 encourages less blurring than L2, we also investigate this possibility using the following formula: 5.4.

5.2 is our ultimate goal. The net would still be able to learn an x to y mapping without z, but it would yield predictable results and be unable to match any distribution other than a delta function.In the past, conditional GANs have taken this into account and added Gaussian noise z as an input to the generator along with x. Only applying dropout noise to a number of the generator's layers during test and training periods. Despite the dropout noise, we only detect a very little amount of stochasticity in our networks' output.

The present study leaves an essential point unanswered: how can conditional GANs be designed to provide output that is highly stochastic and therefore reflect the complete entropy of the conditional distributions they model?

5.4.3 Network architecture

The convolution-BatchNorm-ReLu type of modules is used by both the generator and discriminator. an engine with skips The mapping of a high resolution input grid to a high resolution output grid is a distinguishing characteristic of image-to-image translation difficulties. Additionally, although both the input and the output are renderings of the same underlying structure, they appear differently on the outside. As a result, the structure of the input and the output are roughly aligned. These factors are the foundation upon which we develop the generator architecture. An encoder-decoder network has been widely employed in the past to solve issues in this field.

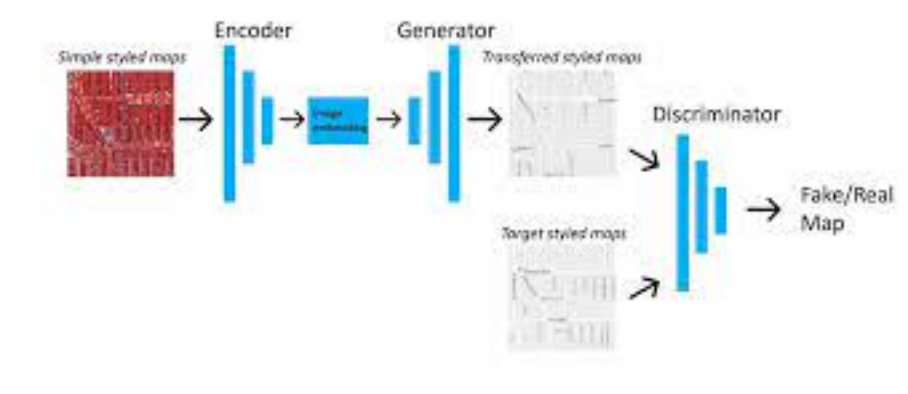


Figure 5.8: PIX2PIX model structure[11]

A bottleneck layer in such a network causes the process to be switched around, after which the input is transmitted via a succession of layers that gradually downsample. Any information flow in such a network must transit through every layer, including the bottleneck. There is a lot of low-level information that is exchanged between the input and output for many picture translation difficulties, and it would be preferable to provide this information directly over the internet. For instance, in the case of image colorization, the locations of salient edges are shared between the input and output.

We add skip connections, roughly following the shape of a "U-Net," to provide the generator with a way to get around the information bottleneck. To be more precise, skip connections are added between each layer i and layer n - i where n is the total number of layers. Each skip connection merely concatenates all layer i channels with layer n - i channels.

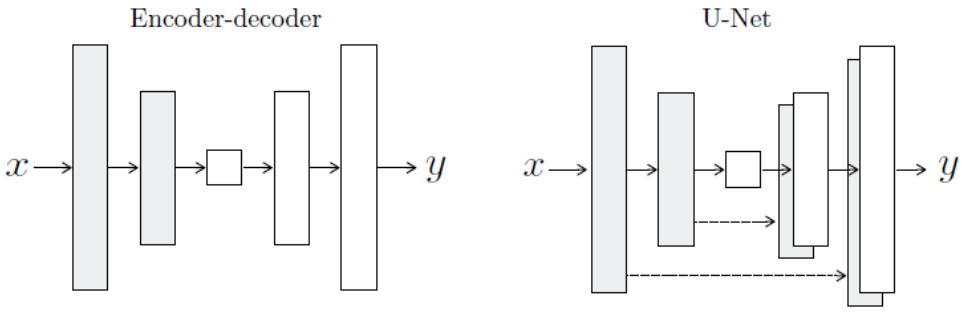


Figure 5.9: Encoder-decoder structure versus U-NET structure[11]

5.4.4 Markovian discriminator (PatchGAN)

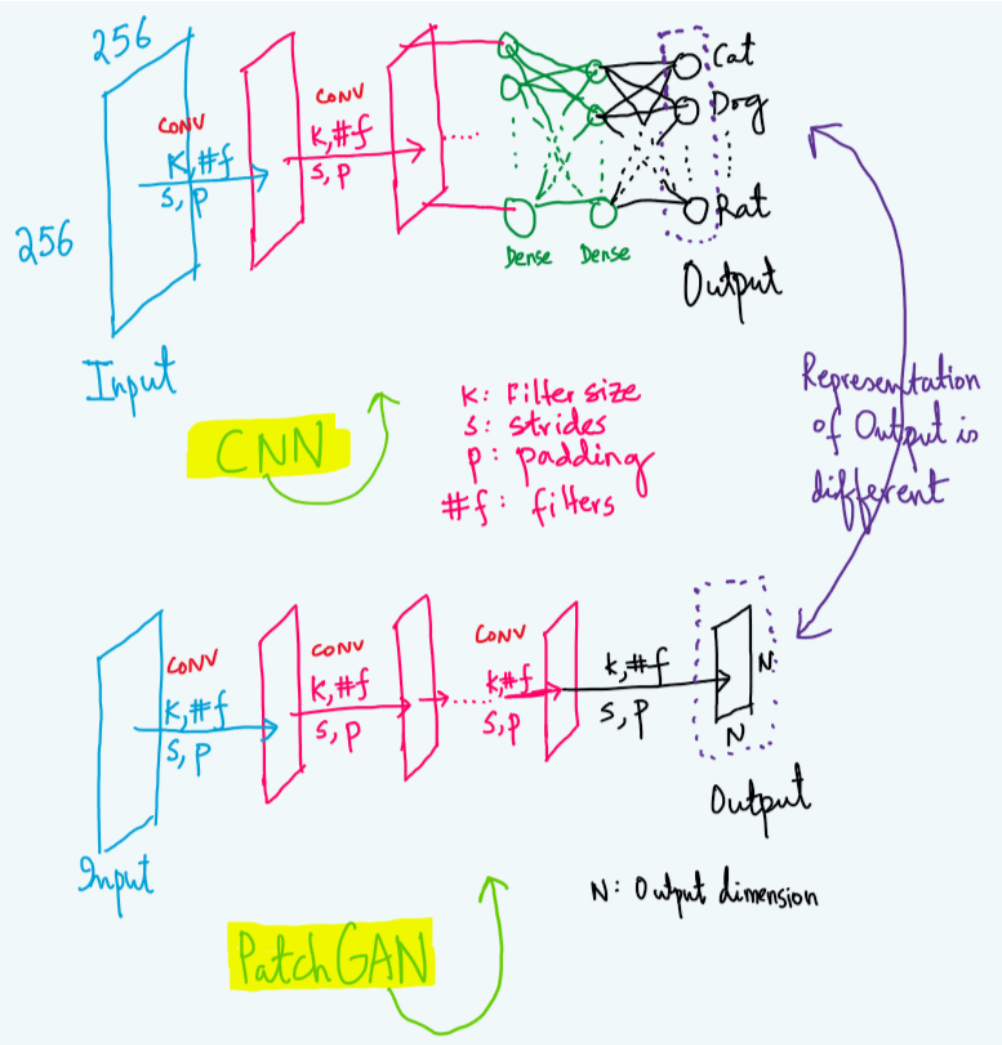


Figure 5.10: PatchGan structure versus CNN structure[11]

It is commonly known that image production issues caused by L2 loss and L1 create fuzzy results. Even while these losses don't promote high-frequency sharpness, they frequently nonetheless effectively capture the low frequencies. We don't require a whole new framework to impose correctness at the low frequency for cases when this is the case. L1 has already started. In order to enforce low-frequency accuracy, the GAN discriminator is constrained to exclusively represent high-frequency structure. We only need to focus on the structure of local image patches in order to model high frequencies. As a result, we create a discriminator architecture that only penalizes structure at the scale of patches, which we refer to

as a PatchGAN.

The objective of this discriminator is to determine whether each NxN patch in an image is authentic or fraudulent. To produce the final output of D, we convolutionally apply this discriminator to the image and average all responses. We are aware that N can be significantly smaller than the original image size while still delivering excellent results. This is beneficial since a smaller PatchGAN may be used for images of any size and has fewer parameters, runs faster. With the assumption of independence between pixels separated by more than a patch diameter, such a discriminator effectively models the image as a Markov random field. In models of texture and style, this relationship is also a frequent tenet. Our PatchGAN can therefore be viewed as a type of texture/style loss.

5.5 MAGNUSGAN

The MAGNUSGAN model's description was drawn from "MagnusGAN: Using Gans to play like Chess Masters"[12].

5.5.1 Dataset

The dataset is made up of a sizable PGN file with over 3000 instances of games played online by Magnus Carlsen, the best chess player at the moment. These files provide a lengthy record of all the moves that were used in that specific game as well as some game-related metadata (date, time, who was playing each side). The python-chess package can parse the PGN. The library plays out every move on the chess board in the PGN file, turning it into data that computers can understand. An array of shapes can then be created using the data (8,8,12). This is a detailed description of how to: It generates an empty 8,8 by 8 matrix. This is due to the chessboard's 8 by 8 square dimensions.

In order to preserve all the information, it is necessary to define 12 different values because there are 6 different types of chess pieces on each side. Except for one value at a specified index, which will be used to indicate which component occupies the square, the array will be entirely filled with zeros. You can also leave this array empty to describe an empty square. It's vital to remember that we will only record Magnus Carlsen's moves because we want to emulate him rather than his rivals. This can be accomplished by adjusting the decision-making processes utilising the metadata.

As soon as the data has been gathered, it must be transformed into a collection of X and Y values so that the GAN can be trained. Following Magnus Carlsen's move, the X values represent the current condition, while the Y values represent the final board. This is because encoding a move is far more difficult than encoding a board since many pieces can enter a square and because en passant and castling

are also factors. The assumption that the board and the board alone hold all the information required to make the optimal decision is another one. This has a slight problem because the opponent's move order may reveal some information about their strategy, which implies the GAN may be operating without all the necessary knowledge.

5.5.2 Network Structure

The Pix2Pix model is the foundation of the MagnusGAN. It has two components: a generator and a discriminator, much like any other GAN. A deep convolutional neural network that conducts image classification serves as the discriminator. Specifically, classification of conditional images. It forecasts the likelihood that the target image is a true translation of the source image or a fake one based on the input of both the source image and the target image. The effective receptive field of the model, which specifies the relationship between one output of the model and the quantity of pixels in the input image, serves as the foundation for the discriminator design. A patch of the input corresponds to each output prediction of this model, which is known as a PatchGAN model.

The generator employs a U-Net design and is an encoder-decoder model. The model creates a target image from a source image. This is accomplished by first upsampling or decoding the bottleneck representation to the size of the output image, then downsampling or encoding the input image to a bottleneck layer. According to the U-Net architecture, skip-connections are introduced to create a U-shaped pattern between the encoding levels and the appropriate decoding layers.

Chapter 6

AI Models

This chapter describes the structure, operation, and results of the two models studied: OKPLAYGAN AND ALPHAOKPLAY. OKPLAYGAN is the model based on the GAN network based on the PIX2PIX model, while ALPHAOKPLAY is the model derived from ALPHAGO ZERO. For each model, various studies modify the input data to adapt it to different contexts of OKPLAY.

6.1 OKPLAYGAN

6.1.1 Introduction

The basic goal is to create a bot that can play without utilizing Reinforcement Learning and can be trained through games. The is an experiment, and the behavior of this model in this new field of application has been explored. In the literature, learning games using neural networks almost solely relies on Reinforcement Learning, which corrects errors by improving following plays by evaluating the neural network's actions by playing against itself.

Pix2pix[11], a GAN network made of Generator and Discriminator, is the project's reference model. The Generator's job is to create data that must be evaluated by the Discriminator, who must determine if the data received was generated by the Generative module or whether it is ground truth data. The Generator has a "U-Net" architecture, which is an encoder-decoder with residual connections between mirrored layers in the encoder and decoder stacks, and is made up of three layers of convolutional (encoder) and deconvolutional (decoder) layers separated by batchnormalization transformations, and also LeakyReLU activatfion function. The Discriminator uses a PatchGAN configuration made up of four convolutional layers with batchnormalization intervals and LeakyReLU without any fully connected layer at the end. The Discriminator is able to capture local

style statistics in this configuration.

The number of layers and the sizing for the convolutions have been adjusted to the 16x16 grid size, that reduced the computational during training and, in the case of larger grid sizes, has increased the calculation times significantly without great improvements. Okplay, the game that will be used to test the model, is a two-player tile-placing game. The following are the game's rules: Every player has 21 tiles that they alternately place on the ground; the first player places his tile in the center, and following players can only place their tiles on the edge of another tile already on the field. If both players have finished their tiles, they can begin moving the tiles that have already been set.

In theory, the game would have no edge constraints, but in this case, it has been converted to a form constrained by a 16x16 square grid used as input for the model. OKPlay is a game that is not well-known at the Chess and Go levels, and it lacks a set of high-quality games comparable to the those played by the grandmasters. The Monte Carlo Tree Search (MCTS) was used to establish movements with the least amount of logic in order to learn the game and create a dataset.

6.1.2 Default Parameters

OKPLAYGAN Parameters

The network parameters of the Siren network are:

- Convolutional initializaion weigth : Normal Distribution
- Batchonorm initializaion weigth : Normal Distribution
- Convoltuional layer init mean : 0
- Convoltuional layer init stddev : 0.2
- Batchonorm layer init mean : 1.0
- LeakyReLU negative slope : 0.2
- kernel size : 4
- strides : 2
- padding : 1

Train Parameters

Unless differently specified, those are the default train parameters used in this paper:

- Loss function: BCELoss and L1Loss
- Optimizer: Adam
- Batch size: 32
- LR: 5e-5
- Number of epochs: 400

6.1.3 Related works

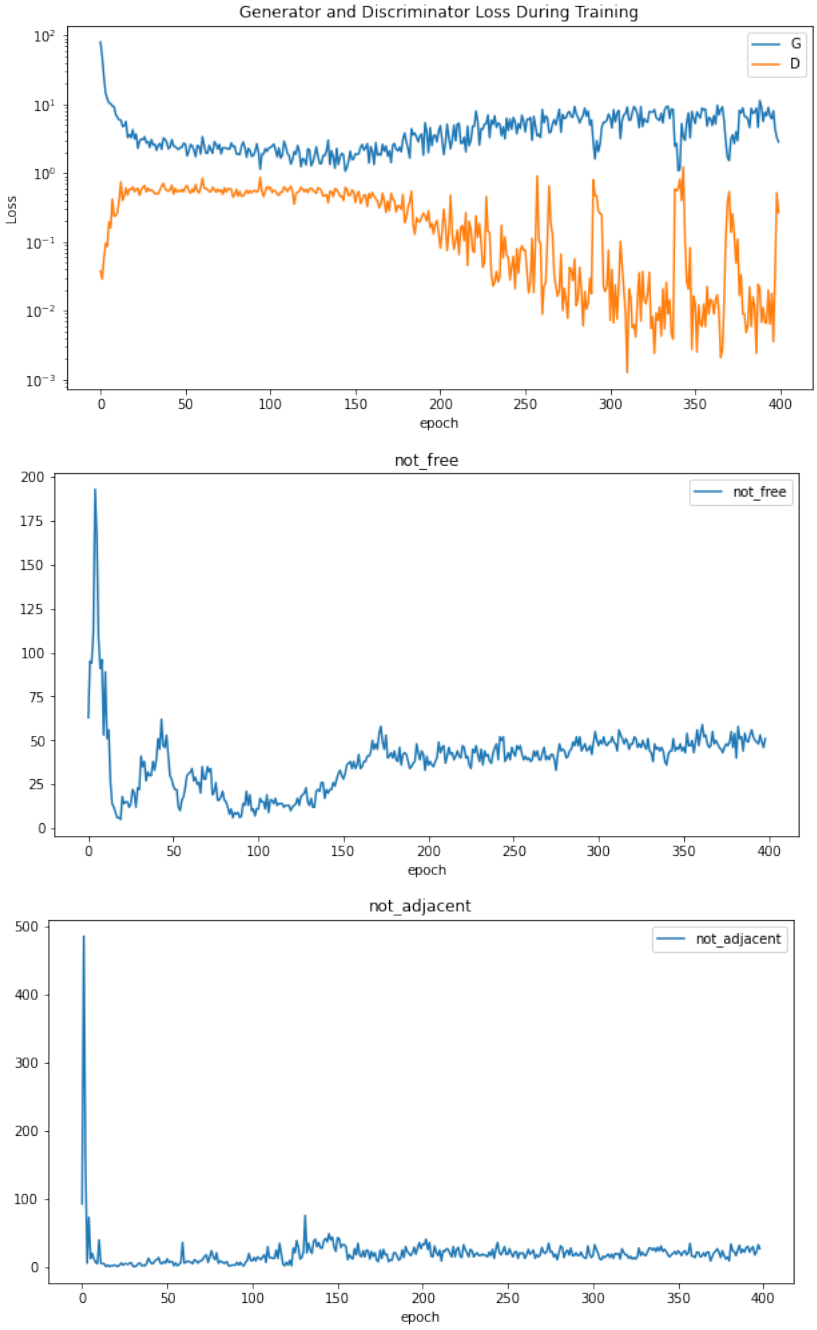
Reinforcement Learning (RL) has traditionally been used in conjunction with neural nets for learning games in order to improve their performance during training.The use of neural network models without the use of RL is a little-explored subject in the literature, with few references; this project is an attempt to foray into new ground by combining pieces from various previous research. In image-to-image translation, the Pix2pix architecture was used: it is good at synthesizing photos from label maps, rebuilding objects from edge maps, and colorizing images. Given enough training data, the key challenge is to translate one possible representation of a scene into another.

The MagnusGAN[12] project, which aims to train a GAN to play Chess using grandmaster games as dataset, is an example of Pix2pix architecture in action. The primary structure of the GAN was influenced by this research. Finally, the structure of the data used for training was based on the structure of the data used to play Othello[13], which has many parallels with OKPlay. This included mainly adding information to the GAN linked to the possible moves that may be done from the turn player.

Ablation studies

Initially, the data input format consisted of three tables: the first contained player tiles, the second contained opponent tiles, and the third contained free cells on the board; similarly, the output format consisted of three tables with the same meaning. The Generator employed a Softmax as the final activation function because the original encoding used 1 to signal the value and 0 to indicate the absence. Regrettably, the model did not converge, and the losses took a different path until dispersing. Also, the model has a lot of issues with finding moves because

Figure 6.1: Initial model configuration results

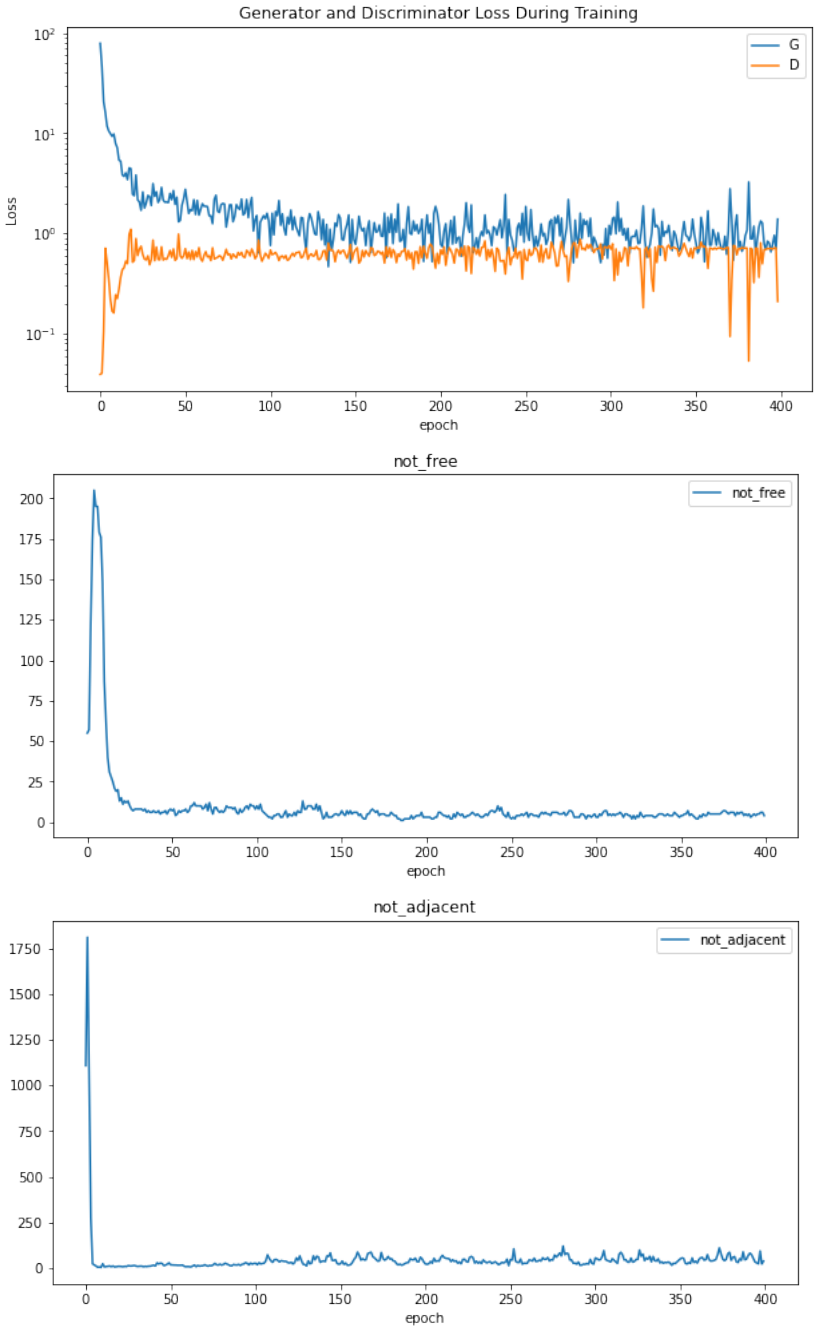


it tries to choose a lot of non-free and non-adjacent cells to the ones that have already been placed6.1.

The third table was removed after reasoning on the input format, and the

encoding was altered to utilize -1 instead of 0, and the Generator final activation function was replaced with a Tanh. Following these improvements, the model began to converge, and the number of non-free placements reduced significantly6.2.

Figure 6.2: Updated version with Tanh and only 2 tables input model results



The next update involved reducing the Generator output from two tables to only one, which could condense the player and opponent tiles, and this allowed to reduce to zero the errors related to choosing an already placed cell, but it also introduced some instability to the choice of not adjacent cells, and the Discriminator loss began to decrease and oscillate6.3.

The final version was obtained by adding a new table to the input that highlights with a 1 the cells that were possible positions for the next move, allowing the training losses to be better converged, as well as keeping the non-free positioning at 0 and reducing the not adjacent positioning slightly6.4.

6.1.4 Methodological section

Dataset

The dataset is made up of matches played between two players in which the winner's movements are extracted. Because there are no known matches to populate the dataset, the MCTS algorithm (Monte Carlo Tree Search) was used to create it. The MCTS algorithm decides how to behave for each placement of the tiles on the table by simulating games composed of random moves in large enough numbers to have a move that can truly have a valuable response on the game. The dataset is made up of the moves of the players, which were determined by analyzing 10,000 games that were randomly played to identify the player's action at each round.

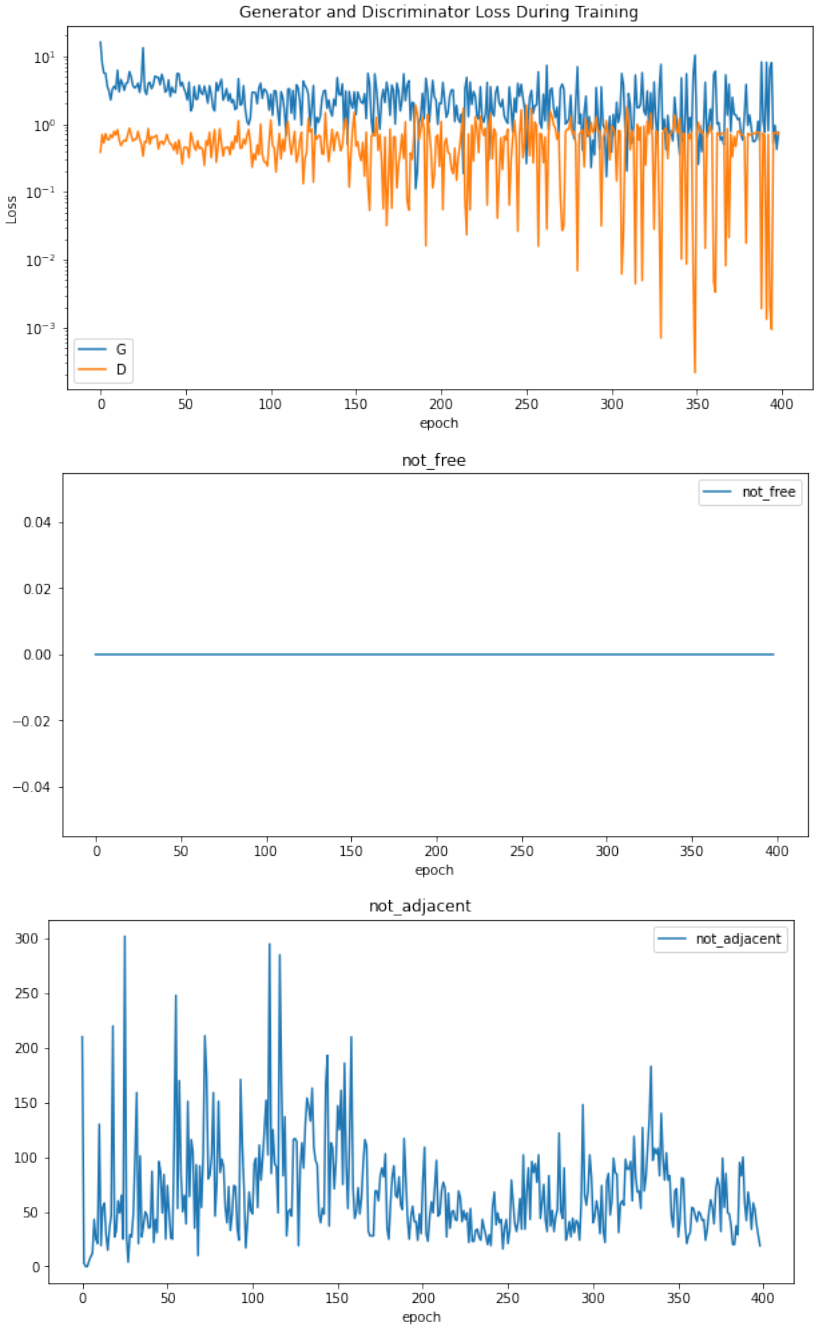
Finally, redundant moves that may have taken more than one feasible move from MCTS starting the same initial board were reduced to one possible move, helping the model to improve the training losses.
The final dataset has 1546 movements, and before training, each table is given a random rototranslation to aid understanding that it should reason on the already placed tiles rather than solely evaluating the central cells.

During the games, the players, who were simulated by the MCTS, played valuing their mosses up to 10,000, as well as versus opponents who only valued their mosses at 10, 100, and 1000, to win games against various types of opponents who could choose to play in different ways, such as following more or less effective strategies.

Validationset

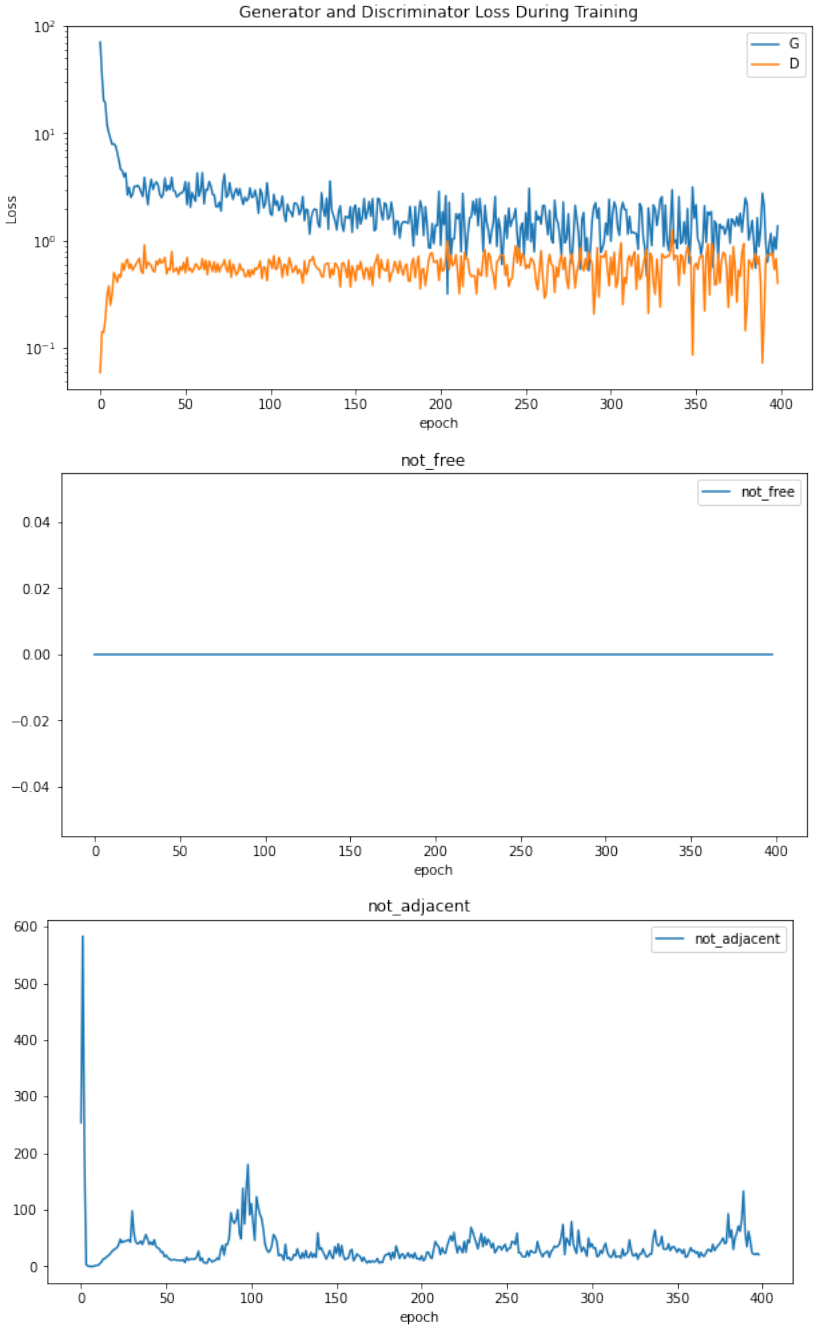
Despite some redundancy with the first moves of the game, the Validationset comprises of some games created by playing the MCTS and analyzing roughly 40 games every move to obtain placements not included in the dataset; the final size is 80 moves. The moves are evaluated by accumulating the values of each cell produced by the Generator in descending order, indicating the priority according to the model of placing a tile in that spot.

Figure 6.3: Only 1 output table configuration results



The estimate calculates how many times the network would have put the tile in non-game-relevant places until it reached a game-relevant position. The valid places are those that are close to the already placed tiles, and the data obtained

Figure 6.4: final version with input neighbours table results



are for ones that are placed in areas that are already occupied or that are not next to the others.

Method

The games in the dataset are divided down into single moves, with the sample table representing the grid before the player move and the target table representing the same table after the player move. A 16x16 square grid with 0 for empty spaces, 1 for first player tiles , and 2 for second player tiles make up the boards.

The sample and target tables are then separated into three 16x16 value -1 or 1 tables, each with the following meanings: The first contains the turn player's tiles marked with 1 and the invalid cells marked with -1; the second contains the opponent's tiles marked with 1 and the invalid cells marked with -1; The third contains all the cells where it can be placed a tile marked with 1 and the invalid cells marked with -1; During training, the sample is sent to the Generator, who creates a new target fake, after which the Discriminator receives alternating concatenations of the sample with the original target and the target fake, as real and fake values, respectively.

The Binary Cross Entropy(BCE) loss function lets the model to improve swiftly from early training epochs till convergence, while the L1 loss function, multiplied for 100, improves Generator learning.

Result analysis

The MCTS is commonly used to evaluate the chosen moves in games, however it is not appropriate in this scenario because it is the method used to acquire the dataset. To see the model's behavior and how it learned the game, as well as to evaluate the quality of the moves, some tables containing a specified move to choose in all case are utilized. The training graphic6.4 indicates that the Generator does not completely confuse the Discriminator; in fact, after multiple bendings, the Generator's loss value is clearly higher than the Discriminator's. It was determined to design several circumstances in front of the net when the moves to win or avoid defeat were fairly evident. Position the fourth case in a row to win6.5: When it comes to placing the fifth tile in a row to win, it is usual to choose a position that is completely different from the intended one, making the game meaningless.

In this case, the opponent locks for the fourth time in a row6.8. As in the previous scenario, in the situation of being able to stop the opponent by laying a tile to prevent his triumph, the net tends to choose a move with a more focused purpose of obtaining his victory while avoiding stopping the opponent. The network was implemented as a bot to play against, and it was discovered that it tries to play particular game patterns found in the dataset's moves without adapting to the current scenario, which should make different judgments than those already seen. Finally, to test the network's behavior, we compared the moves made by the network in the same situations, but with the tiles moved or rotated, and in almost every case, you can see how the network positions its tile differently.

This could indicate that the network does not decide how to play purely on the basis of the tile scheme, but also on their physical position on the grid.

Considerations

We may make several inferences from the findings gained with this model, including the fact that the net does not appear to fully grasp the rules of the game or adapt them to simpler scenarios with fairly clear moves, but rather appears to repeat strategies or patterns learnt from the dataset. The network does not always recognize a precise move to be performed, nor does it always appear to realize which tiles are already in place, and so does not always clearly decide what to do by analyzing other probable places in a not insignificant way6.11.

The model appears to be able to learn tactics and strategies from the dataset's elements; this is a useful feature that could be enhanced.

Figure 6.5: turn player doesn't win in easy condition

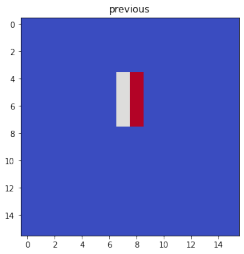


Figure 6.6: Previous

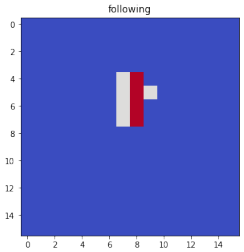


Figure 6.7: Following

6.1.5 Possible optimizations

Dataset

The dataset obtained from the MCTS may not be appropriate for determining basic tactics such as completing files from 5 or interrupting them to the opponent

Figure 6.8: turn player doesn't stop opponent win in easy condition

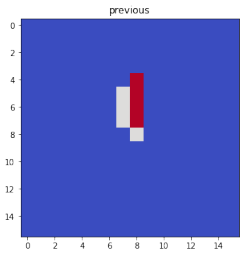


Figure 6.9: Previous

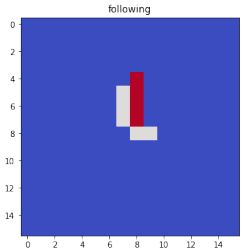


Figure 6.10: Following

Figure 6.11: Network evaluates next move not clearly

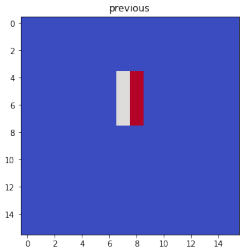


Figure 6.12: Previous

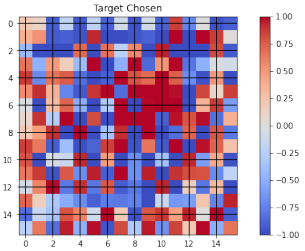


Figure 6.13: Following

because the network appears to be able to learn, at least a portion, of a strategy existent in the matches. As a result, it may be more helpful to learn if the dataset is changed to include not only a player's winning games, but also other actions that are not part of a strategy that leads to success. Instead of attempting to define what is filled by -1 and 1 for each position, another method to improve the model is to investigate an alternative board encoding to better understand it or to provide more information necessary for learning the game.

Network Structure

It is impossible to say whether the model based on convolutional layers is unable to learn the game entirely; however, other structures that can store more information, such as a state of the game, as well as taking into account a basic assessment of the current situation, may produce better results.

6.2 ALPHAOKPLAY

6.2.1 Introduction

Before discussing the ALPHAGO ZERO based model, the DeepMind Google AI model that has defeated Go and Chess champions, we will first go over some fundamental concepts to help you understand how the model works. Keep in mind that the objective is to confirm that the model can comprehend the game in order to train a smart player as an opponent for application users rather than a champion. The model we're going to use for the OKPLAY application is a simplified version of AlphaGo Zero, which has become necessary to employ to reduce training time, the complexity of the model, and the necessary resources in order to acquire an useful result to our ultimate purpose in a suitable amount of time.

The neural network at the core of our model assesses the game board, the surrounding environment, and generates a value that calculates the probability vector for each action to be taken and estimates the current state according to the player and the policy. The agent in the model is a neural network made up of linear and convolutional layers, while the environment is the grid of tiles from the OKPLAY game. The agent employs a representation of the game board that has been encoded with the values 1, -1, and 0 to represent respectively the player's turn tiles, the opponent's turn tiles, and the empty cells without any tiles yet put inside.

The policy and the value that evaluates the received state are both produced by the neural network from the incoming received information. The policy represents a mapping of all possible actions and explains how the agent should evaluate potential action options.

We use the UCT to decide what action to take, which includes taking into account the neural network's evaluation as well as running MCTS simulations on the game's current state and getting a second opinion on what action should be taken.

The predicted net value and the reward range from -1 to 1, with the positive value denoting the turn player's advantage and the negative value the opposite. As the absolute value rises, the level of advantage or disadvantage also rises. The value ascribed to each state of a match based on the outcome that led to the player, that is, victory or defeat, is utilised along with the net rating of a specific state.

To enable the network to identify the player in a favourable circumstance, their difference must be reduced to the absolute minimum.

6.2.2 Implementation adaptation

For this model, the ability to play phase 2 of the game might be added to the mapping of the activities. In the end, this option was not pursued because that would have required a significant increase in training time, even for smaller versions of the game, and would have gone beyond the restrictions placed on the usage of the trained model, which were to be limited to a phase 1 with few tiles.

6.2.3 Default Parameters

Net Structure

1. Convolutional layer : 4
2. BatchNorm2d layer : 4
3. Relu activation function
4. Linear layer : 2
5. final Linear layer for policy : 1
6. final Linear layer for state value : 1
7. BatchNorm1d layer : 2

Net Parameters

I parametri della rete neurale sono:

1. Convolutional layer init weight : Normal Distribution
2. Batchnorm initialization weight : Normal Distribution

3. kernel size : 3
4. strides : 1
5. padding : 1
6. Activation Function : Softmax on pi e Tanh on v

Similar to batchnormalization layers, convolutional layers have been initialised with random weights spread according to a normal distribution. With the intention of providing input non-negative values to the following layers, the network employs a variety of activation functions, such as the Relu for all levels. The network's final section is separated into two distinct linear layers, the first of which creates a policy with a cardinality equal to the total number of moves that can be mapped. The Softmax activation function, which converts values in a range between [-inf, 0], is used for the policy. The Tanh function is used by the state estimate value to transform values between [-1,1].

Train Parameters

1. number of rollouts : 25
2. cpuct : 10
3. threshold : 0.6
4. Loss function : composition of differencesref!!
5. Number of episodes : 100
6. num iterazioni : 500
7. arena compare : 40
8. sliding windows max dim : 20 episodes
9. tempThreshold : 15
10. lr: 0.001
11. dropout: 0.3
12. epochs: 10
13. batch size: 64
14. num channels: 512

We shall describe the functions of each parameter and discuss the rationale behind the final version's selection of these values. The quantity of simulations carried out by the MCTS as Policy Improvement was applied to the policy determined by the neural network is the number of rollouts. Its value has been set to 25, as it was discovered that increasing it significantly did not lead to substantial improvements to training but rather extended training times without producing knowledge gains from a portion of the model.

The cpuct hyperparameter is the factor that controls the use and exploration of the MCTS; in particular, it was thought that in order to achieve noticeable improvements in training, it was required to increase it in comparison to its original value of 1 to 10. Increasing it provided the exploration element more weight, which was required to enable the model to adhere more closely to the moves that were first dismissed. The model's win rate when retrained is what determines whether it should be replaced over a previous version, or what is known as the threshold value. Its value has not altered because increasing it raised the step to overtake too much, preventing it from becoming better enough to adjust to defeat the predecessor so decisively, while decreasing it did not result in the model improving and gaining actual knowledge.

The number of iterations represents how many times the complete training procedure was repeated before the Self-Play stage, network training, and model comparison. Due to the time constraints, our model has been trained for 500 iterations. The number of episodes reveals how many games the neural network engages in with itself in order to produce dataset elements. Given that the dataset did not appear to provide any specific limits to the model, its value has been set at 100. The arenacompare value is the number of matches played between the network previous the training phase and after the training to determine whether the new version is better compared to the first. Its value has been set at 40 in order to produce positive results with a modest number while minimising the training time.

In order to avoid having an excessively large dataset given the continuous generation of new elements that improve more and more in quality and to eliminate for the training phase old elements now transmitted to the network after 20 iterations while focusing on new items with knowledge of the previous and improved, the sliding windows is the queue that collects in each block, for each episode, the moves made during the games in the Self-Play phase. The queue is limited to 20 iterations. The tempThreshold is a value that modifies the policy during the first 15 iterations of the MCTS to produce a policy that forces you to confidently choose one move, highlighting that move as the only actionable.

Although the initial moves may not be the optimal ones, this element aids in the early phases of training by giving the network a clear starting indication of the behaviour it must have. As a result, the network can begin training with a clear understanding of how to move in the game and move on to moves with a less

certain probability after it has started to consolidate its understanding of the game. It goes without saying that the elements produced in this manner will be replaced by new training values that adhere to the original model. The difference between the state value estimated by the network and the actual result of the match in which that state is a part, as well as the product of the policy produced by the MCTS and the logarithm of the policy produced by the neural network, make up the Loss function.

In order to help the network determine whether a player is ahead or behind in a given situation, this feature aims to minimise the gap between the estimated state of the game and its eventual outcome. In order to extract the necessary knowledge to be able to duplicate the policy of the MCTS in all aspects, we therefore want the network's policy to be as similar to the MCTS's policy as possible. For each round of the neural network retraining, only 10 training epochs were employed because the network has previously learned from the old dataset components.

In order to improve the individual neurons that are occasionally employed, the network has also been subjected to a dropout of 30%, which is a random shutdown of the nodes of the linear layers during the training phase. Given the positive outcomes that had already been seen in prior games and the fact that they were not the primary parameters to modify for the game modification, the Learning Rate, the batch size, and the number channel were left at the values originally established.

6.2.4 MCTS for Policy Improvement

The UCT is determined by the MCTS using the following formula during the Policy Improvement phase:

Where:

1. $P(s, \cdot) = p_\theta(s)$ represents the initial estimate of the action a on the state s according to the policy piteta(s) returned by the neural network to its current state;

2. $Q(s, a)$ represents the expected reward by taking the action on the state s;

3. $N(s, a)$ represents the number of times the action has been taken by the state s during the simulations; and Q represents the expected reward by taking the action on the state s.

$$U(s,a) = Q(s,a) + c_{puct} * P(s,a) * \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \tag{6.1}$$

The degree of investigation in this case is controlled by the hyperparameter c_{puct}. We initialise our empty search tree using s as the root in order to employ MCTS

to enhance the initial policy supplied by the current neural network. The steps in a single simulation are as follows. We determine the course of action that will maximise the upper confidence bound $U(s, a)$. We recursively call the search on s' if the next state s' (obtained by playing action a on state s) exists in our tree.

If it doesn't already exist, we add the new state to our tree, initialise $P(s', \cdot) = p_\theta(s')$, the value for $v(s') = v_\theta(s')$ from the neural network, and initialise $Q(s', a)$ and $N(s', a)$ to 0 for each a. We then update all $Q(s, a)$ values and propagate $v(s')$ up along the path shown in the current simulation in place of rolling out values. A terminal condition, on the other hand, causes us to propagate the actual reward, +1 if the player wins, otherwise -1.

6.2.5 Training Phases

1. SELF-PLAY: The neural network competes with itself in games during the Self-Play phase with the objective of incorporating additional dataset elements. The dataset is made up of a finite queue of episodes, each of which is made up of a specific number of games played by the network between itself. If the number of episodes in the queue exceeds the chosen limit, the oldest episode will be eliminated. Three components make up the dataset's elements more precisely: the game's state, the policy the neural network created for that state, and the reward that signals whether a player won or lost the game at the end of their turn.

   To boost the dataset's cardinality and, more importantly, to transmit to the network the idea that rotations of the same game map are identical, each element will be copied for every possible rotation of the game grid.

2. NET TRAINING: Following the addition of the new block of episodes to the dataset, the current network undergoes a brief training cycle with few iterations. In actuality, the objective is to take into consideration the network with the earlier components rather than amplify it on the fresh information. In comparison to the knowledge it already possessed, this enables the network to learn new information in a cumulative manner.

3. ARENA COMPARE: The newly trained network is compared to the previous network at the conclusion of the training period. The win ratio of the new model is assessed after the conclusion of a predetermined number of matches in which the first and second networks alternate in an equivalent manner. If the win rate above the set Threshold, the new trained network will be employed as the starting network for the next iteration of the entire model's training; otherwise, the previous one will be used.

6.2.6 Training Explanation

The neural network will obviously be configured with random weights at the start of training, thus the initial matches will produce elements whose primary logic is based on the MCTS. The quality of each block will improve as the training goes on mostly as a result of recent progress after the training phase and for having followed a previously unconsidered route. The training of the network will increasingly focus on decisions that are far more rational and effective when the queue starts to delete the older blocks. The threshold set aims to recognise a network as legitimate when it reaches a particular level of ability to surpass its predecessor so that this occurrence can be repeated over time, getting better and better.

Even if this new version is rejected if the threshold is not met, the network information that was learned during the previous training will still be retained. In fact, the dataset that will still be employed in following trainings will still contain the factors that allowed the intermediate version to improve, and those trainings will determine whether or not the improvements were truly profitable to next versions.

$$Loss = \min \sum_{t} (v_\theta(s_t) - z_t) - \pi_t * \log p_\theta(s_t) \tag{6.2}$$

It should be noted that it is possible to have identical elements repeated across multiple blocks of the dataset, much like in the game's standard opening movements, however this does not negatively impact training by confusing the network. The same holds true for various moves made from the same state because it has access to a wide range of options and because the MCTS also weighs in on the final decision, which tends to modify the chosen move. Due to this, there is no chance that the model would become stuck in an unimprovable state in the middle of training when there is still opportunity for progress.

6.2.7 Ablation Studies

Given the familiarity of the game strategy on a square grid, we began with a functioning version of the model created for the game Othello to obtain the final parameters. Last but not least, the routines to retrieve the legal moves, verify the win condition, and map out potential actions such as improving all of the game grid's cells have been modified. The game has only ever been reduced to its first phase, without having an infinite supply of tiles, up till the completion of the full game grid, in order to simplify training, especially in the early stages, and to produce tangible results quickly. In order to maintain a centre and simplify the grid view for the model, the grid has also been shrunk to 11x11, odd-sided to ensure a centre and lower the computational load and training time.

Due to the simplicity of the game's rules, we are only able to test the model's capabilities by playing matches against a trained bot, switching between the first

and second player, rather than rigorously evaluating each intermediate model by checking the same scenarios repeatedly. The behaviour of the model in responding to the attempt to immediately complete its own row of tiles without acting on those of the bot provided evidence of whether the learning level was increasing. Additionally, see how the model behaves when given the opportunity to finish his own row and not be forced to block the opponent. In truth, the test was creating a straightforward game to determine whether the model's goal was to attempt to win and stop the opponent in the most drastic circumstances.

The test began right away by seeing how quickly the model could learn the game with W=5, which had a stagnant learning curve and poor performance in practise. We attempted to set W=3 to look for certainties, however the model quickly discovered the clear winning tactic in less than 5 epochs. This demonstrated the model's capacity to comprehend the strategy for victory, if any. We have successfully tested the model's capacity to learn with W=4 in order to determine how to enhance learning in the version with W=5. Even after numerous learning eras when the model had surpassed the threshold, the initial results were delayed and ineffectual because the model didn't aim to win and instead tended to play fairly randomly. We started to try other parameters to see better improvements than the initial configuration because it could be risky to have taught him for a few eras and because the model was not focusing particularly on valuable moves but that and victories were still the result of random moves.

We tried changing the value of threshold by increasing it from 60% to 70% and then to 80%, but we didn't get any better results. It's clear that the model took a lot longer to get better, and in reality, the new option was no better than the old one. The Threshold was further lowered to 50%, but the knowledge learnt remained unchanged. Subsequently, when we tried to test the rollout value from 25 to 15 and then to 50, all that happened was that the training duration changed, and the move choices in the 15 version just got worse.

After that, we made changes to the c_{puct} hyperparameter, trying to increase it from 1 to 5 in order to improve both the number of times the model surpassed the threshold limit as well as the actual move selection. This development was restrained up until the use of a few tiles, after which the moves appeared to be done at random, making it quite simple to defeat the bot. After a decade of training iterations, it was then passed to a value of 10 that caused the model to discover the first player's winning strategy.

This success led to the decision of the final setting for the version with W=5, which was chosen after a few test games revealed that the bot made more reasonable moves after around 40 iterations than the starting setting. The moves were more efficient for many more shifts after 200 iterations. Even after 500 iterations, the bot was still able to play well in the early going before reducing the quality of their move choices as the number of tiles on the field grew above 30. By deciding on the

conclusion of the trials, this limit was adequate to create a model that could be used for the application's purposes.

6.2.8 Result and comments

The outcome is excellent in the early phases since it successfully blocks the opponent's advance while also attempting to benefit from following plays. If given the chance, the bot tries to finish its line to win the game by following the patterns that guarantee success. With an increase in the number of tiles on the field, this playing ability seems to start to degrade, and after 30 moves, it starts to lose some of its effectiveness. This is most likely because more training iterations are required, or more likely, because some training parameters must be changed in order to gain the knowledge required to enhance the model. Unfortunately, we lacked the time and resources necessary to try the improvements.

6.2.9 Possible Optimizations

The primary improvements unquestionably are pushing the model closer to its ALPHA ZERO starting point. The game version with W=5 could still benefit from some attempts to increase the value of the cpuct hyperparameter because doing so lends greater weight to unexplored paths. Given the difficulty of the model when the number of tiles on the field increases, this type of training may be beneficial. The limited move exploration at that point in the game is probably to blame for this difficulty. Increase the rollout value of the MCTS to increase the number of iterations so you can explore more pathways in the training phases, when the challenge can be that you can't analyse enough options, as this will help the model be more inquisitive.

The network structure could be changed, mostly by adding layers, as this is another potential optimization. This adjustment is not advised because you have already obtained excellent results with this version of the game, which is not as complex as Chess or Go, and you could notice some improvements with a change to the other parameters.

Chapter 7

Educational Aspects

This chapter of Educational Aspects describes how the gamification aspects of tasks allow better learning and is an approach that is becoming increasingly widespread for its effectiveness. As said above, the RL model is adapted to the SMaILEApp in the park and station games. Here it is described how these tasks have an educational impact on students are treated. The final comment shows the research status of the SMaILE project. Unprecedented prospects for our society have arisen as a result of the digital revolution and the widespread adoption of AI technologies in practically all spheres of our business and daily life. Humanity's ability to successfully use AI is heavily dependent on the skills and knowledge of those who create, implement, and use it. As a result, developing people's digital capabilities, of which AI is a key component, is a necessary precondition for dealing with the enormous changes that our society is facing[14].

The development of sophisticated digital, cognitive, and socio-technical abilities in our educational system is of highest importance if we want to help people maintain a lucrative relationship with technology and a high degree of employability throughout their lives. All nations will surely find that investing in education will yield great returns since it will increase their human capital and boost their competitiveness on a global scale. Additionally, it will strengthen our capacity for change and help us better steer future technological advancement. The curricula of basic and secondary schools in several nations, including Italy and the United Kingdom, are currently somewhat deficient in digital capabilities, including but not limited to AI. As a result, there is a pressing need to incorporate AI into these curricula as a field of study as well as broader computer science competencies and as a pedagogical tool to raise the standard and efficacy of teaching[15].

To meet this problem, new teaching strategies must be created that help both young students and laypeople learn digital skills and become conscious of AI. Nowadays, AI is more common in day-to-day life. Therefore, it must be introduced to young adults as soon as feasible. The SMaILE project aims to create a pervasive

education in several fields, starting from Game Theory and Machine Learning topics.

The simulation game SMaILEApp, designed for children between the ages of 10 and 18, will be unveiled. It intends to teach and clarify the concepts of AI. According to the constructionist teaching approach, students learn AI through gaming. The SMaILEApp's formalization of pleasant activities aims to lead users toward a solid understanding of themselves.

7.1 Gamification

Gamification is a technique that turns a task into a game to make its advancement more engaging, interactive, and stimulating. Learning through play happens best when it is experienced as joyful, meaningful, actively engaging, iterative and involves social interaction. Learning can take a long time and success can seem far away, so joy is a motivation to continue learning when the task is arduous. When a task makes sense to whoever is doing that and builds on what is known, learning becomes meaningful. So people will remember how they went about solving the problem.

When children solve a real world problem that matters to them, they are immersing themselves in what they are doing reaching a great level of involvement. Making mistakes can help better than communicating only how to do it in the right way to enhance. Also changing something around and repeating testing helps a sense deeper of what and why works to get there in the end. Sharing ideas helps to see things from another perspective, like negotiating, and reaching a compromise with other people learns to communicate and engage others to build together[16].

7.2 AI Games

Now, we will pay special attention to the educational and gamification aspects that have been integrated into park and station games that use bots that have been taught using the research from this thesis.

7.2.1 Park Game

Park game aims to introduce the users to adversarial search techniques. In particular, players should become familiar with tree search with some basic knowledge of the min-max technique and alfa-beta pruning. This aim was gamified using a step-by-step approach. First, the user must define the destination of a green area in the city. To do so, it must consider the environmental impact, trying to minimize it, and the citizen deciders, trying to maximize them. While playing this selection, the

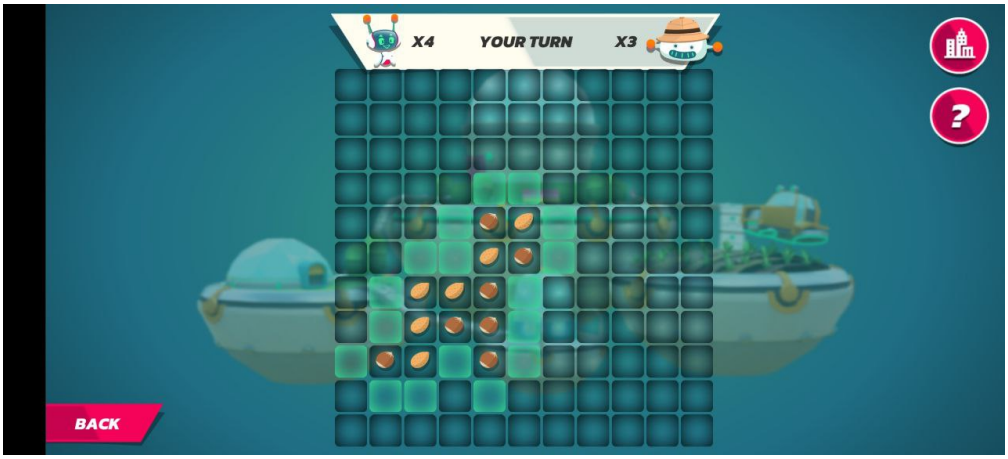


Figure 7.1: An illustration of a park game

user is guided on a tree creation using a min-max technique. If the final decision regards a playground, then in that area, from now on, it will appear as a game with the nim's mechanics. Nim's is a mathematical game in which it is always possible to find the winning strategy.

Therefore, the user learns how to detect the winning strategy using a tree. Instead, in the area destination is chosen to be a lawn, and then the player opens the "plant and sow" game. That is, a game recalling OKPLAY mechanics. These activities aim to show how complex the winning strategy research could become if one exists. Although, by playing a couple of times, users may adopt an alpha-pruning approach that will be formalized correctly in a short video tutorial available in the city hall.

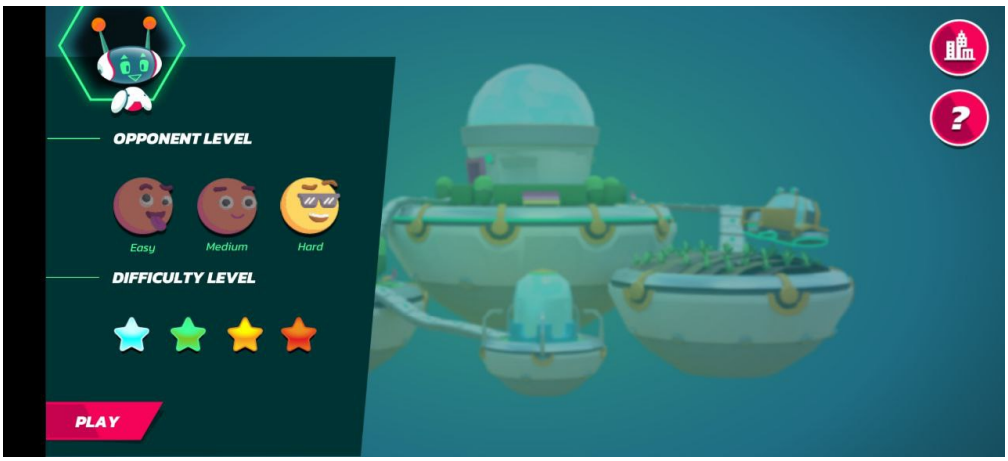


Figure 7.2: An illustration of how to customise the park game's difficulty

Entering in more detail, players compete against an AI that has been programmed to play OKPLAY. Each player has a type of seed in a limited number available. The lawn, an 11x11 grid, is where players plant the seeds. In turn, they must place a seed in one of the available and empty pieces of land. The user is always the first player. Win who is able to sow five seeds in a row. They could be in a vertical, horizontal or diagonal row.

Before starting, the user can choose the difficulty and opponent levels. The difficulty level is made by reducing the number of seeds available. The opponent level determines the AI bot to use. Each opponent level affects the random pick percentage of a randomised move played by the bot from all viable possibilities. In the easy one, the random percentage is 70%. At the intermediate level, the percentage is at 30%. Finally, at the maximum level, the bot chooses all moves.

7.2.2 Station Game

The station game serves as a tool to help the user understand what it takes to train a neural network, identify the key elements, and formulate an improvement strategy. The game aims to teach how to set training to achieve the desired goal properly in a limited time and space. In particular, the player should train the smart transporter to win a food supply against a smart transporter set by another user. The user can choose against which smart transporter to compete and have access to its training setting. Then, s/he appropriately picks the parameters of her/his own transporter, runs the training and starts competing. The food supply will be done by letting the two bot play against each other in a game with OK PLAY mechanics.

The setup is made by selecting values for three parameters that are used to train the bot, resulting in a neural network that has been trained using these criteria. Practically speaking, all training models have already been constructed, and the user's selection of parameters determines the final bot, due to the time and resources required for the real-time training of the potential choices of additional users. On a grid of nine by nine tiles, the objective of the game for which the bot is trained is to create a row of four tiles.

In order to alter the selection of training parameters and shorten the training period to achieve the required number of bots in a fair amount of time, simplification turned out to be necessary. In order to keep things simple and why many bots that had been trained differently were able to figure out the winning strategy as the first player in approximately ten epochs, there were only 5 training epochs for all of the models. This prevented the second player from improving during training.

Three variables are within the user's control; because they have an impact on both the complexity of the calculations and the training times, three values have been selected for each of them:

- The Threshold, which compares the percentage of victories between the new model and the previous one, establishes whether the new model should be used as the new starting point for the following training epoch. It has three values: 40%, 60%, and 80%.
- When choosing a move to play, consider the value of the rollout, the number of matches to simulate, and the weighting that must be applied to the chosen move. There are three options: 5, 25, and 50.
- The c_{puct}, a multiplicative factor that determines the weight to be given to lightly travelled actions while determining the next move, enables the bot to explore alternate routes when it is heavily marked on established courses. The different elements can be 1, 5, or 10.

The results of the matches are not always the same, as can be shown from a comparison of multiple matches between bots using the same settings; this is because the MCTS always introduces an uncertainty factor. While examining various options, player can see that there is usually one that is preferable to the rest. However, this is not always the case because, given the training constraints imposed, retraining bots could be generated differently and still be preferable to the alternatives given the same parameters.

All of this is done to teach children how to handle and make better decisions when it comes to building neural networks. The winning percentage is not too low to accept any new bot nor too high to lose the improvements found, so player does not need to weigh the choices excessively through an excessive number of attempts but giving enough weight to the less travelled choices. In the end, user can see how the bot trained with Threshold at 60%, c_{puct} at 10 and rollout at 25.

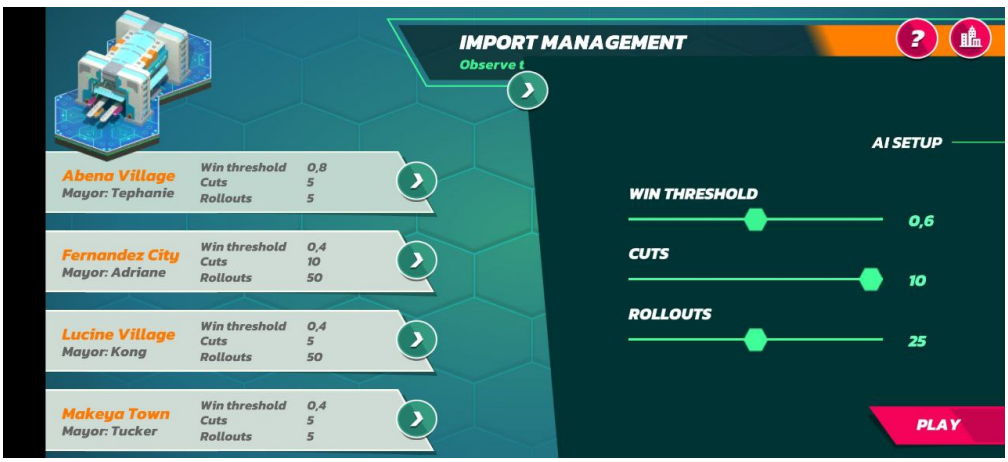


Figure 7.3: An illustration of parameter selection for the station game

7.3 Educational Objective

Using the SMaILEApp is hoped to raise young kids' awareness of AI. Thanks to how the app design was done, there are no prerequisites, and users can start the learning experience without being strictly instructed on the theoretical underpinnings of the various activities. Users can freely formalise the concept they have explored during the game through two short videos of about 3 minutes each, following the learning-by-doing methodology. The first is dedicated to introducing theoretical concepts, while the second relates to the game application. The paradigm is "first test, then formalise", increasing the youngsters' interest and motivation to pursue the deeper personal exploration of the knowledge.

The game experience becomes less forceful and more engaging after being formalised to fill in the gaps and disprove previous objections, as opposed to being frequently misinterpreted and ineffective, as is the case with traditional frontal lesson approaches in the classroom. Gamification has done an excellent job of providing a similar context for all the activities related to the central theme, the smart city, to make the application and its contents more appealing to kids. In particular, the decision to make the layout of your city navigable in 3D gives you a sense of greater immersion in the game.

Children who feel less capable of handling challenges in logical and mathematical games frequently give up on them. That is why users have the chance to take on any challenge at the level at which they feel capable and to adapt it in the case of failure. In this way, the game encourages players to increase the difficulty after finishing it at a moderate level so they may achieve a higher score and help their city advance.

The learning experience has some secondary outcomes: reinforcement in STEM subjects, visio-spatial ability training and fostering STEM careers. This was done with particular regard for gender differences. For this reason, the GUI was designed to be attractive for girls and boys, and all the AI assistants inside the game are gender balanced.

7.4 Research Study

During all the development stages, the games (both in a paper version and inside the app) were evaluated through a couple of focus groups at the "Convitto Nazionale Umberto I", partner of the project. Related to the app launch, a research study has started. The research aims to verify the impact of the application through a randomized controlled trial. The results will allow an understanding of how and to what extent the free and autonomous play of boys and girls supports the knowledge of AI and the increased interest in scientific and ICT subjects.

The testing and validation phase is conducted in collaboration with the Institute for Evaluation Research on Public Policies (IRVAPP) of the Bruno Kessler Foundation (FBK) of Trento. The study is aimed at all lower secondary schools located in Piedmont and is aimed at the second-year classes as a target of interest. The adhesion of the schools is voluntary and requires the participation of at least two classes (participating with more classes is also possible).

In the case of two classes, one will receive SMaILEApp already in the 2022-23 school year, while the other will only be at the end of the school year. During the year, students will participate in two tests, one before and one after the intervention, to measure computational skills, attitudes towards AI and intentions to continue with higher education.

The experimentation of the SMaILEApp is currently in progress in Piedmont, with 20 schools participating with 58 classes, that is, more than 1800 students. The final results of the experimentation are expected in September 2023.

Chapter 8

Conclusion

It has been proved that solving OKPLAY by considering every possible move is impossible since it is difficult to interpret the results in a clear and understandable manner. Even if a winning strategy for the first player has not been identified, it can be declared that one exists thanks to the rules of OKPLAY and game theory for combinatorial games. Studying the reduced cases of the game has emerged as the first player has an enormous advantage over the second since the beginning of the game. Unfortunately, the strategy based on the GAN network did not give the results that would have allowed us to say that the model can play accurately and understand how to play in conditions never encountered during training. After presenting these results, It is possible to aspire to potential performance improvements by expanding the dataset elements to provide the game model with additional knowledge.

Another improvement could be adding a history of previous positions during the game to give a sense of continuity that could express the strategy used. When used with the game's simplified versions, the RL model operates brilliantly, but it is hard to make it intelligent in long matches because it doesn't often make the right decisions. By changing the network's structure by adding more layers, the model has proven capable of improving its interpretation of the logic of the game. In addition, changing training parameters to accelerate the process brings better results and reduced training time. After 500 epochs of training, the OKPLAY final model, based on RL with W at 5, showed how to successfully play up to about 30 tiles placed on the field. Since the W 4 version required at least 18 tiles, it was evident experimentally that the model could not have developed a successful strategy with a player restriction of 21 tiles.

The user can select the number of cards and difficulty level of AI he desires for each task. From an educational point of view, designing the task increases students' desire to repeat the experience to improve their scores. Students haven't to face the challenge to improve, but they want to. This is an excellent achievement

at the educational level. In a few sessions of first player training, the same RL model in the W as 4 version identifies the winning strategy, demonstrating its ability of game adaptation. It is used in the station game to show how to influence during the model's training phase after the model succeeded with W as 4. By allowing the selection of relevant training parameter values, users can observe how the model responds to altering them and learn whether they result in improvement or deterioration.

The students' decisions to try to enhance the bot were verifiable in concrete ways, and this experience has shown them how to improve their own choices through competition among the many bots. Further study is needed as there is still a long way to go before using GAN models to explore combinatorial games bringing good results like those produced by RL models.

Bibliography

[1] Randi Williams, Hae Won Park, Lauren Oh, and Cynthia Breazeal. «PopBots: Designing an Artificial Intelligence Curriculum for Early Childhood Education». In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (July 2019), pp. 9729–9736. DOI: 10.1609/aaai.v33i01.33019729 (cit. on p. 1).

[2] *smaile project*. https://www.smaile.it/ (cit. on p. 4).

[3] Anna R. Karlin and Yuval Peres. *Game Theory Alive*. Dec. 2016, pp. 12–27 (cit. on pp. 9, 11–13).

[4] Eric Duch^ene. «Combinatorial games: from theoretical solving to AI algorithms». In: () (cit. on pp. 9, 25).

[5] Jos W.H.M. Uiterwijk. «Solving Strong and Weak 4-in-a-Row». In: () (cit. on p. 16).

[6] Maureen T. Carroll and Steven T. Dougherty. «Tic-Tac-Toe on a Finite Plane». In: () (cit. on p. 17).

[7] Cameron B. Browne et al. «A Survey of Monte Carlo Tree Search Methods». In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 1–43. DOI: 10.1109/TCIAIG.2012.2186810 (cit. on p. 26).

[8] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. «Deep Reinforcement Learning: A Brief Survey». In: *IEEE Signal Processing Magazine* 34.6 (Nov. 2017), pp. 26–38. DOI: 10.1109/msp.2017.2743240. URL: https://doi.org/10.1109%5C%2Fmsp.2017.2743240 (cit. on pp. 27, 30).

[9] David Silver et al. «Mastering the game of Go without human knowledge». In: *Nature* 550.7676 (2017), pp. 354–359. ISSN: 0028-0836. DOI: 10.1038/nature24270 (cit. on pp. 28, 29, 31, 32).

[10] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2015. DOI: 10.48550/ARXIV.1511.06434. URL: https://arxiv.org/abs/1511.06434 (cit. on pp. 32, 33).

[11] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. *Image-to-Image Translation with Conditional Adversarial Networks*. 2018. arXiv: 1611.07004 [cs.CV] (cit. on pp. 34, 36, 37, 40).

[12] Victor Sim. *MagnusGAN: Using GANs to play like Chess Masters*. 2020 (cit. on pp. 38, 42).

[13] Pawel Liskowski, Wojciech Jaskowski, and Krzysztof Krawiec. «Learning to Play Othello With Deep Neural Networks». In: *IEEE Transactions on Games* 10.4 (Dec. 2018), pp. 354–364. ISSN: 2475-1510. DOI: 10.1109/tg.2018.2799997. URL: http://dx.doi.org/10.1109/TG.2018.2799997 (cit. on p. 42).

[14] Rahul Reddy Nadikattu. «THE EMERGING ROLE OF ARTIFICIAL INTELLIGENCE IN MODERN SOCIETY». In: 4 (Dec. 2016), pp. 906–911 (cit. on p. 60).

[15] Zornitsa Yordanova. «Gamification as a Tool for Supporting Artificial Intelligence Development – State of Art». In: *Applied Technologies*. Ed. by Miguel Botto-Tobar, Marcelo Zambrano Vizuete, Pablo Torres-Carrión, Sergio Montes León, Guillermo Pizarro Vásquez, and Benjamin Durakovic. Cham: Springer International Publishing, 2020, pp. 313–324. ISBN: 978-3-030-42517-3 (cit. on p. 60).

[16] Claire Liu, Lynneth Solis, Hanne Jensen, Emily Hopkins, Dave Neale, Jennifer Zosh, Kathy Hirsh-Pasek, and David Whitebread. *Neuroscience and learning through play: a review of the evidence*. Nov. 2017. DOI: 10.13140/RG.2.2.11789.84963 (cit. on p. 61).