BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND
COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE

DIPLOMA THESIS

MiniMax Algorithm for Gomoku Game: A Multi-Language Analysis and Performance Evaluation

Supervisor
Arthur-Jozsef Molnar

Author
Mădălin-Augustin Raiu

2023

UNIVERSITATEA BABEŞ-BOLYAI
CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICA
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENŢĂ

Algoritmul MiniMax pentru jocul Gomoku : Analiză multilingvă și evaluare a performanțelor

Conducător științific
Lect. Dr. Arthur-Jozsef Molnar

Absolvent
Mădălin-Augustin Raiu

2023

ABSTRACT

Popular in the field of artificial intelligence, the MiniMax algorithm is a game theory decision-making system. Its main goal is to determine a player's best move by examining all of the alternative actions and their results. This thesis evaluates the MiniMax algorithm's effectiveness and performance in C# and Python, two widely used programming languages. In order to test and contrast the algorithm's performance across several programming languages, we use the Gomoku game as an intermediary.

To comprehensively assess the algorithm's performance across different programming languages. Gomoku serves as an intermediary for comparing and contrasting the algorithm's performance in C# and Python. By using this well-known game, we can analyze various performance metrics, such as execution time and memory usage, to gauge the algorithm's efficiency in each language.

Furthermore, we analyze the quality of code implementation and evaluate the code's readability and maintainability. We assess the code quality based on factors such as flexibility, modularity, and ease of modification. The study aims to identify the most suitable programming language for the long-term development of software applications, particularly in scenarios where code maintainability is critical. Our objective is to compare the efficiency and readability of the code implementation in each programming language, examining how syntax and language features affect the algorithm's performance and maintainability.

By presenting a thorough analysis of the MiniMax algorithm's performance across different programming languages, this thesis offers insights that can assist both academic researchers and business professionals in selecting the most suitable programming language for their specific use cases. The conclusions drawn from this study will aid in informed decision-making regarding the optimal programming language for implementing the MiniMax algorithm and, by extension, other similar software applications.

Contents

1 Introduction **1**

1.1 Background and motivation 1

1.2 Methods and objectives 2

1.3 Structure of the Paper 2

2 Literature Review **3**

2.1 Overview of Gomoku 3

2.1.1 Origin of the game 3

2.1.2 How to play - rules and playing strategies 3

2.2 Technologies used 5

2.2.1 Python 5

2.2.2 C# 7

2.2.3 WPF 9

2.3 Overview of the MiniMax algorithm 10

2.3.1 MiniMax 11

2.3.2 Brute force 11

2.3.3 Alpha-Beta 12

3 Methodology **18**

3.1 Implementation 18

3.1.1 Steps 18

3.1.2 Application Presentation 21

3.1.3 Moves 22

3.1.4 MVVM 23

3.1.5 Exploring Important Classes and Their Roles 27

3.1.6 Evaluation and Results 29

3.2 Languages comparison criteria 30

3.3 Testing approach 30

4 Results and Analysis **33**

4.1 Technical comparison 33

4.1.1 Performance 33
4.1.2 Language Features and Libraries 36
4.1.3 Resource Usage 38
4.1.4 Community and Documentation 38
4.1.5 Development Productivity 39
4.2 Results Interpretation 40

5 Conclusion **41**

Bibliography **42**

Chapter 1

Introduction

1.1 Background and motivation

The current landscape of AI research in video game bots is heavily influenced by the pursuit of achieving high win percentages. While achieving victory in strategy games can be difficult, game developers need also take into account the creation of AI opponents who are both challenging and fun to play against. Balancing competitiveness with entertainment value is a significant challenge for the game industry. While the algorithm itself provides a powerful decision-making framework, the choice of programming language can significantly impact the efficiency, flexibility, and overall performance of the AI agent.

Gomoku has been the subject of study for artificial intelligence researchers for a lot of years. Especially the MiniMax algorithm has been used to make programs that play at a high level. It is a decision making algorithm used in game theory that computes the best move to make.

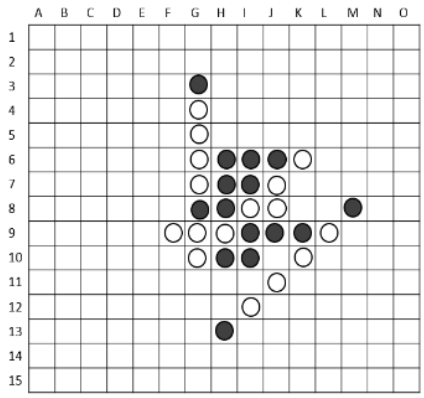


Figure 1.1: An image of a Gomoku board [7]

Ultimately, the comparison between C# and Python aims to provide valuable guidance to developers in selecting the most appropriate language for their specific game

development needs. By understanding the strengths and limitations of different languages in the context of implementing the MiniMax algorithm, developers can make informed choices that optimize performance, code quality, and the overall developer experience.

1.2 Methods and objectives

In the realm of game development, the MiniMax algorithm has emerged as a powerful tool for creating intelligent game-playing agents. This algorithm, rooted in artificial intelligence and decision theory, aims to optimize the performance of game-playing agents by efficiently searching through the game tree to determine the best possible moves. The MiniMax algorithm has found applications in various strategy games, including Gomoku, a well-defined and popular game.

This thesis focuses on the utilization of the MiniMax algorithm to create a Gomoku game implementation using different programming languages, predominantly Python and C#. The aim is to investigate and compare the performance of the algorithm across these languages, treating Gomoku as a benchmark for evaluation. By examining the strengths and weaknesses of various programming languages in the context of strategy game development, this research provides valuable insights for developers seeking to choose the most suitable language for their specific requirements.

Using different programming languages for implementing MiniMax allows us to dive into the strengths and weaknesses of different languages for strategy games development. It also provides a practical example of how different languages can be used to implement the same algorithm by comparing the code and performance of the MiniMax algorithm in different languages, this way developers can choose the most appropriate language for their specific needs.

1.3 Structure of the Paper

The paper begins by introducing the **Literature Review** in chapter 2, which discusses the **Origin of the game**, **How to play** and the **Technologies used**.

The discussion from chapter 3 outlines the methodology that is used in this study, including the implementation details, presentation of the application, performance evaluation criteria and testing approaches.

Chapter 4 presents the results and analysis of the comparative study, highlighting the strengths and weaknesses of Python and C# in game development using the MiniMax algorithm with alpha-beta pruning.

Finally chapter 5 concludes the thesis, summarizing the findings and providing recommendations for future research directions.

Chapter 2

Literature Review

2.1 Overview of Gomoku

2.1.1 Origin of the game

Considered to be one of the world's greatest strategy games, the Gomoku board game was brought to Japan around 270 BC, with the name of Kakugo which has a meaning similar to "five steps" and became fast the national game , in English dominant countries Gomoku is also known as 5 in a row. [2]

In the Japanese language, "Go" means five and "moku" has a lot of meanings in their tradition, but it is know as a counter word for pieces. When it was introduced to Britain the game as known as "Go Bang" or "Pegity". Even though Gomoku has simple rules, it is one of the most complex and challenging board games that exists. [23]

Gomoku has gained a spike of popularity in the 19th and 20th centuries when it was introduced to Europe and North America by immigrants and as technology was advancing, it became a popular game for artificial intelligence researchers and developers when they wanted to test their algorithms.

Throughout its long and rich history, Gomoku has undergone various rule standardization and has been adapted into different variations, with people implementing their own rules, thus making the game always feel fresh and of course, fun. The most popular rule is the "Renju" rule which weakens the advantages for the first player in Gomoku by adding special restrictions, usually this rule is used in competitive play only.

2.1.2 How to play - rules and playing strategies

For the standard version of the game, the rules are pretty simple:

1. Players alternate in placing a piece of their colour on an empty spot

2. There's no restriction where to put the piece, only for the place to be empty

3. The players who puts their pieces alternately until an unbroken row of five pieces are placed either horizontally, vertically, or diagonally wins, but I added a special rule for my version of the game, this was it makes the game more challenging, the rule states that if you are blocked by your opponent's piece on both sides of the chained pieces that would normally give you the win, you will not be declared winner and the game will continue.

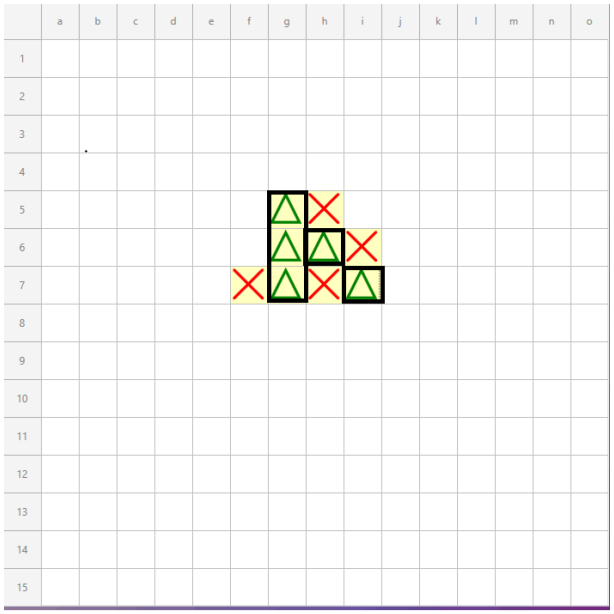


Figure 2.1: Playing Strategy

Gomoku has a strong advantage for the first player when unrestricted and if the player knows how to play in an optimum way, being the first player to put the piece down allows you to control and dictate the start of the game.

It's recommended to have a good strategy when playing Gomoku, use your opponent's turn to think about your next potential moves. If your opponent has 4 in a row, don't waste your time thinking about what you'll do next, simply block your opponent since that's what you need to do to continue the game.

The beginning of the game is often the most important part and it mostly defines how it will end, since the options remaining on the board are less and less as the game goes on, if you put yourself in a bad position in the first part of the game you will have a hard time making a comeback.

Try to read your opponent, in most games that are player versus player one of the best skills you can have is reading your opponent, try to predict their next move, learn the style and strengths that your opponent has, decide whether you should be more aggressive or defensive with your placements.

Try to create 2 lines that share a common piece (Figure 2.1), this will give you a higher chance of closing the game. This is one of the most efficient strategies in playing Gomoku, but keep in mind that this is an aggressive strategy and you can be punished by a smart player.

One of the worst moves you can do in Gomoku is letting your opponent achieve an open 4, which means that they have space at both ends of the line, if this happens, your opponent will win the game with his next move. A move that you are forced to make is that when your opponent manages to have 3 pieces in a line, you have block one of the ends or you will lose next turn.

Gomoku can be a really stressful and challenging game, so the best decisions have to be well planed and be as creative as possible, the possibilities are almost endless, which makes this game more exciting.

2.2 Technologies used

2.2.1 Python

It was first released in 1989 by Guido Van Rossum, and its development continued throughout the 1990s with contributions from a growing user and developer community. Van Rossum's love for the British comedy group Monty Python inspired the name "Python." Python's first public release, version 0.9.0, in 1991, included features such as exceptions, functions, and modules, laying the groundwork for the language's development as a general-purpose programming language.[20]

Python's popularity has continuously grown over time, owing to its simplicity, readability, and versatility. Python 2.0, released in 2000, included significant innovations such as a garbage collector and list comprehensions, which substantially enhanced Python's capabilities. Python 3.0 was launched in 2008, with important language modifications to increase consistency and eliminate unnecessary functionality. Unfortunately, for certain customers, the shift from Python 2 to Python 3 was not flawless, resulting in a time of coexistence and support for both versions.

Despite this, Python's popularity has grown, notably in sectors such as web development, scientific computing, and data analysis. Several of the compatibility concerns were overcome with the introduction of Python 3.5 in 2015 and subsequent releases, and Python 3 became the recommended version for new projects. Python's adaptability and huge library ecosystem, which includes popular frameworks like Django for web development and NumPy for scientific computing, have cemented its status as one of the most extensively used programming languages in the world.

It has also acquired substantial interest in the fields of artificial intelligence and machine learning in recent years. Python has been a popular choice for developing

advanced AI applications due to the availability of powerful libraries such as TensorFlow, Keras, and PyTorch. Python's ease of use, combined with its rich ecosystem and strong community support, has contributed to its rapid rise in the field of artificial intelligence, with various cutting-edge solutions being built utilizing Python as the primary programming language.[20]

Python is still evolving and thriving today as a versatile language utilized in a wide range of applications, from simple scripts to sophisticated software systems and advanced AI technologies. Python is a popular programming language for both beginners and expert programmers because to its long history, simplicity, readability, and broad library support.

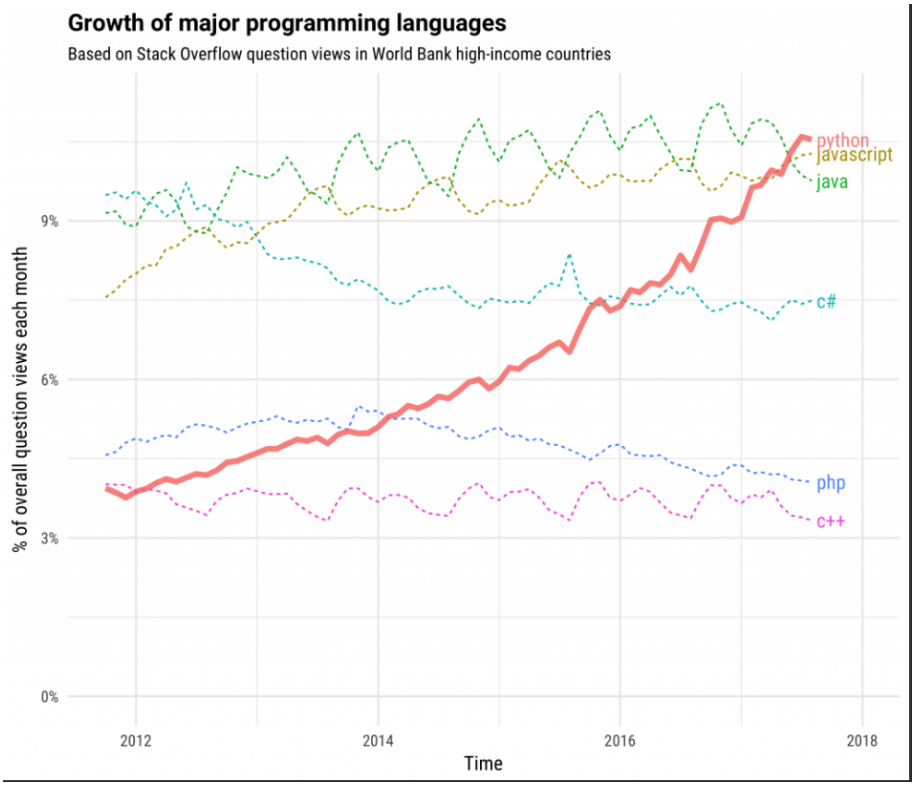


Figure 2.2: Growth of major programming languages [16]

Python has grown in popularity as a programming language for implementing artificial intelligence (AI) algorithms such as the MiniMax algorithm. The MiniMax algorithm is a popular AI decision-making and game-playing strategy, particularly in games with perfect knowledge, where all possible movements and outcomes are known.

The MiniMax algorithm can be implemented in Python using the language's built-in data structures and functions, making it very simple for programmers. Python's structure and readability make sophisticated algorithms like MiniMax simple to understand and implement.

The MiniMax algorithm is a recursive method that investigates all possible movements in a game tree, estimating the best move for a player by taking into account the probable outcomes of each step. Python's flexibility and variety enable quick implementation of the MiniMax algorithm, making it well-suited for games such as chess, tic-tac-toe, and others.

Furthermore, Python's rich library ecosystem provides other tools that can be linked with the MiniMax algorithm to improve its efficiency, such as NumPy for numerical computing and scikit-learn for machine learning. Machine learning approaches, for example, can be used to assess the desirability of specific moves or to optimize the evaluation function utilized in the MiniMax algorithm.

2.2.2 C#

Microsoft C# is a modern, object-oriented programming language. It is a language that is part of the .NET framework, and it is often used for building Windows desktop applications, web applications, cloud services, and game development. C# is well-known among developers for its ease of use, adaptability, and scalability , which makes it an appealing option for a wide range of applications while also allowing the development of AI applications that are capable of processing and analyzing data from several sources.[1]

While you can master the mechanics of C# without knowing anything about the context in which it was built, you'll have a more intimate relationship with it if you understand why it looks the way it does—its genealogy, in a logical sense.While it could be a stretch to say that C# and.NET would not have existed without Java, it is equally difficult to argue that it had no impact.[11]

C# and.NET were officially unveiled at the Professional Developers Conference (PDC) in July 2000, however some aspects had been released before and the same technology had been discussed under alternative names.

This language also has a number of libraries and frameworks for developing machine learning models, including ML.NET, Accord.NET,. Developers can use these libraries to create models for categorization, regression, and other machine learning-related tasks.C# provides deep learning capabilities through frameworks such as TensorFlow.NET and Keras.NET. Developers can use these libraries to create and train deep neural networks for image and speech recognition, natural language processing, and other purposes.

C# is a programming language that is frequently used to create AI applications for a range of platforms, including Windows, Linux, and macOS. As a result, it is a versatile language for creating AI applications that can run on a variety of platforms.

Overall, C# has a broad collection of AI development features and tools, and its

flexibility to integrate with other technologies makes it a powerful language for building AI systems.

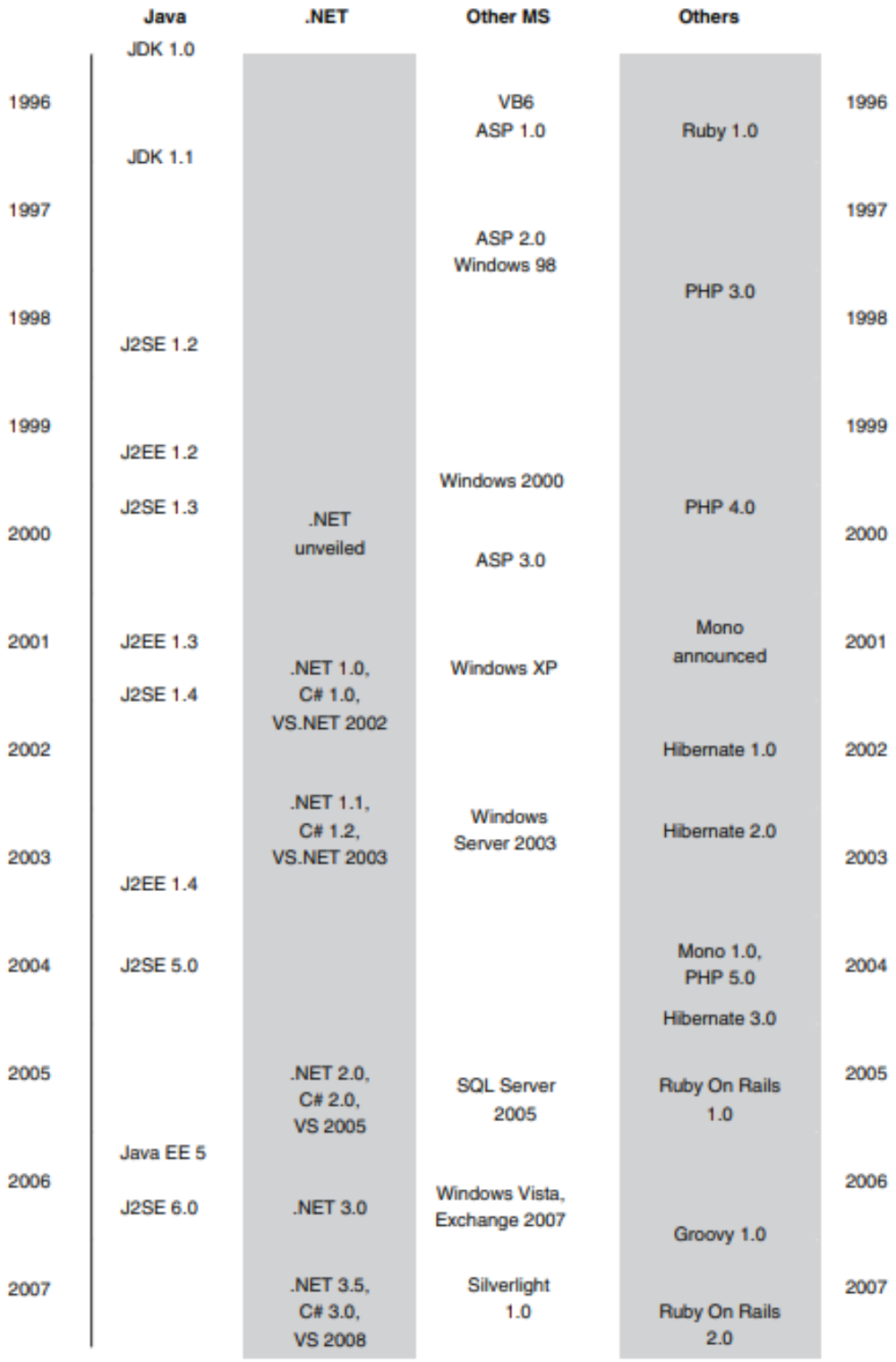


Figure 2.3: A brief history of C# (and related technologies) [11]

2.2.3 WPF

Windows Presentation Foundation is referred to as WPF. It is a graphical framework and subsystem that Microsoft developed for building visually beautiful user interfaces (UIs) in desktop programs that run on Windows. WPF is the replacement for Windows Forms and was introduced with the release of the.NET Framework 3.0.[4]

A few key concepts and features are:

- Data Binding: Developers can create bindings between UI elements and data sources using WPF's powerful data binding capabilities. It facilitates numerous binding types, including as one-way, two-way, and event-based bindings, making it simple to maintain UI and data synchronization.
- XAML: The UI elements, layout, and behavior are defined by WPF using the declarative language eXtensible Application Markup Language (XAML). Developers can segregate the application functionality from the user interface design using XAML.
- Vector Graphics: WPF utilizes vector graphics rendering, allowing for scalable and resolution-independent graphics and controls. This enables smooth rendering on screens with varying resolutions and sizes.
- UI Composition: Developers can organize UI elements, apply transformations, and animate properties using WPF's robust layout and composition mechanism. To arrange the controls, it supports a broad variety of layout panels, including StackPanel, Grid, and Canvas.
- Rich Media Support: Multimedia, including music, video, and 2D/3D graphics, is supported in great detail by WPF. It has controls for playing back media, support for media file types, and DirectX integration for cutting-edge graphics rendering.
- Styling and Templating: WPF supports a rich styling and templating model that enables developers to customize the appearance and behavior of UI elements. Styles define a consistent look and feel, while control templates define the structure and visual appearance of controls.
- MVVM Pattern: WPF works well with the Model-View-ViewModel (MVVM) pattern, which promotes separation of concerns and facilitates testability and maintainability of applications.

For creating aesthetically pleasing and engaging desktop applications on the Windows platform, WPF provides a flexible and strong foundation. It is a favorite among

Windows program developers because it offers a large selection of controls, layout possibilities, and customization options.

2.3 Overview of the MiniMax algorithm

The MiniMax algorithm is a decision-making algorithm commonly used in game theory, specifically in two-player zero-sum games. Game Theory is a field of mathematics and economics called game theory examines the tactical interactions of rational decision-makers. It offers a framework for examining and comprehending the actions and results of scenarios where the decisions made by one person or entity affect those of other people or other entities [6]. It aims to find the optimal strategy for a player by considering the choices of both players and minimizing the maximum potential loss (hence the name "MiniMax").

According to the algorithm, both participants are rational and will make decisions that will maximize their own rewards while minimizing those of their adversaries. It works by building a game tree that depicts all potential actions and results, then assessing each node of the tree according to the players' strategy. [21]

Here's a general overview of how the MiniMax algorithm works:

- Building the Game Tree: The algorithm begins by building a game tree that depicts all feasible moves and game outcomes. A game state is represented by each node of the tree, and potential paths between states are shown by the edges.
- Evaluation: Based on a utility function that assigns a value to each terminal node, the algorithm evaluates the terminal nodes of the tree (i.e., the nodes indicating game states where the game ends). This value often takes the form of a score or payout in zero-sum games.
- Backward Propagation: The algorithm moves values up the tree, taking into account each player's turn, starting at the terminal nodes. It calculates the value depending on the values of each non-terminal node's sibling nodes. It chooses the highest value among its children when it is the player's turn; when it is the opponent's turn, it chooses the lowest value.
- Making a decision: Assuming the opponent plays optimally, the algorithm chooses the action that, after values have been transmitted to the tree's root node, leads to the child node with the highest value.

The MiniMax algorithm selects the best course of action for a player in a two-person zero-sum game by employing this iterative evaluation, propagation, and decision-making process. It is frequently employed in games where the game tree may be effectively examined, such as chess, gomoku, and tic-tac-toe.

2.3.1 MiniMax

2.3.2 Brute force

The brute force method is a simplistic search technique that may be applied to situations involving selecting the best answer from a limited number of alternatives. It entails methodically producing and assessing each potential solution until the best one is identified.[8]

The benefit of the brute force method is that, if the ideal solution exists inside the search area, it will be discovered. The biggest disadvantage of this strategy is its high computational cost. The search becomes intractable for large issues as the search space expands because the number of potential solutions rises exponentially.

Pros :[19]

1. It lists all the possible solutions for the problem, therefore you always have the correct solution available.
2. It can be used in any domain of problems.
3. The ideal use for this method is solving small and simple problems.
4. It can serve as a comparison benchmark.

Cons:[19]

1. Its a really inefficient method, the complexity often exceeds O(N!).
2. The algorithms based on brute force are really slow.
3. Usually brute force algorithms don't have a good design and are not creative.

The MiniMax algorithm is a technique used in brute force search to identify the best move in two-player, zero-sum games like chess, checkers, and tic-tac-toe, gomoku. A zero-sum game is one in which one player's gain is the other's loss, and the sum of their payoffs is always zero. The MiniMax method evaluates all potential future movements from the current game state recursively up to a set depth or until the game is ended. At each stage, the algorithm believes that the opponent would play optimally in order to decrease the player's score while increasing their own. Until it reaches the necessary depth or the game is ended, the algorithm alternates between increasing the player's score and decreasing the opponent's score.[17]

MiniMax is useful in brute force search because it allows the algorithm to focus on the most promising steps while still evaluating all potential moves. The method can exclude vast areas of the search tree that are unlikely to result in a positive outcome for the player by assuming that both players play optimally. This decreases the size

of the search space and makes brute force search more realistic for determining the optimum step. MiniMax, of course, has several limits, especially in games with a large branching factor or inadequate information. [17]

The brute force algorithm is a method that, while it takes a long time to execute, and is inefficient, guarantees solutions for problems in any domain. It is useful for tackling smaller issues and offering a result that may be used as a baseline for assessing other design methodologies

2.3.3 Alpha-Beta

Introduction

Alpha-beta pruning is a search algorithm is utilised in game theory to optimize the efficiency of MiniMax algorithms, which are used to determine the optimal move in a two-player game where it serves to speed up game searching without loss of information. The algorithm moves through the tree in a predetermined direction, say from left to right, skipping any nodes that can no longer affect the MiniMax value of the root. This process yields the root's MiniMax value.[14]

The History Heuristic

A problem space is a collection of states and operators that map to one another, according to Newel1 and Simon [13]. One repeatedly applies operators to states, starting from an initial state of knowledge, in search of a desired state.

Operators can be thought of as parameterized functions, with the parameters defining the precise actions the operator should take in relation to the state. States and movements are both operators in the context of game trees. The from-square and to-square of the move are two parameters that can be seen as functions of the move operation.

Making a move from one position results in a successor position that shares many of the same attributes as its predecessor for many types of game trees. A move deemed good in the prior position will likely still be good since the situation's key elements, which determine what the good movements are, have not altered considerably. Since the two views are comparable, one may adopt a complex strategy and employ analogies to demonstrate the correspondence. [18]

For example, considering a tree build by an algorithm. A move X may be the best in one position, later in the search tree a similar position can appear, maybe just a little bit different. This little difference may not alter the move X from still being the most valuable move, that means that X may be considered the first move for this position, improving the ordering of moves, increasing the chances of a cutoff to occur.

In Fig 2.4 we can see how the heuristic can affect searching.

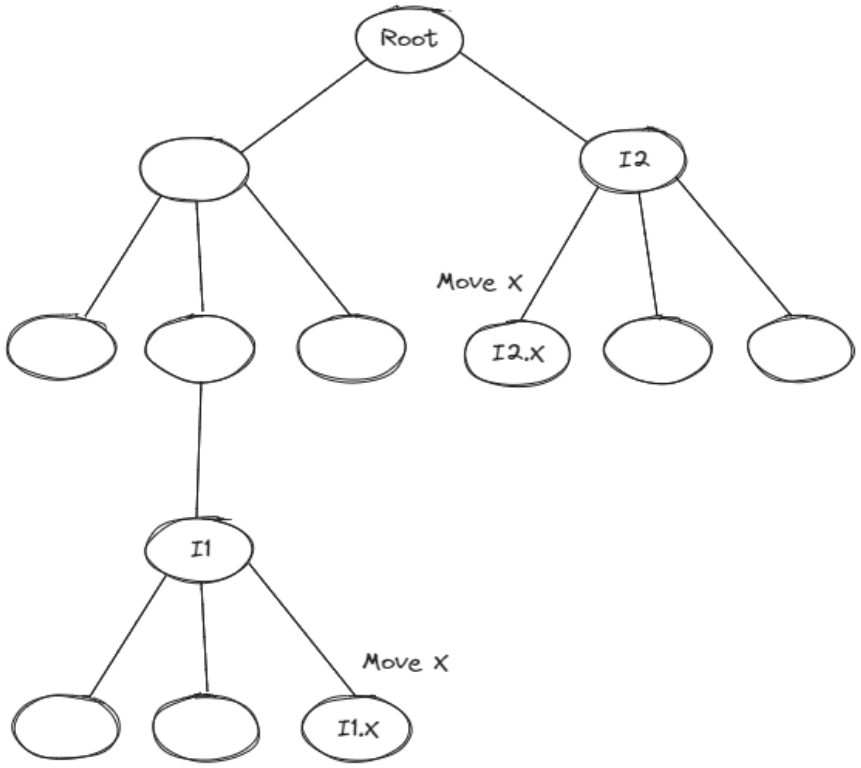


Figure 2.4: History heuristic example

Conditions for Alpha-Beta pruning

- Along the Maximizer path, the best option or highest value we can discover is Alpha.
- On the Minimizer path, the best option or the lowest value is Beta.
- An important condition for this is that alpha is bigger or equal to beta
- Alpha will be updated only when it's MAX's time and Beta when it's MIN's turn.
- The MIN player will update the beta values and the MAX player will update the alpha values.
- The upper nodes receive the values from the lower nodes, not the actual alpha, beta values.
- The values of Alpha and Beta are passed only to child nodes.

Pseudo-code for Alpha-Beta pruning [5]

```
function alphabetapruning(node,depth,alpha,beta,maximizingPlayer) is
    if depth == 0 or node is terminal then
        return the heuristic value of node
    if maximizingPlayer then
        value := 0
        for each child of node do
            value := max(value,
                alphabetapruning(child, depth-1, alpha, beta, FALSE))
            if value > beta then
                break (* beta cutoff *)
            alpha := max(alpha, value)
        return value
    else
        value := 0
        for each child of node do
            value := min(value,
                alphabetapruning(child, depth - 1, alpha, beta, TRUE))
            if value < alpha then
                break (* alpha cutoff *)
            beta := min(beta, value)
        return value
```

Example of Alpha-Beta pruning

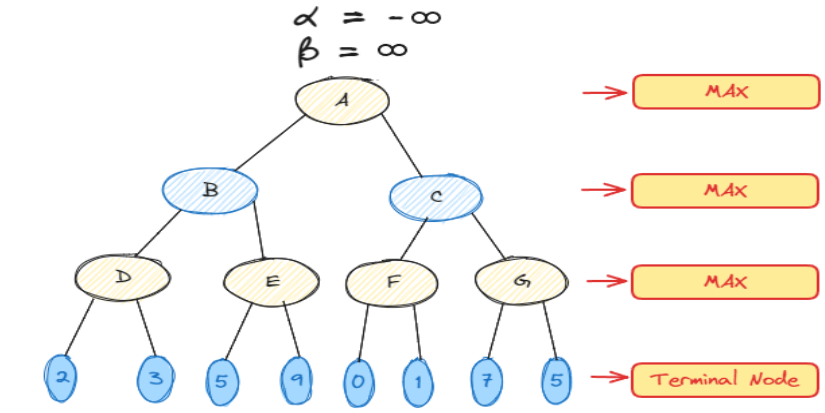


Figure 2.5: Step 1

Alpha is less than beta so we don't prune it, that means that it's MAX turn, such that at node D, alpha is computed. Alpha will be max(2,3).

The next move will be on node B and its turn for MIN. So, the value of alpha beta will be min(3, ∞). So, node B will be alpha = -∞, and beta will be 3.

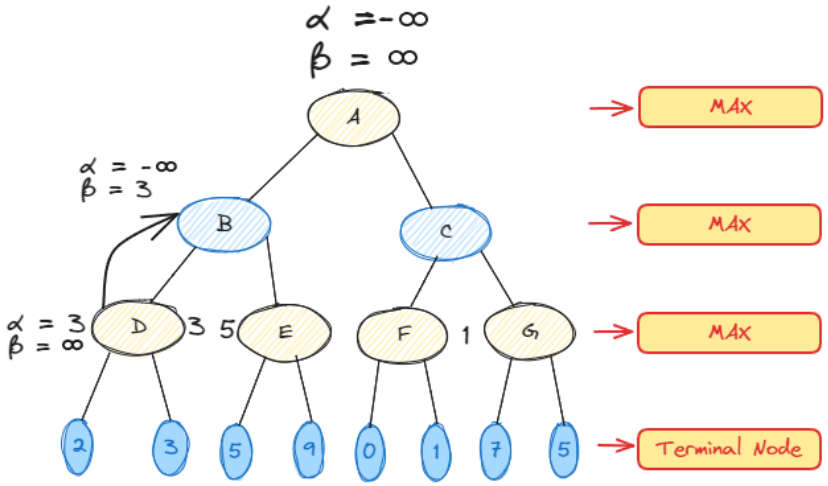
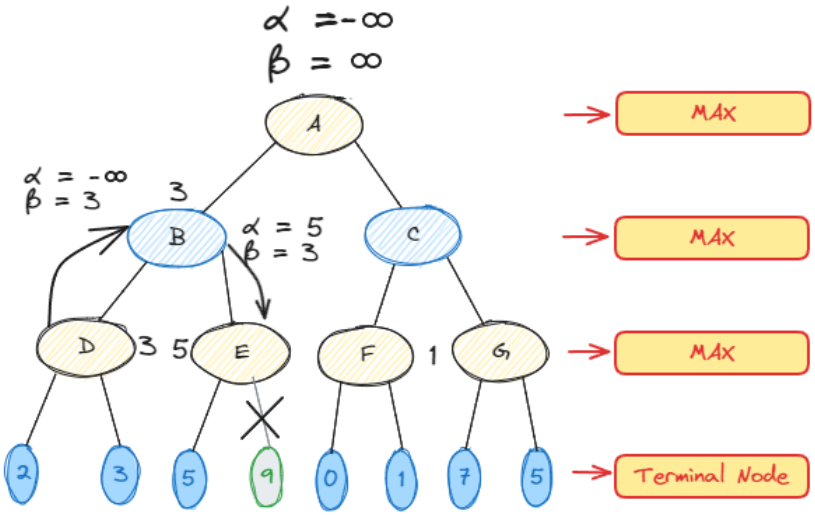


Figure 2.6: Step 2

It's MAX turn, so now we will look for MAX. Alpha at E is -∞ and we now compare it with 5, such that MAX(-∞, 5) will be 5, so at E, alpha is 5 and beta 5. As we can see alpha is the same as beta which satisfies the pruning condition and the value of



node E will be 5.

Figure 2.7: Step 3

Now the algorithm comes to node A from B, and now the value of alpha and beta in A will be (3, ∞) and they will go to node C. The F node alpha value will be compared to the left branch , the result being 0, so we make the MAX(0,3) and then compare with the other child MAX(3,1) = 3, but the node value of F will become 1.

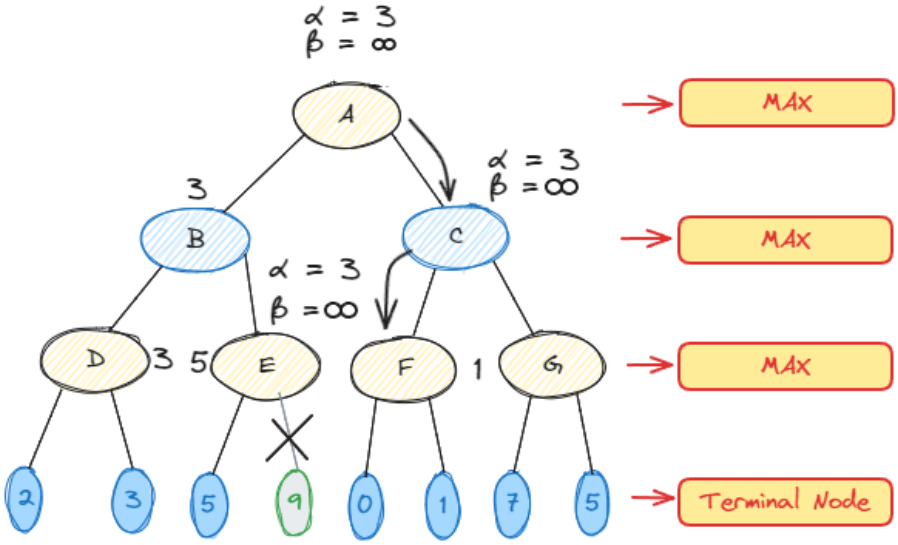


Figure 2.8: Step 4

F returns the value 1 to C and will compare the beta value, now it's MIN's turns, MIN(∞, 1) will be 1. SO node C has alpha = 3 and beta = 1, and this satisfies the pruning condition. So the successor of node C, that being G will not be computed.

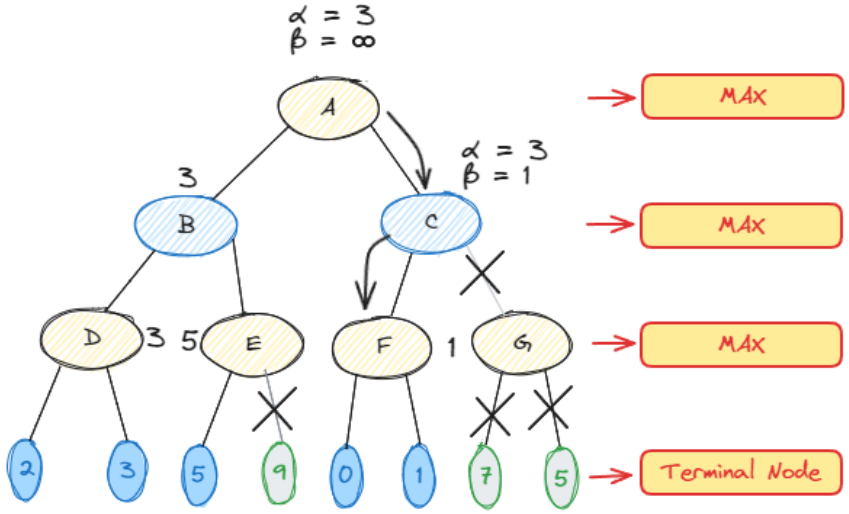


Figure 2.9: Step 5
The node value will now be returned to A by C, and A's optimal value will be MAX (1, 3), which is 3.

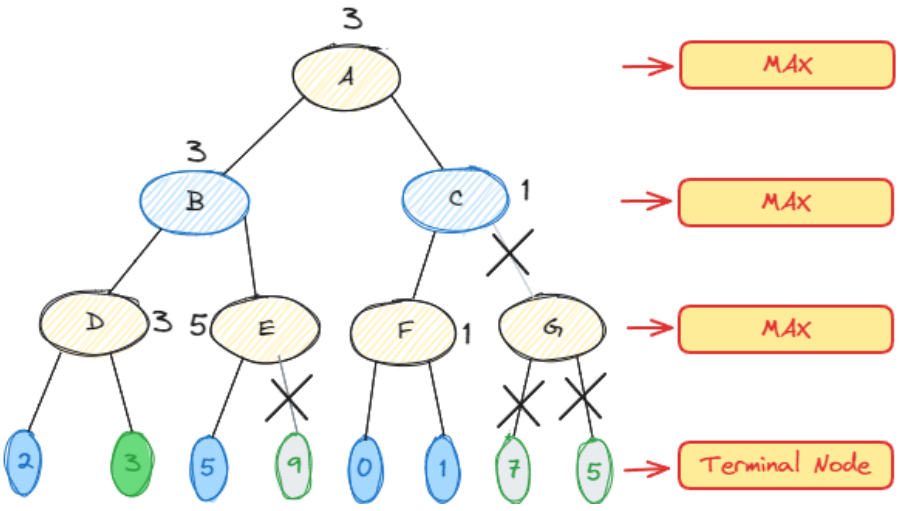


Figure 2.10: Step 6
The final tree is showing the nodes which are computed and the ones that are not, the optimal value in the end being 3.

Chapter 3

Methodology

3.1 Implementation

Gomoku, commonly known as Five in a Row, is a strategy board game in which players must arrange pieces on a grid in order to make a row of five consecutive pieces. The goal of this project is to construct an AI player for Gomoku using the MiniMax algorithm, a popular decision-making method in artificial intelligence. The MiniMax algorithm considers all potential game states and chooses the best move.

The project's main goal is to develop the MiniMax algorithm in Python and C# for the game of Gomoku and evaluate and contrast their performance and efficiency.

3.1.1 Steps

1. Understanding the Gomoku and MiniMax algorithms.
2. Using object-oriented programming (OOP) ideas and data structures, implement the MiniMax algorithm in Python and C#.
3. Using WPF, create a graphical user interface (GUI) for the Gomoku game.
4. Evaluating the MiniMax algorithm's performance and efficiency in each language by analyzing aspects such as execution time, memory utilization, and developer experience.
5. Analyzing the results and forming conclusions.

MiniMax will be implemented in Python and C#, with performance enhancements such as alpha-beta pruning. To represent the game state and make the best movements, OOP ideas and data structures will be used. To provide an interactive interface for playing against the AI, the Gomoku game's GUI will be built using WPF.

The project will let people play a full game versus the AI, player vs player, or let the AI's battle each other. The user interface will be simple and straightforward, you put a piece using your mouse and if you are playing versus the AI he will make his move as fast as possible, this process will be repeated until a draw, win, or defeat.

A WPF .NET project will be used for the user interface and the C# implementation of the algorithm, the python side of the code will be called inside this project. The user interface will be straightforward because computing moves and comparing the languages is the main goal, but nevertheless it will allow users to place pieces on the board in line with the rules set of Gomoku.

Gomoku may seem simple at first, but it's one of the most complex games you can play. The possible moves and strategies you make can become really complex, scaling with size the board, the bigger the board, the more complex it gets.

The majority of moves include placing piece X to a position P such that it gets you closer to making 5 in a row, or restricting your opponents moves. Also my algorithm has a rule such that if you are restricted at both ends of your 5 in a row pieces, you don't win.

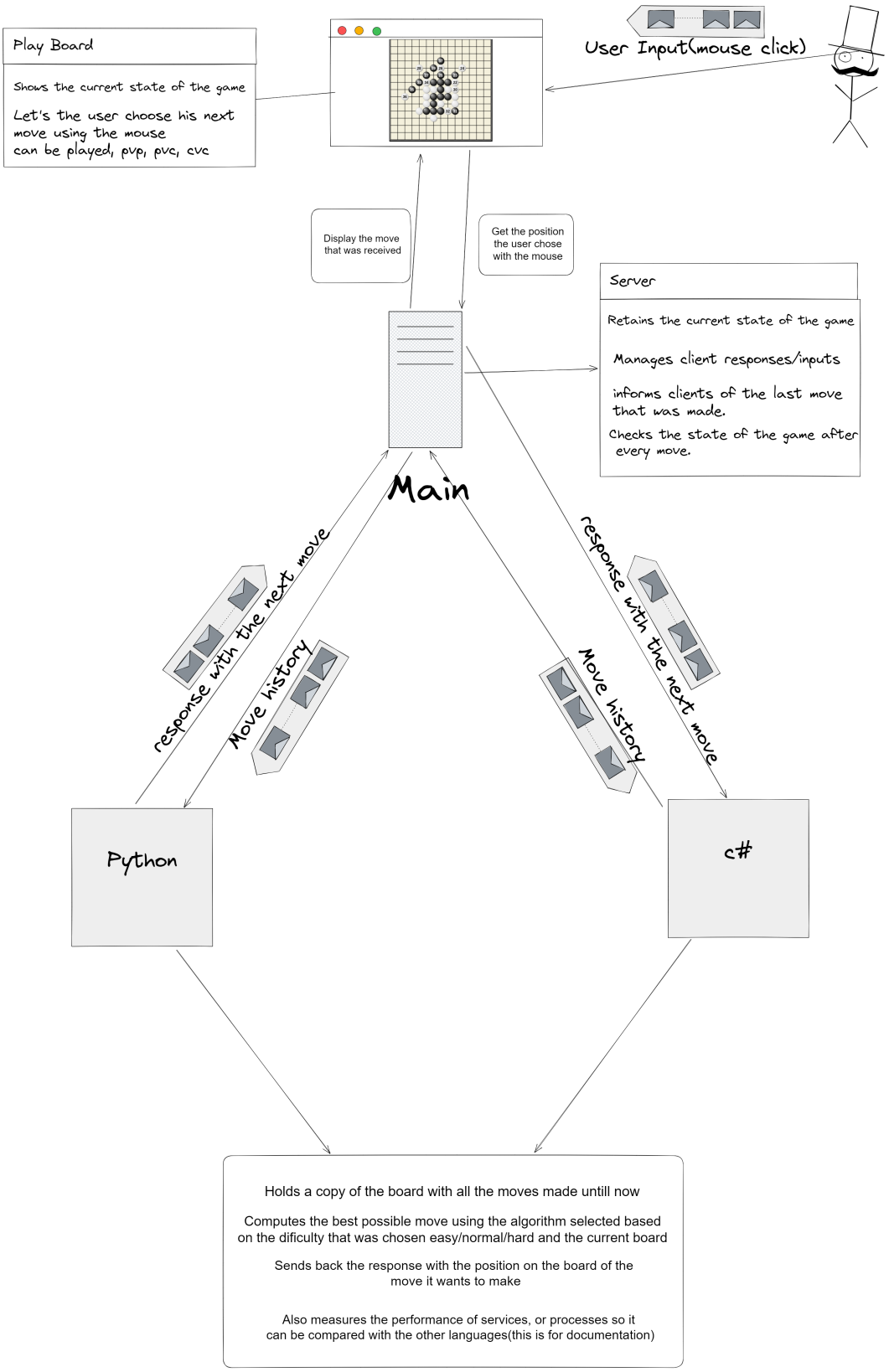


Figure 3.1: Application Diagram

3.1.2 Application Presentation

The application has a minimalistic design containing a white board with tiles on which the moves will be made and a side panel that features easy-to-use toggle buttons with white text that control who plays the game, either the AI's or human and a few other helpful commands.

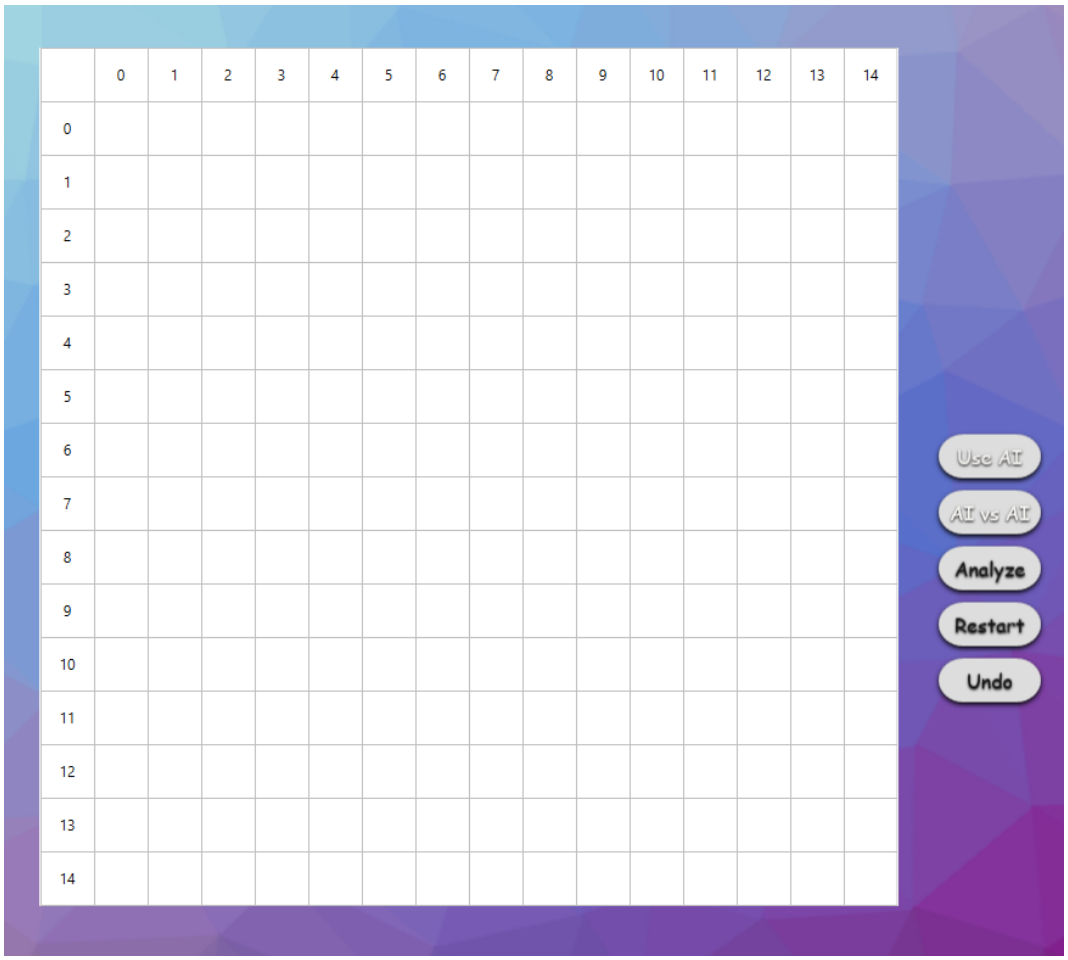


Figure 3.2: Application Presentation

Use AI

The Use AI is a toggle button that when it's activated you play versus the AI and when it's disabled you play versus a human, you can activate/deactivate it whenever you want over the course of the game.

AI vs AI

The AI vs AI toggle button when activated makes the Python implementation of the AI play againts the C# implementation. This can be turned on/off at any time over the course of the game.

Analyze

This option is a really helpful one if you need help in making your first move, when it's your turn and you press the button, it will use the AI to calculate the best move that you can make and will show you the option.

Restart

When pressing this button the board will get cleared and the game will be reset.

Undo

If you ever put a piece in the wrong place or want to undo a move you made, pressing the undo button will revert the last move that was made.

When the game is finished, a message will be displayed on the screen with the winner or the "Tie" message for when the game ends in a draw.

3.1.3 Moves

The game is rich in strategy and offers numerous possibilities, making it an engaging and challenging experience.

Here are some examples of move types in Gomoku:

1. **Opening Move:** being the first move of the game it is usually placed in the centre of the board so you can establish an initial position and gain flexibility.
2. **Offensive move:** this move aims to create a winning formation or potential winning combination, being placed in a strategic position that is threatening to the other player.
3. **Defensive move:** a move made to block the opponent's winning formation or disrupt their strategy.
4. **Forking move:** an advanced move that involves placing a stone in a position that simultaneously threatens multiple winning formations.
5. **Connecting Move:** a move made to strengthen their position by connecting multiple pieces of the same color.
6. **Sacrifice move:** sacrificing a stone can help you gain advantage in another part of the board. They are usually made to distract your opponent's attention.
7. **Luring move:** a move made to force your opponent to respond in a specific way, giving you control over the board.

8. **End Game move:** as the game is close to the end and the board becomes more crowded, end game moves focus on solidifying positions and securing winning formations.

3.1.4 MVVM

What is MVVM?

Model-View-ViewModel is referred to as MVVM. By adding a layer called ViewModel, it is a software architecture design that divides the user interface (View) from the underlying data (Model). Particularly in the context of desktop and mobile apps, MVVM is frequently employed in the construction of contemporary user interfaces.[3]

Model

The Model represents the data and business logic of the application. It could include data structures, database operations, network requests, and any other data-related functionality.

View

The application's visual representation and user interaction are handled by the View. It could be a page, a window, or a user interface screen. Typically, the View is inactive and simply shows the data that the ViewModel provides.

ViewModel

Between the View and the Model, the ViewModel serves as a bridge. In addition to handling user interactions and updating the Model as necessary, it presents data and commands to the View. The ViewModel concentrates on delivering the necessary information and actions that the View requires, while being intended to be independent of the particular UI framework.

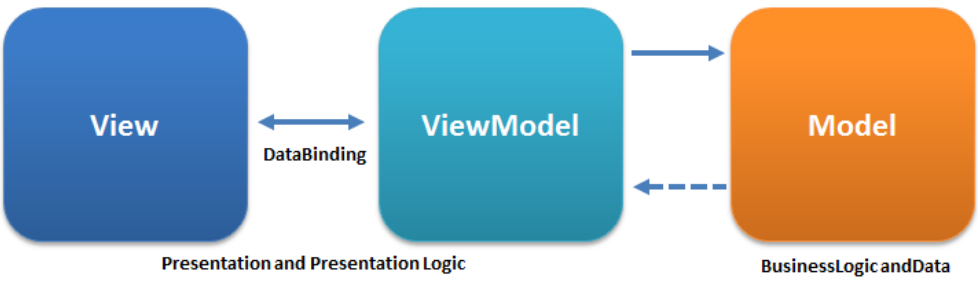


Figure 3.3: MVVM Pattern [22]

Advantages of MVVM

1. **Easier to develop:** It is possible to have various teams work on various components at the same time by separating the logic from the view. While programmers are constructing the logic (ViewModel and Model), a team of designers can concentrate on the user interface (UI).

2. **Easier to test:** The user interface (UI) of an application is among the most challenging components to test. Developers can build tests for the ViewModel and Model without using the View because they are totally independent of the View.

3. **Easier to maintain:** The code is made easier and cleaner by separating the distinct parts of the application. Because of this, it is considerably simpler to comprehend and maintain the application code. Where to use new features and how they relate to the pattern already in place are clearer. Implementing additional architectural patterns (such as dependency inversion, services, and others) is also made simpler with an MVVM.

Disadvantages

1. **Complexity:** When it comes to designing straightforward user interfaces, MVVM is overkill. It can be challenging to construct the viewmodel with the appropriate level of generality while working on larger projects.

2. **Difficult to debug:** Data binding is declarative, which makes it potentially more difficult to debug than conventional, imperative programming.

TileViewModel

The TileViewModel is responsible for representing the state and behaviour of an individual tile on the game board.

- `public bool IsHighlighted`

  Gets or sets a boolean value indicating whether the tile is currently highlighted. When the value changes, it also checks if the tile is selected (IsSelected == true) and automatically unselects it if IsHighlighted is set to false.

- `public bool IsSelected`

  Gets or sets a boolean value indicating whether the tile is currently selected.

- ```
  public Piece Piece
  {
    get => Tile.Piece;
    set
    {
      Tile.Piece = value;
      NotifyPropertyChanged();
    }
  }
  ```

  Gets or sets the Piece value of the tile. It retrieves the Piece property from the associated Tile object and notifies any subscribers of the property change using the NotifyPropertyChanged method (inherited from ViewModelBase).

BoardViewModel

The BoardViewModel class is responsible for managing the presentation logic and interaction between the game board, the graphical user interface and the underlying game logic.

- ```
  public TileViewModel this[int x, int y] => TileVMs[x, y];
  ```

  Allows accessing the TileViewModel objects at specific coordinates on the board. It returns the TileViewModel at the given (x, y) position.

- ```
  public void Clear(int x, int y)
  {
    var tileVm = this[x, y];
    tileVm.Piece = (Piece)Pieces.None;
    HighlightedTiles.Remove(tileVm);
  }
  ```

  Clears the tile at the specified (x, y) position by setting its Piece property to Pieces.None and removing it from the HighlightedTiles collection.

- ```
  public void Highlight(params ICoordinates[] positionals)
  ```

  Adds the TileViewModel objects at the specified positions to the HighlightedTiles collection, highlighting them.

- ```
  public void Select(ICoordinates position)
  ```

  Marks the tile at the specified position as selected by setting its IsSelected property to true.
  ```

- ```
  public void Set(int x, int y, Piece piece)
  {
    var tileVm = this[x, y];
    tileVm.Piece = piece;
    HighlightedTiles.Add(tileVm);
  }
  ```

  Sets the tile at the specified (x, y) position to the given piece, updates its Piece property, and adds it to the HighlightedTiles collection.

- ```
  private void GameBoardChanged(object sender, BoardChangedEventArgs e)
  ```

  Event handler that responds to changes in the game board. It clears highlighted tiles, removes any tiles that have been removed from the board, and adds new tiles to the board while updating their presentation accordingly.

- ```
  private void HighlightedTilesCollectionChanged(
      object sender,
      NotifyCollectionChangedEventArgs e)
  {
      switch (e.Action)
      {
          case NotifyCollectionChangedAction.Remove or
          NotifyCollectionChangedAction.Reset:
          {
              if (e.OldItems != null)
                  foreach (TileViewModel item in e.OldItems)
                  {
                      item.IsHighlighted = false;
                  }

              break;
          }
          case NotifyCollectionChangedAction.Add:
          {
              if (e.NewItems != null)
                  foreach (TileViewModel item in e.NewItems)
                  {
                      item.IsHighlighted = true;
                  }
              break;
          }
      }
  }
  ```
  ```

Event handler that responds to changes in the HighlightedTiles collection. It updates the IsHighlighted property of the affected TileViewModel objects based on the action performed on the collection.

Overall, the BoardViewModel class acts as a mediator between the game logic and the graphical user interface, providing functionality to manipulate and update the game board's presentation, handle highlighting of specific tiles, and respond to changes in the board or the highlighted tiles.

3.1.5 Exploring Important Classes and Their Roles

Board

The Board class serves as the underlying data structure for managing the game board in the Gomoku game. It provides methods to access and manipulate individual tiles on the board, clone the board, and iterate over tiles in various directions.

Properties: The Board class has two read-only properties, Width and Height, that provide the dimensions of the game board.

Indexer:

```
public Tile this[int x, int y] => _tiles[x, y];
```

The Board class defines an indexer that allows accessing individual tiles on the board using the syntax board[x, y]. It returns the Tile object at the specified coordinates.

The IterateTiles method iterates over a 2D grid of tiles based on a given starting position (x and y coordinates), a specified direction (Directions enum), a provided predicate (Predicate¡Tile¿) and an optional flag to determine whether the iteration should include the starting position, the method has a switch case which has all the relevant cases for direction Left, Right, Up, Down, DownRight, DownLeft, UpRight, UpLeft. Within each loop, the code doesn't perform any action other than iterating through the tiles until the predicate returns false or the boundaries of the board are reached. The intent is to allow the caller to provide the desired logic within the predicate function. An example of one of the switch case can be found bellow.

```
var startingOffset = iterateSelf ? 0 : 1;
        case Directions.Left:
            for (var i = x - startingOffset;
                i >= 0 && predicate(_tiles[i, y]);
                i--){}
            break;
```

In our case the predicate determines when the iteration should continue or stop while keeping track of the different types of tiles, such as blocked tiles, consecutive tiles, blank spaces etc. encountered during the iteration of the line in the specified direction.

```
if (count++ == maxTile)
  {
      return false;
  }
  if (t.Piece.Type == piece)
  {
      if (blank > 0)
      {
          chainBreak = true;
      }
      tiles.Enqueue(t);
      sameTiles.Enqueue(t);
      return true;
  }
  else if (t.Piece.Type == Pieces.None)
  {
      if (blank++ == blankTolerance)
      {
          return false;
      }
      tiles.Enqueue(t);
      blankTiles.Enqueue(t);
      return true;
  }
  else
  {
      tiles.Enqueue(t);
      blockTiles.Enqueue(t);
      return false;
  }
```

This predicate is really helpful because it provides all the necessary information such as blocked tiles, free tiles, consecutive tiles etc. on the provided direction for computing the best move while also being easy to read and it keeps the code cleaner.

LineBasedCandidates

```
public IEnumerable<ICoordinates> Search(Game game,
    int maxTile = 2, int blankTolerance = 1)
{
    var placedTiles = game.History;
```

```
    var tiles = new HashSet<Tile>();
    foreach (var tile in placedTiles)
    {
        var orientations = new[]
        {
          Orientations.Horizontal, Orientations.Vertical,
          Orientations.Diagonal, Orientations.SecondDiagonal
        };

        foreach (var orientation in orientations)
        {
            foreach (var t in
              OrientedlLine.FromBoard(game.Board,
                tile.X, tile.Y,
                (Piece)Pieces.None, orientation,
                maxTile: maxTile, blankTolerance: blankTolerance)
              .GetSameTiles())
            {
                tiles.Add(t);
            }
        }
    }
    return tiles;
}
```

The method Search is used to find the possible moves that can be made by the AI but instead of finding all the moves on the board, we only search the ones that are in a line with another move that was made, this improves the search in the early stages of the game.

The method OrientedLine.FromBoard iterates through all the directions possible starting from the placed tile using the predicate and IterateTiles method that we defined earlier.

3.1.6 Evaluation and Results

The MiniMax algorithm's performance and efficiency will be assessed in each language using numerous test scenarios to simulate different gameplay circumstances. Execution time, memory consumption will be measured. The AI's performance against human players will be measured in terms of win, loss, and draw rates.

The outcomes will be documented and analyzed. The MiniMax algorithm will be expected to perform well in both languages with differences in execution time and memory usage across Python and C#. The gameplay experience shouldn't be different between the implementations, with the GUI providing an intuitive interface for playing versus the AI.

3.2 Languages comparison criteria

To compare the implementation of the MiniMax algorithm in Python and C# in the context of Gomoku we will compare various criteria, including performance, language features, memory and overall responsiveness.

Performance

Comparison of the execution speed and efficiency of the implementations. Measure and compare factors such as the average time taken to compute moves.

Resource Usage

Assess the memory consumption and resource utilization of the application. Analyzing factors such as memory usage.

Language Features and Libraries

Compare the ease of implementation, code readability, and maintainability of the implementations in Python and C#.

Development Productivity

Consider the development experience and productivity when working with each language. Evaluation factors such as the availability of development tools, ease of debugging, and overall developer satisfaction during the implementation process.

Community and Documentation

Assess the size and activity of the respective language communities. Evaluating the availability and quality of documentations, tutorials, and overall resources.

3.3 Testing approach

In today's technology-driven world, where applications play a vital role in various aspects of our lives, ensuring their quality and reliability has become paramount. Testing, as a fundamental component of the software development lifecycle, holds the key to uncovering defects, mitigating risks, and delivering exceptional user experiences. This text explores the critical importance of testing in the development process, emphasizing its role in enhancing application performance, reducing errors, and instilling confidence in both developers and end-users.

Testing serves as a rigorous and systematic approach to validate the functionality and performance. It involves executing predefined test cases, analyzing results, and identifying potential issues or vulnerabilities. By subjecting the application to a variety of scenarios and conditions, testing ensures that it meets the expected requirements, works as intended, and delivers a seamless user experience.

Some testing approaches will be:

1. **Integration Testing:** Integration testing is crucial to ensure that different components of an application work together seamlessly. In the context of my Gomoku implementation, I performed integration tests to validate the communication and coordination between the user interface, game logic, and the MiniMax algorithm implementation in both languages.

2. **Usability Testing:** Usability testing focuses on evaluating the user experience and interface of an application. In this case, I conducted tests to assess the ease of navigation, clarity of instructions, intuitiveness of the game controls, and overall satisfaction of players using as data the experiment that I will be mentioning in the next section. Collect feedback from users to identify any areas of improvement and make the necessary adjustments.[12].

3. **Functional Testing:** This type of testing focuses on verifying whether your application meets the specified functional requirements. It involves testing various features and functionalities of the game, such as placing markers, detecting winning conditions, handling invalid moves, and ensuring proper interaction with the MiniMax algorithm.

   For this type of testing I've conducted an experiment. I gathered 5 people and gave them the task to play 10 games againts the AI and provide me feedback with the number of times they managed to beat it and what bugs they find. This helped me a lot, even though the people that were part of the experiment are beginners at the game and I was expecting them to not have a great win rate againts the AI, they provided me with good feedback that made me improve the algorithm, for example one of the persons found a bug when the number of pieces in a row was more than 5, for example you manage to win the game with 6 pieces in a row, my GameOver() function would not recognize it because the check that I made was that the number of consecutive pieces needs to be equal to 5, which I changed in bigger or equal to 5, so this bug was fixed.[9]

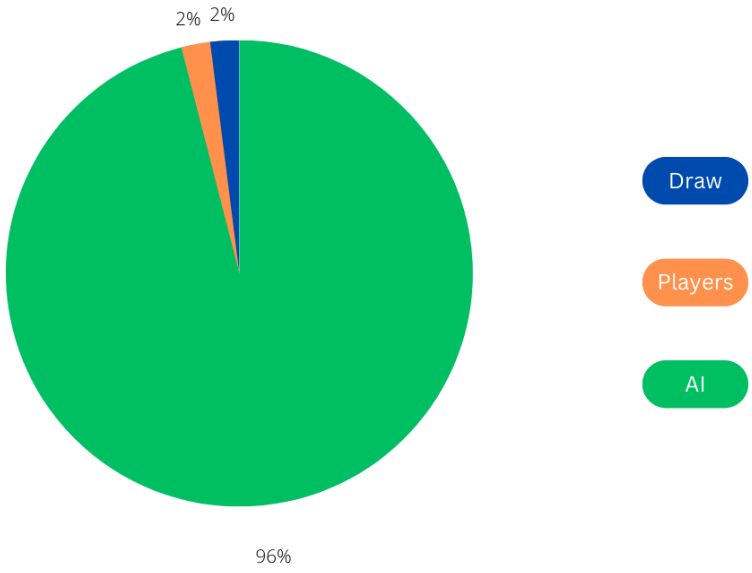


Figure 3.4: Win percent of AI in 50 games againts 5 different beginner players

To further test the application and implementations, I also conducted a smaller experiment, putting the algorithms to play each other for 100 games to determine which programming language, Python or C#, has a higher win rate in Gomoku, but it is important to note that the outcome of a game can be influenced by various factors.

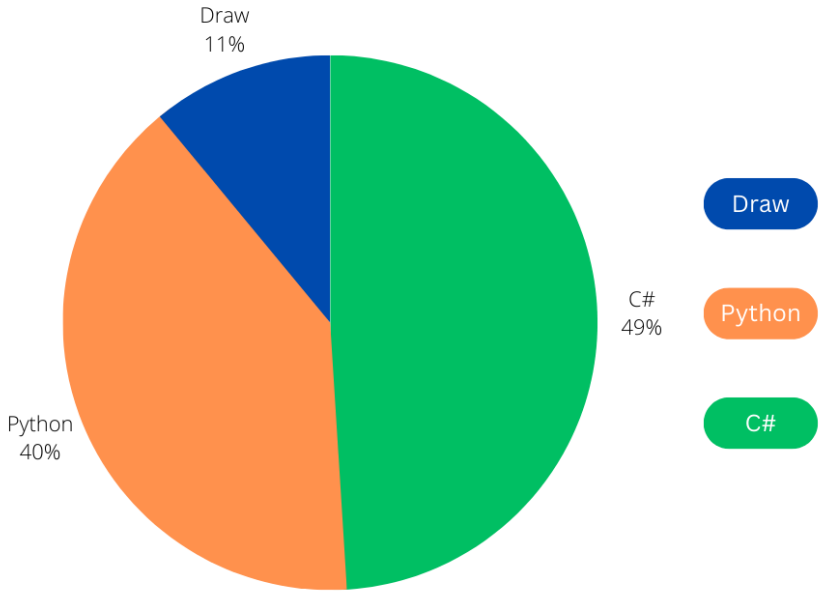


Figure 3.5: Python vs C# win rate over the course of 100 games

Chapter 4

Results and Analysis

4.1 Technical comparison

4.1.1 Performance

When it comes to implementing the MiniMax algorithm for the game of Gomoku, the choice of programming language can significantly impact the execution speed and efficiency of the algorithm. This papers dives into a comparative analysis of the execution speed and efficiency between C# and Python MiniMax implementations, with a specific focus on factors such as the average time taken to compute moves.

The MiniMax algorithm is a widely-used approach for decision-making in games, including Gomoku. It works by evaluating possible future game states and selecting the move that maximizes the player's chances of winning or minimizes the opponent's chances. However, the efficiency of the algorithm heavily relies on the speed at which it evaluates these moves.

C# and Python, as popular programming languages, offer distinct characteristics that can impact the execution speed and efficiency of MiniMax implementations. C# is known for its compiled nature, which allows for optimization during the compilation process. The compiled code can often lead to faster execution times, making it well-suited for performance-critical applications. On the other hand, Python is an interpreted language that offers flexibility and ease of use but may suffer from slower execution speeds compared to compiled languages.

In the context of Gomoku, where the search space can be large due to the game's branching factor, the choice of programming language can significantly affect the average time taken to compute moves. This metric reflects the efficiency and responsiveness of the algorithm in analyzing the game state and selecting the optimal move. Therefore, measuring and comparing the average time taken by each implementation provides valuable insights into their respective computational capabilities and their suitability for real-time decision-making.

Furthermore, the availability of libraries and ecosystem support can significantly influence the performance of MiniMax implementations. C# benefits from a robust ecosystem, including the .NET Framework and .NET Core, which provide optimized algorithms and data structures for efficient computation. Python, renowned for its extensive library collection, offers tools such as NumPy for efficient numerical computations potentially enhancing the performance of the MiniMax algorithm, which has been used in the implementation of the python version of the algorithm, some examples of the use of this in our implementation are:

- `numpy.fliplr(board)`

  This function is used to flip an array horizontally (left to right). It returns a new array with the columns of values reversed.

- `numpy.diag(array, k=i)`

  This function extracts the values along a specified diagonal of a 2D array. The array argument represents the input array, and k specifies the diagonal offset. Here, i is a variable representing an integer value.

Using the NumPy library can improve the efficiency of doing computations on games like Gomoku where the search space can get really large because of the branching factor.

Move computation comparison

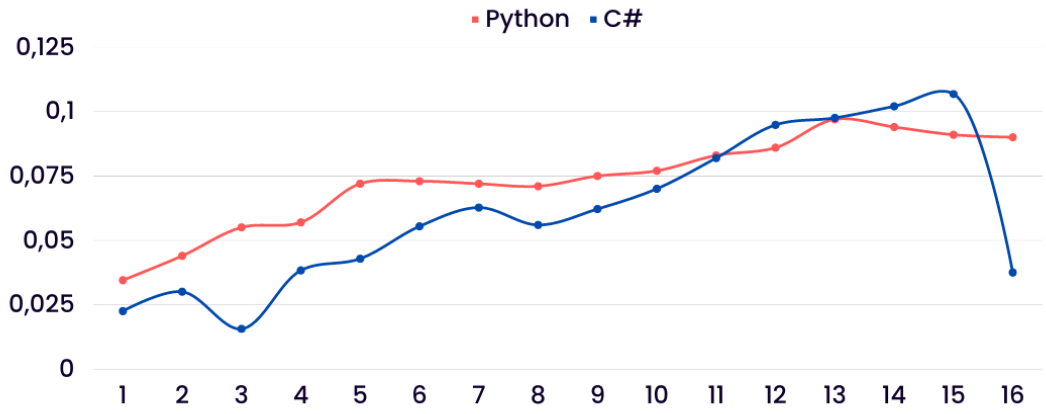


Figure 4.1: Time taken to compute a move in a short 16 move game

If we examine the graph, we observe certain patterns and trends that shed further light on the performance comparison between the python and C# implementations.

We can see that the red line which represents the Python implementation has a gradual increase over time. This suggests that as the algorithm explores deeper and more complex game states, the computational effort required to evaluate possible moves also increases. The fluctuations in execution times can be attributed to variations in the number of possible moves and the complexity of the game state encountered in each scenario. Despite the increasing trend, the execution times for the Python implementation remain within an acceptable range, indicating that the Python implementation can handle move computation effectively for most game scenarios.

In contrast, the blue line which represents the c# exhibits lower execution times compared to Python. The C# implementation consistently performs more efficiently, with shorter execution times across the board. This demonstrates the advantage of C#'s compiled nature and optimized execution, resulting in faster decision-making and more responsive gameplay. The execution times for C# also exhibit fluctuations, which can be attributed to variations in the number of possible moves and the complexity of the game state, similar to the Python implementation. However, even with these fluctuations, the C# implementation maintains a clear advantage in terms of overall execution speed.

The stark performance difference between the Python and C# versions can be attributed to a number of factors. As a statically-typed language, C# enables improved memory management and compilation optimization, which results in speedier execution speeds. C# has the added benefit of a strong ecosystem of libraries and frameworks designed for game development; these tools can further enhance performance. Python's dynamic type and interpreted nature, however, incur additional overhead, leading to noticeably slower execution times.

It is important to keep in mind that the graph 4.1 reflects a particular set of situations or iterations, and the performance comparison between Python and C# may change based on the complexity of the encountered game states. The two implementations performance differences might not be as obvious in simpler circumstances. The effectiveness of the C# implementation, however, becomes more obvious as the game state complexity increases.

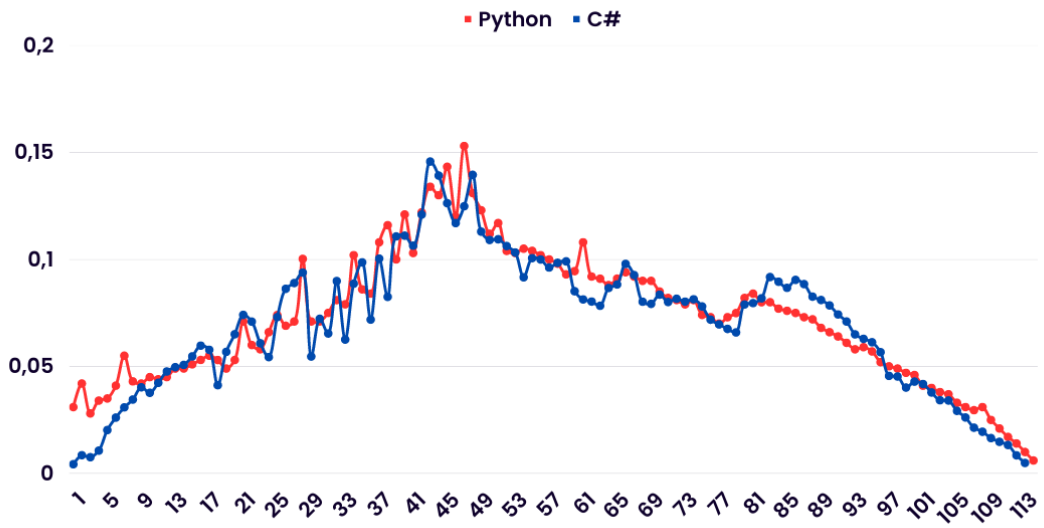


Figure 4.2: Time taken to compute a move in a game that ended in a draw

The graph 4.3 represents the performance comparison between the implementations for a game that ends in a draw. Upon analyzing the graph we can see the same pattern as for the graph 4.1, the execution times varying throughout the game, hitting it's peak when the game comes close to the middle, because there are more options to explore that could be potential moves. As the game goes on and nobody wins we can see a drastic decline in execution time because the number of moves that can be made gets smaller and smaller, thus the execution time starts to stabilize towards the end of the game. This also indicates that both implementations are capable of efficiently exploring the game tree and evaluating the available moves to determine a draw outcome. However we have to consider other factors that affected the fluctuation of the execution time such as when a good move was found first or when one of the implementations needed to block the other opponent so it doesn't win.

4.1.2 Language Features and Libraries

When comparing the ease of implementation, code readability, and maintainability of the implementations in Python and C#, several factors come into play. Here's a breakdown of each aspect:

- **Ease of Implementation:**

  Python is often praised for its simplicity and ease of use. Its clean and expressive syntax allows developers to write code quickly and efficiently. Python's extensive standard library provides a wide range of modules and functions that simplify the implementation of complex algorithms, including the MiniMax algorithm.

Additionally, Python's dynamic typing eliminates the need for explicit variable declarations, reducing the cognitive load during implementation. The availability of numerous third-party further enhances the ease of implementing various functionalities.

C# is a statically-typed language that requires explicit type declarations. While this can introduce additional complexity during implementation, C# offers a rich set of language features and libraries specifically designed for game development and algorithmic tasks. C# benefits from its integration with the .NET framework, providing extensive functionality and tools that enhance development efficiency. The strong typing in C# promotes type safety and early error detection, resulting in more robust implementations. But overall the vast and specific documentation that C# has makes it easier to implement.

- **Code Readability:**

  When we talk about code readability and ease of understanding, Python always comes to mind, because of its readability and clean syntax, the use of indentation as a structural element promotes well-structured and readable code. Python's focus on code clarity allows developers to express complex ideas concisely, making it easier to understand.

  On the other hand, C# follows a more explicit syntax with the use of semicolons and curly braces to define code block. While this can make the code appear more messy that Python, it also enforces a clear structure and facilitates code navigation, supporting various naming conventions and coding styles.

- **Maintainability**

  Python promotes maintainable code through its focus on simplicity, modularity, and code reusability. The language encourages the creation of small, reusable functions and classes, which can be easily tested and modified. Python's docstring conventions and support for unit testing enables developers to document and test their code, making it easier to maintain over time.

  C# emphasizes maintainability through its strong type system and OOP principle. The use of classes, interfaces and namespaces in C# facilitates code organization, reusability and encapsulation, while also providing robust debugging and profiling tools, making it easier to diagnose and fix issues, which enhance maintainability and productivity.

4.1.3 Resource Usage

To asses the memory consumption and resource utilization of the application, I used a tool from Jetbrains called DotMemory [10].

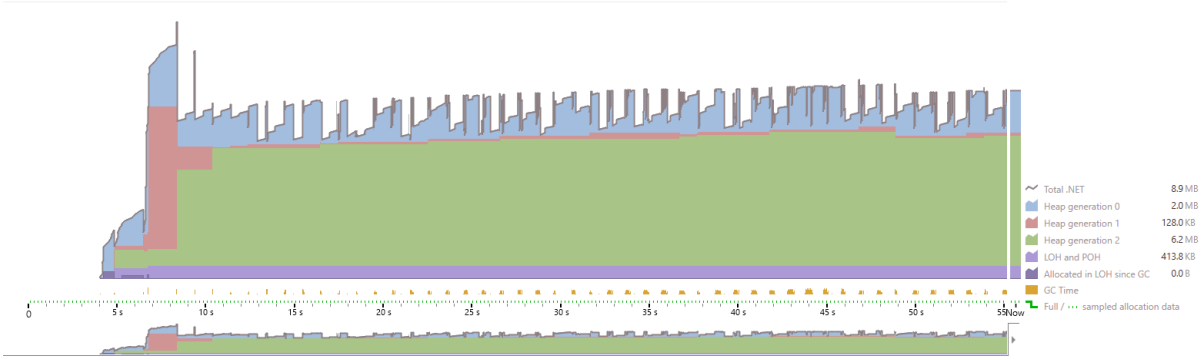


Figure 4.3: Memory usage of the application during a game of Gomoku

Total .Net is the total memory usage of the application, here being 8.9 MB. This includes all memory allocated by the application.

Heap Generation 0 is the memory used by objects in Generation 0 of the garbage collector's heap which is 2.0 MB. The garbage collector organizes objects into different generations based on their age and usage patterns.

Heap Generation 1 and 2 are 128 KB and 6.2 MB respectively because generation 2 contains long-lived objects that have survived multiple garbage collections.

LOH and POH are the memory used by objects in the Large Object Heap (LOH) and Pinned Object Heap (POH) and are 413.8 KB.

Allocated in LOH since GC is the amount of memory allocated in the LOH since the last garbage collection and has the value 0 bytes here.

These metrics provide insights into how memory is being utilized in the .NET application. By monitoring and analyzing memory usage, you can optimize memory allocation and deallocation strategies, identify potential memory leaks or excessive memory consumption, and ensure efficient memory management in my application.

4.1.4 Community and Documentation

Being one of the most popular programming languages, Python is has a large and active community. It is used for various purposes such as web development, data analysis, machine learning and automation. The Python community is known for its inclusive and welcoming nature, making it easy for newcomers to get involved.

The quality and availability of the Python documentation is really good, having a great official documentation website [15] that provides a well-organized documentation

for the language and various modules, including detailed explanations and examples for both beginners or experienced developers. In addition to the official documentation, for Python there are numerous online tutorials, guides and forums that are available and they provide great information and explanations. The Python community also actively contributes to open-source projects, creating libraries and frameworks that are well-documented and supported.

C# also has a sizable and active language community. Developed by Microsoft, C# is primarily used for Windows application development, game development with Unity and backend development with ASP.NET. It is widely adopted language in the enterprise world and has a dedicated developer base.

In terms of documentation, personally I find that C# has one of the best, if not the best documentation. Microsoft provides official documentation through its website, which covers the language syntax, libraries, frameworks and development tools. Same as for Python, apart from the official documentation there are also a lot of online resources available for learning C#.

While the C# community may not be as large as the Python one, it is still active and supportive with dedicated groups, conferences and forums where developers connect and share knowledge.

In summary, both Python and C# have active language communities with excellent documentation resources, while C# is popular for its ecosystem and enterprise development, Python is popular for it's ease of use and community. Whether you choose Python or C#, you can expect to find ample learning materials and support without any problems.

4.1.5 Development Productivity

Both Python and C# offer productive development experiences with robust tooling and debugging capabilities. Python's simplicity and extensive library ecosystem make it a popular choice for various domains, while C# excels in the Microsoft ecosystem and enterprise development. Developers satisfaction can vary depending on their familiarity with the language and the specific requirements of their projects.

For this application, the development of the C# algorithm took way longer than expected, I ran into a lot of bugs and most of the time I was stuck and I didn't know how to continue with the implementation, but my personal experience with C# and WPF which I used for the GUI over the years allowed me to finish the application in a reasonable time, without any big troubles.

The implementation for the Python part was a lot easier than the C# one, maybe because I already finished the C# one, so I already had a taste of how the algorithm should look and what I am supposed to do. Even when you compare the length of

the code that I wrote, you can clearly see why Python is often preferred when writing small applications. The code looks simpler and easier to understand, even though it's the same as in the C# one.

Overall implementing the algorithm in both Python and C# helped me to learn the strengths and weaknesses of these languages so I have a better understanding of what should I use in the future when doing something similar.

4.2 Results Interpretation

In terms of execution speed, the performance measurements indicated that the C# MiniMax implementation showcased better speed compared to its Python counterpart. The average time taken to compute moves in the C# implementation was consistently lower than in Python, suggesting that C# exhibited better computational efficiency in Gomoku game scenarios. This finding suggests that developers who prioritize speed and real-time decision-making might find the C# implementation more suitable for Gomoku game AI development.

Additionally, Python demonstrated a benefit in terms of simplicity and ease of implementation, according to the report. The MiniMax algorithm for Gomoku was simpler to construct thanks to Python's clear syntax, substantial library support, and dynamic typing. On the other hand, because of its static type and stricter syntax, the C# implementation had a slightly higher learning curve. The performance benefits provided by the C# implementation might, in certain cases, exceed the fact that Python was easier to construct, which is crucial to keep in mind.

Furthermore, when considering code readability and maintainability, both languages had their strengths. Python's emphasis on readability and its expressive nature made the code more readable and easier to understand, which can be beneficial for long-term maintenance and collaboration. C#, with its strong type system and object-oriented features, provided better code organization and maintainability in larger codebases. However, readability and maintainability can also depend on individual coding practices, conventions, and the experience of the development team. Also when debugging the program I found it easier to do and understand what's going on in C# because of Visual's Studio amazing features and information, but if we talk about development speed, the Python implementation was much faster to implement.

In conclusion, this study has provided valuable insights into the comparison of C# and Python MiniMax implementations in the context of Gomoku. The results suggest that C# offers superior execution speed and computational efficiency, while Python excels in ease of implementation and code readability. The choice between the two languages depends on the specific priorities and requirements of the Gomoku game development project.

Chapter 5

Conclusion

In conclusion, this study proposes a comprehensive analysis of Python and C# in various aspects of software development using the game Gomoku as an intermediate with the implementation of MiniMax with performance improvements. It explores factors such as the performance, efficiency, ease of implementation, code readability, and maintainability of the MiniMax algorithm in both programming languages.

Additionally, this study highlights the differences between statically typed languages like C# and dynamically typed languages like Python and examines their implications on the implementation of the MiniMax algorithm. Overall this study contributes to the field by providing valuable insights and recommendations for developers and researchers interested in implementing the MiniMax algorithm for any strategy game, helping them make informed decisions based on the comparison and analysis presented in the paper.

To improve the accuracy of the comparison, further research into other performance enhancements of MiniMax should be conducted and implemented. Additionally, incorporating more programming languages, like C++, C etc. in this study could be a great benefit. Alternative Game variants can extend the comparison, such as different board sizes within Gomoku or even different board games altogether. Machine learning integration could also be a great improvement to our study using for example neural networks or reinforcement learning.

By pursuing these future work directions, this study can be significantly improved in terms of performance, accuracy, user experience, ease of implementation and more, benefiting developers interested in this line of work.

Bibliography

[1] C# documentation. https://learn.microsoft.com/en-us/dotnet/csharp/. Accessed: 2023-02-10.

[2] Gomoku world. http://gomokuworld.com/gomoku/3.

[3] Mvvm documentation. https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm. Accessed: 2023-06-11.

[4] Wpf documentation. https://learn.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-7.0. Accessed: 2023-06-1.

[5] B. S. , V. R, and s. R. *Analysis of Minimax Algorithm Using Tic-Tac-Toe*. 11 2020.

[6] R. J. Aumann. Acceptable points in general cooperative n-person games. In *Contributions to the Theory of Games (AM-40), Volume IV*. Princeton University Press, n.d.

[7] Carolina Ruiz, Samuel Ogden and Ahmedul Kabir. Wpi cs4341: Artificial intelligence. http://web.cs.wpi.edu/~cs4341/a17/Projects/Proj2/, Accessed 2023.

[8] W. contributors. Brute-force search. https://en.wikipedia.org/wiki/Brute-force_search, 2022. Accessed April 22, 2023.

[9] M. Dabbous, A. Kawtharani, I. Fahs, Z. Hallal, D. Shouman, M. Akel, M. Rahal, and F. Sakr. The role of game-based learning in experiential education: Tool validation, motivation assessment, and outcomes evaluation among a sample of pharmacy students. *Education Sciences*, 12(7), 2022.

[10] JetBrains. JetBrains DotMemory. https://www.jetbrains.com/dotmemory/, Accessed 2023.

[11] E. L. Jon Skeet. *C# In Depth: Covers C# 2 And C# 3*. Dreamtech Press, 2008.

[12] A. Nareyek. Ai in computer games: Smarter games are making for a better user experience. what does the future hold? *Queue*, 1(10):58–65, feb 2004.

[13] A. Newell and H. A. Simon. *Human Problem Solving*, 1972.

[14] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1982.

[15] Python Software Foundation. Python 3 documentation. `https://docs.python.org/3/`, Accessed 2023.

[16] D. Robinson. The incredible growth of python. 2017.

[17] S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice Hall, 3rd edition, 2010.

[18] J. Schaeffer. The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1203–1212, 1989.

[19] soubhikmitra98. Brute force approach and its pros and cons. `https://www.geeksforgeeks.org/brute-force-approach-and-its-pros-and-cons/`, 2022. Accessed April 22, 2023.

[20] V. Thangarajah. Python current trend applications-an overview. page 8, 10 2019.

[21] Y. Wang. Mastering the game of gomoku without human knowledge. 06 2018.

[22] Wikimedia Commons contributors. MVVM Pattern. `https://commons.wikimedia.org/wiki/File:MVVMPattern.png`, n.d. Image retrieved from Wikimedia Commons, licensed under CC BY-SA 3.0.

[23] Wikipedia. Gomoku. `https://en.wikipedia.org/wiki/Gomoku`.