CSE472s (UG2018) - Artificial Intelligence (29162)

project: Othello AI Player

submitted to: Dr. Manal Morad Zaki

Eng. Mahmoud Osama Soliman

Eng. Ahmed Mostafa Soliman

submitted by:

Group number: 21

Names and IDs:		
	Waleed Saber Mohamed Hassan	1701664
	Bassem Hussein Fawzy Yakout	1200427
	Mahmoud Hassan Ahmed Dawood	1501341
	Mohamed Amr Yehia	1601251
	Hussein Mahmoud Fouad	1700462
	Mohamed Eid Youssef	1701261
	Mostafa Sami Fayez	15X0076
	Neven Nabil Shokri	1601626

link for the Game EXE:

https://drive.google.com/drive/folders/1ZApoDdkrOQ8_tlXoG6r1G3kVZnOs6J-d?fbclid=IwAR2eTEziUDGESVhnKt5RsQrRQZ2z9C471cGVVgnKrLizoa_ZjNaYSUgdPnE

Table of content

Table of Contributions

Name	ID	Contibution
Waleed Saber Mohamed Hassan	1701664	-designed and implemented the whole GUI. -code for alpha beta pruned and iterative deepening. -searched for the mobility part in the heurestics. -code integration and repo management.
Bassem Hussein Fawzy Yakout	1200427	-Code for: calculating the heuristic function. static cost & stabled disks. -did most of the heuristics analysis . -Class Diagram. -created README.md
Mahmoud Hassan Ahmed Dawood	1501341	-Code for: Finding all possible moves. -State diagram. -recording and editing the youtube video.
Mohamed Amr Yehia	1601251	-Code for: counting the coins each player has. check the last move for any captured coins. -Sequence diagrams. -researched the programming languages and frameworks for the project.
Hussein Mahmoud Fouad	1700462	-Code to check possible coins to flip in any direction. check the validity. -participated in the documentation.
Mohamed Eid Youssef	1701261	-Code for best move implementation. - Flow chart -participated in the documentation.
Mostafa Sami Fayez	15X0076	-Code for mobility heuristic using potential moves. -participated in the heuristics search. - converted python script into exe.
Neven Nabil Shokri	1601626	-Code for minmax algorithm. -participated in the documentation. -testing the game for logical bugs.

GitHub link

https://github.com/Waleedsaber22/othello_project/blob/main/othello.py?fbclid=IwAR2l0eTaL5kjmDxA4UFBOFV-ypw0ynG8oWklBySUMH4hKLE0fQq_ixshDIk

link for the Game EXE

https://drive.google.com/drive/folders/1ZApoDdkrOQ8_tlXoG6r1G3kVZnOs6J-d?fbclid=IwAR2eTEziUDGESVhnKt5RsQrRQZ2z9C471cGVVgnKrLizoa_ZjNaYSUgdPnE

programming languages and frameworks:

The project was written in Python, while Python has many advantages.

Ease of Use: Python has a simple and readable syntax, making it easy to learn and understand. It emphasizes readability and simplicity, which helps developers write clean and maintainable code. This ease of use makes Python a great language for beginners and experienced programmers alike.

Vast Ecosystem: Python has a rich ecosystem of libraries and frameworks that are specifically designed for AI and machine learning. Popular libraries like TensorFlow, PyTorch, and sci-kit-learn provide powerful tools and pre-built functions for various AI tasks. These libraries enable you to leverage existing algorithms and models, saving you time and effort.

Community Support: Python has a large and active community of developers. This means that there are ample resources available, including documentation, tutorials, forums, and open-source projects. If you encounter any issues or need help, you can easily find support from the Python community.

Flexibility: Python is a versatile language that can be used for a wide range of AI projects, including natural language processing, computer vision, data analysis, and more. It allows you to experiment with different algorithms and approaches, adapting to the specific needs of your project.

Integration Capabilities: Python seamlessly integrates with other languages and platforms. You can easily combine Python with C/C++, Java, or other languages to leverage existing codebases or take advantage of performance-critical components. Python also integrates well with web frameworks, allowing you to create AI-powered applications with ease.

Prototyping and Rapid Development: Python's simplicity and readability make it an excellent choice for prototyping and rapid development. You can quickly test ideas and build prototypes to validate your concepts before investing significant time and resources. Python's interactive shell and extensive libraries enable a faster development cycle.

Deployment Options: Python offers various deployment options, including cloud platforms, containerization, and web frameworks. You can deploy your AI project as a web application, a RESTful API, or a standalone executable. This flexibility allows you to choose the deployment method that best suits your project's requirements.

For the GUI we selected Tkinter as there are several reasons why you might choose to use Tkinter for developing graphical user interfaces (GUIs) in Python:

1.Built-in Library: Tkinter is a standard library in Python, which means it comes bundled with the Python installation. You don't need to install any additional packages or dependencies to use Tkinter, making it easily accessible and widely available across different platforms.

2.Simplicity and Ease of Use: Tkinter has a straightforward and easy-to-understand API. Its simplicity makes it a great choice for beginners who are learning GUI programming or Python itself. The syntax is clean and intuitive, allowing you to quickly create basic GUI applications with minimal code.

3.Rapid Prototyping: Tkinter provides a quick way to create prototypes or simple applications due to its simplicity. You can easily create basic windows, buttons, text boxes, and other UI elements using Tkinter's built-in widgets. This makes it suitable for rapid prototyping or developing small-scale applications with limited GUI requirements.

4.Cross-Platform Compatibility: Tkinter is available on multiple platforms, including Windows, macOS, and Linux. This cross-platform compatibility allows you to develop GUI applications on one operating system and run them on different platforms without significant modifications.

5.Integration with Python: Tkinter integrates seamlessly with Python, allowing you to leverage the full power of the Python programming language. You can easily combine Tkinter with other Python libraries for data processing, networking, or scientific computations. This integration provides a robust foundation for developing applications that require both GUI elements and complex functionality.

6.Customization and Styling: Tkinter offers various options for customizing the appearance of your GUI applications. You can change the colors, fonts, and sizes of UI elements to match your

application's design requirements. Additionally, you can create custom styles using the Tkinter styling mechanisms, giving you control over the overall look and feel of your application.

7. Active Community and Resources: Tkinter has a dedicated and active community of developers who contribute to its development and provide support to users. There are numerous tutorials, examples, and resources available online, making it easier to learn and get started with Tkinter. If you encounter any issues, you can find assistance from the Tkinter community.

Introduction

In this project, we will explore the game playing of Othello using search algorithms and heuristics. Othello is a two-player strategy game played on a 8x8 board, where each player has pieces that are either black or white. The goal of the game is to have the most pieces of your color on the board at the end of the game.

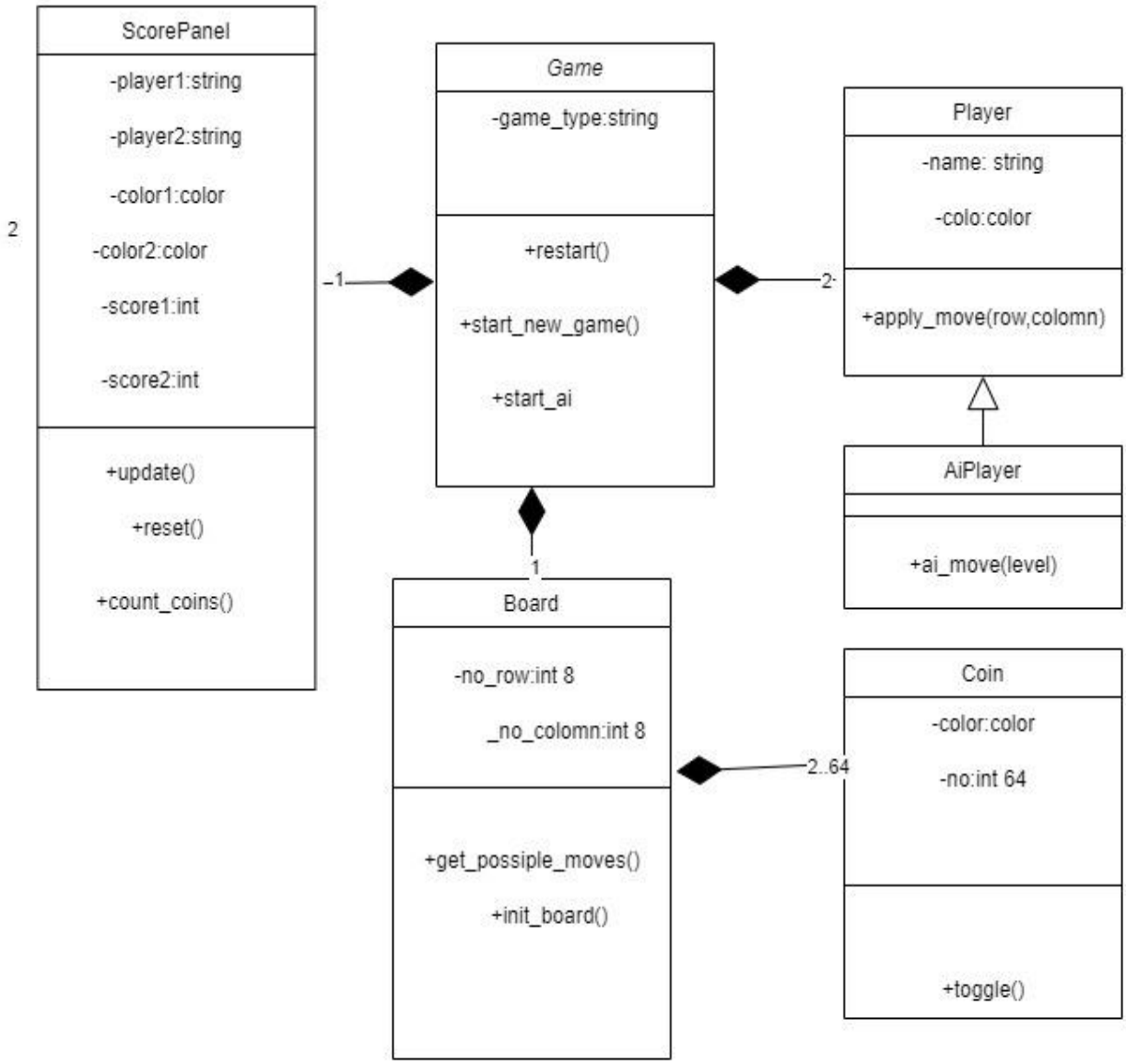
Othello is a strategy board game for two players (Black and White), played on an 8 by 8 board. The game traditionally begins with four discs placed in the middle of the board as shown below. Black moves first.

Black must place a black coin on the board, in such a way that there is at least one straight (horizontal, vertical, or diagonal) occupied line between the new disc and another black disc, with one or more contiguous white pieces between them.

when 2 consecutive turns get passed or there is no place on the board for a coin to be placed the game ends and the player with the higher number of coins win

UML Diagrams

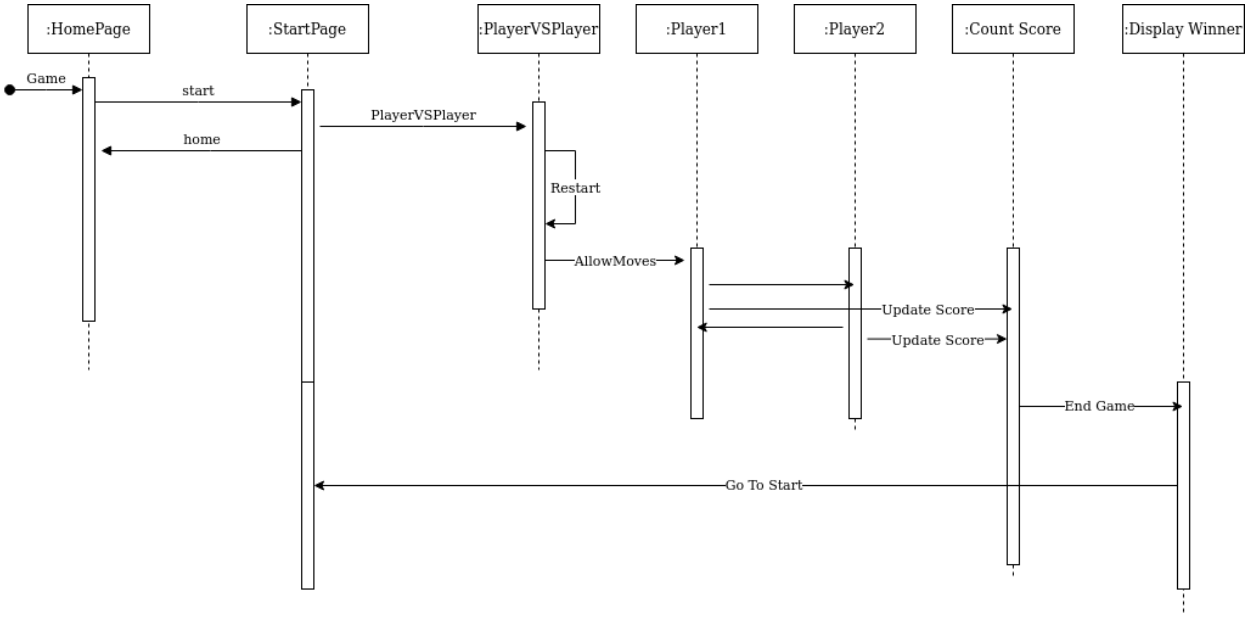
Class Diagram



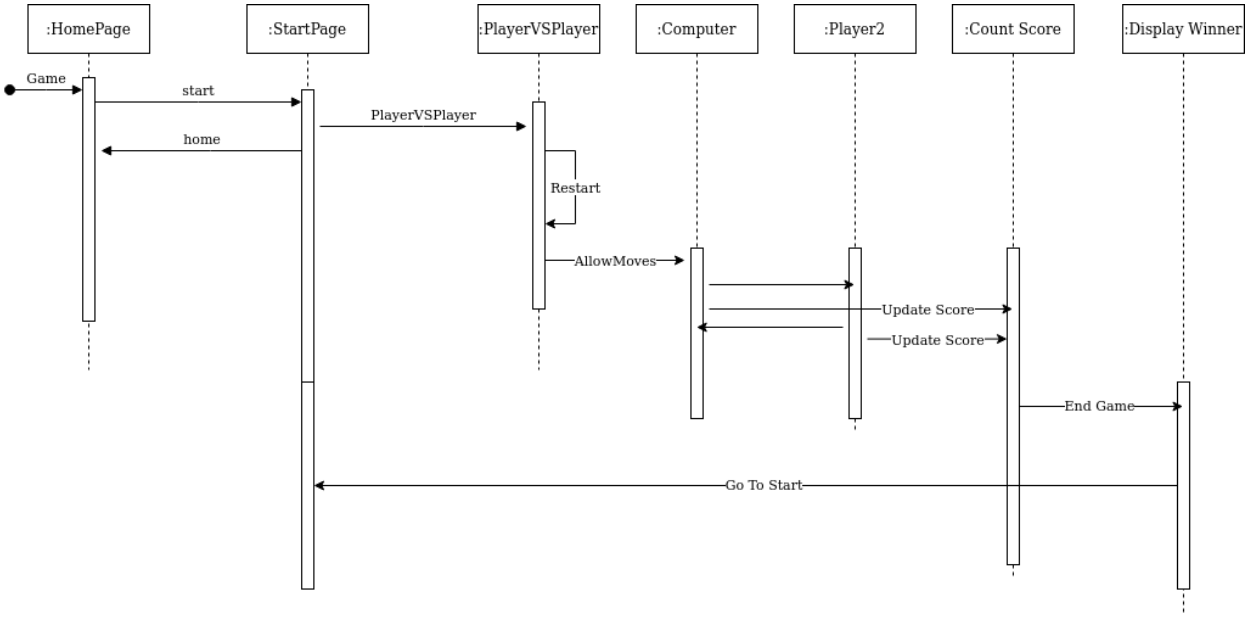
class diagram

Sequence Diagrams

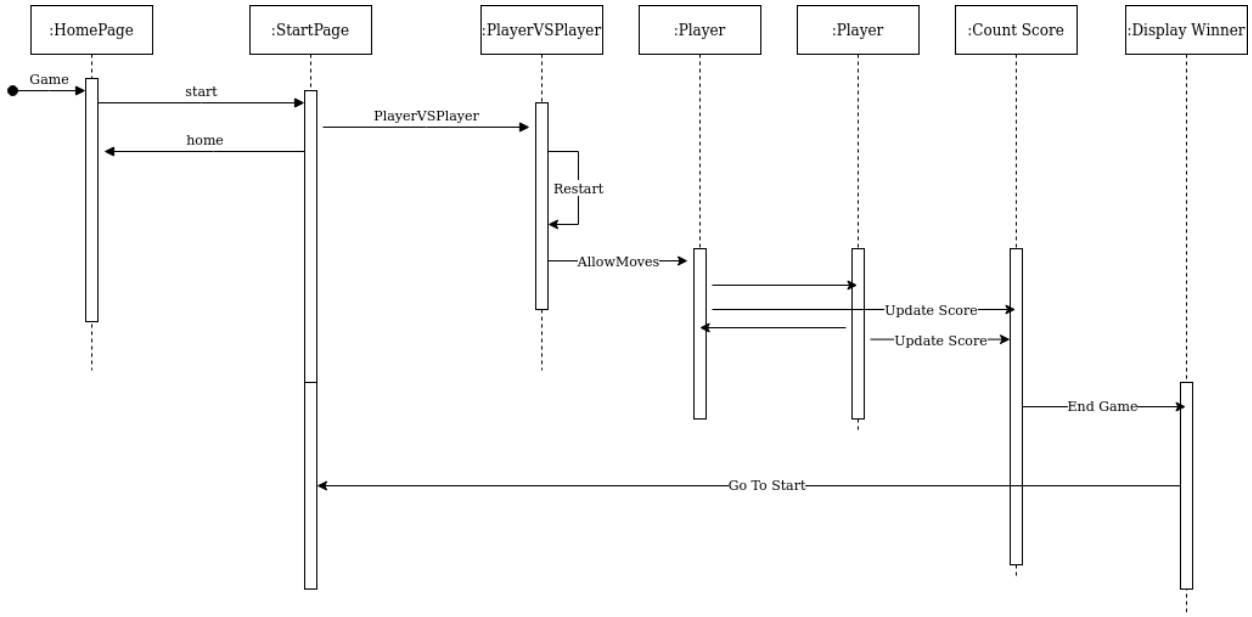
1.player vs player



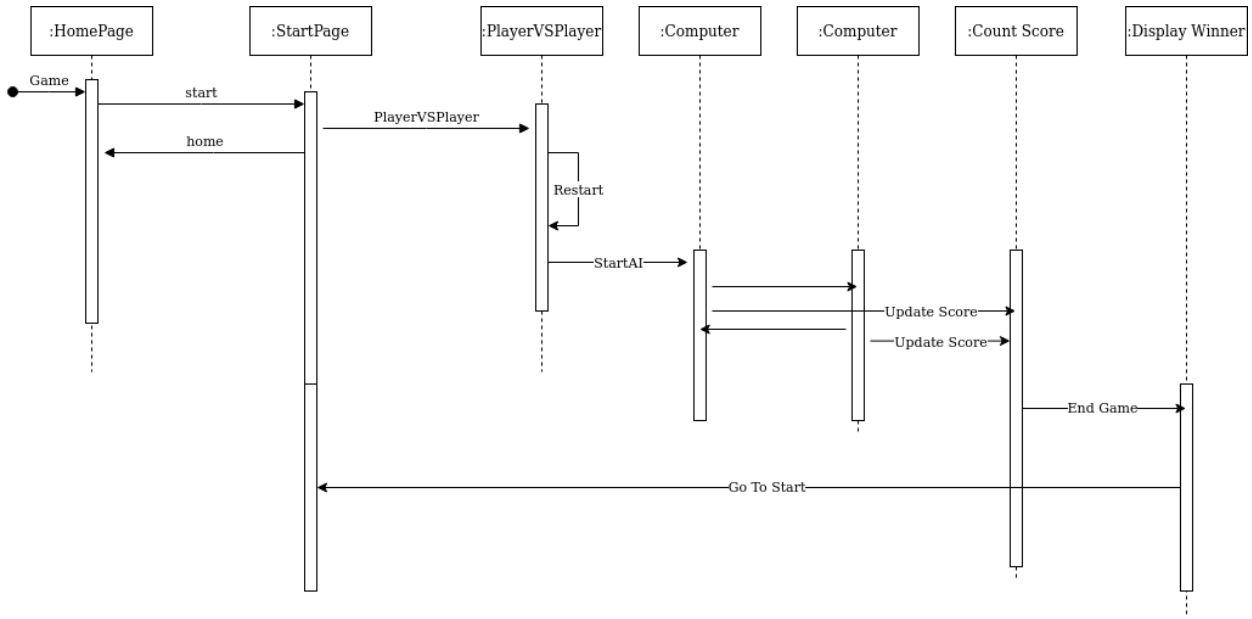
2.computer vs player



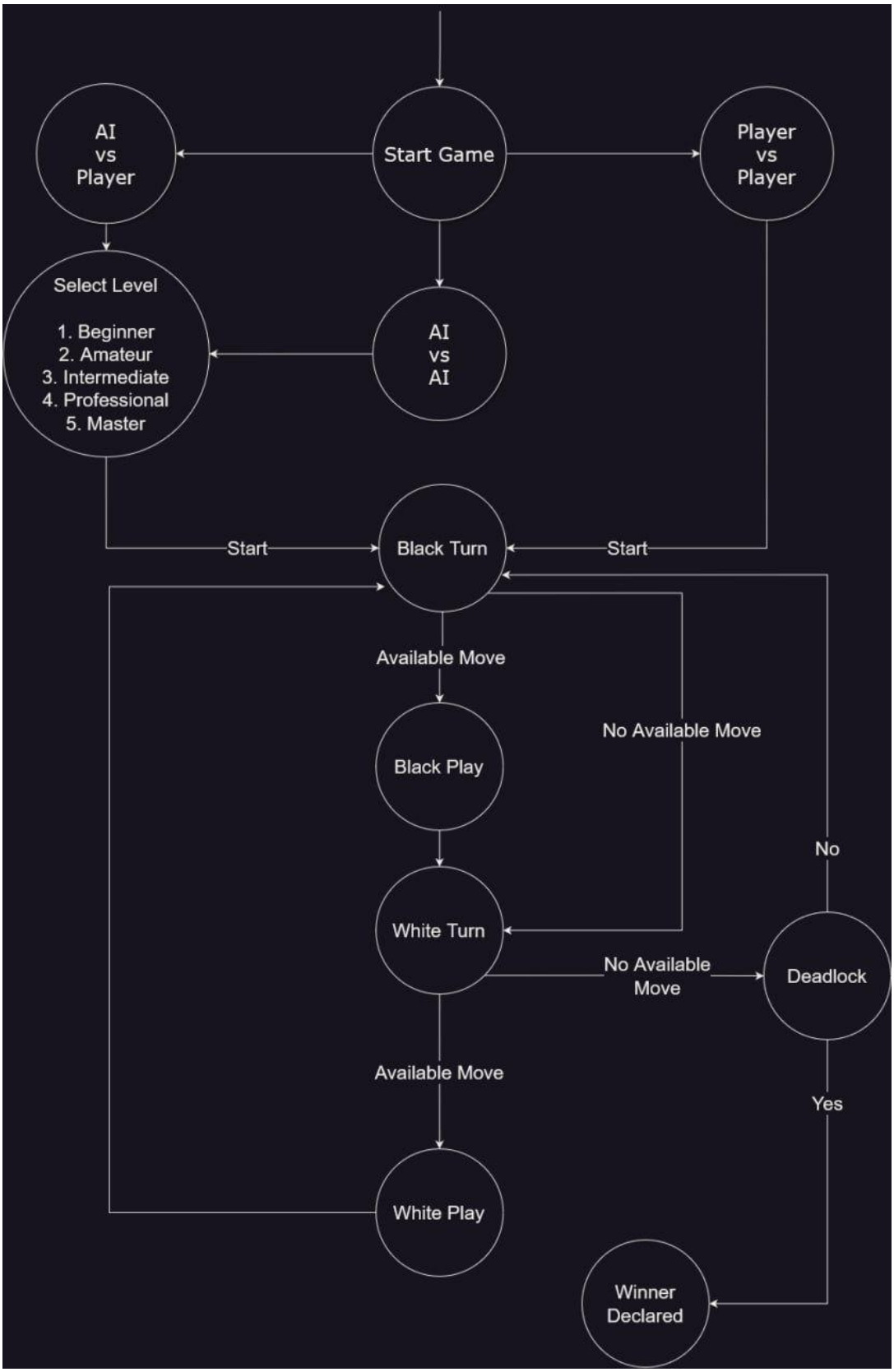
3.player vs computer



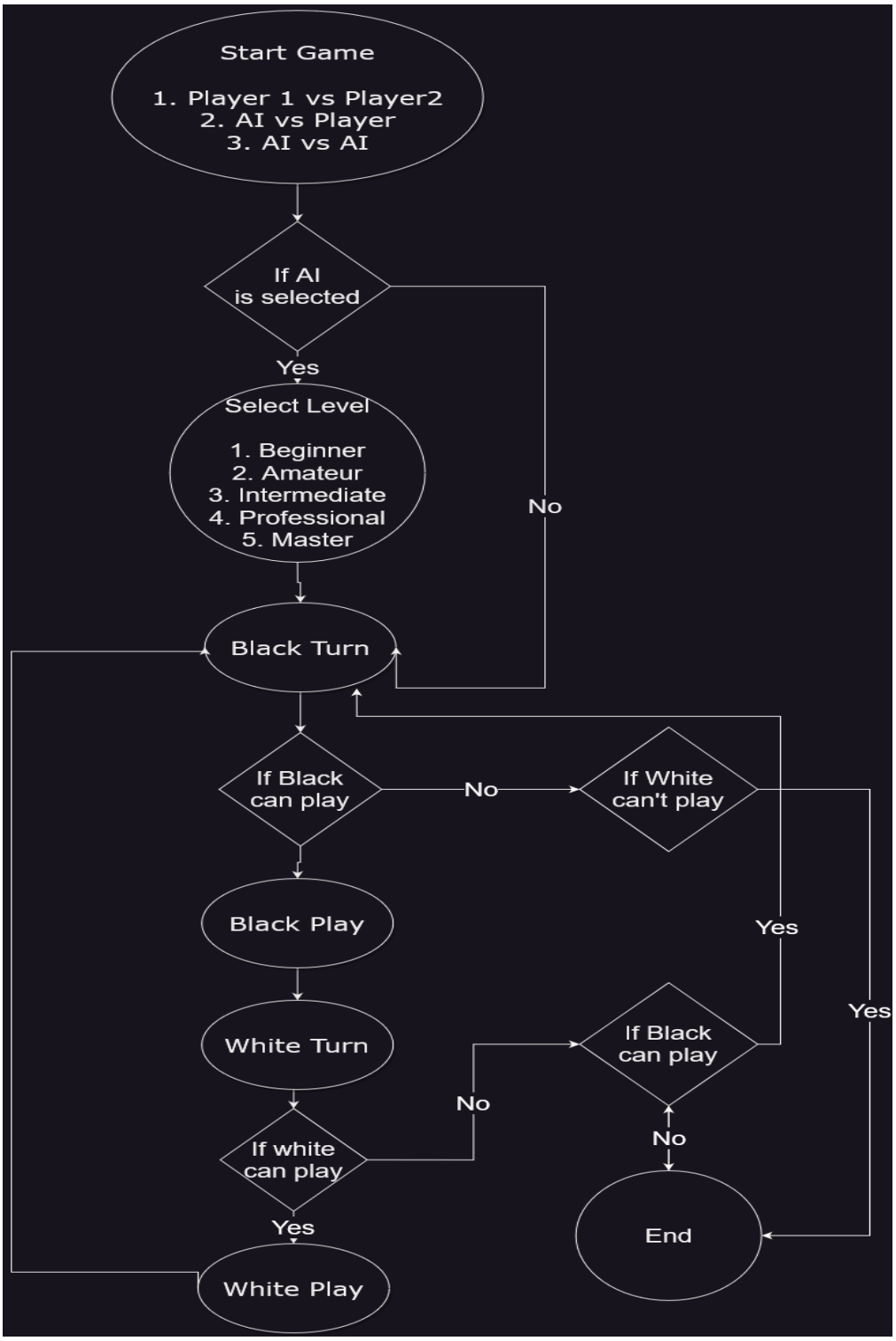
4.computer vs computer



State Diagram:



Flow Chart


Start Game
1. Player 1 vs Player2
2. AI vs Player
3. AI vs AI
If AI is selected
Yes
No
Select Level
1. Beginner
2. Amateur
3. Intermediate
4. Professional
5. Master
Black Turn
If Black can play
No
If White can't play
Yes
Black Play
White Turn
If Black can play
Yes
No
If white can play
No
Yes
End
White Play

Game playing supported algorithms:

we used

Minimax Algorithm:

The Minimax algorithm is a decision-making algorithm used in two-player zero-sum games, where one player's gain is the other player's loss. The algorithm works by recursively evaluating the possible moves of both players and choosing the move that minimizes the maximum loss. The algorithm assumes that both players play optimally, and it tries to minimize the maximum loss that can occur.

Alpha-Beta Pruning:

The Alpha-Beta pruning algorithm is a variation of the Minimax algorithm that reduces the number of nodes evaluated in the search tree by pruning subtrees that are guaranteed to be worse than previously explored subtrees. The algorithm maintains two values, alpha and beta, that represent the best values found so far for the maximizing and minimizing players, respectively. If a node's value falls outside the range defined by alpha and beta, the algorithm prunes the subtree rooted at that node, as it cannot affect the final result.

Alpha-Beta Pruning with Iterative Deepening:

The Alpha-Beta Pruning with Iterative Deepening algorithm is a combination of the Alpha-Beta pruning algorithm and the Iterative Deepening algorithm. It works by performing a series of depth-limited searches with increasing depth, each time using the Alpha-Beta pruning algorithm to evaluate the nodes at the current depth. The algorithm terminates the search when it reaches a maximum depth or when it finds a winning move. The advantage of this algorithm is that it allows for a more efficient search by exploiting the information gained from the previous searches.

Used Heuristics

to help with the heuristic analysis there is some terminology we need to agree on to make things more easy to describe

	A	B	C	D	E	F	G	H
1	corner	c					c	corner
2	c	x					x	C
3								
4								
5								
6								
7	c	x					x	C
8	corner	c					c	corner

raw 1 : north boarder

raw 8 : south boarder

colomn A : west boarder

colomn B : east boarder

cells (A,1),(H,1),(A,8),(H,8) are corners

cells (A,2)&(B,1) are adjacent to the corner(A,1) we give them the symbol c

cell (B,2) is adiagonal cell to the corner (A,1) we give it the symbol x

we assume that we want the player with the colour black to win

1.Static cost with corners captured heuristics:

from playing lots of games and reading about Othello stratigies and how actual players take decisions in real competitions and in blitz games ,we came up with a value for each cell in the board this values are completely empirical we didn't copy it from any particular source it is completely based on our understanding of the game and our experience playing it
these values are just the first step in calculating the initial cost value of a leaf node

80	1	10	10	10	10	1	25
1	2	4	4	4	4	2	1
10	4	7	6	6	7	4	10
10	4	6	7	7	6	4	10
10	4	6	7	7	6	4	10
10	4	7	6	6	7	4	10
1	2	4	4	4	4	2	1
25	1	10	10	10	10	1	25

to calculate the static cost : we multiply each cell value with

1 incase it's black

-1 incase it's white

0 incase it's empty

then we sum all of them up and we get a single value representing the initial cost of the state

the corner cells have a value of 80 as a good incentive to always choose to occupy the corner cell even if the alternative option will toggle a lot of coins (this is a good generalization unless we have un balanced edges which we won't get into in this project)

2.Stabled disks:

The reason why corner cells have such high value is because if there are occupied the can't be flanked from both sides thus they can't be turned .

it is a good strategy to try to make more stable cells , we in the project were content by just modifying the value of the c cells as it is by default has lowest value over all but before we calculate the static cost we check if it's adjacent corner is occupied by the same color as the c cell we're playing into; we change its value from 1 to 80.

3.Frontiers (mobility)

there are a couple of ways to take mobility into consideration, one approach we took is to focus on minimizing the number of frontiers cells black has and to maximize the number of cells white has.

		W	W	W	W		
B		W	W	W	B		
B	B	B	B	B	W	W	W
B	W	B	W	B	W	W	W
B	W	B	W	B	W	W	W
	W	W	W	W	W	B	W

		W	W	W	W		
B		W	W	W	B		
B	B	B	B	B	W	W	W
B	W	B	W	B	W	W	W
B	B	B	W	B	W	W	W
B	B	B	B	B	B	B	W

first we need to define what frontier means in the context of the game
it is the cell that has empty diagonals or adjacent cell so potentially it can be flipped

in the above case in the first table white has 6 frontiers in raw 6 and 1 at(B,5), if it is black ‘s turn and he played A6 the black increased it’s frontiers by 6 and decreased white frontiers by 6 so he over all lost 12 frontiers from making this move

we need to calculate how many frontiers our opponent have and subtract them from our own and then multiply the amount by a factor and modify the total cost accordingly.

4.coin parity:

it’s a very poor strategy , we simply calculate how much more cells we occupy by each move considering the opponents toggled one.

e.g. in the last to tables making the move (A,8) is an excellent choice according to the coin parity heuristic how it’s not the case in the long run as it limits the number of options we have thus reducing the mobility and a better strategy is to try to get stable cells as having a large

number of coins from the beginning doesn't count for anything, the game can turn around very quickly specially at the end stage.

that's why we didn't use it in our project on it's own ,but we took the value of the flipped coins into consideration when we applied the static cost.

5.Parity of empty spaces:

1	1	W	W	W	W	2	2
B	1	W	W	W	B	2	2
B	B	B	B	B	W	W	W
B	W	B	W	B	W	W	W
B	W	B	W	B	W	W	W
3	W	W	W	W	W	W	W
3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3

for this heuristic we need to find all clusters of empty cells which are connected by being adjacent to each other (not diagonal) . we find this groups by applying a a 4- neighborhood connected algorithm (N4(p)) x

x p x

x

in the above table we have 3 groups of empty spaces
group 1 has 3 cells (odd number)

group 2 has 4 cells (even number)

group 3 has 17 cells (odd number)

as black always begin playing and the number of cells are even white always ends the game if neither player pass it's turn, and because every play flip at least one coin so the score of the white inreases at least by 2 and the gab between the two players by 3

it would be very beneficial if the difference between the passed turns from both players are even from black prespective .so black play the final move instead.

one heuristics to achieve that

black wants the parity of the empty regions after his move to be odd
we calculate the number of empty regions and multiply each one of the by a factor(+ve if the parity is odd,-ve if the parity is even) ,then we add all of them up and use the number to change the value of the initial cost.

(not implemented yet)

6.wedges:

	W				W		
		W	W	W	B		
		B	B	B	W	B	W
		W	W	W	W	B	W
	W	W	B	W	B	W	
		B	W	W	B	W	
		W	B	B	B		W
	W		W	W	W		

when considering using wedges as a heuristic our whole focus is on raw A ,raw B, column 1 and column 8.

we already established the importance of the 4 corner cells and their 8 adjacent cells
the other 16 cells (4 for each raw or column stated above) are also important and have a very high value to occupy.
It is not enough just to place a coin in a cell in it is a possible move as it easily can be flipped later.
in the above table if it's black's turn, it has the option to play in the north boarder in(c,3),(c,4)or(c,5) or play in the east boarder in (H,5)or (H,6) or play in the south boarder in (c,8).

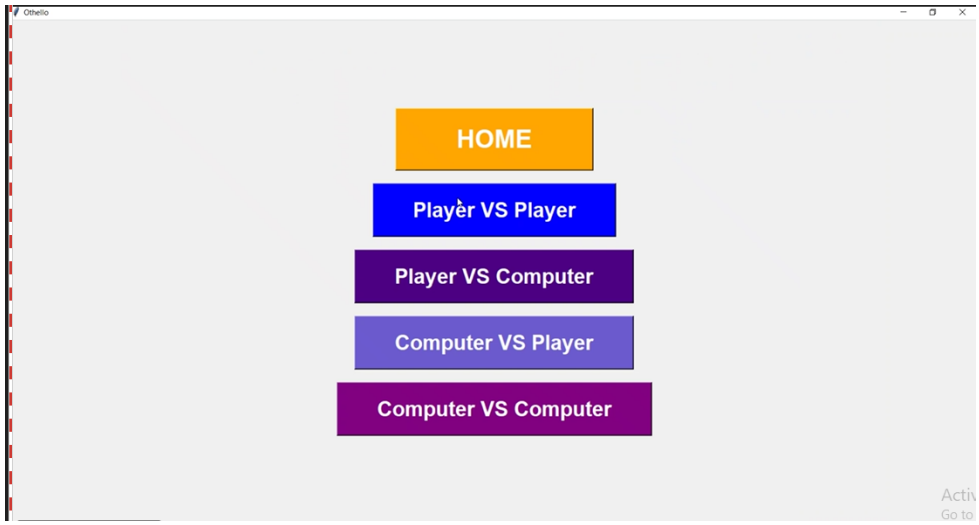
to make a choice of putting a coin between the two white coin (placing a wedge)
black has to count the number of empty consecutive empty cells in between if the number is odd it is in black's best interest to place a tile because even in white flipped it the next turn black will eventually flip it back .
so considering the table above it is better for black from the wedges heuristic point of view to place a coin at the south boarder instead of the east or the north one.
so to apply it to our cost function
we need first to count all the possible places left for a wedge if the number of cells is even it is a positive otherwise it is a negative and multiply each one by a constant ,then sum all of them up to get a single value can be used to change the total cost.

(not implemented yet)

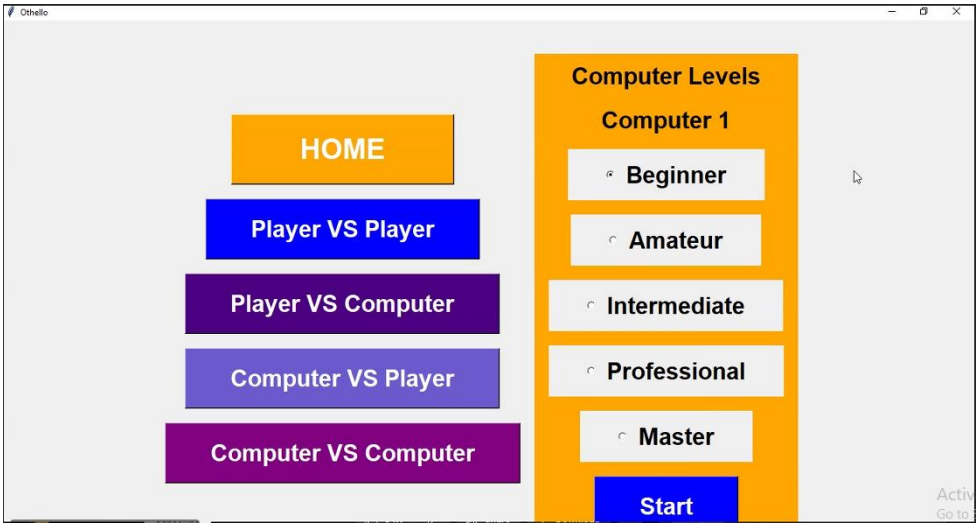
Supported Features:

the game has 4 modes of play:

player vs player
player vs computer
computer vs player
computer vs computer

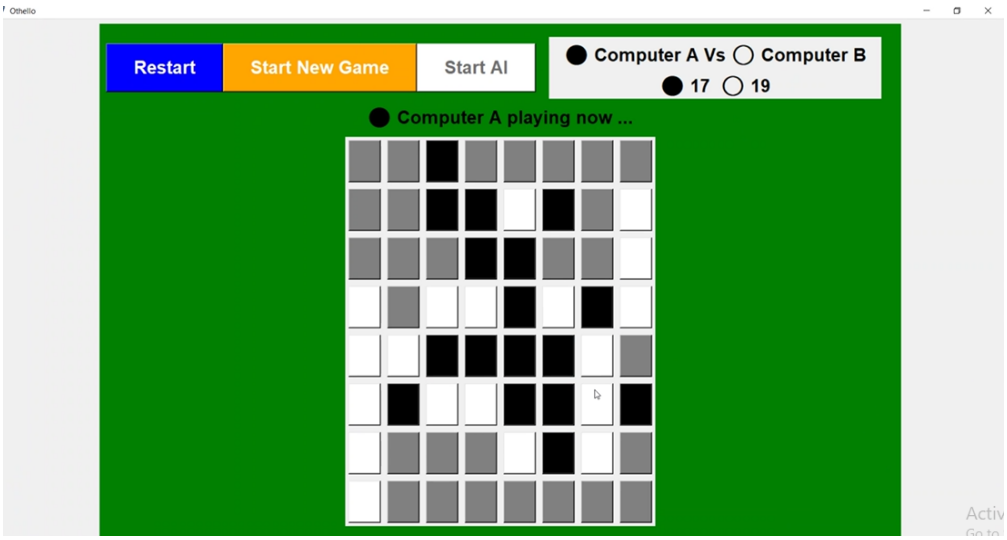


when we choose the computer as an opponent,
we need to choose a difficulty level from 5 options.



due to the design the first player is always black ;so threw choosing the mood you implicitly choose the color
when we go to the portion of the playing itself , we have an 8x8 board
each cell is either black ,white or grey if it is empty.

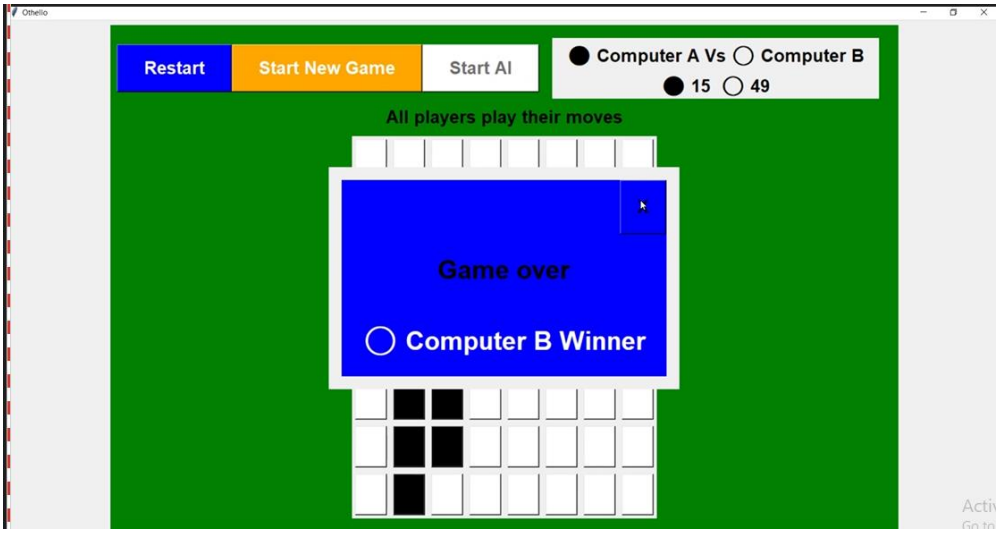
after calculating the possible moves the possible cells change color to blue , after choosing a move it turns temporarily into orange.



there is a score panel in the upper right corner that has the players' names , colors and scores that gets updated after each move .

in the upper right corner there is a bar with 3 buttons
Restart: restart the game with the same configuration
Start New Game: moves us back to choose different options for the game if we want .

Start AI: for the Ai to start a move in case of Computer vs Computer.



at the end of the game a message board appear in the center of the screen announcing the winner , then the user has the option to choose from Restart and Start New Game.

Maximum Difficulty Level:

we have five levels of difficulty arranging in an ascending order based on their difficulty as follows:

-Beginner
-Amateur
-Intermediate
-Professional

-Master

at the Beginner level we use The minimax algorithm at depth level 1.

at the Amateur level we use The minimax algorithm at depth level 2.

at the Intermediate level we use The Alpha-beta pruning algorithm at maximum depth level 4 (While using the iterative deepening search technique).

at the Professional level we use The Alpha-beta pruning algorithm at depth level 5.

at the Professional level we use The Alpha-beta pruning algorithm at depth level 6.

The level at which it becomes very hard to play the AI:

it gets really difficult to play against level 4 (Professional)

Link to YouTube video

we recorded 3 games

the first is Ai level5 against Ai level2 , level 5 won.

the second player vs Ai level 1 ,player won.

the third player vs Ai level 5 ,level 5 won.

https://www.youtube.com/watch?v=wQR46rxgvcw