GCE Computer Science (7517)

The Practical Project

Centre number [redacted]

Centre name [redacted]

Candidate's full name

Sahil [redacted] Karanth

Candidate number [redacted]

Project title

Surakarta

Contents

1 Analysis *(9 marks)*

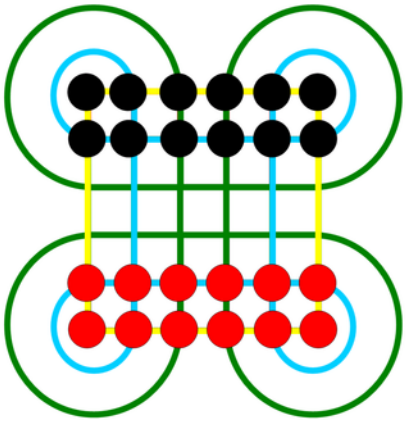
This first section of the document is the analysis done for my project.

1.1 Rules of Surakarta

Surakarta is a two-player turn-based strategy board game. Each player has two rows of six pieces and the aim of the game is to capture all the opponent's pieces. Draws are not possible since a player can always make a legal move.

The 6x6 board contains inner and outer looped tracks, circling around the game board which are used to capture pieces. The diagram below shows a Surakarta board in its starting position where the green track is the outer looped track, and the light blue track is the inner looped track.

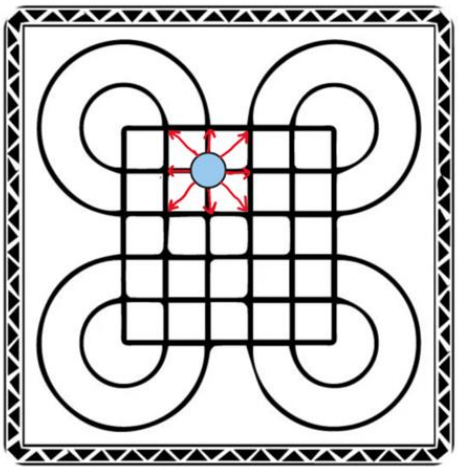
Note that some pieces reside on no track, some are on one track, and some are on both tracks.



Pieces can reside on any intersection between two lines on the board, meaning there are 36 available positions that a piece could be on.

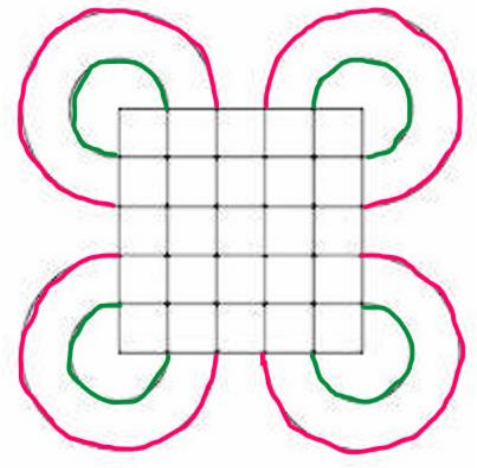
Pieces can move one space in any direction (including diagonal) to a blank space or perform a special capturing move.

A piece's available non-capturing moves are illustrated in the diagram below:

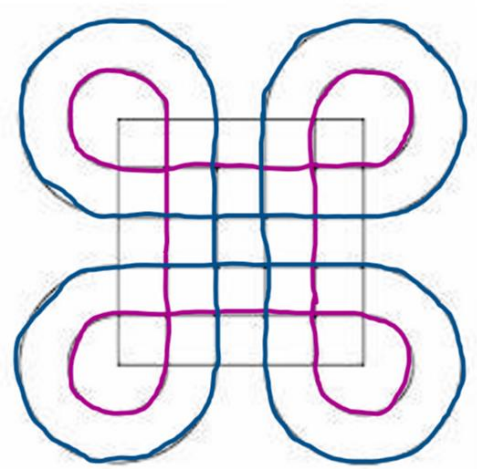


To capture a piece, a piece moves along the inner or outer track around one or more of the four board loops until it lands on an opposing piece and replaces it. Note that throughout this document, the phrase "inner/outer looped track" will be used to refer to the two looped paths a piece can traverse while "board loop" will be used to refer to the four looped sections in each track.

The image below shows the four outer board loops in pink and the four inner board loops in green.

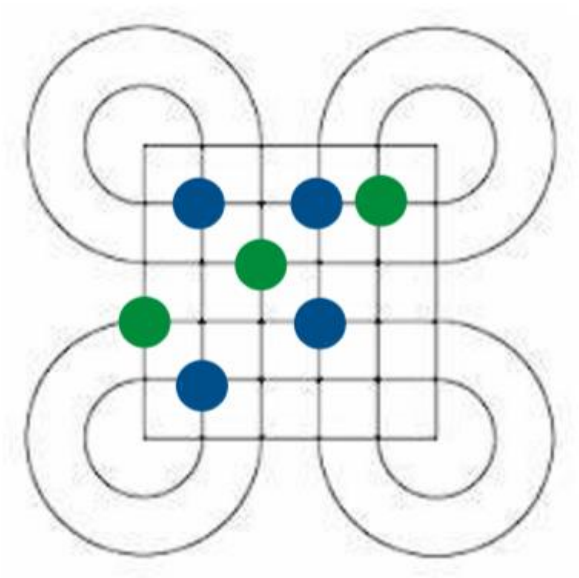


The image below shows the outer looped track in blue and the inner looped track in purple.

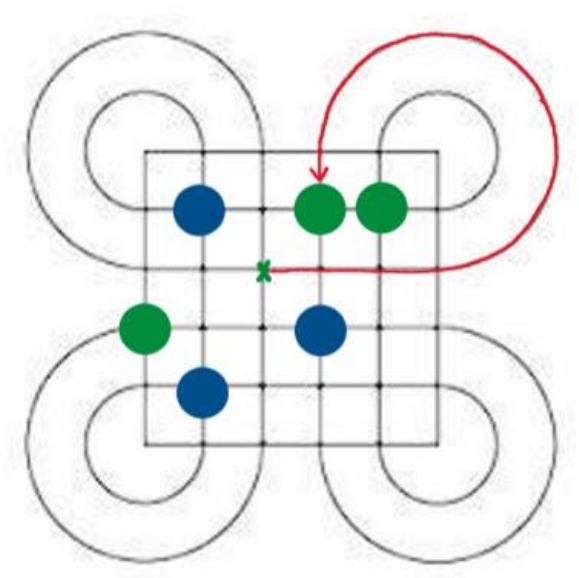


During this path to capture, the capturing piece cannot encounter any of its own pieces before it meets an opposing piece. In other words, for a capture to be legal, no pieces can be in-between the capturing piece and the piece being captured on its capture path. Captures can only be made by making this skating manoeuvre around the board. The image below illustrates a capturing move. Note that the green “x” represents where old position of the green piece before the capture.

Before:



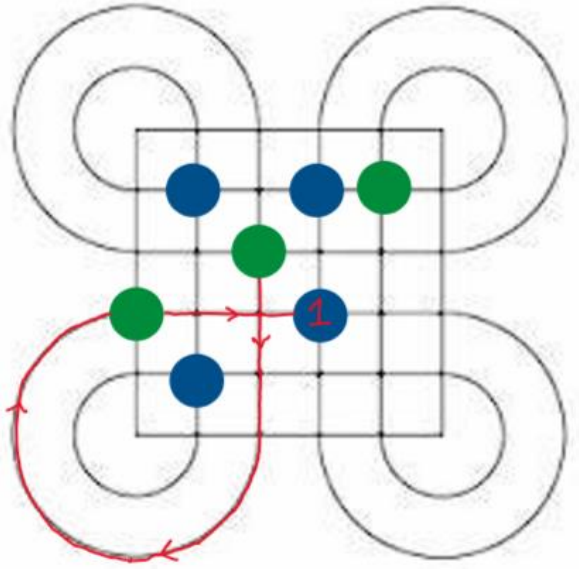
After:



This is a valid capture because the green piece travels through at least one of the 4 board loops and no pieces obstruct the capture.

However, the same green piece could not have captured the blue piece labelled “1” in the diagram below:

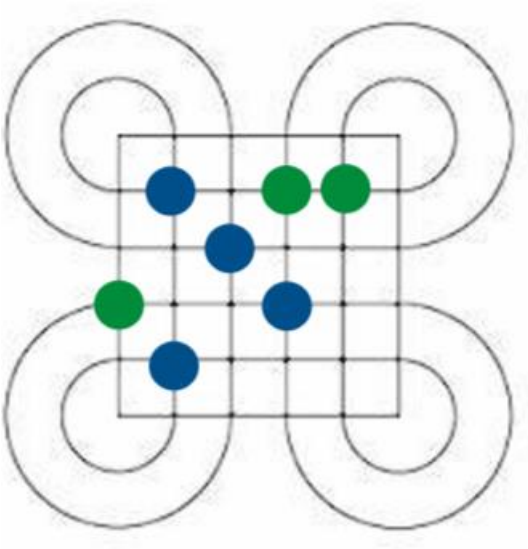
Illegal capture:



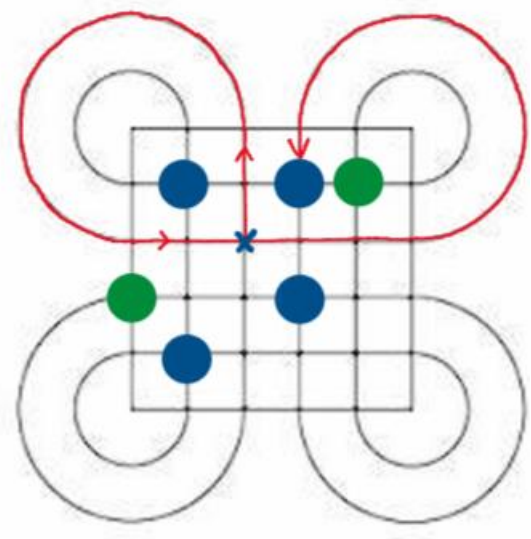
This capture is illegal because the green piece is blocked by another green piece on its capturing path.

However, a piece can travel through its starting position during a capture path. This is illustrated in the more complex capture below. Note that the blue “x” represents the initial position of the blue piece performing a capture.

Before:

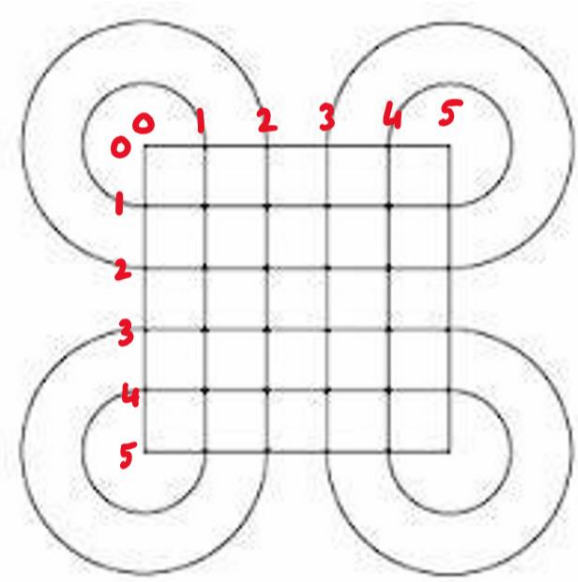


After:



1.1.1 Coordinate System

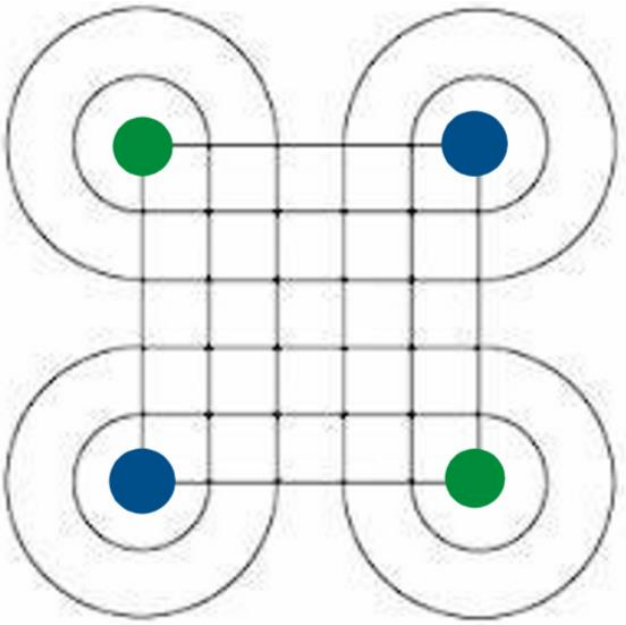
Throughout this project, a coordinate system will be used to locate positions on the board. Locations are identified in the form (r,c) where r is the row index and c is the column index of the location. The top-left location has coordinates (0,0). Moving down increases row index and moving right increases column index. The diagram below is a blank board with row and column indexes labelled.



1.2 Strategic Play

Surakarta is a perfect information game where each player is aware of their opponent's moves. It is also a zero-sum game as when one player gains (for instance by capturing an opposing piece), the other player loses value.

In Surakarta, pieces on the four corners of the board are in a weak position. This is illustrated in the board below.



Corner positions are weak because they are not on either of the board's looped tracks so cannot perform captures. During the middle and end of the game, it is also likely that a corner piece moving onto a looped track could result in its immediate capture because of opposing pieces already on the looped tracks.

Captures are typically strong moves however they are not always the best to play. Capturing can often result in "capture chains" where many consecutive captures take place. In Surakarta, the best move is not always obvious because of defensive play. For example, a capture is not ideal if after the resulting consecutive captures, you end up capturing less pieces.

1.3 Identification of the Problem

My user has identified several problems with their current setup for playing Surakarta which they would like me to solve with my project. They currently play with a physical board and say that they can't play when they are out and about such as on public transport because it is too much of a hassle to carry a board with them with lots of pieces that are easy to lose. My creation of a digital version of the game will solve this problem. The user has also stated that they want to be able to play the game locally against another human which will be implemented in my digital solution. As well as this, they want the option to play against an AI opponent with variable difficulty so they and their family can each play at a suitable level of difficulty even with their varying skill levels. The user cannot play against an AI on a physical board, but this problem will be addressed in my software. Another problem that my user is facing is that they would like a way to save their game easily and come back to it later. This is difficult to do with a physical game but could be implemented with a digital solution. The user also wants a way to keep track of their wins and losses against the computer opponents because they often forget to record the results of matches on paper and would like the software to do it for them.

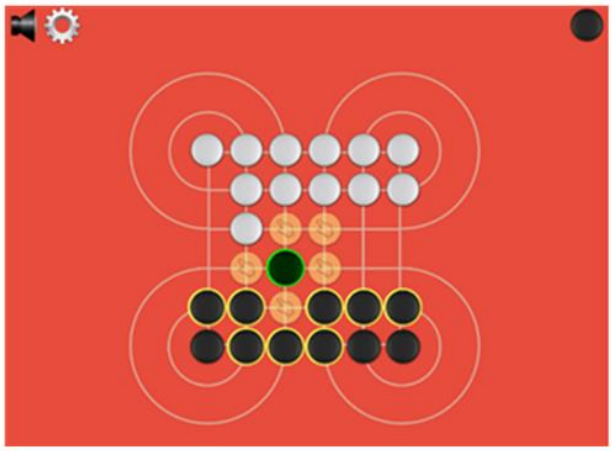
1.4 Analysis of Existing Solutions

I analysed the advantages and disadvantages of three currently available digital Surakarta games to determine the features that need to be a part of my application. Overall, there are very few digital Surakarta programs.

The three existing programs analysed in this section are cited in section 6.2 of this document.

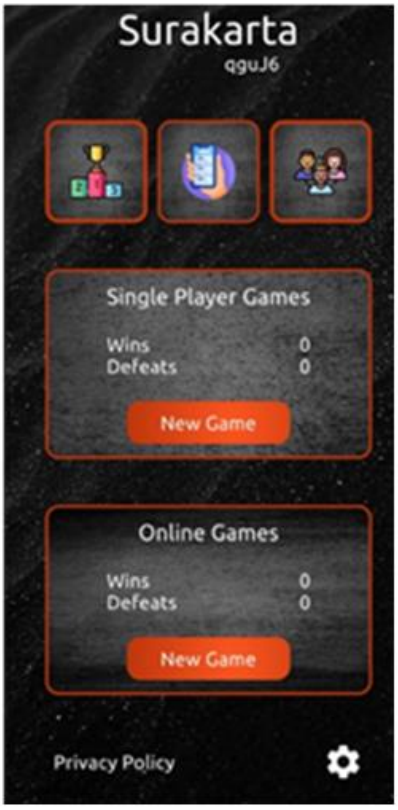
The first program is the only desktop option available which can be found at https://www.onlinesologames.com/surakarta (accessed on the 10/06/2023)

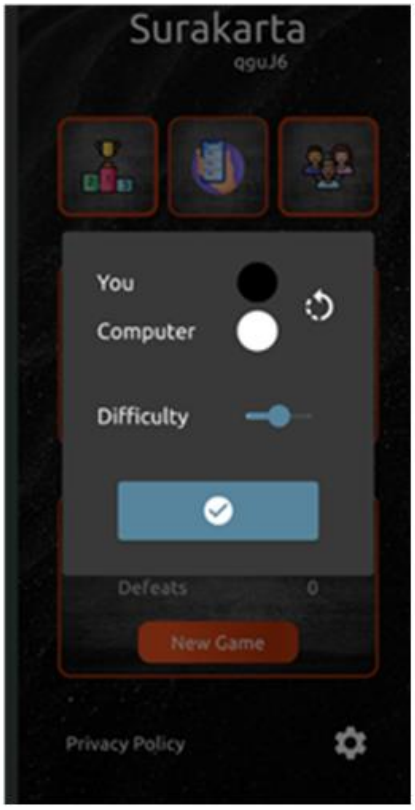
Below is a screenshot from the main menu (left image) and the match page (right image).

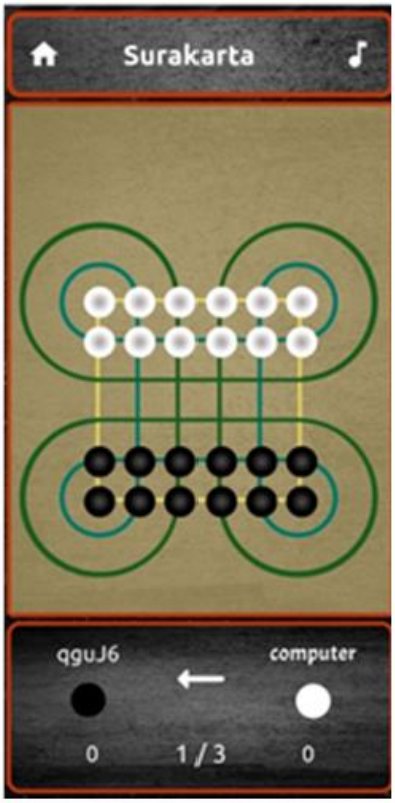


This program offers a simple user interface which is easy to use and understand. It also supports both local play and play against the computer. However, it only has one AI difficulty level and has no account system, undo button or game state saving functionality.

The next program is a mobile application. Screenshots of the application's home menu and match page are shown below.

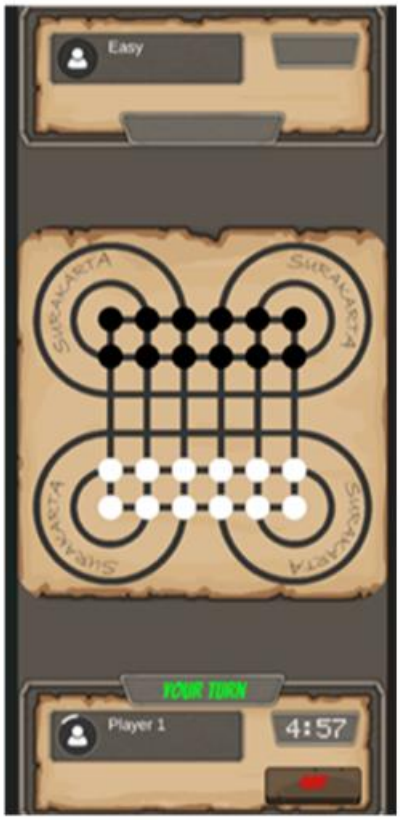






Unlike the previous example, this game includes three AI difficulty levels, an account system and a leaderboard. It also has a simplistic design that is easy for a user to understand. However, it does not include an undo button or game saving functionality.

The final analysed example is another mobile application. Below is an image of its home page (left image) and its match page (right image).



This program benefits from having a variable difficulty AI, simple user interface and timer for each player. However, it doesn't have an undo button, account system or the ability to save a game.

Overall, the simplistic user interface used by all three applications, seen especially through the uncluttered main menu screen, will be used to inspire my user interface and main menu. The option to play locally and against a computer was present in all three applications, illustrating the importance of being able to play both by yourself and with someone. Thus, both local and AI play will be available in my game. Quality of life features like displaying the name of the current player and having a page to display the rules of the game also make the game easier to understand so will be brought forward to my project.

However, saving game state and an undo button were not present in any of the analysed options and so my application will introduce these new features. Accounts were only present in the second application to store aggregate match statistics. My application will also store match statistics in user accounts but in addition to this, my account system will save and allow a user to view their match history as well as customise their piece colour. My program will also be a desktop application as there is currently only one such option available. As well as this, none of the three applications displayed the number of pieces captured by each player which would make it much easier to determine which player has the most material. Thus, piece counts for each player will be displayed during a match in my program.

My project will bring forward the positive aspects and solve the problems identified in the three analysed applications. Features that are scattered between multiple current Surakarta programs will all present in one application and new features will be implemented.

The following are the main features that will be included in my application:

- Local play
- AI play with three difficulty levels
- A user-friendly graphical user interface
- The ability to undo a move
- Saving game state
- Accounts to store:
    - User match statistics (wins and losses against AI, match history)
    - Visual customisations

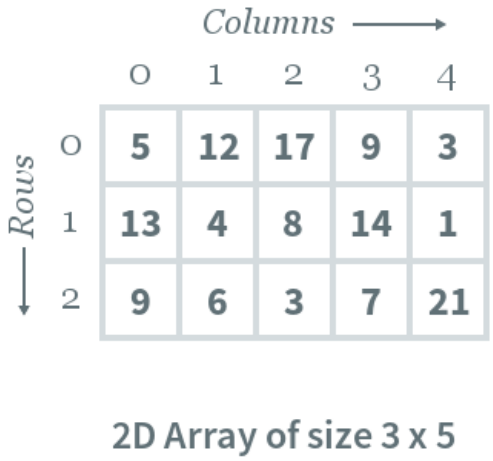
The table below summarises features included in my project, the objective stage they are a part of and whether they are features not found in the analysed existing solutions or features brought forward from existing solutions. Objective stages are described fully in section 1.8.

Feature	Objectives Section	Brought forward (and built upon) or introduced?
Local play	MVP – Basic Terminal	Brought forward
Simplistic graphical user interface	Stage 2 – Initial GUI	Brought forward
Help page	Stage 2 – Initial GUI	Brought forward
Displaying the number of captured pieces	Stage 2 – Initial GUI	Introduced
Undo button	Stage 3 – Advanced GUI	Introduced
AI play	Stage 4 – Easy AI	Brought forward
Variable difficulty AI (3 difficulties)	Stage 5 – Tree Search AI	Brought forward
User accounts for visual customisation	Stage 6 – Accounts and Customisation	Introduced
Saving game state	Stage 7 – Saving Game State	Introduced
User accounts for match statistics	Stage 8 – Saving User Stats	Brought forward
User accounts to store game history	Stage 8 – Saving User Stats	Introduced

1.5 Data Structures

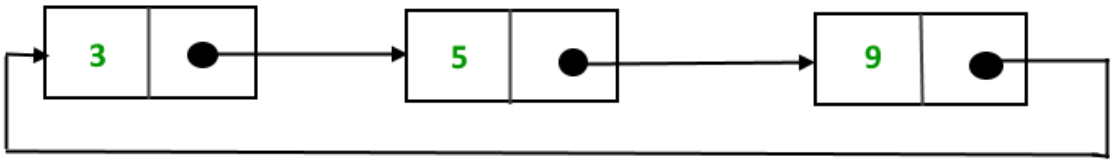
This section will provide an overview of the key data structures used throughout the project. More detailed explanations of these data structures in the context of my project can be found in the design section of this document.

The game board will be represented as a 2D array since it matches the tabular structure in which pieces can reside in. Below is an image of a 2D array.

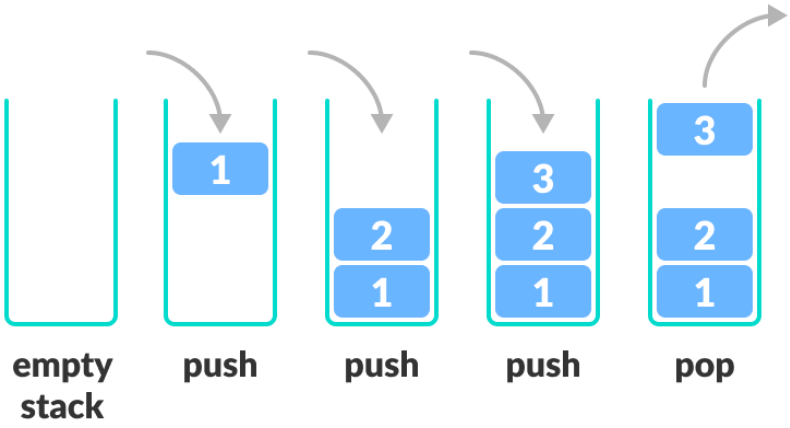


2D Array of size 3 x 5

The board's outer and inner looped tracks will each be stored as circular lists as they form an infinite cycle with no start or end position. These circular lists can be traversed in either direction (forwards or backwards iteration), allowing checks for all possible captures from a given position. Below is a visualisation of a circular list and how iteration in the forwards direction would operate.



A stack will be used to facilitate the undoing of moves because it is a last-in first-out data structure which enables the most recent moves to be undone first by popping off the stack. A stack underflow error can be handled, where the user is informed that no more moves can be undone. Below is visualisation of stack operations.



A tree is used to make a game tree as part of the Monte Carlo Tree Search (MCTS) algorithm used for the medium and hard AI opponents. The root node is the current game state. Child nodes in the tree are positions that can be reached in one move from the parent game state. Moves alternate between players as depth increases in the game tree.

1.6 Algorithms

The following section outlines some of the key algorithms used in this application. More detailed information about these algorithms is found in the design section of this document. Note that the algorithms for generating all legal moves for a board and generating a single random legal move for a board are not included in this document's analysis section but are included in the design section. This is because they are largely just extensions of non-capturing and capturing legality checks.

1.6.1 Checking Non-capturing Move Legality

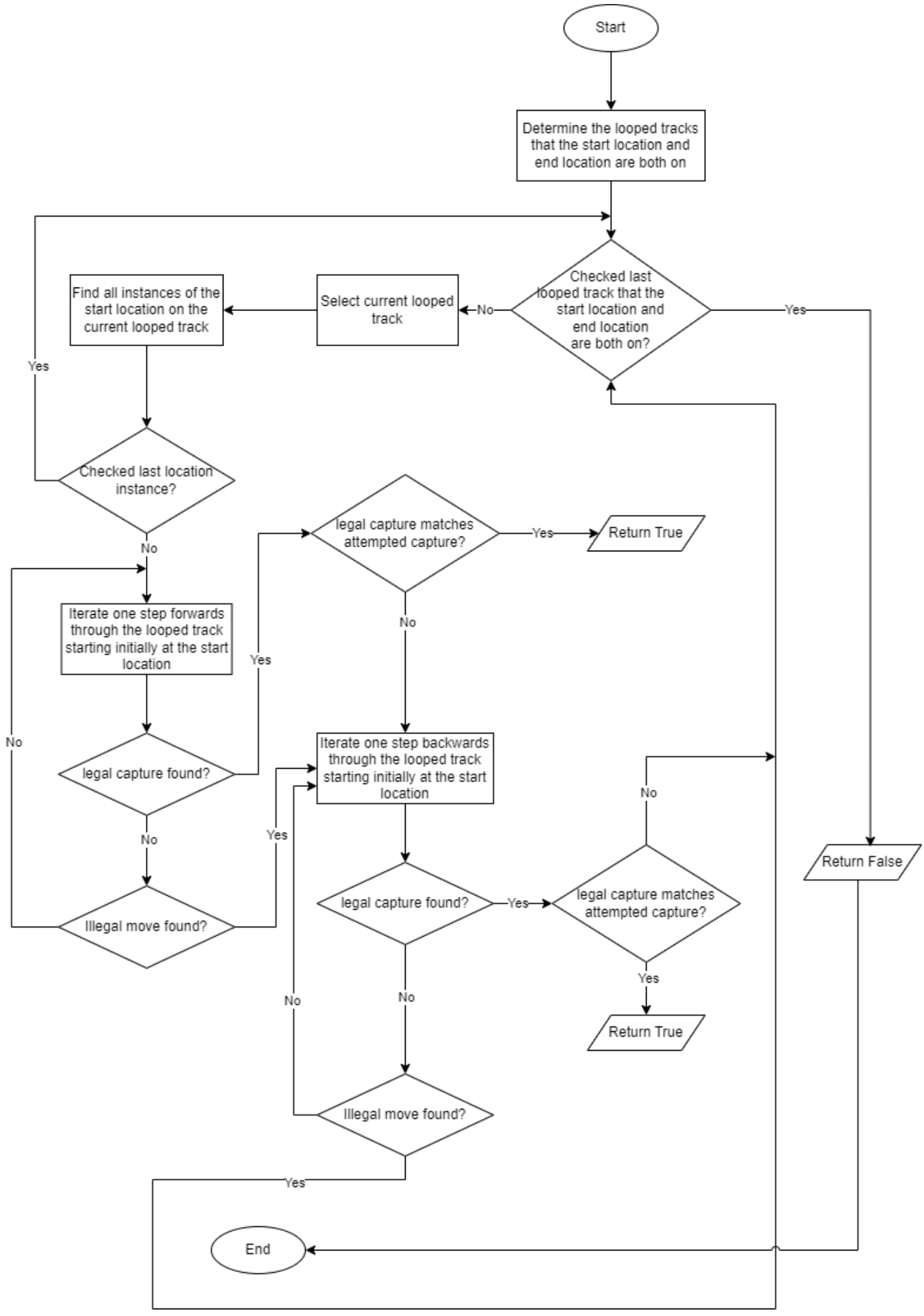
An algorithm is used to determine if a non-capturing move to an adjacent location is legal. For a non-capturing move to be legal, the attempted move must be:

1) Using the player's own piece when it is their turn
2) Moving to an unoccupied position
3) Moving to an adjacent position

If these three conditions are met, the move is legal. Otherwise, it is illegal.

1.6.2 Checking Capturing Move Legality

Checking capturing move legality involves the board's looped tracks which are stored as circular lists. The following flowchart describes the broad steps taken in this algorithm.



Each occurrence of the attempted capture's starting location (a GridLocation object) is iterated forwards and backwards from in both circular lists that the starting location and ending location are on. This has

the effect of attempting to capture the desired piece in every direction using all available looped tracks from the starting location.

Iteration in a direction through the looped track continues until a condition is met indicating that a capture can no longer be made in this direction. If all directions have been iterated in (forwards and backwards) from each occurrence of the start location in each circular list and no legal capture has been found, the attempted capture is illegal. If a legal capture is found during iteration and it matches the capture attempted, the capture attempted is legal.

Detailed explanations, pseudocode, flowcharts, diagram examples and dry runs of this algorithm are in the design section of this document.

1.6.3 Checking if a Game has Ended

In Surakarta, a legal move can always be played, meaning after each move, we do not need to check if any legal moves can be made. If no pieces of one colour are present on the board, the player who still has pieces is the winner. Only after a capturing move does the program need to check for a winner.

1.6.4 Making Moves

An algorithm will be needed to make a move on the board. When a move is made, it first needs to be pushed to the move history stack. If the move is a normal move, the pieces at the start and end locations of the moves need to be switched in both the 2D array and circular lists. For a capture, the piece at the start location needs to displace the end location in both the 2D array and circular lists.

1.6.5 Easy AI with a Greedy Algorithm

The easy AI opponent will use a greedy algorithm in combination with randomly chosen moves. The computer will always make a capturing move where possible. Where no capturing move can be made, the computer attempts to move a piece off one of the four corners of the board. If no capture or corner moves can be made, a piece is randomly moved.

1.6.6 Minimax

In this section, the minimax algorithm will be described and evaluated for use for the medium and hard AI opponents. Note that after research, it was decided that the Monte Carlo Tree Search (MCTS) algorithm would be a better option and so minimax was not used in the project.

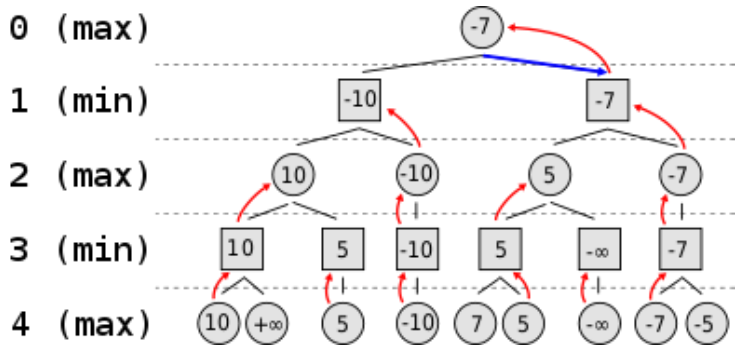
Minimax is a recursive backtracking algorithm used in two-player strategy games. It searches a game tree of possible moves to find an optimal move in each position. The algorithm uses an evaluation function to assign each game state in the tree a value, quantifying which player is winning.

There are two ways in which the medium and hard AI could be implemented. In one way, a weaker evaluation function could be used for the medium AI and a stronger evaluation function could be used for the hard AI. Another method would be to use the same evaluation function for both difficulties but increase the tree depth for more difficult AI opponents. Varying tree depth is a more reliable way of varying difficulty as it can be difficult to predictably vary the strength of an evaluation function.

The depth first search algorithm is used to create a game tree whose root node is the current game state. Child nodes of a node represent all possible game states that can be reached with one move. Ideally, the leaf nodes of the game tree would be terminal game states, meaning at those states the game has ended. However, due to computational limitations, the maximum depth of the tree is limited, and an evaluation function is instead used to determine how well each player is doing in each position. Evaluation functions are specific to the game they are being used for. The evaluation function is then

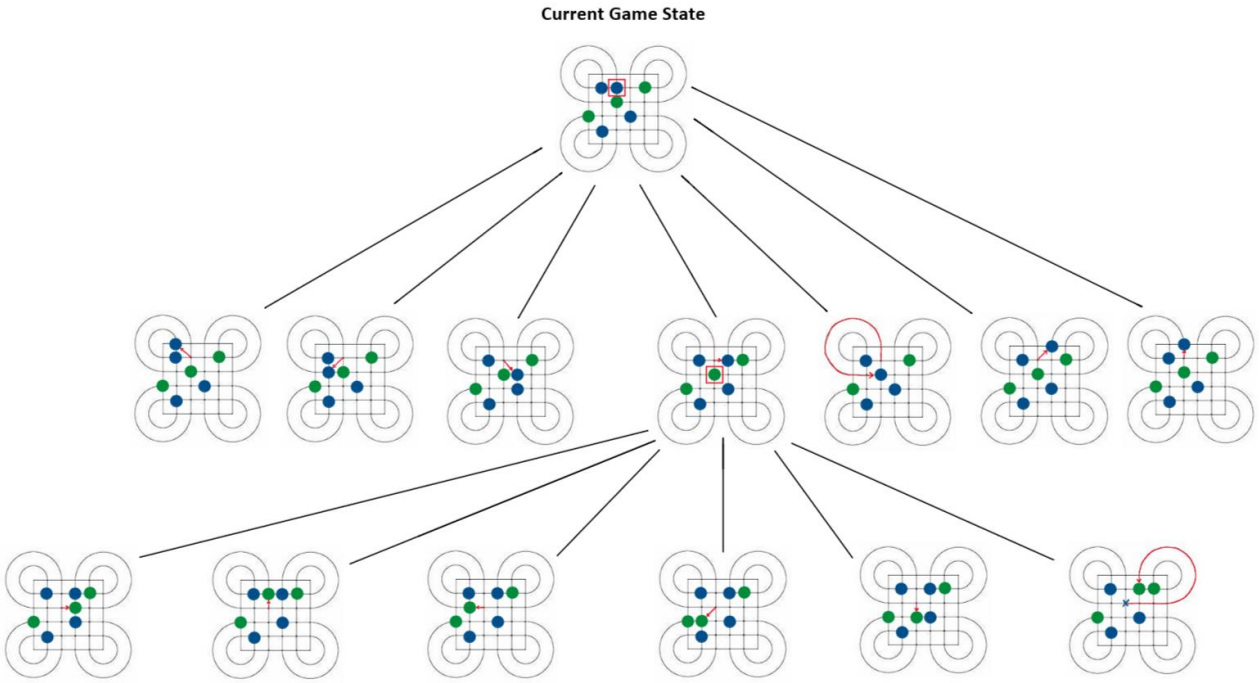
used to assign a score to the leaf nodes which is backtracked up the tree to assign a score to all nodes in the tree. Moves alternate between a maximise and minimise player. The maximise player aims to maximise the board's evaluation and the minimise player aims to minimise the board's evaluation. If a node represents a game state which is the maximise player's turn, the node's evaluation is taken to be the maximum evaluation of its child nodes. If a node represents a game state which is the minimise player's turn, the node's evaluation is taken to be the minimum evaluation of its child nodes. If the root node's evaluation is positive, the maximise player is winning and if it is negative, the minimise player is winning.

The diagram below illustrates a game tree with a depth of four where at each move there are two possible moves.

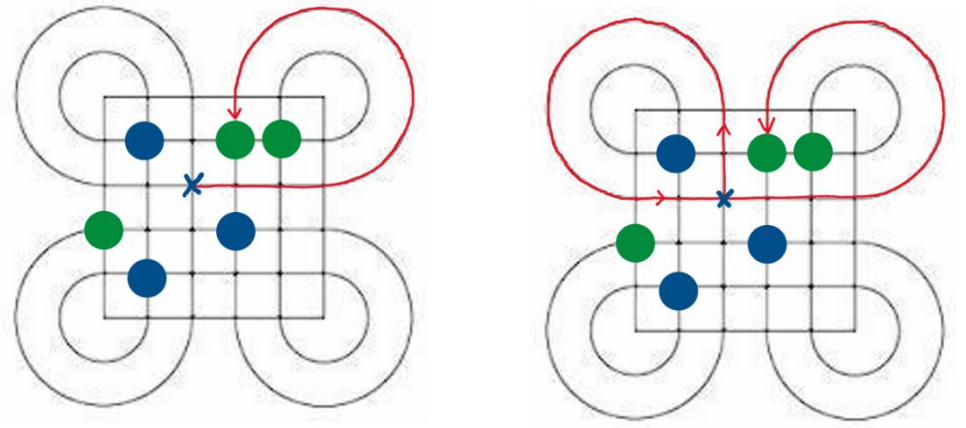


The root node was evaluated at -7, meaning the minimise player is winning. Since it is the maximise player's turn, they make the move that leads to the game state with the highest evaluation out of their child nodes which in this case is an evaluation of -7.

Branching factor is the average number of moves that can be made from a random board position and higher branching factors are found in more complex games. Surakarta has a high branching factor of 22 (Winands, 2015) while a simpler game like connect four only has a branching factor of 7. The game's very high branching factor would limit the depth of the minimax game tree. In general, minimax is not as well suited as the MCTS algorithm (discussed in section 1.6.7) for games with high branching factors. The tree diagram below illustrates the possible moves for only two pieces on the Surakarta board over two turns. These pieces are enclosed in a red square.



Note that while in some cases a capture can be made in more than one-way, different paths are abstracted away as the minimax algorithm will only deal with the end results of captures. This same principle applies to MCTS. For example, the capture in the 2nd layer of the game tree above can be made in one of two ways shown in the image below.



Another disadvantage of minimax is that it assumes the opposing player is playing optimally. The game tree also grows very quickly for complex games with many possible moves for each game state which can limit the effectiveness of the algorithm by limiting the number of moves the computer can look ahead. Minimax is also heavily reliant on a good evaluation function which can be difficulty to come up with for games with little research like Surakarta.

1.6.7 Medium and Hard AI with MCTS

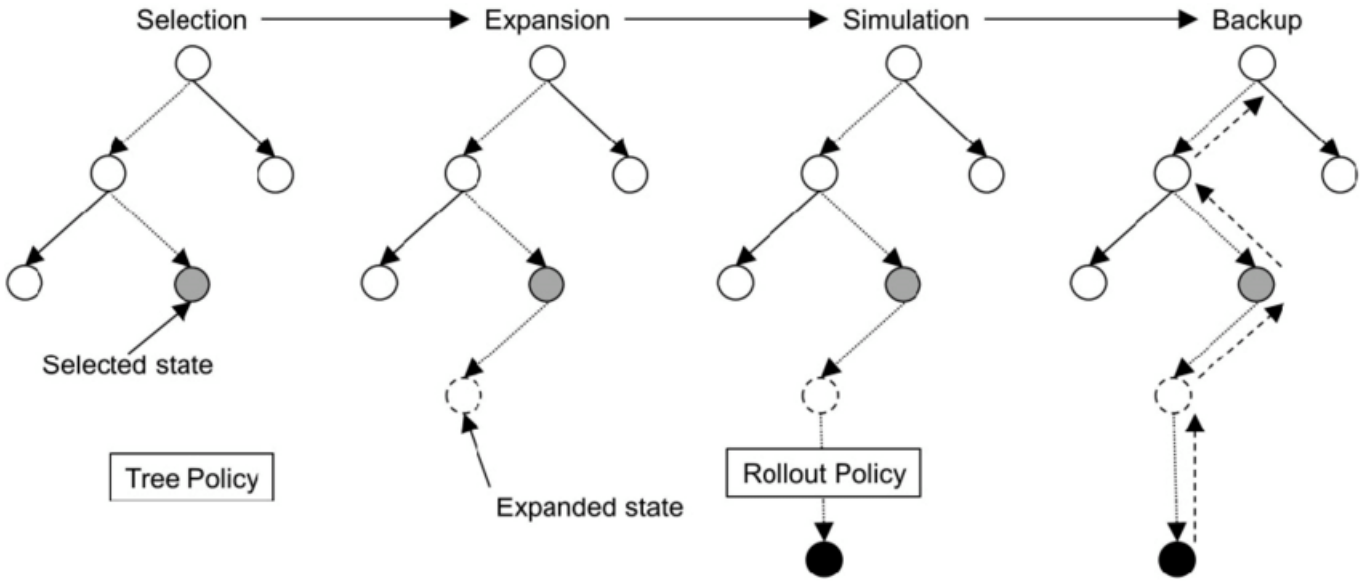
The Monte Carlo Tree Search (MCTS) algorithm is another tree search algorithm for board games. MCTS can be divided into four stages:

1) Selection
2) Expansion
3) Simulation
4) Backpropagation

These four steps form one iteration of MCTS. Many iterations of MCTS are performed, resulting in a game tree being created.

Each node in an MCTS tree has its own associated value and the number of times it has been visited so far. These two variables are used by the UCB1 formula to determine the most promising node in the selection stage.

In the selection stage, the most promising child node is repeatedly selected until a leaf node is reached. If this leaf node has never been visited before, we simulate the outcome of this node (called a rollout) by randomly playing moves for each side until the game reaches a terminal state. The result of this rollout is then backpropagated up the tree, meaning each node that was selected in reaching the simulated leaf node has its visited count incremented and the result of the simulation added to its value. If a rollout ends with a computer victory, the value 1 is backpropagated, meaning each selected node has its value incremented. Similarly, a computer loss is represented by -1 and a draw is represented by a 0. If a selected leaf node has been visited before, it is first expanded. This is where every possible game state that can be reached in one move is added as a child node. We then select one of these new child nodes before simulating and backpropagating.



For certain games like Go, simulating a game until completion is unrealistic because of the game's extremely high branching factor. One solution in these cases is for the game to be simulated until a fixed number of moves have been made, at which point a heuristic evaluation function is used on the board. In my case I can simulate games until a terminal state but to ensure outlier rollouts don't take too much time, I have used a rollout depth of 500 moves. For my evaluation function I use the number of pieces each player has remaining to determine which player is winning. If the computer has more pieces, the rollout ends in a 1. If the human player has more pieces, the rollout ends in a -1. If both players have the same number of pieces, the rollout ends in a 0.

The UCB1 formula is as follows:

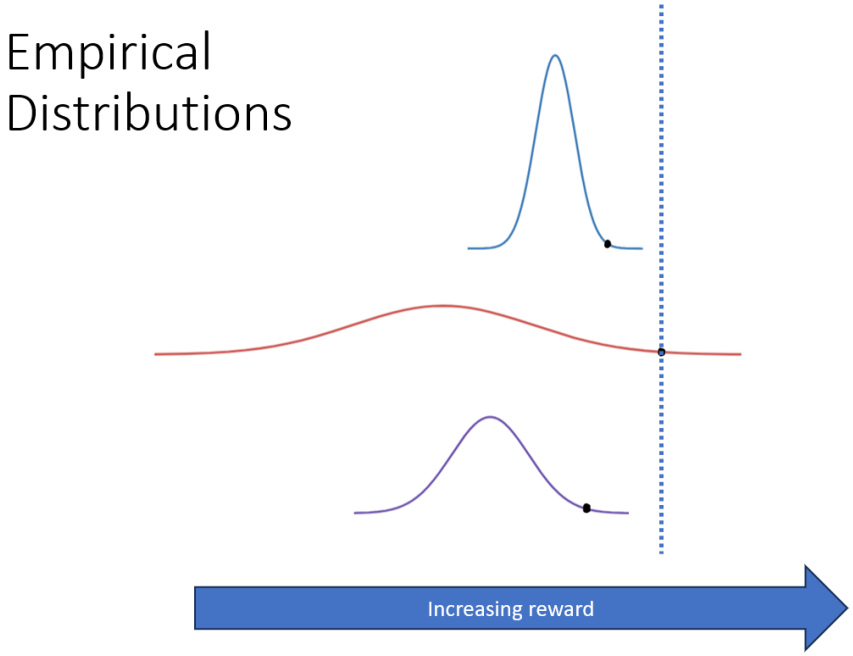
$$UCB1 = \frac{w_i}{n_i} + \mathrm{c}\sqrt{\frac{\ln(N)}{n_i}},$$

Where w_i is the value of node i, N is the total number of MCTS iterations that have been run so far, n_i is the number of times node i has been visited, and c is a constant that controls the amount of exploration that the algorithm performs. For my implementation, a commonly used value of $c = 2$ was used.

At the end of the allotted time for the algorithm to run, the move that leads to the child of the root node with the highest UCB1 score is made.

MCTS is a classic example of the multi-armed bandit problem where exploration and exploitation must be intelligently balanced. Exploitation refers to going deeper into the tree to nodes that have a high value. Exploration refers to visiting nodes that have not been frequently visited because they could end up having a higher value. Thus, a node's overall potential is increased by having a high value and a lower visited count.

As we visit a node more, we become more confident in our empirically determined reward distribution for that node.



In the example above, the red distribution has the highest UCB1 score because even though it has the lowest mean value out of the three options, it has by far the greatest variance.

While minimax can take a variable amount of time to fully search a tree to a given depth at different points in the game, MCTS can be run for a fixed length of time, with longer runtimes resulting in better play. This makes the algorithm more flexible than minimax in that more minor changes can be made between AI difficulty levels by changing the algorithm runtime as opposed to reducing tree depth since changing tree depth by even just one can led to a significant change in AI performance. Furthermore, MCTS doesn't require an evaluation function like minimax does and if one is used, MCTS can still perform well with a weak evaluation function as it also relies on the Monte Carlo method. This is advantageous for games like Surakarta without well-established evaluation functions.

1.6.8 Undoing Moves

As mentioned in section 1.5, a stack will be used to facilitate the undoing of moves. The algorithm for undoing moves must restore the state of the main board and looped tracks to before the move being undone was made. It must also restore pieces to the board when captures are undone and update player piece counts.

1.6.9 Serialising a Board State

An algorithm will need to be implemented to convert a board state to a string representation so that this string can be stored in the database to save a game. The string can then be retrieved from the database to reconstruct the board when a game is being loaded.

1.6.10 Analysis of Tree Depth

Before implementing a tree search algorithm, we can estimate a lower bound on the depth we expect our tree to be able to reach. The maximum number of nodes in the tree (N) is given by,

$$N = 22^0 + 22^1 + 22^2 + \cdots + 22^d$$

where d is the depth of the tree and 22 is the branching factor of Surakarta. This is a geometric series. The sum of the first n terms of a geometric series (S_n) is given by,

$$S_n = \frac{a(r^n - 1)}{r - 1}$$

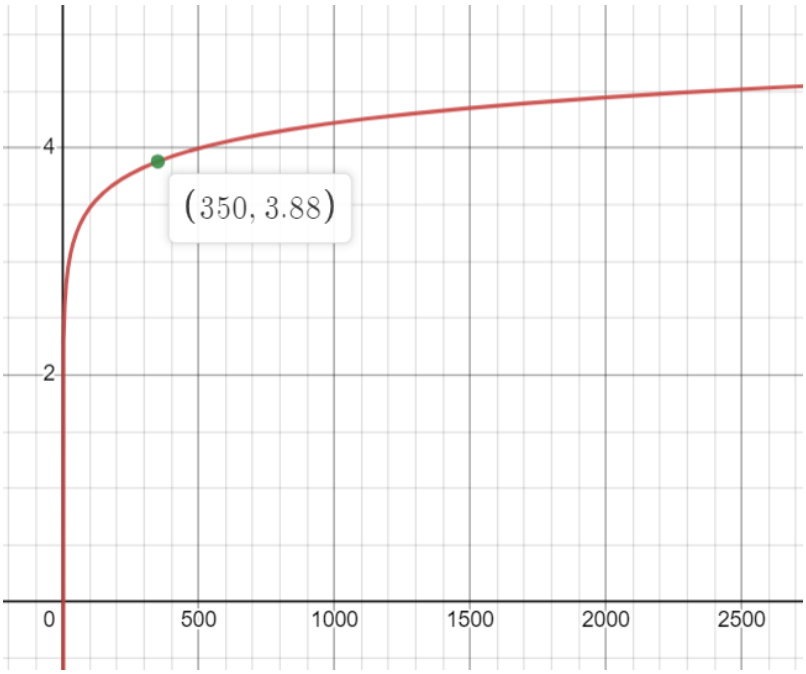
where r is the common ratio and a is the first term. In our series, this gives,

$$N = \frac{22^{d-1} - 1}{21}$$

which we can rearrange to give,

$$d = \log_{22}(21N + 1) + 1$$

Empirical testing running the MCTS algorithm for 15 seconds showed that the number of nodes in the tree at the start and middle of the game saw a minimum of around 350. The graph $d = \log_{22}(21N + 1) + 1$ has been plotted below, with the key point mentioned highlighted.



The graph suggests that at the start and middle of the game, we should be able to explore 3-4 moves ahead and in the end game, we should be able to explore deeper in the tree.

This model assumes that every node in the tree is explored which is true for the base minimax algorithm. However, these figures are estimates because the MCTS algorithm does not explore every possible node in the tree and branching factor is an average over the whole game (actual branching factor will vary throughout the game).

1.7 Identification of the prospective user

Surakarta is a very strategic game so is aimed at people who enjoy solving complex problems.

An interview with the client of this project was conducted to determine the necessary features for the game. They are an avid board game player who enjoy the unique nature and strategic play of Surakarta but are not satisfied with existing solutions.

Below is the full transcript of the interview.

Thank you for coming here to interview today. What are your favourite types of games to play and why?

So, I really like old historical strategy games. I find that they help you think more critically and that they are typically more engaging than more modern games that I know.

And would you say you're also really interested in the history and finding out more about these games?

Yeah, I think the history is cool. I've always had a passion for history and ancient games which is why I'm really interested in Surakarta because it's an unknown ancient Indonesian board game.

So specifically, the Indonesian game Surakarta, which you asked me to make, what drew you into it?

So, I really like the fact that not many people know about this game. It has really simple rules, but it can lead to very complex games. And I love the mechanics of the game as well, such as skating around the board to capture other players. I've never encountered anything similar to the unique capturing mechanic in Surakarta in any other board game.

Okay, so you like the fact that it has got simple rules, but it involves complex strategy.

Yeah.

How do you usually play these board games?

Typically, I play board games on the computer because I think it's just so much more convenient because you don't have to worry about losing pieces and it's quicker to setup. These issues are addressed by a digital version of a game. But with Surakarta, I play with a physical board because I'm not fully satisfied with the current applications available. Hence I want you to make a digital version of the game.

Overall, what aspect of strategy games makes them your favourite?

So, I really like the fact that you need to think sharply while you're playing the game so you can outwit your opponent. There's often like a lot of different strategies you can use, and it's also really satisfying to get better at these games. Different players also have different play styles which can make it a varied and interesting experience.

So how did you get into strategy games?

So, I often play strategy games with my family a lot, and I really like the fact that we can play with each other and it's always really competitive when I play with my family. And

then I started playing more games against the computer and AIs, which allows you to vary your difficulty. And especially when you play against the computers, you can easily improve as you can eventually get to harder difficulties.

So would you like an AI opponent in the application I am making?

I definitely think the game should have one. Sometimes I don't want the stress of playing against a real person or there isn't anyone around to play. Like if I just wanted a quick game or wanted to play at an odd time I would definitely rather play against an AI.

- Led to the inclusion of an Easy AI opponent (objectives 47-53)

Okay, so in terms of the actual AI, would you like multiple difficulty levels?

Yeah, I think multiple levels of difficulty would help an individual improve a lot because as you begin to find the easy AI too easy, then you can move to a more challenging AI. It's also really satisfying seeing yourself move up the difficulty levels. And it's also good for keeping me engaged because personally, if I just find myself beating the easy AI over and over again, it would get a bit boring. I'd say three levels of difficulty is the sweet spot.

And I guess in your family, if you could say, are you the best or are there people better than you?

I'd say I'm the best in my family.

But your family would need like an easier AI to play against when they're practicing.

Yeah.

- Led to the inclusion of three levels of AI difficulty (objectives 54-59)

Okay, so when you play strategy games, do you usually play online, locally or against a computer?

Generally, when I play with other people, it's often done in person because I think that adds more to the social aspect rather than playing online over computer. So, yeah, I'm not fussed about the online part of the game and I don't need it included. Typically, when I play, I normally play either in person with someone or by myself against the computer.

So, you say an online mode is not important for you?

No. But I would definitely want a local play mode where I can play against someone on the same computer.

- Led to the inclusion of local play against another player on the same computer (objectives 6-13)

Okay, so if you were playing, what would you like the game to look like?

Surakarta has pretty simple rules, so I'd like a simple graphical user interface to go along with it, but I want it to be nice on the eyes as well. So simple but aesthetic. It shouldn't have a design that's so intrusive that it takes you away from the game board, which at its core is the most important part of the program. But there are also some basic features it would need like a “help” page to display the rules and a “quit” button to exit the program. Oh, and during a game it should show the number of pieces captured for each player and who’s turn it is. As well as those, a restart button would be a nice quality-of-life feature. And obviously it needs other buttons for any other features that we decide on.

You don't want it to be completely boring and black and white. But would you like some colour?

Yeah, I would but not too much. Probably a darker colour scheme.

- Led to the initial graphical user interface design (objectives 14-41)

What are some other features of the graphical user interface you would like to see?

I think in game I would like an undo button, so I can take back a mistake and learn from it. It would also be great for when I play against some of my family who are worse than me to give them a chance.

- Led to the undo button in the advanced GUI (objectives 42-46)

Is there anything else you want to request about the user interface?

I would also like a terminal version of the game which can function as a minimal viable product. It should allow me to simply play a local play game of Surakarta which I would value because every now and then, especially at night, I just want something super simple to look at and use.

- Led to the inclusion of a terminal version of the game (objectives 1-5)

Do you think it'd be useful to have an account system?

I would love an account system because it would be nice to maybe customise the user interface.

What kind of customisation are you looking for?

Just something like changing the colour of your piece would be a really nice touch. Especially because I don't think my little sister would play if she couldn't use a red counter! But there shouldn't be too much customisation in the game because that would be too confusing.

- Led the ability to customise the graphical user interface and have customisation settings saved to an account (objectives 60-81)

Do you think it'd also be useful to keep track of your progress?

Yes, I think having an account to keep track of player stats would be great, and the ability to customize your experience with Surakarta would be a really good idea.

Would you like player stats to be recorded for an account? What kind of stats would you like to be recorded?

So, I would like to see how good I am overall against each AI difficulty. I think just the number of wins and losses against each difficulty would be perfect because it's super easy to understand and shows you a good metric of performance. And while I guess it's not strictly a stat, I want to also be able to view my match history to show the games I've lost and won because I think it's really satisfying being able to have a long history log.

- Led to user stats and game history being stored in an account (objectives 82-100)

Do you think accounts should be mandatory to play the game?

No, I think you should be able to play as a guest without logging in too. Like, if you don't login you can play a match, but the match obviously won't update any user stats. If you play as guest, you should be able to input the names of player 1 and player 2 for local play and just player 1 for AI play. But if you're logged in, you should only have to input player 2's name for local play and no name for AI play because the game should use the logged in player's username.

- Led to varying input forms prior to a match depending on whether a user is logged in or not (objectives 47-51)

When do you think you'd most likely be playing the game?

Probably while I'm studying because I can just play for a few moves and then come back to it later. So, while I won't finish a game in one sitting because the games can be quite long, I would definitely play bits and pieces throughout the day.

So, would you want to have the ability to save your game state?

Yes, I think that would be good because games may go a bit longer than what I intended and sometimes you don't get to finish your game. So yeah, I'd like the option to save my game state for later. And these saves can be tied to accounts which would make things neater. I think this would be great for me as currently I play Surakarta with a physical board which makes it quite tricky to save a game as it is and come back to it later. And since games can sometimes be quite long, I can see myself playing a single game throughout the day whenever I get a chance.

- Led the ability for a game state to be saved and loaded again later (objectives 61-69)

Do you want it to be a mobile app or a desktop app?

I'd say desktop app is probably better because I generally like the bigger screen because on my tiny phone, I can't really see where all the pieces are very well. So, yeah, I'd say a desktop app is probably better and it's also easy to load up when you're doing work as I'll be working on the same desktop device. I can just alt tab and go straight onto Surakarta.

Would you say you are satisfied with the options on desktop already?

No, I don't like the options on desktop.

I've just got some currently available options to show you here. This is the one desktop option available right now. What do you like about it?

So, I really like the fact that it's a simple UI. And I also like the fact that you can play against the computer or play locally with people who may be sitting next to you.

So what do you not like about it?

So, on this particular application, there's only one difficulty, so I guess once you've mastered that, it will get really boring really quickly. There's also no undo move button. Let's say you have suddenly moved a piece by accident, you can't undo that, which I think is quite annoying. And there's also no accounts or saving game state.

That was a desktop app. Now this is a mobile example. So what do you like about this one?

So, I like the fact that on this one there's an account system because it helps keep track of your progress. And I also really like the fact that the AI in this one has three difficulty options so you can choose the right one for you.
And then I've just got one final example to show you which is another mobile app here because sadly, there aren't many options available on desktop to show. So, what do you like about this one?

So, I like the fact that you can change difficulty as well on this one, similar to the last application, and also on this one, there's local play. The menu screen is also very aesthetically pleasing.

And what's bad about this one.

So, the multiplayer mode doesn't work on this application and there's no accounts to track your progress. So, you can't really see how well you've improved. There's no undo button either to take your move back.

So, overall, you like some things from these programs, but none of them have all the things that you want.

Yeah, that's it. All the features I want are not in a single application and some of the features like an undo button aren't in any of them.

Thank you very much.

1.8 Numbered measurable, appropriate specific objectives of the project

The following section describes the 100 objectives of this project.

MVP - Basic Terminal

1. The program can be executed in terminal mode by entering "t". The program will keep asking the user for input until the input is valid (only "t" is a valid mode before the GUI has been implemented)

2. The program asks the user to enter a name for player 1

3. The program asks the user to enter a name for player 2

4. When executing in terminal mode, the board is printed after each move. The rows and columns are numbered so a coordinate system can be used to identify board cells

5. Player 1 uses yellow pieces denoted by "y" and player 2 uses green pieces denoted by "g"

6. The program asks the current player to enter the type of move (normal adjacent move or capturing move specified by a string) that the player wants to make. If an invalid move type is

entered, an error message is shown, and they are asked again to enter a move type until a valid move type is entered

7. A player is asked to enter a row and column number separated by a comma indicating the piece they want to be moved. Any other input results in an error being reported to the terminal and the user is asked to enter the coordinates again

8. A player is asked to enter a row and column number separated by a comma indicating where they want to move their piece to. Any other input results in an error being reported to the terminal and the user is asked to enter the coordinates again

9. The program can detect whether a normal adjacent move is legal or not

10. The program can detect whether a capturing move is legal or no

11. Turns alternate between players playing on the same terminal locally and the name of the current player is displayed

12. The program can detect when a player has won which is when a player has captured all the opponent's pieces.

13. If a game is won, the winning player is reported, and the program terminates

Stage 2 - Initial GUI

14. The program can be executed in GUI mode when entering "g". The program will keep asking the user for a mode until a valid mode (“g” or “t”) is entered

15. The GUI displays a "Help" and a "New Game" button on the home page and a “Quit” button in a menu bar

16. Pressing the “Quit” button closes the program

17. Pressing the “Help” button opens a window that displays the rules of Surakarta

18. A menu bar is displayed at the top of the main windows which has a “Home” button

19. If the menu bar “Home” button is pressed, the user is returned to the home page

20. When the “New Game” button is pressed, there are two input fields to enter the names for Player 1 and Player 2 which will be used during the match. Pressing the “Submit” button creates a new window with the Surakarta board displayed. Pieces are in their starting locations.

21. A “submit move” button is displayed above the game board

22. A pair of radio buttons labelled “move” and “capture” are displayed to choose between a normal move and a capturing move

23. The player who's move it is to make has their name and the colour of their pieces displayed on the screen

24. The number of pieces each player has captured is displayed

25. The board is displayed as a 6x6 grid of buttons which have images of pieces on them

26. Clicking on a position on the board highlights this position

27. If a user accidentally clicks on a position, they can click on it again to un-highlight it

28. The game only allows two positions to be highlighted at once since a move will only ever involve a start position and an end position

29. Pressing the “submit” button makes the move onscreen specified by the highlighted positions and move type choice if the move is legal

30. If an illegal move is attempted, an error popup is displayed, and the user can attempt a different move

31. When a player captures a piece, their captured piece count is incremented onscreen

32. A “show board” button is displayed above the board during a match

33. If the “show board” button is pressed, a small display board popup window appears that shows the current board state and the looped tracks. The display board can be moved on screen.

34. The game can be played while the display board is open and any updates to the game state are shown on the display board. The display board window remains ontop of the main window while open

35. The display board can be closed and opened during the match

36. A “Restart Match” button in the menu bar can be clicked during a game to reset the current match

37. If the “Restart Match” button is pressed when not in a match, an error popup is displayed

38. If a game state is a win for either player, a popup message is displayed with the winning player's name

39. After a game is completed, the program returns to the home page

40. If the “submit move” button in a match is pressed when no move type has been selected, an error popup is displayed

41. If the “submit move” button in a match is pressed when two positions have not been highlighted, an error popup is displayed

Stage 3 - Advanced GUI

42. An “undo” button is displayed during a match

43. An “undo” button can be clicked to take back the most recent move

44. If the undo button is pressed with no moves left to undo, an error popup is displayed

45. If the display board is open and a move is undone, the display board is updated to show the state of the board after the undo

46. If the undo button is pressed and a capturing move was undone, the captured piece count of the player who made the capture originally is decremented onscreen

Stage 4 - Easy AI

47. Pressing the “New Game” button on the home page opens a new window which displays a button for “Local Play” and “AI Play”

48. When the “Local Play” button is pressed, there are two input fields present for the user to enter player 1’s name and player 2’s name. A submit button is also shown.

49. When the “AI Play” button is pressed, there is one input field present for the user to enter their name. A submit button is also shown.

50. Pressing the “Submit” button after filling in the local play fields starts a match in local play using the player names entered.

51. Pressing the “Submit” button after filling in the AI play field for player 1’s name starts a match against the Easy AI with the entered name being used.

52. In AI mode, the computer makes moves against the human player with a greedy algorithm

53. When the undo button is pressed in a match against the AI, the most recent human player and AI moves are both undone

Stage 5 – Tree Search AI

54. After pressing the “AI Play” button, a slider appears to allow a difficulty level (1, 2 or 3) to be chosen

55. Using difficulty 1 results in an easy AI opponent playing against the human player with a greedy algorithm
56. Starting a match with difficulty level 2 results in a match against a medium AI opponent

57. Starting a match with difficulty level 3 results in a match against a hard AI opponent

58. The medium AI opponent uses the Monte Carlo Tree Search algorithm running for 15 seconds

59. The hard AI opponent uses the Monte Carlo Tree Search algorithm running for 30 seconds

Stage 6 - Accounts and Customisation

60. In the home page, a “Login” button and a “Sign Up” button are displayed

61. If the “Sign Up” button is pressed, the user is prompted to enter a new username, password, and piece colour and a submit button is below these fields

62. If the “Submit” button is pressed on the sign-up form and the entered username is not reserved and does not already exist in the database, a new account is created, and the user is informed of the successful account creation via a popup

63. If the “Submit” button is pressed on the sign-up form, the entered username is looked up in the database. If the entered username already exists in the database or is one of the names reserved for AI players (“Easy AI”, “Medium AI” or “Hard AI”), an error is reported to the user

64. If the “Login” button is pressed, a popup window is displayed with a form to enter a username and password with a submit button

65. If the “Submit” button is pressed in the login form, the user's credentials are looked up in the database and if they are found, the user is logged in and informed of this with a popup message

66. If the “Submit” button is pressed in the login form and a user's entered credentials are not found in the database, an error message is shown to the user

67. The main window’s menu bar displays a “Show Login Status” button

68. If the “Show Login Status” button is pressed and no user is logged in, a popup will appear saying that no user is logged in

69. When the “Show Login Status" button is pressed, if a user is logged in, a popup will appear informing them of who they are logged in as

70. The main windows’ menu bar displays a “Logout” button

71. When the “Logout” button is pressed and no user is logged in, an error popup appears

72. When the “Logout" button is pressed and a user is logged in, the currently logged in user is logged out and a popup message informs the user of the successful logout.

73. When a user is logged in and presses the “Local Play” button, they only enter one name which is the name of player 2 because the game will use their username for player 1

74. When a user is logged in and presses the “AI Play” button, they don’t have to enter any name because the game will use their username for player 1

75. The menu bar ontop of all main windows will have a “Change Piece Colour” button

76. If the “Change Piece Colour” button is pressed and no user is logged in, an error popup is displayed

77. If the “Change Piece Colour” button is pressed and a user is logged in, the user is prompted to select a new piece colour from a drop-down menu

78. When the “Submit” button in the change piece colour window is pressed, if the selected piece colour is not none, the user’s piece colour is changed in the database

79. When the “Submit” button in the change piece colour window is pressed, if the selected piece colour is none, a popup error message is displayed

80. A user's customisation choices are saved to their account so when they login, their customisations are used

81. By default, the computer player and player 2 in local play will use green pieces. If a user is logged in and has green as their piece colour, the computer / player 2 will use yellow pieces.

Stage 7 - Saving Game State

82. In the home page, a “Load Game” button is displayed

83. If the “Load Game” button is pressed when no user is logged in, an error message is displayed via a popup

84. If the “Load Game” button is pressed and a user is logged in, a load game popup window is displayed which shows the games the user has saved.

85. A “Save Game” button is displayed in the menu bar of main windows

86. If the “Save Game” button is pressed while not in a match, an error message is shown via a popup

87. If the “Save Game" button is pressed during the game when a user is not logged in, an error is shown via a popup

88. If the “Save Game” button is pressed during a game when a user is logged in, the current game state is saved to the user’s account and a popup displays saying the game has been saved

89. The load game popup window can accept a game ID as input to load a specified game to be continued after a “load” button is pressed

90. If the current match being played is loaded and the current game state is saved, the old save for the loaded game is deleted and a saved game is deleted when it is completed

91. The load game popup window can accept a game ID as input to delete a specified game so that it is no longer saved after a “delete” button is pressed

92. If the game ID passed to the load game input form is empty or doesn’t match a game ID saved for the user when the “load” button is pressed, an error popup message is displayed

93. If the game ID passed to the delete game input form is empty or doesn’t match a game ID saved for the user when the “delete” button is pressed, an error popup message is displayed

Stage 8 - Saving User stats

94. A “Show Stats” button is displayed on the home page

95. If a user is not logged in and presses the “Show Stats” button, an error message is displayed via a popup

96. If a logged in user presses the “Show Stats" button, a stats window appears.

97. The stats popup window has a table showing their wins and losses against each AI difficulty. The initial values for wins and losses against each AI difficulty are 0.

98. The stats popup window has a match history table with columns for each game’s game ID, date, opponent, and the name of the winning player

99. After a match against an AI player is completed, a logged in user’s stats against the corresponding AI difficulty in the stats table are updated

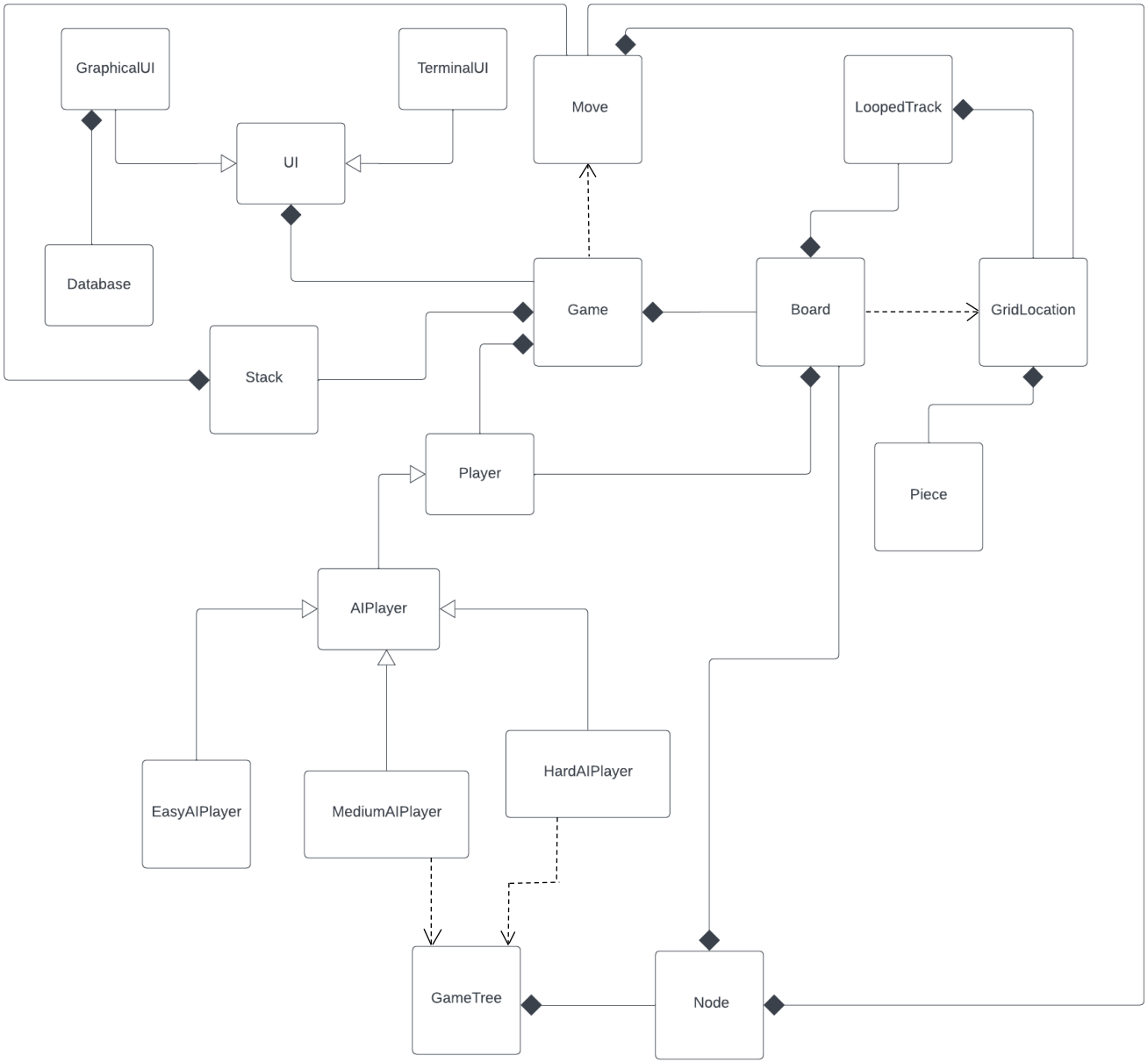
100. After a match is completed, the player’s username, the match’s game ID, the game’s date, the opponent’s name, and the name of the winning player are added as a new row in the game history table

1.9 Modelling diagrams

The following section includes a high-level UML diagram and a high-level database entity relationship model diagram.

1.9.1 High-level UML

Below is a high-level UML diagram, displaying the relationships between classes used in this project. The MultiClassBoardAttributes class is a data class, used by some of the classes in this project. It has not been included in this diagram to avoid clutter.



The table below describes the purpose of each class used in this project.

Class Name	Purpose
UI	An abstract base class which enforces child classes to implement the UI type being used, a game object and requires a play_game method to be implemented.
TerminalUI	A child class of the UI class which displays the game in the terminal to be played with ASCII characters representing the game graphics.
GraphicalUI	A child class of the UI class which displays the game with a graphical user interface with the PySimpleGUI module.
Game	A class which stores a Board object, creates the Player objects, manages player turns, contains

	the move history stack, and determines when a game is over.
Board	A class which stores the board state (2D array of GridLocation objects and two LoopedTrack objects), handles moving pieces on the board, checks move legality and serialises the board to be stored in the database. The Board class also stores a reference to the Player objects since nodes in the MCTS tree are Board objects. Storing the player objects in Board as well as in Game enables MCTS to access the player objects without needing to create more instances of Game objects. All other uses of player objects are handled in the Game class.
LoopedTrack	A class which implements a circular list data structure to be used for the inner and outer looped tracks of the board. The class enables moving and removing pieces from GridLocation objects on the LoopedTrack as well as traversal of the data structure in either direction.
GridLocation	A class which encapsulates information about a position on the board by storing its coordinates, the looped tracks that a location lies on and the piece object that is present at the location.
Move	A class which represents a move being made on the board. It stores the start and end GridLocation objects of the move, whether the move was a capture or a normal move, the colour of the piece at the start location and the colour of the piece at the end location. Colours need to be stored to maintain information integrity since a GridLocation's stored Piece can change throughout the game.
Piece	A class which represents a Piece at a GridLocation and stores the colour of the piece.
Player	A class which represents a base Player object, storing the player's name, piece colour, and the number of pieces the player has. This class is used to make human players and AI player classes inherit from this class.
AIPlayer	An abstract base class which inherits from Player and defines the abstract method get_move which must be implemented by all its child classes.
EasyAIPlayer	A class which inherits from AIPlayer and implements get_move with the greedy algorithm described in section 1.6.5 and section 2.4.7.
MediumAIPlayer	A class which inherits from AIPlayer and implements get_move to perform the Monte Carlo Tree Search algorithm. The algorithm is run for 15 seconds, and each rollout is performed up to a

	depth of 500 moves before a static evaluation function is used.
HardAIPlayer	A class which inherits from AIPlayer and implements get_move to perform the Monte Carlo Tree Search algorithm. The algorithm is run for 30 seconds, and each rollout is performed up to a depth of 500 moves before a static evaluation function is used.
Stack	A class which implements a stack data structure. The stack is used to make the game history stack in the Game class.
Node	A class which represents a single node in a GameTree object. Each node has a Board object and has a value and visited count attribute which are used as part of the Monte Carlo Tree Search algorithm.
GameTree	A class which is used to implement a game tree and executes the Monte Carlo Tree Search algorithm on itself to determine the next move for an AI opponent. Used by medium and hard AI opponents.
Database	A class which contains methods to execute SQL commands to request and send data to a relational database. Used by the UI class when loading saved games, saving games, retrieving user stats, updating user stats, retrieving and updating game history, and when creating and logging into accounts.
MultiClassBoardAttributes	A class which contains constants about the board which are accessed by multiple classes. It also stores variables for player 1 and player 2's piece colour which can be changed with a static method. All attributes are class attributes, and no instances are made of this class. This class is not included in the UML diagram. This class is accessed in Board, Game, GridLocation, Player, GameTree, Database, GraphicalUI, and TerminalUI.

The table below describes the attributes stored in the MultiClassBoardAttributes class. Attributes in lowercase are variables and attributes in all capitals are constants.

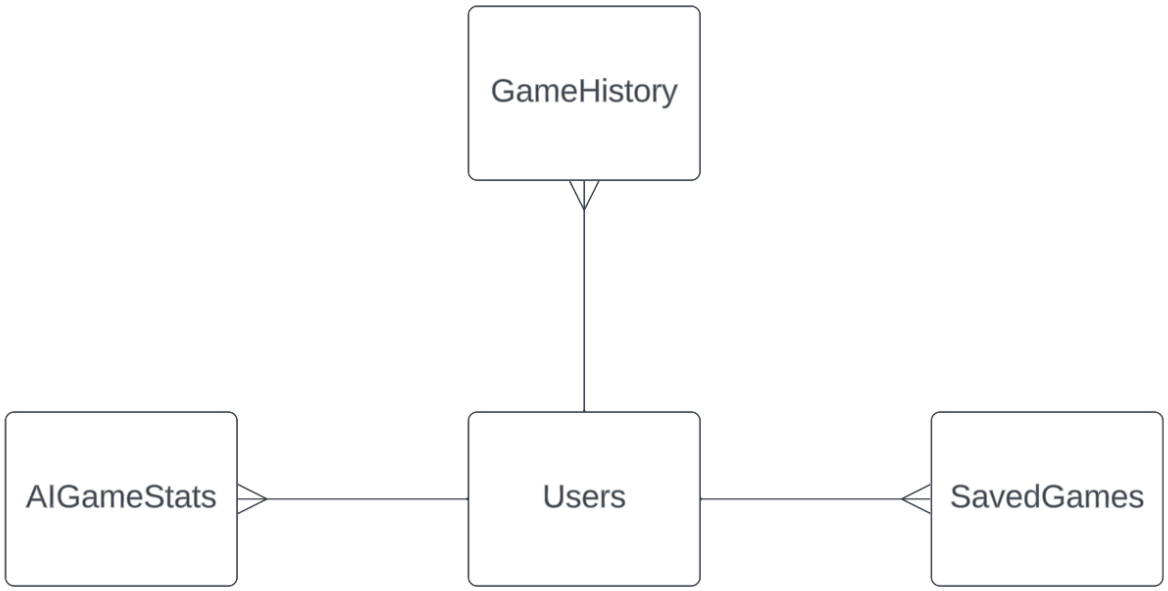
Attribute	Classes Used in	Description
player_1_colour	Board, Game, GridLocation, TreeSearch, GraphicalUI	A string representing the colour of player 1's pieces
player_2_colour	Board, Game, GridLocation, TreeSearch	A string representing the colour of player 2's pieces

DEFAULT_PLAYER_1_COLOUR	GraphicalUI	A string representing the default colour used by player 1 when no user is logged in. This is set to “yellow”.
DEFAULT_PLAYER_2_COLOUR	GraphicalUI	A string representing the default colour used by player 2. This is set to “green”.
NOMRAL_MOVE_TYPE	GraphicalUI, TerminalUI, Board	A string representing a normal move on the board. This is set to “move”.
CAPTURE_MOVE_TYPE	GraphicalUI, TerminalUI, Board	A string representing a capturing move on the board. This is set to “capture”.
INNER_TRACK_STRING	Board, GridLocation	A string representing the inner looped track on the board. This is set to “INNER”.
OUTER_TRACK_STRING	Board, GridLocation	A string representing the outer looped track on the board. This is set to “OUTER”.
BOTH_TRACK_STRING	Board, GridLocation	A string representing both looped tracks. This is set to “BOTH”.
MIN_ROW_INDEX	Board, GraphicalUI, TerminalUI	An integer representing the index used to describe the first row of the board. This is set to 0.
MAX_ROW_INDEX	Board, GraphicalUI, TerminalUI	An integer representing the index used to describe the last row of the board. This is set to 5.
NUM_STARTING_PIECES_EACH	GraphicalUI, Game, Player	An integer representing the number of pieces each player starts with at the beginning of a match. This is set to 12.
EASY_AI_NAME	GraphicalUI, Database, Game, EasyAIPlayer	A string representing the name of the easy AI player which is set to “Easy AI”

MEDIUM_AI_NAME	GraphicalUI, Database, Game, MediumAIPlayer	A string representing the name of the medium AI player which is set to “Medium AI”
HARD_AI_NAME	GraphicalUI, Database, Game, HardAIPlayer	A string representing the name of the hard AI player which is set to “Hard AI”

1.9.2 Database ER Model

Below is an entity relationship model for the tables used in this project’s database.



The table below describes each table in the relational database.

Table	Purpose
Users	Stores a user's username and password along with the account creation date and user's piece colour.
SavedGames	Stores data about a game that a user has saved. This data is used when loading a specific game.
GameHistory	Stores information about each game a user has completed. This data is used to display a table of game history information to the user.
AIGameStats	Stores the wins and losses for a user against each AI difficulty. This data is displayed in a stats table to the user.

2 Design (12 marks)

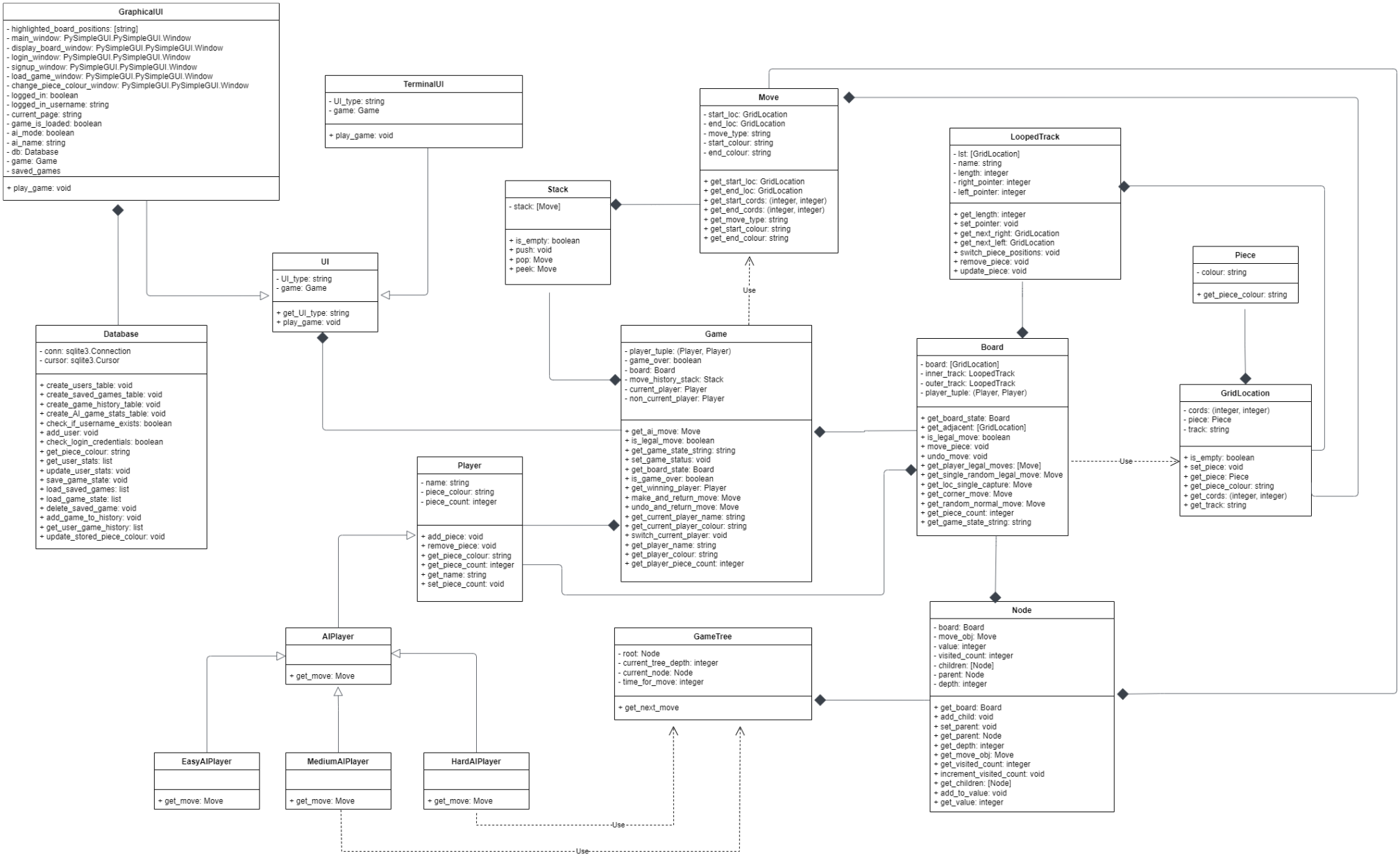
This next section is the design section of my project.

2.1 System design overview

The following section includes a detailed UML diagram with methods and attributes, data flow diagrams and a flow through the program flowchart.

2.1.1 Full UML Diagram

Below is a full UML diagram of all the classes used in this project and their relationships. All instance attributes and public methods are included.


GraphicalUI
- highlighted_board_positions: [string]
- main_window: PySimpleGUI.PySimpleGUI.Window
- display_board_window: PySimpleGUI.PySimpleGUI.Window
- login_window: PySimpleGUI.PySimpleGUI.Window
- signup_window: PySimpleGUI.PySimpleGUI.Window
- load_game_window: PySimpleGUI.PySimpleGUI.Window
- change_piece_colour_window: PySimpleGUI.PySimpleGUI.Window
- logged_in: boolean
- logged_in_username: string
- current_page: string
- game_is_loaded: boolean
- ai_mode: boolean
- ai_name: string
- db: Database
- game: Game
- saved_games
+ play_game: void
TerminalUI
- UI_type: string
- game: Game
+ play_game: void
Move
- start_loc: GridLocation
- end_loc: GridLocation
- move_type: string
- start_colour: string
- end_colour: string
+ get_start_loc: GridLocation
+ get_end_loc: GridLocation
+ get_start_cords: (integer, integer)
+ get_end_cords: (integer, integer)
+ get_move_type: string
+ get_start_colour: string
+ get_end_colour: string
LoopedTrack
- lst: [GridLocation]
- name: string
- length: integer
- right_pointer: integer
- left_pointer: integer
+ get_length: integer
+ set_pointer: void
+ get_next_right: GridLocation
+ get_next_left: GridLocation
+ switch_piece_positions: void
+ remove_piece: void
+ update_piece: void
Stack
- stack: [Move]
+ is_empty: boolean
+ push: void
+ pop: Move
+ peek: Move
Piece
- colour: string
+ get_piece_colour: string
UI
- UI_type: string
- game: Game
+ get_UI_type: string
+ play_game: void
Use
Database
- conn: sqlite3.Connection
- cursor: sqlite3.Cursor
+ create_users_table: void
+ create_saved_games_table: void
+ create_game_history_table: void
+ create_AI_game_stats_table: void
+ check_if_username_exists: boolean
+ add_user: void
+ check_login_credentials: boolean
+ get_piece_colour: string
+ get_user_stats: list
+ update_user_stats: void
+ save_game_state: void
+ load_saved_games: list
+ load_game_state: list
+ delete_saved_game: void
+ add_game_to_history: void
+ get_user_game_history: list
+ update_stored_piece_colour: void
Game
- player_tuple: (Player, Player)
- game_over: boolean
- board: Board
- move_history_stack: Stack
- current_player: Player
- non_current_player: Player
+ get_ai_move: Move
+ is_legal_move: boolean
+ get_game_state_string: string
+ set_game_status: void
+ get_board_state: Board
+ is_game_over: boolean
+ get_winning_player: Player
+ make_and_return_move: Move
+ undo_and_return_move: Move
+ get_current_player_name: string
+ get_current_player_colour: string
+ switch_current_player: void
+ get_player_name: string
+ get_player_colour: string
+ get_player_piece_count: integer
Board
- board: [GridLocation]
- inner_track: LoopedTrack
- outer_track: LoopedTrack
- player_tuple: (Player, Player)
+ get_board_state: Board
+ get_adjacent: [GridLocation]
+ is_legal_move: boolean
+ move_piece: void
+ undo_move: void
+ get_player_legal_moves: [Move]
+ get_single_random_legal_move: Move
+ get_loc_single_capture: Move
+ get_corner_move: Move
+ get_random_normal_move: Move
+ get_piece_count: integer
+ get_game_state_string: string
Use
GridLocation
- cords: (integer, integer)
- piece: Piece
- track: string
+ is_empty: boolean
+ set_piece: void
+ get_piece: Piece
+ get_piece_colour: string
+ get_cords: (integer, integer)
+ get_track: string
Player
- name: string
- piece_colour: string
- piece_count: integer
+ add_piece: void
+ remove_piece: void
+ get_piece_colour: string
+ get_piece_count: integer
+ get_name: string
+ set_piece_count: void
Node
- board: Board
- move_obj: Move
- value: integer
- visited_count: integer
- children: [Node]
- parent: Node
- depth: integer
+ get_board: Board
+ add_child: void
+ set_parent: void
+ get_parent: Node
+ get_depth: integer
+ get_move_obj: Move
+ get_visited_count: integer
+ increment_visited_count: void
+ get_children: [Node]
+ add_to_value: void
+ get_value: integer
AIPlayer
+ get_move: Move
GameTree
- root: Node
- current_tree_depth: integer
- current_node: Node
- time_for_move: integer
+ get_next_move
EasyAIPlayer
+ get_move: Move
MediumAIPlayer
+ get_move: Move
HardAIPlayer
+ get_move: Move
Use
Use

2.1.2 Attribute and Public Method Descriptions

In this section, the attributes and public methods of each class mentioned in the UML diagram are described. Attributes and public methods which have been inherited but not overridden are not included in a child class's description.

Board

- **Attributes**
  - **board** – A 2D array of GridLocation objects representing the board state.
  - **inner_track** – A LoopedTrack object which contains GridLocation objects representing the inner track on the board.
  - **outer_track** – A LoopedTrack object which contains GridLocation objects representing the outer track on the board.
  - **player_tuple** – A tuple of length two with two Player objects for player 1 and player 2 used in the Monte Carlo Tree Search algorithm.

- **Public Methods**
  - **get_board_state()** – Returns the board attribute (2D array).
  - **is_legal_move(start_loc, end_loc, move_type)** – Returns true if the move specified by move_type using the piece at start_loc to end_loc is a legal move. Otherwise, false is returned.
  - **move_piece(move_obj)** – Makes the move specified by the provided Move object argument on the board.
  - **undo_move(move_obj)** – Undoes the move specified by the Move object on the board.
  - **get_player_legal_moves(player_colour)** – Returns a list of all legal moves as Move objects that can be made by the player who has the colour specified by the player colour passed as an argument.
  - **get_single_random_legal_move(player_colour)** - Returns a single, random legal Move object that can be made by the player who has pieces of colour player_colour.
  - **get_loc_single_capture(start_loc)** – Returns a single legal capture that can be made with the piece at the start_loc GridLocation. If no legal captures can be made, None is returned.
  - **get_corner_move(start_loc)** - If start_loc is not a corner location, returns None. Otherwise, returns a possible adjacent move for start_loc and if no moves are possible, the method returns None.
  - **get_random_normal_move(player_colour)** – Returns a random move to an adjacent location that can be made using the pieces specified by player_colour.
  - **get_piece_count(player_number)** - Returns the number of pieces the player with the number specified by player_number has on the board.
  - **get_game_state_string()** - Returns a string representation of the current game state. Pieces are represented by their colour and empty locations are represented by a pre-determined character. Pieces are separated by a pre-determined character. The first and last characters are not separator characters.

Game

- **Attributes**
  - **player_tuple** - A tuple of length two with two Player objects for player 1 and player 2. Used when managing player piece counts, and the current player who's turn it is to move.
  - **game_over** – A boolean representing whether the game is over. Initially it is set to false.
  - **board** – a Board object on which the game is played.
  - **move_history_stack** – A Stack object storing the moves made on the board so they can be undone.
  - **current_player** – The Player object who's turn it is to make a move.
  - **non_current_player** – The Player object who's turn it isn't to make a move.

- **Public Methods**
  - **get_ai_move()** - Returns a Move object generated by the AI player.
  - **is_legal_move(start_loc, end_loc, move_type)** - Calls the Board object's is_legal_move method.
  - **get_game_state_string()** - Returns a call to the Board object's get_game_state_string method.
  - **set_game_status()** - Sets the game_over attribute to True if either player has no pieces left. A legal move can always be played in Surakarta, so this is the only way the game can end.
  - **get_board_state()** - Returns a call to the Board object's get_board_state method.
  - **is_game_over()** - Returns the game_over attribute.
  - **get_winning_player()** - Returns the Player object of the winning player. If the game is not over, an exception is raised.
  - **make_and_return_move(move_obj)** - Pushes move_obj to the move_history_stack and returns a call to the Board object's move_piece method with move_obj.
  - **undo_and_return_move()** - Pops a Move object off the move_history_stack, adds a piece back to the current player's piece count if the move was a capture and returns a call to the Board object's undo_move method.
  - **get_current_player_name()** – Returns the name of the player who's turn it is to make a move.
  - **get_current_player_colour()** – Returns the piece colour of the player who's turn it is to make a move.
  - **switch_current_player()** - Switches the values of the current_player and non_current_player attributes.
  - **get_player_name(player_number)** - Returns the name of the player specified by player_number.
  - **get_player_colour(player_number)** - Returns the piece colour of the player specified by player_number.
  - **get_player_piece_count(player_number)** - Returns the piece count of the player specified by player_number.

UI

- **Attributes**
  - **UI_type –** A string describing the type of UI object. This is defined by child classes of UI.
  - **game –** A Game object which is used to interact with the game. This is defined by child classes of UI.

- **Public Methods**
  - **get_UI_type()** - Returns the UI_type attribute.
  - **play_game** – An abstract method which must be defined by child classes of UI to start the game.

TerminalUI

- **Attributes**
  - **UI_type** – A string describing the type of UI object. For a TerminalUI object this is "TERMINAL". This attribute is declared in the parent UI class and is implemented when a TerminalUI object is first created.
  - **game** – A Game object which is used to interact with the game. This attribute is declared in the parent UI class and is implemented when a TerminalUI object is first created.

- **Public Methods**
  - **play_game()** – The main loop of the terminal user interface which handles all events and updates to the UI. This is the main public method of the TerminalUI class which is called in the Main.py file to launch the application in terminal mode. This method is declared in the parent UI class and implemented in the TerminalUI class.

GraphicalUI

- **Attributes**
  - **highlighted_board_positions** – A list of strings used as keys by PySimpleGUI to uniquely identify GUI piece buttons that have been pressed and highlighted on the board.
  - **main_window** – A PySimpleGUI.PySimpleGUI.Window object for the main window. This could be the home page window, help page window, new game page window or match page window.
  - **display_board_window** - A PySimpleGUI.PySimpleGUI.Window object for the display board window which shows the piece positions on a board with looped tracks displayed.
  - **login_window** - A PySimpleGUI.PySimpleGUI.Window object for the login page where a user can login to their account.
  - **signup_window** - A PySimpleGUI.PySimpleGUI.Window object for the signup page where a user can create an account.
  - **load_game_window** - A PySimpleGUI.PySimpleGUI.Window object for the load game window where a logged in user can select a saved game to load and delete saved games.
  - **change_piece_colour_window** - A PySimpleGUI.PySimpleGUI.Window object where a logged in user can change their piece colour.
  - **logged_in** – A boolean to reflect whether an account is logged in currently or not.
  - **logged_in_username** – A string which is the username of the logged in account.
  - **game** – A Game object which is used to interact with the game. This attribute is declared in the parent UI class and is implemented when a match is started in the GUI (not as soon as the GraphicalUI object is created).
  - **game_is_loaded** – A boolean which reflects whether the current match was loaded from the database or not.
  - **ai_mode** – A boolean which reflects whether the current match is being played against an AI opponent or not.
  - **ai_name** – A string which is the name of the AI opponent in the current match against an AI.
  - **db** – A Database object which is used to execute SQL commands to interact with the Database.db file.
  - **saved_games** – A list of records which are a logged in user's saved games. This list is populated when the user first requests to load a game since this data is needed for both loading and deleting specific games which can occur together in the game loading window. This storing of data prevents an unnecessary duplicate request to the database.
  - **current_page** – A string description of the current main window which can be "match_page", "home_page", "new_game_page" or "help_page".

- **Public Methods**
  - **play_game()** – The main event loop of the graphical user interface which handles all events and updates to the UI. This is the main public method of the GraphicalUI class which is called in the Main.py file to launch the application in GUI mode.

GridLocation

- **Attributes**
  - **cords** – A tuple of length two with two integers between 0 and 5 inclusive which represents the location's position on the board.
  - **track** – A string representing the looped tracks that the location is on. The track attribute is either "INNER", "OUTER" or "BOTH".
  - **piece** – A Piece object representing the piece currently at the location. If there is no piece at the location, this attribute is set to None.

- **Public Methods**
  - **is_empty()** – Returns a boolean representing whether there is a piece at the location or not.

- **set_piece(piece)** – sets the piece attribute to the piece argument.
- **get_piece()** – Returns the Piece object at the location.
- **get_piece_colour()** – If no piece is at the location, returns None. Otherwise, returns the colour, which is a string, of the piece attribute.
- **get_cords()** – Returns the cords attribute.
- **get_track()** – Returns the track attribute.

Move

- **Attributes**
  - **start_loc** – The starting GridLocation object of the move. This is the location of the piece being moved.
  - **end_loc** – The ending GridLocation object of the move. This is the location being moved to for a normal adjacent move or the location of the piece being captured for a capturing move.
  - **move_type** – A string representing the type of move which is either “move” or “capture”.
  - **start_colour** – The colour of the piece (string) at start_loc when the move was made. This needs to be stored to allow the full move to be understood and reconstructed at any point in the game as pieces at GridLocation objects change throughout the game.
  - **end_colour** – The colour of the piece (string) at end_loc when the move was made if the move was a capture. For a normal adjacent move, this attribute is None. This needs to be recorded for the same reason that start_colour is recorded.

- **Public Methods**
  - **get_start_loc()** – Returns the start_loc attribute.
  - **get_end_loc()** – Returns the end_loc attribute.
  - **get_start_cords()** - Returns the cords attribute of the start_loc GridLocation object.
  - **get_end_cords()** – Returns the cords attribute of the end_loc GridLocation object.
  - **get_move_type()** – Returns the move_type attribute.
  - **get_start_colour()** – Returns the start_colour attribute.
  - **get_end_colour()** – Returns the end_colour attribute.

Stack

- **Attributes**
  - **stack** – a list containing the elements in the stack. The Stack class is used for the move_history_stack attribute in the Game class so contains Move objects.

- **Public Methods**
  - **push(item)** – Pushes the item passed as an argument to the top of the stack.
  - **pop()** – Pops the item at the top of the stack off the stack and returns the item.
  - **peek()** – Returns the item at the top of the stack without removing it from the stack.
  - **is_empty()** – Returns true if the stack is empty and false if it contains one or more elements.

LoopedTrack

- **Attributes**
  - **lst** – A list of GridLocation objects stored in the LoopedTrack data structure.
  - **name** – A string which identifies the LoopedTrack
  - **length** – An integer which is the number of elements in the loopedTrack data structure.
  - **right_pointer** – An integer representing the current index of the right pointer used for iterating right through the LoopedTrack object.
  - **left_pointer** – An integer representing the current index of the left pointer used for iterating left through the LoopedTrack object.

- **Public Methods**
  - **get_length()** – Returns the length attribute

- **set_pointer(index, pointer_type)** – Sets the pointer attribute specified by the pointer_type argument (which is the string “left” or “right”) to the value of the index argument (positive integer between 0 and the length attribute minus one inclusive or a negative integer such that (index * -1) is a positive integer between 1 and the length attribute inclusive.
- **get_next_right()** – Returns the GridLocation at the index specified by the right_pointer attribute in the lst attribute and increments the right pointer such that if it increases past the last element in lst, it is wrapped back around to the start of lst.
- **get_next_left()** – Returns the GridLocation at the index specified by the left_pointer attribute in the lst attribute and decrements the left pointer such that if it decreases past the first element in lst, it is wrapped back around to the end of lst.
- **switch_piece_positions(loc1, loc2)** – Replaces all occurrences of the Piece object at loc1 with the Piece object at loc2 (loc1 and loc2 are GridLocation objects) in the LoopedTrack.
- **remove_piece(cords)** – Replaces all occurrences of a Piece object at the GridLocation object in the LoopedTrack with a cords attribute equal to the cords argument with None.
- **update_piece(cords, piece_colour)** – Replaces all occurrences of the Piece at GridLocation specified by cords argument with a new Piece object which has a piece_colour attribute of piece_colour.

Piece

- **Attributes**
  - **colour** – A string which is the colour of the piece.

- **Public Methods**
  - **get_piece_colour()** – Returns the colour attribute.

Database

- **Attributes**
  - **conn** – A sqlite3.Connection object that connects to the database.
  - **cursor** – A sqlite3.Cursor object that is used to execute SQL commands.

- **Public Methods**
  - **create_users_table()** – Creates the Users table if it doesn’t already exist.
  - **create_saved_games_table()** – Creates the SavedGames table if it doesn’t already exist.
  - **create_game_history_table()** - Creates the GameHistory table if it doesn’t already exist.
  - **create_AI_game_stats_table()** - Creates the AIGameStats table if it doesn’t already exist.
  - **check_if_username_exists(username)** – Returns true if the username passed as an argument is already in the Users table and false if it is not.
  - **add_user(username, password, piece_colour)** - Adds a new user to the Users table with the data passed as arguments.
  - **check_login_credentials(username, password)** – Returns true if the stored password hash in the Users table for the user specified by the username argument matches the hash of the password argument. Otherwise, returns false.
  - **get_piece_colour(username)** – Returns the piece colour from the Users table of a user given their username.
  - **get_user_stats(username)** – Returns a user’s win and loss count for each AI difficulty from the AIGameStats table given their username.
  - **update_user_stats(username, human_won, ai_difficulty)** – If human_won is true, the user specified by username has their win count incremented in the AIGameStats table against the AI specified by ai_difficulty. If human_won is false, the loss count is incremented instead.
  - **save_game_state(username, game_state_string, opponent_name, player2_starts, player1_num_pieces, player2_num_pieces, player1_colour)** – Saves the game described by the arguments in the SavedGames table.

- **load_saved_games(username)** – Returns a list of saved games for the user specified by username from the SavedGames table.
- **load_game_state(saved_game_id)** – Returns the data stored for a saved game given its saved_game_id from the SavedGames table.
- **delete_saved_game(saved_game_id)** – Deletes a saved game from the SavedGames table given its saved_game_id.
- **add_game_to_history(username, player2_name, winner_name)** – Adds a game specified by the arguments to the GameHistory table.
- **get_user_game_history(username)** – Returns a list of games that a user specified by username has played from the GameHistory table.
- **update_stored_piece_colour(username, new_colour)** – Updates a user specified by username's piece colour in the Users table.

Player

- **Attributes**
  - **name –** A string representing the name of the player.
  - **piece_colour** – A string representing the colour of the player's pieces.
  - **piece_count** – An integer representing the number of pieces the player has left on the board.

- **Public Methods**
  - **get_piece_colour()** - Returns the piece_colour attribute.
  - **get_name()** - Returns the name attribute.
  - **get_piece_count()** - Returns the piece_count attribute.
  - **remove_piece()** - Decrements piece_count if piece_count is greater than or equal to one. Otherwise, the method raises a ValueError.
  - **add_piece()** - Increments piece_count if piece_count is less than or equal to the number of pieces each player starts with (12). Otherwise, the method raises a ValueError. This method is used when undoing a capturing move.
  - **set_piece(piece_count)** - Sets the piece_count attribute to be the new value passed in as argument.

AIPlayer

- **Public Methods**
  - **get_move(board)** - An abstract method which is defined by child classes of the AIPlayer class. Returns a move (in the form of a Move object) for the AI player given a Board object as input.

EasyAIPlayer

- **Attributes**
  - **name –** The name attributed inherited from Player is set to "Easy AI".

- **Public Methods**
  - **get_move(board)** - Implements the greedy algorithm described in section 2.4.7 to return a move object.

MediumAIPlayer

- **Attributes**
  - **name –** The name attributed inherited from Player is set to "Medium AI".

- **Public Methods**
  - **get_move(board)** - Implements the Monte Carlo Tree Search algorithm described in section 2.4.8 to return a move object. The algorithm is run for 15 seconds per move.

HardAIPlayer

- **Attributes**
  - **name –** The name attributed inherited from Player is set to “Hard AI”.

- **Public Methods**
  - **get_move(board)** - Implements the Monte Carlo Tree Search algorithm described in section 2.4.8 to return a move object. The algorithm is run for 30 seconds per move.

Node

- **Attributes**
  - **board** – A board object which is the data stored by a node in the game tree.
  - **move_obj** – The Move object that led to the board attribute’s current state. The root node has its move_obj set to None.
  - **value** – An integer representing the value of the node used in the UCB1 formula.
  - **visited_count** – An integer representing the number of time the node has been visited during the Monte Carlo Tree Search algorithm. This attribute is used in the UCB1 formula.
  - **children** – A list of Node objects that are children of the Node object. This list is initially empty.
  - **parent** – A node object which is the parent of this node object. The root node has its parent attribute set to None.
  - **depth** – An integer representing the depth of the node in the game tree. The root node has a depth of 0.

- **Public Methods**
  - **get_board()** – Returns the board attribute.
  - **add_child(child)** – Takes child (a Node object) as argument and makes this new node a child of the node object that calls this method.
  - **set_parent(parent)** – Assigns the parent attribute the value of the parent argument.
  - **get_parent()** – Return the parent attribute.
  - **get_depth()** – Returns the depth attribute.
  - **get_move_obj()** – Returns the move_obj attribute.
  - **get_visited_count()** – Returns the visited_count attribute.
  - **increment_visited_count()** – Increments the visited_count attribute.
  - **get_children()** – Returns the children attribute.
  - **get_value()** – Returns the value attribute.
  - **add_to_value(value)** – Increases the value attribute by the value argument passed to this method.

GameTree

- **Attributes**
  - **root** – A Node object which is the root node of the game tree. This node’s data is the current board of the match. The root node has a depth of 0.
  - **current_tree_depth** – An integer representing the maximum depth of any node in the game tree currently.
  - **current_node** – A Node object which is the current node being examined during tree traversal in the Monte Carlo Tree Search algorithm.
  - **time_for_move** – The time that the MCTS algorithm is run for per move.

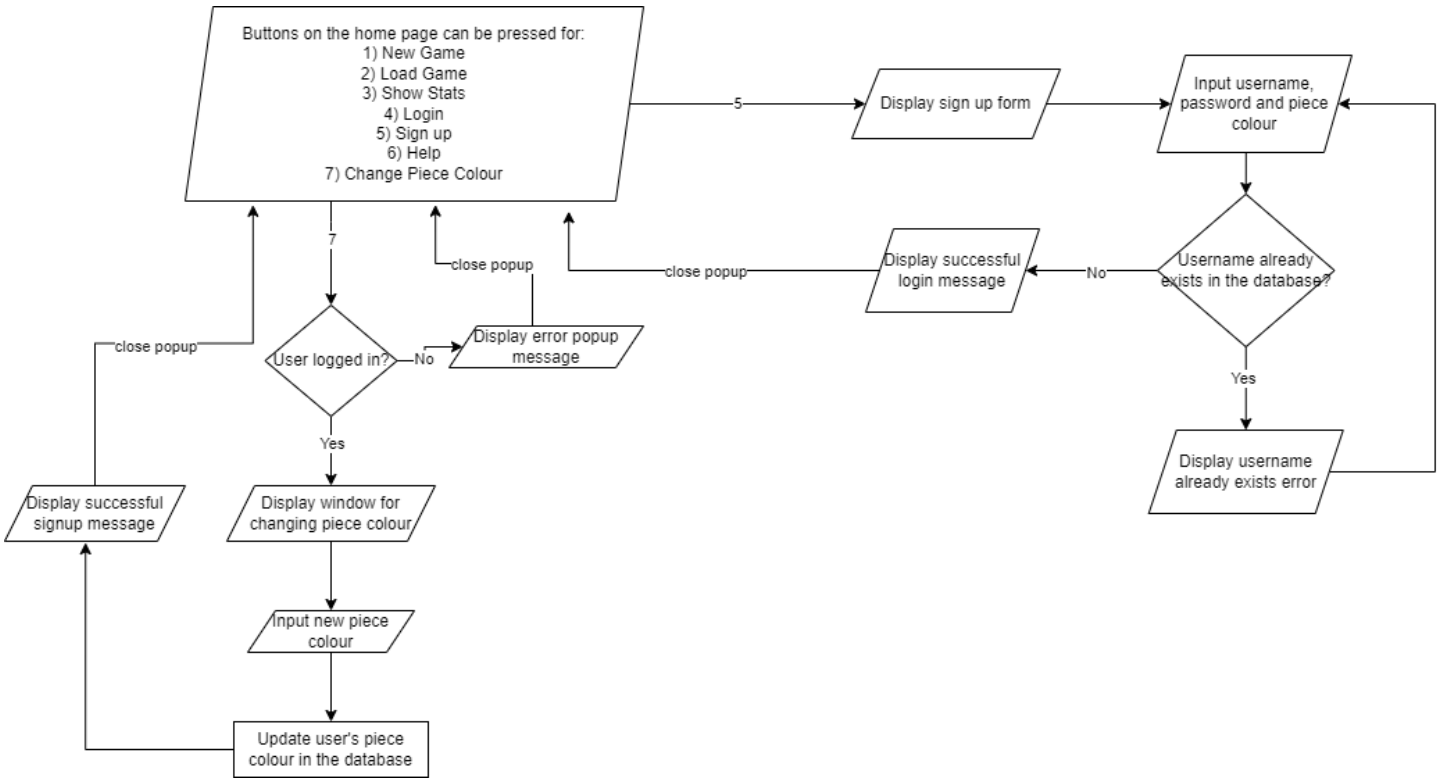
- **Public Methods**
  - **get_next_move()** – Runs the Monte Carlo Tree Search algorithm for the time specified by the time_for_move attribute starting from the root node and returns a Move object for the AI player to make.

2.1.3 Flow through the program flowchart

Below is a flowchart describing the full flow through the program from a user’s perspective. The process of signing up and changing your piece colour are in a separate flowchart for space reasons.

Start
Buttons on the home page can be pressed for:
1) New Game
2) Load Game
3) Show Stats
4) Login
5) Sign up
6) Help
7) Change Piece Colour
1
2
3
4
5
6
Quit
End
home button
Display match page with new game
Input player 1 name
Input player 2 name
Input player 1 and player 2 name
Display game mode popup and buttons for:
Local Play
AI Play
Local Play
AI Play
Input AI difficulty level
Display saved games
Display not logged in error popup
No
Yes
close saved games window
Delete saved game
Display match page with selected game
Input move from current player
Make AI player
Is the move legal?
Make move on board
Update display with move
Is move a capture?
Update player piece counts
Update player piece counts on display
Update display to say the new current player
Switch current player
Is game over?
Display winner in a popup
close popup
Display illegal move popup
Display login form
Input username and password
Username and password entered in database for the same user?
Display incorrect username or password error message
Display successful login message
close popup
Show error message informing the user that they must be logged in to view their stats
Fetch user stats from the database
Display user stats
close popup
Display help page

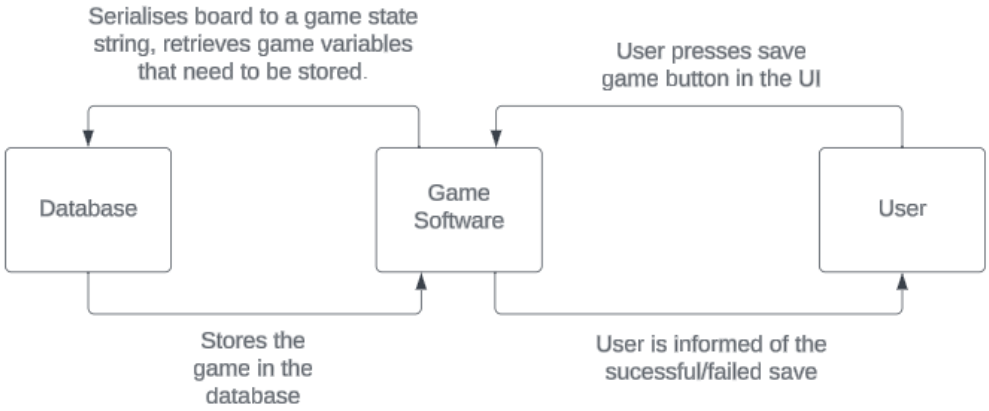
The diagram below shows the remaining flowchart for the flow through the program for changing piece colour and signing up.



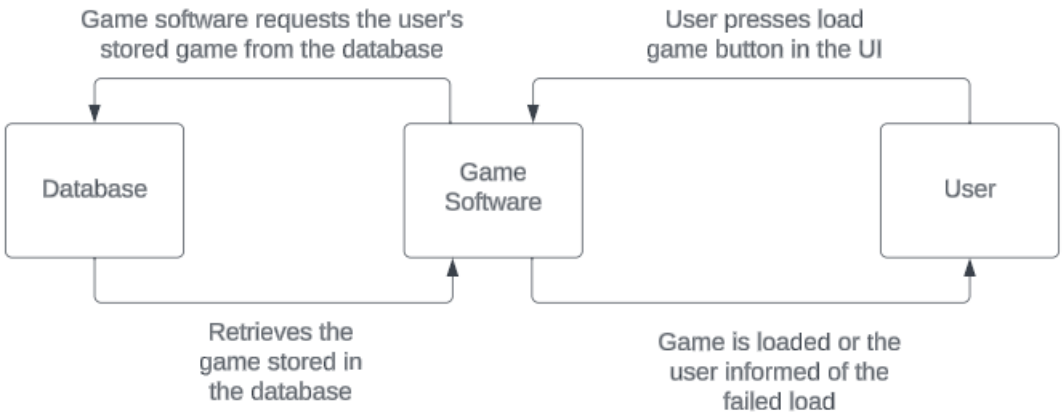
2.2 Data Flow Diagrams

The diagrams below describe the flow of data between the user, game software and database for:

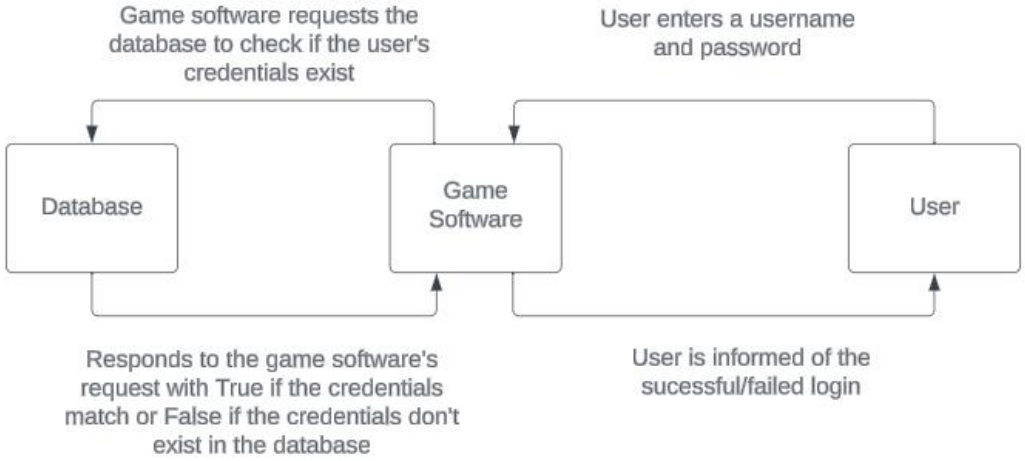
1) Saving a game



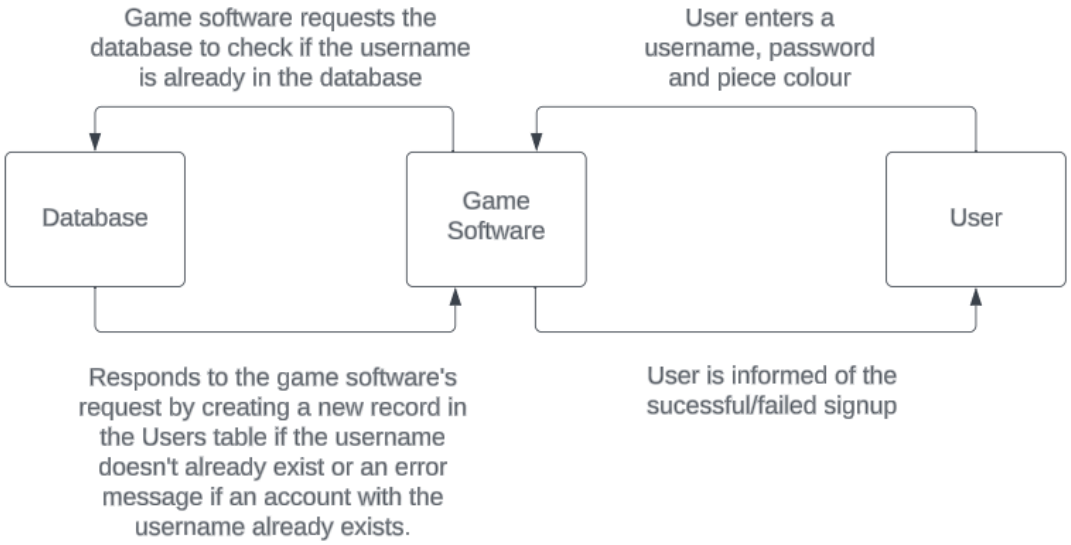
2) Loading a game



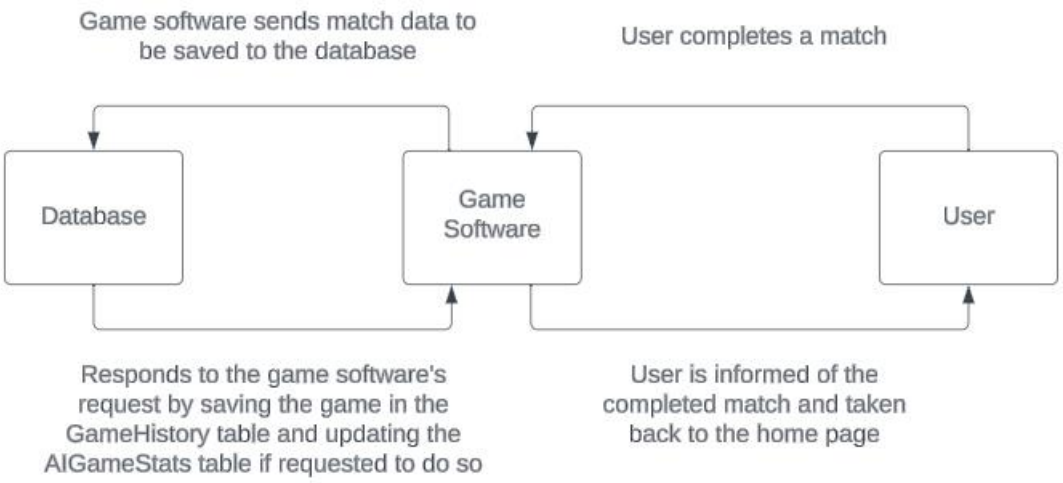
3) Logging in



4) Signing up



5) Adding a completed match to a player's match history and updating the AI game stats



2.3 Data structures

This section describes the key data structures used in this project.

2.3.1 Gameboard

The grid-shaped portion of the game board where pieces can reside will be stored as a 2D array. This is because it allows for easy lookup of piece position using row and column indexes in constant time O(1). A 2D array also makes the board easy to visualise during the programming process and when displaying the board as it matches the tabular structure of the grid-shaped portion of the board.

Below is a diagram illustrating the 2D array used for the game board.

GridLocation()	GridLocation()	GridLocation()	GridLocation()	GridLocation()	GridLocation()
GridLocation()	GridLocation()	GridLocation()	GridLocation()	GridLocation()	GridLocation()
GridLocation()	GridLocation()	GridLocation()	GridLocation()	GridLocation()	GridLocation()
GridLocation()	GridLocation()	GridLocation()	GridLocation()	GridLocation()	GridLocation()
GridLocation()	GridLocation()	GridLocation()	GridLocation()	GridLocation()	GridLocation()
GridLocation()	GridLocation()	GridLocation()	GridLocation()	GridLocation()	GridLocation()

Each position in the 6x6 grid will be represented by a GridLocation object which stores a Piece object, the position's coordinates and a string indicating the looped track(s) the position lies on.

As pieces are moved throughout the game, GridLocations are not moved. Instead, the piece attribute of the GridLocations is changed.

Below is a depiction of the Piece object positions at the start of a game.

Piece(COLOUR_1)	Piece(COLOUR_1)	Piece(COLOUR_1)	Piece(COLOUR_1)	Piece(COLOUR_1)	Piece(COLOUR_1)
Piece(COLOUR_1)	Piece(COLOUR_1)	Piece(COLOUR_1)	Piece(COLOUR_1)	Piece(COLOUR_1)	Piece(COLOUR_1)
None	None	None	None	None	None
None	None	None	None	None	None
Piece(COLOUR_2)	Piece(COLOUR_2)	Piece(COLOUR_2)	Piece(COLOUR_2)	Piece(COLOUR_2)	Piece(COLOUR_2)
Piece(COLOUR_2)	Piece(COLOUR_2)	Piece(COLOUR_2)	Piece(COLOUR_2)	Piece(COLOUR_2)	Piece(COLOUR_2)

When a location does not have a piece, that corresponding GridLocation has its piece attribute set to None.

Near the end of the game, the gameboard's piece structure could resemble the diagram below.

None	None	None	None	None	None
None	None	None	None	None	None
None	None	None	None	None	None
None	Piece(COLOUR_1)	None	None	Piece(COLOUR_2)	None
None	None	None	None	None	None
None	None	None	None	None	None

If player 1 were to capture player 2's last piece, the gameboard would look like this:

None	None	None	None	None	None
None	None	None	None	None	None
None	None	None	None	None	None
None	None	None	None	Piece(COLOUR_1)	None
None	None	None	None	None	None
None	None	None	None	None	None

At which point the game would terminate and player 1 would be the winner.

2.3.2 Looped Tracks

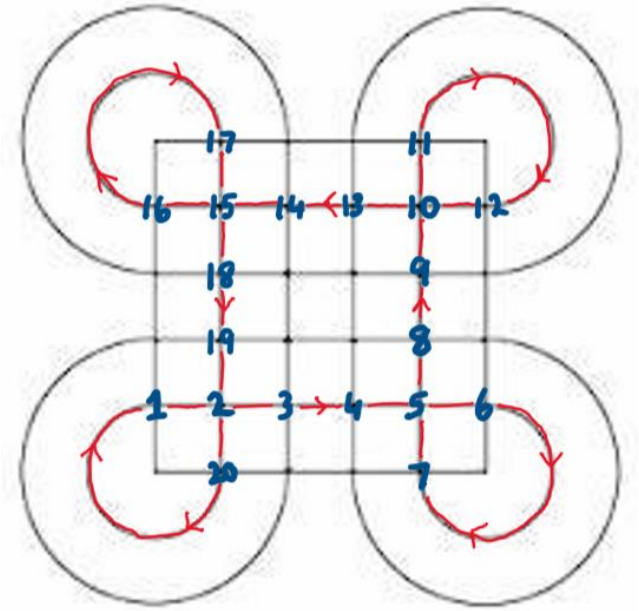
There are two looped tracks on the Surakarta board which will be stored in circular lists. This is because each looped track is continuous so there is no end element. Each GridLocation object that is on a looped track will be stored in its corresponding circular list. Thus, when moves are made, the board (2D array) and both circular lists need to be updated in conjunction with each other.

The lists can be iterated over in either direction. The circular lists have no end element because once an end of the underlying list data structure is reached, pointers wrap around the list.

A right pointer and a left pointer whose values can be set before iteration are used to enable traversal in in either direction. Below is a code snippet illustrating how the two pointers are used to traverse one step in either direction.

```
def get_next_right(self):

    """returns the next item in the circular list, starting from the right pointer."""

    item = self.__lst[self.__right_pointer]
    self.__right_pointer = (self.__right_pointer + 1) % len(self.__lst)
    return item

def get_next_left(self):

    """returns the next item in the circular list, starting from the left pointer."""

    item = self.__lst[self.__left_pointer]
    self.__left_pointer = (self.__left_pointer - 1) % len(self.__lst)
    return item
```

In the circular list, GridLocations objects are stored in sequential order that they are encountered during a full traversal of the looped track. Thus, GridLocations that are encountered more than once during a traversal are repeated in the circular list. To illustrate this a full traversal of the inner looped track is shown below where each GridLocation has been asigned a number label for clarity.



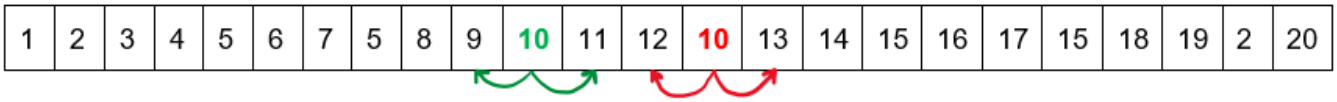
In this example, the GridLocations would be stored in the following order:

1	2	3	4	5	6	7	**5**	8	9	10	11	12	**10**	13	14	15	16	17	**15**	18	19	**2**	20

The numbers bolded in blue represent a repeated GridLocation which has been returned to during a full traversal.

This repeated GridLocation storage enables a looped track to be traversed in every possible direction. For instance, starting from grid location 10, we can traverse the inner looped track by moving in the direction of 11 (up), 13 (left), 9 (down) or 12 (right).

This is explained using the Circular List setup below.



To traverse the inner looped track left from 10, we iterate forwards starting from the red 10 in the diagram. To go right, we iterate backwards from the red 10. To go down, we iterate backwards from the green 10. To go up, we iterate forwards from the green 10.

Therefore, a looped track can be traversed in every way possible from a starting GridLocation by iterating forwards and backwards from every occurrence of that GridLocation in the looped track.

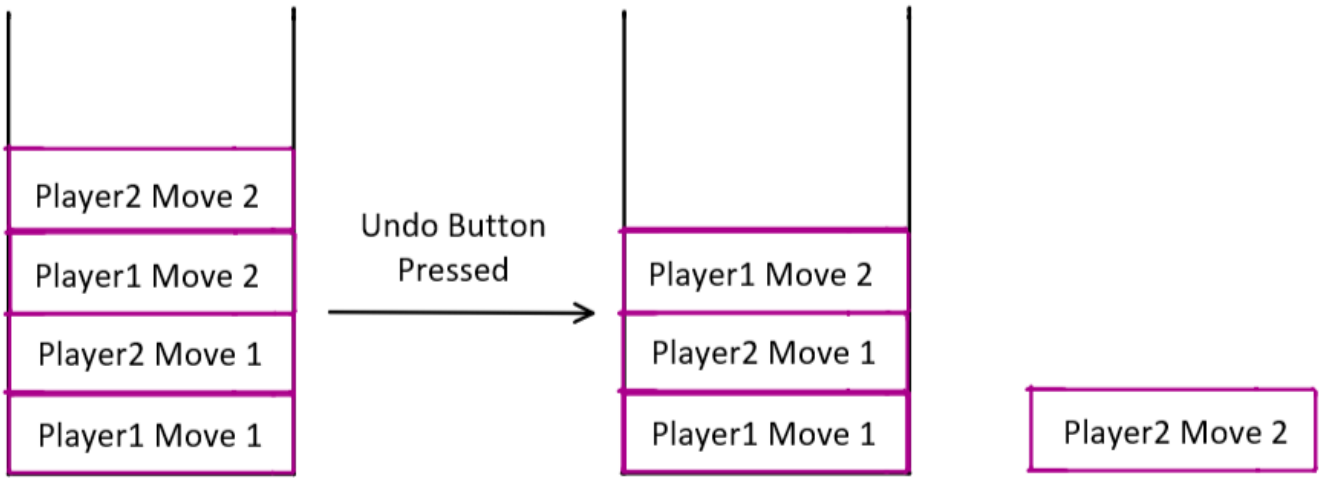
The two tables below depict the inner looped track (which is 1 dimensional but displayed as a 2D array for space reasons) at the start of the game. The first table shows the corresponding coordinates of the piece positions table below it. Note that the looped track actually contains GridLocation objects but for clarity, coordinates and pieces are shown instead. Also note that both 1-dimension arrays below are to be read row by row from left to right with for example the element immediately after (4,3) being (4,4). The arrows on the table show how to read the list.

(4,0)	(4,1)	(4,2)	(4,3)
(4,4)	(4,5)	(5,4)	(4,4)
(3,4)	(2,4)	(1,4)	(0,4)
(1.5)	(1,4)	(1,3)	(1,2)
(1,1)	(1,0)	(0,1)	(1,1)
(2,1)	(3,1)	(4,1)	(5,1)

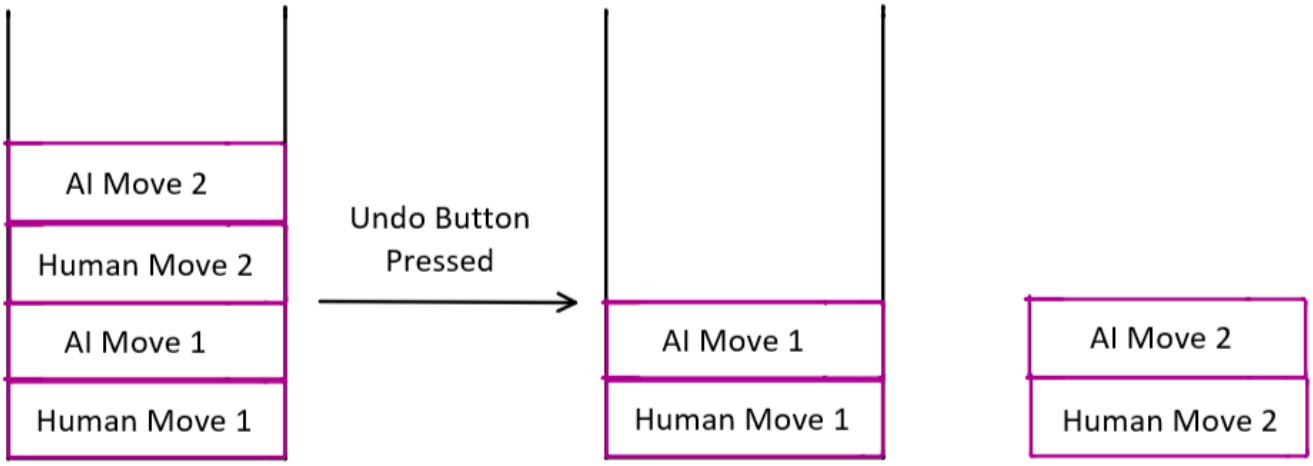
Piece(COLOUR_1)	Piece(COLOUR_1)	Piece(COLOUR_1)	Piece(COLOUR_1)
Piece(COLOUR_1)	Piece(COLOUR_1)	Piece(COLOUR_1)	Piece(COLOUR_1)
None	None	Piece(COLOUR_2)	Piece(COLOUR_2)
Piece(COLOUR_2)	Piece(COLOUR_2)	Piece(COLOUR_2)	Piece(COLOUR_2)
Piece(COLOUR_2)	Piece(COLOUR_2)	Piece(COLOUR_2)	Piece(COLOUR_2)
None	None	Piece(COLOUR_1)	Piece(COLOUR_1)

2.3.3 Undo Button with a Stack

A stack will be used to facilitate the game's undo button. Each time a move is played, a Move object will be pushed onto the top of the stack. When the undo button is pressed during local play, the move at the top of the stack is popped off. This is shown in the diagram below.



If the undo button is pressed during AI play, the top two moves are popped off the stack so that it is the human player's turn after the undo. This is shown below.



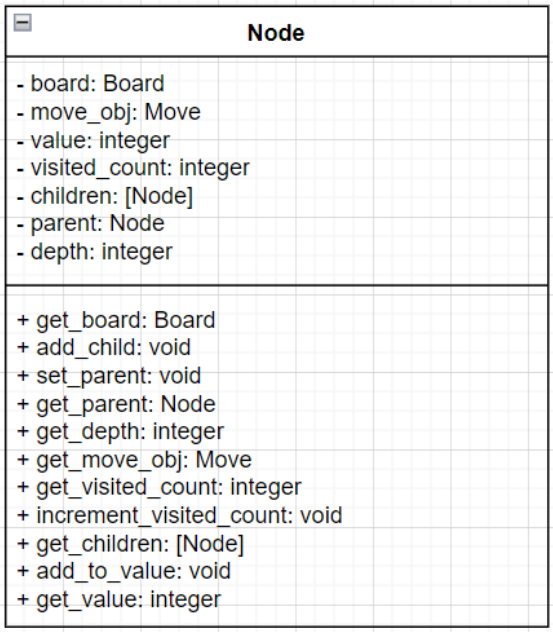
If the user attempts to undo a move when no previous moves have been made, a stack underflow will occur. When this error is caught, the user is informed that no more moves can be undone.

Because Surakarta games can vary greatly in length, the stack is stored dynamically using a list attribute in the Stack class. Thus, there is no need to handle a stack overflow error.

2.3.4 Trees with MCTS

A tree is used to make a game tree in the Monte Carlo Tree Search algorithm used by the medium and hard AI difficulties. Each tree node is made with a dedicated Node class. The GameTree class utilises Node objects to make a game tree and has methods to perform the four stages of MCTS.

The class diagram below shows the key methods and attributes of the Node class.



The data stored by each tree node is a board object. The value and visited_count attributes are updated during backpropagation and are used by the UCB1 formula during the selection stage of MCTS.

For descriptions of the purpose of each of the Node class's attributes and public methods, see section 2.1.2.

2.4 Algorithms

This section describes the key algorithms used in this project, including detailed explanations, pseudocode, and dry runs where necessary.

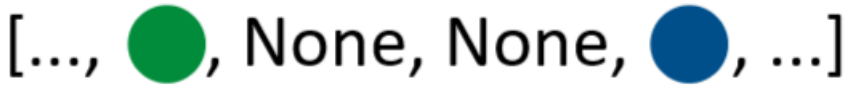
2.4.1 Undoing Moves

When a move is undone, it is first popped off the move history stack.

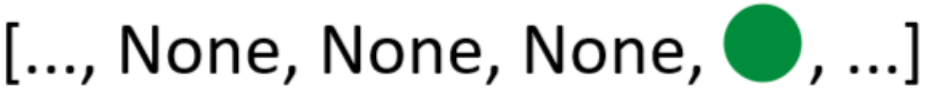
For a non-capturing move, the algorithm for moving a piece described in sections 1.6.4 and 2.4.4 is used to make the opposite move (where the start GridLocation becomes the end GridLocation and vice versa).

The process of undoing a capturing move is more complex so the following example will be used to illustrate it:

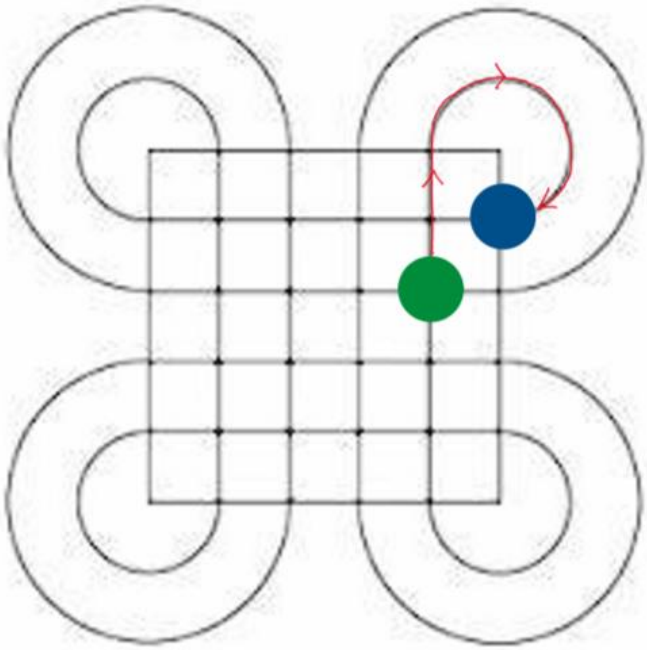
Before Capture:



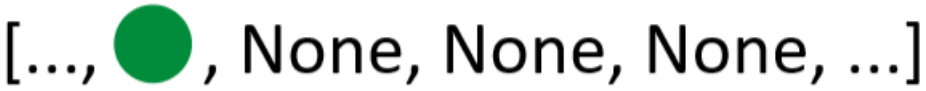
After Capture:



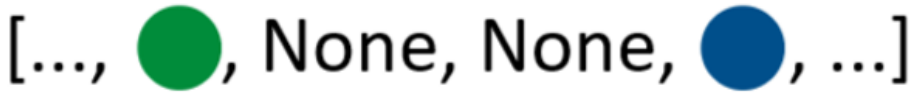
In this example, a sample section of one of the looped tracks is depicted and the green piece has captured the blue piece. Let's also assume that one of the board's four loops is used in this capture between the two pieces so the capture is legal. For example, if the looped track is the inner looped track, it could show the scenario below.



First, we switch all occurrences of the piece at the capture's start GridLocation with that of the captures end GridLocation in the inner looped track. This has the effect of moving the capturing piece back to its original location.



We then replace the capture's end GridLocation with the colour that used to occupy the end location (specified by the historical move popped off the stack). The move has now been correctly undone in the inner looped track which now looks like the diagram below.



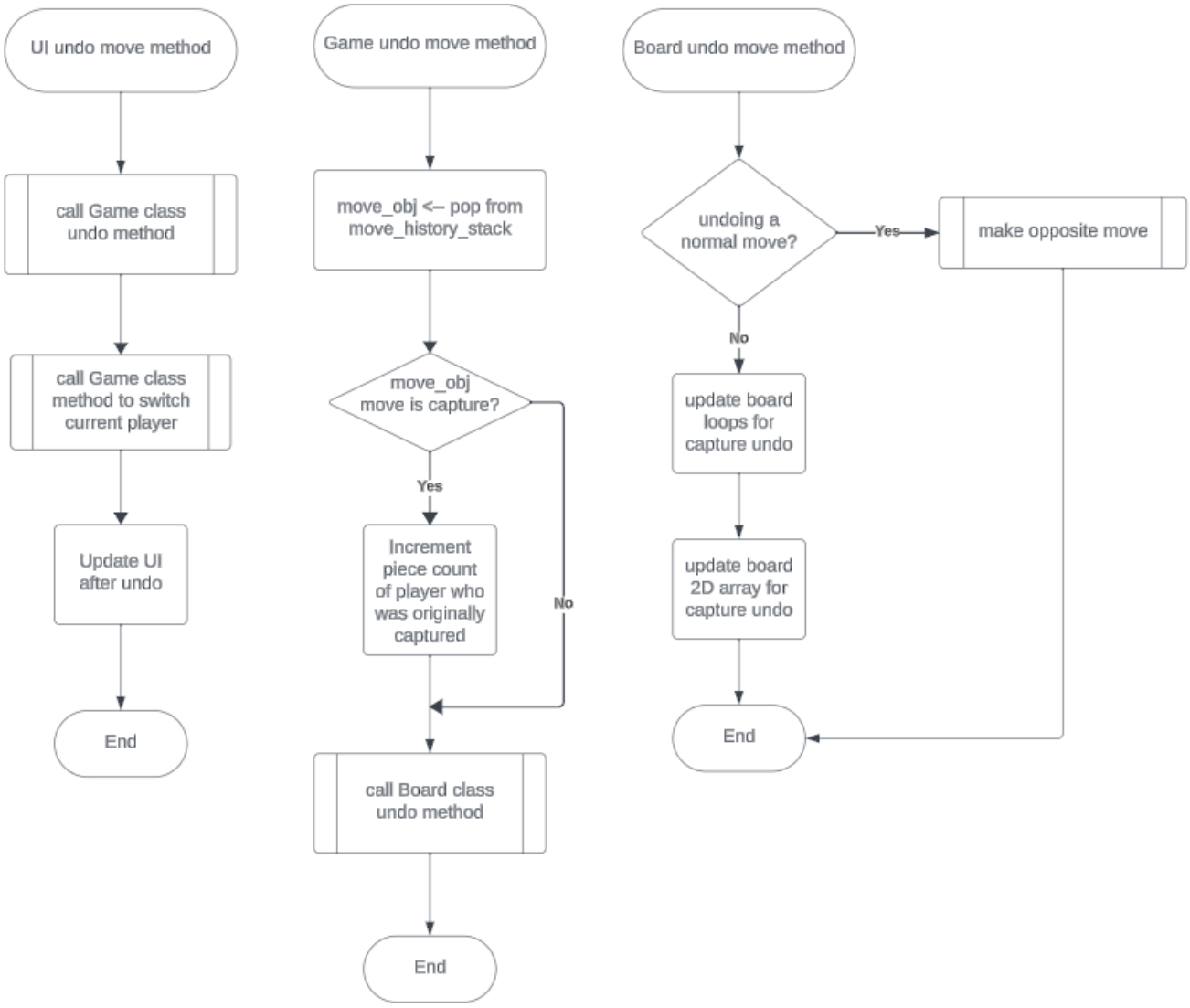
This whole process is then repeated with the outer looped track. Note that while the blue piece's GridLocation is not on the outer looped track, the same process still works because during the initial switch, we will simply replace the green piece with no piece.

To undo the capture in the board's 2D array, we replace the pieces at the capture's start and end location with their historical pieces specified by the move popped off the stack.

If the game is being played in AI mode, the whole undoing process is repeated one more time.

The undoing process takes place across the UI, Game and Board classes. The UI class calls a method in Game to undo the move and after this completes, it calls another Game method to switch the current player and then it updates the UI with the new undone state. The Game class contains the move_history_stack and pops the most recent move off of the stack. The Game class then calls a method in the Board class to undo the move and after this completes, if the undone move was a capture, it adds a piece to the total piece count of the correct player. The Board class makes the move it receives from the Game class in reverse as described in the example above and updates both of the board's looped tracks and the main 2D array. This division of responsibilities makes the code easier to understand and maintain.

The flowchart below summarises this division of responsibilities between the UI, Game and Board classes when undoing a move:



The process of undoing a move is summarised in the pseudocode below:

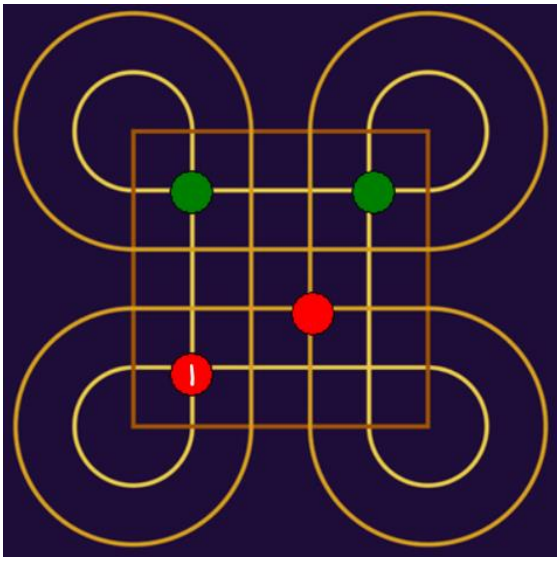
```
IF move_history_stack is empty THEN
    return None

move_obj = POP from move_history_stack


IF move_type is "capture" THEN

    INCREMENT piece count of player who was captured

    // update looped tracks:
    FOR each looped_track DO
        switch pieces currently at move_obj start and end locations
        update current move_obj end location piece to be the end_location piece before the capture

    // update main board 2D array:
    spawn the start_location piece before the capture at start_location
    spawn the end_location piece before the capture at end_location


ELSE IF move_type is "move" THEN
    board.move_piece(opposite_move)


switch current player
update UI
```

The rest of this section will be a dry run of the undoing algorithm. Two dry runs are included: one for undoing a normal adjacent move and one for undoing a capturing move.

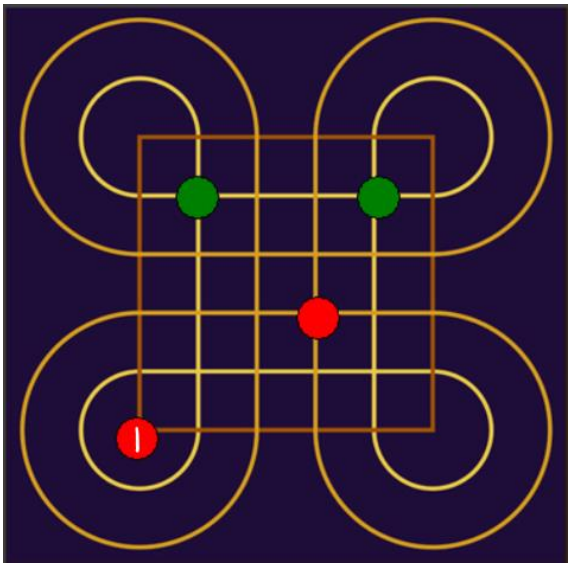
Note that all of the dry runs in this document that involve the use of the inner or outer looped tracks use a LoopedTrack method get_lst_dryruns() which returns all of the contents of the LoopedTrack object as a list. This method was only used as part of the dry runs and was deleted from the code afterwards.

For the first dry run, the initial board state is as follows. Note that in this test case, the game has been loaded from the database to this state, so the move history stack is empty at this point.

The red piece labelled “1” is initially in the position below at coordinates (4,1).



It is then moved to the location with coordinates (5,0) shown below.



The following dry run will demonstrate how this move is undone so that the red piece labelled “1” is back in its initial state.

After the move is made, the game history stack contains one Move object.

```
self._Game__move_history_stack: <Stack.Stack object at 0x0000014D9A76B2E0>
  special variables
  function variables
  _Stack__stack: [<Move.Move object at...D9A769D50>]
```

At the start of the undoing algorithm, the Move object is popped off the move history stack.

```
self._Game__move_history_stack: <Stack.Stack object at 0x0000014D9A76B2E0>
  special variables
  function variables
  _Stack__stack: []
move_obj: <Move.Move object at 0x0000014D9A769D50>
```

The image below shows the current state of the inner track after the move was made where each GridLocation object is displayed as a tuple of its piece colour and coordinates.

```
[(loc.get_piece_colour(), loc.get_cords()) for loc in self._Board__inner_track.get_lst_dryruns()]:
  special variables
  function variables
  00: (None, (4, 0))
  01: (None, (4, 1))
  02: (None, (4, 2))
  03: (None, (4, 3))
  04: (None, (4, 4))
  05: (None, (4, 5))
  06: (None, (5, 4))
  07: (None, (4, 4))
  08: (None, (3, 4))
  09: (None, (2, 4))
  10: ('green', (1, 4))
  11: (None, (0, 4))
  12: (None, (1, 5))
  13: ('green', (1, 4))
  14: (None, (1, 3))
  15: (None, (1, 2))
  16: ('green', (1, 1))
  17: (None, (1, 0))
  18: (None, (0, 1))
  19: ('green', (1, 1))
  20: (None, (2, 1))
  21: (None, (3, 1))
  22: (None, (4, 1))
  23: (None, (5, 1))
  len(): 24
```

The move is undone in the inner looped track so that it is of the same state as it was before the move was made. Note that the outer track would also be updated but in this case, the start and end locations of the piece being moved are both only on the inner track, so the outer track won't change.

The image below shows the state of the inner looped track after the move is undone in the inner looped track.

```
[(loc.get_piece_colour(), loc.get_cords()) for loc in self._Board__inner_track.get_lst_dryruns()]:
  special variables
  function variables
  00: (None, (4, 0))
  01: ('red', (4, 1))
  02: (None, (4, 2))
  03: (None, (4, 3))
  04: (None, (4, 4))
  05: (None, (4, 5))
  06: (None, (5, 4))
  07: (None, (4, 4))
  08: (None, (3, 4))
  09: (None, (2, 4))
  10: ('green', (1, 4))
  11: (None, (0, 4))
  12: (None, (1, 5))
  13: ('green', (1, 4))
  14: (None, (1, 3))
  15: (None, (1, 2))
  16: ('green', (1, 1))
  17: (None, (1, 0))
  18: (None, (0, 1))
  19: ('green', (1, 1))
  20: (None, (2, 1))
  21: (None, (3, 1))
  22: ('red', (4, 1))
  23: (None, (5, 1))
  len(): 24
```

The move also needs to be undone in the board 2D array. The state of the board 2D array after the move is made is shown below. The GridLocation objects in the board are represented by the colour of the piece at the location.

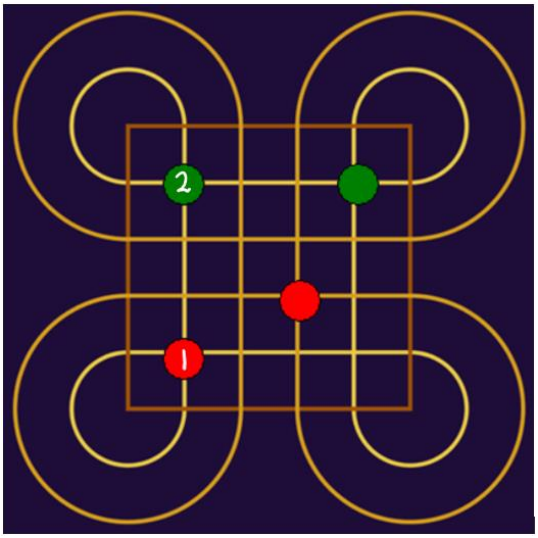
```
[[loc.get_piece_colour() for loc in row] for row in self._Board__board]: [[None, None, None, None, None,…
  special variables
  function variables
  0: [None, None, None, None, None, None]
  1: [None, 'green', None, None, 'green', None]
  2: [None, None, None, None, None, None]
  3: [None, None, None, 'red', None, None]
  4: [None, None, None, None, None, None]
  5: ['red', None, None, None, None, None]
  len(): 6
```

To undo the move in the board 2D array, the move is made in reverse on the board 2D array.

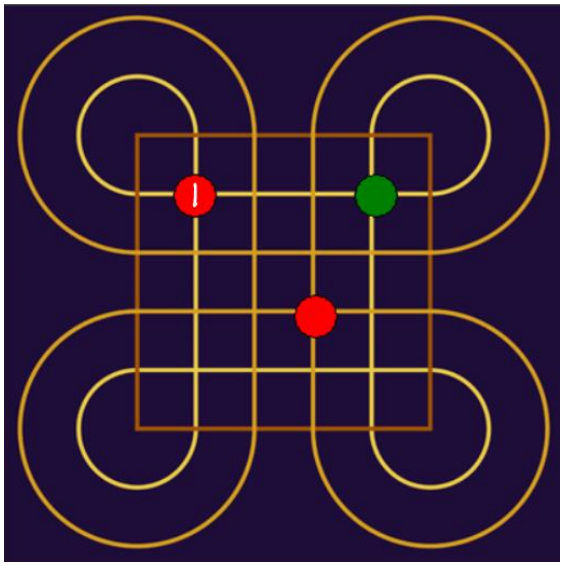
```
[[loc.get_piece_colour() for loc in row] for row in self._Board__board]: [[None, None, None, None, None,…
 special variables
 function variables
 0: [None, None, None, None, None, None]
 1: [None, 'green', None, None, 'green', None]
 2: [None, None, None, None, None, None]
 3: [None, None, None, 'red', None, None]
 4: [None, 'red', None, None, None, None]
 5: [None, None, None, None, None, None]
   len(): 6
```

The move has now been fully undone.

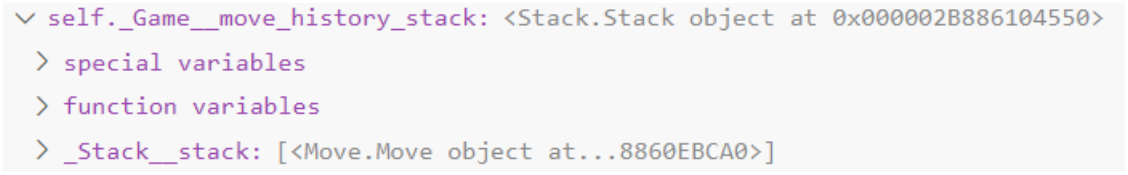
In this next dry run, the red piece labelled “1” at coordinates (4,1) is used to capture the green piece labelled “2” at coordinates (1,1). The initial state of the board is shown below.



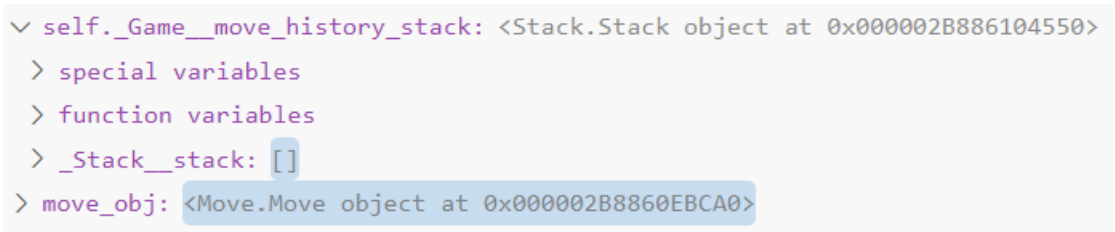
The state of the board after the capture is shown below.



As was with the first dry run, the move history stack initially contains only one move (the move we want to undo)

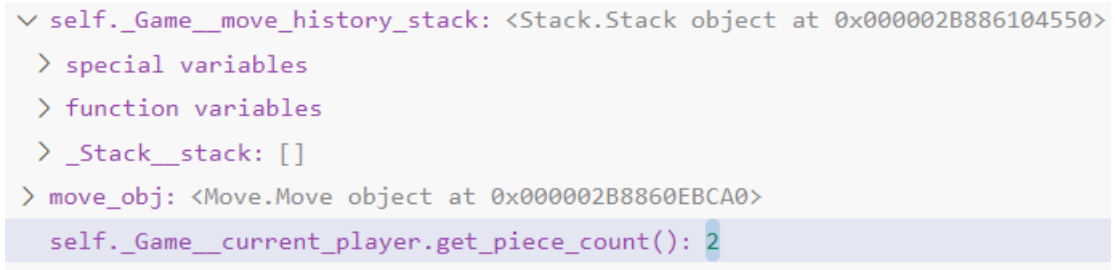


This move is popped off the move history stack.



The piece count of the current player after the capturing move in this case will be 1 as shown below.

When the move is being undone, this piece count is incremented to 2 to reflect that a piece has returned to the board.



Below is the state of the inner track after the capture has been made. As with the first dry run, we will only be considering the inner track as the start and end locations of the capture are both only on the inner track, so the outer track remains unchanged after the capture and the undo.

```
[(loc.get_piece_colour(), loc.get_cords()) for loc in self._Board__inner_track.get_lst_dryruns()]:
 special variables
 function variables
 00: (None, (4, 0))
 01: (None, (4, 1))
 02: (None, (4, 2))
 03: (None, (4, 3))
 04: (None, (4, 4))
 05: (None, (4, 5))
 06: (None, (5, 4))
 07: (None, (4, 4))
 08: (None, (3, 4))
 09: (None, (2, 4))
 10: ('green', (1, 4))
 11: (None, (0, 4))
 12: (None, (1, 5))
 13: ('green', (1, 4))
 14: (None, (1, 3))
 15: (None, (1, 2))
 16: ('red', (1, 1))
 17: (None, (1, 0))
 18: (None, (0, 1))
 19: ('red', (1, 1))
 20: (None, (2, 1))
 21: (None, (3, 1))
 22: (None, (4, 1))
 23: (None, (5, 1))
  len(): 24
```

When updating the inner track in the undo operation, the pieces at all occurrences of the capture's start location are switched with all occurrences of the pieces at the capture's end location. This has the effect of putting the red piece labelled "1" in the diagram back into its initial position in the inner track. This is shown below.

```
[(loc.get_piece_colour(), loc.get_cords()) for loc in self._Board__inner_track.get_lst_dryruns()]:
 special variables
 function variables
 00: (None, (4, 0))
 01: ('red', (4, 1))
 02: (None, (4, 2))
 03: (None, (4, 3))
 04: (None, (4, 4))
 05: (None, (4, 5))
 06: (None, (5, 4))
 07: (None, (4, 4))
 08: (None, (3, 4))
 09: (None, (2, 4))
 10: ('green', (1, 4))
 11: (None, (0, 4))
 12: (None, (1, 5))
 13: ('green', (1, 4))
 14: (None, (1, 3))
 15: (None, (1, 2))
 16: (None, (1, 1))
 17: (None, (1, 0))
 18: (None, (0, 1))
 19: (None, (1, 1))
 20: (None, (2, 1))
 21: (None, (3, 1))
 22: ('red', (4, 1))
 23: (None, (5, 1))
  len(): 24
```

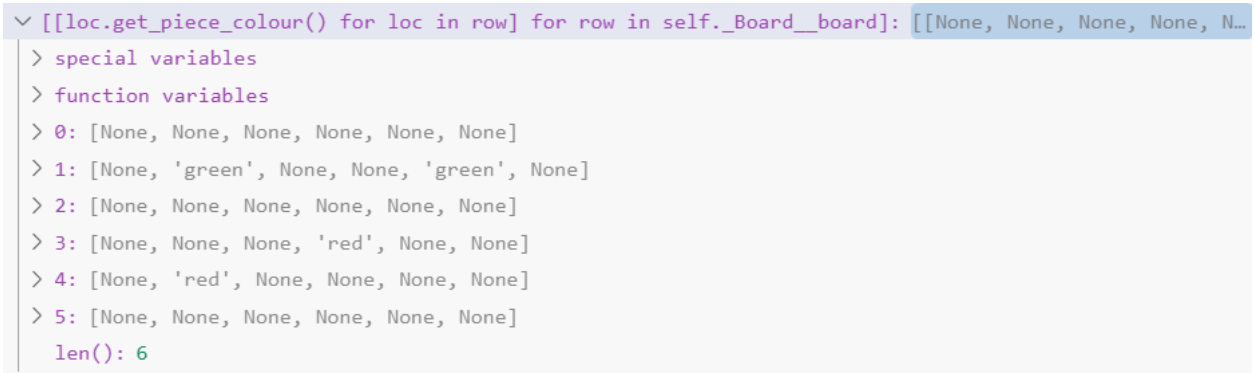
Then, we add the piece that was captured back to its initial GridLocation objects in the inner track as shown below. The capturing move has now been correctly undone in the inner track.

```
[(loc.get_piece_colour(), loc.get_cords()) for loc in self._Board__inner_track.get_lst_dryruns()]:
  special variables
  function variables
  00: (None, (4, 0))
  01: ('red', (4, 1))
  02: (None, (4, 2))
  03: (None, (4, 3))
  04: (None, (4, 4))
  05: (None, (4, 5))
  06: (None, (5, 4))
  07: (None, (4, 4))
  08: (None, (3, 4))
  09: (None, (2, 4))
  10: ('green', (1, 4))
  11: (None, (0, 4))
  12: (None, (1, 5))
  13: ('green', (1, 4))
  14: (None, (1, 3))
  15: (None, (1, 2))
  16: ('green', (1, 1))
  17: (None, (1, 0))
  18: (None, (0, 1))
  19: ('green', (1, 1))
  20: (None, (2, 1))
  21: (None, (3, 1))
  22: ('red', (4, 1))
  23: (None, (5, 1))
  len(): 24
```

The capturing move now needs to be updated in the board's 2D array. The state of the board's 2D array after the capture is made is shown below. GridLocation objects are represented by their piece colour.

```
[[loc.get_piece_colour() for loc in row] for row in self._Board__board]: [[None, None, None, None, N…
  special variables
  function variables
  0: [None, None, None, None, None, None]
  1: [None, 'red', None, None, 'green', None]
  2: [None, None, None, None, None, None]
  3: [None, None, None, 'red', None, None]
  4: [None, None, None, None, None, None]
  5: [None, None, None, None, None, None]
  len(): 6
```

To undo the move in the board's 2D array, we restore the pieces that were present at the start and end locations prior to the capture as specified by the move object popped of the move history stack.



The capturing move has now been fully undone.

2.4.2 Checking if a Normal Move is Legal

To check if a normal move is legal, only the board's 2D array data structure needs to be used. A legal position that a piece can move to is one that is adjacent to the piece's current position and unoccupied. In Surakarta, adjacency includes diagonal moves so a piece can have at most 8 locations it can move to. The board snippet below illustrates the adjacent locations to the location at coordinates (2,3).

(1,2)	(1,3)	(1,4)
(2,2)	(2,3)	(2,4)
(3,2)	(3,3)	(3,4)

First, we do the following early stop check to ensure the player is not attempting to move a non-existent piece (a colour of None) or an opponent's piece.

```
IF piece colour at start location is not the current player's colour THEN
    return False
```

To then determine if an attempted move is to an adjacent position, we can use the pseudocode below.

```
x_diff = ABSOLUTE_VALUE(start_cord_x - end_cord_x)
y_diff = ABSOLUTE_VALUE(start_cord_y - end_cord_y)

IF (x_diff <= 1) AND (y_diff <= 1) THEN
    return TRUE

ELSE
    return FALSE
```

The algorithm must also check to ensure that the attempted move's end GridLocation is unoccupied. The overall algorithm is described in the pseudocode below.

```
IF (is_adjacent(start_loc, end_loc)) AND (end_loc.is_emtpy()) THEN
    return TRUE

ELSE
    return FALSE
```

2.4.3 Checking if a Capture is Legal

Checking the legality of an attempted capture requires both of the board's looped tracks (circular lists) and the board's 2D array.

First, we do some early stop checks to ensure that the attempted capture is not using an opponent's piece, capturing your own piece or involving any empty locations with no piece. This is summarised in the pseudocode below.

```
IF piece colour at start location is not the current player's colour THEN
    return False

ELSE IF piece colour at start location is the same as the piece colour at end location THEN
    return False

ELSE IF either of the start or end locations are empty THEN
    return False
```

The main abstracted process of capture legality checking is described in the pseudocode below.

```
FOR each looped_track common to start_loc and end_loc DO
    starting_indexes <-- get indexes of start_loc in looped_track

    FOR start_loc_index in starting_indexes DO

        right_move = search_direction_for_capture(start_loc, start_loc_index, looped_track, "right")

        IF right_move and right_move.get_end_cords() = end_loc.get_cords() THEN
            return True

        left_move = search_direction_for_capture(start_loc, start_loc_index, looped_track, "left")

        IF left_move and left_move.get_end_cords() = end_loc.get_cords() THEN
            return True

return False
```

In this pseudocode, start_loc and end_loc refer to the start and end GridLocation objects of the attempted capture.

As described in section 2.3.2, we can traverse a looped track from a given position in every possible direction by iterating forwards and backwards from each occurrence of the starting location in a looped track. We traverse the looped tracks that both the start_loc and end_loc are a part of. We call the search_direction_for_capture() function which returns a Move object if a valid capture can be made in the direction specified. Otherwise, it returns False. Then if a discovered capture matches the capture attempted by the user, the attempted capturing move is legal.

The pseudocode below illustrates the search_direction_for_capture() function:

```
invalid = FALSE
loop_count = 0
prev_loc = start_loc

IF direction == "right" THEN
    looped_track.set_pointer(start_loc_index, "right")
    looped_track.get_next_right()

ELSE IF direction == "left" THEN
    looped_track.set_pointer(start_loc_index, "left")
    looped_track.get_next_left()

WHILE NOT invalid DO

    IF direction == "left" THEN
        curr_loc = looped_track.get_next_left()

    ELSE IF direction == "right" THEN
        curr_loc = looped_track.get_next_right()

    IF loop_used(prev_loc, curr_loc) THEN
        INCREMENT loop_count

    IF is_valid_capture(start_location, curr_loc, loop_count) THEN
        return Move(start_location, curr_loc, "capture")

    IF check_direction_invalid(start_location, curr_loc, loop_count) THEN
        invalid = TRUE

    prev_loc = curr_loc


return FALSE
```

In this pseudocode, curr_loc is the current location being inspected during iteration and prev_loc is the location that was iterated over in the last pass of the while loop.

We set the corresponding pointer to the starting index of iteration and skip over the first location in this direction as it is the same as start_loc.

At each iteration of the while loop, we move one step through the looped_track in the specified direction. If a board loop has been used in this step, we increment the loop_count variable. If a valid capture has been found from the start location to the curr_loc, we return a Move object describing this capture. We continue iteration until a legal capture can no longer be made in the current direction. This can be seen through the looped track example below.

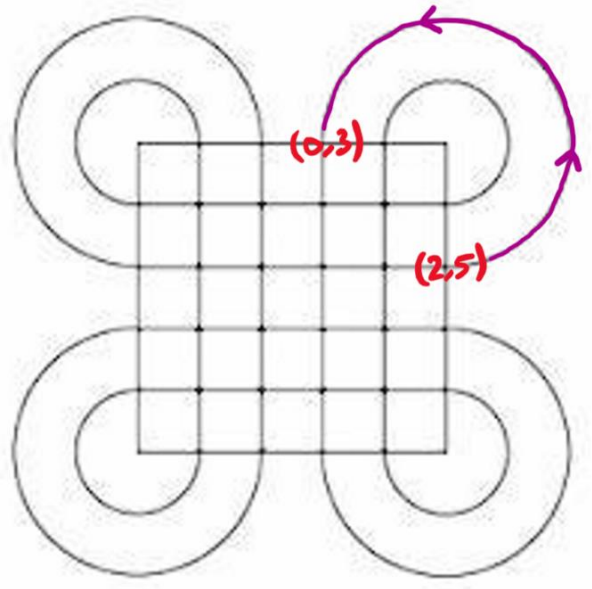
In this example, we are checking to see if the blue piece labelled “1” can capture the green piece labelled “3” by going left.

When curr_loc reaches the highlighted None as shown below, we have not encountered any blocking pieces, so it is still possible for the capture to be made in the left direction.

However, once curr_loc reaches the blue piece labelled “2”, the path has been blocked and so no further iteration left needs to be done as we know the desired capture cannot be made in this direction.

To determine if a looped track has been used between two locations, we can simply check to see if both the x-coordinate and y-coordinate have changed between the two locations as this is only possible between two adjacent locations on a looped track if one of the four board loops has been used.

For example, in the diagram below we know that a board loop has been used if in one step through the outer looped track we move from (2,5) to (0,3) because both the x and y coordinates have changed in only one step through the track.



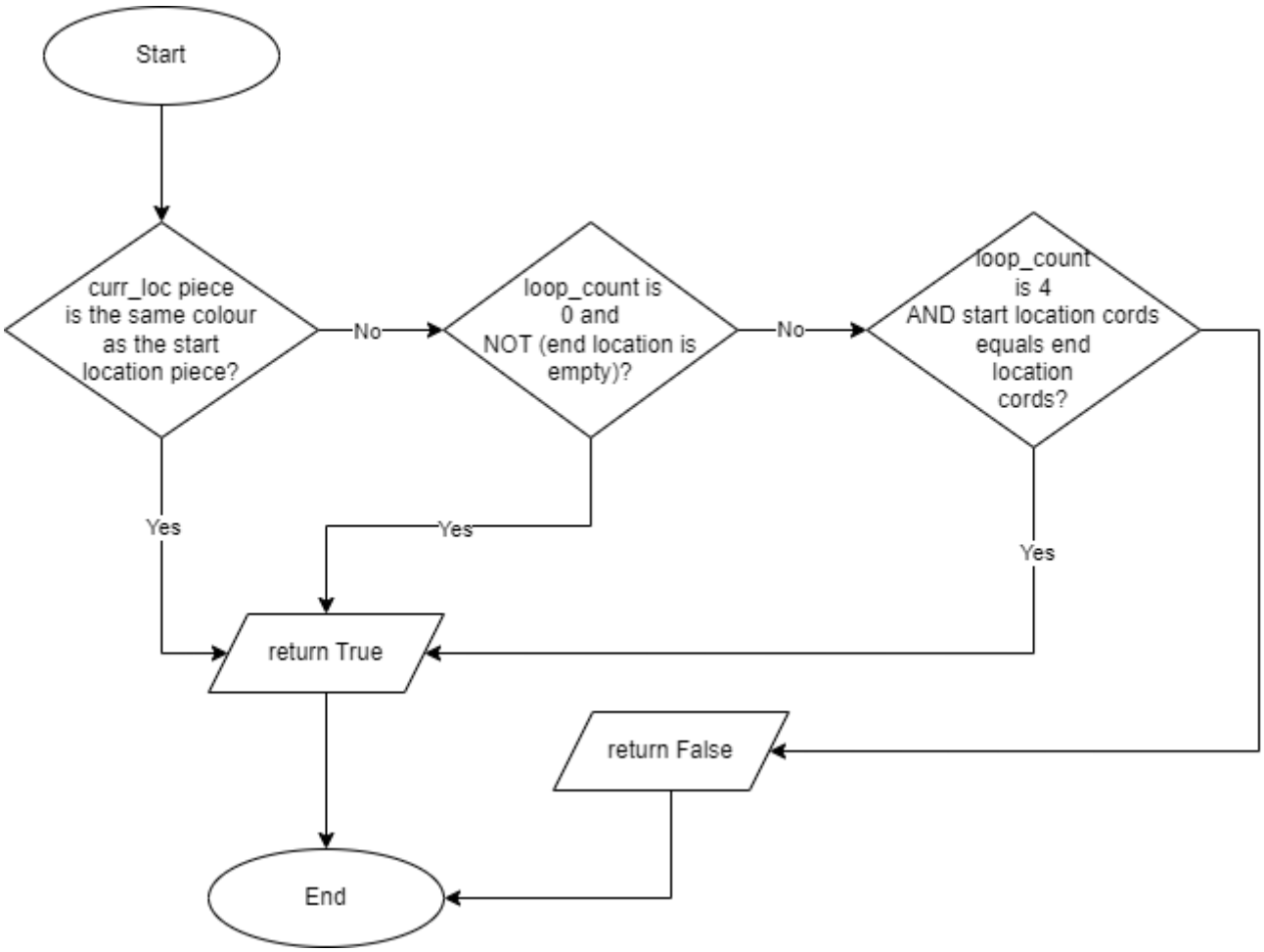
The pseudocode below shows the operation of the is_valid_capture() function.

```
IF end_location.is_empty() THEN
    return False

IF (end_location.get_colour() is not start_location.get_colour()) AND (loop_count > 0) THEN
    return True

return False
```

For a capture to be valid, at least one board loop must have been used and the end location of the capture must be an opposing piece.

But as established in the visualised example above, just because a capture is invalid does not necessarily mean we stop checking the current direction for captures. Only if the check_direction_invalid function evaluates to true do we stop checking the current direction for captures.

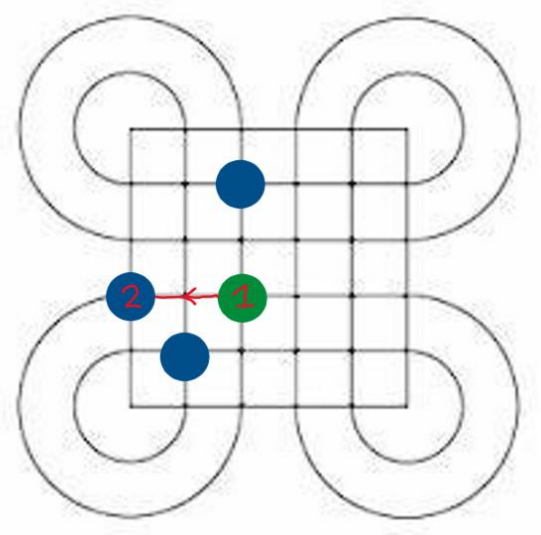
The flowchart below describes the check_direction_invalid function:



The piece at curr_loc being the same colour as the piece at the start location means the capture in the search direction has been blocked by one of your own pieces.

If loop_count is 0 and the end location is occupied, the direction is invalid because the direction has been blocked before one of the four board loops can be traversed. An example of this is shown on the board below where the blue piece labelled “2” has blocked the green piece labelled “1” from making a capture in the left direction.

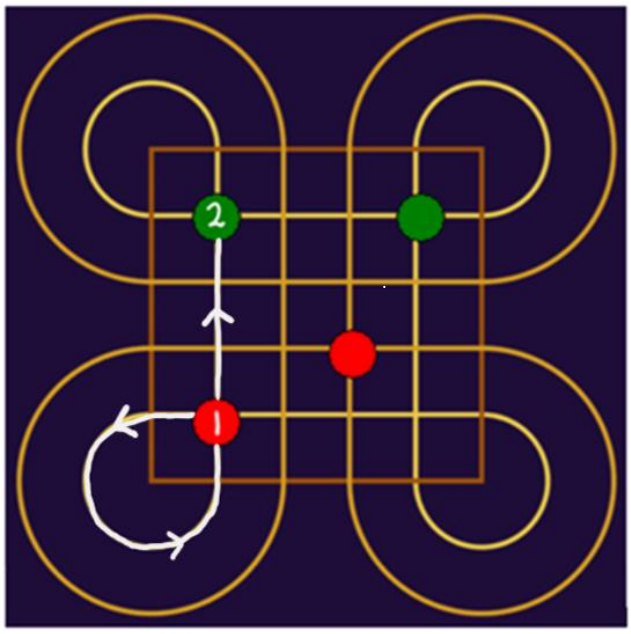
Invalid capture:



If the loop_count is four and the start_location and end_location coordinates are the same, the whole looped track has been traversed with no captures found and so the function returns True.

The rest of this section will be a dry run of this algorithm. The board shown below will be used for this dry run where the red piece labelled “1” will attempt to use the white path shown to capture the green piece labelled “2”. In this case red is the current player’s piece.

The rest of this section will be a dry run to check the legality of the red piece labelled “1” at coordinates (4,1) capturing the green piece labelled “2” at coordinates (1,1) in the board below.



We first obtain a list of the looped tracks that both pieces are a part of. In this case, it is only the inner track as shown below.

```
> [(track, track.get_name()) for track in looped_track_tuple]: [(<LoopedTrack.LoopedT...4B98C6FE0>, 'INNER')]
```

We then iterate through each looped_track in the list of common looped tracks. In this case, there is only one looped track in this list.

```
> [(track, track.get_name()) for track in looped_track_tuple]: [(<LoopedTrack.LoopedT...4B98C6FE0>, 'INNER')]
v looped_track: <LoopedTrack.LoopedTrack object at 0x00000194B98C6FE0>
  > special variables
  > function variables
  > _LoopedTrack__get_all_occurence_indexes: <bound method LoopedTrack.__get_all_occurence_indexes of <LoopedTrack…
    _LoopedTrack__left_pointer: 0
    _LoopedTrack__length: 24
  > _LoopedTrack__lst: [<GridLocation.GridLo...4B98C7010>, <GridLocation.GridLo...4B98C70D0>, <GridLocation.GridLo…
    _LoopedTrack__name: 'INNER'
    _LoopedTrack__right_pointer: 0
```

The state of the looped_track is shown below where each GridLocation object is represented by a tuple of its piece colour and coordinates.

```
[(loc.get_piece_colour(), loc.get_cords()) for loc in looped_track.get_lst_dryruns()]: [(None, (...)), ('red…
> special variables
> function variables
> 00: (None, (4, 0))
> 01: ('red', (4, 1))
> 02: (None, (4, 2))
> 03: (None, (4, 3))
> 04: (None, (4, 4))
> 05: (None, (4, 5))
> 06: (None, (5, 4))
> 07: (None, (4, 4))
> 08: (None, (3, 4))
> 09: (None, (2, 4))
> 10: ('green', (1, 4))
> 11: (None, (0, 4))
> 12: (None, (1, 5))
> 13: ('green', (1, 4))
> 14: (None, (1, 3))
> 15: (None, (1, 2))
> 16: ('green', (1, 1))
> 17: (None, (1, 0))
> 18: (None, (0, 1))
> 19: ('green', (1, 1))
> 20: (None, (2, 1))
> 21: (None, (3, 1))
> 22: ('red', (4, 1))
> 23: (None, (5, 1))
  len(): 24
```

Next, we obtain the indexes where the starting GridLocation object occurs in the looped_track which in this case are 1 and 22. There are two in this case because the start location is a point where the looped_track intersects with itself.

```
> [(track, track.get_name()) for track in looped_track_tuple]: [(<LoopedTrack.LoopedT...4B98C6FE0>, 'INNER')]
> looped_track: <LoopedTrack.LoopedTrack object at 0x00000194B98C6FE0>
> starting_indexes: [1, 22]
```

We then iterate through each index in starting_indexes.

```
> [(track, track.get_name()) for track in looped_track_tuple]: [(<LoopedTrack.LoopedT...4B98C6FE0>, 'INNER')]
> looped_track: <LoopedTrack.LoopedTrack object at 0x00000194B98C6FE0>
> starting_indexes: [1, 22]
  ind: 1
> [(loc.get_piece_colour(), loc.get_cords()) for loc in looped_track.get_lst_dryruns()]: [(None, (...)), ('red', (…
```

First, we search for a possible capture iterating right through looped_track starting at index 1. The invalid, loop_count and prev_loc variables have their initial values set before the iteration. Note that the initial value of prev_loc is the starting location of the attempted capture.

```
  invalid: False
  loop_count: 0
> (prev_loc.get_piece_colour(), prev_loc.get_cords()): ('red', (4, 1))
```

The initial state of the looped_track pointers is shown below.

```
looped_track: <LoopedTrack.LoopedTrack object at 0x00000194B98C6FE0>
 > special variables
 > function variables
 > _LoopedTrack__get_all_occurence_indexes: <bound method LoopedTrack.__get_all_occurence_indexes of <LoopedTrack.…
   _LoopedTrack__left_pointer: 0
   _LoopedTrack__length: 24
 > _LoopedTrack__lst: [<GridLocation.GridLo...4B98C7010>, <GridLocation.GridLo...4B98C70D0>, <GridLocation.GridLo.…
   _LoopedTrack__name: 'INNER'
   _LoopedTrack__right_pointer: 0
  ind: 1
> [(loc.get_piece_colour(), loc.get_cords()) for loc in looped_track.get_lst_dryruns()]: [(None, (...)), ('red', (…
```

The right_pointer attribute of looped_track is set to 1.

```
looped_track: <LoopedTrack.LoopedTrack object at 0x000002A84E55B640>
 > special variables
 > function variables
 > _LoopedTrack__get_all_occurence_indexes: <bound method LoopedTrack.__get_al…
   _LoopedTrack__left_pointer: 0
   _LoopedTrack__length: 24
 > _LoopedTrack__lst: [<GridLocation.GridLo...84E55B670>, <GridLocation.GridLo…
   _LoopedTrack__name: 'INNER'
   _LoopedTrack__right_pointer: 1
```

We then iterate over each GridLocation in looped_track (stored in curr_loc) and stop iteration if a legal capture is found or if a legal capture can no longer be found in this direction. These two stop conditions are checked for in each iteration. The first value of curr_loc is shown below.

```
  invalid: False
  loop_count: 0
> (prev_loc.get_piece_colour(), prev_loc.get_cords()): ('red', (4, 1))
> looped_track: <LoopedTrack.LoopedTrack object at 0x000002A84E55B640>
> (curr_loc.get_piece_colour(), curr_loc.get_cords()): (None, (4, 2))
```

Eventually, we reach the stage of iteration shown below where curr_loc is the GridLocation with coordinates (4,5). We will use this iteration to dry run checking if the capture from the start location to curr_loc is legal.

```
> looped_track: <LoopedTrack.LoopedTrack object at 0x00000194B98C6FE0>
> [(loc.get_piece_colour(), loc.get_cords()) for loc in looped_track.get_lst_dryruns()]: [(None, (.…
  invalid: False
> (prev_loc.get_piece_colour(), prev_loc.get_cords()): (None, (4, 4))
> (curr_loc.get_piece_colour(), curr_loc.get_cords()): (None, (4, 5))
  direction: 'right'
  loop_count: 0
```

At this point, curr_loc (shown as end_location in the image below) is empty so this capture is not legal. loop_count is also not greater than 0 as no loop has been used during iteration yet so this condition is also failed.

```
loop_count: 0
end_location.is_empty(): True
end_location.get_piece_colour(): None
start_location.get_piece_colour(): 'red'
loop_count > 0: False
```

We now check if the current iteration indicates that a capture cannot be found in the current direction. Since none of the three conditions below (which indicate no capture can be found) are True, the direction still has potential to yield a valid capture.

```
loop_count == 0 and not end_location.is_empty(): False
(loop_count == self.NUM_BOARD_LOOPS) and (start_location.get_cords() == end_location.get_cords()): False
(start_location.get_piece_colour() == end_location.get_piece_colour()) and (start_location.get_cords() != end_location.get_cords()): False
```

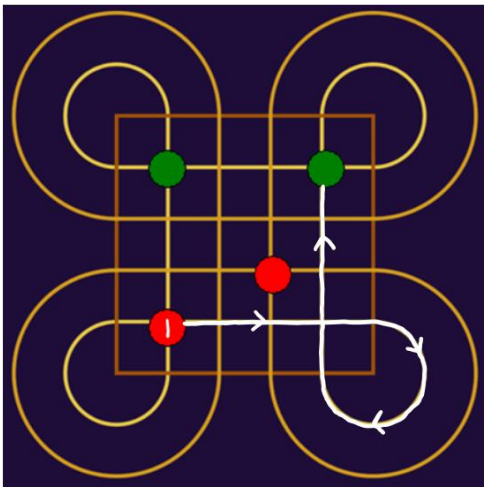
We will now skip to the iteration below where curr_loc is the GridLocation with coordinates (1,4). At this point, one of four the board loops has been used.

```
  invalid: False
> (prev_loc.get_piece_colour(), prev_loc.get_cords()): (None, (2, 4))
> (curr_loc.get_piece_colour(), curr_loc.get_cords()): ('green', (1, 4))
  direction: 'right'
  loop_count: 1
```

At this point the iteration will stop because loop_count is greater than 0, curr_loc (shown as end_location below) is not empty and the colour of the pieces at curr_loc and the capture's start location are not the same. Thus, the capture from the start location (which uses the red piece at (4,1)) to curr_loc is a legal capture.

```
loop_count: 1
end_location.is_empty(): False
end_location.get_piece_colour(): 'green'
start_location.get_piece_colour(): 'red'
loop_count > 0: True
```

However, the end location of this capture is not the same as the end location of the attempted capture. We have in fact validated the capture shown below:



As this is not the attempted capture, we must iterate left from index 1 in looped_track.

In the same way that the right_pointer was set, the left_pointer of looped_track is set to 1.

The third iteration left from index 1 leads to curr_loc being the GridLocation with coordinates (4,1) shown below.

```
> looped_track: <LoopedTrack.LoopedTrack object at 0x0000019E1DD86FE0>
> [(loc.get_piece_colour(), loc.get_cords()) for loc in looped_track.get_lst_dryruns()]: [(None, …
  invalid: False
> (prev_loc.get_piece_colour(), prev_loc.get_cords()): (None, (5, 1))
> (curr_loc.get_piece_colour(), curr_loc.get_cords()): ('red', (4, 1))
  direction: 'left'
  loop_count: 1
```

Iteration continues in this direction after this state because although the piece colour at curr_loc is the same as the piece colour on the start location, the coordinates of these two locations are the same because we have simply returned to the same location during traversal of looped_track (and loop_count is not 4 so we have not traversed the whole looped track yet). The image below illustrates that none of the conditions for a direction being invalid are True in this iteration.

```
(start_location.get_piece_colour() == end_location.get_piece_colour()) and (start_location.get_cords() != end_location.get_cords()): False
loop_count == 0 and not end_location.is_empty(): False
(loop_count == self.NUM_BOARD_LOOPS) and (start_location.get_cords() == end_location.get_cords()): False
```

Eventually, curr_loc becomes the GridLocation with coordinates (1,1).

```
> looped_track: <LoopedTrack.LoopedTrack object at 0x00000138D5E2B430>
> [(loc.get_piece_colour(), loc.get_cords()) for loc in looped_track.get_lst_dryruns()]: [(None,…
  invalid: False
> (prev_loc.get_piece_colour(), prev_loc.get_cords()): (None, (2, 1))
> (curr_loc.get_piece_colour(), curr_loc.get_cords()): ('green', (1, 1))
  direction: 'left'
  loop_count: 1
```

Iteration left now terminates as a valid capture has been found. The capture from the start location to curr_loc is legal because loop_count is greater than 0, curr_loc is not empty and the piece at curr_loc and the start location are not the same colour. This is summarised below where end_location refers to curr_loc.

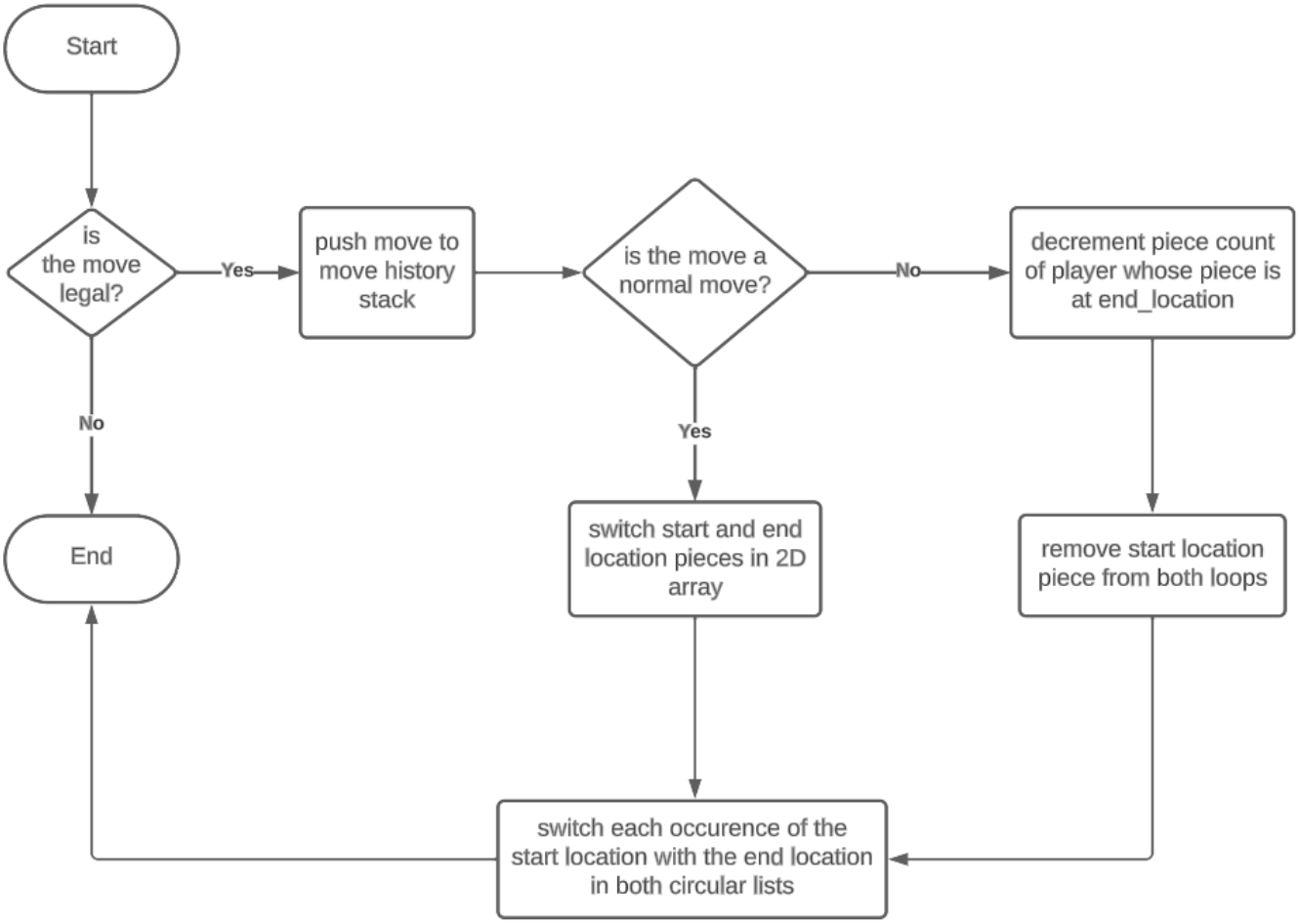
```
end_location.is_empty(): False
end_location.get_piece_colour(): 'green'
start_location.get_piece_colour(): 'red'
loop_count > 0: True
```

This capture's end location is the same as the attempted capture's end location so we can now confirm that the attempted capture is legal.

If after finishing iterating left from the index of 1 in looped_track the attempted capture had not been confirmed to be legal, the algorithm would iterate right through looped_track starting at the index of 22 and then left from this same index. If after this the attempted capture was not found, the attempted capture would be illegal.

2.4.4 Making a Move

Making both a capturing move and a normal move result in changes to the board's circular lists and 2D array. This algorithm is shown in the flowchart bellow.



2.4.5 Generating All Legal Moves from a Board

The following algorithm is used to generate all the legal moves that a player can make in a given board state. This algorithm is used by the Monte Carlo Tree Search algorithm's expansion phase.

Each GridLocation object in the board is iterated over and if the piece at the location belongs to the current player, another algorithm is used to obtain the legal moves that can be made using this piece. These new legal moves are added to a list of all the legal moves that can be made which is returned at the end of the algorithm. This is summarised in the pseudocode below.

```
legal_moves = []

FOR row in board DO
    FOR loc in row DO

        IF loc.get_colour() == player.get_colour() THEN
            add the possible legal moves from loc to legal_moves

return legal_moves
```

To obtain the legal moves for a single location, possible normal adjacent moves and captures need to be considered. Each adacent location is iterated over and if a legal move can be made to it, the move is added to a list of legal moves. To check for captures, each location which does not have the current player's piece is checked to see if it can legally be captured by the current location. If it can, this move is

added to the list of legal moves. This is summarised in the pseudocode below where curr_loc is the GridLocation we want to get the legal moves for.

```
legal_moves = []

// checking for normal adjacent moves
FOR each adjacent_loc to curr_loc DO

    IF moving to adjacent_loc is legal THEN
        add move to legal_moves

// checking for captures
FOR row in board DO
    FOR end_loc in row DO

        IF end_loc.get_colour() != player.get_colour and curr_loc to end_loc is a legal capture THEN
            add move to legal_moves

return legal_moves
```

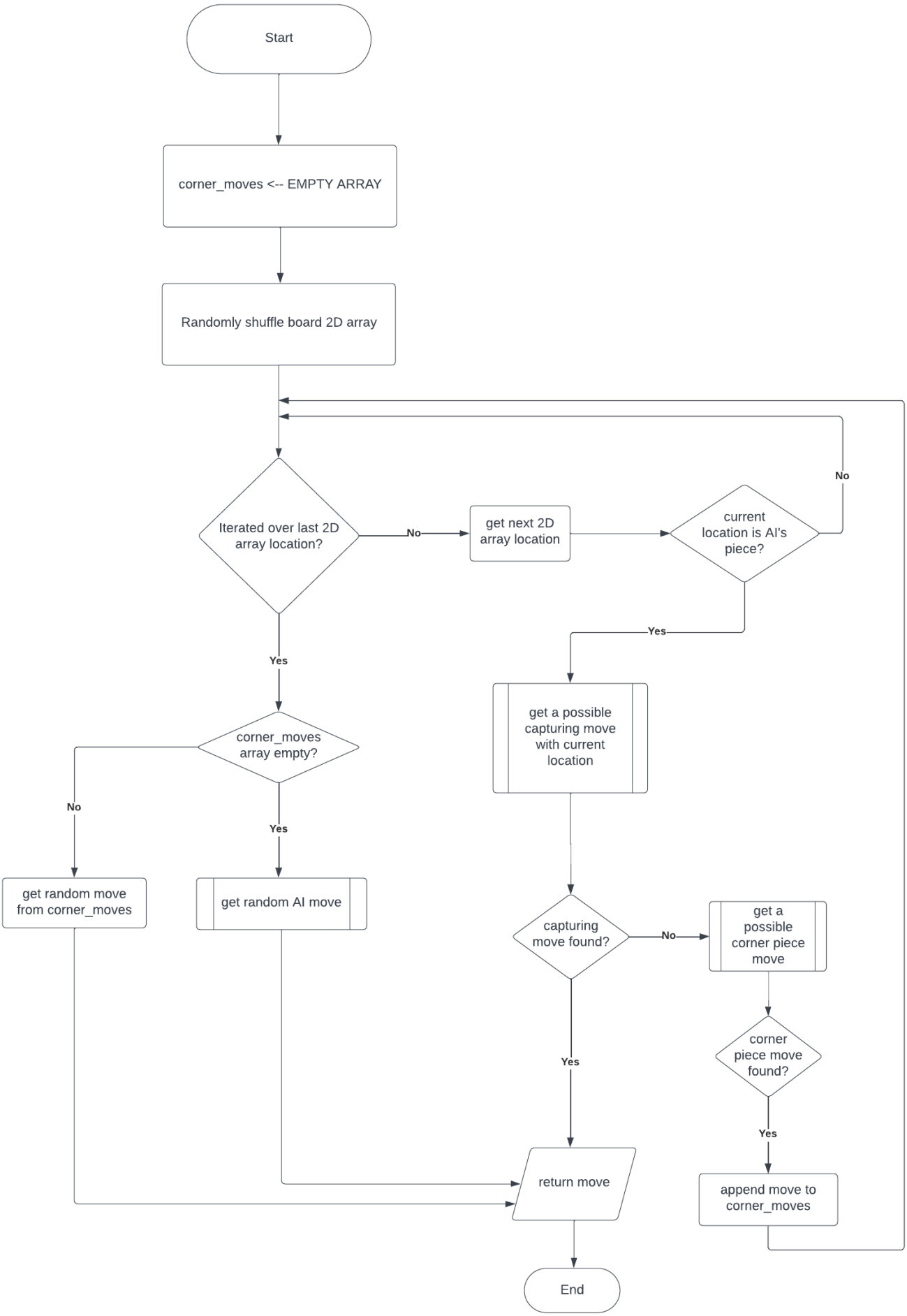
2.4.6 Generating a Single Random Legal Move for a Board

This algorithm is used to generate a single legal move for a board. This is used as part of the rollout phase of the Monte Carlo Tree Search algorithm where random moves must be played. The board is randomly shuffled before the first move is found instead of selecting a random move out of all possible moves to increase the speed of the algorithm. This is summarised in the pseudocode below. Note that the algorithm used to generate the legal moves for a location is explained in the latter part of section 2.4.5.

```
randomly shuffle board

FOR row in board DO
    FOR loc in row DO

        loc_legal_moves = get legal moves for loc

        IF len(loc_legal_moves) > 0 THEN
            return random move from loc_legal_moves
```

2.4.7 Greedy Algorithm for Easy AI

The flowchart below summarises the algorithm used for the Easy AI opponent.


Start
corner_moves <-- EMPTY ARRAY
Randomly shuffle board 2D array
Iterated over last 2D array location?
No
get next 2D array location
current location is AI's piece?
No
Yes
Yes
corner_moves array empty?
get a possible capturing move with current location
No
Yes
get random move from corner_moves
get random AI move
capturing move found?
No
get a possible corner piece move
corner piece move found?
Yes
Yes
return move
append move to corner_moves
End

After the board is shuffled, each location is checked for a capturing move and if one can be made, the move is returned. The location is then checked to see if a corner move can be made which is where a piece on one of the board's four corners can be moved out of the corner. If a corner move can be made, it is added to a list of corner moves. If, after iterating through the whole 2D array, no capturing move was found but the corner moves list is not empty, a random move from this list is returned. But if the corner moves list is empty, a random move that the AI can make is returned. Thus overall, the easy AI makes a capture if possible, otherwise it makes a corner move if possible, otherwise it makes a random move.

As described in section 1.2, captures and corner moves are typically good moves but are not always ideal. Thus, the easy AI is able to win but will lose to a player who is able to set up consecutive captures and play an effective defence.

The pseudocode for getting a possible capture with a given start location is given below.

```
FOR each looped_track common to start_loc and end_loc DO
    starting_indexes = get indexes of start_loc in looped_track

    FOR start_loc_index in starting_indexes DO

        right_move = search_direction_for_capture(start_loc, start_loc_index, looped_track, "right")

        IF right_move THEN
            return right_move

        left_move = search_direction_for_capture(start_loc, start_loc_index, looped_track, "left")

        IF left_move THEN
            return left_move

return False
```

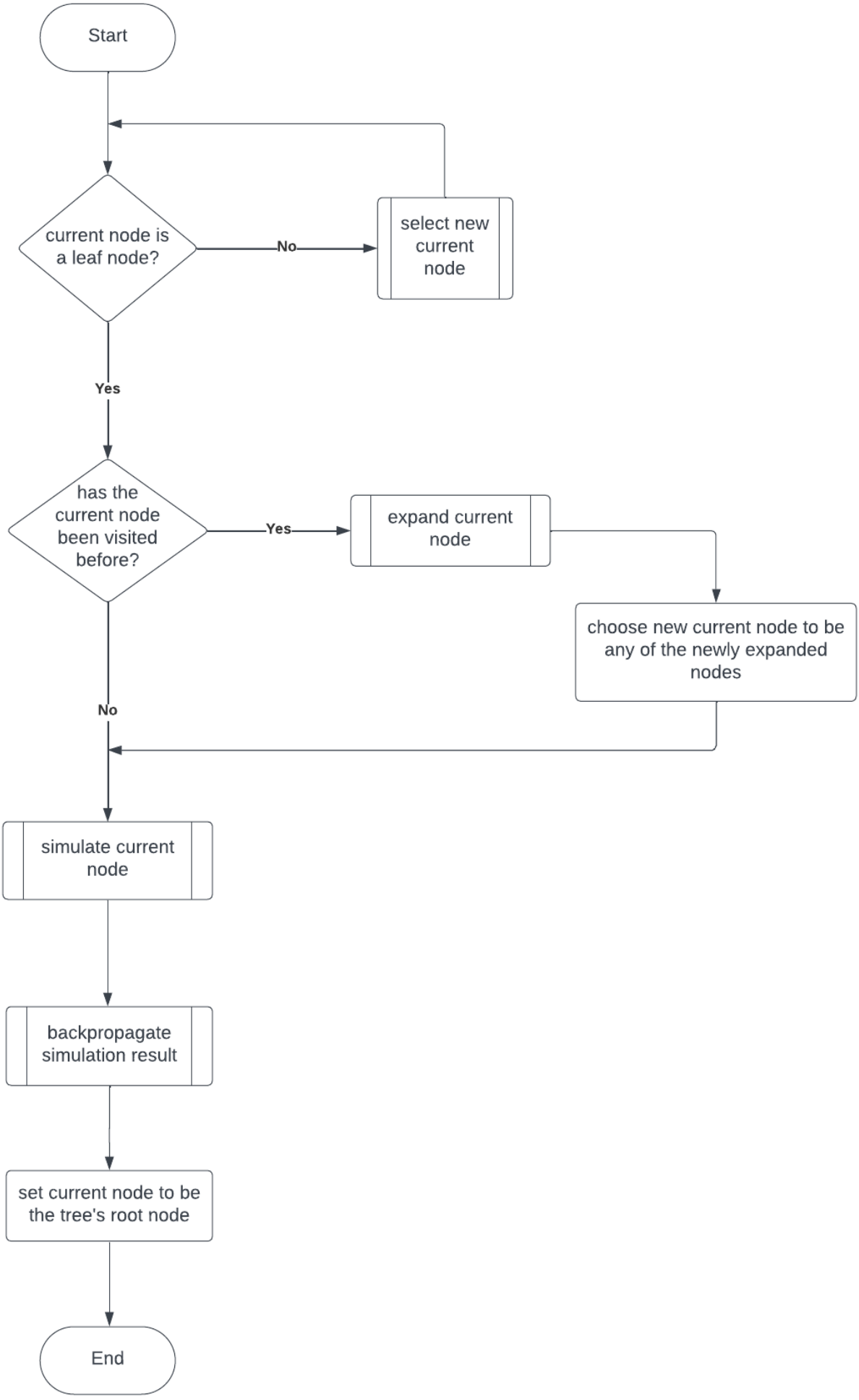
This algorithm is similar to checking for a legal capture but is different in that it:

1) Returns a capturing move as opposed to a boolean
2) Returns any possible capture, not just a specific one

Getting possible corner moves is achieved by checking if a legal move can be made from one of the four corners to any of their adjacent locations.

2.4.8 MCTS for Medium and Hard AI

The medium and hard AI difficulties are achieved with the Monte Carlo Tree Search (MCTS) algorithm. The medium AI runs the algorithm for 15 seconds while the hard AI runs the algorithm for 30 seconds. The following flowchart describes one iteration of MCTS. Function calls are used to indicate each of the four major steps of MCTS.


Start
current node is a leaf node?
No
select new current node
Yes
has the current node been visited before?
Yes
expand current node
choose new current node to be any of the newly expanded nodes
No
simulate current node
backpropagate simulation result
set current node to be the tree's root node
End

The flowchart above describes one full MCTS iteration which is run as many times as possible in the time allotted for the AI to make a move. Thus, by running the algorithm for a longer period of time, more MCTS iterations are performed meaning the tree grows more in depth and laterally and the AI can return a better move.

Note that before the MCTS iterations are run, one initial expansion is made of the root node.

In the selection stage, the UCB1 score of each of the current node's children is calculated. The child node with the highest UCB1 score becomes the new current node. This is repeated until a leaf node is reached.

The expand stage is shown in the pseudocode below:

```
FOR each move in current node legal moves DO
    child_board = deepcopy of current node's board
    child_board.move_piece(move)

    add_node(child_board, move)
```

When expanding a node, all the legal moves that can be made from the node's board are generated (see section 2.4.5 for this algorithm) and each of these moves is used separately on the board to make a new child board. A deep copy of the original board is used to prevent changes to the current node's board during expansion. The add_node method is described below and is used to create a new Node object and add it to the tree.

```
new_depth = current node depth + 1

new_node = Node(board, new_depth, move_obj)

append new_node to current node children list
```

The new node is created with the board and Move object passed as arguments to add_node and the node's depth is calculated to be the current node's depth (which will become this new node's parent) plus one.

A node's tree depth is used to determine the current player. The pseudocode below summarises how it is used.

```
IF depth is even THEN
    return AI turn

ELSE IF depth is odd THEN
    return human turn
```

Player turns alternate as depth increases in the tree. Since the tree is being used for an AI move, the root node will always be the AI's turn. Zero is an even number so all nodes at an even depth have their boards at the AI's turn.

Below is the pseudocode for the rollout stage of the algorithm.

```
rollout_board = deep copy of current node board
num_moves = 0

WHILE num_moves < max moves per rollout DO
    IF rollout_board is terminal state THEN
        return rollout result

    rollout_depth = current node depth + num_moves
    rollout_colour = tree_depth_to_turn(rollout_depth)

    simulated_move = rollout_board.get_single_random_legal_move(rollout_colour)

    rollout_board.move_piece(simulated_move)

    INCREMENT num_moves

return evaluation_function(rollout_board)
```

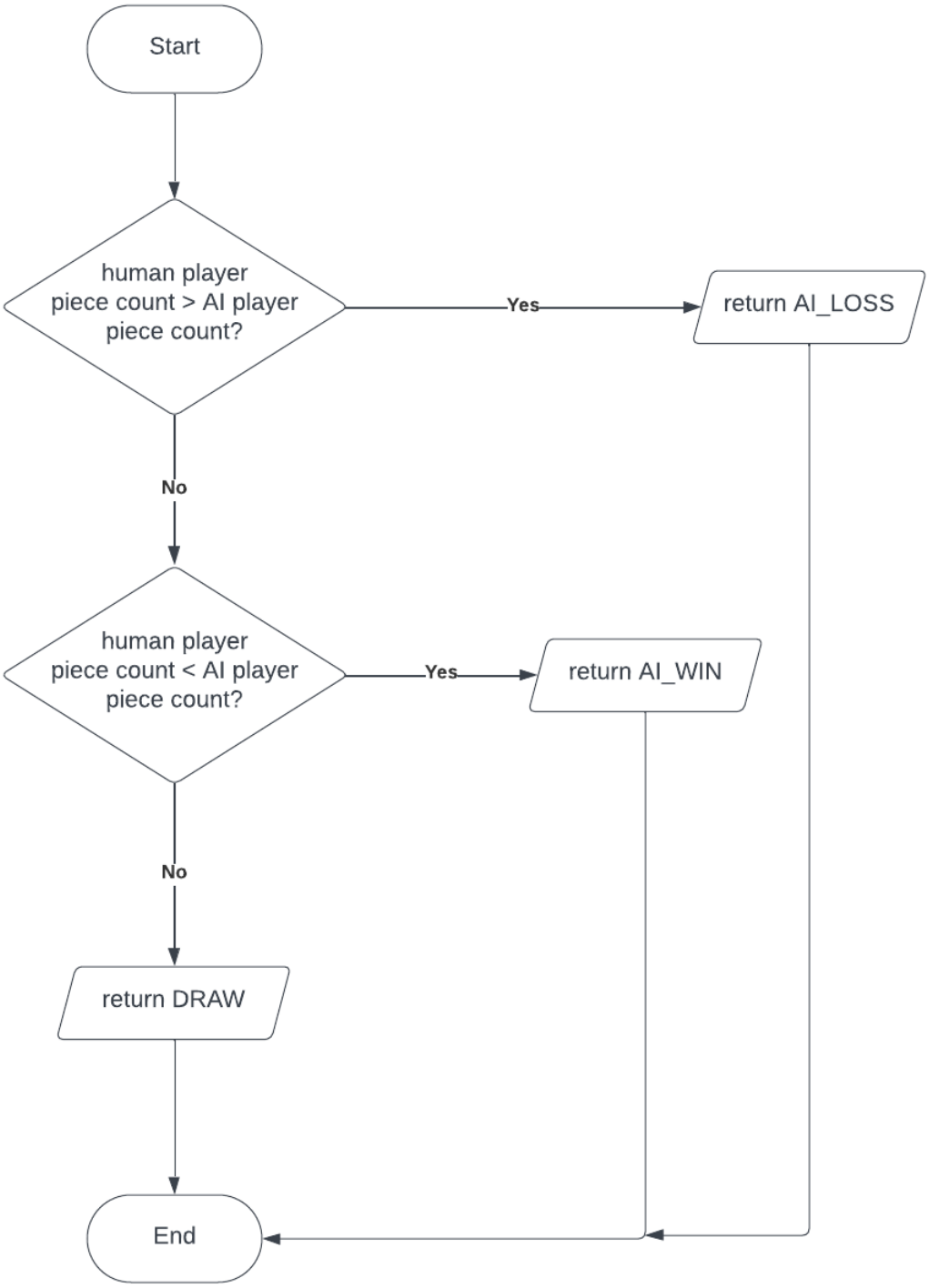
A deep copy of the current node board is made so that simulated moves made during the rollout do not affect the current tree node's board. Throughout the rollout process, the current node in the tree is not changed because no new nodes are added to the tree; The board states and moves made in a rollout are discarded once the rollout result has been determined.

Rollout depth is used in the same way that a node's depth is used to determine the current player's colour in the tree_depth_to_turn() function. Even depths passed to the function return the colour of the AI's pieces and odd depths passed to the function return the colour of the human player's pieces.

To obtain the simulated move, the algorithm to get a single random legal move described in section 2.4.6 is used.

If a terminal board state is not reached in the maximum allotted moves for a rollout, a heuristic evaluation function is called on the rollout board to determine the rollout's result.

The evaluation function uses the number of pieces each player has remaining and is summarised in the flowchart below.



The AI_LOSS constant has a value of -1, the AI_WIN constant has a value of 1, and the DRAW constant has a value of 0.

The algorithm is effective with this evaluation function for two main reasons:

1) Every piece in Surakarta is of the same value
2) A node's evaluation is not solely based on the evaluation function as it also relies on the Monte Carlo method through rollouts

A rollout depth of 500 moves was used for both the medium and hard AI opponents.

Rollouts are also not typically expected to use this evaluation function as most rollouts (even at the start of the game) terminate earlier than 500 moves. The evaluation function instead serves mostly as a safety net in case some rollouts take longer than expected.

The final stage of the MCTS algorithm is backpropagation and is described in the following pseudocode:

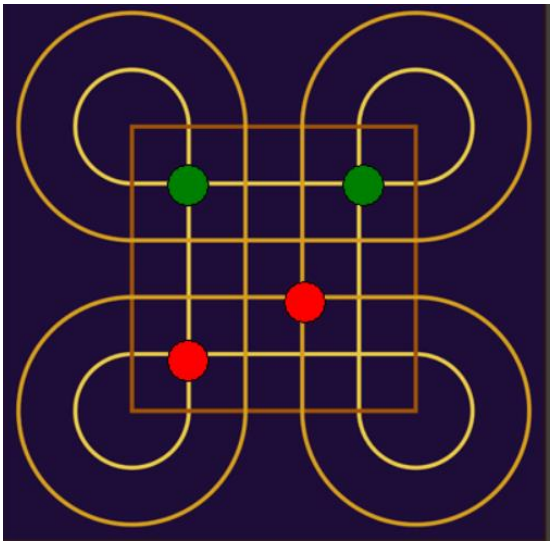
```
node = current_node

WHILE node is not None DO:

    INCREMENT node visited count
    add rollout result to node value
    node = parent of node
```

This results in every node selected in reaching the simulated leaf node having its visited count incremented and the rollout result added to its value attribute.

In my project, the MCTS algorithm is implemented in the GameTree class and when the get_move method is called on a MediumAIPlayer or HardAIPlayer object, a new GameTree object is created, with its root node being the current game state.

The rest of this section will be a dry run of the first iteration of the MCTS algorithm in the context of my project.

The following board state is used for the dry run where it is the AI's turn (green pieces).



First, a GameTree object is created, and its root node (a Node object) is shown below.

```
self._GameTree__root: <TreeSearch.Node object at 0x00000174350D9CC0>
  special variables
  function variables
  _Node__board: <Board.Board object at 0x00000174350DB340>
  _Node__children: []
    _Node__depth: 0
    _Node__move_obj: None
    _Node__parent: None
    _Node__value: 0
    _Node__visited_count: 0
```

After the initial expansion (before the MCTS iterations), the root node's children array is populated with Node objects that can be reached with one move. In this case, there are 18 legal moves that can be made and so 18 child nodes are added. This can be seen in the image below.

```
self._GameTree__root: <TreeSearch.Node object at 0x00000174350D9CC0>
 special variables
 function variables
 _Node__board: <Board.Board object at 0x00000174350DB340>
 _Node__children: [<TreeSearch.Node obj...43511FCA0>, <TreeSearch.Node obj...434D6A560>, <TreeSearch.N…
  _Node__depth: 0
  _Node__move_obj: None
  _Node__parent: None
  _Node__value: 0
  _Node__visited_count: 0
```

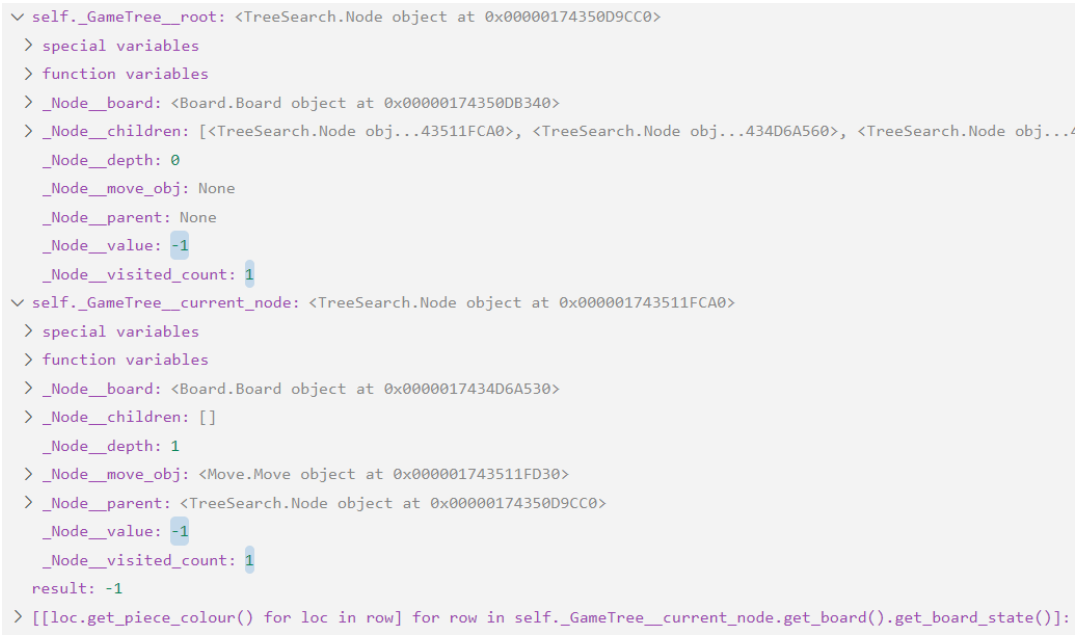
The current node is initially the root node. This is shown below, and the board state of the root/current node is also shown. Each GridLocation object in the board 2D array is represented with its piece colour.

```
self._GameTree__root: <TreeSearch.Node object at 0x00000174350D9CC0>
self._GameTree__current_node: <TreeSearch.Node object at 0x00000174350D9CC0>
[[loc.get_piece_colour() for loc in row] for row in self._GameTree__current_node.get_board().get_board_state()]:
 special variables
 function variables
 0: [None, None, None, None, None, None]
 1: [None, 'green', None, None, 'green', None]
 2: [None, None, None, None, None, None]
 3: [None, None, None, 'red', None, None]
 4: [None, 'red', None, None, None, None]
 5: [None, None, None, None, None, None]
  len(): 6
```

Since the root node is no longer a leaf node after the initial expansion, the root node child with the highest UCB1 score is chosen to be the new current node. Since none of the children have been visited/simulated, they all have infinite UCB1 scores, so we simply choose the first child to be the new current node. The current node is now a leaf node, so the selection stage ends. The new current node is shown below.

```
self._GameTree__root: <TreeSearch.Node object at 0x00000174350D9CC0>
self._GameTree__current_node: <TreeSearch.Node object at 0x000001743511FCA0>
[[loc.get_piece_colour() for loc in row] for row in self._GameTree__current_node.get_board().get_board_state()]:
 special variables
 function variables
 0: ['green', None, None, None, None, None]
 1: [None, None, None, None, 'green', None]
 2: [None, None, None, None, None, None]
 3: [None, None, None, 'red', None, None]
 4: [None, 'red', None, None, None, None]
 5: [None, None, None, None, None, None]
  len(): 6
```

Since the current node has never been visited, we do not expand it. Instead, we go straight to the simulation phase where we make random moves until a terminal state, or the rollout depth is reached. In this case the rollout ended with a loss for the AI so -1 is backpropagated up the tree to every node that was selected in reaching the current node (which in this case is just the current node and the root node). The image below shows the backpropagation phase.



The first iteration of MCTS is now complete so we set the current node back to being the root node. This is shown below.

2.4.9 Serialising a Board State

When saving a match to the database, the current board state needs to be recorded. To do this, the following algorithm is used to serialise the board state into a string.

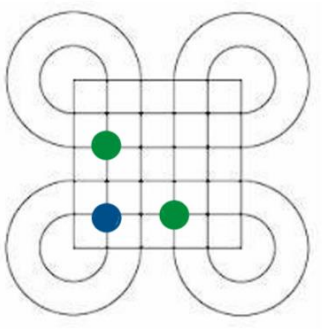
```
flat_board = flatten board state to 1D array

EMPTY_CHAR = "."
SEPARATOR_CHAR = "$"

game_state_lst = []

FOR location in flat_board DO
    IF location is empty THEN
        game_state_lst.append(EMPTY_CHAR)

    ELSE
        colour = colour of piece at location
        game_state_lst.append(colour)

    game_state_lst.append(SEPARATOR_CHAR)

// get rid of last separator
game_state_lst.pop()

game_state_string = join each element in game_state_lst with no spaces into a string

return game_state_string
```

The board state is flattened such that a row's elements are placed after the previous row's elements as follows:

[row1, row2, row3, row4, row5, row6]

Each empty location is represented by a pre-determined character and each piece is represented by its colour. Locations are separated by a pre-determined character in the serialised string. I use a dollar sign ($) as my separator character and a full stop (.) for my empty location character.

Below is an example of a board state and its serialised string representation.



Serialises to:

.$.$.$.$.$.$.$.$.$.$.$.$green$.$.$.$.$.$.$.$.$.$.$blue$.$green$.$.$.$.$.$.$.

Row: 0 1 2 3 4 5

2.5 File structure and organisation

The table below describes the contents of each file in this project. The images in the PieceImages folder are not included in this table for clarity but are 42x42 pixel images containing a circle of a specific colour.

File Name	Description
Main.py	The main file which is executed to run the program.
Board.py	Contains the Board class.
Game.py	Contains the Game class.
GridLocation.py	Contains the GridLocation class.
Move.py	Contains the Move class.
Piece.py	Contains the Piece class.
Player.py	Contains the Player, AIPlayer, EasyAIPlayer, MediumAIPlayer, and HardAIPlayer classes.
UI.py	Contains the UI, TerminalUI, and GraphicalUI classes.
LoopedTrack.py	Contains the LoopedTrack class.
Database.py	Contains the Database class.
TreeSearch.py	Contains the Node and GameTree classes.
UtilityFunctions.py	Contains the utility functions oneD_to_twoD_array and twoD_to_oneD_array. oneD_to_twoD_array converts a one-dimensional array to a two-dimensional array with each sub-array being the same length. twoD_to_oneD_array flattens a two-dimensional array.
MultiClassBoardAttributes.py	Contains the MultiClassBoardAttributes class.
Database.db	The database file.
HelpPageText.txt	Contains the text which is displayed on the help page explaining some background information about the game and the rules of Surakarta.
blank_board.png	An image of a board with no pieces which is used with the display board window as the current piece positions are drawn ontop of this image.
starting_board.png	An image of the board with its pieces in their starting positions. This image is displayed at the bottom of the help page.

Below is a diagram illustrating the folder structure for this project. Image file names are in snake case and other files and inner folders (folders inside the Surakarta-NEA folder) are in pascal case. Image file names use snake case as it enables the file name to be split into a list on the underscores. Pascal case would not be suitable for splitting an image file name as different colour names have different lengths.

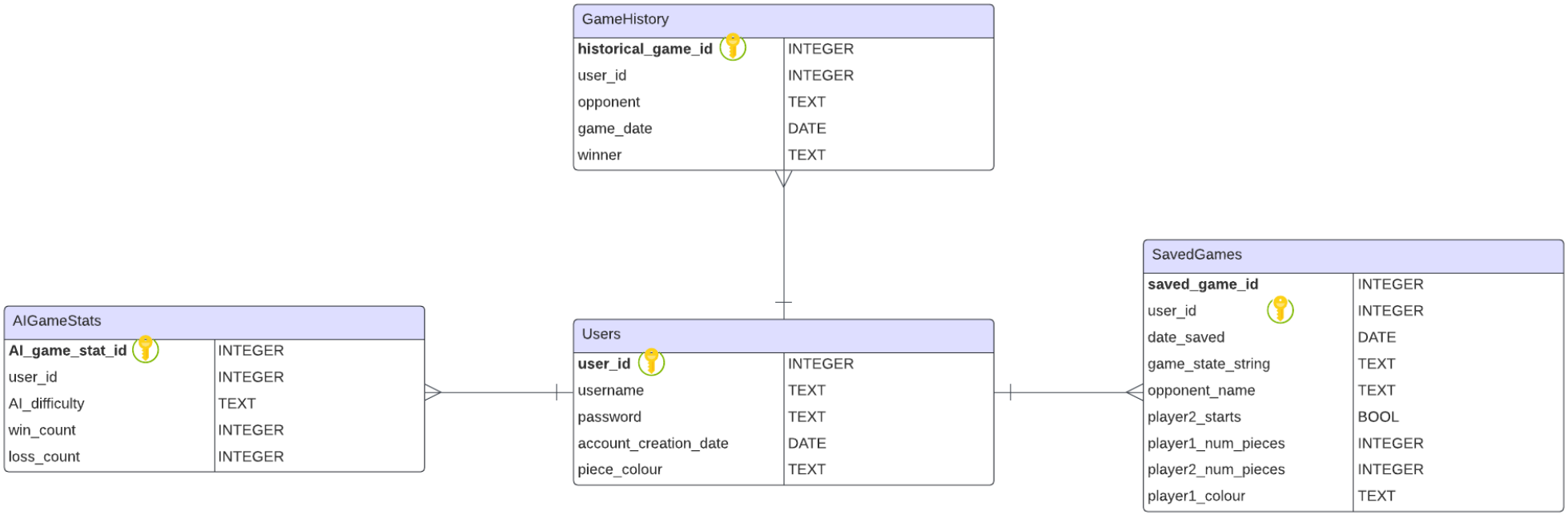
```
Surakarta-NEA
│   Board.py
│   BoardConstants.py
│   Database.db
│   Database.py
│   Game.py
│   GridLocation.py
│   HelpPageText.txt
│   LoopedTrack.py
│   Main.py
│   Move.py
│   MultiClassBoardAttributes.py
│   Piece.py
│   Player.py
│   TreeSearch.py
│   UI.py
│   UtilityFunctions.py
│
└───Images
    ├───BoardImages
    │       blank_board.png
    │       starting_board.png
    │
    └───PieceImages
            black_counter.png
            blank_counter.png
            green_counter.png
            lightblue_counter.png
            orange_counter.png
            red_counter.png
            yellow_counter.png
```

2.6 Database design

In this section, the full schema for the relational database and a description of how passwords are stored is provided. The relational database is in third normal form.

2.6.1 Database Schema

The schema for the project's relational database is shown in the diagram below. Primary keys are indicated in bold with a key icon next to them.


GameHistory
historical_game_id INTEGER
user_id INTEGER
opponent TEXT
game_date DATE
winner TEXT
AIGameStats
AI_game_stat_id INTEGER
user_id INTEGER
AI_difficulty TEXT
win_count INTEGER
loss_count INTEGER
Users
user_id INTEGER
username TEXT
password TEXT
account_creation_date DATE
piece_colour TEXT
SavedGames
saved_game_id INTEGER
user_id INTEGER
date_saved DATE
game_state_string TEXT
opponent_name TEXT
player2_starts BOOL
player1_num_pieces INTEGER
player2_num_pieces INTEGER
player1_colour TEXT

2.6.2 Password storage

To ensure password are stored securely in the database, the sha512 hashing algorithm is used over 100,000 iterations to hash passwords before they are saved. A salt of 16 random bytes is used in the hash and appended in plain text to the hashed password. The use of a salt minimises the risk of a rainbow table attack.

2.7 SQL queries

The following sections show the various SQL commands used to interact with the database. These commands are executed by methods in the Database class with the sqlite3 module. Where relevant, dummy data has been used in the SQL commands in section 2.7.1 to illustrate their usage. The actual SQL in this project is parameterised.

2.7.1 Table creation

The CREATE TABLE commands were run once when the database was first created.

```
CREATE TABLE IF NOT EXISTS Users (
    user_id INTEGER,
    username TEXT,
    password TEXT,
    account_creation_date DATE,
    piece_colour TEXT,
    PRIMARY KEY (user_id)
);

CREATE TABLE IF NOT EXISTS GameHistory (
    historical_game_id INTEGER,
    user_id INTEGER,
    opponent TEXT,
    game_date DATE,
    winner TEXT,
    PRIMARY KEY (historical_game_id),
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

```
CREATE TABLE IF NOT EXISTS AIGameStats (
    AI_game_stat_id INTEGER,
    user_id INTEGER,
    AI_difficulty TEXT,
    win_count INTEGER,
    loss_count INTEGER,
    PRIMARY KEY (AI_game_stat_id),
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

CREATE TABLE IF NOT EXISTS SavedGames (
    saved_game_id INTEGER,
    user_id INTEGER,
    date_saved DATE,
    game_state_string TEXT,
    opponent_name TEXT,
    player2_starts BOOLEAN,
    player1_num_pieces INTEGER,
    player2_num_pieces INTEGER,
    player1_colour TEXT,
    PRIMARY KEY (saved_game_id),
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

2.7.2 Insert statements

When a new user is created, a new record is added to the Users table. In addition to this, three new records are added to the AIGameStats table to store the number of wins and losses against each AI difficulty level, with win_count and loss_count being set to 0. Note that “510708684ea4” is part of a hashed password. The actual hashes stored in the database are 160 characters long (including the salt).

```
INSERT INTO User VALUES (0, "user123", "510708684ea4", "2023-10-23", "orange");
INSERT INTO AIGameStats VALUES (0, 0, "Easy AI", 0, 0)
INSERT INTO AIGameStats VALUES (0, 1, "Medium AI", 0, 0)
INSERT INTO AIGameStats VALUES (0, 2, "Hard AI", 0, 0)
```

The following two insert statements are used to add new records to the GameHistory and SavedGames tables respectively. The GameHistory insert is used after a game is completed when a user is logged in. The SavedGames insert is used when a logged in user saves a game.

```
INSERT INTO GameHistory VALUES (0, 0, "Medium AI", "2023-10-16", "user123");
```

```
INSERT INTO SavedGames
VALUES      (0,
             0,
             "2023-10-16",
             "red$red$.$red$red$.$.",
             "opponent123",
             True,
             4,
             5,
             "black");
```

Note that the serialised board passed as the game_state_string field to SavedGames is a shortened version of an actual game_state_string for clarity.

2.7.3 Single table select statements

This section contains the single table select statements used throughout the program.

The select statements that select from the Users table below are separated because the data is required at different points during the program's execution.

The select command below is used when authenticating a user as it obtains the stored password hash corresponding to a username. If an inputted password hashes (with the stored salt) to the stored hash, the user is logged in successfully.

```
SELECT password FROM Users WHERE username = "user123";
```

The select command below is used to obtain a player's piece colour prior to a match. The game then makes the player's piece colour their desired colour. By default, the opposing player (local play or AI play) uses green pieces. If the logged in user's piece colour is green, then the opposing player will use yellow pieces.

```
SELECT piece_colour FROM Users WHERE username = "user123";
```

The select command below is used to obtain a user_id from a username. This is used when inserting and updating records into the tables which aren't the Users table because username is only stored in the Users table.

```
SELECT user_id FROM Users WHERE username = "user123";
```

The select command below is used to determine if a username already exists in the database when creating a new account. If the query returns an empty result, the username is not present in the database and so can be used for a new account. If the query result is not empty, the user is informed that they need to choose a different username.

```
SELECT username FROM Users WHERE username = "user123";
```

The select command below is used to obtain a user's username given their user_id. This is used to obtain the user ID of a logged in user when adding a completed game to the GameHistory table or when saving a game state to the SavedGames table.

```
SELECT username FROM Users WHERE user_id = 3;
```

The pseudocode select statement below is used when getting the next primary key in a table. The next primary key is taken to be the result of the following select statement plus one.

```
SELECT MAX(primary key field) FROM TABLENAME;
```

2.7.4 Multi-table select statements

This section contains the multi-table select statements used in the program where the user_id foreign key is used to obtain the username attribute from the Users table.

The select statement below is used to obtain a user's game stats from the AIGameStats table given a username. This is used to display the user's stats when requested.

```
SELECT AI_difficulty, win_count, loss_count FROM AIGameStats
    INNER JOIN Users ON Users.user_id = AIGameStats.user_id
WHERE username = "user123";
```

The select statement below is used to obtain a user's saved games from the SavedGames table. This information is used to populate the table in the "Load Game" popup modal window.

```
SELECT saved_game_id, date_saved, opponent_name FROM SavedGames
    INNER JOIN Users ON Users.user_id = SavedGames.user_id
WHERE username = "user123";
```

The select statement below is used to get all of a user's previously played game results from the GameHistory table to be displayed when a user's stats are requested.

```
SELECT historical_game_id, game_date, opponent, winner FROM GameHistory
    INNER JOIN Users ON Users.user_id = GameHistory.user_id
WHERE username = "user123";
```

The select command below is used when loading a specific saved game for a user so that it can be continued.

```
SELECT username,
       game_state_string,
       opponent_name,
       player2_starts,
       player1_num_pieces,
       player2_num_pieces,
       player1_colour
FROM   SavedGames
       INNER JOIN Users
               ON Users.user_id = SavedGames.user_id
WHERE  saved_game_id = 10;
```

2.7.5 Update statements

The following update statements are used to update the AIGameStats table after a match. If the match ends in a win for the user, the relevant AI difficulty record for the user has its win_count field

incremented. On the contrary if the match ended in a loss for the user, the loss_count field is incremented.

```
UPDATE AIGameStats
SET win_count = win_count + 1
WHERE user_id = 3 AND AI_difficulty = "Easy AI";

UPDATE AIGameStats
SET loss_count = loss_count + 1
WHERE username = 3 AND AI_difficulty = "Hard AI";
```

The update command below is used to update a user's piece_colour field if a logged in user decides to change their preferred piece colour.

```
UPDATE Users
SET piece_colour = "black"
WHERE username = "user123";
```

2.7.6 Delete statements

The following delete statement is used to delete a saved game. This command is executed after the saved game has been loaded and completed or deleted manually from a user's saved games by the user.

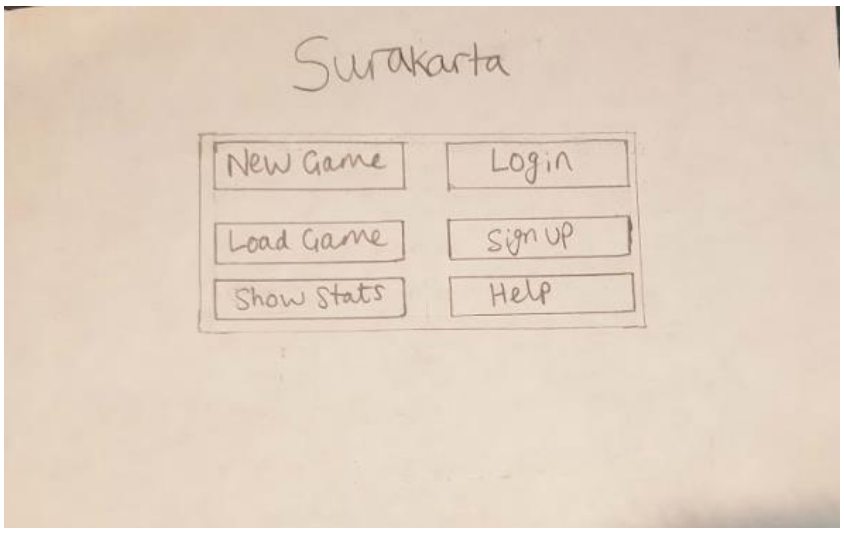
```
DELETE FROM SavedGames
WHERE saved_game_id = 1;
```

2.8 User interface design

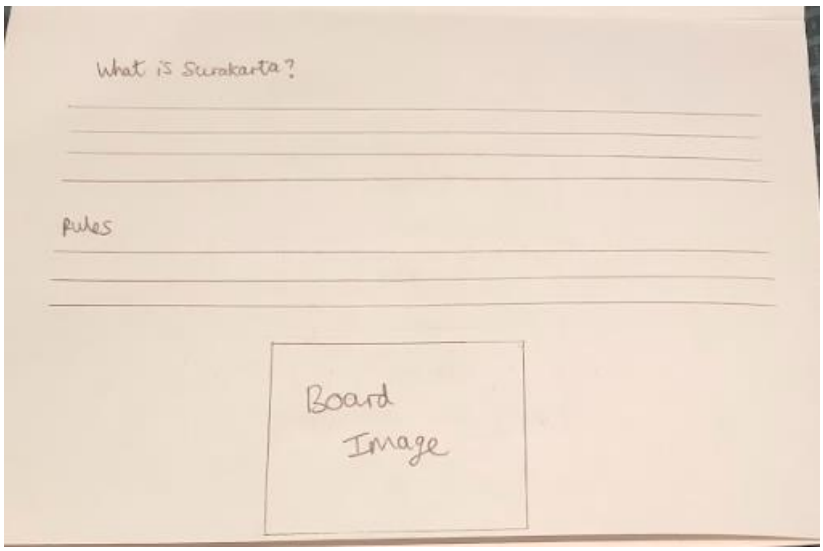
In this section, sketches of the GUI are provided. The sketches were drawn after researching the capabilities of the PySimpleGUI library and the actual program GUI was modelled after these sketches. It was designed and implemented such that at the top of each window, a menu bar is present containing buttons.

Windows described as "modal windows" are such that the user cannot interact with the background window while the modal window is open. Only when the modal window is closed can the user interact with the background window.

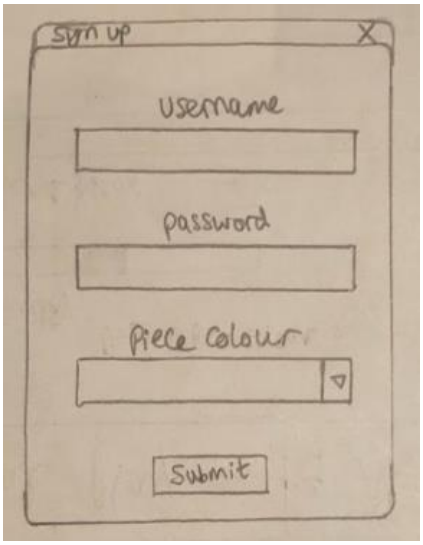
1) "Home" page
    a. On the home page, there are six buttons which can be pressed.



2) “Help” page
    a. The help page describes the game and explains the rules of the game in a paragraph of text. It also has a board image at the bottom of the page. You arrive at the help page after clicking the “Help” button on the home page. This closes the home page window and opens the help page window.

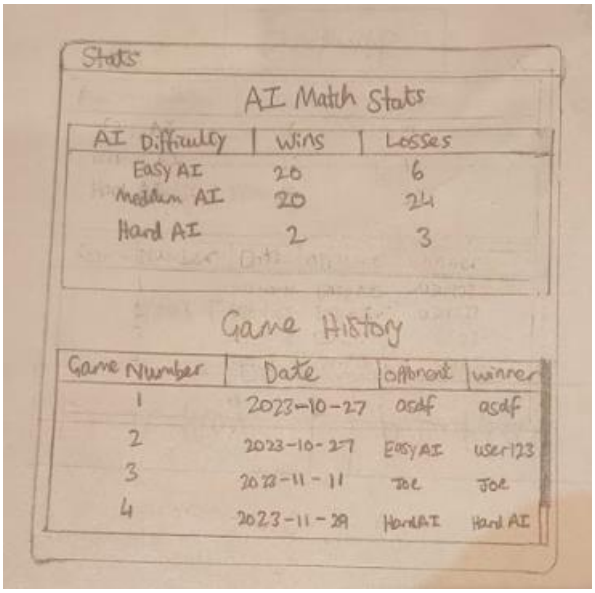


3) “Sign Up” modal window
    a. The sign-up modal window is a popup window that appears when you press the “Sign Up” button on the home page. There are fields to enter your username and password. There is also a piece colour drop-down menu and a submit button.

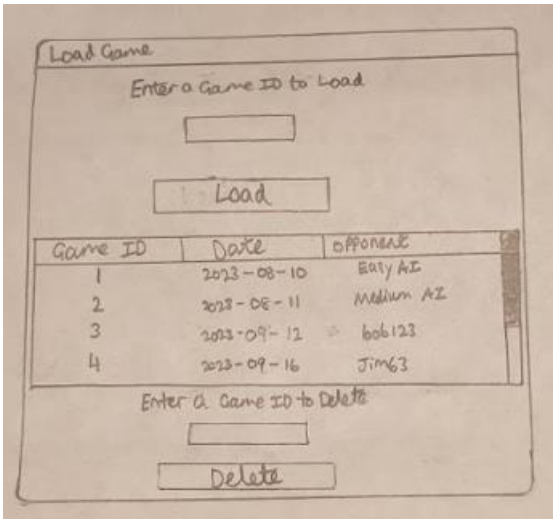


4) “Login” modal window
    a. The login modal window is a popup window that appears when you press the “Login” button on the home page. There are fields to enter your username and password and there is submit button.

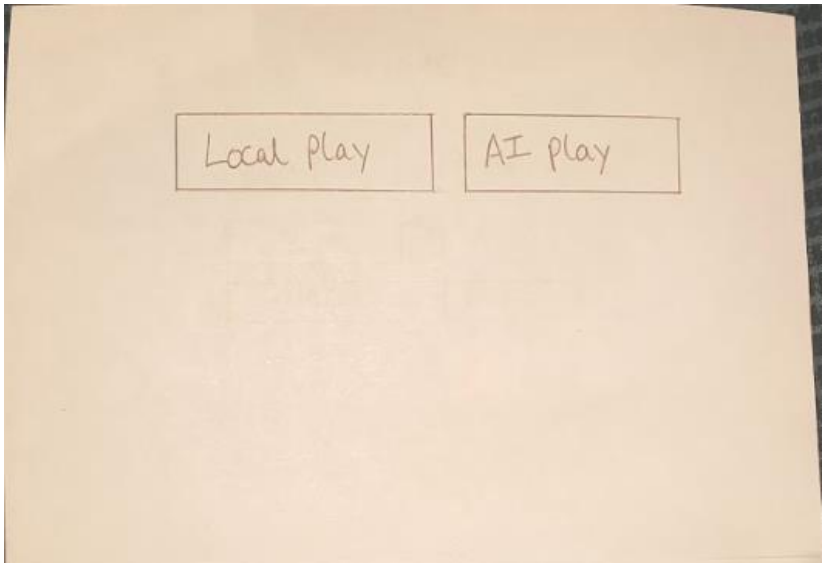
5) “Stats” modal window
    a. The stats modal window is a popup window that appears when you press the “Show Stats” button on the home page. Two tables are displayed in this window: one showing the user’s stats against each AI difficulty and another showing the user’s game history.



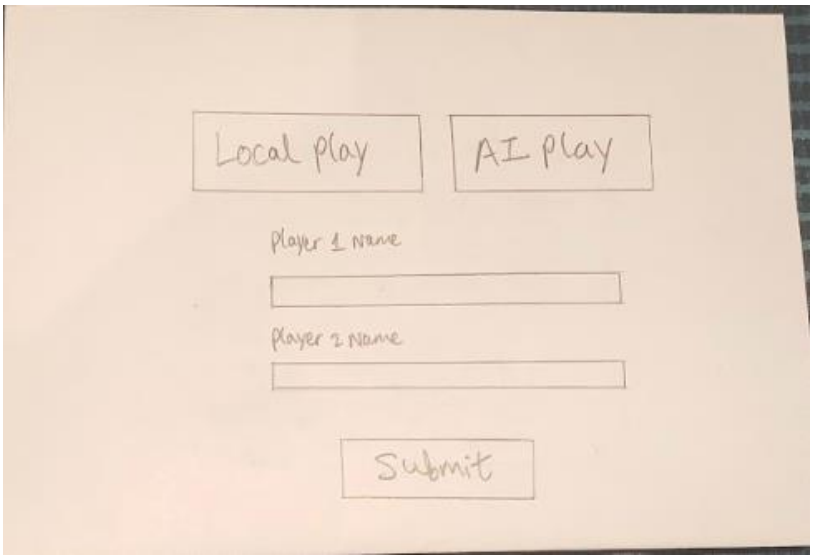
6) “Load Game” modal window
    a. The load game modal window is a popup window that appears when you press the “Load Game” button on the home page. In the centre of the window, a table shows the logged in user’s currently saved games. At the top of the window, there is an input box where the user can type in a Game ID from the table and once they press the “Load” button, they are brought to the match page with the saved game loaded. At the bottom of the window, there is an input box where the user can type in a Game ID from the table and when they press the “Delete” button, that game will be deleted from their saved games and the table is updated.



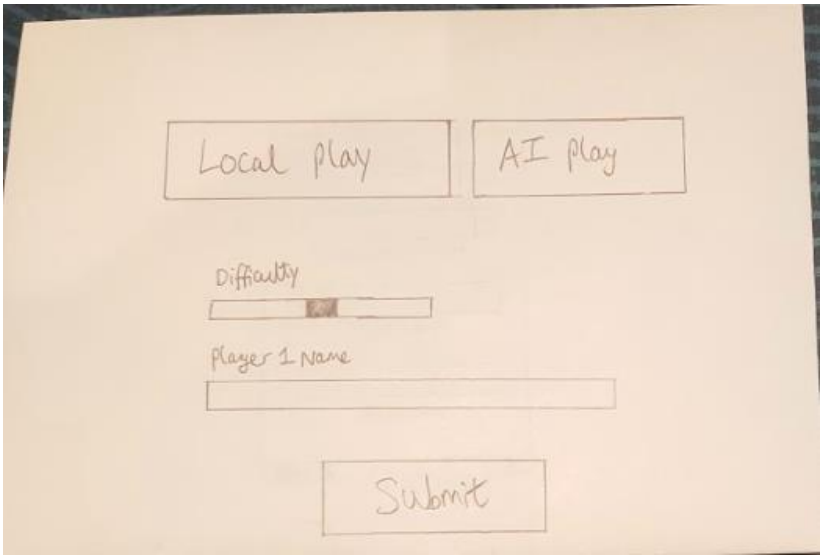
7) “New Game” page
    a. The play type selection page is arrived at after pressing the “New Game” button on the home page. When this button is pressed, the home page window is closed, and a new play type selection window is opened. There are “Local Play” and “AI Play” buttons which are toggle buttons. When one is pressed, the other’s form information disappears and is replaced by a new form.



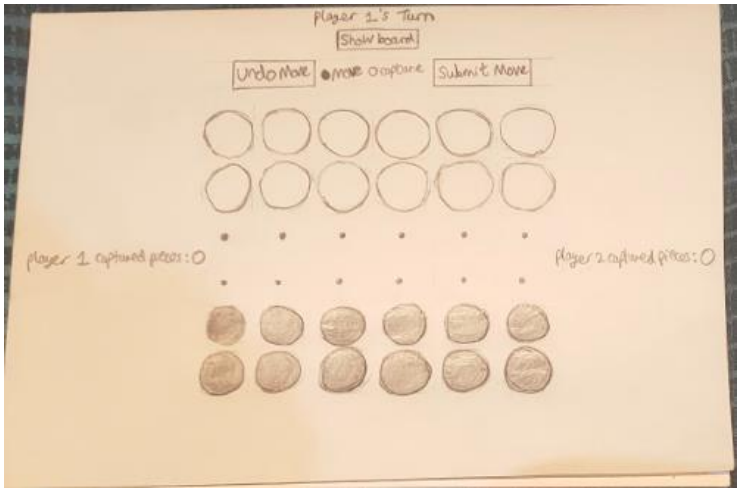
8) Local play input form (shown after “Local Play” button is pressed)
    a. When the “Local Play” button is pressed, a form appears where the user can input the names for player 1 and player 2 and press submit to enter a new game. If a user is logged in, there will only be an input field for player 2 as their username will be used as the name for player 1.



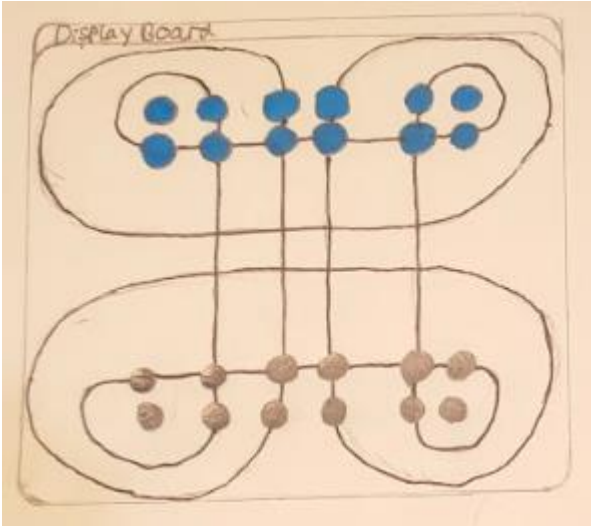
9) AI play input form (shown after “AI Play” button is pressed)
    a. When the “AI Play” button is pressed, a form appears where the user can input the name for player 1 and use a slider to select the AI difficulty (1, 2 or 3). A submit button can be pressed after this to enter the game. If a user is logged in, there will only be the AI difficulty slider since their username will be used as the name for player 1.



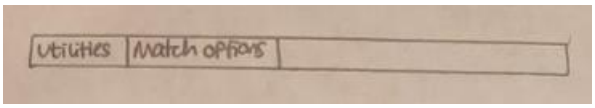
10) "Match" page
    a. The match page is arrived at after pressing the "Submit" button on the "New Game" page. The game board is displayed as a 6x6 grid of buttons with images of pieces on them. The user presses the piece they wish to move and then presses the location they wish to move to. By pressing a piece, the button is highlighted pink. The program only allows a user to highlight two pieces at once. Clicking a piece unhighlights that piece. Pressing the "undo move" button undoes the last move. The radio buttons allow the user to select "move" or "capture" for what type of move they want to perform. Pressing the "submit move" button makes the move specified by the highlighted board positions and the radio buttons. As pieces are captured, the captured piece counts are updated on the display. The player whose turn it is displayed at the top of the board. It was later decided after implementation that the colour of the current player's pieces should also be displayed next to the player's name at the top of the board. Since this was added later, it is not on the sketch.



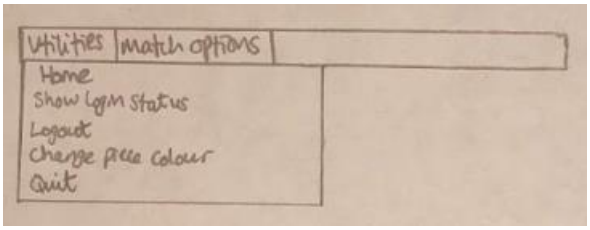
11) Display board modal window ("show board" button is pressed)
    a. Pressing the "show board" button opens a popup modal window showing the current board state with the board's looped tracks displayed. The user can continue to play the match on the board piece buttons while the display board is open.



12) Menu bar
    a. At the top of every main window (non-modal window) which refers to the “Home”, “New Game” and “Match” pages, there is a menu bar.



    b. Pressing the “Utilities” button opens a dropdown menu with buttons.



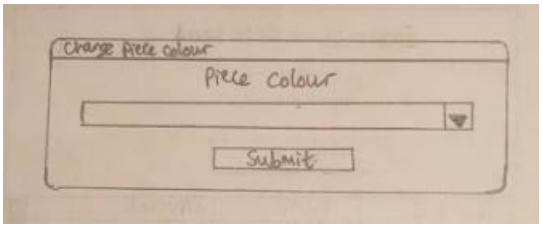
Pressing the “Home” button returns the user to the home page.

Pressing the “Show Login Status” button shows a small popup modal window informing the user if they are logged in or not and their username if they are logged in.

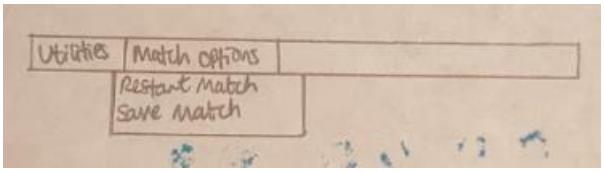
Pressing the “Logout” button logs out the currently logged in user and the user is informed of the success/failed logout with a small popup modal window. Failure occurs if the button is pressed when no user is logged in.

Pressing the “Quit” button exits the program.

Pressing the “Change Piece Colour” button opens a popup modal window where a logged in user can change their piece colour. The user can select a new piece colour from a drop-down menu and then press a submit button to make the change. If no user is logged in, an error popup is displayed. If a user is logged in but no piece colour is selected when the submit button is pressed, a different appropriate error popup is displayed. The change piece colour modal window is shown below.



c. Pressing the “Match Options” button opens another dropdown menu with buttons



If the “Restart Match” button is pressed on the “Match” page, the match page is reloaded with a new game. If this button is pressed on any other page, an error popup is displayed.

If the “Save Match” button is pressed on the “Match” page and a user is logged in, the current game state is saved to the database. If no user is logged in when the button is pressed, an error popup is displayed. If this button is pressed on any other page, a different error popup is displayed.

2.9 User guide

To run the program, execute the Main.py file. You will be prompted to enter either “t” for terminal mode or “g” for graphical user interface mode. The table below has the modules that require installation and their pip install commands.

Module	Pip Install Command
Pillow	pip install Pillow
PySimpleGUI	pip install PySimpleGUI

PySimpleGUI is used to make the graphical user interface. The underlying principle of the module is that widgets like a button or text box are defined and placed inside a layout array. Each row in a layout array is a new row on the screen and widgets in the same array are placed adjacent to each other on the screen. Interaction with the user interface is handled by a main event loop. Layouts can be nested, and widgets can have their parameters updated during runtime.

Below is an example layout used for the buttons on the home page.

```
buttons_layout = [
    [new_game_button, login_button],
    [load_game_button, signup_button],
    [show_stats_button, help_button]
]
```

This layout is then nested in the main home page layout.

PySimpleGUI provides an abstraction ontop of the Tkinter library while also allowing access to underlying Tkinter functionality.

Pillow is used with the underlying Tkinter Canvas object in the display board window to show an image of a blank board onto which the current piece positions are drawn.

3 Technical Solution *(42 marks)*

3.1 Completeness Section

All the objectives set for this project (which were obtained from a user interview) have been fully met. The technical solution fully matches the original project description and requests from the user.

3.1.1 Overview Guide

In this section, two tables are presented. The first describes the class A and class B skills used in this project as well as the files that they are found in. The second shows the coding style used throughout the project and the category that they belong to.

3.1.2 Technical Skills

The following table shows the class A and class B skills demonstrated in this project and the files that they are found in. These skills are also annotated in comments in the files they're found in.

Skill	File it's found in	Skill class
Monte Carlo Tree Search algorithm	TreeSearch.py	Class A
Greedy algorithm for Easy AI	Player.py	Class A
Circular list data structure and traversal	LoopedTrack.py, Board.py	Class A
Stack data structure	Game.py, Stack.py	Class A
Tree data structure and traversal	TreeSearch.py	Class A
Cross-table parameterised SQL	Database.py	Class A
Complex OOP model with encapsulation, inheritance, composition, and polymorphism	Complex OOP with encapsulation in all files Polymorphism and inheritance in UI.py and Player.py Composition in UI.py, TreeSearch.py, LoopedTrack.py, GridLocation.py, Game.py, Move.py and Board.py	Class A
Determining if a capturing move is legal (user defined algorithm)	Board.py	Class A
Determining if a normal move to an adjacent location is legal (user defined algorithm)	Board.py	Class A
Moving a piece and updating the LoopedTrack objects and board 2D array (user defined algorithm)	UI.py, Game.py, Board.py	Class A

Undoing a move with a stack and passing information between the UI, Game and Board classes (user defined algorithm)	UI.py, Game.py, Board.py	Class A
Obtaining all legal moves that a player can make on the board (user defined algorithm)	Board.py	Class A
Updating the UI based on user interaction such as moving pieces, navigating through the application's pages, and displaying information such as stats, saved games and game history	UI.py	Class A
Regex for the validation of board coordinates	UI.py	Class A
Multi-dimensional arrays	Board.py	Class B
Serialising a board object's state into a string for storage in a database (user defined algorithm)	Board.py	Class B
Dictionary data structure	UI.py, Game.py, Board.py	Class B

3.1.3 Coding styles

The following table shows the coding styles used in this project and the style type category they belong to. Note that coding styles which are found in all files have not been included in the code's comments. But coding styles which are not found in all files are included in the comments of the files they are found in.

Characteristic	File	Style Type
Modules with appropriate interfaces	All files	Excellent
Loosely coupled modules	All files	Excellent
Cohesive modules	All files	Excellent
Subroutines with common purpose are grouped into modules	All files	Excellent
Good exception handling	TreeSearch.py	Excellent
Defensive programming	All files	Excellent
Modularisation of code	All files	Excellent

Appropriate indentation	All files (style choice as part of Database.py SQL commands)	Good
Well-designed user interface	UI.py	Good
Good use of local variables	All files	Good
No global variables	All files	Good
Managed casting of types	All files	Good
Use of constants	UI.py, TreeSearch.py, MultiClassBoardAttributes.py, GridLocation.py, Game.py, Board.py	Good
Self-documenting code	All files	Good
Consistent stye throughout	All files	Good
File paths parameterised	UI.py, Database.py	Good
Meaningful identifier names	All files	Basic
Annotation used effectively where required	All files	Basic

3.1.4 Code listing

Board.py

```
from LoopedTrack import LoopedTrack
from GridLocation import GridLocation
from MultiClassBoardAttributes import MultiClassBoardAttributes
from UtilityFunctions import oneD_to_twoD_array, shuffle_2D_array, twoD_to_oneD_array
from Piece import Piece
from Move import Move
import random

class Board:

    """Represents the board for the game. The main board is represented by a 2D array of GridLocation objects.
    The board also contains two LoopedTrack objects representing the inner and outer looped tracks of the board.

    ######################################################################
    CLASS A SKILL: Complex OOP model with encapsulation and composition
    GOOD CODING STYLE: Use of constants
    ######################################################################

    """

    NUM_BOARD_LOOPS = 4

    # character used to separate pieces in the game state string
    SAVED_GAME_STATE_SEPARATOR = "$"
    SAVED_GAME_STATE_EMPTY_CHAR = "."

    # coordinates of locations on the outer and inner tracks
    OUTER_TRACK_CORDS = [
        (5,2), (4,2), (3,2), (2,2), (1,2), (0,2),
        (2,0), (2,1), (2,2), (2,3), (2,4), (2,5),
        (0,3), (1,3), (2,3), (3,3), (4,3), (5,3),
        (3,5), (3,4), (3,3), (3,2), (3,1), (3,0),
    ]

    INNER_TRACK_CORDS = [
        (4,0), (4,1), (4,2), (4,3), (4,4), (4,5),
        (5,4), (4,4), (3,4), (2,4), (1,4), (0,4),
        (1,5), (1,4), (1,3), (1,2), (1,1), (1,0),
```

```
        (0,1), (1,1), (2,1), (3,1), (4,1), (5,1),
    ]


    def __init__(self, player1, player2, game_state_string=None):

        # data structures for the board
        self.__board = []
        self.__inner_track = LoopedTrack([GridLocation(i) for i in self.INNER_TRACK_CORDS],
MultiClassBoardAttributes.INNER_TRACK_STRING)
        self.__outer_track = LoopedTrack([GridLocation(i) for i in self.OUTER_TRACK_CORDS],
MultiClassBoardAttributes.OUTER_TRACK_STRING)

        # populate the board with GridLocation objects
        self.__build_board()

        # edit the pieces at certain GridLocation objects to match the game state string
        if game_state_string:
            self.__load_game_state(game_state_string)

        # player objects are also stored in Board solely for MCTS. All other player related methods are in the Game class
        self.__player_tuple = (player1, player2)

        """
        ######################################################################
        CLASS B SKILL: Dictionary
        ######################################################################
        """

        # maps a text representation of a track to a tuple containing the LoopedTrack objects for the inner and outer tracks
        self.track_text_to_tuple_map = {
            MultiClassBoardAttributes.INNER_TRACK_STRING: (self.__inner_track, None),
            MultiClassBoardAttributes.OUTER_TRACK_STRING: (None, self.__outer_track),
            MultiClassBoardAttributes.BOTH_TRACK_STRING: (self.__inner_track, self.__outer_track),
            None: (None, None)
        }

        # maps a player colour to a player object
        self.__player_colour_map = {
            MultiClassBoardAttributes.player_1_colour: player1,
            MultiClassBoardAttributes.player_2_colour: player2
```

```
        }

    def get_board_state(self):
        return self.__board

    def __load_game_state(self, game_state_string):

        """Updates the pieces at each GridLocation in the board 2D array and LoopedTrack objects
        to match the game_state_string passed in as an argument"""

        # split the game state string into a list of strings representing the pieces at each location
        game_state_lst = game_state_string.split(self.SAVED_GAME_STATE_SEPARATOR)

        # replace character representing an empty location with None
        game_state_lst = [None if i == self.SAVED_GAME_STATE_EMPTY_CHAR else i for i in game_state_lst]

        # convert the list of strings into a 6x6 2D array
        game_state_lst = oneD_to_twoD_array(game_state_lst, MultiClassBoardAttributes.MAX_ROW_INDEX + 1)

        for i in range(MultiClassBoardAttributes.MAX_ROW_INDEX + 1):
            for j in range(MultiClassBoardAttributes.MAX_ROW_INDEX + 1):

                # get the string representing the piece at the current location
                curr_piece_str = game_state_lst[i][j]
                curr_cords = (i, j)

                # empty location
                if curr_piece_str == None:
                    self.__board[i][j].set_piece(None)

                # location with a piece
                else:
                    self.__board[i][j].set_piece(Piece(curr_piece_str))

                # update the piece at the corresponding location in the outer track
                if curr_cords in self.OUTER_TRACK_CORDS:
                    self.__outer_track.update_piece(curr_cords, curr_piece_str)

                # update the piece at the corresponding location in the inner track
                if curr_cords in self.INNER_TRACK_CORDS:
                    self.__inner_track.update_piece(curr_cords, curr_piece_str)
```

```
    def __get_common_tracks(self, text_track_1, text_track_2):

        """Returns a tuple in the form in the form (inner_track, outer_track) containing the common
        tracks between the two text track representations passed in as arguments. If a track is not common,
        the corresponding element in the tuple will be None."""

        # get the LoopedTrack tuples for the inner and outer tracks from the text representations
        track_1_tuple = self.track_text_to_tuple_map[text_track_1]
        track_2_tuple = self.track_text_to_tuple_map[text_track_2]

        common_tracks = []

        for a,b in zip(track_1_tuple, track_2_tuple):
            if a and b:
                common_tracks.append(a)

        return tuple(common_tracks)

    def __get_track_from_text(self, text_track):

        """Returns the LoopedTrack tuple corresponding to the text representation of a track passed in as an argument"""

        return self.track_text_to_tuple_map[text_track]

    def __build_board(self):

        """Populates the board with GridLocation objects

        ######################################################################
        CLASS B SKILL: Multi-dimensional array
        ######################################################################

        """

        board = []

        for i in range(MultiClassBoardAttributes.MAX_ROW_INDEX + 1):
            for j in range(MultiClassBoardAttributes.MAX_ROW_INDEX + 1):

                location = GridLocation((i, j))
```

```
            board.append(location)

        # convert the 1D array into a 6x6 2D array
        self.__board = oneD_to_twoD_array(board, MultiClassBoardAttributes.MAX_ROW_INDEX + 1)

    def __is_valid_coordinate(self, coordinate):

        """Returns True if coordinate is a valid coordinate on the board otherwise returns False.
        A valid coordinate is a tuple of the form (x, y) where x and y are integers between 0 and 5 inclusive"""

        if coordinate[0] < MultiClassBoardAttributes.MIN_ROW_INDEX or coordinate[0] > MultiClassBoardAttributes.MAX_ROW_INDEX:
            return False

        if coordinate[1] < MultiClassBoardAttributes.MIN_ROW_INDEX or coordinate[1] > MultiClassBoardAttributes.MAX_ROW_INDEX:
            return False

        return True

    def __is_valid_cord_pair(self, cord1, cord2):

        """Returns True if cord1 and cord2 are valid coordinates in the form (x,y) on the board otherwise returns False"""

        if not (self.__is_valid_coordinate(cord1) and self.__is_valid_coordinate(cord2)):
            return False
        return True

    def __are_locs_adjacent(self, start_loc, end_loc):

        """Returns True if start_loc and end_loc are adjacent to each other on the board otherwise returns False.
        Diagonal locations are considered adjacent."""

        start_cord = start_loc.get_cords()
        end_cord = end_loc.get_cords()

        x_diff = abs(start_cord[0] - end_cord[0])
        y_diff = abs(start_cord[1] - end_cord[1])


        # if the absolute value of the two x cords and two y cords are different by 1, the locations are adjacent
        if x_diff <= 1 and y_diff <= 1:
            return True
```

```
    def __get_adjacent_locations(self, loc):

        """Returns a list of the GridLocation objects on the board that are adjacent to the loc GridLocation object passed in
as an argument"""

        cords = loc.get_cords()
        adjacent_lst = []

        # iterating over the 3x3 grid of locations surrounding the loc GridLocation object
        for i in range(-1, 2):
            for j in range(-1, 2):

                # adding (0,0) to the location's coordinates would just give the location itself so we skip it
                if (i, j) == (0, 0):
                    continue

                # the coordinates of the adjacent location by adding the two points together
                adjacent_cord = (cords[0] + i, cords[1] + j)


                if self.__is_valid_coordinate(adjacent_cord):
                    adjacent_lst.append(adjacent_cord)

        # converting the list of coordinates into a list of GridLocation objects
        return [self.__board[i[0]][i[1]] for i in adjacent_lst]

    def __check_normal_legal(self, start_loc, end_loc, player):

        """Returns True if a normal adjacent move from start_loc to end_loc (non-capturing move) is legal for the player
provided as an
        argument otherwise returns False

        ######################################################################
        CLASS A SKILL: Determining if a normal move to an adjacent location is legal
        ######################################################################

        """

        start_cord = start_loc.get_cords()
        end_cord = end_loc.get_cords()
```

```
        # early return conditions
        if start_loc.is_empty():
            return False

        if not self.__is_valid_cord_pair(start_cord, end_cord):
            return False

        if start_loc.get_piece_colour() != player.get_piece_colour(): # attempting to move opponent's piece
            return False

        # if the end location is adjacent to the start location and is empty, the move is legal
        if self.__are_locs_adjacent(start_loc, end_loc) and end_loc.is_empty():
            return True

        return False

    def __get_looped_track_loc_indexes(self, looped_track, loc):

        """Returns a list of indexes where loc is found in looped_track. looped_track is a LoopedTrack object which
        is an implementation of a circular list data structure."""

        ind_lst = []

        cords = loc.get_cords()

        # start at the beginning of the circular list
        looped_track.set_pointer(0, "right")

        for i in range(looped_track.get_length()):

            item = looped_track.get_next_right()

            if item.get_cords() == cords:
                ind_lst.append(i)

        return ind_lst

    def __either_locations_vacant(self, start_location, end_location):

        """Returns True if either start_location or end_location is vacant otherwise returns False"""
```

```
        if start_location.is_empty() or end_location.is_empty():
            return True
        return False

    def __check_capture_legal(self, start_loc, end_loc, player):

        """Returns True if a capture move from start_loc to end_loc is legal for the player object provided as an argument
otherwise returns False

        ######################################################################
        CLASS A SKILL: Determining if a capturing move is legal
        ######################################################################

        """

        start_cords = start_loc.get_cords()
        end_cords = end_loc.get_cords()

        # early return conditions
        if not self.__is_valid_cord_pair(start_cords, end_cords):
            return False

        if start_loc.get_piece_colour() != player.get_piece_colour(): # attempting to move opponent's piece
            return False

        if start_loc.get_piece_colour() == end_loc.get_piece_colour(): # attempting to capture own piece
            return False

        if self.__either_locations_vacant(start_loc, end_loc): # attempting to capture or capture with an empty location
            return False

        # get the LoopedTrack objects for the inner and outer tracks that both start_loc and end_loc are on
        looped_track_tuple = self.__get_common_tracks(start_loc.get_track(), end_loc.get_track())

        for looped_track in looped_track_tuple:

            # indexes where start_loc is found in looped_track
            starting_indexes = self.__get_looped_track_loc_indexes(looped_track, start_loc)

            for start_loc_index in starting_indexes:
```

```
                # Move object representing a legal capture if one is found iterating right through the LoopedTrack
                right_move = self.__search_direction_for_capture(start_loc, start_loc_index, looped_track, "right")

                # legal capture found by iterating right through the looped_track which matches the attempted capture
                if right_move and right_move.get_end_cords() == end_loc.get_cords():
                    return True

                # Move object representing a legal capture if one is found iterating left through the LoopedTrack
                left_move = self.__search_direction_for_capture(start_loc, start_loc_index, looped_track, "left")

                # legal capture found by iterating left through the looped_track which matches the attempted capture
                if left_move and left_move.get_end_cords() == end_loc.get_cords():
                    return True

        return False

    def is_legal_move(self, start_loc, end_loc, player, move_type):

        """Returns True if a move from start_loc to end_loc is legal for the player provided as an argument otherwise returns
False.
        This public method is used by the Game class to check if a move is legal."""

        if move_type == MultiClassBoardAttributes.NORMAL_MOVE_TYPE:
            return self.__check_normal_legal(start_loc, end_loc, player)

        elif move_type == MultiClassBoardAttributes.CAPTURE_MOVE_TYPE:
            return self.__check_capture_legal(start_loc, end_loc, player)

        return False

    def __loc_pieces_same_colour(self, loc1, loc2):

        """Returns True if the pieces at loc1 and loc2 are the same colour and are not the same location otherwise returns
False"""

        if (loc1.get_piece_colour() == loc2.get_piece_colour()) and (loc1.get_cords() != loc2.get_cords()):
            return True
        return False

    def __is_valid_capture(self, start_location, end_location, loop_count):
```

```
        """Returns True if a capture from start_location to end_location is valid with the assumption that no pieces block the
path between the two locations
        and the correct player is attempting to make the capture. Otherwise returns False"""

        if end_location.is_empty():
            return False

        # not capturing a piece of the same colour and at least one of the 4 board loops has been traversed
        if (end_location.get_piece_colour() != start_location.get_piece_colour()) and (loop_count > 0):
            return True

        return False

    def __check_direction_invalid(self, start_location, end_location, loop_count):

        """Returns False if a capture could still potentially be made in the direction moving to end_location otherwise
returns True.
        If the method returns True, the capture legality algorithm should continue iterating in this direction."""

        # one of your own pieces is blocking the path in the current direction of iteration through the LoopedTrack
        if self.__loc_pieces_same_colour(start_location, end_location):
            return True

        # a piece has been enountered during iteration through the LoopedTrack and none of the 4 board loops have been
traversed
        if loop_count == 0 and not end_location.is_empty():
            return True

        # you have returned to the starting location and all of the 4 board loops have been traversed
        if (loop_count == self.NUM_BOARD_LOOPS) and (start_location.get_cords() == end_location.get_cords()):
            return True

        return False

    def __search_direction_for_capture(self, start_location, start_loc_index, looped_track, direction):

        """Returns a Move object if a valid capture is found in the direction specificed by the direction argument, iterating
        through looped_track starting at the occurence of start_location in looped_track at start_loc_index. Otherwise it
returns False.
```

```
        If a valid capture cannot be made with adjacent locations, the next locations in each direction are checked until
either
        a valid capture is found or the direction being checked can no longer have a valid capture on it.

        ######################################################################
        CLASS A SKILL: Circular list data structure traversal
        ######################################################################

        """

        invalid = False
        loop_count = 0
        prev_loc = start_location

        # set the correct looped_track  pointer to the given index where start_location is found
        # and skip the first location because it is the same as start_location
        if direction == "right":
            looped_track.set_pointer(start_loc_index, "right")
            looped_track.get_next_right()

        elif direction == "left":
            looped_track.set_pointer(start_loc_index, "left")
            looped_track.get_next_left()

        while not invalid:

            # get the next location in the direction specified by direction
            if direction == "left":
                curr_loc = looped_track.get_next_left()
            elif direction == "right":
                curr_loc = looped_track.get_next_right()

            # increment loop_count if a board loop has been traversed to get from prev_loc to curr_loc
            if self.__board_loop_used(prev_loc, curr_loc):
                loop_count += 1

            # return a Move object representing the capture if a valid capture is found
            if self.__is_valid_capture(start_location, curr_loc, loop_count):
                return Move(start_location, curr_loc, MultiClassBoardAttributes.CAPTURE_MOVE_TYPE)

            # stop iterating if a blocking condition occurs that means a capture cannot be made in the current direction
```

```
            if self.__check_direction_invalid(start_location, curr_loc, loop_count):
                invalid = True

            prev_loc = curr_loc

        return False

    def __switch_piece_positions(self, start_loc, end_loc):

        """Switches the positions of the pieces at start_loc and end_loc in self.__board"""

        start_cords = start_loc.get_cords()
        end_cords = end_loc.get_cords()

        self.__board[end_cords[0]][end_cords[1]].set_piece(start_loc.get_piece())
        self.__board[start_cords[0]][start_cords[1]].set_piece(None)

    def __board_loop_used(self, prev_loc, curr_loc):

        """Returns True if a loop has been used to get from prev_loc to curr_loc. Otherwise it returns False"""

        prev_cords = prev_loc.get_cords()
        curr_cords = curr_loc.get_cords()

        # a change in x and y cords between adjacent elements in a LoopedTrack object mean a loop has been used
        if prev_cords[0] != curr_cords[0] and prev_cords[1] != curr_cords[1]:
            return True

        return False

    def __move_piece_with_undo_arg(self, move_obj, undo=False):

        """Makes the move specified by move_obj. If undo is True, the move is made in reverse."""

        # decrement the piece count of the player that has had a piece captured
        if move_obj.get_move_type() == MultiClassBoardAttributes.CAPTURE_MOVE_TYPE:
            self.__decrement_piece_count(move_obj.get_end_colour())


        # update the LoopedTrack objects to reflect the move
        self.__update_tracks_after_move(move_obj)
```

```
        # for an undo move, the start and end locations are switched
        if undo:
            self.__switch_piece_positions(move_obj.get_end_loc(), move_obj.get_start_loc())

        else:
            self.__switch_piece_positions(move_obj.get_start_loc(), move_obj.get_end_loc())

    def move_piece(self, move_obj):

            """Makes the move specified by move_obj. This public method is used by the Game class to make a move.
            This method calls the __move_piece_with_undo_arg method with undo=False because this argument is only True
            as a part of the undoing process and undoing is handled by the undo_move method.

            ######################################################################
            CLASS A SKILL: Moving a piece on the board and updating the LoopedTrack objects
            ######################################################################

            """

            self.__move_piece_with_undo_arg(move_obj, undo=False)

    def __update_tracks_after_move(self, move_obj, undo=False):

        """Updates the inner and outer tracks to reflect the move specified by move_obj. If undo is True, the move is made in
reverse."""

        # switch the positions of the pieces at the start and end locations in the LoopedTrack objects
        self.__inner_track.switch_piece_positions(move_obj.get_start_loc(), move_obj.get_end_loc())
        self.__outer_track.switch_piece_positions(move_obj.get_start_loc(), move_obj.get_end_loc())

        if move_obj.get_move_type() == MultiClassBoardAttributes.CAPTURE_MOVE_TYPE:

            if undo:
                # add the piece that was captured back to its starting location in the LoopedTrack objects
                self.__inner_track.update_piece(move_obj.get_end_cords(), move_obj.get_end_colour())
                self.__outer_track.update_piece(move_obj.get_end_cords(), move_obj.get_end_colour())

            else:
                # remove the piece used to capture from its starting location in the LoopedTrack objects
```

```
            self.__inner_track.remove_piece(move_obj.get_start_cords())
            self.__outer_track.remove_piece(move_obj.get_start_cords())

    def __decrement_piece_count(self, end_colour):

        """Decrements the piece count of the player that has had a piece captured"""

        if end_colour == MultiClassBoardAttributes.player_1_colour:
            self.__player_tuple[0].remove_piece()

        elif end_colour == MultiClassBoardAttributes.player_2_colour:
            self.__player_tuple[1].remove_piece()

    def undo_move(self, move_obj):

        """Undoes the move specified by move_obj by making the move in reverse

        ######################################################################
        CLASS A SKILL: Undoing a move
        ######################################################################

        """

        if move_obj.get_move_type() == MultiClassBoardAttributes.CAPTURE_MOVE_TYPE:

            # update the LoopedTrack objects to reflect the undo move
            self.__update_tracks_after_move(move_obj, undo=True)

            # update the board to reflect the undo move
            self.__spawn_piece(move_obj.get_start_colour(), move_obj.get_start_loc())
            self.__spawn_piece(move_obj.get_end_colour(), move_obj.get_end_loc())

        elif move_obj.get_move_type() == MultiClassBoardAttributes.NORMAL_MOVE_TYPE:
            self.__move_piece_with_undo_arg(move_obj, undo=True)

    def __spawn_piece(self, colour, loc):

        """spawn a piece on the board at loc with colour specified by colour. Only used by the undo_move method"""

        cords = loc.get_cords()
        piece = Piece(colour)
```

```
        self.__board[cords[0]][cords[1]].set_piece(piece)

    def __get_loc_legal_moves(self, loc, player):

        """Returns a list of legal moves that can be made from loc (GridLocation object) by player (Player object)"""

        legal_moves = []

        # get normal non-capturing legal moves
        for end_loc in self.__get_adjacent_locations(loc):
            if self.is_legal_move(loc, end_loc, player, MultiClassBoardAttributes.NORMAL_MOVE_TYPE):
                legal_moves.append(Move(loc, end_loc, MultiClassBoardAttributes.NORMAL_MOVE_TYPE))

        # get legal captures
        for row in self.__board:
            for end_loc in row:
                if end_loc.get_piece_colour() != player.get_piece_colour() and self.is_legal_move(loc, end_loc, player,
MultiClassBoardAttributes.CAPTURE_MOVE_TYPE):
                    legal_moves.append(Move(loc, end_loc, MultiClassBoardAttributes.CAPTURE_MOVE_TYPE))

        return legal_moves

    def get_player_legal_moves(self, player_colour):

        """Returns a list of legal moves that can be made by the player specified by player_colour"""

        legal_moves = []

        player = self.__player_colour_map[player_colour]

        for row in self.__board:
            for loc in row:

                # append legal moves if the location is occupied by a piece of the player's colour
                if loc.get_piece_colour() == player.get_piece_colour():
                    legal_moves += self.__get_loc_legal_moves(loc, player)

        return legal_moves

    def get_single_random_legal_move(self, player_colour):
```

```
        """Returns a single, random legal move (Move object) that can be made by the player specified by player_colour"""

        # returns a 6x6 2D array with the elements shuffled (randomised between rows and columns)
        shuffled_board = shuffle_2D_array(self.__board)

        for row in shuffled_board:
            for loc in row:
                if loc.get_piece_colour() == player_colour:
                    loc_legal_moves = self.__get_loc_legal_moves(loc, self.__player_colour_map[player_colour])

                    if len(loc_legal_moves) > 0:
                        move = random.choice(loc_legal_moves)

                        return move

    def get_loc_single_capture(self, start_loc):

        """returns a possible capture with the piece at start_loc if one is available otherwise returns None"""

        # get the LoopedTrack objects for the inner and outer tracks that start_loc is on
        track_tuple = self.__get_track_from_text(start_loc.get_track())

        for track in track_tuple:
            if track == None:
                continue

            # indexes where start_loc is found in track
            starting_indexes = self.__get_looped_track_loc_indexes(track, start_loc)

            for start_loc_index in starting_indexes:

                # Move object representing a legal capture if one is found iterating right through the LoopedTrack
                right_move = self.__search_direction_for_capture(start_loc, start_loc_index, track, "right")

                # not checking for the legality of a specific capture
                if right_move:
                    return right_move

                # Move object representing a legal capture if one is found iterating left through the LoopedTrack
                left_move = self.__search_direction_for_capture(start_loc, start_loc_index, track, "left")
```

```
                # not checking for the legality of a specific capture
                if left_move:
                    return left_move

        return None

    def __get_edge_locations(self):

        """Returns a list of 4 GridLocation objects that are on the edge of the board. This is not a constant
        because pieces at each edge location can change."""

        edge_locs = [
            self.__board[MultiClassBoardAttributes.MIN_ROW_INDEX][MultiClassBoardAttributes.MIN_ROW_INDEX],
            self.__board[MultiClassBoardAttributes.MIN_ROW_INDEX][MultiClassBoardAttributes.MAX_ROW_INDEX],
            self.__board[MultiClassBoardAttributes.MAX_ROW_INDEX][MultiClassBoardAttributes.MIN_ROW_INDEX],
            self.__board[MultiClassBoardAttributes.MAX_ROW_INDEX][MultiClassBoardAttributes.MAX_ROW_INDEX]
        ]

        return edge_locs

    def get_corner_move(self, start_loc):

        """Returns a move using a corner location to move out of the corner if one is available and legal otherwise returns
None"""

        edge_locs = self.__get_edge_locations()

        if start_loc not in edge_locs:
            return None

        return self.__get_adjacent_move(start_loc)

    def __get_adjacent_move(self, start_loc):

        """Returns a move using a location adjacent to start_loc if one is available otherwise returns None"""

        for end_loc in self.__get_adjacent_locations(start_loc):
            if end_loc.is_empty():
                return Move(start_loc, end_loc, MultiClassBoardAttributes.NORMAL_MOVE_TYPE)
```

```
        return None

    def get_random_normal_move(self, player_colour):

        """Returns a random normal adjacent move (non-capturing move) that can be made. Used by the Easy AI opponent."""

        return random.choice(self.get_player_legal_moves(player_colour))

    def get_piece_count(self, player_number):

        """Returns the number of pieces the player with the number specified by player_number has on the board"""

        return self.__player_tuple[player_number - 1].get_piece_count()

    def get_game_state_string(self):

        """Returns a string representation of the current game state. Pieces are represented
        by their colour and empty locations are represented by a pre-determined character. Pieces
        are separated by a pre-determined character. The first and last characters of the string are not
        separator characters.

        ######################################################################
        CLASS B SKILL: Serialising a board object's into a string for storage in a database
        ######################################################################

        """

        # convert the 2D array into a 1D array
        flat_board = twoD_to_oneD_array(self.__board)

        # list to store game state being saved
        game_state_lst = []

        for loc in flat_board:
            if loc.is_empty():
                game_state_lst.append(self.SAVED_GAME_STATE_EMPTY_CHAR)
            else:
                game_state_lst.append(loc.get_piece_colour())

            game_state_lst.append(self.SAVED_GAME_STATE_SEPARATOR)
```

```
        # remove the last separator character
        game_state_lst.pop()

        # convert the list into a string
        game_state_string = "".join(game_state_lst)

        return game_state_string
```

Database.py

```
import sqlite3
import hashlib
import os
from datetime import datetime
from MultiClassBoardAttributes import MultiClassBoardAttributes

class Database:

    """Class for interacting with the database. Used inside of the UI class.

    ######################################################################
    CLASS A SKILL: Cross-table parameterised SQL queries (see get_user_stats, load_saved_games and get_user_game_history
methods)
    CLASS A SKILL: Complex OOP model with encapsulation
    GOOD CODING STYLE: Appropriate indentation (style choice for SQL queries)
    ######################################################################

    """

    NUM_RANDOM_BYTES_HASH = 16
    NUM_HASH_ITERATIONS = 100000

    def __init__(self, db_name):

        # connect to the database
        self.__conn = sqlite3.connect(db_name)

        # cursor object used to execute SQL commands
```

```
        self.__cursor = self.__conn.cursor()

    def create_users_table(self):

        """Creates the Users table if it doesn't already exist."""

        self.__cursor.execute(

            """

            CREATE TABLE IF NOT EXISTS Users (
                user_id INTEGER,
                username TEXT,
                password TEXT,
                account_creation_date DATE,
                piece_colour TEXT,
                PRIMARY KEY (user_id)
            );

            """

        )

        self.__conn.commit()

    def create_saved_games_table(self):

        """Creates the SavedGames table if it doesn't already exist. Note that player 2's colour doesn't
        need to be stored because it can be inferred from player 1's colour (yellow if player 1 is green else
        green)"""

        self.__cursor.execute(

            """

            CREATE TABLE IF NOT EXISTS SavedGames (
                saved_game_id INTEGER,
                user_id INTEGER,
                date_saved DATE,
                game_state_string TEXT,
                opponent_name TEXT,
```

```
            player2_starts BOOLEAN,
            player1_num_pieces INTEGER,
            player2_num_pieces INTEGER,
            player1_colour TEXT,
            PRIMARY KEY (saved_game_id)
            FOREIGN KEY (user_id) REFERENCES users(user_id)
        );

        """

    )

    self.__conn.commit()

def create_game_history_table(self):

    """Creates the GameHistory table if it doesn't already exist."""

    self.__cursor.execute(

        """

        CREATE TABLE IF NOT EXISTS GameHistory (
            historical_game_id INTEGER,
            user_id INTEGER,
            opponent TEXT,
            game_date DATE,
            winner TEXT,
            PRIMARY KEY (historical_game_id)
            FOREIGN KEY (user_id) REFERENCES users(user_id)
        );

        """

    )

    self.__conn.commit()

def create_AI_game_stats_table(self):

    """Creates the AIGameStats table if it doesn't already exist."""
```

```
        self.__cursor.execute(

            """

            CREATE TABLE IF NOT EXISTS AIGameStats (
                AI_game_stat_id INTEGER,
                user_id INTEGER,
                AI_difficulty TEXT,
                win_count INTEGER,
                loss_count INTEGER,
                PRIMARY KEY (AI_game_stat_id)
            );

            """

        )

        self.__conn.commit()

    def __get_new_primary_key(self, table_name, primary_key_name):

        """Returns the next available primary key for a table."""

        # get the current highest primary key value
        self.__cursor.execute(f"SELECT MAX({primary_key_name}) FROM {table_name};")
        old_key = self.__cursor.fetchone()[0]

        # the first key in the table will be 1
        if old_key == None:
            return 1

        # otherwise, increment the old key by 1
        else:
            return old_key + 1

    def __get_user_id_from_username(self, username):

        """Returns the user_id of a user given their username."""

        self.__cursor.execute("SELECT user_id FROM Users WHERE username = ?;", (username,))
```

```
        return self.__cursor.fetchone()[0]

    def check_if_username_exists(self, username):

        """Returns True if a username exists in the database, otherwise returns False."""

        self.__cursor.execute("SELECT username FROM Users WHERE username = ?;", (username,))
        return self.__cursor.fetchone() != None

    def add_user(self, username, password, piece_colour):

        """Adds a user to the database. The password is hashed and salted before being stored."""

        # hash and salt the password
        salt = os.urandom(self.NUM_RANDOM_BYTES_HASH)
        hashed_password = hashlib.pbkdf2_hmac('sha512', password.encode(), salt, self.NUM_HASH_ITERATIONS)
        hashed_password = salt.hex() + hashed_password.hex()

        # get the current date
        account_creation_date = datetime.now().strftime("%Y-%m-%d")

        user_id = self.__get_new_primary_key("Users", "user_id")

        # add the user to the Users table
        self.__cursor.execute("INSERT INTO Users VALUES (?, ?, ?, ?, ?);", (user_id, username, hashed_password,
account_creation_date, piece_colour))

        # add a record for each AI difficulty to the AIGameStats table for the new user
        AI_game_stat_id = self.__get_new_primary_key("AIGameStats", "AI_game_stat_id")

        for difficulty in [MultiClassBoardAttributes.EASY_AI_NAME, MultiClassBoardAttributes.MEDIUM_AI_NAME,
MultiClassBoardAttributes.HARD_AI_NAME]:
            self.__cursor.execute("INSERT INTO AIGameStats VALUES (?, ?, ?, ?, ?);", (AI_game_stat_id, user_id, difficulty, 0,
0))
            AI_game_stat_id += 1

        self.__conn.commit()

    def check_login_credentials(self, username, password):

        """Returns True if the username and password match a record in the database, otherwise returns False.
```

```
        The password is hashed and salted before being compared to the stored password."""

        # get the stored password and salt for the username
        stored_password = self.__cursor.execute("SELECT password FROM Users WHERE username = ?;", (username,)).fetchone()

        if stored_password == None:
            return False

        stored_password = stored_password[0]

        # split the stored password into the salt and the hashed password
        stored_salt = stored_password[:32]
        stored_password = stored_password[32:]

        # hash and salt the password passed as an argument
        hashed_password = hashlib.pbkdf2_hmac('sha512', password.encode(), bytes.fromhex(stored_salt), 100000)
        hashed_password = hashed_password.hex()

        # compare the stored password and the hashed password
        return stored_password == hashed_password

    def get_piece_colour(self, username):

        """Returns the preferred piece colour of a user given their username."""

        self.__cursor.execute("SELECT piece_colour FROM Users WHERE username = ?;", (username,))
        return self.__cursor.fetchone()[0]

    def get_user_stats(self, username):

        """Returns a user's win_count and loss_count for each AI difficulty."""

        self.__cursor.execute(

                """

                SELECT AI_difficulty, win_count, loss_count FROM AIGameStats
                    INNER JOIN Users ON Users.user_id = AIGameStats.user_id
                WHERE Users.username = ?;

                """, (username,)
```

```
        )

        return self.__cursor.fetchall()

    def __increment_win_stat(self, username, ai_difficulty):

        """Increments the win_count for a user and AI difficulty."""

        self.__cursor.execute(
            f"""

            UPDATE AIGameStats
            SET win_count = win_count + 1
            WHERE AI_difficulty = ? AND user_id = ?;

            """, (ai_difficulty, self.__get_user_id_from_username(username))
        )

        self.__conn.commit()

    def __increment_loss_stat(self, username, ai_difficulty):

            """Increments the loss_count for a user and AI difficulty."""

            self.__cursor.execute(
                """

                UPDATE AIGameStats
                SET loss_count = loss_count + 1
                WHERE AI_difficulty = ? AND user_id = ?;

                """, (ai_difficulty, self.__get_user_id_from_username(username))
            )

            self.__conn.commit()

    def update_user_stats(self, username, human_won, ai_difficulty):

        """Updates a user's win_count and loss_count for a given AI difficulty depending on whether the user won or lost."""

        if human_won:
```

```
            self.__increment_win_stat(username, ai_difficulty)

        else:
            self.__increment_loss_stat(username, ai_difficulty)

    def delete_table(self, table_name):

        """Deletes a table from the database."""

        self.__cursor.execute(f"DROP TABLE {table_name};")
        self.__conn.commit()

    def save_game_state(self, username, game_state_string, opponent_name, player2_starts, player1_num_pieces,
player2_num_pieces, player1_colour):

        """Saves a match to the database (SavedGames table)."""

        # get the current date
        date_today = datetime.now().strftime("%Y-%m-%d")

        # get the next available primary key for the SavedGames table
        saved_game_id = self.__get_new_primary_key("SavedGames", "saved_game_id")

        # get the user_id of the user
        user_id = self.__get_user_id_from_username(username)

        # add the game state to the SavedGames table
        self.__cursor.execute("INSERT INTO SavedGames VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);", (saved_game_id, user_id,
date_today, game_state_string, opponent_name, player2_starts, player1_num_pieces, player2_num_pieces, player1_colour))
        self.__conn.commit()

    def load_saved_games(self, username):

        """Returns a list of saved games for a user."""

        self.__cursor.execute(
            """

            SELECT saved_game_id, date_saved, opponent_name FROM SavedGames
                INNER JOIN Users ON Users.user_id = SavedGames.user_id
            WHERE username = ?;
```

```
            """, (username,)
        )

        return self.__cursor.fetchall()

    def load_game_state(self, saved_game_id):

        """Returns the data stored for a saved game given its saved_game_id."""

        self.__cursor.execute(
            """

            SELECT username, game_state_string, opponent_name, player2_starts, player1_num_pieces, player2_num_pieces,
player1_colour FROM SavedGames
                INNER JOIN Users ON Users.user_id = SavedGames.user_id
            WHERE saved_game_id = ?;

            """, (saved_game_id,)
        )

        return self.__cursor.fetchone()

    def delete_saved_game(self, saved_game_id):

        """Deletes a saved game from the database given its saved_game_id."""

        self.__cursor.execute(
            """

            DELETE FROM SavedGames
            WHERE saved_game_id = ?;

            """, (saved_game_id,)

        )

        self.__conn.commit()

    def add_game_to_history(self, username, player2_name, winner_name):
```

```
        """Adds a game to the GameHistory table."""

        # get the current date
        game_date = datetime.now().strftime("%Y-%m-%d")

        # get the next available primary key for the GameHistory table
        historical_game_id = self.__get_new_primary_key("GameHistory", "historical_game_id")

        # get the user_id of the user
        user_id = self.__get_user_id_from_username(username)

        # add the game to the GameHistory table
        self.__cursor.execute("INSERT INTO GameHistory VALUES (?, ?, ?, ?, ?);", (historical_game_id, user_id, player2_name,
game_date, winner_name))
        self.__conn.commit()

    def get_user_game_history(self, username):

        """Returns a list of games from the GameHistory table for a user given their username."""

        self.__cursor.execute(
            """

            SELECT historical_game_id, game_date, opponent, winner FROM GameHistory
                INNER JOIN Users ON Users.user_id = GameHistory.user_id
            WHERE username = ?;

            """, (username,)
        )

        return self.__cursor.fetchall()


    def update_stored_piece_colour(self, username, new_colour):

        """Updates a user's piece colour in the Users table."""

        self.__cursor.execute(
            """

            UPDATE Users
```

```
            SET piece_colour = ?
            WHERE username = ?;

            """, (new_colour, username)
        )

        self.__conn.commit()
```

Game.py

```
from Player import Player, EasyAIPlayer, MediumAIPlayer, HardAIPlayer
from Board import Board
from MultiClassBoardAttributes import MultiClassBoardAttributes
from Move import Move
from Stack import Stack


class GameNotOverError(Exception):
    pass

class Game:

    """The Game class manages the game state. It contains the board, the players, and the move history stack.

    ######################################################################
    CLASS A SKILL: Stack data structure (see comments in make_and_return_move and undo_and_return_move methods)
    CLASS A SKILL: Complex OOP model with encapsulation and composition
    GOOD CODING STYLE: Use of constants
    ######################################################################

    """

    AI_NAME_TO_CLASS_MAP = {
        MultiClassBoardAttributes.EASY_AI_NAME: EasyAIPlayer,
        MultiClassBoardAttributes.MEDIUM_AI_NAME: MediumAIPlayer,
        MultiClassBoardAttributes.HARD_AI_NAME: HardAIPlayer,
    }
```

```
    def __init__(self, player1name, player2_name, ai_level=None, game_state_string=None, player2_starts=False,
player1_num_pieces=MultiClassBoardAttributes.NUM_STARTING_PIECES_EACH,
player2_num_pieces=MultiClassBoardAttributes.NUM_STARTING_PIECES_EACH):

        self.__player1 = Player(player1name, MultiClassBoardAttributes.player_1_colour, player1_num_pieces)

        # If ai_level is not None, player 2 is an AI player. Otherwise, player 2 is a human player.
        if ai_level:
            self.__player2 = self.__make_ai_player(player2_name, player2_num_pieces)
        else:
            self.__player2 = Player(player2_name, MultiClassBoardAttributes.player_2_colour, player2_num_pieces)

        self.__player_tuple = (self.__player1, self.__player2)

        self.__game_over = False

        self.__board = Board(self.__player1, self.__player2, game_state_string)

        self.__move_history_stack = Stack()

        self.__current_player = self.__player1
        self.__non_current_player = self.__player2

        if player2_starts: # if a game is loaded from a save it might be player 2's turn first
            self.switch_current_player()

    def __make_ai_player(self, ai_name, player2_num_pieces):

        """Returns an AI player object using the class specified by ai_name with a piece count of player2_num_pieces"""

        return self.AI_NAME_TO_CLASS_MAP[ai_name](MultiClassBoardAttributes.player_2_colour, player2_num_pieces)

    def get_ai_move(self):

        """Returns a Move object generated by the AI player."""

        move = self.__current_player.get_move(self.__board)

        return move

    def is_legal_move(self, start_loc, end_loc, move_type):
```

```
        return self.__board.is_legal_move(start_loc, end_loc, self.__current_player, move_type)

    def get_game_state_string(self):
        return self.__board.get_game_state_string()

    def set_game_status(self):

        """Sets self.__game_over to True if either player has no pieces left. A legal move can always
        be played in Surakarta, so this is the only way the game can end."""

        if (self.__player1.get_piece_count() == 0 or self.__player2.get_piece_count() == 0):
            self.__game_over = True

    def get_board_state(self):
        return self.__board.get_board_state()

    def is_game_over(self):
        return self.__game_over

    def get_winning_player(self):

        """Returns the Player object of the winner. If the game is not over, an exception is raised."""

        if self.is_game_over():
            if self.__player1.get_piece_count() > self.__player2.get_piece_count():
                return self.__player1

            elif self.__player2.get_piece_count() > self.__player1.get_piece_count():
                return self.__player2

        else:
            raise GameNotOverError("Attempting to get winner when game is not over.")

    def make_and_return_move(self, start_location, end_location, move_type):

        """Makes a move on the board and returns the Move object. The Move object is pushed onto the private
move_history_stack attribute.

        ######################################################################
        CLASS A SKILL: Stack data structure (pushing onto the stack in this method)
        ######################################################################
```

```
        """

        move_obj = Move(start_location, end_location, move_type)

        # push the move onto the move history stack
        self.__move_history_stack.push(move_obj)

        # make the move on the board
        self.__board.move_piece(move_obj)

        return move_obj

    def undo_and_return_move(self):

        """Calls a method in the Board class to undo the last move made on the board and returns the Move object.
        The Move object is popped off the move_history_stack.

        ######################################################################
        CLASS A SKILL: Stack data structure (popping off the stack in this method)
        CLASS A SKILL: Undoing a move and passing information to the Board class's undo_move method
        ######################################################################

        """

        if self.__move_history_stack.is_empty():
            return None

        # pop the last move off the move history stack
        move_obj = self.__move_history_stack.pop()

        # add a piece back to the current player's piece count if the move was a capture
        if move_obj.get_move_type() == "capture":
            self.__current_player.add_piece()

        # undo the move on the board
        self.__board.undo_move(move_obj)

        return move_obj

    def get_current_player_name(self):
```

```
        return self.__current_player.get_name()

    def get_current_player_colour(self):
        return self.__current_player.get_piece_colour()

    def switch_current_player(self):
        self.__current_player, self.__non_current_player = self.__non_current_player, self.__current_player

    def get_player_name(self, player_number):
        return self.__player_tuple[player_number - 1].get_name()

    def get_player_colour(self, player_number):
        return self.__player_tuple[player_number - 1].get_piece_colour()

    def get_player_piece_count(self, player_number):
        return self.__player_tuple[player_number - 1].get_piece_count()
```

GridLocation.py

```
from Piece import Piece
from MultiClassBoardAttributes import MultiClassBoardAttributes

class GridLocation:

    """represnts information about a location on the board such as
    the piece on it, its coordinates and which of the looped tracks it sits on

    ######################################################################
    CLASS A SKILL: Complex OOP model with encapsulation and composition
    GOOD CODING STYLE: Use of constants
    ######################################################################

    """

    # outer track cordinates have row (2 or 3) and column (2 or 3)
    OUTER_TRACK_NUMBERS = (2, 3)

    # inner track cordinates have row (1 or 4) and column (1 or 4)
    INNER_TRACK_NUMBERS = (1, 4)
```

```
    # player 1 pieces start on rows 4 and 5, player 2 pieces start on rows 0 and 1
    PLAYER_1_ROWS = (4, 5)
    PLAYER_2_ROWS = (0, 1)

    def __init__(self, cords):
        self.__cords = cords
        self.__track = self.__set_track()

        # initial piece set for a fresh board
        self.__piece = self.__set_initial_piece()

    def __set_track(self):

        """determines which track(s) a location sits on"""

        if (self.__cords[0] in self.OUTER_TRACK_NUMBERS and self.__cords[1] in self.INNER_TRACK_NUMBERS) or (self.__cords[0]
in self.INNER_TRACK_NUMBERS and self.__cords[1] in self.OUTER_TRACK_NUMBERS):
            return MultiClassBoardAttributes.BOTH_TRACK_STRING

        elif self.INNER_TRACK_NUMBERS[0] in self.__cords or self.INNER_TRACK_NUMBERS[1] in self.__cords:
            return MultiClassBoardAttributes.INNER_TRACK_STRING

        elif self.OUTER_TRACK_NUMBERS[0] in self.__cords or self.OUTER_TRACK_NUMBERS[1] in self.__cords:
            return MultiClassBoardAttributes.OUTER_TRACK_STRING

        else:
            return None

    def __set_initial_piece(self):

        """determines which piece should be placed on the location at the start of the game"""

        if self.__cords[0] in self.PLAYER_1_ROWS:
            return Piece(MultiClassBoardAttributes.player_1_colour)

        elif self.__cords[0] in self.PLAYER_2_ROWS:
            return Piece(MultiClassBoardAttributes.player_2_colour)

        else:
            return None
```

```
    def is_empty(self):
        return self.__piece == None

    def set_piece(self, piece):
        self.__piece = piece

    def get_piece(self):
        return self.__piece

    def get_piece_colour(self):
        if self.__piece == None:
            return None
        else:
            return self.__piece.get_piece_colour()

    def get_cords(self):
        return self.__cords

    def get_track(self):
        return self.__track
```

LoopedTrack.py

```
from Piece import Piece

class LoopedTrack:

    """An implementation of a circular list data structure used for the board's two looped tracks.
    LoopedTrack objects contain GridLocation objects as elements and can be traversed in either direction.

    ######################################################################
    CLASS A SKILL: Circular list data structure
    CLASS A SKILL: Complex OOP model with encapsulation and composition
    ######################################################################

    """

    def __init__(self, grid_location_lst, name):
```

```
        # underlying list to store the data
        self.__lst = grid_location_lst

        # name of the LoopedTrack
        self.__name = name

        # length of the LoopedTrack
        self.__length = len(self.__lst)

        # pointers to keep track of the current location in the LoopedTrack for right and left traversal
        self.__right_pointer = 0
        self.__left_pointer = 0

    def __str__(self):
        data = [(loc.get_cords(), loc.get_piece_colour()) for loc in self.__lst]
        return str(data)

    def get_length(self):
        return self.__length

    def get_name(self):
        return self.__name

    def set_pointer(self, index, pointer_type):

        """sets the specified pointer's location to the given index.
        Values returned by get_next_left() and get_next_right() will be affected by this change"""

        # making sure index is in the range of the list (negative indexes are allowed)
        if (index * -1) <= len(self.__lst) and index < len(self.__lst):
            if pointer_type == "left":
                self.__left_pointer = index
            elif pointer_type == "right":
                self.__right_pointer = index
        else:
            raise IndexError("Index out of range. Index must be less than or equal to the length of the list.")

    def get_next_right(self):

        """returns the next item in the circular list, starting from the right pointer."""
```

```
        item = self.__lst[self.__right_pointer]
        self.__right_pointer = (self.__right_pointer + 1) % len(self.__lst)
        return item

    def get_next_left(self):

        """returns the next item in the circular list, starting from the left pointer."""

        item = self.__lst[self.__left_pointer]
        self.__left_pointer = (self.__left_pointer - 1) % len(self.__lst)
        return item

    def __get_all_occurence_indexes(self, cords):

        """returns a list of all the indexes that a location with the specified cords is found in lst"""

        ind_lst = []

        for ind, grid_loc in enumerate(self.__lst):
            if grid_loc.get_cords() == cords:
                ind_lst.append(ind)

        return ind_lst

    def switch_piece_positions(self, loc1, loc2):

        """replaces all occurences of loc1's piece with loc2's piece and all occurences of loc2's piece with loc1's piece"""

        # get all the indexes of loc1 and loc2 in the LoopedTrack
        loc1_ind_lst = self.__get_all_occurence_indexes(loc1.get_cords())
        loc2_ind_lst = self.__get_all_occurence_indexes(loc2.get_cords())

        # replace all occurences of loc1's piece with loc2's piece and vice versa
        for i in loc1_ind_lst:
            self.__lst[i].set_piece(loc2.get_piece())

        for i in loc2_ind_lst:
            self.__lst[i].set_piece(loc1.get_piece())

    def remove_piece(self, cords):
```

```
        """replaces all occurences of a piece at cords with None"""

        ind_lst = self.__get_all_occurence_indexes(cords)

        for i in ind_lst:
            self.__lst[i].set_piece(None)

    def update_piece(self, cords, piece_colour):

        """replaces all occurences of the piece at the GridLocation specified by cords's piece with a piece of the specified
colour"""

        ind_lst = self.__get_all_occurence_indexes(cords)

        for i in ind_lst:
            if piece_colour == None:
                self.__lst[i].set_piece(None)
            else:
                self.__lst[i].set_piece(Piece(piece_colour))
```

Move.py

```
class Move:

    """Represents a move in the game. Stores the start and end locations of the move, the type of move
    and the colour of the start and end location pieces at the time of the move. Start and end location
    piece colours are stored to ensure the integrity of a move object if referenced at a later point in
    the game since pieces at a location can change

    ######################################################################
    CLASS A SKILL: Complex OOP model with encapsulation and composition
    ######################################################################

    """

    def __init__(self, start_loc, end_loc, move_type):
        self.__start_loc = start_loc
        self.__end_loc = end_loc
```

```
        self.__move_type = move_type
        self.__start_colour = self.__start_loc.get_piece_colour()
        self.__end_colour = self.__end_loc.get_piece_colour()

    def __str__(self):
        return f"{self.__move_type} from {self.__start_loc.get_cords()} to {self.__end_loc.get_cords()}"

    def get_start_loc(self):
        return self.__start_loc

    def get_end_loc(self):
        return self.__end_loc

    def get_start_cords(self):
        return self.__start_loc.get_cords()

    def get_end_cords(self):
        return self.__end_loc.get_cords()

    def get_move_type(self):
        return self.__move_type

    def get_start_colour(self):
        return self.__start_colour

    def get_end_colour(self):
        return self.__end_colour
```

MultiClassBoardAttributes.py

```
class MultiClassBoardAttributes:

    """This class contains attributes about the Surakarta board that need to be accessed by multiple classes.

    ######################################################################
    GOOD CODING STYLE: Use of constants
    ######################################################################

    """
```

```
    # can be changed using set_player_colour
    player_1_colour = "yellow"
    player_2_colour = "green"

    # unchanged constants
    DEFAULT_PLAYER_1_COLOUR = "yellow"
    DEFAULT_PLAYER_2_COLOUR = "green"

    NORMAL_MOVE_TYPE = "move"
    CAPTURE_MOVE_TYPE = "capture"

    INNER_TRACK_STRING = "INNER"
    OUTER_TRACK_STRING = "OUTER"
    BOTH_TRACK_STRING = "BOTH"

    EASY_AI_NAME = "Easy AI"
    MEDIUM_AI_NAME = "Medium AI"
    HARD_AI_NAME = "Hard AI"

    MIN_ROW_INDEX = 0
    MAX_ROW_INDEX = 5

    NUM_STARTING_PIECES_EACH = 12

    @staticmethod
    def set_player_colour(colour, player_num):

        """sets the colour of a player's pieces. Used by the GUI to change the colour of the pieces for a logged in user."""

        if player_num == 1:
            MultiClassBoardAttributes.player_1_colour = colour
        elif player_num == 2:
            MultiClassBoardAttributes.player_2_colour = colour
```

Piece.py

```
class Piece:
```

```
    """A class to represent a piece on the Surakarta board"""

    def __init__(self, colour):
        self.__colour = colour

    def __str__(self):
        return str(self.__colour)

    def get_piece_colour(self):
        return self.__colour
```

Player.py

```
from MultiClassBoardAttributes import MultiClassBoardAttributes
import random
from UtilityFunctions import shuffle_2D_array
from TreeSearch import GameTree

class Player:

    """Represents a player in the game. AI players inherit from this class and human players use this class directly"""

    def __init__(self, name, piece_colour, piece_count=MultiClassBoardAttributes.NUM_STARTING_PIECES_EACH):
        self.__name = name
        self.__piece_colour = piece_colour
        self.__piece_count = piece_count # a player may not have 12 pieces when the object is created if the game is being
loaded

    def get_piece_colour(self):
        return self.__piece_colour

    def get_name(self):
        return self.__name

    def get_piece_count(self):
        return self.__piece_count

    def remove_piece(self):
        """Removes a single piece from the player's piece count"""
```

```
        if self.__piece_count <= 0:
            raise ValueError("Player has no pieces left, cannot remove piece")

        self.__piece_count -= 1

    def set_piece_count(self, piece_count):
        self.__piece_count = piece_count

    def add_piece(self):

        """Adds a single piece to the player's piece count. Only used to return a piece to a player after a move is undone."""

        if self.__piece_count >= MultiClassBoardAttributes.NUM_STARTING_PIECES_EACH:
            raise ValueError("Player has too many pieces, cannot add piece")

        self.__piece_count += 1

class AIPlayer(Player):

    """An abstract base class for AI opponents. AI opponent classes inherit from this class and implement the get_move method

    ######################################################################
    CLASS A SKILL: Complex OOP model with encapsulation and inheritance
    ######################################################################

    """

    def __init__(self, name, piece_colour, piece_count=MultiClassBoardAttributes.NUM_STARTING_PIECES_EACH):
        super().__init__(name, piece_colour, piece_count)

    def get_move(self, board):

        """Must be implemented by subclasses. Returns a Move object for the AI player to make"""

        raise NotImplementedError("AI opponent classes must have a get_move method")


class EasyAIPlayer(AIPlayer):

    """An Easy AI opponent that inherits from the AIPlayer class and implements the get_move method with a greedy algorithm
```

```
    ######################################################################
    CLASS A SKILL: Complex OOP model with encapsulation, inheritance, and polymorphism
    ######################################################################

    """

    def __init__(self, piece_colour, piece_count):
        super().__init__(MultiClassBoardAttributes.EASY_AI_NAME, piece_colour, piece_count)

    def get_move(self, board):

        """Uses a greedy algorithm to make moves. It will capture if possible and otherwise will move pieces
        towards the corner if possible. Else it will make a random move

        ######################################################################
        CLASS A SKILL: Greedy algorithm for Easy AI
        ######################################################################

        """

        corner_move_lst = []
        shuffled_board = shuffle_2D_array(board.get_board_state())

        for row in shuffled_board:
            for loc in row:
                if (loc.get_piece_colour() == self.get_piece_colour()):
                    move = board.get_loc_single_capture(loc)

                    if move: # capture possible with piece at loc
                        return move

                    move = board.get_corner_move(loc)

                    if move: # corner move possible with piece at loc
                        corner_move_lst.append(move)

        if len(corner_move_lst) > 0: # checks if no captures are found
            return random.choice(corner_move_lst)

        return board.get_random_normal_move(self.get_piece_colour())
```

```
class MediumAIPlayer(AIPlayer):

    """A Medium AI opponent that inherits from the AIPlayer class and implements the get_move method
    with the Monte Carlo Tree Search algorithm running for 15 seconds per move.

    ######################################################################
    CLASS A SKILL: Complex OOP model with encapsulation, inheritance, and polymorphism
    ######################################################################

    """

    TIME_FOR_MOVE = 15 # seconds

    def __init__(self, piece_colour, piece_count):
        super().__init__(MultiClassBoardAttributes.MEDIUM_AI_NAME, piece_colour, piece_count)

    def get_move(self, board):

        """Uses the Monte Carlo Tree Search algorithm to make moves. The algorithm is run for 15 seconds per move"""

        game_tree = GameTree(board, self.TIME_FOR_MOVE)
        return game_tree.get_next_move()


class HardAIPlayer(AIPlayer):

    """A Hard AI opponent that inherits from the AIPlayer class and implements the get_move method
    with the Monte Carlo Tree Search algorithm running for 30 seconds per move.

    ######################################################################
    CLASS A SKILL: Complex OOP model with encapsulation, inheritance, and polymorphism
    ######################################################################

    """

    TIME_FOR_MOVE = 30 # seconds

    def __init__(self, piece_colour, piece_count):
        super().__init__(MultiClassBoardAttributes.HARD_AI_NAME, piece_colour, piece_count)
```

```
    def get_move(self, board):

        """Uses the Monte Carlo Tree Search algorithm to make moves. The algorithm is run for 30 seconds per move"""

        game_tree = GameTree(board, self.TIME_FOR_MOVE)
        return game_tree.get_next_move()
```

Stack.py

```
class Stack:

    """Implements a stack data structure

    ######################################################################
    CLASS A SKILL: Stack data structure
    ######################################################################

    """

    def __init__(self):
        self.__stack = []

    def __str__(self):
        return str(self.__stack)

    def push(self, item):
        self.__stack.append(item)

    def pop(self):
        return self.__stack.pop()

    def peek(self):
        return self.__stack[-1]

    def is_empty(self):
        return len(self.__stack) == 0
```

TreeSearch.py

```
import math
from MultiClassBoardAttributes import MultiClassBoardAttributes
import time
from copy import deepcopy

class Node:

    """Node class reprsenting a node in the game tree"""

    def __init__(self, board, depth, move_obj=None):

        # the board object of the node
        self.__board = board

        # the move that led to this node
        self.__move_obj = move_obj

        # UCB1 parameters
        self.__value = 0
        self.__visited_count = 0

        # the children of the node (list of Node objects)
        self.__children = []

        # the parent of the node (Node object)
        self.__parent = None

        # the depth of the node in the tree (root node has depth 0)
        self.__depth = depth

    def get_board(self):
        return self.__board

    def add_child(self, child):
        """adds a child to the node. child is a Node object."""

        self.__children.append(child)
```

```
            child.set_parent(self)

        def set_parent(self, parent):
            self.__parent = parent

        def get_parent(self):
            return self.__parent

        def get_depth(self):
            """returns the depth of the node in the tree"""

            return self.__depth

        def get_move_obj(self):
            return self.__move_obj

        def get_visited_count(self):
            return self.__visited_count

        def increment_visited_count(self):
            """increments the visited count of the node by 1. This is called when the node is visited during an MCTS
simulation."""

            self.__visited_count += 1

        def get_children(self):
            return self.__children

        def get_value(self):
            return self.__value

        def add_to_value(self, value):
            """updates the value of the node by adding the value passed in to the current value.
            This is called when the node is visited during an MCTS simulation."""

            self.__value += value

class GameTree:

    """GameTree class representing the game tree created by the MCTS algorithm
```

```
######################################################################
CLASS A SKILL: Monte Carlo Tree Search (MCTS)
CLASS A SKILL: Tree data structure and tree traversal
ClASS A SKILL: Complex OOP model with encapsulation and composition
GOOD CODING STYLE: Use of constants
######################################################################

"""

# rollout result constants
LOSS = -1
DRAW = 0
WIN = 1

# moves before early stop
MOVES_PER_ROLLOUT = 500

# exploration constant for the UCB1 formula
EXPLORATION_CONSTANT = 2

def __init__(self, root_board, time_for_move):

    self.__root = Node(root_board, depth=0)

    # the maximum depth of a node in the tree
    self.__current_tree_depth = 0

    # time allowed for the MCTS algorithm to run
    self.__time_for_move = time_for_move

    self.__current_node = self.__root

def __get_current_player_colour(self, depth):
    """returns the colour of the current player based on a depth in the tree"""

    # if the depth is even, it's player 2's (the AI) turn
    if depth % 2 == 0:
        return MultiClassBoardAttributes.player_2_colour

    # if the depth is odd, it's player 1's turn
    elif depth % 2 == 1:
```

```
            return MultiClassBoardAttributes.player_1_colour

    def __add_node(self, child_board, move_obj):
        """adds a node to the tree. child_board is the board state of the new node, and move_obj is the move that led to the node."""

        new_depth = self.__current_node.get_depth() + 1
        child = Node(child_board, new_depth, move_obj)

        self.__current_node.add_child(child)

    def __calc_UCB1(self, node):

        """returns the UCB1 value of a node used to determine which node to select next in the MCTS algorithm

        ######################################################################
        EXCELLENT CODING STYLE: Good exception handling
        ######################################################################

        """

        # using the UCB1 formula
        try:
            return (node.get_value() / node.get_visited_count()) + (GameTree.EXPLORATION_CONSTANT * math.sqrt(math.log(node.get_parent().get_visited_count()) / node.get_visited_count()))

        # if the node has not been visited yet (visited_count is 0), it's value is infinity
        except ZeroDivisionError:
            return math.inf

    def __check_terminal_board(self, board):

        """if the board is terminal, returns the result of the board (1 if the AI won, -1 if the AI lost). Otherwise, returns False."""

        if board.get_piece_count(1) == 0:
            return GameTree.WIN

        elif board.get_piece_count(2) == 0:
            return GameTree.LOSS
```

```
        return False

    def __get_early_stop_rollout_result(self, board):

        """returns the result of the board (1 for AI win, -1 for AI loss, 0 if neither player is winning according to the
evaluation function)
        if the rollout ends early because the maximum number of moves per rollout has been reached."""

        if board.get_piece_count(1) > board.get_piece_count(2):
            return GameTree.LOSS

        elif board.get_piece_count(1) < board.get_piece_count(2):
            return GameTree.WIN

        else:
            return GameTree.DRAW

    def __get_current_legal_moves(self):
        """returns the legal moves for the current node"""

        # board object of the current node
        board = self.__current_node.get_board()

        curr_depth = self.__current_node.get_depth()
        current_player_colour = self.__get_current_player_colour(curr_depth)

        # get the legal moves for the current player
        return board.get_player_legal_moves(current_player_colour)

    def __current_is_leaf(self):
        """returns True if the current node is a leaf node in the tree, otherwise returns False"""

        return len(self.__current_node.get_children()) == 0

    def __select_new_current(self):

        """selects a new current node to be the node with the highest UCB1 value among the current node's children"""

        # list of tuples of the form (node, UCB1 value) for each child of the current node
        ucb1_scores = [(node, self.__calc_UCB1(node)) for node in self.__current_node.get_children()]
```

```
        # set the current node to the child with the highest UCB1 value
        self.__current_node = max(ucb1_scores, key=lambda x: x[1])[0]

        # update the maximum depth of the tree if the current node's depth is greater than the current maximum depth
        if self.__current_tree_depth < self.__current_node.get_depth():
            self.__current_tree_depth = self.__current_node.get_depth()

    def __node_expansion(self):

        """expands the current node by adding all of its legal moves as children"""

        # list of legal moves for the current node
        legal_moves = self.__get_current_legal_moves()

        for move_obj in legal_moves:

            # make a deep copy of the current node's board and make the move on the copy (so that the original board is not
modified)
            board = deepcopy(self.__current_node.get_board())
            board.move_piece(move_obj)

            self.__add_node(board, move_obj)

    def __rollout(self):

        """performs a rollout from the current node to a terminal node or to the rollout depth and returns the result of the
rollout"""

        # deepcopy the current node's board so that the original board is not modified
        rollout_board = deepcopy(self.__current_node.get_board())

        num_moves = 0

        while num_moves < GameTree.MOVES_PER_ROLLOUT:

            # check if the board is terminal and if so return the result of the rollout
            terminal_board_result = self.__check_terminal_board(rollout_board)
            if terminal_board_result:
                return terminal_board_result

            # get the colour of the current player based on the depth of the current node
```

```
            current_depth = self.__current_node.get_depth() + num_moves
            rollout_colour = self.__get_current_player_colour(current_depth)

            simulated_move = rollout_board.get_single_random_legal_move(rollout_colour)
            rollout_board.move_piece(simulated_move)

            num_moves += 1

        # max rollout depth reached
        return self.__get_early_stop_rollout_result(rollout_board)

    def __backpropagate(self, result):

        """backpropagates the result of a rollout up the tree"""

        node = self.__current_node

        # terminate the loop when the root node has had its value and visited count updated
        while node != None:
            node.increment_visited_count()
            node.add_to_value(result)
            node = node.get_parent()

    def __run_MCTS_iteration(self):

        """runs one iteration of the MCTS algorithm from the current node with selection, expansion, rollout, and
backpropagation"""

        # 1. Selection
        while not self.__current_is_leaf():
            self.__select_new_current()

        if self.__current_node.get_visited_count() != 0:

            # 2. Expansion
            self.__node_expansion()
            if not self.__current_is_leaf(): # terminal nodes will not have children
                self.__current_node = self.__current_node.get_children()[0]

        # 3. Rollout/Simulation
        result = self.__rollout()
```

```
        # 4. Backpropagation
        self.__backpropagate(result)

        # reset the current node to the root node for the next iteration
        self.__current_node = self.__root

    def get_next_move(self):

        """Public method that runs the MCTS algorithm for a set amount of time and returns the best move to make"""

        start_time = time.time()

        # initial node expansion before the MCTS iterations begin
        self.__node_expansion()

        num_iterations = 0

        while time.time() - start_time < self.__time_for_move:
            self.__run_MCTS_iteration()
            num_iterations += 1

        # best node/move to make is the child of the root node with the highest UCB1 value
        best_node = max(self.__root.get_children(), key=lambda node: node.get_value())

        return best_node.get_move_obj()
```

UI.py

```
from Game import Game
from UtilityFunctions import oneD_to_twoD_array
import re
from MultiClassBoardAttributes import MultiClassBoardAttributes
import PySimpleGUI as sg
import textwrap
from PIL import ImageTk, Image
from Database import Database

class UI:
```

```
    """Abstract class for the UI. The play_game method is the method called to start the UI"""

    def __init__(self):
        self.__UI_type = None

        # a Game object
        self.__game = None

    def get_UI_type(self):
        raise NotImplementedError

    def play_game(self):
        raise NotImplementedError

class GraphicalUI(UI):

    """Graphical User Interface class for the program. Inherits from the UI class.

    ######################################################################
    CLASS A SKILL: Updating the UI based on user interaction such as moving pieces, navigating
    through the application's pages and displaying information such as stats, saved games and game history

    CLASS A SKILL: Complex OOP model with encapsulation, inheritance, polymorphism and composition

    GOOD CODING STYLE: Well-designed user interface

    GOOD CODING STYLE: Use of constants
    ######################################################################

    """

    LARGE_BUTTON_SIZE = 30
    FONT = "Helvetica"
    LARGE_BUTTON_FONT_PARAMS = (FONT, LARGE_BUTTON_SIZE)
    TITLE_FONT_SIZE = 60
    BUTTON_DIMENSIONS = (10, 1)
    COLUMN_PAD = 12
    LOGIN_PAD = 10
    PARAGRAPH_FONT_SIZE = 15
    HOME_PAGE_BUTTONS_PAD = (15, 10)
```

```
    BUTTON_FRAME_BORDER_WIDTH = 3
    SUBHEADING_FONT_PARAMS = (FONT, PARAGRAPH_FONT_SIZE, "bold")
    PARAGRAPH_FONT_PARAMS = (FONT, PARAGRAPH_FONT_SIZE)
    SMALL_BUTTON_FONT_PARAMS = (FONT, 15)
    SLIDER_SIZE = (40, 15)
    GAME_MODE_BUTTON_PAD = (100, 100)
    PIECE_BUTTON_PAD = (30, 30)
    LOAD_GAME_INPUT_PAD = (200, COLUMN_PAD)

    DISP_BOARD_WINDOW_SIZE = (500, 450)
    DISP_BOARD_IMAGE_SIZE = (400, 400)
    DISP_BOARD_IMAGE_CENTRE = (235, 215)
    DISP_BOARD_PIECE_SPACING = 43
    DISP_BOARD_INITAL_X = 128
    DISP_BOARD_INITAL_Y = 109
    DISP_BOARD_PIECE_RADIUS = 15

    LOGIN_WINDOW_HEIGHT = 270
    SIGNUP_WINDOW_HEIGHT = 350
    STATS_WINDOW_DIMENSIONS = (500, 500)
    CHANGE_PIECE_COLOUR_WINDOW_SIZE = (300, 150)
    SIGNUP_WINDOW_DIMENSIONS = (300, 350)
    LOGIN_WINDOW_DIMENSIONS = (300, 270)
    LOAD_GAME_WINDOW_DIMENSIONS = (500, 550)

    PLAYER_NAME_TEXTWRAP_LENGTH = 10
    HELP_PAGE_TEXTWRAP_LENGTH = 140

    AVAILABLE_PIECE_COLOURS = ["yellow", "green", "red", "lightblue", "orange", "black"]
    AI_RESERVED_NAMES = [MultiClassBoardAttributes.EASY_AI_NAME, MultiClassBoardAttributes.MEDIUM_AI_NAME,
MultiClassBoardAttributes.HARD_AI_NAME]

    PIECE_IMAGES_PATH = "Images/PieceImages/"
    BOARD_IMAGES_PATH = "Images/BoardImages/"

    with open("HelpPageText.txt", "r") as f:
        ABOUT_SURAKARTA_TEXT = textwrap.fill(f.readline(), HELP_PAGE_TEXTWRAP_LENGTH)
        RULES_TEXT = textwrap.fill(f.readline(), HELP_PAGE_TEXTWRAP_LENGTH)

    # display board background image
    DISP_BOARD_BACKGROUND_IMG = None
```

```
    """
    ######################################################################
    CLASS B SKILL: Dictionary
    ######################################################################
    """

    # maps difficulty level numbers to AI level names
    AI_NAME_TO_LEVEL_NUM_MAP = {
        1: MultiClassBoardAttributes.EASY_AI_NAME,
        2: MultiClassBoardAttributes.MEDIUM_AI_NAME,
        3: MultiClassBoardAttributes.HARD_AI_NAME
    }

    # maps AI level names to difficulty level numbers
    AI_LEVEL_NUM_TO_NAME_MAP = {
        MultiClassBoardAttributes.EASY_AI_NAME: 1,
        MultiClassBoardAttributes.MEDIUM_AI_NAME: 2,
        MultiClassBoardAttributes.HARD_AI_NAME: 3
    }

    def __init__(self):
        super().__init__()

        self.__UI_type = "GRAPHICAL"

        sg.theme('Dark')

        self.__highlighted_board_positions = [] # stores board positions that the user has clicked on

        # windows
        self.__main_window = None
        self.__display_board_window = None
        self.__login_window = None
        self.__signup_window = None
        self.__load_game_window = None
        self.__change_piece_colour_window = None

        # login status variables
        self.__logged_in = False
        self.__logged_in_username = None
```

```
        # game variables
        self.__game = None
        self.__game_is_loaded = False
        self.__ai_mode = False
        self.__ai_name = None

        # database object
        self.__db = Database("Database.db")
        self.__saved_games = None # stored to enable deletion of saved games

        self.__current_page = None
        self.__setup_home_page()

    def __create_window(self, title, layout, justification, maximise=True, size=(700, 700), modal=False, disable_close=False,
keep_on_top=False):

        """Creates a window with the given parameters"""

        window = sg.Window(
            title=title,
            layout=layout,
            size=size,
            resizable=False,
            keep_on_top=keep_on_top,
            modal=modal,
            disable_close=disable_close,
            element_justification=justification,
            text_justification=justification
        ).finalize()

        if maximise:
            window.maximize()

        return window

    def get_UI_type(self):
        return self.__UI_type

    def __setup_home_page(self):
```

```
        """Creates the home page window. This is the first window that is displayed when the program is run."""

        new_game_button = sg.Button("New Game", pad=self.HOME_PAGE_BUTTONS_PAD, font=self.LARGE_BUTTON_FONT_PARAMS,
size=self.BUTTON_DIMENSIONS, key="new_game_button")
        load_game_button = sg.Button("Load Game", pad=self.HOME_PAGE_BUTTONS_PAD, font=self.LARGE_BUTTON_FONT_PARAMS,
size=self.BUTTON_DIMENSIONS, key="load_game_button")
        show_stats_button = sg.Button("Show Stats", pad=self.HOME_PAGE_BUTTONS_PAD, font=self.LARGE_BUTTON_FONT_PARAMS,
size=self.BUTTON_DIMENSIONS, key="show_stats_button")
        login_button = sg.Button("Login", pad=self.HOME_PAGE_BUTTONS_PAD, font=self.LARGE_BUTTON_FONT_PARAMS,
size=self.BUTTON_DIMENSIONS, key="login_button")
        signup_button = sg.Button("Sign Up", pad=self.HOME_PAGE_BUTTONS_PAD, font=self.LARGE_BUTTON_FONT_PARAMS,
size=self.BUTTON_DIMENSIONS, key="signup_button")
        help_button = sg.Button("Help", pad=self.HOME_PAGE_BUTTONS_PAD, font=self.LARGE_BUTTON_FONT_PARAMS,
size=self.BUTTON_DIMENSIONS, key="help_button")

        buttons_layout = [
            [new_game_button, login_button],
            [load_game_button, signup_button],
            [show_stats_button, help_button],
        ]

        # surround buttons with a frame
        buttons_frame = sg.Frame(title="", layout=buttons_layout, border_width=self.BUTTON_FRAME_BORDER_WIDTH, pad=(0,
self.COLUMN_PAD))

        layout = [
            [self.__create_menu()],
            [sg.Text("Surakarta", pad=(0, self.COLUMN_PAD), font=(self.FONT, self.TITLE_FONT_SIZE))],
            [buttons_frame]
        ]

        if self.__main_window:
            self.__main_window.close()

        # in case the user was just in a match and exited to the home page
        self.__reset_game_variables()

        self.__current_page = "home_page"
        self.__main_window = self.__create_window("Surakarta", layout, "center")

    def __create_menu(self):
```

```
        """Creates the menu bar that is displayed at the top of every main non-modal window"""

        menu_layout = [
            ["Utilities", ["Home", "Show Login Status", "Logout", "Change Piece Colour", "Quit"]],
            ["Match Options", ["Restart Match", "Save Game"]]
        ]

        return sg.Menu(menu_layout, pad=(0, self.COLUMN_PAD))

    def __make_change_piece_colour_window(self):

        """Creates the change piece colour window which lets a logged in user change their piece colour"""

        layout = [
            [sg.Text("piece colour", pad=(0, self.LOGIN_PAD), font=self.PARAGRAPH_FONT_PARAMS)],
            [sg.Combo(self.AVAILABLE_PIECE_COLOURS, font=self.PARAGRAPH_FONT_PARAMS, expand_x=True, enable_events=True,
readonly=True, key="piece_colour_choice")],
            [sg.Button("Submit", pad=(0, self.COLUMN_PAD), font=self.SMALL_BUTTON_FONT_PARAMS, size=self.BUTTON_DIMENSIONS,
key="submit_change_piece_colour_button")]
        ]

        return self.__create_window("Change Piece Colour", layout, "center", modal=True, keep_on_top=True,
size=self.CHANGE_PIECE_COLOUR_WINDOW_SIZE, maximise=False, disable_close=False)

    def __make_login_or_signup_window(self, login_or_signup):

        """if login_or_signup is "login", creates the login window. If login_or_signup is "signup", creates the signup
window"""

        if login_or_signup not in ["login", "signup"]:
            raise ValueError("login_or_signup parameter of the __make_login_or_signup_window method must be either 'login' or
'signup'")

        drop_down_menu_layout = []

        if login_or_signup == "signup":

            # add piece colour dropdown menu to signup window
            drop_down_menu_layout = [
                [sg.Text("piece colour", pad=(0, self.LOGIN_PAD), font=self.PARAGRAPH_FONT_PARAMS)],
```

```
                [sg.Combo(self.AVAILABLE_PIECE_COLOURS, font=self.PARAGRAPH_FONT_PARAMS, expand_x=True, enable_events=True,
readonly=True, key="piece_colour_choice")]
            ]

            # signup window has a greater height than the login window
            modal_dimensions = self.SIGNUP_WINDOW_DIMENSIONS

        elif login_or_signup == "login":
            modal_dimensions = self.LOGIN_WINDOW_DIMENSIONS

        layout = [
            [sg.Text("Username", pad=(0, self.LOGIN_PAD), font=self.PARAGRAPH_FONT_PARAMS)],
            [sg.InputText("", pad=(0, self.LOGIN_PAD), key=f"{login_or_signup}_username_input",
font=self.PARAGRAPH_FONT_PARAMS)],
            [sg.Text("Password", pad=(0, self.LOGIN_PAD), font=self.PARAGRAPH_FONT_PARAMS)],
            [sg.InputText("", pad=(0, self.LOGIN_PAD), key=f"{login_or_signup}_password_input", password_char="*",
font=self.PARAGRAPH_FONT_PARAMS)],
            [drop_down_menu_layout],
            [sg.Button("Submit", pad=(0, self.COLUMN_PAD), font=self.SMALL_BUTTON_FONT_PARAMS, size=self.BUTTON_DIMENSIONS,
key=f"{login_or_signup}_submit_button")]
        ]

        return self.__create_window(login_or_signup.title(), layout, "center", modal=True, keep_on_top=True,
size=modal_dimensions, maximise=False, disable_close=False)

    def __get_new_game_text_and_input_layout(self, disp_text, inp_default_text, inp_key):

        """Returns a layout containing a text element and an input element"""

        layout = [
            [sg.Text(disp_text, pad=(0, self.COLUMN_PAD), font=self.SUBHEADING_FONT_PARAMS)],
            [sg.InputText(inp_default_text, pad=(0, self.COLUMN_PAD), key=inp_key, font=self.PARAGRAPH_FONT_PARAMS)],
        ]

        return layout

    def __setup_new_game_page(self):

        """Creates the new game page window where a user can choose to play a local or AI game, enter the names of the
player(s) and choose the AI difficulty (if playing an AI game)"""
```

```
        # player 1's name is the logged in user's name if they are logged in, otherwise it is entered by the user
        player_1_input_visible = not self.__logged_in

        # player 1 name input layout for AI play
        player_1_input_AI_layout = self.__get_new_game_text_and_input_layout("Player 1 Name", "Player 1", "player_1_AI_input")

        # adding the player 1 name input layout to a column for formatting purposes
        player_1_AI_input_col = sg.Column(player_1_input_AI_layout, visible=player_1_input_visible)

        AI_input_layout = [
            [sg.Text("Difficulty", pad=(0, self.COLUMN_PAD), font=self.SUBHEADING_FONT_PARAMS)],
            [sg.Slider(range=(1, 3), default_value=1, orientation="h", size=self.SLIDER_SIZE, pad=(0, self.COLUMN_PAD),
key="difficulty_slider")],
            [player_1_AI_input_col],
        ]

        # main column for the AI play inputs
        AI_input_col = sg.Column(key="AI_play_inputs", layout=AI_input_layout, pad=(0, self.COLUMN_PAD), visible=False)

        # player 1 name input layout for local play
        player_1_input_local_layout = self.__get_new_game_text_and_input_layout("Player 1 Name", "Player 1",
"player_1_local_input")

        # adding the player 1 name input layout to a column for formatting purposes
        player_1_local_input_col = sg.Column(player_1_input_local_layout, visible=player_1_input_visible)

        Local_input_layout = [
            [player_1_local_input_col],
            [sg.Text("Player 2 Name", pad=(0, self.COLUMN_PAD), font=self.SUBHEADING_FONT_PARAMS)],
            [sg.InputText("Player 2", pad=(0, self.COLUMN_PAD), key="player_2_local_input", font=self.PARAGRAPH_FONT_PARAMS)],
        ]

        # main column for the local play inputs
        Local_input_col = sg.Column(key="local_play_inputs", layout=Local_input_layout, pad=(0, self.COLUMN_PAD),
visible=False)

        # main page layout
        layout = [
            [self.__create_menu()],
```

```
            [sg.Button("Local Play", font=self.LARGE_BUTTON_FONT_PARAMS, pad=self.GAME_MODE_BUTTON_PAD,
size=self.BUTTON_DIMENSIONS, key="local_play_button"), sg.Button("AI Play", font=self.LARGE_BUTTON_FONT_PARAMS,
pad=self.GAME_MODE_BUTTON_PAD, size=self.BUTTON_DIMENSIONS, key="AI_play_button")],
            [AI_input_col, Local_input_col],
            [sg.Button("Submit", font=self.LARGE_BUTTON_FONT_PARAMS, size=self.BUTTON_DIMENSIONS,
key="submit_local_play_button", visible=False), sg.Button("Submit", font=self.LARGE_BUTTON_FONT_PARAMS,
size=self.BUTTON_DIMENSIONS, key="submit_AI_play_button", visible=False)],
        ]

        self.__main_window.close() # close the previous window (home page)
        self.__current_page = "new_game_page"
        self.__main_window = self.__create_window("New Game", layout, "center")

    def __setup_help_page(self):

        """Creates the help page window where the rules of the game are displayed"""

        text_layout = [
            [sg.Text("What is Surakarta?", pad=(0, self.COLUMN_PAD), font=self.SUBHEADING_FONT_PARAMS)],
            [sg.Text(self.ABOUT_SURAKARTA_TEXT, pad=(0, self.COLUMN_PAD), font=self.PARAGRAPH_FONT_PARAMS)],
            [sg.Text("Rules", pad=(0, self.COLUMN_PAD), font=self.SUBHEADING_FONT_PARAMS)],
            [sg.Text(self.RULES_TEXT, pad=(0, self.COLUMN_PAD), font=self.PARAGRAPH_FONT_PARAMS)],
        ]

        layout = [
            [self.__create_menu()],
            [text_layout],
            [sg.Image(f"{self.BOARD_IMAGES_PATH}starting_board.png", expand_x=True, expand_y=True)],
        ]

        self.__main_window.close() # close the previous window (home page)
        self.__current_page = "help_page"
        self.__main_window = self.__create_window("Help Page", layout, "center")

    def __toggle_new_game_input_visibility(self, key_to_make_visible):

        """Toggles the visibility of the AI play inputs and the local play inputs. When the AI play inputs are visible, the
local play inputs are not and vice versa"""

        # these two lists contain the keys of the elements that need to be toggled
        local_play_element_keys = ["local_play_inputs", "submit_local_play_button"]
```

```
        AI_play_element_keys = ["AI_play_inputs", "submit_AI_play_button"]

        if key_to_make_visible == "AI_play_inputs":
            AI_visible = True
            local_visible = False

        elif key_to_make_visible == "local_play_inputs":
            AI_visible = False
            local_visible = True

        else:
            raise ValueError("key_to_make_visible parameter of the __toggle_new_game_input_visibility method must be either 'AI_play_inputs' or 'local_play_inputs'")


        # make the elements visible or invisible
        for key in local_play_element_keys:
            self.__main_window[key].update(visible=local_visible)

        for key in AI_play_element_keys:
            self.__main_window[key].update(visible=AI_visible)

    def __make_piece_button(self, piece_type, key, visible=False):

        """Creates a button used to represent a board piece with the given piece type and key. The button is invisible by default"""

        return sg.Button("", image_filename=f"{self.PIECE_IMAGES_PATH}{piece_type}_counter.png", pad=self.PIECE_BUTTON_PAD, visible=visible, key=key, button_color=(sg.theme_background_color(), sg.theme_background_color()), border_width=0)

    def __create_table(self, database_table_data, headers, key):

        """Creates a table with the given rows and headers"""

        rows = [[element for element in row] for row in database_table_data]

        return sg.Table(rows, headings=headers, expand_x=True, background_color="light gray", text_color="black", key=key)

    @staticmethod
    def __create_circle(canvas, x, y, radius, fill):
```

```
        """Creates a circle on the given Tkinter canvas with the given radius, fill colour and centre coordinates.
        Method credit to https://stackoverflow.com/questions/17985216/draw-circle-in-tkinter-python"""

        # oval coordinates are given as the top left and bottom right coordinates of the bounding box
        x0 = x - radius
        y0 = y - radius
        x1 = x + radius
        y1 = y + radius

        return canvas.create_oval(x0, y0, x1, y1, fill=fill, outline="black", tags="counter")

    def __draw_pieces_on_disp_board(self):

        """Draws the pieces on the display board window"""

        # underlying Tkinter canvas
        canvas = self.__display_board_window['-CANVAS-'].TKCanvas

        # delete all existing drawn pieces
        canvas.delete("counter")

        # 6x6 2D array of the board state where each element is a string representing the colour of the piece
        display_board = [[i.get_piece_colour() for i in row] for row in self.__game.get_board_state()]

        for i, row in enumerate(display_board):
            for j, counter in enumerate(row):

                if counter == None: # don't draw a piece if there is no piece at that location
                    continue

                else:
                    # new centre cords can be calculated from the piece at (0,0)'s cords and the spacing between pieces as the
board is square shaped
                    centre_x = self.DISP_BOARD_INITAL_X + (self.DISP_BOARD_PIECE_SPACING * j)
                    centre_y = self.DISP_BOARD_INITAL_Y + (self.DISP_BOARD_PIECE_SPACING * i)
                    self.__create_circle(canvas, centre_x, centre_y, self.DISP_BOARD_PIECE_RADIUS, counter)

    def __make_display_board_window(self):

        """Creates the display board window where the user can view the board state of the current game including the looped
tracks around the board". The canvas is initially empty"""
```

```
        # prevents the display board window from being opened multiple times (making the window modal would prevent the user
from moving pieces while the display board is open)
        if self.__display_board_window:
            self.__display_board_window.close()

        layout = [
            [sg.Canvas(size=self.DISP_BOARD_WINDOW_SIZE, key='-CANVAS-')],
        ]

        display_board_window = self.__create_window("Display Board", layout, "center", size=self.DISP_BOARD_WINDOW_SIZE,
maximise=False, modal=False, disable_close=False, keep_on_top=True)

        return display_board_window

    def __make_load_game_window(self):

        """Creates the load game window where the user can load a saved game or delete a saved game. A table is displayed
showing the user's saved games."""

        if not self.__logged_in:
            sg.popup("You must be logged in to load a game", title="Error Loading Game", keep_on_top=True)
            return

        # request the user's saved games from the database
        self.__saved_games = self.__db.load_saved_games(self.__logged_in_username)
        saved_games_table_headers = ["Game ID", "Date", "Opponent"]

        saved_games_table = self.__create_table(self.__saved_games, saved_games_table_headers, "saved_games_table")

        layout = [
            [sg.Text("Enter a Game ID to Load", pad=(0, self.COLUMN_PAD), font=self.SUBHEADING_FONT_PARAMS)],
            [sg.Input("", pad=self.LOAD_GAME_INPUT_PAD, key="loading_game_id_input", font=self.PARAGRAPH_FONT_PARAMS,
justification="center")],
            [sg.Button("Load", pad=(0, self.COLUMN_PAD), font=self.SMALL_BUTTON_FONT_PARAMS, size=self.BUTTON_DIMENSIONS,
key="submit_loading_game_id_button")],
            [saved_games_table],
            [sg.Text("Enter a Game ID to Delete", pad=(0, self.COLUMN_PAD), font=self.SUBHEADING_FONT_PARAMS)],
            [sg.Input("", pad=self.LOAD_GAME_INPUT_PAD, key="deleting_game_id_input", font=self.PARAGRAPH_FONT_PARAMS,
justification="center")],
```

```
            [sg.Button("Delete", pad=(0, self.COLUMN_PAD), font=self.SMALL_BUTTON_FONT_PARAMS, size=self.BUTTON_DIMENSIONS,
key="submit_deleting_game_id_button")],
        ]

        load_game_window = self.__create_window("Load Game", layout, "center", size=self.LOAD_GAME_WINDOW_DIMENSIONS,
maximise=False, modal=True, disable_close=False, keep_on_top=True)

        return load_game_window

    def __make_stats_window(self):

        """Creates the stats window where the user can view their match stats and game history"""

        # request the user's match stats and game history from the database
        ai_match_stats = self.__db.get_user_stats(self.__logged_in_username)
        ai_stats_table_headers = ["AI Difficulty", "Wins", "Losses"]

        ai_stats_table = self.__create_table(ai_match_stats, ai_stats_table_headers, "ai_stats_table")

        # request the user's game history from the database
        game_history = self.__db.get_user_game_history(self.__logged_in_username)
        game_history_table_headers = ["Game Number", "Date", "Opponent", "Winner"]

        game_history_table = self.__create_table(game_history, game_history_table_headers, "game_history_table")

        layout = [
            [sg.Text("AI Match Stats", pad=(0, self.COLUMN_PAD), font=self.SUBHEADING_FONT_PARAMS)],
            [ai_stats_table],
            [sg.Text("Game History", pad=(0, self.COLUMN_PAD), font=self.SUBHEADING_FONT_PARAMS)],
            [game_history_table],
        ]

        stats_window = self.__create_window("Stats", layout, "center", size=self.STATS_WINDOW_DIMENSIONS, maximise=False,
modal=False, disable_close=False, keep_on_top=True)

        return stats_window

    def __setup_match_page(self, player1name, player2_name, ai_level=None, game_state_string=None, player2_starts=False,
player1_num_pieces=MultiClassBoardAttributes.NUM_STARTING_PIECES_EACH,
player2_num_pieces=MultiClassBoardAttributes.NUM_STARTING_PIECES_EACH):
```

```
        """Creates the match page window where the user can play a game of Surakarta. The board state is displayed and the
user can move pieces by clicking on the board"""

        # create the Game object
        self.__create_game_object(player1name, player2_name, ai_level, game_state_string, player2_starts, player1_num_pieces,
player2_num_pieces)

        if game_state_string:
            self.__game_is_loaded = True

        # 6x6 2D array of the board state where each element is a string representing the colour of the piece
        board_colours = [[i.get_piece_colour() for i in row] for row in self.__game.get_board_state()]

        # layout to show the board state
        board_layout = []

        for i, row in enumerate(board_colours):
            for j, colour in enumerate(row):

                key = f"{i},{j}" # unique key for each piece button

                if colour == None: # blank location (shown with a small white dot)
                    button = self.__make_piece_button("blank", key, visible=True)

                else:
                    button = self.__make_piece_button(colour, key, visible=True)

                board_layout.append(button)

        # make the board_layout the same shape as the actual board (6x6 2D array)
        board_layout = oneD_to_twoD_array(board_layout, len(board_colours))

        # text to show whose turn it is
        player_turn_layout = [
            [sg.Text(f"{self.__game.get_player_name(1)}'s Turn ({self.__game.get_player_colour(1)})", key="player1_turn_text",
pad=(0, self.COLUMN_PAD), font=self.SUBHEADING_FONT_PARAMS)],
        ]

        # current player turn in a frame for formatting purposes
        player_turn_frame = sg.Frame("", layout=player_turn_layout, border_width=0)
```

```
        # radio buttons to select the move type (capture or normal move)
        move_option = sg.Radio("Move", key="move_type_radio_move", group_id="move_type_radio",
font=self.SUBHEADING_FONT_PARAMS)
        capture_option = sg.Radio("Capture", key="move_type_radio_capture", group_id="move_type_radio",
font=self.SUBHEADING_FONT_PARAMS)

        submit_move_button = sg.Button("Submit Move", font=self.SMALL_BUTTON_FONT_PARAMS, key="submit_move_button")
        undo_move_button = sg.Button("Undo Move", font=self.SMALL_BUTTON_FONT_PARAMS, key="undo_move_button")

        # button to show the display board window
        show_display_board_button = sg.Button("show board", key="show_board_button", font=self.SMALL_BUTTON_FONT_PARAMS)

        # text to show the number of pieces captured by each player
        pieces_captured_player1_text = self.__get_pieces_captured_display_text(1)
        pieces_captured_player2_text = self.__get_pieces_captured_display_text(2)

        # layout to show the number of pieces captured by each player
        player1_captured_layout = [[sg.Text(pieces_captured_player1_text, key="player1_captured_text",
font=self.PARAGRAPH_FONT_PARAMS)]]
        player2_captured_layout = [[sg.Text(pieces_captured_player2_text, key="player2_captured_text",
font=self.PARAGRAPH_FONT_PARAMS)]]

        # main layout for the match page
        layout = [
            [self.__create_menu()],
            [player_turn_frame],
            [show_display_board_button],
            [undo_move_button, move_option, capture_option, submit_move_button],
            [sg.Column(player1_captured_layout), sg.Column(board_layout), sg.Column(player2_captured_layout)],
        ]

        self.__main_window.close() # close the previous window (new game page)
        self.__current_page = "match_page"
        self.__main_window = self.__create_window("Match", layout, "center")

        if player2_starts: # player 2 can start if the game is being loaded (i.e. not a new game)
            self.__update_current_player_display(self.__game_is_loaded)
            self.__update_number_captured_pieces_display()

    def __update_number_captured_pieces_display(self):
```

```
        """Updates the text showing the number of pieces captured by each player on the match page"""

        self.__main_window["player1_captured_text"].update(self.__get_pieces_captured_display_text(1))
        self.__main_window["player2_captured_text"].update(self.__get_pieces_captured_display_text(2))

    def __get_pieces_captured_display_text(self, player_number):

            """Returns a string containing the number of pieces captured by the given player"""

            if player_number == 1:
                other_player_num = 2

            elif player_number == 2:
                other_player_num = 1

            else:
                raise ValueError("player_number parameter of the __get_pieces_captured_display_text method must be either 1 or
2")

            # calculate the number of pieces captured by the player (starting pieces - pieces left for the other player)
            player_num_captured = MultiClassBoardAttributes.NUM_STARTING_PIECES_EACH -
self.__game.get_player_piece_count(other_player_num)

            # textwrap the player names to prevent the text from being too long and pushing the board off the screen
            player_name = self.__game.get_player_name(player_number)
            padded_player_name = self.__pad_player_name(player_name)
            text_wrapped_player_name = textwrap.fill(padded_player_name, self.PLAYER_NAME_TEXTWRAP_LENGTH)

            return f"{text_wrapped_player_name} captured pieces: {player_num_captured}"

    def __pad_player_name(self, player_name):

        """Pads the given player_name with spaces before the name if it is shorter than 10 characters
        until it is 10 characters long. Returns the padded player name."""

        if len(player_name) < self.PLAYER_NAME_TEXTWRAP_LENGTH:
            return (" " * (self.PLAYER_NAME_TEXTWRAP_LENGTH - len(player_name))) + player_name

        else:
            return player_name
```

```
    def __update_game_and_UI_after_move(self, start_loc, end_loc, move_type):

        """updates the game object and the match page GUI after a move has been made"""

        # move object reprsenting the move that was made on the Board object
        move_obj = self.__game.make_and_return_move(start_loc, end_loc, move_type)

        # updating the UI with the move object
        self.__update_board_display_after_move(move_obj.get_start_cords(), move_obj.get_end_cords(),
move_obj.get_start_colour())

        # update GUI and game object with the new current player
        self.__update_current_player_display()
        self.__game.switch_current_player()

    def __make_move_on_display(self, values, disp_board_open, ai_mode=False):

        """Makes the move on the GUI board and the game object's board if the move is legal"""

        # if the user has not selected a start and end location, return
        if len(self.__highlighted_board_positions) != 2:
            sg.popup("Please select a start and end location", keep_on_top=True)
            return

        # convert the start and end locations from coordinate strings to coordinate tuples
        start_cords = self.__str_key_to_cords_tuple(self.__highlighted_board_positions[0])
        end_cords = self.__str_key_to_cords_tuple(self.__highlighted_board_positions[1])

        # get corresponding GridLocation objects from the game object's board
        start_loc = self.__game.get_board_state()[start_cords[0]][start_cords[1]]
        end_loc = self.__game.get_board_state()[end_cords[0]][end_cords[1]]

        # getting the move type from the radio buttons
        if values["move_type_radio_move"]:
            move_type = MultiClassBoardAttributes.NORMAL_MOVE_TYPE

        elif values["move_type_radio_capture"]:
            move_type = MultiClassBoardAttributes.CAPTURE_MOVE_TYPE

        else:
            sg.popup("Please select a move type", keep_on_top=True)
```

```
            # unhighlight the selected locations
            self.__toggle_highlight_board_position(self.__highlighted_board_positions[1])
            self.__toggle_highlight_board_position(self.__highlighted_board_positions[0])
            return

        # prev_move_legal is used to determine whether the AI should make a move after the user has made a move
        prev_move_legal = True

        end_game = False

        if self.__game.is_legal_move(start_loc, end_loc, move_type): # check if the attempted move is legal

            # make move on board and GUI
            self.__update_game_and_UI_after_move(start_loc, end_loc, move_type)

            # unhighlight the selected locations
            self.__toggle_highlight_board_position(self.__highlighted_board_positions[1])
            self.__toggle_highlight_board_position(self.__highlighted_board_positions[0])

            if move_type == MultiClassBoardAttributes.CAPTURE_MOVE_TYPE:
                self.__update_number_captured_pieces_display() # update the number of pieces captured by each player

                end_game = self.__end_if_game_over(disp_board_open) # only need to check if the game is over after a capture
move

        else:
            sg.popup("ILLEGAL MOVE", keep_on_top=True)
            prev_move_legal = False # the AI should not make a move if the user's move was illegal

            # unhighlight the selected locations
            self.__toggle_highlight_board_position(self.__highlighted_board_positions[1])
            self.__toggle_highlight_board_position(self.__highlighted_board_positions[0])

        if ai_mode and prev_move_legal and not end_game:
            move = self.__game.get_ai_move() # get the AI's move

            # make the AI's move on the board and GUI
            self.__update_game_and_UI_after_move(move.get_start_loc(), move.get_end_loc(), move.get_move_type())

            if move.get_move_type() == MultiClassBoardAttributes.CAPTURE_MOVE_TYPE:
```

```
                self.__update_number_captured_pieces_display()
                self.__end_if_game_over(disp_board_open) # check if AI has won

    def __reset_game_variables(self):

        """Resets the match variables to their initial values"""

        self.__game = None
        self.__game_is_loaded = False
        self.__ai_mode = False
        self.__ai_name = None
        self.__highlighted_board_positions = []


    def __end_if_game_over(self, disp_board_open):

        """Uses the game object's set_game_status method to update the game's status and ends the game with a popup if
necessary"""

        # update the game's status using the number of pieces left on the board
        self.__game.set_game_status()

        if self.__game.is_game_over():

            # draw the final board state on the display board window
            if disp_board_open:
                self.__draw_pieces_on_disp_board()

            winning_player = self.__game.get_winning_player()

            sg.popup(f"{winning_player.get_name()} has won the game!", title="Game Over", keep_on_top=True)

            # if the user is logged need to add the game to their history and update their stats (if the match was against an
AI)
            if self.__logged_in and self.__ai_mode:

                human_won = False

                if winning_player.get_name() == self.__logged_in_username:
                    human_won = True
```

```
                # increment the user's win/loss count against the AI difficulty depending on whether they won or lost
                self.__db.update_user_stats(self.__logged_in_username, human_won, self.__ai_name)

            # remove the game from the database if it was loaded
            if self.__game_is_loaded:
                self.__db.delete_saved_game(self.__loaded_game_id)

            # add game to game history if the user is logged in
            if self.__logged_in:
                self.__db.add_game_to_history(self.__logged_in_username, self.__game.get_player_name(2),
winning_player.get_name())

            return True

        return False

    def __handle_change_piece_colour(self, new_piece_colour):

            """Changes the piece colour of the player. This method is used when a logged in user changes their piece colour"""

            if not new_piece_colour:
                sg.popup("Please select a piece colour", title="Error Changing Piece Colour", keep_on_top=True)
                return

            # update the piece colours in the MultiClassBoardAttributes class (for the current running application)
            self.__update_piece_colour(new_piece_colour)

            # update the new piece colour in the database
            self.__db.update_stored_piece_colour(self.__logged_in_username, new_piece_colour)

            sg.popup("Piece colour changed", title="Piece Colour Changed", keep_on_top=True)

            self.__change_piece_colour_window.close()

    def __handle_delete_saved_game(self, game_id):

        """Deletes the game with the given ID from the database provided that saved game data has been loaded and the user is
logged in"""

        # list of game IDs of the user's saved games
        game_id_lst = [i[0] for i in self.__saved_games]
```

```
        # check if the game ID is valid (i.e. the user has a saved game with that ID)
        if not self.__saved_games or int(game_id) not in game_id_lst:
            sg.popup(f"No game with that ID exists", title="Error Deleting Game", keep_on_top=True)
            return

        # delete the game from the database
        self.__db.delete_saved_game(game_id)
        sg.popup("Game deleted", title="Game Deleted", keep_on_top=True)

        # remove the game from the list of saved games
        self.__saved_games = [i for i in self.__saved_games if i[0] != int(game_id)]

        # update the table to not show the deleted game
        self.__load_game_window["saved_games_table"].update(values=self.__db.load_saved_games(self.__logged_in_username))

    def __handle_load_game(self, game_id):

        """Loads the game with the given ID from the database and sets up the match page with the loaded game data"""

        # load game data from the database
        loaded_game_data = self.__db.load_game_state(game_id)

        # check if the game ID is valid (i.e. the user has a saved game with that ID)
        if not loaded_game_data or loaded_game_data[0] != self.__logged_in_username:
            sg.popup(f"No game with that ID exists", title="Error Loading Game", keep_on_top=True)
            return

        # make instance variable to allow the loaded game to be deleted from the database if the user saves the game again in
the __handle_save_game method
        self.__loaded_game_id = game_id

        # unpack the loaded game data
        username, game_state_string, player2_name, player2_starts, player1pieces, player2pieces, player1_colour =
loaded_game_data

        # update the player 1 colour in the MultiClassBoardAttributes class to be the colour that the user had when they saved
the game
        self.__update_piece_colour(player1_colour)

        # if the player 2 name is an AI name, the game is an AI game
```

```
        if player2_name in self.AI_NAME_TO_LEVEL_NUM_MAP.values():
            self.__ai_mode = True
            self.__ai_name = player2_name

            # use the AI name to get the AI level and set up the match page
            ai_level = self.AI_LEVEL_NUM_TO_NAME_MAP[self.__ai_name]
            self.__setup_match_page(self.__logged_in_username, self.__ai_name, ai_level=ai_level,
game_state_string=game_state_string, player2_starts=player2_starts, player1_num_pieces=player1pieces,
player2_num_pieces=player2pieces)

        # if the player 2 name is not an AI name, the game is a local game
        elif game_state_string and player2_name:
            self.__setup_match_page(self.__logged_in_username, player2_name, game_state_string=game_state_string,
player2_starts=player2_starts, player1_num_pieces=player1pieces, player2_num_pieces=player2pieces)

        else:
            sg.popup(f"No game with id {game_id} could be found", title="Error Loading Game", keep_on_top=True)

        self.__load_game_window.close()

    def __handle_logout(self):

        """Logs the user out if they are logged in and on the home page. Otherwise, displays an error message"""

        if self.__logged_in:
            self.__logged_in = False
            self.__logged_in_username = None

            # reset the player 1 piece colour to the default colour
            self.__update_piece_colour(MultiClassBoardAttributes.DEFAULT_PLAYER_1_COLOUR)

            sg.popup("Logged out", title="Logged Out", keep_on_top=True)

        elif self.__current_page != "home_page":
            sg.popup("You can only logout from the home page", title="Error Logging Out", keep_on_top=True)

        else:
            sg.popup("You are not logged in", title="Error Logging Out", keep_on_top=True)

    def __handle_save_game(self):
```

```
        """Saves the current game state to the database if the user is logged in and on the match page. Otherwise, displays an
error message"""

        if self.__current_page == "match_page" and self.__logged_in:

            # if saving a loaded game, delete the old game from the database and replace it with the new game
            if self.__game_is_loaded:
                self.__db.delete_saved_game(self.__loaded_game_id)

            # serialise the game state to a string to store in the database
            game_state_string = self.__game.get_game_state_string()

            # determine if player 2 should start when the game is loaded again
            player2_starts = self.__game.get_player_name(2) == self.__game.get_current_player_name()

            # save the game to the database
            self.__db.save_game_state(self.__logged_in_username, game_state_string, self.__game.get_player_name(2),
player2_starts, self.__game.get_player_piece_count(1), self.__game.get_player_piece_count(2),
MultiClassBoardAttributes.player_1_colour)
            sg.popup("Game saved", title="Game Saved", keep_on_top=True)

        elif self.__current_page != "match_page":
            sg.popup("You can only save a game from the match page", title="Error Saving Game", keep_on_top=True)

        elif not self.__logged_in:
            sg.popup("You must be logged in to save a game", title="Error Saving Game", keep_on_top=True)

    def __update_piece_colour(self, new_colour):

        """Updates the piece colour of the player. This method is used when a logged in user changes their piece colour"""

        MultiClassBoardAttributes.set_player_colour(new_colour, 1)

        # if player 1's colour is changed to the default player 2 colour, player 2's colour should be changed to the default
player 1 colour
        if new_colour == MultiClassBoardAttributes.DEFAULT_PLAYER_2_COLOUR:
            MultiClassBoardAttributes.set_player_colour(MultiClassBoardAttributes.DEFAULT_PLAYER_1_COLOUR, 2)

        # if player 1's colour is changed to the default player 2 colour, player 2's colour should be changed to the default
player 1 colour
        elif new_colour == MultiClassBoardAttributes.DEFAULT_PLAYER_1_COLOUR:
```

```
            MultiClassBoardAttributes.set_player_colour(MultiClassBoardAttributes.DEFAULT_PLAYER_2_COLOUR, 2)

    def __handle_login(self, username, password):

        """Attempts to log the user in with the given username and password. Displays an error message if the username or
password is incorrect"""

        if self.__db.check_login_credentials(username, password):
            sg.popup("Logged in", title="Logged In", keep_on_top=True)

            self.__logged_in_username = username
            self.__logged_in = True

            # query the database for the user's piece colour and update the piece colours in the MultiClassBoardAttributes
class
            piece_colour = self.__db.get_piece_colour(username)
            self.__update_piece_colour(piece_colour)

            self.__login_window.close()

        else:
            sg.popup("Incorrect username or password", title="Error Logging In", keep_on_top=True)

    def __handle_sign_up(self, username, password, piece_colour):

        """Attempts to sign the user up with the given username and password. Displays an error message if the username is
already taken or if the username is reserved"""

        # ensures usernames are unique
        if self.__db.check_if_username_exists(username):
            sg.popup("Username already exists", title="Error Signing Up", keep_on_top=True)

        # username can't be one of the AI names (Easy AI, Medium AI, Hard AI)
        elif username in self.AI_RESERVED_NAMES:
            sg.popup("Username is reserved and cannot be used", title="Error Signing Up", keep_on_top=True)

        else:
            # add the user to the database
            self.__db.add_user(username, password, piece_colour)

            sg.popup("Account created", title="Account Created", keep_on_top=True)
```

```
            self.__signup_window.close()

    def __handle_showing_display_board(self):

        """Shows the display board window"""

        self.__display_board_window = self.__make_display_board_window()

        # loading the board image to be the background of the canvas
        image_path = f"{self.BOARD_IMAGES_PATH}blank_board.png"
        image = Image.open(image_path)

        # Resize the image to fit the canvas
        image.thumbnail(self.DISP_BOARD_IMAGE_SIZE)

        # storing the image in an instance variable to prevent garbage collection
        self.DISP_BOARD_BACKGROUND_IMG = ImageTk.PhotoImage(image)

        # add the background board image to the underlying Tkinter canvas
        canvas = self.__display_board_window['-CANVAS-']

        canvas.TKCanvas.create_image(self.DISP_BOARD_IMAGE_CENTRE[0], self.DISP_BOARD_IMAGE_CENTRE[1],
image=self.DISP_BOARD_BACKGROUND_IMG , anchor="center")

        # draw the pieces on the board
        self.__draw_pieces_on_disp_board()

    def __handle_restart_match(self):

        """Restarts the match if the user is on the match page. Otherwise, displays an error message."""

        if self.__current_page == "match_page":
            # make a new game with the same player names
            player1_name = self.__game.get_player_name(1)
            player2_name = self.__game.get_player_name(2)

            ai_level = None
            if self.__ai_mode:
                ai_level = self.AI_NAME_TO_LEVEL_NUM_MAP[player2_name]

            self.__setup_match_page(player1_name, player2_name, ai_level=ai_level)
```

```
        else:
            sg.popup("You can only restart a match from the match page", title="Error Restarting Match", keep_on_top=True)

    def __show_login_status_popup(self):

        """Shows a popup displaying whether the user is logged in or not"""

        if self.__logged_in_username:
            sg.popup(f"Logged in as '{self.__logged_in_username}'", title="Logged In", keep_on_top=True)
        else:
            sg.popup("Not logged in", title="Not Logged In", keep_on_top=True)

    def __update_board_display_after_move(self, start_cords, end_cords, start_colour):

        """updates the GUI board display after a move has been made"""

        # convert the start and end locations from coordinate tuples to coordinate strings (the format of the GUI board piece
element keys)
        start_cords_str = f"{start_cords[0]},{start_cords[1]}"
        end_cords_str = f"{end_cords[0]},{end_cords[1]}"

        # update the start and end location piece buttons on the GUI board with their new images
        self.__main_window[f"{start_cords_str}"].update(image_filename=f"{self.PIECE_IMAGES_PATH}blank_counter.png")
        self.__main_window[f"{end_cords_str}"].update(image_filename=f"{self.PIECE_IMAGES_PATH}{start_colour}_counter.png")

    def __update_current_player_display(self, game_is_loaded=False):

        """updates the text showing whose turn it is on the match page. Updated after every move."""

        # the text element showing whose turn it is
        current_text = self.__main_window["player1_turn_text"]

        # switches the current player display to player 2's name if the game is loaded and it is player 2's turn immediately
after loading the game
        if game_is_loaded and self.__game.get_current_player_name() == self.__game.get_player_name(2):
            current_text.update(f"{self.__game.get_player_name(2)}'s Turn ({self.__game.get_player_colour(2)})")
            return

        # switch to player 2's turn
        if self.__game.get_current_player_name() == self.__game.get_player_name(1):
```

```
            current_text.update(f"{self.__game.get_player_name(2)}'s Turn ({self.__game.get_player_colour(2)})")

        # switch to player 1's turn
        elif self.__game.get_current_player_name() == self.__game.get_player_name(2):
            current_text.update(f"{self.__game.get_player_name(1)}'s Turn ({self.__game.get_player_colour(1)})")

    def __is_board_position(self, key):

        """checks if the given key is a valid board piece button (i.e. a key of the form 'x,y' where x and y are integers)"""

        # make sure the key is not None which would cause an error with the regex
        if not key:
            return False

        min_row_index = MultiClassBoardAttributes.MIN_ROW_INDEX
        max_row_index = MultiClassBoardAttributes.MAX_ROW_INDEX

        """
        ######################################################################
        CLASS A SKILL: Regex for the validation of board coordinates
        ######################################################################
        """

        pattern = fr'^[{min_row_index}-{max_row_index}],[{min_row_index}-{max_row_index}]$'
        if bool(re.match(pattern, key)):
            return True

        return False

    def __str_key_to_cords_tuple(self, string_key):

        """converts a string key of a GUI element in the form 'x,y' to a tuple of the form (x,y) where x and y are integers"""

        return tuple(int(i) for i in string_key.split(","))

    def __cords_tuple_to_str_key(self, tuple_key):

        """converts a tuple key of the form (x,y) to a string of the form 'x,y' where x and y are integers"""

        return f"{tuple_key[0]},{tuple_key[1]}"
```

```
    def __get_player1_name(self, ai_mode):

        """returns the player 1 during a match. If the user is logged in, this is the logged in username."""

        if self.__logged_in:
            return self.__logged_in_username

        elif ai_mode:
            return self.__main_window["player_1_AI_input"].get()

        else:
            return self.__main_window["player_1_local_input"].get()

    def __toggle_highlight_board_position(self, key):

        """toggles the highlighting of a board position. If the position is already highlighted, unhighlights it. If it is not
highlighted, highlights it.
        Positions are highlighted in pink. A maximum of two positions can be highlighted at once."""

        if not self.__is_board_position(key):
            raise ValueError("Attempting to highlight a non-board position")

        # piece button specified by the key parameter
        button = self.__main_window[key]

        # if the position is already highlighted, unhighlight it
        if key in self.__highlighted_board_positions:
            button.update(button_color=('pink', sg.theme_background_color()))
            self.__highlighted_board_positions.remove(key)

        # if the position is not highlighted, highlight it
        elif key not in self.__highlighted_board_positions and len(self.__highlighted_board_positions) < 2:
            button.update(button_color=(sg.theme_background_color(), 'pink'))
            self.__highlighted_board_positions.append(key)

    def __undo_move(self, ai_mode=False):

        """Undoes the last move made on the board and updates the GUI board display. If the last move was a capture move, the
piece that was captured is restored to the board.
        If the last move was made by the AI, the move is undone twice to undo the AI's move and the human's move.
```

```
        ######################################################################
        CLASS A SKILL: Undoing a move and passing information to the Game class's undo_and_return_move method
        ######################################################################

        """

        # the move object representing the move that was undone
        move_obj = self.__game.undo_and_return_move()

        if move_obj == None:
            sg.popup("No moves to undo", keep_on_top=True)
            return

        # update the GUI board display by making the last move in reverse
        self.__update_board_display_after_move(move_obj.get_end_cords(), move_obj.get_start_cords(),
move_obj.get_start_colour())

        # if the last move was a capture move, restore the piece that was captured to the board GUI
        if move_obj.get_move_type() == MultiClassBoardAttributes.CAPTURE_MOVE_TYPE:
            cords = self.__cords_tuple_to_str_key(move_obj.get_end_cords())

self.__main_window[f"{cords}"].update(image_filename=f"{self.PIECE_IMAGES_PATH}{move_obj.get_end_colour()}_counter.png")

            # show the new number of pieces captured by each player
            self.__update_number_captured_pieces_display()

        # update the current player in game and on the GUI's display text
        self.__update_current_player_display()
        self.__game.switch_current_player()

        # if the last move was made by the AI, undo the move again to undo the AI's move
        if ai_mode:
            self.__undo_move()

    def __difficulty_level_to_ai_name(self, difficulty_level):

        """returns the AI name corresponding to the given difficulty level"""

        return self.AI_NAME_TO_LEVEL_NUM_MAP[difficulty_level]

    def __create_game_object(self, name1, name2, ai_level, game_state_string, player2_starts, player1pieces, player2pieces):
```

```
        """Creates a Game object with the given parameters and stores it in an instance variable"""

        self.__game = Game(name1, name2, ai_level=ai_level, game_state_string=game_state_string,
player2_starts=player2_starts, player1_num_pieces=player1pieces, player2_num_pieces=player2pieces)

    def play_game(self):

        """The main event loop of the GUI. Handles all events and updates the GUI accordingly. This is the only public method
of the UI class and is called by the main.py file to launch the application."""

        # boolean to keep track of whether the display board window is open
        disp_win_open = False

        while True:
            # read events and values from the all active windows
            window, event, values = sg.read_all_windows()

            # a window is closed
            if event == sg.WIN_CLOSED or event == 'Quit':

                # close the display board window if it is open
                if window == self.__display_board_window:
                    disp_win_open = False
                    self.__display_board_window.close()

                # terminate the application if the main window is closed
                elif window == self.__main_window:
                    window.close()
                    break

                else:
                    window.close()

            # go to the new game page
            if event == "new_game_button":
                self.__setup_new_game_page()

            # go to the help page
            elif event == "help_button":
                self.__setup_help_page()
```

```
            # show the user's stats in a modal window
            elif event == "show_stats_button":
                if self.__logged_in_username:
                    self.__make_stats_window()

                else:
                    sg.popup("You must be logged in to view your stats", title="Error Showing Stats", keep_on_top=True)

            # save the game if the user is logged in and on the match page
            elif event == "Save Game":
                self.__handle_save_game()

            # show the saved games modal window
            elif event == "load_game_button":
                self.__load_game_window = self.__make_load_game_window()

            # load the game with the given ID
            elif event == "submit_loading_game_id_button":
                game_id = values["loading_game_id_input"]
                self.__handle_load_game(game_id)

            # delete the game with the given ID
            elif event == "submit_deleting_game_id_button":
                game_id = values["deleting_game_id_input"]
                self.__handle_delete_saved_game(game_id)

            # show the change piece colour modal window
            elif event == "Change Piece Colour":
                if self.__logged_in:
                    self.__change_piece_colour_window = self.__make_change_piece_colour_window()

                else:
                    sg.popup("You must be logged in to change your piece colour", title="Error Changing Piece Colour",
keep_on_top=True)

            # change the piece colour of the logged in user
            elif event == "submit_change_piece_colour_button":
                new_colour = values["piece_colour_choice"]
                self.__handle_change_piece_colour(new_colour)
```

```
            # show the login status popup window
            elif event == "Show Login Status":
                self.__show_login_status_popup()

            # logout the user
            elif event == "Logout":
                self.__handle_logout()

            # show the login modal window
            elif event == "login_button":
                self.__login_window = self.__make_login_or_signup_window("login")

            # attempt to login the user
            elif event == "login_submit_button":
                username, password = values["login_username_input"], values["login_password_input"]
                self.__handle_login(username, password)

            # show the signup modal window
            elif event == "signup_button":
                self.__signup_window = self.__make_login_or_signup_window("signup")

            # attempt to sign the user up
            elif event == "signup_submit_button":
                username, password, piece_colour = values["signup_username_input"], values["signup_password_input"],
values["piece_colour_choice"]
                self.__handle_sign_up(username, password, piece_colour)

            # show the AI play input fields and hide the local play input fields in the new game page
            elif event == "AI_play_button":
                self.__toggle_new_game_input_visibility("AI_play_inputs")

            # show the local play input fields and hide the AI play input fields in the new game page
            elif event == "local_play_button":
                self.__toggle_new_game_input_visibility("local_play_inputs")

            # submit the local play input fields and set up the match page
            elif event == "submit_local_play_button":
                player_1_name = self.__get_player1_name(ai_mode=False)
                self.__setup_match_page(player_1_name, values["player_2_local_input"])

            # submit the AI play input fields and set up the match page
```

```
        elif event == "submit_AI_play_button":
            difficulty_level = int(values['difficulty_slider'])
            self.__ai_name = self.__difficulty_level_to_ai_name(difficulty_level)
            self.__ai_mode = True

            player_1_name = self.__get_player1_name(ai_mode=True)

            self.__setup_match_page(player_1_name, self.__ai_name, ai_level=difficulty_level)

        # show the display board window in a match
        elif event == "show_board_button":
            disp_win_open = True
            self.__handle_showing_display_board()

        # undo the last move in a match
        elif event == "undo_move_button":
            self.__undo_move(self.__ai_mode)

            # redraw the pieces on the display board if it is open
            if disp_win_open:
                self.__draw_pieces_on_disp_board()

        # highlight a board position if it is not already highlighted. If it is already highlighted, unhighlight it
        elif self.__is_board_position(event):
            self.__toggle_highlight_board_position(event)

        # go back to the home page
        elif event == "Home":
            self.__main_window.close()
            self.__setup_home_page()

        # restart the match
        elif event == "Restart Match":
            self.__handle_restart_match()

        # make a move in a match
        elif event == "submit_move_button":
            self.__make_move_on_display(values, disp_win_open, self.__ai_mode)

            if disp_win_open:
                self.__draw_pieces_on_disp_board()
```

```
                if self.__game.is_game_over():

                    # draw the final board state on the display board window
                    if disp_win_open:
                        self.__display_board_window.close()

                    # return to the home page and reset the game variables
                    disp_win_open = False
                    self.__reset_game_variables()
                    self.__setup_home_page()

        self.__main_window.close()

class TerminalUI(UI):

    """Terminal user interface for the program. Inherits from the UI class. Only supports local play.

    ######################################################################
    CLASS A SKILL: Complex OOP model with encapsulation, inheritance, polymorphism and composition
    GOOD CODING STYLE: Well-designed user interface
    GOOD CODING STYLE: Use of constants
    ######################################################################

    """

    # constants used to display the board
    EMPTY_SPACE_CHAR = "."
    ROW_SPACING = " " * 2
    COL_INDENT = " " * 5
    COL_SPACING = " " * 2
    COL_UNDERLINE_INDENT = " " * 3
    COL_UNDERLINE = "—" * 18

    def __init__(self):
        self.__UI_type = "TERMINAL"
        self.__game = Game(input("Enter player 1's name: "), input("Enter player 2's name: "))

    def get_UI_type(self):
        return self.__UI_type
```

```
    def __get_cords_from_user(self, prompt):

        """Gets a valid coordinate from the user in the form 'row,col' where row and col are integers between 0 and 5
inclusive."""

        valid = False

        min_row_index = MultiClassBoardAttributes.MIN_ROW_INDEX
        max_row_index = MultiClassBoardAttributes.MAX_ROW_INDEX

        # regex pattern to check if the user's input is valid
        pattern = fr'^[{min_row_index}-{max_row_index}],[{min_row_index}-{max_row_index}]$'

        while not valid:

            """
            ######################################################################
            CLASS A SKILL: Regex for the validation of board coordinates
            ######################################################################
            """

            choice = input(prompt)
            if bool(re.match(pattern, choice)):
                valid = True

            else:
                print("Invalid Coordinate. Must be of the form 'r,c' where r and c are integers between 0 and 5 inclusive.")

        # convert the user's input to a tuple of integers of the form (r,c)
        return tuple([int(i) for i in choice.split(",")])

    def __display_board(self):

        """Displays the board in the terminal"""

        board = self.__game.get_board_state()

        disp_board = []

        for row in board:
            for loc in row:
```

```
                if loc.is_empty():
                    disp_board.append(f"{self.EMPTY_SPACE_CHAR}")

                else:
                    # single character representing the colour of the piece
                    disp_board.append(loc.get_piece_colour()[0])

        # convert the one dimensional array to a 6x6 two dimensional array
        disp_board = oneD_to_twoD_array(disp_board, MultiClassBoardAttributes.MAX_ROW_INDEX + 1)

        # display the column indexes
        self.__display_col_indexes()

        # displaay the row indexes and the board
        for ind,row in enumerate(disp_board):
            print(f"{ind} | ", end=" ")
            print(self.ROW_SPACING.join(row))

    def __display_col_indexes(self):

        """Displays the column indexes of the board in the terminal"""

        print(self.COL_INDENT, end="")
        print(self.COL_SPACING.join([str(i) for i in range(MultiClassBoardAttributes.MAX_ROW_INDEX + 1)]))
        print(self.COL_UNDERLINE_INDENT, end="")
        print(self.COL_UNDERLINE)

    def __display_winner(self):

        """Displays the winner of the game in the terminal"""

        winner = self.__game.get_winning_player()
        print(f"{winner.get_name()} won!")

    def __get_move_type(self):

        """Gets the move type from the user. Either a normal move or a capture move."""

        valid = False
        while not valid:
```

```
            move_type = input(f"Enter {MultiClassBoardAttributes.NORMAL_MOVE_TYPE} for an ordinary move to an adjacent
position or {MultiClassBoardAttributes.CAPTURE_MOVE_TYPE} for a capturing move: ")

            if move_type == MultiClassBoardAttributes.NORMAL_MOVE_TYPE or move_type ==
MultiClassBoardAttributes.CAPTURE_MOVE_TYPE:
                valid = True

            else:
                print("Invalid move type. Please try again.")

        return move_type

    def play_game(self):

        """The main loop of the terminal UI. Handles all events and updates the UI accordingly.
        This is the main public method of the UI class and is called by the main.py file to launch the application."""

        while not self.__game.is_game_over():

            self.__display_board()

            print(f"\n{self.__game.get_current_player_name()}'s turn ({self.__game.get_current_player_colour()}).\n")

            valid = False
            while not valid:

                move_type = self.__get_move_type()

                start_cords = self.__get_cords_from_user("Enter a row and column pair in the format r,c for the piece you want
to move: ")
                end_cords = self.__get_cords_from_user("Enter a row and column pair in the format r,c for where you want to
move to: ")

                # getting the corresponding GridLocation objects from the game object's board
                start_loc = self.__game.get_board_state()[start_cords[0]][start_cords[1]]
                end_loc = self.__game.get_board_state()[end_cords[0]][end_cords[1]]

                if self.__game.is_legal_move(start_loc, end_loc, move_type):
                    valid = True
                else:
                    print("Invalid move. Please try again.")
```

```
            self.__game.make_and_return_move(start_loc, end_loc, move_type)

            self.__game.switch_current_player()

            # game can only be over after a capture move
            if move_type == MultiClassBoardAttributes.CAPTURE_MOVE_TYPE:
                self.__game.set_game_status()

        # display the final board and the winner
        self.__display_board()
        self.__display_winner()
```

UtilityFunctions.py

```
import random

def oneD_to_twoD_array(lst, width):

    """returns a 2D array with the specified width from a 1D array. Each element in the 2D array is a list with a length equal
to the specified width parameter."""

    return [lst[i:i+width] for i in range(0, len(lst), width)]

def twoD_to_oneD_array(lst):

    """returns a 1D array from a 2D array. Each row in the 2D array is appended to the end of the 1D array."""

    return [item for sublist in lst for item in sublist]

def shuffle_2D_array(arr):

    """returns a 2D array with the same elements as the specified 2D array but with the elements shuffled.
    Elements are shuffled between rows and columns."""

    width = len(arr[0])

    arr = twoD_to_oneD_array(arr)
```

```
    random.shuffle(arr)
    arr = oneD_to_twoD_array(arr, width)

    return arr
```

Main.py

```
from UI import TerminalUI, GraphicalUI

if __name__ == "__main__":

    valid_ui_type = False

    while not valid_ui_type:

        ui_type = input("Enter 't' for terminal UI or 'g' for graphical UI: ")

        if ui_type == 't':
            ui = TerminalUI()
            valid_ui_type = True

        elif ui_type == 'g':
            ui = GraphicalUI()
            valid_ui_type = True

        else:
            print("Invalid input. Please try again.")

    ui.play_game()
```

4 System testing *(8 marks)*

The following section is the testing section of this document.

4.1 Test plan

The following section will be a series of test tables, describing the tests carried out for each objective outlined in the analysis section of this document. Each objective is then tested in a video.

4.1.1 MVP (Stage 1) Test Table

Below is the testing table for stage 1 of the project (the MVP).

Test number	Purpose of the test	Test data	Expected outcome	Reference to test result	Pass/Fail
1	The program can be executed in terminal mode by entering "t". The program will keep asking the user for input until the input is valid (only “t” is a valid mode before the GUI has been implemented).	1) Enter “asdf” into the terminal on start 2) Enter “t” into the terminal on start	1) The program rejects the invalid mode and asks the user to enter the mode again 2) The program is executed in terminal mode	0:27-0:52	**Pass**
2	The program asks the user to enter a name for player 1	Enter “Sahil K”	Program accepts the name without error	0:52-1:01	**Pass**
3	The program asks the user to enter a name for player 2	Enter “Bob D”	Program accepts the name without error	1:01-1:10	**Pass**
4	When executing in terminal mode, the board is printed after each move. The rows and columns are numbered so a coordinate system can be used to identify board cells	N/A	Board is printed out as described in the test purpose	1:10-1:19	**Pass**

5	Player 1 uses yellow pieces denoted by “y” and player 2 uses green pieces denoted by “g”	N/A	Player 1 pieces on the board are shown as “y” and player 2 pieces are shown as “g” when the board is printed	1:19:1:36	**Pass**
6	The program asks the current player to enter the type of move (normal adjacent move or capturing move specified by a string) that the player wants to make. If an invalid move type is entered, an error message is shown, and they are asked again to enter a move type until a valid move type is entered	1) Enter “sadffs” 2) Enter “move” 3) Enter “capture”	1) Program rejects the invalid move type and asks the user to enter it again 2) Program accepts the move type without error 3) Program accepts the move type without error	1:36-2:23	**Pass**
7	A player is asked to enter a row and column number separated by a comma indicating the piece they want to be moved. Any other input results in an error being reported to the terminal and the user is asked to enter the coordinates again	1) Enter “6,3” 2) Enter “4_2” 3) Enter “2,2”	1) Program rejects the invalid board coordinate and asks the user to enter it again 2) Program rejects the invalid board coordinate and asks the user to enter it again 3) Program accepts the coordinate without error	2:23-3:59	**Pass**
8	A player is asked to enter a row and column number separated by a comma indicating where they want to move their	1) Enter “6,3” 2) Enter “4_2” 3) Enter “2,4”	1) Program rejects the invalid board coordinate and asks the user to enter it again	3:59-4:40	**Pass**

	piece to. Any other input results in an error being reported to the terminal and the user is asked to enter the coordinates again		2) Program rejects the invalid board coordinate and asks the user to enter it again 3) Program accepts the coordinate without error		
9	The program can detect whether a normal adjacent move is legal or not	1) Attempt to make normal move from 1,1 to 2,1 2) Attempt to make normal move from 4,0 to 3,2 3) Attempt to make normal move from 4,5 to 3,5	1) Program does not make move as it is illegal and informs the use of the unsuccessful move 2) Program does not make move as it is illegal and informs the use of the unsuccessful move 3) Program makes move without error	4:40-6:23	**Pass**
10	The program can detect whether a capturing move is legal or no	1) Attempt to make capture from 4,1 to 1,1 2) Attempt to make a capture from 0,2 to 2,0 where there is a green piece at 0,2 and a	1) Program does not make move as it is illegal and informs the use of the unsuccessful move 2) Program makes move without error	6:23-9:54	**Pass**

		yellow piece at 2,0 and it is green's turn			
11	Turns alternate between players playing on the same terminal locally and the name of the current player is displayed	Make a legal move for both players. Make a move which is illegal due to using the other player's pieces to prove that player turn has switched and it's not just the text switching.	After the first move, the next move is made by the other player and the current player text is changed. The attempted move with the other player's pieces is displayed to be illegal.	9:54-11:44	**Pass**
12	The program can detect when a player has won which is when a player has captured all the opponent's pieces.	Achieved when objective 13 is passed	N/A	11:44-13:09	**Pass**
13	If a game is won, the winning player is reported, and the program terminates	A winning move is made	The game terminates, displaying the name of the winning player. The program terminates	11:44-13:09	**Pass**

4.1.2 Initial GUI (Stage 2) Test Table

Below is the testing table for the stage 2 (initial GUI) objectives.

Test number	**Purpose of the test**	**Test data**	**Expected outcome**	**Reference to test result**	**Pass /Fail**
14	The program can be executed in GUI mode when entering "g". The program will keep asking the user for a mode until a	1) Enter "t" into the terminal on start 2) Enter "sf" into the terminal	1) The program is executed in terminal mode 2) The program rejects the invalid mode and asks the user to enter	13:09-13:54	**Pass**

	valid mode (“g” or “t”) is entered	3) Enter “g” into the terminal on start	the mode again 3) The program is executed in GUI mode		
15	The GUI displays a "Help" and a "New Game" button on the home page and a “Quit” button in a menu bar	N/A	Buttons are displayed on home page	13:54-14:20	**Pass**
16	Pressing the “Quit” button closes the program	Press the “Quit” button	Program successfully terminates	14:20:14:33	**Pass**
17	Pressing the “Help” button opens a window that displays the rules of Surakarta	Press the “Help” button	The help page window is loaded	14:33-15:04	**Pass**
18	A menu bar is displayed at the top of the main windows which has a “Home” button	N/A	“Home” button is displayed in the menu bar	15:04:15:24	**Pass**
19	If the menu bar “Home” button is pressed, the user is returned to the home page	Press the “Home” button	Program returns to the home page	15:24-15:43	**Pass**
20	When the “New Game” button is pressed, there are two input fields to enter the names for Player 1 and Player 2 which will be used during the match. Pressing the “Submit” button creates a new window with the Surakarta board displayed.	Achieved when objective 50 succeeds	A local play match is started with the player names specified	34:00-34:06	**Pass**

	Pieces are in their starting locations.				
21	A “submit move” button is displayed above the game board	N/A	Button is displayed on match page	16:30-16:41	**Pass**
22	A pair of radio buttons labelled “move” and “capture” are displayed to choose between a normal move and a capturing move	N/A	Buttons are displayed on match page	16:41-17:00	**Pass**
23	The player who's move it is to make has their name and the colour of their pieces displayed on the screen	N/A	The name of the current player and the colour pieces the current player uses is displayed	17:00-17:19	**Pass**
24	The number of pieces each player has captured is displayed	N/A	Text for each player showing the number of pieces they have captured is displayed	17:19-17:36	**Pass**
25	The board is displayed as a 6x6 grid of buttons which have images of pieces on them	N/A	Piece buttons are displayed on match page	17:36-17:55	**Pass**
26	Clicking on a position on the board highlights this position	Click on a board position	The clicked-on board position is highlighted pink	17:55-18:15	**Pass**
27	If a user accidentally clicks on a position, they can click on it again to un-highlight it	Click on a highlighted board position	The highlighted board position is unhighlighted	18:15-18:32	**Pass**
28	The game only allows two positions to be highlighted at	1) Click on a third board position when two	1) The clicked-on position will not be highlighted	18:32:19:19	**Pass**

	once since a move will only ever involve a start position and an end position	are already highlighted 2) With two positions highlighted, click on one of the highlighted positions and click on a different position	2) Clicking on one of the highlighted positions unhighlights it. Clicking on a new position highlights the new position		
29	Pressing the “submit” button makes the move onscreen specified by the highlighted positions and move type choice if the move is legal	Make a legal move by pressing the “submit” button when a move type has been selected and two board positions have been highlighted	The move is made on the board	19:19-19:58	**Pass**
30	If an illegal move is attempted, an error popup is displayed, and the user can attempt a different move	Attempt an illegal move followed by a legal move	A popup informs the user that the move was illegal, and they can enter a different move. Player turn does not switch after an illegal move is attempted. After a legal move is made, player turn does switch.	19:58-20:43	**Pass**
31	When a player captures a piece, their captured piece count is incremented onscreen	Make a legal capture	The player who made the capture’s text showing the number of pieces they have captured will increment	20:53-22:16	**Pass**
32	A “show board” button is displayed above the board during a match	N/A	Button is displayed above the board in a match	22:16-22:34	**Pass**
33	If the “show board” button is pressed, a small display board popup window appears that	Press the “show board” button and move it around.	A display board showing the current board state on the looped tracks is shown and can be moved around.	22:34-23:09	**Pass**

	shows the current board state and the looped tracks. The display board can be moved on screen.				
34	The game can be played while the display board is open and any updates to the game state are shown on the display board. The display board window remains ontop of the main window while open	Demonstrate a series of normal moves and captures while the display board is open	The moves made are updated on both the piece buttons on the main window and the display board	23:09-24:52	**Pass**
35	The display board can be closed and opened during the match	Open and close the display board during a sequence of moves	The display board can be opened and closed without causing an error during a match and it always shows the current board state	24:52-25:29	**Pass**
36	A “Restart Match” button in the menu bar can be clicked during a game to reset the current match	Press the “Restart Match” button when in a match	The match is restarted	25:29-26:08	**Pass**
37	If the “Restart Match” button is pressed when not in a match, an error popup is displayed	Press the “Restart Match” button on the home page	An error popup is displayed informing the user that they can only restart a match from the match page	26:08-26:34	**Pass**
38	If a game state is a win for either player, a popup message is displayed with the winning player's name	Make a winning move	A popup appears, displaying the name of the winning player	26:34-27:21	**Pass**

39	After a game is completed, the program returns to the home page	Make a winning move and close the winning player popup	The program returns to the home page	27:21-27:36	**Pass**
40	If the “submit move” button in a match is pressed when no move type has been selected, an error popup is displayed	Press the “submit move” button when no move type has been selected but two positions have been highlighted	An error popup is displayed informing the user that they need to select a move type	27:36-28:09	**Pass**
41	If the “submit move” button in a match is pressed when two positions have not been highlighted, an error popup is displayed	1) Press the “submit move” button with one highlighted position 2) Press the “submit move” button with no highlighted positions	Both tests result in a suitable error popup message	28:09-28:52	**Pass**

4.1.3 Advanced GUI (Stage 3) Test Table

Below is the testing table for the stage 3 (Advanced GUI) objectives.

Test number	**Purpose of the test**	**Test data**	**Expected outcome**	**Reference to test result**	**Pass/Fail**
42	An “undo” button is displayed during a match	N/A	Button is displayed during a match	28:52-29:12	**Pass**
43	An “undo” button can be clicked to take back the most recent move	Make a series of moves and then repeatedly press the “undo” button	The board state is updated to be the state it was in before the last move was made each time the “undo” button is pressed	29:12-29:58	**Pass**

44	If the undo button is pressed with no moves left to undo, an error popup is displayed	Press the “undo” move at the start of the game	An error popup is displayed saying that there are no more moves to undo	29:58-30:26	**Pass**
45	If the display board is open and a move is undone, the display board is updated to show the state of the board after the undo	Make a series of moves. Press the “undo” button with the display board open	The moves made are undone when the “undo” button is pressed, and the pre-move board state is restored on the main piece buttons board and the display board	30:26-31:14	**Pass**
46	If the undo button is pressed and a capturing move was undone, the captured piece count of the player who made the capture originally is decremented onscreen	Make a capture. Press the “undo” button to undo a capture	The capture is undone and the player who made the capture has their capture count decremented onscreen	31:14-31:55	**Pass**

4.1.4 Easy AI (Stage 4) Test Table

Below is the testing table for the stage 4 (Easy AI) objectives.

Test number	**Purpose of the test**	**Test data**	**Expected outcome**	**Reference to test result**	**Pass/Fail**
47	Pressing the “New Game” button on the home page opens a new window which displays a button for “Local Play” and “AI Play”	N/A	Both buttons are displayed on the “New Game” page	31:55-32:27	**Pass**
48	When the “Local Play” button is pressed, there are two input fields present for the user to enter	Press the “Local Play” button	There are two input fields for player 1’s name and player 2’s name are shown as	32:27-32:49	**Pass**

	player 1's name and player 2's name. A submit button is also shown.		well as a submit button		
49	When the "AI Play" button is pressed, there is one input field present for the user to enter their name. A submit button is also shown.	Press the "AI Play" button	An input field for player 1's name is shown as well as a submit button	32:49-33:18	**Pass**
50	Pressing the "Submit" button after filling in the local play fields starts a match in local play using the player names entered.	Press the "Local Play" button, enter names for player 1 and player 2 and press "Submit"	A local play match is started with the player names specified	33:18-34:06	**Pass**
51	Pressing the "Submit" button after filling in the AI play field for player 1's name starts a match against the Easy AI with the entered name being used.	Press the "AI Play" button, enter a name for player 1 and press "Submit". The slider present is part of the tree search objective stage. Since the default value of the slider is 1 (which corresponds to the easy AI), it does not need to be changed.	An AI play match is started with the specified name for player 1.	34:06-34:42	**Pass**
52	In AI mode, the computer makes moves against the human player with a greedy algorithm	Make a series of moves against the Easy AI	The AI makes moves such that it captures if possible, otherwise if it can it tries to move out of the corner, else it makes a random move.	34:42-36:12	**Pass**
53	When the undo button is pressed in a match against the AI, the most	Make a series of moves against the Easy AI and press	The most recent human and AI moves	36:12-36:53	**Pass**

	recent human player and AI moves are both undone	the “undo” button repeatedly	(two moves) are undone in pairs		

4.1.5 Tree Search AI (Stage 5) Test Table

Below is the testing table for the stage 5 (Tree Search AI) objectives.

Test number	Purpose of the test	Test data	Expected outcome	Reference to test result	Pass/Fail
54	After pressing the “AI Play” button, a slider appears to allow a difficulty level (1, 2 or 3) to be chosen	Press the “AI Play” button	A slider with three levels appears below the player 1 name input field	36:53-37:38	**Pass**
55	Using difficulty 1 results in an easy AI opponent playing against the human player with a greedy algorithm	Press the “AI Play” button, enter a name for player 1, set the slider to level 1 and press the “Submit” button. Make a series of moves against the Easy AI	A match is started against the Easy AI opponent. The Easy AI plays moves against the human player	37:38-38:11	**Pass**
56	Starting a match with difficulty level 2 results in a match against a medium AI opponent	Press the “AI Play” button, enter a name for player 1, set the slider to level 2 and press the “Submit” button	A match is started against the Medium AI opponent	38:11-38:42	**Pass**
57	Starting a match with difficulty level 3 results in a match against a hard AI opponent	Press the “AI Play” button, enter a name for player 1, set the slider to level 3 and press the “Submit” button	A match is started against the Hard AI opponent	38:42-39:12	**Pass**
58	The medium AI opponent uses the Monte Carlo Tree Search algorithm	Play against the Medium AI and evaluate its move choices	Medium AI takes 15 seconds and then makes a move	39:12-43:02	**Pass**

	running for 15 seconds				
59	The hard AI opponent uses the Monte Carlo Tree Search algorithm running for 30 seconds	Play against the Hard AI and evaluate its move choices	Hard AI takes 30 seconds and then makes a move	43:02-45:26	**Pass**

4.1.6 Accounts and Customisation (Stage 6) Test Table

Below is the testing table for the stage 6 (Accounts and Customisation) objectives.

Test number	**Purpose of the test**	**Test data**	**Expected outcome**	**Reference to test result**	**Pass/Fail**
60	In the home page, a “Login” button and a “Sign Up” button are displayed	N/A	Buttons are displayed on the home page	46:07-46:20	**Pass**
61	If the “Sign Up” button is pressed, the user is prompted to enter a new username, password, and piece colour and a submit button is below these fields	Press the “Sign Up” button	The sign-up modal window is opened	46:20-46:47	**Pass**
62	If the “Submit” button is pressed on the sign-up form and the entered username is not reserved and does not already exist in the database, a new account is created, and the user is informed of the successful account creation via a popup	Press the “Submit” button after entering a valid username along with a password and piece colour	A popup is shown informing the user of the successful account creation	46:47-48:17	**Pass**

63	If the “Submit” button is pressed on the sign-up form, the entered username is looked up in the database. If the entered username already exists in the database or is one of the names reserved for AI players (“Easy AI”, “Medium AI” or “Hard AI”), an error is reported to the user	Press the “Submit” button after entering an invalid username (along with the other necessary account details). Both the case for attempting to use a username which already exists, and a reserved AI name are tested.	In both cases, an error popup is shown informing the user of the unsuccessful sign-up	48:17-49:48	**Pass**
64	If the “Login” button is pressed, a popup window is displayed with a form to enter a username and password with a submit button	Press the “Login” button	The Login modal window is opened	49:48-50:15	**Pass**
65	If the “Submit” button is pressed in the login form, the user's credentials are looked up in the database and if they are found, the user is logged in and informed of this with a popup message	Press the “Submit” button after entering a valid username and password	The user is logged in and a popup message informs the user of the successful login	50:15-50:54	**Pass**
66	If the “Submit” button is pressed in the login form and a user's entered credentials are not found in the database, an error message is shown to the user	Press the “Submit” button after entering an invalid username or password	The user is informed of the failed login	50:54-51:58	**Pass**

67	The main window's menu bar displays a "Show Login Status" button	N/A	Button is shown in the menu bar	51:58-52:18	**Pass**
68	If the "Show Login Status" button is pressed and no user is logged in, a popup will appear saying that no user is logged in	Press the "Show Login Status" button when no user is logged in	A popup is shown informing the user that no user is logged in	52:18-52:42	**Pass**
69	When the "Show Login Status" button is pressed, if a user is logged in, a popup will appear informing them of who they are logged in as	Press the "Show Login Status" button when a user is logged in	A popup is shown informing the user of the logged in user's username	52:42-53:20	**Pass**
70	The main windows' menu bar displays a "Logout" button	N/A	Button shown in the menu bar.	53:20-53:39	**Pass**
71	When the "Logout" button is pressed and no user is logged in, an error popup appears	Press the "Logout" button when no user is logged in.	An error popup message appears stating that no user is logged in.	53:39-54:07	**Pass**
72	When the "Logout" button is pressed and a user is logged in, the currently logged in user is logged out and a popup message informs the user of the successful logout.	Press the "Logout" button when a user is logged in. Press the "Show Login Status" button.	The user is successfully logged out and are informed of this with a popup message. A popup appears stating that there is no currently logged in user after the "Show Login Status" button is pressed.	54:07-54:50	**Pass**
73	When a user is logged in and presses the "Local Play" button, they only enter one	Show the local play input fields when not logged in. Log into an account and	When not logged in, the user must enter a name for player 2. This field is not	54:50-55:53	**Pass**

	name which is the name of player 2 because the game will use their username for player 1	show the local play input fields.	present when logged in.		
74	When a user is logged in and presses the “AI Play” button, they don’t have to enter any name because the game will use their username for player 1	Show the AI play input fields when not logged in. Log into an account and show the AI play input fields.	When not logged in, the user must enter a name for player 2. This field is not present when logged in.	55:53-56:26	**Pass**
75	The menu bar ontop of all main windows will have a “Change Piece Colour” button	N/A	Button shown in the menu bar	56:26-56:44	**Pass**
76	If the “Change Piece Colour” button is pressed and no user is logged in, an error popup is displayed	Press the “Change Piece Colour” button when no user is logged in	An error popup is displayed informing the user that they can only change their piece colour if they are logged in	56:44-57:14	**Pass**
77	If the “Change Piece Colour” button is pressed and a user is logged in, the user is prompted to select a new piece colour from a drop-down menu	Press the “Change Piece Colour” button when a user is logged in.	The change piece colour window is displayed.	57:14-57:54	**Pass**
78	When the “Submit” button in the change piece colour window is pressed, if the selected piece colour is not none, the user’s piece colour is changed in the database	Show the current piece colour in the database. Select a new piece colour in the change piece colour” window. Show the updated piece colour in the database	Ther user’s piece colour will be updated in the database.	57:54-58:58	**Pass**

79	When the “Submit” button in the change piece colour window is pressed, if the selected piece colour is none, a popup error message is displayed	Login and then press the change piece colour button to open the selection window and press “Submit” without choosing a piece colour	An error popup is displayed informing the user that they need to select a piece colour	58:58-59:32	**Pass**
80	A user's customisation choices are saved to their account so when they login, their customisations are used	Enter a match while not logged in. Log into an account with a changed piece colour (not yellow) and enter a match	The logged in user will have a different piece colour to the default player 1 colour (yellow)	59:32-1:00:35	**Pass**
81	By default, the computer player and player 2 in local play will use green pieces. If a user is logged in and has green as their piece colour, the computer / player 2 will use yellow pieces.	1) Enter a match when not logged in 2) Login to an account with a piece colour of green	1) Player 1 will use yellow pieces and player 2 will use green pieces 2) Player 1 will use green pieces and player 2 will now use yellow pieces	1:00:35 - 1:02:13	**Pass**

4.1.7 Saving Game State (Stage 7) Test Table

Below is the testing table for the stage 7 (Saving Game State) objectives.

Test number	**Purpose of the test**	**Test data**	**Expected outcome**	**Reference to test result**	**Pass/Fail**
82	In the home page, a “Load Game” button is displayed	N/A	Button is shown on the home page	1:02:13 - 1:02:34	**Pass**
83	If the “Load Game” button is pressed when no user is logged in,	Press the “Load Game” button when no user is logged in	An error popup informs the user that they can only load a	1:02:34 - 1:03:01	**Pass**

	an error message is displayed via a popup		game if they are logged in		
84	If the “Load Game” button is pressed and a user is logged in, a load game popup window is displayed which shows the games the user has saved.	Press the “Load Game” button when a user is logged in	A modal window is opened showing the games in a table that the logged in user has saved.	1:03:01 - 1:03:35	**Pass**
85	A “Save Game” button is displayed in the menu bar of main windows	N/A	Button displayed on home page	1:03:35 - 1:03:51	**Pass**
86	If the “Save Game” button is pressed while not in a match, an error message is shown via a popup	Press the “Save Game” button when not in a match.	An error popup informing the user that they cannot save a game if they are not in a match (not on the match page).	1:03:51 - 1:04:20	**Pass**
87	If the “Save Game" button is pressed during the game when a user is not logged in, an error is shown via a popup	Press the “Save Game” button when not logged in.	An error popup informing the user that they cannot save a game if they are not logged in.	1:04:20 - 1:04:59	**Pass**
88	If the “Save Game” button is pressed during a game when a user is logged in, the current game state is saved to the user’s account and a popup displays saying the game has been saved	Press the “Save Game” button when logged in and in a match and then view the saved games table.	A popup message informs the user the game has been saved. The saved games table contains the match that was just saved.	1:05:10 - 1:06:23	**Pass**
89	The load game popup window can accept a game ID as input to load a specified game to	Enter a game ID into the load input field in the load game window, press the “Load”	The game specified by the entered game ID is loaded and can be played	1:06:23 - 1:07:13	**Pass**

	be continued after a “load” button is pressed	button and play a few moves in the loaded game			
90	If the current match being played is loaded and the current game state is saved, the old save for the loaded game is deleted and a saved game is deleted when it is completed	Load a saved game, make a few moves, save the game, go back to the home page, open the saved games window, and load the game that was just saved. Then load a game, complete it, and open the saved games table.	When the loaded game is saved again, it overwrites the old game save. When the saved game is completed, it no longer appears in the saved games table.	1:07:13 - 1:09:34	**Pass**
91	The load game popup window can accept a game ID as input to delete a specified game so that it is no longer saved after a “delete” button is pressed	Enter a game ID into the delete input field in the load game window, press the “Delete” button	A popup informs the user that the game has been deleted and the game is removed from the table of saved games	1:09:34 - 1:10:13	**Pass**
92	If the game ID passed to the load game input form is empty or doesn’t match a game ID saved for the user when the “load” button is pressed, an error popup message is displayed	Enter a game ID into the load input field which is not present in the saved games table and press “load”. Then press the “load” button when no game ID has been entered.	A popup informs the user that the provided game ID doesn’t match any of their saved games in both cases.	1:10:13 - 1:11:13	**Pass**
93	If the game ID passed to the delete game input form is empty or doesn’t match a game ID saved for the user when the “delete” button is pressed, an error popup message is displayed	Enter a game ID into the delete input field which is not present in the saved games table and press “delete”. Then press the “delete” button when no game ID has been entered.	A popup informs the user that the provided game ID doesn’t match any of their saved games in both cases.	1:11:42 - 1:12:24	**Pass**

4.1.8 Saving User Stats (Stage 8) Test Table

Below is the testing table for the stage 8 (Saving User Stats) objectives.

Test number	Purpose of the test	Test data	Expected outcome	Reference to test result	Pass/Fail
94	A “Show Stats” button is displayed on the home page	N/A	Button displayed on the home page	1:12:48 - 1:12:58	**Pass**
95	If a user is not logged in and presses the “Show Stats” button, an error message is displayed via a popup	Press the “Show Stats” button when no user is logged in.	An error popup informs the user that they can only view stats if they are logged in.	1:12:58 - 1:13:20	**Pass**
96	If a logged in user presses the “Show Stats" button, a stats window appears.	Press the “Show Stats” button when logged in.	The stats window is shown.	1:13:46 - 1:14:29	**Pass**
97	The stats popup window has a table showing their wins and losses against each AI difficulty. The initial values for wins and losses against each AI difficulty are 0.	Press the “Show Stats” button when a user is logged into a brand-new account.	The stats window is shown and contains a table showing the number of wins and losses the user has against each AI difficulty. The wins and losses against each AI difficulty are 0.	1:14:29 - 1:14:59	**Pass**
98	The stats popup window has a match history table with columns for each game’s game ID, date, opponent, and the name of the winning player	Press the “Show Stats” button when a user is logged in	The stats window is shown and contains a table showing the historical games played by the user	1:15:37 - 1:16:03	**Pass**

99	After a match against an AI player is completed, a logged in user's stats against the corresponding AI difficulty in the stats table are updated	Login to an account, load a match against an AI, complete the match, press the "Show Stats" button.	If the user won the match, their wins field in the stats table against the corresponding AI opponent is incremented. If the user lost, the losses field is incremented.	1:16:03 - 1:18:57	**Pass**
100	After a match is completed, the player's username, the match's game ID, the game's date, the opponent's name, and the name of the winning player are added as a new row in the game history table	Login to an account, load a match, complete the match, press the "Show Stats" button.	The completed match will be added to the user's match history, viewable from the stats window.	1:16:03 - 1:18:57	**Pass**

4.2 Testing Videos

Below is a link to my testing video which has been uploaded to YouTube.

https://bit.ly/3wjXLAh

5 Evaluation *(4 marks)*

The following section is the evaluation section of this document.

5.1 Comparison of project performance against the objectives

Overall, I met all the objectives that were set out for this project. The objectives in the MVP were first completed to create a working game of Surakarta in the terminal (with local play). The user did request a simple terminal version of the game to be available but even if they didn't, a terminal MVP version would still have been created because it enabled me to get some of the core game algorithms like move legality checks and making moves implemented early in the programming process.

After the MVP, I implemented the stage 2 objectives for the Initial GUI since the looped tracks on the board were unable to be displayed in the terminal which made debugging capturing moves more difficult.

I then decided to implement stage 3 (the advanced GUI) to add undoing and restarting functionality as this would complete the quality-of-life features in this project. Undoing moves also proved very useful when debugging MCTS, as it allowed me to look at the variation in moves made by the AI opponent on the same board state to assess how well the AI was playing. Thus, it was a good decision to add this feature early in the project (before the implementation of the medium and hard AI opponents).

After this, I added the Easy AI to the game which was done before MCTS as it enabled me to set up the inheritance structure for AI players prior to adding the more complex Medium and Hard AI difficulties which were added together in the next stage due to them both using the MCTS algorithm.

The next stage of objectives to be implemented was setting up accounts for user customisation. This had to be the next stage as saving and loading game states as well as user stats are tied to an account. Loading game states was very useful for the dry runs performed in the design section of this document as I was able to run SQL commands directly to add certain game states to an account to be used. This also helped me with further testing of the MCTS algorithm in varying game states.

5.2 Effectiveness of the solution

The complete solution is very effective since it meets all the requirements requested by the user and meets all the project's objectives derived from these requirements. The graphical use interface is simple to use, and the project has many features which enhance the application beyond playing on a physical board and the few currently available software options.

As for changes I would make if I was doing the project again, I would have used the Tkinter framework over PySimpleGUI as PySimpleGUI lacks features like making a multi-page application and is more limiting in the way elements can be structured on the page. Using Tkinter would have prevented needing to open a new window and close the old window when moving between main pages. Tkinter would also have prevented the need for a display board as the library would allow me to draw the looped tracks directly onto the main board between the piece buttons. Another improvement I would make is I would make the graphical user interface dynamically resize depending on device screen size which would also be achieved with the Tkinter framework. This would prevent any part of the application from being cut off the edge of the screen on smaller devices.

As well as this, while I determined that MCTS would be a better AI algorithm to use than minimax from a theory perspective, it would have been beneficial to implement both solutions and evaluate which made a better player from empirical testing. I could do this by making the MCTS AI play the minimax AI many times, recording how many times each AI won. Since MCTS is time dependent while minimax is run to a certain tree depth, I would first determine suitable parameters for both algorithms. I would run the MCTS algorithm for 30 seconds (the time the algorithm is run for the hard AI) and I would experimentally determine what tree depth on average results in an approximate minimax runtime of 30 seconds. Then, I would play a 30 second MCTS AI against this minimax opponent for the win count testing. If one of the AIs had significantly more wins than the other at the end, it would likely indicate the better algorithm. I would also improve my evaluation function when using minimax as while a strong evaluation is not

necessary for MCTS (because the vast majority of my simulations end in a terminal state), the evaluation function is far more important in minimax.

To improve the evaluation function, I would test other evaluation functions and record how many times the algorithm (at a fixed tree depth) wins against a fixed AI opponent in a set number of total games. If an evaluation function led to significantly more wins than the others, it would indicate the best evaluation function. An example of a different evaluation function I would test is where pieces on different board positions are weighted differently instead of simply totalling the number of pieces each player has before comparing. For example, pieces on the board's 4 corners could detract points, pieces in the centre could grant n points, and pieces on other positions could grant less than n points. Then, the total number of evaluation points each player has would be compared.

A more advanced evaluation function could use a neural network. I could train the neural network with board states and would label each data point with a combined statistic which combines the number of moves it took a player to win from that state and which player won. The statistic would need to be created in such a way that AI wins are determined to be a good label and AI losses are determined to be a bad label. As well as this, an AI win which is reached in fewer moves than some other AI win should have a more positive evaluation and an AI loss which is reached in fewer moves than some other AI loss should have a worse evaluation. Then when minimax or MCTS needs to use the evaluation function, it can pass the board state to be evaluated to the neural network. If the computational requirements for operating the neural network were large, the neural network could be stored on a separate powerful REST server and the program would communicate with this server over HTTP.

5.3 Analysis of user feedback

Below is a second interview with my client, asking them for their feedback on the final project. The points they made have been summarised below each question asked. Improvements suggested by the user are in italics.

Overall, how do you like the software?

- I'm very pleased with the application
- I like the colour scheme and UI very much – it's nice and simplistic
- The AI opponents are great, there's a clear jump in difficulty between each one which is great for my whole family
- Account features like saving a game and user stats are done well
    - I like the look of the UI tables and the fact that only a few stats are stored, making it easy to understand

How does the final project compare to the objectives?

- All the objectives I requested have been met
- Some improvements can be made but overall, I am very happy with it

What are your thoughts on the AI play?

- AI play has turned out really good
- There is a distinct level of challenge between the three difficulties
- Although the hard AI does take some time to move, I am happy with it as it gives me needed time to plan my move and if I'm after a quicker game I can just use the medium AI
- *I think an even easier AI which for example made purely random moves would have been nice as the current easy AI can be a little tricky for a complete beginner who isn't good at spotting captures.*

How well do you feel user accounts are managed?

- Overall, I'm pleased with the account system since signing up and logging in are easy
- Saving games, loading games, and viewing user stats are all easy to do

- *I think it would have been nice to do local play against another logged in user so that both users have their game history updated*

How do you like the user interface?

- I think the UI is well made because it's easy to use, has a simple colour scheme and has a nice customisation feature with changing piece colour
- *The only improvement I would make is to make it so the looped tracks are displayed on the actual board you move pieces on, but I still think the tracks are shown well with the display board*

Do you think the game was made accessible to a wide audience?

- Definitely, through the varied AI difficulties and simple UI, I think the software is a great option for anyone wanting to play Surakarta at any level of experience
- User stats are great for tracking progress
- Saving and loading games are nice as they give more of an incentive to play even if you don't have much time
- *Perhaps the game's unique capturing move could have been better explained with a diagram in the help page.*

Would you recommend this project to someone else?

- I would recommend this software to Surakarta players of all skills levels
- I am looking forward to using this application and sharing it with my friends to introduce them to the game
- I will play the game during breaks at school and on the train and I won't need to carry a board and pieces around anymore

What improvements would you make?

- *I think a hint system would have been nice where the software can recommend a move to the user*
- *An achievement system would have been cool where after a certain number of wins against an AI difficulty you unlock an achievement*

Overall, the user interview made it clear that they were happy with how the project turned out and it has met all their needs. They make it clear that they will make use of the project and are satisfied with it enough that they would recommend it to others. But they have outlined some improvements they would make which range from user interface improvements to new functionality like an easier AI difficulty than the current Easy AI. Because of the good use of OOP in this project and loosely coupled classes, all suggested features which are not about the user interface could be implemented with the current code architecture. For instance, the additional AI difficulty would just need to be another class which inherits from AIPlayer. The suggested improvement to the help page could also be made with the same code architecture but displaying the looped tracks on the main board would require recoding the GUI with the Tkinter framework.

In section 5.4, I explore possible ways of implementing the improvements suggested by the user. I also explore some additional improvements which I believe would be good to implement.

5.4 Possible improvements

In this section, possible improvements and additional features suggested by the user as well as some additional features I believe would be good to add in the future have been described.

Hints – USER SUGGESTION

Hints could be added to the game so that when a "hint" button is pressed in a match, a recommended move is highlighted on the board.

To do this, the medium or hard AI MCTS algorithms could be run on the board from the human player's perspective and the move suggested by the AI would be highlighted on the board or appear as text on screen.

More AI Difficulties – USER SUGGESTION

Currently, the game offers an easy, medium, and hard AI difficulty.

An even easier AI could be added which makes purely random moves unlike the current easy AI's greedy capturing algorithm. An even harder AI could also be added which runs the MCTS algorithm on a powerful server over HTTP so that more MCTS iterations are performed in the time. A powerful server would also allow rollouts to be executed in parallel on separate CPU cores without a performance slow down.

Improved Help Page – USER SUGGESTION

The user suggested that the help page should have a better explanation of captures.

To do this, an example capture would be shown on a board image as well as some illegal captures to solidify a user's understanding of the rules. These images would be accompanied with text explanations.

Achievement System – USER SUGGESTION

The user suggested that an achievement system where achievements can be earned after a certain number of wins against each AI difficulty should be added.

For example, there could be an achievement for winning against the hard AI 10 times. Achievements earned would be viewable in an achievements popup modal window which can be reached from a button on the home page. Achievements could reward the user with certain piece colours or designs on the piece colours.

Additional GUI Customisation

Currently, the user can only customise the colour of their pieces.

Additional GUI customisation could include a light and dark mode or the ability to choose from a set of colour pallets to use for the background and buttons (currently only a grey and green colour scheme can be used).

Displaying Looped Tracks on the Main Board – USER SUGGESTION

Currently, to view the board's looped tracks in the GUI a user presses the "display board" button to show an image of the board state with the looped tracks.

To show the looped tracks on the main board that is played on, the whole main board should be a Tkinter canvas where pieces and looped tracks are drawn. Moves would be detected by clicking pieces on the canvas and using the coordinates of the click to determine the piece selected. This is in contrast to the current setup where pieces are shown as buttons with images of the piece on them.

Local Play with Two Accounts – USER SUGGESTION

Currently, a logged in user can only play locally against a guest player who enters their name. Player 2 is not tied to an account.

The user suggested that a logged in user should be able to play locally against another logged in user so that both players have their match history updated after the match. To implement this, the GUI should present the option to login a second user for player 2 while still allowing player 2 to not be logged in.

LAN Play

The current system only allows a user to play locally or against an AI.

LAN play could be implemented to allow two logged in users on the same local area network to play against each other where each user is using a different computer. To implement this, the WebSocket protocol would be used to establish a two-way communication over TCP/IP between the two computers over which moves, piece counts and whether the game is over would be communicated between the devices. Alternatively, each client device could connect to a central server over HTTP which manages the game state. Clients would make requests to the server to make moves and request the board state.

Ending Matches in a Draw

The current system does not allow the game to end in a draw because a legal move can always be played in Surakarta.

However, this can sometimes lead to stale play if both players each have only one piece and play optimally, causing the game to never end. Thus, a draw system could be implemented where after a certain number of moves where no capture takes place, the game ends in a draw.

6 Appendix

6.1 Git log

commit d4168154c9e08c5cb3779dc56004f863e8236e1a
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Feb 12 15:42:59 2024 +0000

    FINAL COMMIT - renamed database.db and main.py to use a first captial letter. Added coding style skills to comments. Tidied up some of the main event loop code in the GUI and added some comments

commit e1eada230ececa1edf210abca354bd4adc53e895
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Feb 5 22:23:05 2024 +0000

    fixed a bug where after deleting a game from your saved games you could still delete that game and you'd get a success popup eventhough nothing was deleted

commit 1d4001a2ef9490307665a76a9ea2d91d2d2a0df5
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Feb 5 22:06:32 2024 +0000

    fixed bug where the game would crash if you finished the game with the display board out. Fixed bug where after backing out of a loaded game going into a local play match and making a move would cause a crash

commit 4b77cbd17acb911d0cceae01520f40eecb03e4cf
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Jan 29 22:33:44 2024 +0000

    deleted print statements in the TreeSearch.py file

commit fb5dec7a4bc1a19b1cd9af6655d32385ab884ec0
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Jan 29 22:17:40 2024 +0000

    Rearranged some of the constants in GraphicalUI and added a constant for the text wrap length of the help page

commit e09ddaf310b8162baa4ac21d06af9ab6d5a83a41
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Jan 29 22:00:12 2024 +0000

    updated some file and folder names to use camel case

commit a7ce80782c121a9478ac1565d336248483228ce8
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Jan 29 20:32:11 2024 +0000

    fixed a bug where the game would crash after a completed AI match. Fixed a bug where a game wouldn't be deleted after completion from the saved games table/database

commit 0c66f49bd91ddd3a2d2d178707ddca9240c2d51e
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jan 28 21:44:26 2024 +0000

    renamed the Terminal_UI and Graphical_UI classes to remove the underscore

commit ba4cb379c3ff9f8024e67ed495f7f77a1105af16
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jan 28 19:46:54 2024 +0000

    fixed issue where a player could load another accounts saved games,

commit a6830bae47566fd4e7905f9386db02d10f5813d5
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jan 28 13:17:53 2024 +0000

    fixed bug where the game would crash after finishing a match because I was missing a return statement

commit 8cc13f5886bae5070991e439c260bd3065c2c789
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jan 28 12:23:48 2024 +0000

    added player colour to the current player text in the GUI

commit ccd6e7a13f78ba0dfde33d2ce38580949276b27d
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Jan 27 12:12:53 2024 +0000

    added more constants to the Graphical_UI class for window dimensions and button dimensions. Added the remaining skill annotations for dictionary usage

commit 73ba2e7c17a4628ed338f3888cc7f15fffaa4e44
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Jan 25 21:51:25 2024 +0000

    added constants for hashing, added some dictionary skill comments

commit 2b24c0639e7ca602c2f4de5f38edfaf0fa40db99
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jan 14 12:05:53 2024 +0000

    added Hard AI with 30 second time to run MCTS

commit 0c367b37f25d5f04373ebee040788c10ea9a9425
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jan 14 11:51:15 2024 +0000

    moved the initial looped track pointer setting skipping of first location during capture checks to the search_direction_for_capture method

commit 98160f71c64351c3ef8239b8613d312de9e623cd
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jan 14 11:11:09 2024 +0000

    improved the efficiency of capture legal checks by checking for right and left captures individually and returning as soon as a legal capture is found instead of finding all possible captures first

commit 6992d85e8871facba1cb1498e1a8b0de2a9d0020
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jan 14 10:47:24 2024 +0000

    improved the docstrings of some methods. Added a custom exception to Game for when you try to find a winner but the game isn't over. Renamed update_value in Node to add_to_value

commit c6341952b90142f611085b5e352e7c7622adee3a
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Jan 13 15:54:02 2024 +0000

    added skill comment annotations to the code

commit 5d818978a9917943766ad4886ef768c089862ea3
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Jan 13 13:38:02 2024 +0000

    made GameTree methods private and added a test case to the admin account in the database for medium AI dry runs

commit aa22c5ae7b43ac0485a1188833c4f3738ed78b75
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Jan 13 11:05:29 2024 +0000

    fixed loading game bug where looped tracks weren't properly being updated because when loading a game in Board I would only check one loops because I used elif instead of if. Also added a few comments

commit ae75ce49e2a824ec7d6c58d41c6a05b6c4a2d80f
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Jan 12 19:00:22 2024 +0000

    renamed Game's undo_move method to undo_and_return_move

commit 151b27980912f5ddfb3d5765bfed4375bb5ee32d
Merge: 67caf5f f3f9180
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Jan 12 18:55:14 2024 +0000

    Merge branch 'main' of https://github.com/1paleking1/Surakarta-NEA

commit 67caf5f9d93371562cfeb60629c8759c7975a3d8
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Jan 12 18:54:45 2024 +0000

    fixed instance in Board.py where I forgot to use __move_piece_with_undo_arg instead of move_piece

commit 6c85c4e16275e467c19f9fc80a18a4d3884babb4
Author: 1paleking1 <1paleking1@gmail.com>
Date: Wed Jan 10 21:13:45 2024 +0000

    moved some dictionaries that aren't changed during runtime from instance variables to class constants. made the undo argument in the move_piece method in Board hidden from the public interface. reworked get_random_normal_move to just return a random move from the get_player_legal_moves method. Renamed player_lst to player_tuple

commit 47232a028bec2cf9ec803176ae48cbcb552735df
Merge: 2a0df05 0e0b877
Author: 1paleking1 <1paleking1@gmail.com>
Date: Wed Jan 10 20:01:25 2024 +0000

    Merge branch 'main' of https://github.com/1paleking1/Surakarta-NEA

commit 2a0df05bfa7dcbd34cf8a6bbeeb787911039475a
Author: 1paleking1 <1paleking1@gmail.com>
Date: Wed Jan 10 20:00:17 2024 +0000

changed the names of some variables to include the word piece for context and mamde help_page_text file now have .txt on the end

commit f3f918029f11dc0054e190a6f39273735969f318
Author: Sahil <1paleking1@gmail.com>
Date: Wed Jan 10 18:18:14 2024 +0000

fixed bug where undo_move in Board was using move_piece instead of move_piece_with_undo_arg

commit 0e0b8774ca4231e4fafe603e698f1e6850f5368a
Author: Sahil <1paleking1@gmail.com>
Date: Thu Jan 4 14:49:53 2024 +0000

added a __pad_player_name method to UI.py to make names that are shorter than 10 characters be padded withs paces so they don't push the board

commit ba327b9b9784a67676ed7d7d6e09da950306fd7e
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Dec 30 23:11:20 2023 +0000

flipped the board so player 1's pieces are at the bottom which makes more sense

commit 15c7bbfef335ee83adab2fb35a3b6a8a8ec121d6
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Dec 30 23:01:56 2023 +0000

fixed issues introduced with new methods added where ai wouldn't make moves after a capture (due to a return statement in the wrong place) and fixed an issue where piece count messages in GUI were reversed between players

commit f53eb1ad1ab6165fa938442e60756ab8f8f3076c
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Dec 29 22:59:44 2023 +0000

added the Terminal_UI class to the application so you can select terminal mode 't' or GUI mode 'g' at the start. Documented the Terminal_UI class

commit 7765ef427db2de29c631be3e88915311af13b2cd
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Dec 29 19:06:13 2023 +0000

replaced the help page dummy text with actual descriptions

commit 62cce9fbcca1e0bbba4fd69cb6c9faa182217de7
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Dec 29 18:50:42 2023 +0000

added the updated paths to a few more images I missed and also made the folder paths a class attribute constant

commit 0b7f0987446a95487939ebf25e1a30bdfe7a7c0d
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Dec 29 18:35:29 2023 +0000

restructured files to have an images folder and two sub folders in images. Also removed the old code files

commit c6cb4e26c7aca7da2b809b96b802c734807f6276

Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Dec 28 22:24:36 2023 +0000

    went through all the method names in the program and changed some of them to be better

commit 1429f454fba5153084a1597fb8898c120e2c11b5
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Dec 28 20:32:58 2023 +0000

    game no longer crashes after a match is complete (fixed unhighlighting issue)

commit 4e06656233e42fad430299a173b345ff0c6b3bdf
Author: 1paleking1 <1paleking1@gmail.com>
Date: Wed Dec 27 18:19:33 2023 +0000

    the text showing a player name and the number of captured pieces now wrap to new lines so long names don't push the board in a match.

commit a9caf389af3923d8f25396082eab30c4583c00d8
Author: 1paleking1 <1paleking1@gmail.com>
Date: Wed Dec 27 17:54:05 2023 +0000

    fixed a bug in UI.py where you couldn't change your name in non-logged in matches against AI.

commit 0026ff849be3a66da14c55c1596294b737c4f8cf
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Dec 24 19:00:30 2023 +0000

    changed the names of variables that saay 'loop' like inner loop etc and changed them to use the word 'track' so there is no ambiguiuty when refering to looped tracks and the 4 board loops

commit 2d75234a2638907afd67a8aaf1f20bbbb7e26f7f
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Dec 24 17:33:25 2023 +0000

    added the strings 'capture', 'move', 'INNER', 'OUTER' and 'BOTH' to the MultiClassBoardAttributes class to centralise them as constants

commit 761f762a14cda9a1611aabfa938d87477fe3d872
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Dec 24 17:19:46 2023 +0000

    renamed BoardConstants (file and class) to MultiClassBoardAttributes. Moved the constants in MultiClassBoardAttributes that are not needed in multiple claasses (moved them to be class attributes in their respective classes)

commit 21b75ad82133efb30620115cea56ab7e1bc2ed00
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Dec 24 16:22:56 2023 +0000

    Renamed the CircularList.py file and CircularList class to LoopedTrack.py and LoopedTrack. Added comments to the LoopedTrack class

commit 6c435389b016183c51f4c3764780a32642419e4d
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Dec 24 16:09:20 2023 +0000

    added docstrings and comments to the Database class

commit c162f5422c482ad12909d15b78dc92eec69cf55e
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Dec 24 14:44:56 2023 +0000

    added more comments to TreeSearch.py and Game.py

commit f2ef588d34370dfa96de4744d42350326651c4fe
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Dec 24 14:07:24 2023 +0000

    added comments and docstrings to the Board class. Also changed the formaat of the saved game string to not have separator characters at the start and end

commit e85f562bec6d54df47b6e8573d89874c3d017083
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Dec 23 19:30:42 2023 +0000

    removed the old self.__preferred_piece_colour attribute which is no longer needed because I use the BoardConstants class instead. It's better this way as i only have to update it in one place

commit c00bc07bd291af7c4b7247fa66ffde0b74669839
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Dec 23 19:13:22 2023 +0000

    added comments and docstrings to the Graphical_UI class. Also added a couplle of methods for clarity

commit 9e31cdc6d24b4f9544be219dfecaedc38574bf7c
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Dec 23 13:18:31 2023 +0000

    added titles above the game history and ai match stat tables

commit 099d419df5feef2bbad2fd74895ffbd3f62e34bf
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Dec 23 13:16:45 2023 +0000

    a player can no longer spawn multiple display board windows

commit 6a7e7fcd1fd8cbceec2be661587a2913eb02b458
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Dec 23 13:14:27 2023 +0000

    fixed a bug where legal checks were wrong if you change piece colour and load an old game. Now the game properly makes you play with the old colour in the save

commit b01b2f2420366baf66d013fb86189de0ef0250de
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Dec 23 12:34:23 2023 +0000

    added a logout button to the menu bar which logs you out. Also added a change preferred piece colour button to the menu bar which lets you change your piece colour when logged in and on the home page. Added an update SQL command to the Database class to facilitate this

commit 3756a9b7262e8196697403509b914b727e0d0d3d
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Dec 19 18:46:51 2023 +0000

    added functionality to let you delete a saved game from the load game modal window

commit e3644cba6fbab25831f2d802a180a05f42af4ac4
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Dec 19 17:46:07 2023 +0000

    fixed bug where captures were not being properly legal checked from a loaded game. Error was caused by empty spaces being turned into piece objects

commit 2a86a8383dc11ca217ef7cf15ce282be8be45acd
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Dec 19 15:49:59 2023 +0000

    now if you load a game and then save it, it replaces the old loaded game save

commit 825f94844974d7e487311f677c8d9fa830233911
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Dec 19 13:37:53 2023 +0000

    changed the database schema to include the new table names and have user_id for inner joins which makes the system more organised. Made a load game window where you can select a game to load by entering an ID. This required changes to the saving game code as well so that you can now save multiple games

commit 3f2c3dc8f58954e2bfbc73a6698b06b439463ff9
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Dec 18 20:11:05 2023 +0000

    moved some of the code in the main game UI loop to there own private methods

commit 4ce3e6f96ec2a8a07daa80b43afc1d274a381480
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Dec 18 17:55:13 2023 +0000

    added a condition that prevents a user from signing up with a username that exactly matched the AI names

commit 01bd81cc709d4df37b6af23f48c791d6aaf8871b
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Dec 18 17:51:27 2023 +0000

    tidied up code (refactoring, renaming and removing some uncessary code and adding docstrings/comments in Game, GridLocation, Move, Piece, Player and Stack

commit ce84f2e777a72bb027a0515bfd3edf55fda2746d
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Dec 18 16:43:18 2023 +0000

    removed some repeated code and refactored some of the methods in TreeSearch.py. Also added docstrings and cleaned up the file

commit 17263c01ef8e6a08e9fcf385bb268862e64531c3
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Dec 18 15:37:19 2023 +0000

    made a stack class and replaced the list being usedas a stack for undoing in the Game class with my stack class. Also migrated the create_circle method in utility_functions to be in the UI class as a static method

commit 7cfb14d207d672c1d0877e1e30b5e1f482c5cf8b

Merge: 5495b63 2742e80
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Dec 18 15:24:32 2023 +0000

    cleaned up some print statements, made 2 methods private in the Board class which should have been, changed some method names

commit 5495b6309e43d0d79b14789089fbd191c22d729f
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Dec 18 15:21:41 2023 +0000

    cleaned up some print statements, made 2 methods private in the Board class which should have been, changed some method names

commit 2742e802c4d980b72f35b04ba39b27a128c56e9b
Author: Sahil <1paleking1@gmail.com>
Date: Wed Nov 15 09:05:17 2023 +0000

    added new method to Board called get_single_legal_move to make rollouts more efficient. Currently the game crashes on the last backpropagation which I need to debug. I suspect it's something to do with the multiprocessing

commit 45e3d076380d6b497743e82976b4e7c9ac334a1a
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Nov 9 21:44:12 2023 +0000

    removed the legal moves list from a node object so that it only needs to be generated when a node is being expanded, at which point relevant moves are made to board deep copies which are appended to the children list

commit ddf29141e90a55dc33c2b402ca30bc0484d63caf
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Nov 9 17:51:48 2023 +0000

    MCTS now has parallel rollouts working. Overhead costs of parallel execution does reduce the number of MCTS iterations but you get a benefitted trade off with a larger increase in the number of rollouts

commit 70229a04063bb22dbcd132271e95d0734823e740
Author: 1paleking1 <1paleking1@gmail.com>
Date: Wed Nov 8 22:08:50 2023 +0000

    added multithreading for a rollout to be done multiple times. Need to debug to see if there are any shared resources between rollouts that may be the cause of the code working someimtes and crashign at other times

commit fb52b0e217d8a755fe031e07ac4a838e481b7418
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 28 17:34:53 2023 +0100

    further reduced the repeated code by one line in the rollout code

commit bbd92890f37f5879b33d53b7a5d23f394df1bf7b
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 28 16:28:10 2023 +0100

    updated the rollout code to be a bit neater. Reduced some repeated code

commit daea94a7c4dd25b757be38bb0f3154966ae8fa8f

Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Oct 27 15:55:53 2023 +0100

    fixed a bug with capturing legal checks where __get_capture_either_direction was failing early if right was invalid even if left had a valid capture

commit ddf06c4bead6af275abfaa27858b87f838d3034b
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Oct 24 19:26:18 2023 +0100

    fully working --> another table is now displayed showing game history

commit f7749757380b9b9f2f156eecbddc73d270bcea63
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Oct 24 18:47:18 2023 +0100

    loaded games now properly display the number of captured pieces for both local and AI play. Database has new fields for player piece counts in the saved_games table

commit 4f515aadc5488a0421b6a41e7e8f8fe8800d8fe5
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Oct 24 17:13:22 2023 +0100

    player2's turn is now displayed if the loaded game starts with player 2

commit 29435800fd4eb18adb8b2cd99a7a71436d01402f
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Oct 24 16:25:26 2023 +0100

    reverted back to a one game saved per username system. saved_games table now doesn't have game_ids and uses the username as the primary key. If a game is already saved for a username, they are asked if they want to overwrite it.

commit 9a89f480f40d844eaae9bdbdd74fa83e34d25e65
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Oct 23 23:16:25 2023 +0100

    table now properly displays the games available to load

commit 41d4e03fed807177d6bbc4735226059004dfac48
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Oct 23 22:49:56 2023 +0100

    fixed a bug so a logged in user with no saved games that tries to load a game receives an error popup

commit f95f416f8639c746b9ef485f22b1589e69d00b6d
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Oct 23 22:45:52 2023 +0100

    loading a game now works for one game. Meaning you can save multiple but when you load a game it will load the oldest saved game

commit 89cd1eb74d21ace91b4a2544f464dd0c0d0b9eca
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Oct 23 22:13:31 2023 +0100

    moving some constant values to BoardConstants. Debugging Loading Games

commit cf9f899934389b367724773a09138fdc7234d67d

Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Oct 23 20:24:15 2023 +0100

    added SQL methods in the Database class to retrieve the saved game info and am now working on how I will edit the loops to have the correct pieces

commit 47a8c24fe42db791c7c3d7ddf111a3e372b7968b
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Oct 23 18:19:29 2023 +0100

    separated the game restart and added a save game button in the menu bar to a separate menu tab. The save game state button results in a serialised string representation of the match being stored in the database

commit 7832545045d02c37ad73bd665a3699d36cc91ee0
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 22 23:46:38 2023 +0100

    made it so when you create an account, the database class' add_user method will also create relevant records in the AI_game_stats table set to 0 for wins and losses. The game now updates your win/losses against AI correctly and can show you stats in game

commit 669ebef1230757986d174e0a2275a78afadfa2d3
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 22 22:36:38 2023 +0100

    added code to update user stats when logged in after completing an AI match. For some reason the code crashes though when you win/lose so I need to fix this

commit c09537b192dc1a68be3cf18fc99ad42fa08d7aa5
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 22 22:12:11 2023 +0100

    if you are logged in, your username is used as player 1 and the player 1 input fields are not displayed.

commit 224325f3d46c69b830759b837c9a7927c35a3403
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 22 21:08:04 2023 +0100

    changed pysimplegui theme to Dark

commit e34ee77b50c82aabcb1d38701ed04626d415c077
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 22 21:03:16 2023 +0100

    added a query to Database class to get AI game stats and added a table of stats to its own window

commit 650c412d519f75940706f3098bc4c5e62d2bffba
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 22 14:27:17 2023 +0100

    working feature --> you can now use your accounts preferred piece colour in game

commit 075e9d810880bb37fcda3d872ebac6f5bda6baa5
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 22 14:06:48 2023 +0100

    made the player colour constants now variables and made relevant changes in other files

commit 6b5851ed95a171d4e76412fa7dd545de41c0bfdc
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 22 13:32:24 2023 +0100

    added new files for new coloured counters and changed the code to use e.g. yellow instead of just 'y'. Now I need to figure out how I am going to use different colours in practice because I can't just change the constant value of colour in BoardConstants as this is bad practice

commit 699fbee580055e54d9cd2ece2766e26be6c52e22
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 22 12:14:40 2023 +0100

    added menu bar button to show the login status with a popup. e.g. to show that you're not logged in or that you are

commit c1f67ead5d52dd98128f8eb6314b29e9bb3769cf
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 22 10:57:53 2023 +0100

    removed global rank from the database as this can be derived from sorting queries. added a drop down to the sign up page for preferred colours.

commit 3378b2144367efc18fcd00366b99d304a8f4c8e6
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 21 23:12:37 2023 +0100

    password hashing checks now work

commit ebdd2ec63a26398b2b3d5c67bf320242c97930d7
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 21 18:42:40 2023 +0100

    added initial code to hash passwords before storage. Need to now add salts

commit b53130d5569348a9826cbf2eb5c9dc3cade5ce6e
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 21 18:09:56 2023 +0100

    updated GUI for sign up page and login page. Also added methods to the Database class which are add_user and check_if_username_exists

commit b32451a36b01ee64a51d344e43bd8854f2ae2388
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 21 17:20:20 2023 +0100

    table creation methods made in the Database class

commit c428ff48d9bcc861c660db917eab7cc2b798842c
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 21 11:07:12 2023 +0100

    started on the login modal window. Now displays a username and password field

commit bdc46fc978d2a6d62af99b950f5999376dec4e1d
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 21 10:42:11 2023 +0100

    made new folder for stage 6 of the project which is the user accounts

commit 2366c38ae339a7fe2d3e5905e61596752fab78d8
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 21 10:40:56 2023 +0100

    exit button on the home page now works

commit ea652c8739d9741d0a240a22c2cd9dd9bb362cea
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 21 10:37:32 2023 +0100

    fixed a bug so you can now enter a name other than player 1 when playing against the AI

commit 3be34954c81284d3500cea22a1761d9109d14ebb
Merge: f543c3c d83f63d
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 21 10:36:43 2023 +0100

    Merge branch 'main' of https://github.com/1paleking1/Surakarta-NEA

commit f543c3c8885994a9cd2d1147084d5427e95e40b1
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 21 10:35:38 2023 +0100

    fixed a bug so you can now enter a name other than player 1 when playing against the AI

commit d83f63d4c7f84b5d79534e105a4fdee518252239
Author: Sahil <1paleking1@gmail.com>
Date: Mon Oct 9 11:18:48 2023 +0100

    added new home page buttons for sign ups, logins, settings

commit f990651f5afc6576b5a396edc7d5e03199516242
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 8 22:32:55 2023 +0100

    renamed self.__window to self.__main_window and made the display window an instance attribute

commit 1e0df02b0cc0cde34a2038ddf7e4de8e5b3b3fa0
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 8 22:20:56 2023 +0100

    display board now updates when a move is undone

commit 4ff9aa878fe7e72a27aa5eca73d97bb7a4a8c3c9
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 8 22:16:57 2023 +0100

    fixed some issues where you couldn't play the game after closing the display window

commit 41b4bb04f5dbaae7fd7c0e31ed2c43399216e794
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 8 22:10:10 2023 +0100

    display board window now shows the image and can be kept open while playing the game and will dynamically change. Now need to make it update when you undo a move

commit 464fd4cccc7abdc9a14997afedc9b4103eb448d1
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 8 21:36:48 2023 +0100

working on transitioning the modal to now be a window that's always on top (can be toggled away) but also allows you to play the game at the same (i.e. not a modal). Somewhat works --> need to add the updates after each move. Am currently trying to work out why the image isn't being drawn on but the piece are

commit 458c19331951b6cb6a43af078543e0967b3cf8e1
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 8 20:48:33 2023 +0100

the display board window now shows the current game state

commit e364c3e1b73143e790a611d1d736c7a7a9ded293
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 7 18:42:56 2023 +0100

worked out the pixel values for the top left board piece and the piece pixel spacing to place the other pieces on the display board

commit 49f098a2832ed106bfebf13cd808e74519ce76fa
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 7 18:22:22 2023 +0100

fixed an issue where my __is_adjacent() method in the Board class was not working as intended

commit cc4711f8a654aa1ed460c65a654401b94008b159
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 7 17:49:58 2023 +0100

centered the board image in the canvas and made the close button bigger

commit 0612827eff0177b4b15300e89763cedf18cd7bb5
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 7 10:56:47 2023 +0100

did some tidying up of the modal window for the board display and made the button to display it the same format as the undo etc buttons

commit 875fcccd4b1f5ce8a372bbf2d85b920ec1c323fe
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Oct 7 10:50:34 2023 +0100

modal window now can appear and disappear with no errors in the main program

commit addbd5d12159bc45ec158c861e3f47315a5769dd
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Oct 6 20:27:45 2023 +0100

made the window a modal and am debugging why both windows close at the same time

commit 57069941ab90cbd86b7966a849c45a681df63191
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Oct 6 19:53:17 2023 +0100

added display board window code used in test file to UI.py

commit c37367777aba1b7a72baf0d914dd3b5aebb1fdf0
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 1 19:26:26 2023 +0100

added test code to the test.py file for a canvas with a display board

commit 98a84175d3d3aad5fa006e3c0a2d7624ad26d91d
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Oct 1 19:04:29 2023 +0100

did some tidying up of MCTS code and compacted some of it into two more methods

commit 0f83a2212a6a710c387cb0b49802fe2a3fadadd6
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Sep 30 18:45:17 2023 +0100

fixed issue where the first two moves in a rollout would be of the same colour and made sure each rollout starts with the right piece

commit f8bdfd5ba20ead768c1f193706a37e9c3a48fde9
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Sep 30 12:21:40 2023 +0100

fixed an issue in the node expansion method to ensure the right player turn is being used.

commit 8fc8bd34e503bac7d014d929f81e0af4f1f7eb9a
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Sep 30 11:28:36 2023 +0100

MCTS now pretty much works fully. It will make captures that are good but will be smart and not make captures that don't leave it out on top (because if the opponent captures you back its not always the best). To tweak how good it is I can alter the exploration constant and number of moves per rollout. Need to improve the algorithm in the endgame (potentially with an open book) because it doesn't do too well in tricky situations at the very end of the game

commit 8bbc6bbc38ce55d270fc207a71ec299521b76f02
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Sep 30 10:43:45 2023 +0100

reverted change so that you can technically rollout terminal states but instead they instantly evaluate to a win/loss

commit 67c0182118616b604dffdcc6c55f817b1fa13f5a
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Sep 29 22:54:39 2023 +0100

added a tree depth attribute to each node

commit 7ff8042db1b0329a613984574e70b61ba3b81f9b
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Sep 29 21:43:09 2023 +0100

added a check in the selection stage to prevent the selection of terminal states

commit e2ca52876d0e064cda9988c0885c0a6dffe73a46
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Sep 29 21:32:00 2023 +0100

fixed an MCTS bug where the game would rollout terminal states when near the end of the match. Fixed by giving terminal states a visited value of math.inf and adding a condition to the UCB1 assignment method

commit 87a9a6f9267ee9c9b1d2ffc16e946f346d8d7f76
Merge: 17bd146 4e95c21
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Sep 19 22:04:27 2023 +0100

    added code to the test file to debug the MCTS code

commit 17bd146763322aba92eff3eacb20f93a12d93262
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Sep 19 19:30:39 2023 +0100

    MCTS progress --> rollouts now alternate between players and I have seen them end in both wins and losses for the AI. But the AI still only moves the same piece

commit 4e95c214e12008e7198532def8a9fb0da145482a
Author: Sahil <1paleking1@gmail.com>
Date: Tue Sep 12 12:29:28 2023 +0100

    started adding methods to switch player turn in the MCTS class

commit 14bbb25d2b2b7b2ea9181ed5737468a09b7f48c1
Author: Sahil <1paleking1@gmail.com>
Date: Tue Sep 12 08:43:50 2023 +0100

    I think I've found the problem. I need to also simulate opponent moves in the rollous and consider moves from the opponents perspective when expanding all possible moves so research this

commit 9a9f15956e0ecdae33519f8f644a3a5eb523bfbc
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Sep 12 08:06:42 2023 +0100

    legal captures can now be detected but overall there are still bugs

commit 6535cf484c7110c1e615ff9e63a2c36e7552da7d
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Sep 11 21:55:44 2023 +0100

    issue diagnosed to be that in a position legal captures are not recognised as possible moves which means the Board class's get_legal_moves method needs fixing

commit 72ef5eeada900c0ddec735f0bb6f403a278da9fb
Author: Sahil <1paleking1@gmail.com>
Date: Mon Sep 11 09:42:37 2023 +0100

    added an is_hint parameter to Node so that MCTS can be called from the perspective of player 1 for a hint

commit 6a5c2889589f884a1e9cb8c94360250b36652056
Author: Sahil <1paleking1@gmail.com>
Date: Mon Sep 11 09:39:09 2023 +0100

    MCTS now 'works' but for some reason it only moves the same piece until it is captured so need to debug the code

commit 33d4a430e7cc2948369f659619b96b8076fd5280
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Sep 10 18:01:42 2023 +0100

added a general AIPlayer class to make get_move an abstract method for AI classes and used the GameTree class in the medium AI's get_move

commit 9b328bb9124b4307605eb337e54c7082eae80994
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Sep 10 17:37:36 2023 +0100

MCTS classes should be finished although they are untested

commit a037e7598631138c6a79f38bc6ebf2e6eb8c7bdb
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Sep 10 17:11:27 2023 +0100

wrote the backpropagate and rollout methods of the GameTree class. Now adding a timed iteration so MCTS runs for a set amount of time

commit 4285081c4780066c80b8f276987b1319cbdf1495
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Sep 9 11:52:13 2023 +0100

wrote the get legal states from a board in the Board class and added it to the MCTS

commit 6c6a15765456d66677968a34e27dd267111af973
Author: Sahil <1paleking1@gmail.com>
Date: Thu Sep 7 08:37:04 2023 +0100

made the title bigger

commit 2bffec1c881ae047a184a146b62ae75d3b98439a
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Sep 2 20:33:19 2023 +0100

made the node expansion method

commit 7222e47459520274ff8760c7ef09bae91b1cd9ac
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Sep 2 17:49:14 2023 +0100

more work on the tree classes to get a game tree working

commit acceb04812a87da04167c3f0257c27435695b5b7
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Sep 2 17:35:24 2023 +0100

preliminary work on the tree classes

commit af50082ee4ff1567f06f714177974cbe77e37c77
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Sep 2 16:55:42 2023 +0100

copied files into stage 5 tree search AI. KNOWN ISSUE TO FIx: another game can't be played after one is completed will fix this

commit 66f4842bced2a61076ec20134921a565296c77b4
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Sep 2 16:39:14 2023 +0100

new counter images made and put into folders for use later on

commit 54dbb7532161d9bfc44bbefdfdf4d8cbcd4ca98e
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Aug 29 19:21:15 2023 +0100

    working on changing the button piece graphics to include the loops

commit b04486c14fc621009f0115ed8768c5424fe78e77
Merge: 545ae0f e0219b6
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Aug 28 22:20:07 2023 +0100

    improved EasyAI by making it always look for captures first and then making a corner moves only when no captures can be made. Improved the board shuffling function by using a new flattening function for a 2D array in utility_functions

commit 545ae0fb849de376d1986ae08e86b820cac572c5
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Aug 28 22:11:08 2023 +0100

    improved EasyAI to always capture and then make corner move. Improved shuffle to shuffle between rows too aided with a new function in utliity functions which flattens a 2D array

commit c1974b21f0b1e90cf0d0f36025914a85d89e7874
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Aug 28 21:26:00 2023 +0100

    undo code now fully works on both AI and local play modes

commit e0219b610508d841db5e0785649f601aabeabae3
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Aug 28 21:26:00 2023 +0100

    undo code now fully works on both AI and local play modes

commit c5f606794a215dfc25227206ea56e7f2da5d79e2
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Aug 28 18:30:51 2023 +0100

    undoing captures works again with Local play but not with AI. Trying experimental change where I make Board's move_piece method take a move_obj instead of other parameters

commit d44a78fd07105b3e7263c90170c11fe236c19034
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Aug 28 17:15:13 2023 +0100

    loops not updating properly bug after captures fixed

commit 335f6c3da65980d764c23b5b3efd314a646c263a
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Aug 25 18:21:06 2023 +0100

    currently debugging issue where the loops are not properly updated I think after captures

commit e90ff6a4649cecbf36412824d41d9aadd5b8f3ba
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Aug 25 16:56:03 2023 +0100

    undo button works for both AI and local play for captures and normal moves (fixed a series of UI aand board update bugs

commit 3ab473375828eafa157c9fa95d32e7c5969e56d4
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Aug 25 15:54:20 2023 +0100

    changed the game's get move method to also return a move object which is now used as part of the undoing. Undiong AI captures mostly works but there are a few bugs

commit c24f0c38b636d4d7ef3f8cfccb778fc926041292
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 24 15:32:54 2023 +0100

    currently trying to get the undo to work on AI captures and will try an experimental change on the Move class so potentially fall back to this commit

commit e9d43190404872837863b63abf5d85838fa2c29a
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 24 13:18:50 2023 +0100

    added aa 0.1 second delay to the AI move to make it feel more human. Also changed it so the Easy AI only spots and makes captures 50% of the time otherwise it's too good

commit 896b114d466181384fb654441c247c84668db876
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 24 13:13:15 2023 +0100

    piece counts are now correctly updated when the AI makes a capture

commit 94766c3444dd0fb666e0a3046405a1e6a8b78ab7
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 24 12:32:12 2023 +0100

    Easy AI logic now works with random moves, captures and moving out of the corners. Next I need to make sure the easy AI piece counts are updated and and improve the board shuffle

commit 0ebdbb939b23cf1425d713cfe96092775c7310aa
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Aug 22 18:27:41 2023 +0100

    program runs in AI mode but has some wierd bugs which I need to fix

commit d613ebb4469029dd7df961e89abef3a576c502f1
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Aug 22 18:08:47 2023 +0100

    move making functionality for AI implemented but the player methods don't work so I need to debug them

commit ad9f98256acfca153bd467d4cb646188c5de41bd
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Aug 22 16:21:28 2023 +0100

    get_move method in EasyAIPlayer class written but not tested. Also the code needs to be better structure in this method

commit 5ec366a0c11e44d6b3ada88b02de718f8f801418
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Aug 22 15:51:25 2023 +0100

code written for the get_capture_with() method in the Board class used by EasyAI. Not tested it yet. Also an additional parameter to __can_capture_either_direction had to be added and a couple tweaks to keep other things unchanged

commit 2ea78637ec8b59df5b815ea6212728c2ce2e4bdb
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Aug 21 18:14:18 2023 +0100

created new AI player classes and made relevant additions to the Game and UI class to enable a game to be booted up in AI mode

commit 28854c3beef95f8ea99d42db9cadbce113f9d73a
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Aug 21 17:39:31 2023 +0100

copied files into new folder for the easy AI stage

commit 4a9bc41317b8166668d8a9dbc1ad5e2c1694043c
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Aug 21 17:37:16 2023 +0100

added quit button to the menu bar

commit 4eec79ecc367ef68887538221c32d979c796ad51
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Aug 21 17:16:24 2023 +0100

added error popup if you try to restart when not in a match

commit c74ffd2b058aecce67c1e8f3e740d356f3b23415
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Aug 21 17:13:01 2023 +0100

restart button added and working to the menu bar

commit 31a8dd4616f935ad1efa326dc1e0144fa7d52e62
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Aug 21 16:55:37 2023 +0100

minor changes to Ai input and Local play inputs in the GUI where they are now columns instead of frames

commit 30788389c33adfb1e7fc9c4d7e7b8c8bf8cba5b1
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Aug 21 16:35:28 2023 +0100

undo button now fully works for both captures and normal moves. The number of pieces is also updated

commit 54b6f8bda882b0d94474bab4ef974c51e133466a
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Aug 20 16:08:51 2023 +0100

working undo button for normal moves but the current system won't work for captures so I'm going to have to change the Move class to store the start and end colour

commit 863fc92aee8a6b26732a6d6048f249908e59751a
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Aug 18 21:55:45 2023 +0100

working on the undo button code

commit ae1e3132981a39fc925e3f9eb8d260b6d91471c5
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Aug 18 16:40:29 2023 +0100

created the Move class and added a stack to the Game class to facilitate an undo button. Now need to program the actual undo-ing. Also added the undo button to the UI.

commit 416ac406b34ec911a31dd4485508e217964285c9
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Aug 18 15:54:23 2023 +0100

copied code files into Full Gui

commit 34dbdf5423ca845f473d12f258063e02190827c5
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Aug 18 15:49:40 2023 +0100

deleted some old commented out code

commit aefabc85daa3f9db33fe82720892819b81e7547b
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Aug 18 15:46:40 2023 +0100

renamed the __get_loop_from_text method to __get_common_loops and re-coded it so now it only returns a tuple containing the common loops between the two inputted text loop representations. This means I don't need the self.__both_locations_same_loop() check before the board_loop_tuple line. Doesn't seem to be any bugs from this but check this commit if needs be in the future

commit 9175550832319348364b6cefe0d5af708d9dfbf0
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Aug 18 13:43:44 2023 +0100

replaced the repeated accessor methods in Game with more generalised accessor methods that work for both players

commit bf9175d3d1ab6e7bf2cba0d36f578a0a0ddc869a
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Aug 18 13:36:58 2023 +0100

made a few small changes to get_winner

commit 2f0f6b4809adc32f3042198edd5a09481c26d103
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Aug 18 13:21:31 2023 +0100

added optimisaation where you only check if the game is over after a capture (initially thought this wouldn't work but it does. Also moved the number of pieces to the Player class in the GUI stage 2. And I added a piece counter for the number of pieces each player has captured

commit e28a8cb4079317be2c9b30289b8854625a9ddd8d
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 17 16:16:46 2023 +0100

renamed folders and made new folder for the full GUI (stage 3)

commit c2c938f0b12fa3e062077d7b084e2706228f8ce4

Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 17 16:11:45 2023 +0100

    turns out you can always make a legal move in Surakarta so the checking every piece for legal moves was not needed so I got rid of it for both the MVP and basic GUI

commit 11a021c52f1eaf8ebef40bf04fd46b0f71f7736d
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 17 14:07:11 2023 +0100

    added an optimisation where the GUI class only checks if the game is over after a capturing move

commit 18122157f844a27bc14de5517a483922cc270648
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 17 14:01:16 2023 +0100

    the home page button in utilities (menu bar) now works and closes the old window. After a game is over the program returns to the home page

commit 15ed936568d12c901980ee7f09c703ff3f49b896
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 17 13:43:24 2023 +0100

    Both the MVP and Stage 2 GUI work as intended

commit 23837fa93eb6905956c182ecb1187d91e5d5e1e4
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 17 13:29:38 2023 +0100

    added some board constants and changed method orders and added docstrings to Graphical_UI class. Did not go through the methods that make pages aand have layouts will do this another time

commit 99a3a9c69470793b9fceee2a77c209c46eabd3ee
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 17 12:40:49 2023 +0100

    finished going through and improving Board.py

commit 6184fe177d0291afb0b13a807bd57c768ea700b6
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 17 12:10:41 2023 +0100

    same code as previous commit but with better message. Have gone through files to add constants and make changes to improve code. So far I have been through CircularList, Piece, Player, Game, GridLocation and and currently going through Board. I am up to the __can_capture_either_direction and just improved this function by adding another function called __search_direction to reduce repeated code. This commit does also delete the unused method __is_valid_capture_either_direction

commit 7debc1992c01e146263582f128d8de4aeb41480d
Merge: 41eef7a ae4c89c
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 17 12:06:12 2023 +0100

    resolved previous merge

commit 41eef7a5fe57ecec061a50d863a1b1033de3ae66
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 17 10:03:59 2023 +0100

GUI program can now play a proper game with legal move checks etc and a player can win the game

commit ae4c89cd1ad20aa17e4a2795fc76c367bee651e9
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 17 10:03:59 2023 +0100

GUI program can now play a proper game with legal move checks etc and a player can win the game

commit 9ee5653c9dad79e92bc8c5fd17a23dfbecc1dd1c
Author: 1paleking1 <1paleking1@gmail.com>
Date: Wed Aug 16 15:50:24 2023 +0100

moving onto fixing the

commit 461dfb7adae393d32b75a6ce812ade095e4be738
Author: 1paleking1 <1paleking1@gmail.com>
Date: Wed Aug 16 13:53:24 2023 +0100

deleted some unused files

commit dd543ca40cff483c960b5c58dc5e86cdfcb31b2e
Author: 1paleking1 <1paleking1@gmail.com>
Date: Wed Aug 16 13:52:43 2023 +0100

updated the mvp board.py to use the new fixed capturing code and updated it's CircularList file.

commit 632a10b1e7d479fdd533a79431112d2ea0dfa9b9
Author: 1paleking1 <1paleking1@gmail.com>
Date: Wed Aug 16 13:49:05 2023 +0100

diagonal capture bug fixed

commit d2ecacff37538405ec712e97dd9b551e0d0285c9
Author: 1paleking1 <1paleking1@gmail.com>
Date: Wed Aug 16 12:33:55 2023 +0100

most of the capture bugs have been fixed (working in the GUI folder so will copy over code to the terminal version folder after) except for some reason you can now capture diagonally?

commit ff8936c9b60bd22e32d4ed4ea7b25e98a93c5280
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Aug 15 19:53:39 2023 +0100

moved pieces which are not captures are now properly updated in the loops. Still need to debug captures and also tidy up code

commit 821a83b67fa9acc95e9e303936c2c73d0262f026
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Aug 14 15:21:02 2023 +0100

copied the MVP from an older git version which works. I think when I made BoardConstants and renamed some methods something broke along the way so thankfully version control exists

commit d0aefdeec1f12bd544a16a2fb5713d9e1de5268e
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Aug 14 14:30:38 2023 +0100

currently debugging accidental changes to MVP

commit 505e3efdf4d9346b8c10abff8963845231d0bf31
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Aug 12 23:31:32 2023 +0100

    The program now displays the player whose turn it is and alternates between them

commit f7ed54b056103cc2878cad909a1a8f9a5cfae50c
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Aug 12 21:39:15 2023 +0100

    highliighting of positions clicked on the board now works

commit 1f5ada43370c2d68a3911af11f5e2a8b0e4eda24
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Aug 12 21:06:12 2023 +0100

    spacing fixed between board pieces in GUI

commit b62ac1f6139420792fcbb799f21d0b515c472b49
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Aug 12 18:48:17 2023 +0100

    fixed formatting issue with a new blank counter image where the counter is the same size but the overall image space is the same as the other counters (42x42 pixels)

commit 8b8e44ea8bf1958cac87b83e977e254f57ab696b
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Aug 12 18:39:07 2023 +0100

    partially fixed. The issue was when the button was created I was specifying an image size which is why the buttons had lots of space around the image but when the image was swtiched it would revert to a normal image size. So the fix was removing the specified image size in __make_piece_button. But now the spacing is off which I need to fix

commit b06502d0e58dfc184fe66017ae896ca2fca1bc08
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Aug 12 18:00:06 2023 +0100

    working on new board single button system for some reason the board now won't display so this needs debugging

commit 5b9287779e8776c8252f0ec6874a7b68c918af8c
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Aug 12 17:49:15 2023 +0100

    still facing the formatting issue but have an idea of a solution. Instead of swapping button visibilty in and out, I'll just update the image used for a button (so its actually the same button each time

commit 010c538f7cd3ddf0f71c2e0b605bccc6ff927c45
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Aug 12 17:29:00 2023 +0100

    moving pieces and displaying the updated board sort of works now but the formatting is all messed up

commit d8902f9ad5738465484ea98ab31b3c128bc8a96f
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Aug 12 16:38:43 2023 +0100

    last commit didn't include all 'G' changed to 'g' but this commit does

commit d6aa4a1d2dc17f4a99a1d76699a3a0d2aafb7dc7
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Aug 12 16:36:46 2023 +0100

    working on getting the board to update the game state in the GUI with a new method __update_board. Note that since the last commit all capital letter representations of counters like 'Y' and 'G' have been changed to lowercase. Will now try changing is_empty in the GridLocation class to actually check for the string 'blank' instead of None to help with the GUI

commit c21944b085a012843c9e21bf70c2cd157d9c145a
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Aug 11 17:55:29 2023 +0100

    ITS WORKING NOW THE BOARD IS DISPLAYING WITH THE VISIBILLITY SYSTEM

commit 541c072db3f6c83fe5592f190fd355f199796be9
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Aug 11 17:34:25 2023 +0100

    almost fixed the problem where you can't see the buttons with the new system where all buttons are present but only some are present depending on which counters are on the board. Working with test.py because for some reason it works when I don't use a function

commit 23ff27af93b2cfcbb47efe09e16bd2c08a8c84bc
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Aug 11 14:12:10 2023 +0100

    move legallity checks now work one way for normal moves. Inputted names are also displayed

commit 1ccf8086f8a1255d2f18261aecfa9dac5331db57
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 10 17:02:24 2023 +0100

    now testing to try and get the moves back from the GUI and test if they're legal

commit afdad468b901cf05d8f44d1902e1679de11261bf
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Aug 10 16:52:38 2023 +0100

    added buttons for submitting and radio buttons for move types in the GUI. Also added toggling colour functionality to select a piece

commit 0399e177d76579062b5d18895d510957ced2ebe1
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Jul 31 00:25:32 2023 +0100

    GOING ON HOLIDAY READ THIS TO GET BACK TO SPEED. I am currently trying to set the blank_board image as the background with the counter buttons ontop. This is only possible by having a transparent window with the counter buttons and a window with the image behind it. Trouble is tweaking window alhpa values in sg.Window() affects the button transparencies and using the transpaarent_color attribute of Window() makes it a click throug window. I'm looking still for a solution to this which may lie in tkinter by using window.Tkroot and accessing the tkinter. Or I may have to make the second background window a tkinter window instead of sg.Window() because tkinter probably supports this.

commit fb5fff98f2ac9481a8d9cd4b2b518b294df162e2
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jul 30 23:32:48 2023 +0100

counters and blank spaces are displayed. Currently working on getting the lines in

commit 5ea6ab3f6c357ae167ac58f10448904f0d9cfbf1
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jul 30 19:48:18 2023 +0100

started working on the actual game board being displayed in the GUI

commit bdcc41bcdef7a55ed0c55e7cc381f411d90dd7fe
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jul 30 15:22:35 2023 +0100

started working on a tkinter GUI but decided to go back to pysimplegui. Have also decided to ditch the smmoth transisionts between pages because you can't have individual justifications because of the way columns work

commit dcacdb99c4c8b4359520565e2c3fd6b4183c8a53
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Jul 24 18:05:50 2023 +0100

initial work on the basic GUI. working now on getting multiple pages using layout visibility

commit 3b12c4733057e86c6fa3237adfbc0470e4745025
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jul 23 22:59:39 2023 +0100

ready to start work on the GUI

commit 7b91d0ffe6ac498f2d9baa654ce088cb562c0760
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jul 23 22:48:58 2023 +0100

moved files into MVP

commit 0fd012f2325893eaa948684b44447405847bb72c
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Jul 20 16:58:24 2023 +0100

added an error message when an invalid move is attempted

commit 2369de343ae8c333e0c14fd0454c8d7a19720a38
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Jul 20 16:46:50 2023 +0100

added a BoardConstants class, renamed a few methods and changed a couple methods which used two if statements to use a for loop

commit cd1fafe016420df75a533c61b26e1259971a6a01
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Jul 18 17:15:58 2023 +0100

fixed an inefficieny as a result of a mistake in the check_has_legal_moves() method

commit 2fd97722f16390c75fc786e4bf27590717bdeeb7
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Jul 18 17:06:13 2023 +0100

nevermind turns out the inner loop works fine

commit 4aea38c1cccf48f2806a0c47c5c65e01567cd2d6
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Jul 18 16:57:38 2023 +0100

    checking if the game is over and displaying the winner now works. Now need to get inner loop captures working

commit d4046c5f1151bfabaf43072854c3a6a4bc9f3422
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Jul 17 20:45:16 2023 +0100

    THE BIG BUG IS FIXED! Captures can now be performed and checked if they're legal. There were quite a few minor issues and some big ones too

commit 504ff3d73082c40a76f8a0e32804197bdb453f8d
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Jul 17 17:00:26 2023 +0100

    still debugging the capturing move legal check. Have added a function to start at a test board state and have made little changes to lots of functions.

commit 8f0bbc630e49e400388c419bd50745f0f13c07f7
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jul 16 13:03:50 2023 +0100

    added an __edit_board_for_testing method to customise the game state for capturing debugging

commit 4ead42843ef513ed7278ffa96744037535e5dd16
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jul 16 12:08:22 2023 +0100

    rotated the board 180 degrees to fix some indexing issues and since it makes more sense with people playing on the same computer. need to fix the set_game_status() method

commit d915b4ad8259dcc4fb56e2e0e286be44c21c1623
Author: 1paleking1 <1paleking1@gmail.com>
Date: Thu Jul 13 16:39:47 2023 +0100

    fixed lots of bugs including one which was preventing moves from being updated on the game board. CURRENT BUG --> the second player can't move because start_loc is given as blue instead of green. I'm guessing it's for a similar reason to the first board update bug

commit a8c1b7f3101ca297ff5ca43c8227bdaf4e2ce1e4
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jul 9 19:26:01 2023 +0100

    changed all the relevant board fu nctions to use locations instead of cordination positions. Now need to fix bug in UI main loop play_game where the infinite loop for r,c isn't broken. Also need to imrpove the names of my methods in the Board class

commit a40a9da06aa8f36791e4f4d6a0d159dcb1b5f236
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jul 9 18:05:17 2023 +0100

    made relevant methods and attributes private in teh Board class

commit 3de0792a78ea99321f667bfb480c1178afcf83be
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jul 9 17:47:16 2023 +0100

main game loop should hopefully be complete now. Finished thhe legal checking method in Game

commit 478ca2d1dacd0f07aa5845a439ffdcdefdc297b1
Author: Sahil <1paleking1@gmail.com>
Date: Thu Jul 6 11:34:04 2023 +0100

worked on teh is_legal_move function in the Game class which calls legal move checker methods in the Board class

commit 772faeb11e894cd62680176969ef4d60bab9b38d
Author: Sahil <1paleking1@gmail.com>
Date: Thu Jul 6 10:15:37 2023 +0100

added validation to some of the UI methods and combined two UI methods into get_cords_from_user. Next I need to work out how I'm going to combine the validation for whether a move is legal into the UI class

commit bf580b601d7a3c2453da1729e9c3ff04bfcb40e2
Author: Sahil <1paleking1@gmail.com>
Date: Thu Jul 6 09:40:57 2023 +0100

added a player parameter to the capturing methods in the Board class

commit f248e00b0d4c8c7f87185f255075f6467305e298
Author: Sahil <1paleking1@gmail.com>
Date: Wed Jul 5 14:40:30 2023 +0100

some work on passing player to board methods

commit 13cbab5142593ad2013ac33092ec6e0dcdd52ad0
Author: Sahil <1paleking1@gmail.com>
Date: Wed Jul 5 13:38:55 2023 +0100

made display_winner method in the UI class

commit 11334e09e0ab07badc5513c3aee0be50955185e5
Author: Sahil <1paleking1@gmail.com>
Date: Wed Jul 5 13:34:40 2023 +0100

board now displays as intended and has row and column indexes

commit f3bda6d0f3ed63b8bc82d30059d9ca795aeccd99
Author: Sahil <1paleking1@gmail.com>
Date: Tue Jul 4 10:30:13 2023 +0100

updated the circular list class to not need a reversed list. updated the display board method in the UI class. created main.py but no coding in it yet

commit 337460f872afd3f3414656a3eef81b55a6b00fdd
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Jul 3 20:52:23 2023 +0100

more work on the UI class

commit ac705ce9d5827142baf390cfd605a0f271d0dd4a
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Jul 3 20:44:17 2023 +0100

working on the UI class. Made a utility functions file

commit ff6c4cf22f6d10ad2c75de26ff39acff746da3df
Author: 1paleking1 <1paleking1@gmail.com>
Date: Mon Jul 3 19:45:01 2023 +0100

for now I think the Board class and get_winner methods are done

commit a2ebbacf8aba866f1104fa0431ac1b597bcc4071
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jul 2 22:31:54 2023 +0100

working on the Game class

commit ec10a2d293866269c459d1832d66211672543e4d
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jul 2 19:32:53 2023 +0100

added functionality for when a gridlocation is on both loops to check_capture_legal and another function

commit be4f2e59b474e5df89549ef9c82a903d61eb3c2e
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sun Jul 2 10:34:51 2023 +0100

work on the CircularList and Board classes

commit 25ce99f31655127075e6ba7544b3050483f96ebe
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Jul 1 19:19:44 2023 +0100

some initial work on new classes and started the move_piece method for just normal moves (not captures)

commit c50561a16264d178f8590cfc886e88ae2b9d2c87
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Jul 1 11:44:10 2023 +0100

wrote TODO for MVP in comments

commit 8b5f2e7149cc5efde261d0f6e2db025cd66dbeea
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Jul 1 11:40:54 2023 +0100

migrated EDGE_CORDS class method from GridLocation to Board

commit 0731c3144d0cf80a298d26a3fdf140cc77ad3c98
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Jul 1 11:39:11 2023 +0100

decomposed the check_capture_legal method fo the Board class

commit 22c21a52430f1eab5240431b0d15ab7d16f24aca
Author: 1paleking1 <1paleking1@gmail.com>
Date: Sat Jul 1 11:01:49 2023 +0100

check_capture_legal method shoudl work now (haven't tested). Now I'm decomposing the function

commit e877864773d8aa43810f5fa3414c0344da119447

Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Jun 30 22:00:34 2023 +0100

    work on the Board and GridLocation files. Currently working on the check_capture_legal method of the Board class

commit ac782ed8fcf8e60b099fd77314e6885a86dc97ba
Author: 1paleking1 <1paleking1@gmail.com>
Date: Fri Jun 30 21:18:14 2023 +0100

    added loop cords variables to Board class

commit f725a0cab117410ee790dd457eb2ff8be90ca9bd
Author: Sahil <1paleking1@gmail.com>
Date: Fri Jun 30 16:30:16 2023 +0100

    deleted some empty lines

commit 6d4cdf82d3251c1c45a82cb88ccd4a55c7c13bb0
Author: Sahil <1paleking1@gmail.com>
Date: Fri Jun 30 16:27:51 2023 +0100

    added the loop index and relevant setting method to the GridLocation class

commit 91bad29a4574ac40337fd1002b94eb31dbc04c73
Author: 1paleking1 <1paleking1@gmail.com>
Date: Wed Jun 28 22:09:04 2023 +0100

    made grid location class and did some other work

commit 337d5dbcf0e51daa32b22a61ac81b41d76b9914d
Author: Sahil <1paleking1@gmail.com>
Date: Wed Jun 28 19:01:37 2023 +0100

    added some functions to the board class

commit de6e15f0693d6e29a1b12f2ba1890beebe5f1fd9
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Jun 27 22:16:01 2023 +0100

    small chagnes

commit fca1d8433cc612987aca4dd51a35639cfec16d7e
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Jun 27 21:40:54 2023 +0100

    created the board filke

commit db8f8456630d0b491c06c14944ed7988d5a51806
Author: 1paleking1 <1paleking1@gmail.com>
Date: Tue Jun 27 21:26:53 2023 +0100

    CircularList class now works with the three implemented methods working as indented.

commit a32a04aaec57b159345777f0ae112aa6d3c8a14a
Author: Sahil <1paleking1@gmail.com>
Date: Thu Jun 22 13:19:30 2023 +0100

    started work on the Piece and CircularList classes

commit 7e84dedf083dafe57933c08db96e58dc3c234dc4
Author: Sahil <1paleking1@gmail.com>
Date: Thu Jun 22 12:45:11 2023 +0100

    first commit test

6.2 References to web sites or other resources used

Citation for the Research paper used to obtain a branching factor of 22:

Winands, M.H., 2015, July. The Surakarta bot revealed. In *Workshop on Computer Games* (pp. 71-82). Cham: Springer International Publishing.

Credit for the create_circle method in the GraphicalUI class:

https://stackoverflow.com/questions/17985216/simpler-way-to-draw-a-circle-with-tkinter

- This website was accessed on the 6th October 2023

Citations for the Existing Surakarta Programs Analysed in Section 1.4

OnlineSoloGames.com (n.d.). Play Surakarta Online. [online] Online Solo Games. Available at: https://www.onlinesologames.com/surakarta [Accessed 10th Jun. 2023].

Enki Apps (2023) Surakarta E. (Version 1.0.5) [Mobile app]. Available at: Google Play (Downloaded 22nd August 2023)

Ömer Ulusoy (2022) Surakarta. (Version 1.015) [Mobile app]. Available at: Google Play (Downloaded 10th June 2023)

GitHub Copilot

GitHub Copilot was used throughout the development of this project for line completion.