**PIECE CLASS:**

This class concentrates on all the sovrano board pieces such as its position on the initial board state, its legal moves etc. This class module is then imported onto the main code.

|  |  |
| --- | --- |
| **FUNCTION NAME** | **DESCRIPTION- WHAT IT DOES** |
| createMove | Will figure out whose turn is next after a piece has been moved and swaps the current player to the next player. |
| undo | If the player doesn’t like the position they have chosen to put a piece, the board can go back to its previous state. |
| validMoves | Makes sure that all pieces only move to a square that is legal to move to. |
| check | Checks if a capture or draw has taken place. |
| attackSquare | Checks if the opponent is able to attack a piece legally. |
| possibleMoves | Checks to see whose turn is next and places the piece is (row, column) notation. |
| getGuardMoves | The guard is given certain moves they can only move to i.e. up, down, left, and right (one place) then returns the new position if the move made is legal. |
| getArcherMoves | The archer is given certain moves they can only move to i.e. up, down, left, right, diagonal (one place) and diagonal(2 places) then returns the new position if the move made is legal. |
| getEmperorMoves | The emperor is given certain moves they can only move to i.e. up, down, left, right and diagonal(one place) then returns the new position if the move made is legal. |
| boardstate | Gives a 2D array representation of what the board will look like with the pieces on them. |

**ARTIFICIAL INTELLIGENCE:**

This part of the code allows the users to play against the computer by calculating their best move to win against the user.

|  |  |
| --- | --- |
| **FUNCTION NAME** | **DESCRIPTION- WHAT IT DOES** |
| aiMove | Using the greedy algorithm, the computer generates the next move. |
| totalScore | Check to see which player has received the most points making them the winner. |
| bestMove | Calculates which move will allow the AI to receive the most points. |

**MAIN SOVRANO BOARD:**

This section is where the main board is created and played on by the users using the imported piececlass and ai modules.

|  |  |
| --- | --- |
| **FUNCTION NAME** | **DESCRIPTION- WHAT IT DOES** |
| loadImages | Uses the compressed png pictures of the sovrano board pieces and loads them into their position on the board( into the 2d array in the piececlass module). |
| drawLines | Creates the grid of the board 7x9 |
| colourSqaure | Uses colour to highlight to the user where they can place their next legal move. |
| drawGameState | Outputs the pieces, the board, the valid moves and square selected by the user. |
| drawPieces | Outputs the pieces loaded into the correct row and column |
| drawText | Outputs a text message when the game has been finished to notify the user if there has been a win, loss or a draw. |
| main | Fills the background colour, loads the images, flags a variable when a move is made so you know who has the next turn, notifies when the game is over, allows the user to quit / undo/ reset, notifies user when there has been a capture/ draw, adds a timer and adds background sounds. |

Decomposing sovrano using a top-down diagram:

Decomposition is the process of continuously breaking down a problem into smaller problems. This will continue until each sub- problem can be represented as a subroutine. This aims to reduce the complexity of a problem by splitting it up into smaller section that are easier to understand. Through identifying a sub- problem, certain parts of code can be implemented using libraries or pre coded modules which saves time from having to debug, which will be implemented within my project.

**GAME PLAY**

.Select piece to play

.Move piece to play

.Output a win screen

.Output a loss screen

.Output a draw screen

.Start

.Instructions

.Leaderboard

.Quit

. Username input

BOARD

MAIN MENU

.Generate the AI’s ideal game play move

.check if the play made is valid

.Update board- state

.Takes users input

.check if the play made is valid

.Update board- state

AI PLACEMENT

USER PLACEMENT

.Generate the AI’s ideal game play move

.check if the move made is valid

. Check if there’s a capture

.Check if there’s a draw

.Update board- state

.Takes users input

.Checks if the move made is valid

.Check if there’s a capture

.Check if there’s a draw

.Update board- state

AI MOVE

USER MOVE

PIECE PLACEMENT

USER INTERFACE

PIECE MOVEMENT

Using thinking ahead to generate all Input, process and outputs:

function boardstate

boardstate (2d array)

self.playerToMove (string)

b\_pos (integer)

r\_pos (integer)

INPUT

OUTPUT

Updates board- state

Outputs board in the given positions

The board- state is updated into the position given by the red piece (r\_pos) and by the blue piece (b\_pos) which determines the next player to move.

The boardstate function will get updated depending on whose turn it is. If the red player makes a move and choses to select a piece as well as select a square where they want to place the piece, the board will make the changes and output the new board for the player. The b\_pos variable takes the position of the blue player and the r\_pos variable takes the position for the red player. The self.playerToMove decides whose turn it is next depending on whose made the last move( or as known as the current player.)

function undo\_move

boardstate (2d array)

self.playerToMove (string)

b\_pos (integer)

r\_pos (integer)

INPUT

OUTPUT

Updates board- state

Outputs board in its previous board state before game play.

The board is updated back to the previous board state depending on the players turn to move. The previous pieces are also loaded back into its previous red/ blues position.

When a player has made their move but wants to undo their current play or reselect a new piece, the user will press ‘u’ on their keyboard to allow this function to occur. The function boardstate will get updated back into its previous game state and the player to move will have another time to select and move a new piece within the time frame.

function possibleMoves

boardstate (2d array)

self.blueToMove (string)

self.redToMove (string)

self.getGuardMove

self.getEmperorMove

self.getArcherMove

INPUT

OUTPUT

Updates board- state

Outputs board in the given positions if move made is valid.

Checks to see which players turn is next and places the piece in(row, column) notation.

Checks that each move is valid for each piece

All players are assigned its valid moves on where they can move along the sovrano board. The pieces are then placed upon the board in the row column notation. This can be tested using black box testing with different variables using coordinates.

function AIMove

boardstate (2d array)

validMove

createMove

userMove

INPUT

OUTPUT

Updates board- state

Piece moved into a new position

Depending on the users move they make, the AI will select the best possible move to make. This is only possible if it’s a valid move.

The AI uses greedy algorithm to make a next move. Once the player plays their piece, the AI will make a move and try to capture a piece worth the most points if possible. If the computer is blocked, the game will lead to draw if the player is also unable to move. Hopefully my AI will be able to think at least one step ahead so the function will work effectively and become a challenge for the player.