**Items in RED are for assignment 3**

## 4.2 – Module Interface Specification (MIS) for public entities

---

First indentation of <> indicates the python files containing public entities,

Second indentation of <> indicates a Python-defined class inside the corresponding python file,

Lastly, second indentation of <<>> indicates a public method inside the corresponding python file.

---

## <main.py>

- Runs the main application; Checkers

**Uses: screen.py, gameloop.py, classes.py**

**Exports:**

**Constants:** **None (top level hierarchy)**

**Types:** **None**

**Variables:** **None**

<<main>>

- Creates the game window;
  - Generates the Checkers board;
  - Allows users to place RED or BLUE pieces on valid locations on the board consistently
  - Allows users to save and load states of the board
  - Allows users to choose whether to face against a computer as red or blue side, or face against another player after determining the positions of all pieces on the board

## <classes.py>

**Uses: Pygame functions**

**Exports:**

**Constants:None**

**Types:Tile,Piece Board classes**

**Variables:None**

<Tile>

- Defines the state of the tiles on the Checkers board

<<__init__>>

- Generates the initial states for individual tiles of the Checker's board;
  - Defines the location (index value) of the tile;
  - Defines the color of the tile (black or white);
  - Sets the tile's state to unoccupied

<<init>>

- Redefines the state of the tile
  - Defines the size of the board;
  - Defines the location (index value) of the tile;
  - Creates the tile through Pygame

<<isOccupied>> : **Boolean**

- Determines whether the specified tile is occupied by a Checkers piece or not

<Piece>

- Defines the state of the Checkers piece

  <<__init__>>

  - Defines the location (index value) of the Checkers piece;
  - Defines whether it is a computer of player piece;
  - Defines whether it is a RED or BLUE player piece

  <<update>>

  - Determines the state of the specific chess piece: Does it fulfill the requirements to become a king piece?

<Board>

- Defines the matrices used to make the board and the number of pieces on it

  <<__init__>>

  - Defining the matrix of the Checkers board;
  - Defining the matrix of the Checkers pieces

  <<update>>

  - Reloads the state of the Checkers board entirely; updates the board

  <<remove>>

  - Removes a piece from the board

  <<move>>

  - Determines whether a chess piece can be moved to a chosen spot on the board. If so, it is moved to the designated position.

## <draw.py>

- The basis of the Checkers interface; generating the interface for the game

**Uses: Pygame functions**

**Exports:**

**Types:** **Tile, Piece, Board Classes**

**Variables:** **Board Matrix, Piece Matrix, Tile Matrix**

<<drawBoard>>

- Draws the Checkers board
  - Draws the tiles of the board (occupied or not; black or white);
  - Draws the Checkers pieces (computer or player; red or blue)

<<drawButtons1>>

- Draws the taskbar of the Checkers game; options interface for players
  - Draws and defines the state of which option is currently selected;
  - Sets the coordinates and size for the options

<<drawButtons2>>

- Draws the “Save and Quit” button on the options interface
  - Allows the user to save the current state of the board and closes the game

<<drawGameText>>

- Indicates the corresponding side’s turn; user interface (ex. “Red’s Turn”)

<<modeSelect>>

- Buttons are provided for user to select whether they want to face against the computer as red or blue, or face against another player

<<winScreen>>

- Displays a small text for the winning side (ex. “The red player wins!”)

## <gameloop.py>

- Serves to keep the Checkers window consistently running and updated

**Uses: Pygame functions, classes.py, draw.py, AI.py**

**Exports:**

**Types: Tile, Piece, Board Classes**

**Variables: Board Matrix**

<<__run__>>

- Draws the Checkers interface
  - Draws the Checkers board;
  - Constantly updates with each new user action
    - Determines whether the user is clicking on the board; moving chess pieces
    - If facing a computer, AI determines whether a move is valid and whether there is a mandatory move to kill the player’s piece.
    - Determines whether the player or computer has won the game

<<placePieces>>

- Places Checkers pieces onto the board or quitting the game
  - Determines which option on the taskbar was chosen using Boolean factors;

        - If placing a Checkers piece, determines the position of your cursor corresponding with the location of the respective tile on the Checkers board;
        - Redefines the state of the board and updates the interface with the newly placed Checkers piece;

<<createBoard>>

- Creates the Checkers board
    - Creates the board using the matrices of the state of Checkers tiles and pieces

<<createDefaultStart>>

- Creates the standard Checkers board with respective Checkers pieces on both sides

<<clearBoard>>

- Removes all Checkers pieces off the board
    - Resets the matrices for Checkers pieces to null

<<modeSelect>>

- Allows player to choose either to face the computer as red or blue side, or against another player.

## <screen.py>

- Generates the window for Checkers

**Uses: Pygame functions**

**Exports:**

**Constants:** **Window**

**Types:** **Window**

<<makewindow>>

- Creates the window for Checkers with a given width and length

## <save.py>

- Contains the load and save methods that are used in “gameloop.py”; the options interface

**Uses: classes.py**

**Exports:**

**Types: .txt files**

**Variables: board class, piece matrix**

<<loadGame>>

- Loads a specified text file “save1.txt” from the designated location; loads the saved states of the chess pieces onto the board

<<saveGame>>

- Saves the current state of the board; the positions of all chess pieces onto a newly created text file named “save1.txt”

## <AI.py>

- Contains the calculations for piece-movement of the AI and turn logic

Uses: None

Exports:

Types: Move class

Variables: None

<Move>

<<__init__>>

- Initiates the AI movement calculations

<<isSame>>

- Determines whether a move is possible

<<allPossibleMoves>>

- Determines all possible moves for all all pieces and returns an array of the moves

<<possibleMoves>>

- Determines all possible moves for a specific piece and returns an array of the moves

<<checkMove>>

- Used for executing the movement of pieces
  - Checks for killing moves and placed as first priority
  - Checks for validity of the move