Team Iconic’s Design Doc

Checkers Game

**For our project, following are the key features and checkpoints:**

1. Project will be implemented using the Java programming language.
2. The checkers game that we are trying to make, will be a single-player game.
3. We will be making the desktop version of the checker game.
4. For storage of the intermediate state of the game, we won’t be using any database. Instead, we would be using simple text files for achieving the same.

**USER INTERFACE**

* The game will have a minimalistic User Interface—the board itself will be the User’s interface for the game.
* A user clicking on their own unit, during their turn, will be how the user interacts with the game.
* After a piece is selected, highlight the valid moves for the user on the board.
* After selecting a valid square, the selected piece will move there.
* After a move, it will be reflected on the board, then an exchange of turns will happen.

Main Menu/UI :

Main menu button will be on the top corner.

**LIBRARIES USED**

java.awt :-

for button canvas, checkbox, label, textfield, etc.

javafx.application :-

[JavaFX is a software platform for creating and delivering desktop applications, as well as rich web applications that can run across a wide variety of devices](https://en.wikipedia.org/wiki/JavaFX" \o "en.wikipedia.org)

Javafx.scene :- The Scene class in the javafx.scene package represents a scene in a JavaFX program. A Scene object is attached to, at the most,**one stage at a time**. If an already attached scene is attached to another stage, it is first detached from the previous stage.

Javafx.beans :-

The package javafx. beans contains the interfaces that define the most generic form of observability. All other classes in the JavaFX library, that are observable, extend the Observable interface.

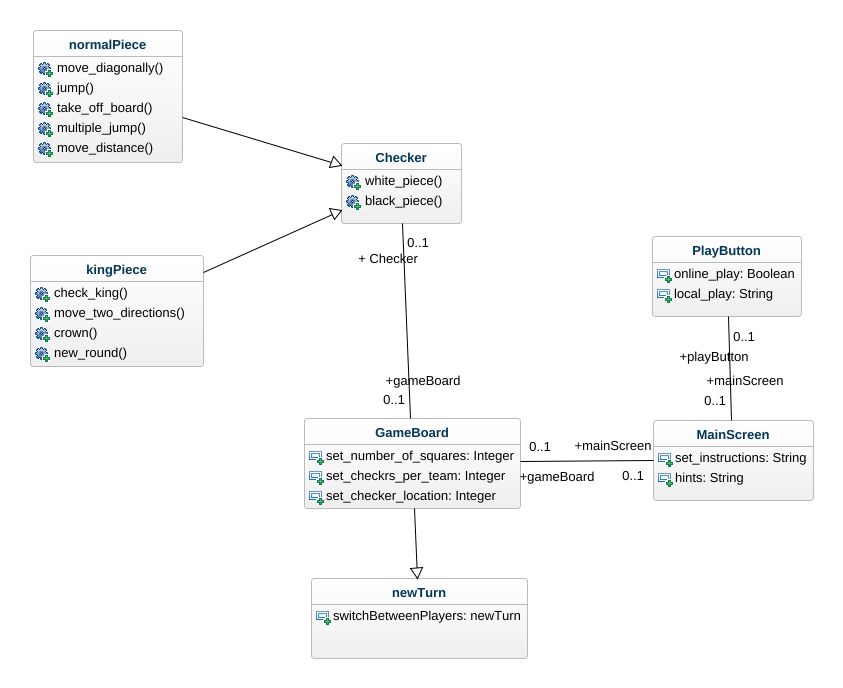
Javafx.geometry :-

Provides the set of 2D classes for defining and performing operations on objects related to two-dimensional geometry.

**COMMON SCENARIOS**

1. Green colour blocks can be used to indicate valid moves for the player to move a checker diagonally.
2. When a player attempts to make a move that is invalid or impossible, that particular block will be highlighted in red.
3. If Player A makes a legal move, then Player B's name dialogue box will illuminate to signify that it is Player B's turn.
4. When a checker from Player A crosses a checker from Player B, the checker from Player B that was crossed is eliminated, and Player A receives one point for the overlap.
5. If any of the player's checker reaches to the opponent's end, then that particular checker will become King
6. The king has the ability to move backwards
7. If Player A wants to save a game then he can save the game.
8. If Player A wants to continue his saved game, then he can do it with the help of the load game option.
9. If any of the players wants to quit the game, then they can choose the quit game option.
10. If player 1 wins the game then a pop up message of "You won" will be shown to the winner screen and a pop message of "You loose" will appear to the player 2 screen.

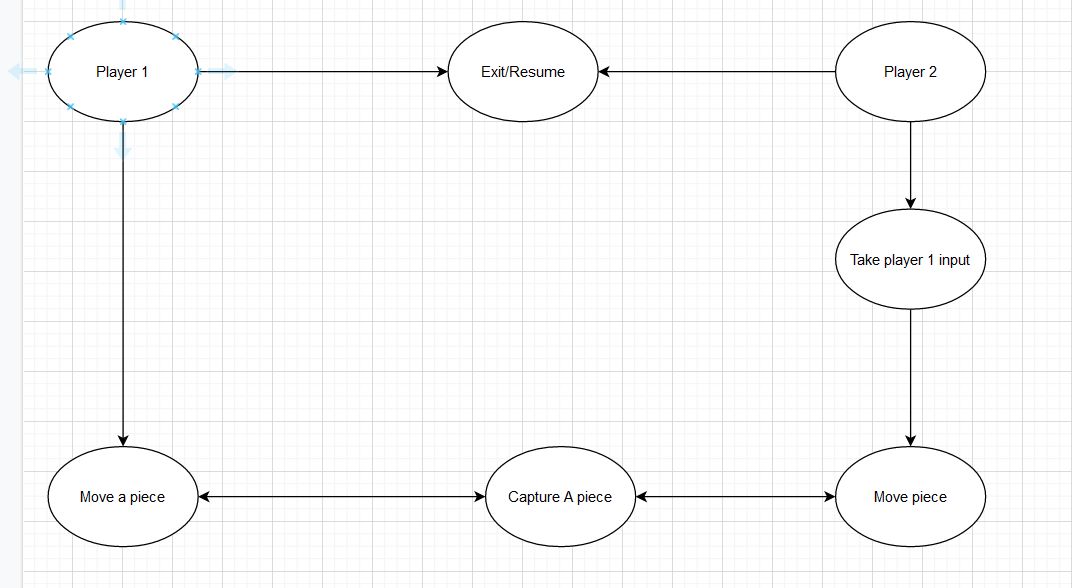
**UML DIAGRAM**



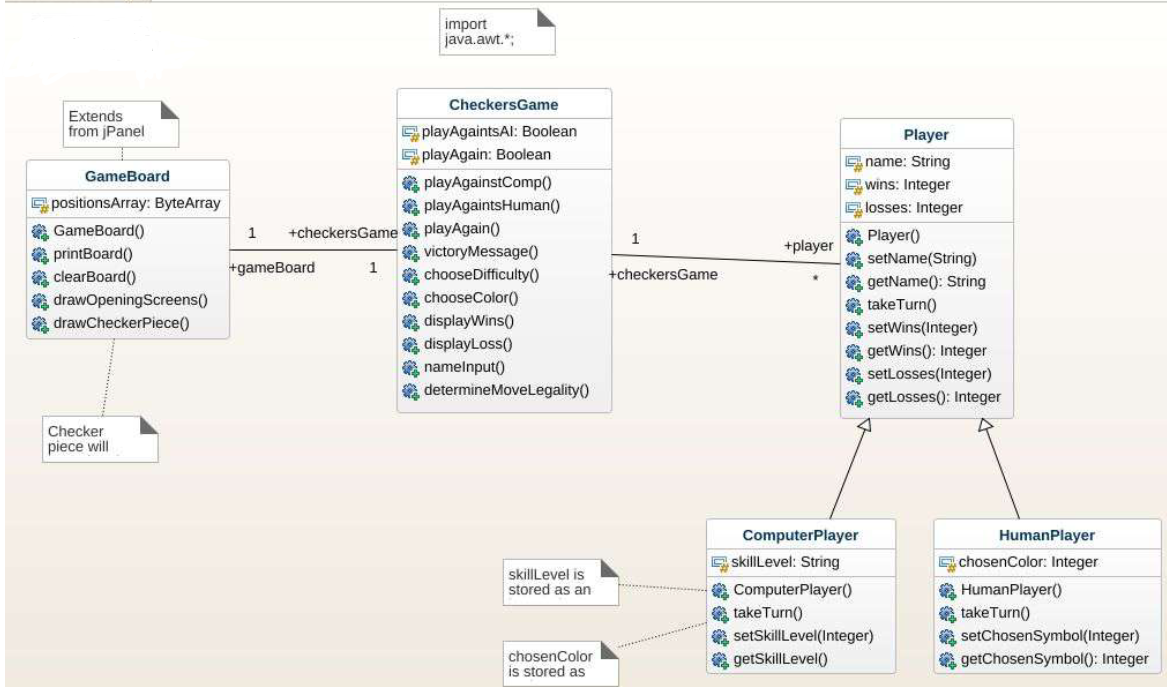
**USE CASE**

Use-case diagrams describe the high-level functions and scope of a system.

The following diagram identifies the interactions between the system and its actors. The use cases and actors in the use-case diagram describes what the system does and how the actors use it, but not how the system operates internally.

****

**CLASS DIAGRAM :-**

****

**Classes:**

* + - 1. **GameBoard**

This class contains functions such as initialising the initial positions of the pieces, displaying the board on the display screen and drawing checkers pieces.

* + - 1. **CheckersGame**

This class contains fundamental functions which displays wins, losses, valid and legal moves, will take input name of the player, colour, victory message which forms the basis of our game.

* + - 1. **Player**

This class will be used to display the name of the player, display total number of wins and losses.

This class contains two subclasses: ComputerPlayer and HumanPlayer