Austin Sizemore

Object Oriented Design

**Checkers Game Data Model Concept**

**Description:**

Using logical movement, we can develop a pseudo application to control logic of a game of checkers if it is being played between two players, two computers or one of each. We will be using the diagram below when deciding how to move each piece and when deciding which moves are legal and which are not.

**Requirements:**

**First Requirement-getPlayers:**

We must obtain who is playing whether that be two players, two computer players or a player and a computer. Prompt any human players to provide a username and provide computer players with pre determined names. These can be taken into two strings called name1 and name2 that can be later displayed on the screen.

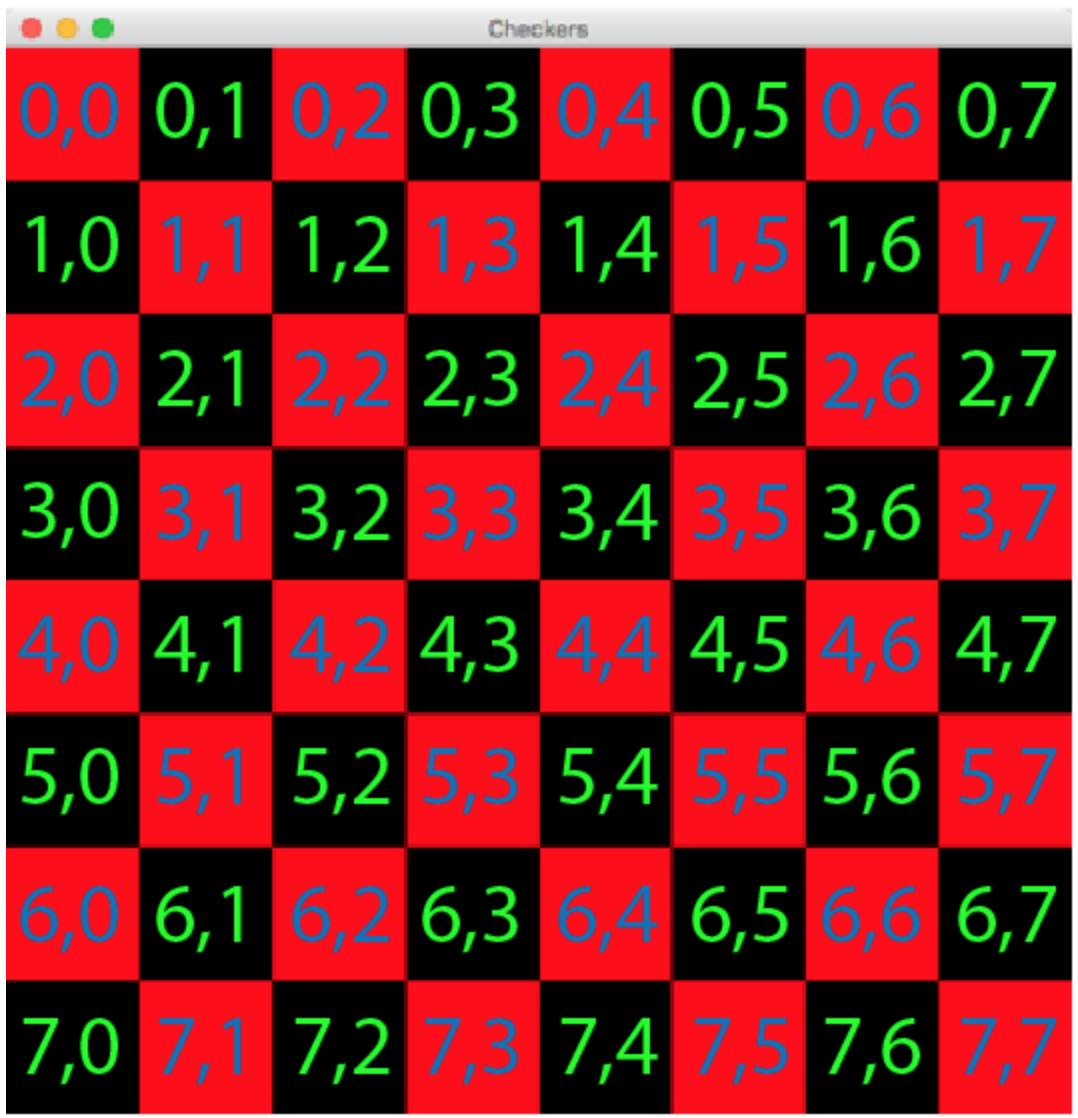
**Second Requirement- assignTurnOrder:**

A randomized function can be given that will determine which player goes first. After this is resolved, players get assigned their respective color as black (or the darker color) traditionally for the one who goes first and red (or the lighter color) for the player who goes second.

**Third Requirement-generateBoardAndPieces:**

A Checkerboard object can be made that has two arrays from 0-7 or that have 8 spots each. These will be the coordinates of the spots on the checkerboard that are determined by the numbers in the 2-Dimensional array.

**Fourth Requirement-Checking Valid Moves:**



Pieces on a board can only move in certain ways. Pieces can only move diagonally and on black spaces. Pieces can only move in a right upwards diagonal or left forward diagonal according to the player’s orientation that they belong to. A piece cannot move into a space that is red or occupied by another piece which can be checked by a status attribute on a square or a squareColor attribute. A piece can hop over an enemy colored piece in that is in the directions as mentioned above and if there is a square of the correct color on the other side. When an enemy piece is captured, the capturer will go again if any legal moves are left to make. Pieces that make it to the other end of the board and land in the back row of the enemy’s beginning area then that piece is considered a king. Kings can move one space in any diagonal direction. Using a 2-Dimensional array, we must perform calculations to check spaces for validity. First checking if a piece is a king or not allows us to know which selection we can make for movement. Then, checking for pieces that have the same color attribute that reside in pieces around the selected piece is needed. We cannot move onto or jump over our own pieces. If a player wants to move onto a square to the right of the piece, then they will have to add one to the second digit in the coordinate and add one to the first. If the piece is a king, then subtract one from the first digit instead of adding one. The same goes for the left side. If any enemy piece exists in a way with an open space on the other side, the path must be calculated to the spot to see if any enemy pieces were in it. Those pieces get removed if they exist then the capturer gets to take another turn.

**Fifth Requirement- gameEndConditions:**

One player will win if they have captured all other pieces from the opposing player. Both players can have the match result in a draw if neither player has any legal moves left. This can happen when one piece of a player is trapped in a corner and all available squares of the same colors are occupied by enemy pieces that are next to the solitary piece. Squares that are also two diagonals away are also occupied by enemy pieces preventing jumps. These conditions can all by checked by checking the diagonals in all directions from a piece a user has selected.