Brogrammers APCS1 pd 02
Kevin Feng, Mohtasim Howlader, Alan Wang 1/02/18

Final Project Proposal

Matrix Game Simulator

We are going to create a tabletop game simulator in which a player (or two) will be able to choose between these three games:

Chess

We are going to use a matrix to represent a chess board and label the sides of the board with letters and numbers. The pieces will be subclasses of the Object gamePiece that correspond with their specific type. The board will be populated with strings that represent each piece. There strings will be colored black or white to indicate possession. This game will be designed for 2 players. The player will then indicate where he wants a piece to move by typing the position he wants to move from to the position he wants to move to, which will only execute if the move is possible. Once the move is executed, the other player is able to take his turn and the game ends when the king is checkmated or when the game ends in a stalemate. We will incorporate the check and checkmate function by having a variable for each object that indicates it's possible moves, and then checking if each square on and around the king are located in this variable using a for loop.

Classes:
-Chessboard
-ChessPiece
-Pawn,Queen, King, Bishop, Rook, Knight
-ChessGame

Features:
- Visual Matrix
-2 Player, turn based

ChessBoard:
-Matrix of the Chessboard denoting the positions of all squares
-Representation of different pieces through letters, using their toStrings
-Constructs the matrix of the chessboard and all of the pieces
-has the methods that updates the matrix of the chessboard.
-determines whether checkmate has occurred
-determines when the turn is over and gives confirmation back to chessGame to to give prompts to player with next turn

ChessPiece:

- Abstract class for chess piece subclasses

Bishop, Rook, Knight, Pawn, Queen:
- Subclasses of ChessPiece
- Determine validity of user input moves
- If valid, then sends confirmation to chessboard to make the change

King:
- Subclass of ChessPiece:
- Determine validity of user input moves
- Has restrictions based on whether it's in check or not

ChessGame:
- Prompts the players for inputs
- Sends those inputs to to chesspieces
- Once checkmate happens, it will end the game and send the user back to the tabletop

Topics:
-Arrays and Matrices: The entire board is a 2d arrray and a matrix. We will have to employ the principles we learned about them to sucessfully place and move the pieces.
-Inheritance: ChessPiece is a superclass of all of the different types of chess pieces. Because gameboard interacts with chesspieces and not the individual types of pieces (other than king because it's special), it is important that they are all subclasses.

Checkers

We are going to use a matrix to represent a checkerboard. Red team will have red pieces, while black team will have black pieces. This will be very similar to the chess game with similar rules, however we are going to restrict the moves differently and allow for jumps if possible. This game will also be designed as a two player game. Just like in chess, the player will select the piece he/she wants to move and to where he wants to move it. The game will conclude when either player has no more pieces or cannot move any of his pieces.

Classes:
-CheckerBoard
-Checker
-CheckerKing
-CheckersGame

Features:
-Visual Matrix of the Checkerboard denoting the positions of all squares
-2 Player, turn based
-Jumping of multiple pieces at once

Checkerboard:

- Constructs the checkerboard in the form of a matrix. Places all of the checkers in their default positions.
- Determine whether or not a player has won by checking if all pieces of one color have been eliminated or all of a players' pieces become trapped and unable to move
- This object is what makes any changes to the checkerboard. All moves by any piece will be completed here. It is also in charge of promoting normal checkers to kings.
- After a player makes a move, this object will check to see if the player is required to make another move as mandated by the rules of the game. If the player is, then this object will prompt the player to make another move.

Checker:

-Color

-has its current Coordinates on Board

-Each piece on the board is this

-Receives inputs for moving chess piece, but verifies if it is a legal move. If verified, then it will send the inputs to the Checkerboard to make the change on the board. If not, it will prompt the player that the move was invalid and to try again.

CheckerKing:

- Subclass of Checker
- Overridden Move method allows for going backwards

CheckerGame:

- Starts the game of checkers
- Displays the prompts to the players to make their moves
- Passes on those inputs to checkers
- Once the gameboard tells it that a player has won, this object will display the victory to the players and end the game. It will also take the user back into the table top.

Topics:

-Arrays and Matrices: The entire board is a 2D arrray and a matrix. We will have to employ the principles we learned about them to successfully place and move the pieces.

-Iteration: going to be needed to set up all of the checkers since all of them in each color are the same.

-

Card Matching (NOT CONFIRMED AS OF YET. DEPENDS ON SPEED OF DEVELOPMENT)

We will create two matrices, one used as a display for the user and one that contains the actual values pertaining to each coordinate. The matrices will be labeled with letters and numbers on the side. The user will be able to pick their difficulty, which will choose the dimensions of the

card game. By picking a pair of coordinates, the game will compare the values behind the scene in the 2nd matrix and if are equal, will be displayed as matched on the display matrix. We will support suits and colors, which could be used for a harder difficulty (matching same color suits only).

Classes:

Variables:

Features:
-Implement mapping of matrix so we know where each card is
-Randomizer so there can be new games
-Ability of changing the difficulty of the game through implementation of larger matrix

Topics:
-Matrixes
-Iteration

We're using OOP by separating out the multiple tasks required for different objects. For example, whether we are making change to the board or vertifying the legality of the moves we are doing so using OOP.

Blue is for Chess
Red is for Checkers
Orange is for card matching

Prioritized To Do List:

- Create super/sub classes
- Change ToString method
- Work with writing functions for superclass to subclasses
- Implement taking of pieces
- Implement Check and Checkmate functions
- Implement the turning of a pawn to a king once it reaches the other side
- Implement super/sub classes
- Change ToString method
- Work with writing functions for superclass to subclass
- Implement taking of pieces
- Implement taking multiple pieces through jumps
- Implement winning through the taking of all pieces
- Implement the kinging of a normal piece once it reaches the other side
- Create super/sub class
- Change ToString
- Work with writing functions for superclass to subclass
- Implement matching algorithm

- Implement randomizer that scrambles the cards
- Implement algorithm for restricting where each piece goes
- Implement castling
- Implement algorithm for restricting where each piece goes

Rough Timeline:
V1:

- Compilable Super/Sub Class
- A working ToString method
- Implement taking a piece
- Implement the changing of a Pawn to any piece once it reaches the other side

V2:

- Implement Super/Sub Class
- A working ToString method
- Implement taking of pieces along with taking of multiple pieces through multiple jumps
- Implement winning
- Implement the kinging of a normal piece once it reaches the other side

V3:

- Create super/sub class
- Change ToString
- Work with writing functions for superclass to subclass
- Implement matching algorithm
- Implement randomizer that scrambles the cards

V4:

- Implement algorithm for restricting where each piece goes
- Implement castling
- Implement algorithm for restricting where each piece goes

-Complete V1 by 01/08/18
-Complete V2 by 01/11/18
-Complete V3 by 01/13/18
-Complete V4 by 01/14/18
-Complete base project by 01/14/18
-Editing and Testing 01/15/18
-Final Revisions and game check by 01/16/18