Team git_it_dun
Angela Tom, Bing Li, Brandon Chong

APCS1 pd 2
2018-01-02

Final Project Proposal
Checkers Board Game

Rules:

- **Movement**: The basic movement is to move a checker one space diagonally forward.
- **Jump**: If one of your opponent's checkers is on a forward diagonal next to one of your checkers, and the next space beyond the opponent's checker is empty, then your checker **must** jump the opponent's checker and land in the space beyond. Your opponent's checker is captured and removed from the board.
    - **Forced capture**: If a jump is available for one of your pieces, you must make that jump (cannot move other pieces). If more jumps are available with that same piece, you must continue to jump with it until it can jump no more.
- **Crowning:** When one of your checkers reaches the opposite side of the board, it is crowned and becomes a King.
    - A King can move backward as well as forward along the diagonals. It can only move a distance of one space.
    - A King can also jump backward and forward. It must jump when possible, and it must take all jumps that are available to it.
    - Except for the fact that it can move backward one diagonal space, it follows the same rule as regular checker
- **Victory**: The game ends when one side capture all the opponent's checkers, or you've surrounded your opponent's checkers so that they have no available moves

User Experience:
Two players are required to play checkers: one will be red with "r" pieces and the other will be black with "b" pieces. The program will print out a prompt to ask for each player to choose their side. Red always starts first. Then our program ask the player to type coordinates of the checker piece they want to move. After the player made their choice, the player will be prompted to enter "fr" (forward right) or "fl" (forward left) to move the checker piece one space diagonally or jump over the opponent's checkers. Forced jump takes higher priority than regular forward movement, meaning that the user must make a capture if there is an opponent checker piece one space diagonally forward on either side of their piece. If the regular move ('fl" and 'fr') is invalid due to a force jump for a different checker piece, then a message will be displayed. If multiple jumps are a possibility, then they will be executed automatically by our program. When a checker piece that is not already a king reaches the end of the opponent's side, it will become a king, which is represented with a capital "B" or "R" on the board. Then the players are given the choices of "fr" and "fl" as well as "br" (backward right) and "bl" (backward left) on the

command prompt. Red and black will continue taking turns until there is a victor, marking the end. That can occur in two ways: either of the sides do not have any valid moves or a side does not have any checkers left. The winner with the least number of moves will appear in the scoreboard.

Possible classes:

- Woo.java (main method, runs everything)
- Board.java (init in Woo.java)
- Checkers.java
    - bKings
    - bCheckers
    - rKings
    - rCheckers
- scoreBoard.java

Woo.java

Instance Variables for Woo.java

- Int numMoves //keep track of the moves that the user made. When the game ends and the winner is decided. numMoves will be passed into scoreBoard.java so that it can be saved and sorted
- String userName //ask the user for the username, so it's easier to identify yourself in the scoreboard and feel proud!

Feature for Woo.java

Class Woo starts the game. It should use initBoard() from game class. It should also print out messages to tell the user what is going on (main class)

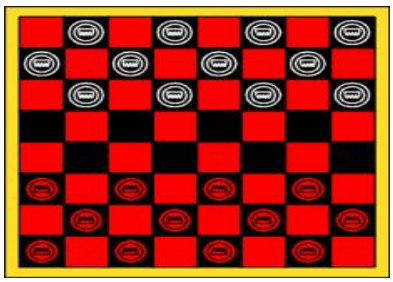
Red starts first
LOOP
Ask player to choose which checker to move
User chooses checker
Check for forced captures (Rules below)
Ask player for spot to move
Checks if move is valid or can be continued or stops
Red ends turn
Black goes
……..
If no move possible (trapped) or no checkers left
    Then end game and declare winner

Board.java

Instance Variables for Board.java

- Final static int SIZE //the board should be 8 x 8 always, user cannot change this
- String[][] board //Create the checkerboard

Feature for board.java:

- populate the board with correct number of checkers on each side (see diagram below)
- print the checker pieces and empty space. Add label to each column and row to insure clarity
- 

Checkers.java

Instance Variables for Checkers.java

- Int Rleft; //Check how many red checkers are left
- Int Bleft; //Check how many black checkers are left
- Boolean redTurn // true = red turn and false = black turn

Features for Checkers:

- Should have accessors for each private instance variables for the subclasses (the different checkers) to use
- Should implement a method that checks for forced capture, because all checker piece (no matter color and types) have to do
- Abstract move method for rCheckers, bCheckers, rKings, bKings
    - abstract methods check if the move the users entered are valid. These should be abstract because different type of checker pieces (color and type) execute their move differently. For example, the red and black checkers are charging at opposite direction. A king can move both backward and forward but a checker can only move forward. If the user entered a command that is valid, the move will be made and displayed at the terminal

scoreBoard.java

- ArrayList scores[][];
    - should be five rows and 2 columns. Item one would be score. Item 2 would be the username
- We plan to create a method that will take an inputted username and the numMove instance variable from Woo.java as parameters, the method will put those parameters into an ArrayList that the method instantiates called ArrayList current[][]. The score of current[][] will be compared to the scores in scores[][] and the entire current[][] will be

inserted in the correct location to keep scores[][] in order. We will then delete the last score from scores[][] to keep the size of scores[][] 5 rows.

rKing.java, bking.java

- isKing method to check if the checkerpiece is a king
- Kings are able to move in 4 directions: forward right, forward left, backward right, backward left

Prioritized-to-do list

1. Board.java
2. Checkers.java
3. rCheckers.java
4. rKing.java
5. bCheckers.java
6. bKing.java
7. Woo.java
8. Scoreboard.java

Rough timeline

- 1/6/18 - Version 1 (minimal but working, can make moves, Woo.java works)
- 1/10/18 - Version 2 (add score board probably)
- 1/16/18 - Version 3 (solidify, add anything necessary)

How are we making meaningful use of OOP?

- The board, each checker piece, and the scoreboard are all objects in our code. Each instance of an object represents an object that could be thought of tangibly. Our objects have both *state* and *behavior*. State includes attributes such as color or scores while behavior, in this case, is moving and jumping.

How are we making use of at least two other first term concepts?

- We are using two-dimensional arrays and arraylists.
- In addition, we are using polymorphism and inheritance.