NOTES FOR THE GUI!!

They should click on the pawn they want to move, not the place they want to move to (if they want to move a pawn from home, they should click on the static image)

For the canMove and move methods:

First you have the player roll the die. Then you check to see if they have a valid move (using canMove(Player p)) if they do have a move, you invoke move(Pawn).

**Pawn class: X**

Each pawn knows its color and position as well as its **start position** and **finish position**.

These two positions are calculated through the color index in the array [blue, green, yellow, red]. Where blue’s start position is in the array at place 1 (since to calculate the finish position I subtract 1). All others increase by 10 to get the starting position. Also has an instance variable canFinish which is a boolean that tells you if the pawn is allowed to go into the finish zone.

*Helper functions:*

isColor(String color): boolean, checks to see if the color is valid (prints out a message if not), helper function

sPosition(String color): int, calculates the start position of the pawn based off its color

*Methods:*

getColor(): String, gets the color of the pawn

getPosition(): int, gets the current position of the pawn, returns -1 if in the homezone and -2 if in the finish zone

setPosition(int position): void, allows you to set the position of the pawn, limited from -2 to 41. If the position is >= 42 it takes the remained when that position/42 and sets the canFinish to true. (which allows you to say that if the pawn position>finish position then set its position to -2)

getStartPosition(): int, gets the starting position for the pawn, varies by the color

getFinishPosition(): int, gets the finish position for the pawn, varies by the color

getCanFinish(): boolean, returns true if the pawn can finish and false if it can’t

setCanFinish(): void, sets canFinish to false (for use when pawn is sent back to home)

toString(): string, returns the string “The ” + color + “ pawn is at position” + position

**Board class: X**

This class creates a Pawn array. It has the instance variable Pawn[] board. It can also check to see if there is a pawn in a certain position. Most of its other functions we thought we would assign to it ended up going to the pawns, or was edited out. (ie the finishzone is now just a space as is the homezone so it doesn’t need to restrict access to those zones.)

*Helper functions:*

checkPosition(int position): boolean, checks to see if there is a pawn in that position of the board

*Methods:*

getBoard() - returns the board which is a pawn array

getPawn(int position): Pawn, uses checkPosition(int) to see if there is a pawn in that position, if there is one, it returns that pawn (useful for the trouble() method)

setPosition(int position, Pawn p): void, sets a position to have a Pawn or a null (for moving pawns around the board)

**Die class: X**

A class that contains the die which when rolled returns a number 1 thru 6, other methods don’t matter. Taken from a previous assignment.

*Methods:*

roll(): int, returns a number 1 thru 6

**Player class: X**

The player class has 4 instance variables: Die, color, name and an array of 4 pawns.

There are two constructors, if no name is given, then the color is assigned to be the name, otherwise they are the same. Both assign the 4 pawns to the color given.

*Methods:*

getColor(): String, returns the color of the player

getPawnArray(): Pawn[], returns the 4 pawns the player has

getName(): String, returns the name of the player

rollDie(): void, rolls the die that returns a number from 1 to 6 and sets it to an instance variable

getDieNumber(): int, gets the number from the instance variable die

inPlay(): int, returns the number of pawns that are on the board which excludes the home and finish zones, (0-41 indicates that the pawn in on the board)

inFinish(): int, returns the number of pawns in the finishzone as determined by the pawns position (-2 indicates that the pawn is in the finish zone)

inHome(): int, returns the number of pawns in the homezone as determined by the pawns position (-1 indicates that the pawn is in the homezone)

noMoves(): boolean, returns false if there is no valid move and true if there is a valid move.

compareTo(Player p2): int, compares the number of pawns in the finishzone between the two players

toString(): String, returns the name of the player plus the number of pawns in the finishzone

**ComputerPlayer class: X**

This class extends the Player class. It includes a method that will randomly choose a pawn that is in play and move that pawn. This method will only be invoked if the die rolled is not a 6 (where the computer player will try to move a pawn from home).

*Helper methods:*

pawnIndex(): int[], returns an array with the indices of the pawns in the pawn array that are in play(NOT their position, but pawn 0 in play or pawn 3 is in play ect)

startOccupied(): boolean, checks to see if there is a pawn in the start position

*Methods:*

pickPawn(): Pawn, returns the pawn that should be moved

* If there is a pawn in the start position, returns the index of that pawn
* If not:
  + Returns an index of a pawn at home if you rolled a 6 & there is a pawn in home
  + If only 1 pawn in play, choses that one
  + Otherwise, choose a pawn that has canFinish==true
  + Else, randomly select a pawn
  + Doesn’t need to worry about the case where there is no valid move as the Player method noMove() is checked beforehand

**PlayGame class:**

The class that runs the game, instantiates a board and the 4 players (who instantiates their pawns)

Helper Methods:

isComputer(): boolean, checks to see if the player is a computer player, given that an empty string indicates a computer player

setupPlayers(String, String, String, String): void, creates the 4 players given 4 strings. If the string is empty creates a computer player with the color as its name

orderPlay(String, String, String, String): LinkedQueue<Player>, takes in 4 strings of the players names. Calls upon setupPlayers() to create the 4 players. Automatically rolls for each of the players and then figures out the order by the highest rolling player.

getBoard(): Board, returns the board object

Methods:

nextPlayer(Player): boolean, checks to see if the player can move, if they can leaves the player alone. If they don’t have any valid moves, enqueues the current player and returns true. Returns true if there is a valid move, false if not (false is given if they didn’t roll a 6 and they have no pawns in play)

move (Player, pawn): void, first checks to see if the player can actually move, if they can’t goes on to the next player. Then it sets the pawn’s current position to null and moves it to its new position based on what was rolled.

* If the pawn is at home and the player rolled a 6, its position is moved to the start position so long as there is no other pawn of that player in the start position
* If the pawn is in play, then it moves forwards the amount as shown on the die
* If the pawn reaches the finishzone, it moves into the finishzone

If in any of these moves, the pawn lands on another player’s pawn, the method trouble is invoked on that pawn and that pawn is sent to home. The method ends by calling the nextPlayer() method. (MAY BE AN ISSUE BECAUSE WE NEED TO MOVE ON TO THE NEXT PLAYER, but nextPlayer only checks to see if they have a valid move, may need to manually enqueue and dequeue the player in this case)

trouble(Pawn): void, takes in a pawn and sends it back to home, ensuring that its previous position is set to null again and that its canFinish variable is set to false

winning(): String, uses a priority queue to figure out the order of who is winning based off how many pawns they have in the finishzone.

isWinner(): boolean, checks to see if any of the players have all 4 pawns in the finishzone, if one does returns true, else returns false.

**Addition notes to be aware of:**

All colors when hardcoded in must be in *lowercase*.

We need to clarify how we identify the computer players. We currently assume that each player will rename themselves, if they stick with the lowercase color, it will create a ComputerPlayer.

For the move method, we need to take in a pawn so that the computer player function works