**Group Chicago - Documentation**

Wes O & Jacob B

CS-171-2

Fall 2021

**Project description:**

For our project, we chose to attempt a simplified, two-player version of the popular board game “Monopoly”. This proved to be a little too ambitious in consideration of all the things we would have to implement to make a completely finished game. For the purposes of documentation we are shifting our focus to a simulation of player pieces moving around the board. Some member variables of the Square class are not utilized anywhere in the program due to this fact (rent, buy, rentOnceBought), but their intended use given a completed monopoly game where every square is a house is apparent. The same goes for Player class functions addMoney and subtractMoney. Code that is unessential for the running of the simulation is included to illustrate initial idea for the monopoly game, but comments are used differentiate non-essential code.

**Assumptions made:**

user will give inputs as prompted, and be familiar with the basic idea of how pieces move around a monopoly board starting in the bottom left corner

User will know to hit ctrl+c to end the program once they have seen how pieces move around the board

**Flowchart:**

1> Ask users to enter their names

2> assign players a designation as “player 1” and “player 2” and a symbol to represent their piece on the board

3> loop begins which will run the simulation

4> use rand() function to simulate a dice roll and has user enter in the number they “rolled”, storing this value with an int value. Repeats for player 2

5> creates an int variable to store player position, set equal to P1.position(int roll) which is a member funciton of the player class, taking in the user’s roll and adding it to the sum of previous rolls to update the player’s position (from 0-16) on the board and reseting it to 0 once it reaches 17 or more

6> for loop iterating from 0-16 begins for purposes of printing out the board after both players have entered their dice roll

7> using variable *i* from the loop to access x and y coordinates of the corresponding square on our board, each square is set to the default icon (defautIcon) of the square class, which is a char ‘0’. Each x cordinate corresponds to an element in the x\_axis array, each y with an element in the y\_axis array so that when *i* in the loop is 0, the first element of both arrays (x being 4, y being 0) is acessed.

8> compares current position of the player to *i*, and with an if statement will change that square on the board to the player’s piece if they are equal.

9> another if statement checks if a variable storing the player’s last position is equal to their current position. If it is not (which it shouldn’t be), then the x,y position indexed using lastPoition is set back to a ‘0’ default icon. This is so that once a player is moved to a new square, the square they previously occupied will be “vacant” again

10> another if statement checks if the player class member function used to access money is less than or equal to 0. If it is, the user is informed that that player is bankrupt and the game is over (this will not happen for obvious reasons).

11> process starting at step 3 is repeated for player 2. Once this is complete, the loop is done with one iteration and continues until user hits ctrl+c to end

**Analysis:**

We wanted to create a board and players that would interact just as they would in the real-life game. We used object-oriented programming to make classes for both players and for each square on the board, and used a loop to run the game. This made the most sense, given that in a game players take turns until something prompts the game to be over, which in this case would be one of the players going bankrupt. Given our end product was just the pieces moving around the board, the process of creating a 2d array out of single dimensional arrays was effective. Overall, the code moving the player pieces around the board did what we wanted it to do.

**Citations:**

All outside help from professor Mao.