Date: 2021-10-24

Group #38

Group Members: Yash Kapoor, Robert Simionescu, Himanshu Singh, Oliver Lu

Author of this Report: Oliver Lu

Milestone 1: Class Description and Choice of Data Structures

Die Class:

This class will handle the dice component of the game, to generate a random number. Therefore, in order to create this component of the game, the group imported java.util.Random, so that it can be used to generate a random integer number between 1 and 6. Based on the UML diagram, this class is in Aggregation with GameLogic class, meaning that they are associated with each other. For example, the die class will exist when a player in GameLogic class decides to role the dice to begin the game or when its their turn to go.

Gameboard Class:

This class represents the Monopoly gameboard. An ArrayList of the Square Object was used to contain all the squares on the board for the players to land on. While a public method called getSquares that returns an ArrayList of the Square Object was used to find a certain distance from the starting point. We chose to use an ArrayList data structure to store all the squares on the board because it allows us to add as many squares as we want to the gameboard. Hence, for future milestones, we can easily add more squares to the gameboard and account for railroads, jail, income tax, and other types of properties such as utility. Also, we can retrieve the position of the player rather easily by iterating through this ArrayList until the player is found. Then, we can find what property the player is on and other relevant information about the player such as the amount of money they have and what properties they own.

Player Class:

Player Class is used to handle the components related to the players in the game such as the money, property the player owns, and positions of the player. An ArrayList is used for properties since the player can accumulate more properties as the game proceeds, which adds to the total properties they may own. Furthermore, the position of the player is stored as an int, representing the index on the gameboard's list of squares of the square they are on. Player class is directly associated with GameLogic class since that class handles the main details of the game.

Property Class:

This is an abstract class that handles the properties that the players can purchase on the square class, such as the cost of the property, which player owns it, and the name of the property. The property class works in association with the player class as the player maintains a list of properties they own. Private final String was used for Property_Name since it can not be changed in the future. Likewise, cost was denoted as private final int since the cost will not change. Specifically, each property has a name and costs a certain amount that will never change throughout the game; therefore, these are classified as "final" attributes. Owner is a Player, but is not final since it can change. It is null when the property does not have an owner and contains the Player that owns the property when it is owned.

Square Class:

The square class is a public interface which is in association with Gameboard Class and Player Class. It is mainly used to provide a common interface for gameboard creation. Basically, it represents a square on a typical Monopoly gameboard. There are many types of squares in the game of Monopoly, which can contain properties, railroads, GO, taxes, or jail.

This interface class is implemented by the property class, which handles squares that can be purchased and owned by players. In this milestone, that is all squares, but future milestones will include other types of squares such as tax squares. We used an interface for the squares on the gameboard to make it extremely easy for us to add additional features without changing a lot in future milestones and because squares do not have much in common aside from having a name, but for a gameboard to exist it must be possible to make a list of squares. We wanted to achieve high cohesion and low coupling. Hence, it will be rather easy for us to add more types of properties, railroads, jail, and GO in future milestones.

Street Class:

The street class is primarily used to determine the street name, colour, cost and the determine the rent the player must pay if they land on that street. For the calculatedRent() method, two for-each loops were used to determine how many streets of the same colour are owned by the owner of the square and how many streets of that colour exist. If the player owns all the streets of the same color, then other players must pay 20% the cost of the street as rent. Otherwise, they must pay 10%. This will be expanded to include houses and hotels in future milestones.

This class extends property since it is a type of property in the game of Monopoly, meaning it will be inheriting attributes and methods of the Property Class. Like all properties, it has a cost, an owner, and must be able to calculate its rent. Colour is unique to streets, as is the method of calculating rent.

GameLogic Class:

The GameLogic class deals with the logic of the game. Specifically, a new ArrayList of Player Objects was created to store all the players that are going to play the game. We asked the user how many players they wanted in the game and added each player to this ArrayList, so we can keep track of them. Hence, this allows us to use a for-loop to let players take turns rolling the dice and buying properties if they land on a specific property on the gameboard.

The while-loop was implemented to allow the players to continue taking turns (i.e., going through the ArrayList repeatedly) until one player is left and all other players have been removed from the ArrayList, meaning they have gone bankrupt.

This class is in aggregation with Die Class. Hence, when "R" is entered, it will generate two separate random numbers between 1 and 6. Then, it will move the player the number of squares the player rolled. Using an if statement, when rollDie == rollDie2, the player will get to roll again as they just rolled doubles. Additionally, this class is in aggregation with Player Class, so that it can indicate the position the user is on.

This class handles all the commands the user can input. These are buyProperty(Player buyer), inspectPlayer(), passTurn(), and quitGame(Player buyer). buyProperty(Player buyer) takes a Player and attempts to purchase the property they are on. The purchase fails if the player is not on a property, the property is already owned, or the player does not have enough money to make the purchase. inspectPlayer() allows the user to see the state of any player in the game. It takes a user input for the name and, if the user has input a real player, displays their current square, a list of properties they own, and how much money they have. passTurn() ends the current turn. quitGame() ends the game.

GameUI Class:

GameUI class is the user interface of the monopoly game. This class handles all user input and output to the user. Also, for-loops were implemented to ensure that players do not have duplicate names. A createBoard() method that returns an ArrayList of a Square Object was implemented to handle all the different square names for the gameboard such as the street names. Street names are added to an ArrayList of Property Objects, which will be utilized in later milestones when we have more than one type of property. As mentioned earlier, there will be more squares added in future milestones.

Additionally, a displaysCommands() method was added to indicate common commands that a player would need such as quitting the game, purchasing properties as well as displaying the state of the player such as the amount of money they have and the place they are on the gameboard. This method takes user inputs until a valid command is typed, then sends the command to GameLogic to handle the actual game. GameUI validates all inputs before sending them to GameLogic, allowing GameLogic to remain solely focused on the core functionality of the game.