Bilkent University

Department of Computer Engineering

CS319 Term Project

3A - Monopoly

Design Report

Ali Taha Dinçer, İrem Ecem Yelkanat, Kutsal Bekçi, Sultan Simge Kürkçü, Saidcan Alemdaroğlu

Instructor: Eray Tüzün
Teaching Assistant(s): Barış Ardıç, Elgun Jabrayilzade, Emre Sülün

Progress/Iteration 1 - Project Design Report
Nov 29, 2020

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Object Oriented Software Engineering course CS319.

Contents

1 Introduction

1.1 Purpose of the System

Monopoly is a square shaped board game where each player has a unique pawn and in each turn according to dice results they move their pawns. The players try to have properties and with time, they are able to build structures on their owned properties. The main purpose of the Monopoly game is to avoid going bankrupt which means losing all the money and the properties. Having structures on the properties will help the players have an income so that they won't go bankrupt easily. The game can be played with at least 2 and at most 8 players.

There exists two same shaped card allocations on the board, which are located for chance cards and community chest cards. These cards are drawn by the players and increases the joy of the game. There are different buyable property squares located along the board. The game starts when the dice are rolled and finishes when every player except one gets bankrupt.

In this project, the purpose is to implement Monopoly into the digital single player and multiplayer platform. The name of the digital version of the game implemented is "Yolopoly". The players can play against the computer to enhance their skills and clash with their friends in multiplayer game mode.

1.2 Design Goals

1.2.1 Usability

A usable program requires performing any task safely, effectively and efficiently. Since Yolopoly has various tasks in the playing process, it is quite sufficient to carry out events in the smallest time and space for the sake of usability. In addition, as the base game Monopoly is a rather complex game, it is aimed to have an expressive and easy to use UI for the players who want to play the game. Furthermore, players can learn the game by clicking the "How To Play" button in the main menu and the whole game can be playable with only a mouse or a trackpad.

1.2.2 Reliability

Functioning without failure is one of the most important points of Yolopoly game since there are accountings and specific calculations in the game as a great importance. Each calculation in states and backend processes need to be saved in case of any system failure. For online game options to be added to the game, the game Yolopoly will not require any private user data except a nickname.

1.2.3 Performance

The response time matters in terms of the performance of the game. In order for players to play Yolopoly without any tardiness, performance is taken into account. The data flow in the game needs to be efficient to prevent latency in the game to consider the game high performance. As only the data created by the Inner Engine is saved in the storage system, it is planned to have only 250 mb of space stored on the disk including the game itself. Furthermore, JVM options for running the game will be optimized and the game Yolopoly is planned to use only 512 mb of RAM storage. For planned online gameplay, response time of each move done by the players and sending a request to the Firebase server will not exceed 1 seconds. Each response and request will contain only the values used in Inner Engine. Therefore, the game will be playable with only a 1 Mbps bandwidth or higher.

1.2.4 Supportability

An extendable system requires the ability for modification or addition to the program without interrupting the dataflow of the program. Since Yolopoly has some sort of ground structure which shouldn't be changed, alterations and/or additions demanded can be applied easily. In addition, as stated in the section above, the game will require only 250 mb of storage from the disk and 512 mb of RAM from the computer. For online gameplay option, Yolopoly can be played with only 1 Mbps bandwidth. Furthermore, the game is implemented on Java 15 and can be played with any JRE that supports Java 7 and higher versions. Hence, the game will have extensive supportability.

1.2.5 Cost

In terms of development time, we want our system to have low cost. Since the time we have for the design and the implementation of the project is limited, we don't want our project to have complicated design that is hard to handle. In addition, since we have no budget for this project, we will not use any tools, database or software that will cost us.

1.2.6 Modifiability

We want our design to be easily modifiable. If a functionality will be changed or a new functionality will be added, it must be acclimated with ease. By making our game modifiable, making new changes will not require the current design to be changed extensively and the process will be less demanding.

1.2.7 Maintainability

We want our design to be easily maintainable. Our design should be repaired, improved and understood easily. If a new change will be performed to our design, it should not cause an error in other parts of the design. In addition, we want to find out the cause of a problem in a short amount of time, and the process will be less challenging.

1.2.8 Functionality

Monopoly is a game that has several rules and features, where the player can perform various actions. We tried to make our design to have all of the features, tried to add extra features to keep up the enthusiasm of the players regarding the game, and tried not to discard the features of the original game.

2 High Level Software Architecture

2.1 Subsystem Decomposition

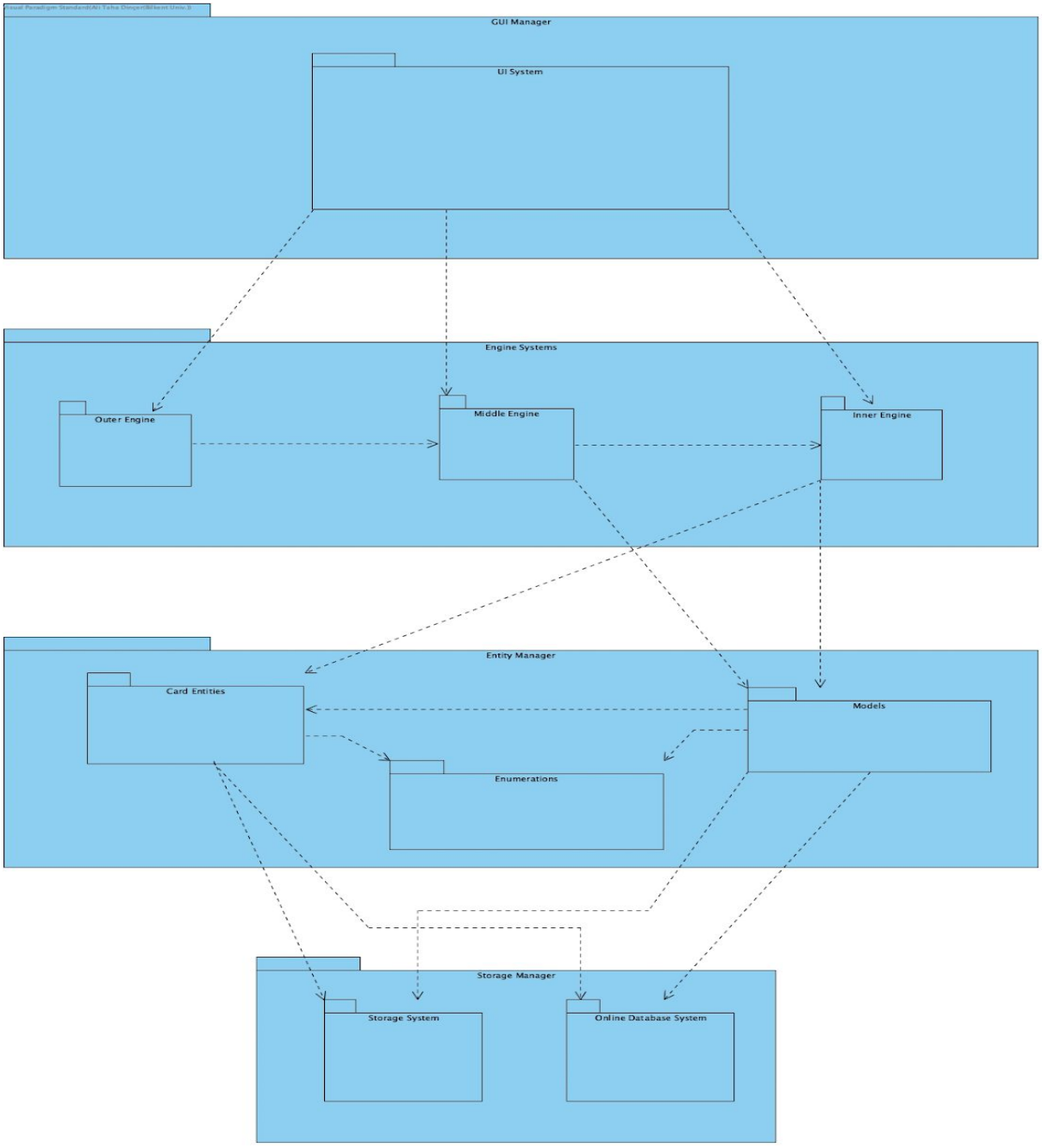

Figure 1: Subsystem Diagram

The system uses a four layer subsystem architecture and hence, is composed of four subsystems. The GUI Manager system deals with the player's navigation from the opening of the game to the playing process. According to coordination of the player the user interface connects to the Engine System and the operations are handled in

the background in different types of engines in the Engine System. The Entity Manager is composed of entity objects in the system and controlled by the Engine Systems in the upper level. The Storage System deals with the storage of the information and online realtime database integrity in the program. Entities load and store data with the help of the Storage System.

2.2 Hardware/Software Mapping

Before the implementation of the game, the latest changes on the final version of the Java programming language has been researched and came to a conclusion that the version of JDK has no limitation for running the game on any computer as long as the JRE is installed on the PC. Hence, the game will be implemented by using Java programming language version 15. For the storage system, JSON will be used and for real time database integrity, Firebase will be used. Whole game will be implemented to be played by using a mouse or a trackpad connected to a computer.

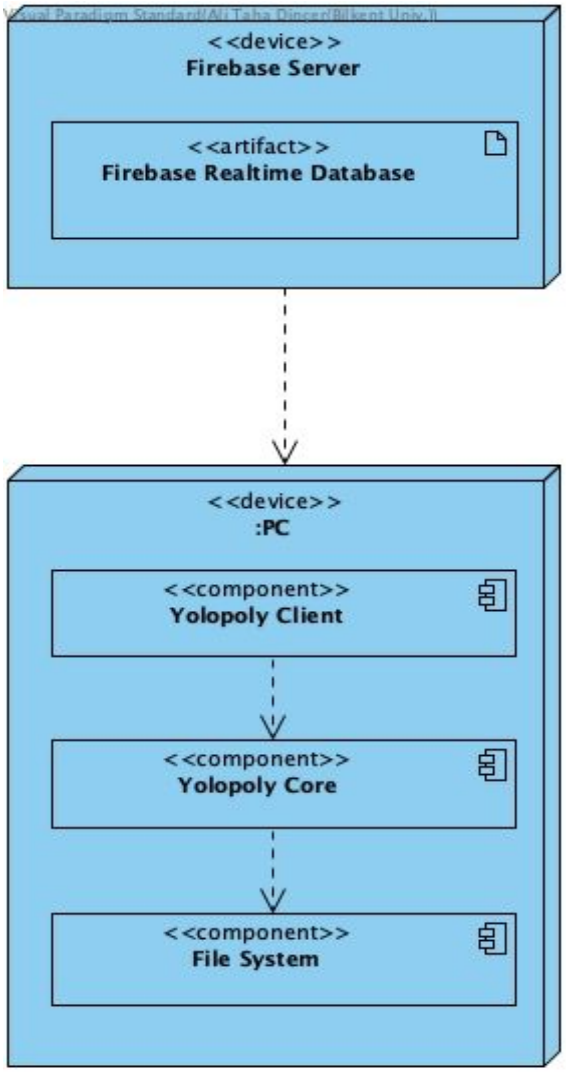

Figure 2: Hardware/Software Mapping

2.3 Persistent Data Management

The game Yolopoly is the digital implementation of the board game Monopoly. Hence, the game is meant to be multiplayer itself. Therefore, we planned to add an online multiplayer future to the game. However, in the first milestone of the project, it is decided to finish the single-player option of the game. Hence, if succeeded in single-player gameplay, it is planned that Google Firebase will be used to implement persistent data management in the game. Firebase serves a secure real-time database option, and Yolopoly is a turn based game. Hence, there will be no conflict or error during reading from and writing to the online database.

Considering the single player game option, the game will be played against the bots and the only human user is the player himself/herself. The implementation of the game relies on a state management system. Hence, every turn is handled with states and in each turn, data of the current game will be saved in a local storage system. This system will use the JSON file system in order to store game data. In addition, there will be no conflict or error during writing to or reading from the inner storage system with the help of the exception handling system.

In both gameplay options, players are limited with the themes that served with the game. Hence, each image and audio resource used on the game is available on the PC of the players. In the online gameplay system, only the data in the Inner Engine will be transferred to the database system. Hence, for the online game option, each player will share the same Inner Engine system.

In addition to the information given above, the game will use different JSON files to load and store the saved games and settings data. The data of the saved games consists of all the information used in the Inner Engine system.

2.4 Access Control and Security

The first milestone during the implementation of the game Yolopoly, is to finish the single-player game option. As the game will be single-player during the next milestone, the game will be safe in means of any internet based security vulnerabilities.

In the online multiplayer game option, the Google Firebase real-time database will be used. For the online real-time database system, Google Firebase is selected because this system is trusted by users and maintained by Google, which is a huge and trusted company. Google Firebase system has its own exception handling system and errors are reported to the caller. Hence, each error returned from the system will be handled in the implementation. As Firebase is maintained by Google

and Firebase itself serves rules for database systems, we are planning to use these rules to avoid any internet based security vulnerabilities to the online database system.

2.5 Boundary Conditions

2.5.1 Initialization

The game will be shipped in a ".jar" file. Hence, any computer who has JRE (which supports Java 7+) installed can start the game by double clicking the game button. The initialization of the game can be divided into three sections.

The first section is the Main Menu section which is managed by Outer Engine system. This engine initalizes settings of the game (JSON file including saved audio settings), instructions of the game (Images of the gameplay instructions of the Yolopoly) and "about us" information (JSON file including informatiın about developers).

The second section is the Game Settings-Lobby section which is managed by the Middle Engine system when the player clicks either Single Player button or Multiplayer button in the Main Menu section. This engine initializes the Single Player settings which are bot count, pawns, mod and theme of the game and Multiplayer settings which are maximum player count, pawn for individual user, game mod and game theme. In both gameplay types, Middle Engine transfers the mod of the game and data of the players including an order to the Inner Engine.

Third and last section şs the In Game section, which is managed by the Inner Engine system when the player launches the game in the Game Settings-Lobby sections. This engine has a huge workload about initializing the game. The game board including each square, properties such as drawable cards and buyable cards, dice and bank is initialized in this section. For online multiplayer game options, as each player starts equally, only the Inner Engine data of the host of the game is transferred to the online database system and Inner Engines of the other players in the game is initialized from the data sent to the online database system.

By dividing initialization steps to different engines, we promise to achieve a better performance on the game by lowering the extensive use of system resources.

2.5.2 Save and Load

The game automatically saves the game at the end of each turn in the In-Game screen and saving and loading the game is only allowed for single-player games.

Users can save and exit from the game by clicking the corresponding button during the game and continue playing the game by clicking the according tile which includes game date, game mod and game theme. The game saves and loads only the data created and used by the Inner Engine system. The saved games are stored in a directory named "saves" and each save file is in JSON file format.

2.5.3 Termination

Direct termination of the game is possible by clicking the "close" button according to different operating systems but this is not a safe operation. In the Main Menu screen, players can safely exit from the game by clicking the Quit Game button. ın addition, players can terminate the operations of each engine. The operations of the Inner Engine can be terminated by exiting from the In-Game screen. The operations of Middle Engine can be terminated by clicking the Return to Main Menu button. As explained, Outer Engine is terminated by clicking the Quit Game button in the Main Menu screen.

Terminating the engines by using in game buttons is the safest way as we aim to save every action to provide better reliability. However, we also aim to provide reliability when the "close" button provided by the operating system is pressed.

The importance of this "close" button is, in the online gameplay option, players can choose to close the game by pressing that button and hence the Inner Engine will not know that this player has exited or making decisions. Hence, only in the online game option, the Inner Engine will wait for 1 minute until the player does any action. If not, the corresponding player will be banned from the game and replaced with a bot player. By this, we aim to provide better security and reliability.

2.5.4 Exceptions and System Failures

During playing the game, it is possible to encounter errors and it is important to solve these issues before players have encountered them. In our implementation, we aim to provide a highly reliable system by using the exception handling system as best as possible. However, there are still some failures that can occur related to internet connection and operating systems. In case of poor or no network connection, we disconnect the according player from the game by checking if the user is doing anything in 1 minute period. In case of any inconsistency or corruption in the name of data, the corresponding engine will terminate itself and return an error which will be displayed to the player. The game is shipped in a ".jar" file, therefore there will be no exception or failure related to the resources such as images, audios or FXML files which are related to the UI of the game.

3 Subsystem Services

3.1 Entity-Model Subsystem

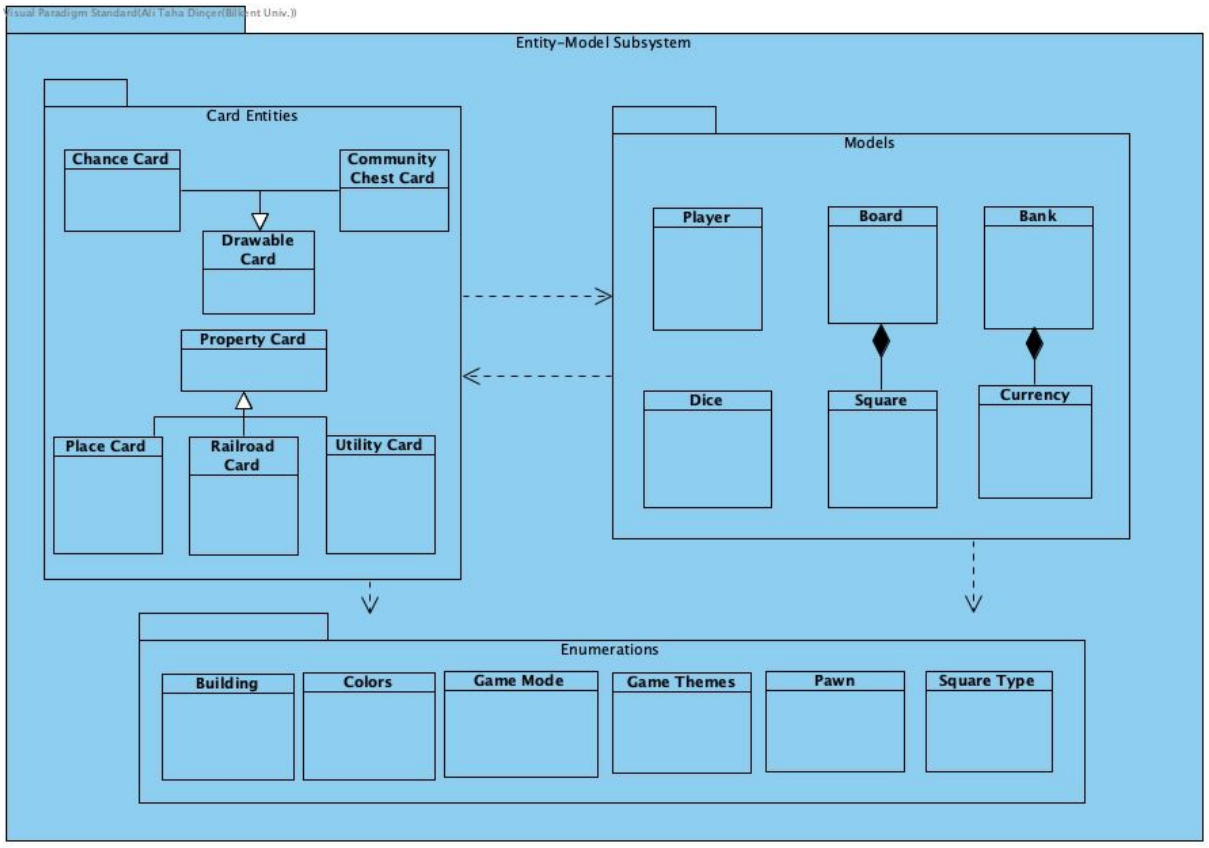

Figure 3: Entity-Model Subsystem Diagram

The Entity-Model Subsystem is divided into three main parts.

First part of the Entity-Model diagram contains the classes of the Drawable Card and Property Card. These classes are abstract and Drawable Card is associated with Chance Card and Community Chest Card classes and Property Card is associated with Place Card, Railroad Card and Utility Card classes.

The second part of the Entity-Model diagram contains base models of the game. These model classes are Player, Board, Square, Dice, Currency and Bank. In this package of the diagram, as the Board class consists of Squares, there is an association between these two classes and for only the Bankman Mode of the game, Bank class has currencies, hence, associated with the Currency class.

The third and last part of the Entity Model diagram is Enumerations. This package holds the enumeration classes that makes differentiation easier during the

implementation process of the Yolopoly. The enumerations are Building, Colors, Game Modes, Game Themes, Pawn and Square Type.

3.2 Controller-Engine Subsystem

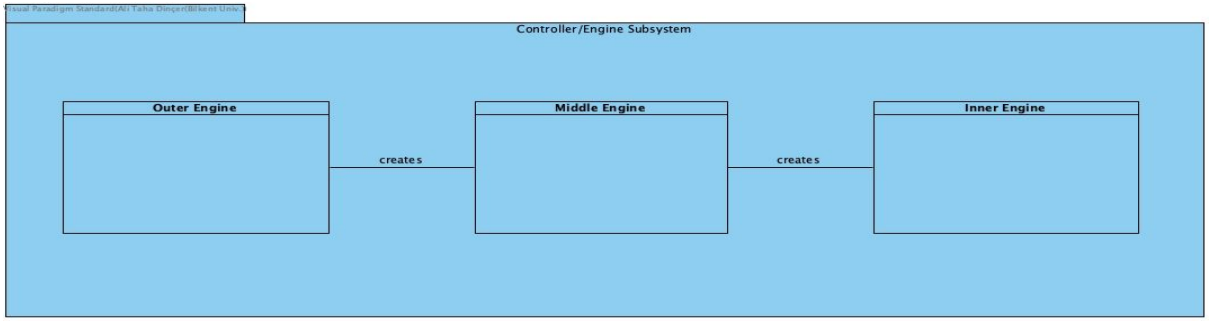

Figure 4: Controller-Engine Subsystem Diagram

The Controller-Engine Subsystem consists of three classes, which are Outer Engine, Middle Engine and Inner Engine. Each of these engines are connected to the corresponding UI system and controls the information flow between models and user interface. Each engine creates the engine in the lower layer in respect to time the screen shown during runtime. Hence, each engine is able to communicate with the higher order engine. For example, before starting the game, an array of players are initialized in the Middle Engine and the same array is resorted and used by the Inner Engine during the game. Outer Engine controls the Main Menu and related screens in the Game. Middle Engine controls the game setup related screens in the game. Finally, Inner Engine controls every data, events, functions and exceptions related to the In-Game Screen.

3.3 View Subsystem

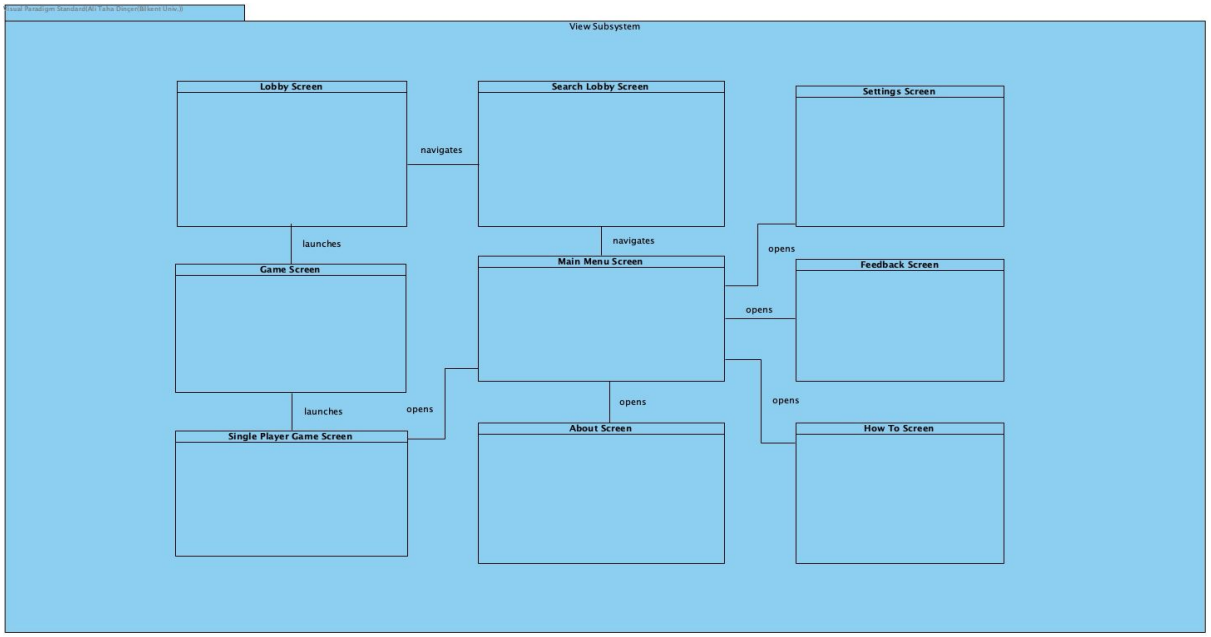

Figure 5: View Subsystem Diagram

The View Subsystem is divided into three systems conceptually. The division is determined by the action. Hence, there are screens that can be opened, screens that can be navigated to and screens that can be launched. In the middle of each class, everything starts with the Main Menu screen. Players can open Settings Screen, Feedback Screen, How To Screen and About Screen in the Main Screen. If the player decides to set up a game, he/she will be navigated to either Search Lobby Screen or Single Player Game Screen. If the player chooses to play a multiplayer game, after the Search Lobby Screen, the player will be navigated to the setup page of the multiplayer game screen, which is Lobby Screen. Finally, after the game setup is complete, Game Screen will be launched.

3.4 Storage Subsystem

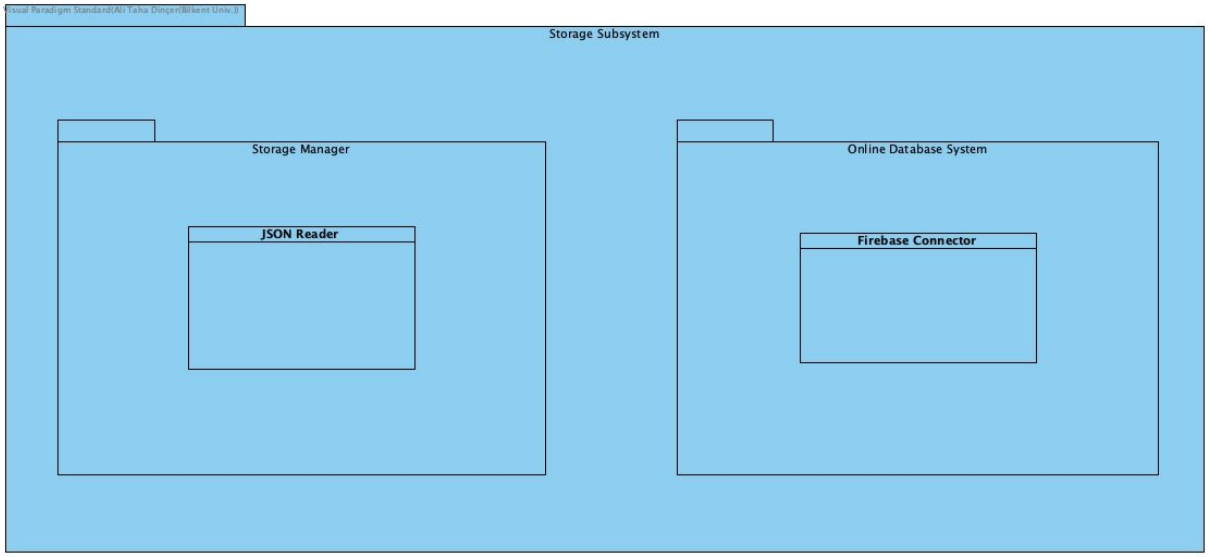

Figure 6: Storage Subsystem Diagram

The final subsystem is the Storage Subsystem. This subsystem is divided into two parts.

The first part is the Storage Manager, which handles the storage related operations such as saving and loading games and changing and saving settings of the game with JSON files. The operations made in that part of the subsystem is local-PC related operations.

The second and final part of this subsystem is Online Database System, which contains Firebase Connector class. As it can be understood, this class handles the communication between game and the online server database system. The operations can be sending requests such as Inner Engine data to the corresponding game lobby area created for the game in the Firebase database and receiving responses such as changes made in the data of the Inner Engine of the game which is on the Firebase database and retrieving them.

4 Low-level Design

4.1 Object Design Trade-offs

4.1.1 Functionality vs. Cost

Since the time for the design and the implementation of the game is limited, we want the functions of the game as simple as possible. We will design the core logic of the game, however, the extra features that we desired to have in the game can be postponed or discarded. Hence, the time required for the software design and implementation can be reduced.

4.1.2 Usability vs. Security

We will store only the game data in the local file system. The system of the game does not hold any private data of the user other than the nickname of the user, and the system does not have signin & login functionality. Hence, we choose usability over security.

4.1.3 Efficiency vs. Portability

We prefer portability over efficiency as Java can run on any platform or environment provided that JVM is installed on the corresponding platform or environment.

4.1.4 Functionality vs. Usability

Even though we are trying to design and implement the software as simple as possible, the Monopoly game has many rules and features, and we didn't want to discard these rules and features of the original game. The game setup screen and game play screen are more complex compared to other screens. However, this will be diminished by the help section with the explanations in "how-to" and guidebook in the help section.

4.2 Class Diagram

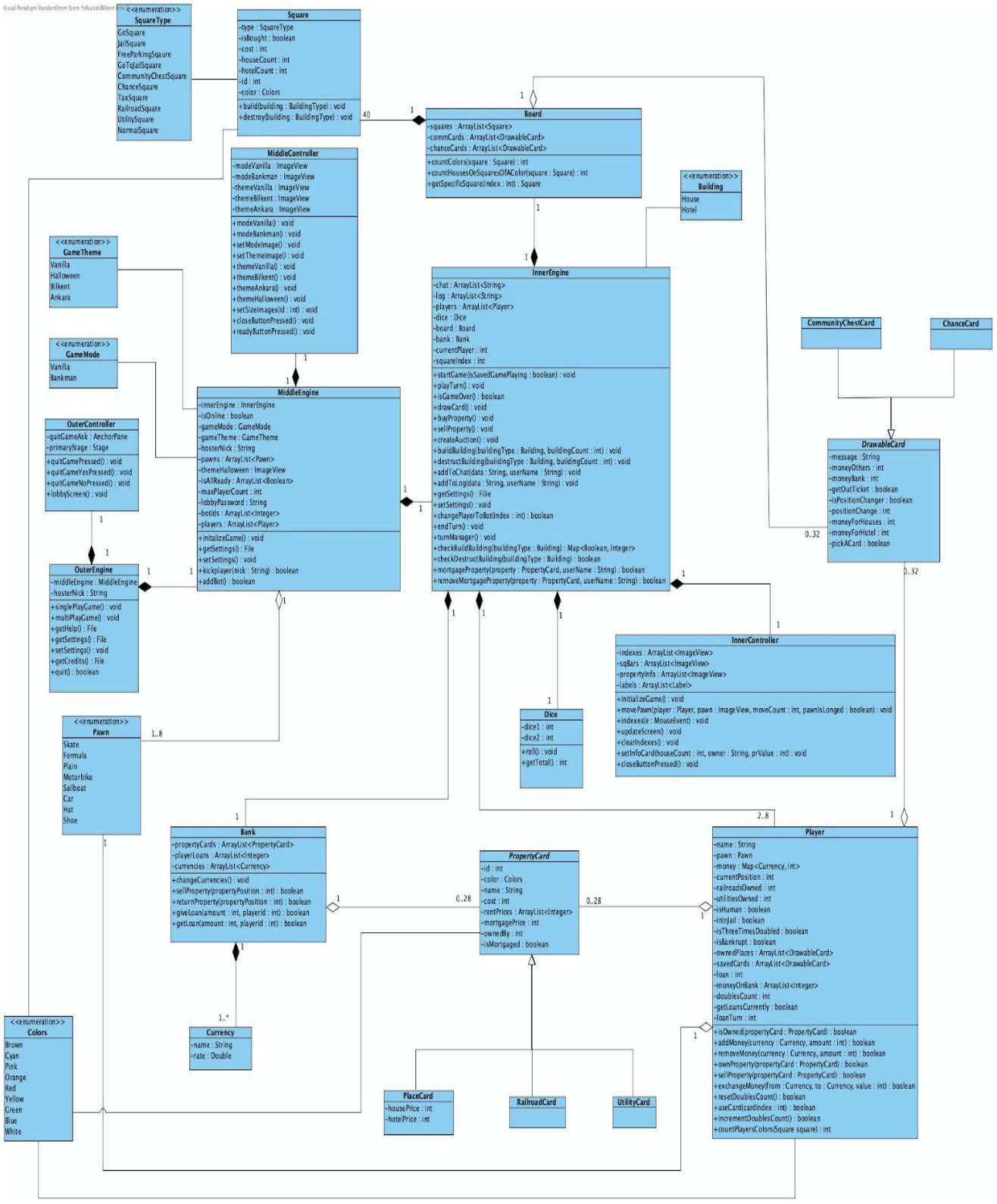

Figure 7: Object-Class Diagram

4.3 Packages

We use JavaFX to build the Graphical User Interface (GUI) of our game. The list of some packages that we use and their brief explanations are listed below.

- javafx.scene: Provides the core set of base classes for the JavaFX Scene Graph API.
- javafx.animation: Provides the set of classes for ease of use transition based animations.
- javafx.fxml: Contains classes for loading an object hierarchy from markup.
- javafx.application: Provides the application life-cycle classes.
- javafx.stage: Provides the top-level container classes for JavaFX content.
- javafx.event: Provides basic framework for FX events, their delivery and handling.
- java.util: Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

4.4 Class Interfaces

4.4.1 Player Class

Figure 8: Player Class Diagram

This class provides the ability to keep track of the player and everything connected to the Player Class. Player Class includes functions for the player actions. The functionality of the class simply directs the game flow.

4.4.1.1 Attributes of Player Class

- private String name: It is used to store the name of the playerç
- private Pawn pawn: It is used to keep the unique pawn of the player.
- private Map<String, int> money: It is used to keep the money currencies available on the game.
- private int currentPosition: It is used to keep the current position of the player.
- private int railroadsOwned: It is used to keep the number of railroads the player has.
- private int utilitiesOwned: It is used to keep the number of utilities the player owns.
- private boolean isHuman: It is used to differentiate whether the player is a real player or a bot.
- private boolean isInJail: It is used to know whether the player is in the jail section of the board or not.
- private boolean isThreeTimesDoubled: It is used to store whether the player doubled their dice three times or not in a turn.
- private boolean isBankrupt: It is used to store whether the player has gone bankrupt or not.
- private ArrayList<PropertyCard> ownedPlaces: It is used to keep the list of property cards that the player has owned so far.
- private ArrayList<DrawableCard> savedCards: It is used to keep the list of drawable cards that the player has drawn. Used to save "Get Out of Jail" cards.
- private int loan: It is used to keep the amount of money that the player loaned so far.
- private ArrayList<Currency, Integer> moneyOnBank: It is used to keep a list of the amount of money on the player in order to currency type.

- private int doublesCount: It is used to store the number of double dices the player rolled in the round.
- private boolean getLoansCurrently: It is used to determine whether the player is getting a loan or not.
- private int loanTurn: It is used to keep track of the number of turns after the player gets a loan since there is a limitation for time to pay back.

4.4.1.2 Methods of Player Class

- public boolean isOwned(int squareIndex): It checks whether the square the player landed is owned by the player or not.
- public boolean addMoney(amount: int): It adds money on the player's bank account according to the actions on the round.
- public boolean removeMoney(amount: int): It removes money from the player's bank account according to the actions on the round.
- public boolean buyProperty(squareIndex: int): It makes the player to own a property in exchange of money on the player's account.
- public boolean sellProperty(squareIndex: int): It makes the player get rid of a property and get a specific amount of money on their account.

4.4.2 DrawableCard Class

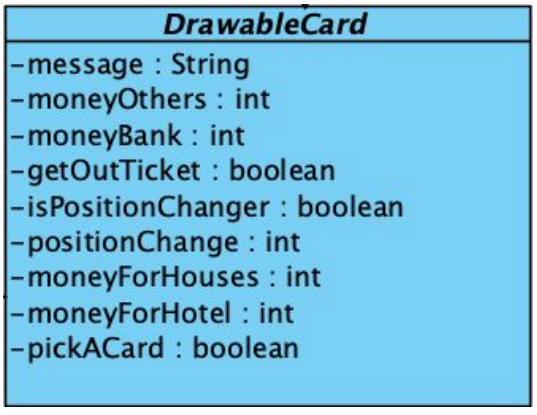

Figure 9: DrawableCard Class Diagram

Everything related to the cards is connected to DrawableCard class.

4.4.2.1 Attributes of DrawableCard Class

- private String message: It is used to store the message or instruction on the drawable card.
- private int moneyOthers: It is used to store the money that is going to be paid to other players.
- private int moneyBank: It is used to store the money that is going to be paid to the bank.
- private boolean getOutTicket: It is used to store the information whether the card is a "Get Out of Jail" card or not.
- private boolean isPositionChanger: It is used to store the information that if the card changes the pawn of the player or not.
- private int positionChange:It is used to store the information that holds the position change value of the card.
- private int moneyForHouses: It is used to store the information that whether the player is paying money to the bank related to the houses owned.
- private int moneyForHotel:It is used to store the information that whether the player is paying money to the bank related to the hotels owned.
- private boolean pickACard: It is used to store whether the drawable card is picked or not.

4.4.3 InnerEngine Class

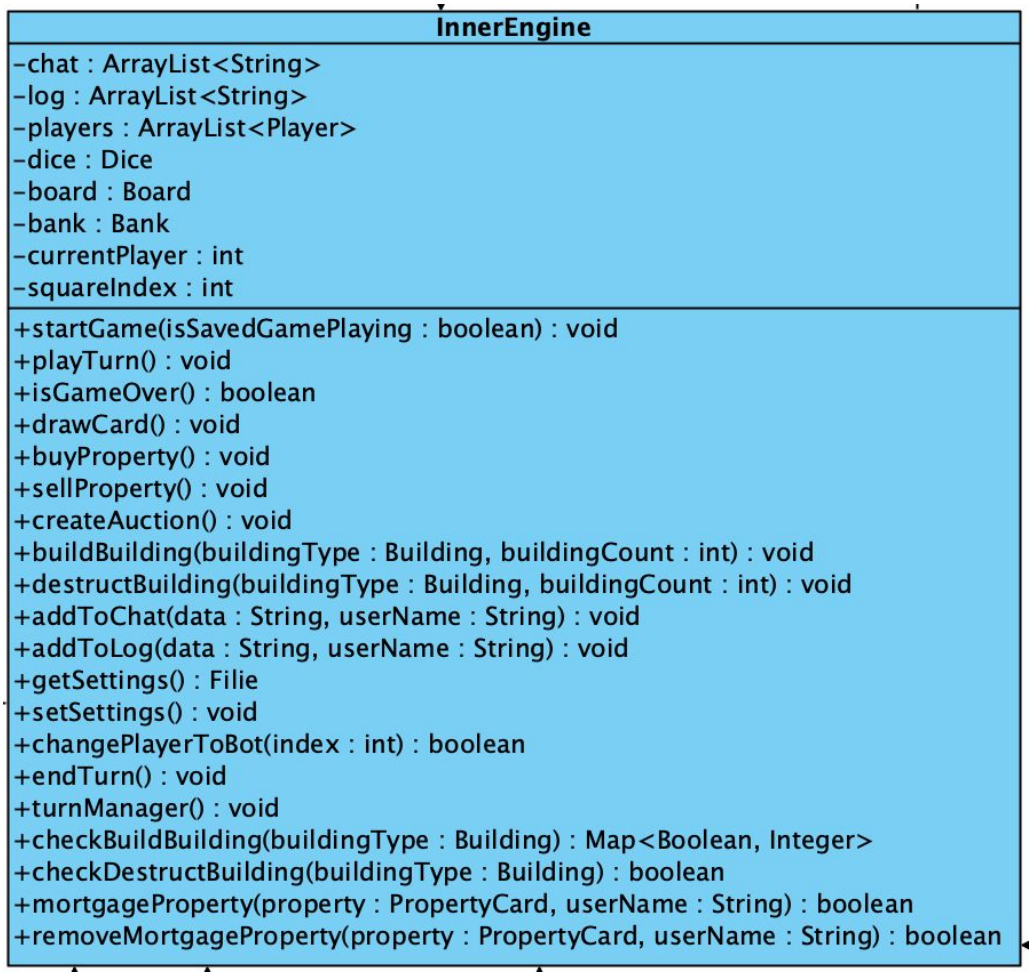

Figure 10: InnerEngine Class Diagram

InnerEngine class includes features and properties related to the screen which shows up while the players are playing the game. This class can be regarded as the game itself since players and their attitudes occur at this stage of the whole game.

4.4.3.1 Attributes of InnerEngine Class

- private ArrayList<String> chat: It is used to store the list of chats in the game.
- private ArrayList<String> log: It is used to store the list of actions done by the users in order.
- private ArrayList<Player>players: It is used to store the list of players that contribute to playing the game.
- private Dice dice: It is used to keep the dice rolled in the round.
- private Board board: It is used to keep the board of the game.

- private Bank bank: It is used to keep the bank of the current game.
- private int currentPlayer: It is used to keep the current player among the players list.
- private int squareIndex: It is used to keep the index of the current square.

4.4.3.2 Methods of InnerEngine Class

- public void startGame(isSavedGamePlaying : boolean): It initializes the game or loads the saved game.
- public void playTurn(): It handles the functions related to the turns.
- public boolean isGameOver(): It checks whether the game is over or it is still on progress.
- public void drawCard(): It makes the player draw a card when it is the time.
- public void buyProperty(): It makes the player buy the property they want to purchase.
- public void sellProperty(): It makes the player sell their property they want to get rid of.
- public void createAuction(): It creates an auction when the player demands to.
- public void buildBuilding(buildingType : Building, buildingCount : int): It makes the player build a structure on the current square when the player wants to.
- public void destructBuilding(buildingType : Building, buildingCount : int):It makes the player destruct a structure on the current square when the player wants to.
- public void addToChat(data : String, userName : String): It takes the player's chat and adds to the chat section with the player's information.
- public void addToLog(data : String, userName : String): It takes the players' actions and adds them to the log section in order.
- public File getSetting(): It gets the settings of the InnerEngine.
- public void setSettings(): It sets the settings of the InnerEngine.
- public boolean changePlayerToBot(index: int): It changes a real player in the game to a bot player.

- public void endTurn(): It ends the turn after players play their turn.
- public void turnManager(): It handles the operations among turns.
- public Map<Boolean, Integer> checkBuildBuilding(buildingType: Building): It checks building a structure process.
- public boolean checkDestructBuilding(buildingType: Building): It checks destructing a structure process.
- public boolean mortgageProperty(property: PropertyCard, userName: String): It checks whether a property is mortgaged or not.
- public boolean removeMortgageProperty(property: PropertyCard, userName: String): It de-mortgages a property from the given player if the given property card is mortgaged.

4.4.4 MiddleEngine Class

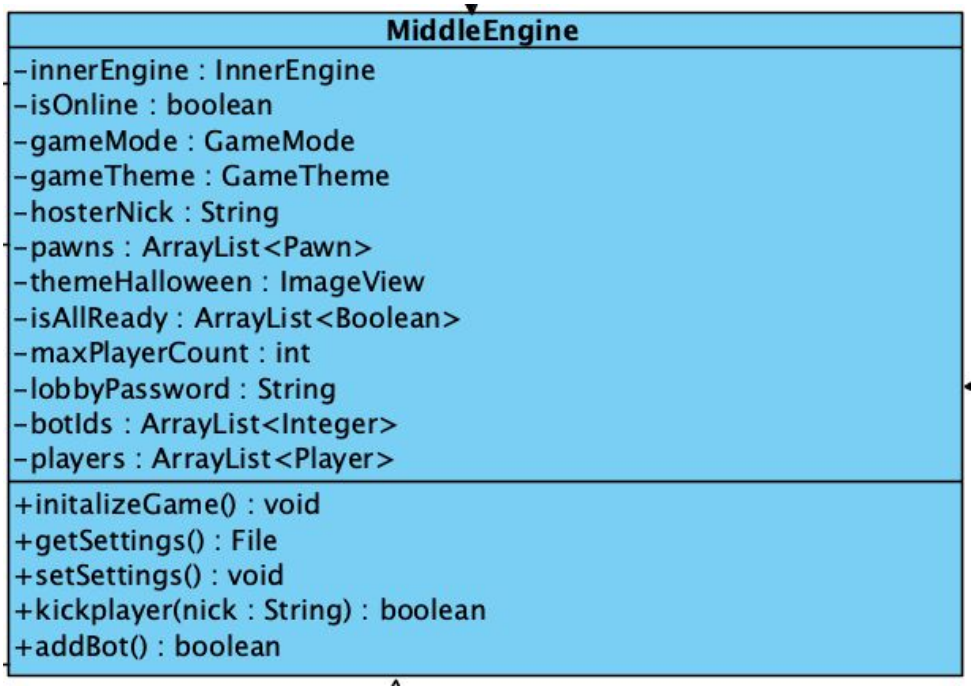

Figure 11: MiddleEngine Class Diagram

MiddleEngine class includes attributes and functionalities of the lobby section of the game. Such options like theme and mode are selected here before the game actually starts. Ground structure of the game is shaped at this class and MiddleEngine class and InnerEngine class are related to each other.

4.4.4.1 Attributes of MiddleEngine Class

- private InnerEngine innerEngine: It is used to transfer the data between the middle engine and the inner engine while playing the game.
- private boolean isOnline: It is used to keep the information about whether the game is online or not.
- private GameMode gameMode: It is used to keep the game mode information.
- private GameTheme gameTheme: It is used to keep the game theme information.
- private String hosterNick: It is used to store the nickname of the host who created the lobby.
- private ArrayList<Pawn> pawns: It is used to keep the list of pawns used by the players.
- private ImageView themeHalloween: It is used to apply the halloween theme of the game.
- private ArrayList<Boolean> isAllReady: It is used to keep the data on whether the players are ready for the game in order to be able to start the game.
- private int maxPlayerCount: It is used to keep the maximum number of players in the game.
- private String lobbyPassword: It is used to keep the lobby password of the current game.
- private ArrayList<Integer> botIds: It is used to store bot players' id numbers.
- private ArrayList<Player> players: It is used to store the players who currently play the game.

4.4.4.2 Methods of MiddleEngine Class

- public void initializeGame(): It initializes the game if all the conditions for starting the game are satisfied.
- public boolean kickplayer(nick : String): It discards the player from the game according to their nickname.
- public boolean addBot(): It adds bot player to the game.

4.4.5 OuterEngine Class

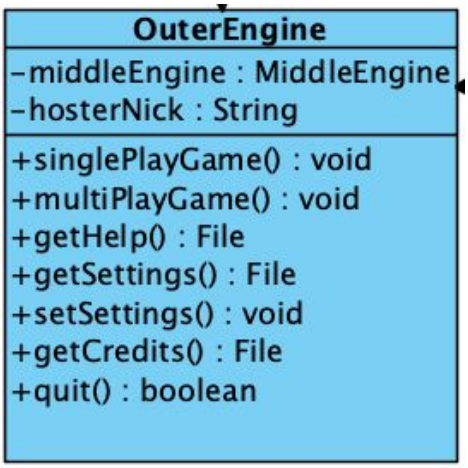

Figure 12: OuterEngine Class Diagram

OuterEngine class consists of the opening page of the game. Settings and information about the game is located here, before the creation of the game.

4.4.5.1 Attributes of OuterEngine Class

- private MiddleEngine middleEngine: It is used to keep data from the lobby section of the game which is related to the middle engine.
- private String hosterNick: It is used to store the nickname of the host who created the lobby, coming from the lobby screen controller.

4.4.5.2 Methods of OuterEngine Class

- public void singlePlayGame(): It initializes the single player game.
- public void multiPlayGame(): It initializes the multiplayer game.
- public File getHelp(): It initializes the help section of the main menu.
- public File getSettings(): It gets the settings of the game.
- public void setSettings(): It changes the settings of the game to the desired settings.
- public File getCredits(): It gets the credits part of the game.
- public boolean quit(): It closes the game.

4.4.6 PropertyCard Class

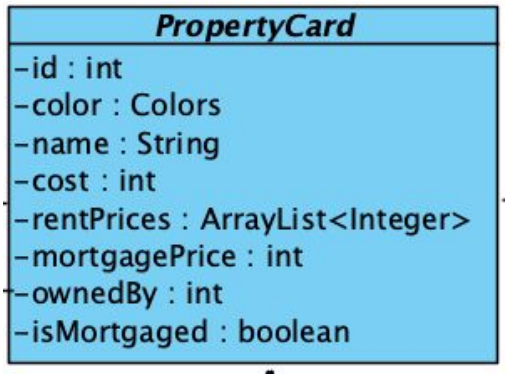

Figure 13: PropertyCard Class Diagram

PropertyCard class is the representative cards which are generated for the properties on squares. Every PropertyCard object has a unique id and some other properties.

4.4.6.1 Attributes of PropertyCard Class

- private int id: It keeps the id of the property.
- private Colors color: It keeps the color of the property cards.
- private String name: It keeps the name of the property card.
- private ArrayList<Integer> rentPrices: It keeps the list of rent prices of properties.
- private int mortgagePrice: It keeps the mortgage price of the properties.
- private int ownedBy: It keeps the information about the owner of the property.
- private boolean isMortgaged: It keeps the information about whether the property is mortgaged or not.

4.4.7 Square Class

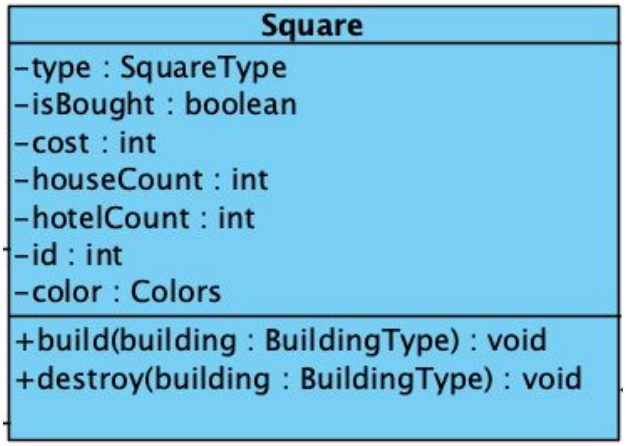

Figure 14: Square Class Diagram

Square class is used for the square representations of the board. It includes type of the square information and some other additional features related attributes.

4.4.7.1 Attributes of Square Class

- private SquareType type: It is used to keep the type of the square.
- private boolean isBought: It is used to keep the information about whether the square is bought by someone or not.
- private int cost: It is used to keep the current cost of the square.
- private int houseCount: It is used to keep the number of houses on a square.
- private int hotelCount: It is used to keep the number of hotels on a square.
- private int id: It keeps the id of the square.
- private Colors color: It is used to keep the color of the square.

4.4.8 SquareType Enum Class

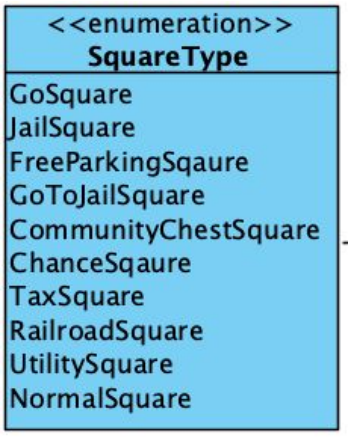

Figure 15: SquareType Enum Class Diagram

This enumeration class is used to store ten available square types on the board such as GoSquare, JailSquare, FreeParkingSquare.

4.4.9 GameTheme Enum Class

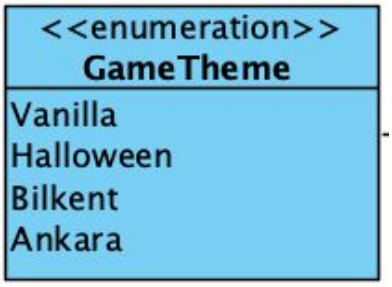

Figure 16: GameTheme Enum Class Diagram

This enumeration class is used to store different types of game themes available at the game which are Vanilla, Halloween, Bilkent and Ankara.

4.4.10 GameMode Enum Class

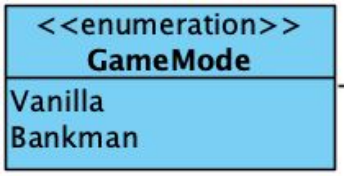

Figure 17: GameMode Enum Class Diagram

This enumeration class is used to store two types of game modes available in the game which are Vanilla and Bankman.

4.4.11 Pawn Enum Class

Figure 18: Pawn Enum Class Diagram

This enumeration class is used to keep the eight different types of pawns that can be chosen by the users. These eight types are Skate, Formula, Plain, Motorbike, Sailboat, Car, Hat and Shoe.

4.4.12 Board Class

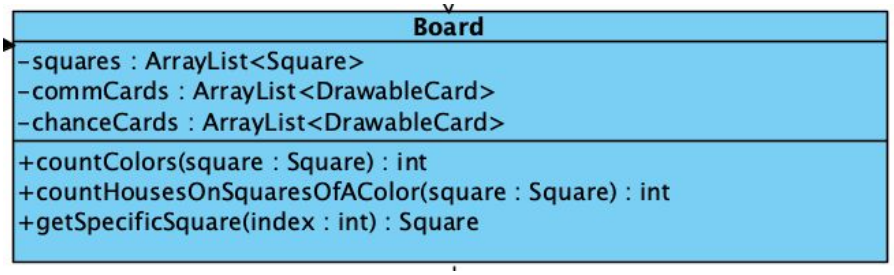

Figure 19: Board Class Diagram

Board class includes all kinds of attributes which can be located on the board. This class' attributes constitute the play.

4.4.12.1 Attributes of Board Class

- private ArrayList<Square> squares: It is used to keep the list of all the squares located on the board.
- private ArrayList<DrawableCard> commCards: It is used to keep the list of community chest cards available on the board at that time.
- private ArrayList<DrawableCard> chanceCards: It is used to keep the list of chance cards available on the board at that time.

4.4.12.2 Methods of Board Class

- public int countColors(square: Square): It counts the colors in the board according to the color given through the Square parameter.
- public int countHousesOnSquaresOfAColor(square: Square): It counts the number of houses on every square on the board which has the same colors.
- public Square getSpecificSquare(index: int): It returns the specific square through the given index parameter.

4.4.13 Currency Class

Figure 20: Currency Class Diagram

Currency class represents the different types of currencies in the game.

4.4.13.1 Attributes of Currency Class

- private String name: It is used to keep the name of the currency.
- private Double rate: It is used to keep the rate of the currency.

4.4.14 Bank Class

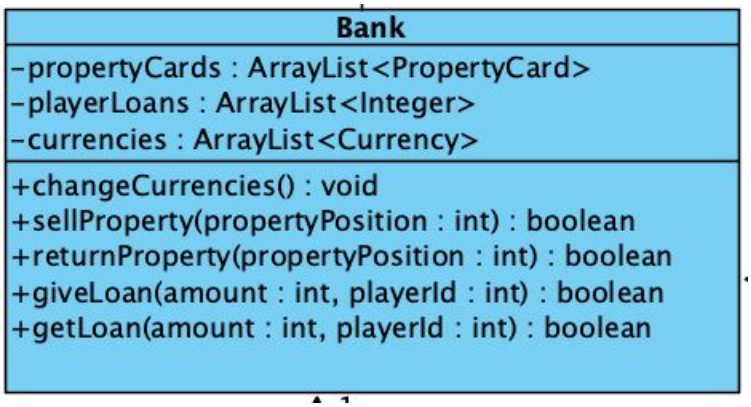

Figure 21: Bank Class Diagram

Bank Class includes actions and attributes that the players are able to do with the bank. It keeps track of cash flow in the game according to the actions of the players'.

4.4.14.1 Attributes of Bank Class

- private ArrayList<PropertyCard> propertyCards: It is used to keep the list of property cards to be used while purchasing.
- private ArrayList<Integer> playerLoans: It is used to keep the list of money loaned by the players.
- private ArrayList<Currency> currencies: It is used to keep the list of currencies.

4.4.14.2 Methods of Bank Class

- public void changeCurrencies(): It changes the currencies and their rates.
- public boolean sellProperty(propertyPosition : int): It sells the property to a user.
- public boolean returnProperty(propertyPosition : int): It returns the property from the user.
- public boolean giveLoan(amount : int, playerId : int): It gives loan to the player.
- public boolean getLoan(amount : int, playerId : int): It gets back the money the bank loaned to the player.

4.4.15 PlaceCard Class

Figure 22: PlaceCard Class Diagram

PlaceCard class includes the information about the prices of the places.

4.4.15.1 Attributes of PlaceCard Class

- private int housePrice: It is used to keep the price of the house.
- private int hotelPrice: It is used to keep the price of the hotel.

4.4.16 RailroadCard Class

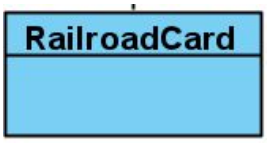

Figure 23: RailroadCard Class Diagram

This class is created in order to specify railroad cards.

4.4.17 Dice Class

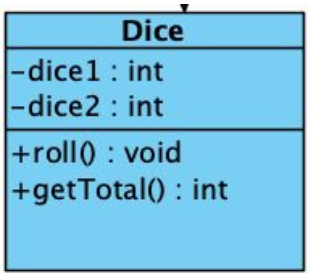

Figure 24: Dice Class Diagram

4.4.17.1 Attributes of Dice Class

- private int dice1: It is used to keep the value of the first dice rolled.

- private int dice2: It is used to keep the value of the second dice rolled.

4.4.17.2 Methods of Dice Class

- public void roll(): It rolls the two dices.
- public int getTotal(): It calculates the sum of the values of two dices.

4.4.18 Building Enumeration Class

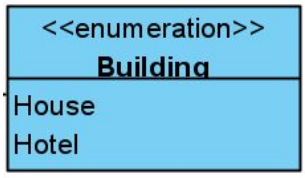

Figure 25: Building Enumeration Class Diagram

This enumeration class is used to specify two different building types which are house and hotel.

4.4.19 CommunityChestCard Class

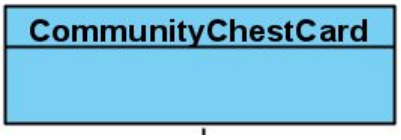

Figure 26: CommunityChestCard Class Diagram

This class is created for the community chest card type.

4.4.20 ChanceCard Class

Figure 27: ChanceCard Class Diagram

This class is created for the chance card type.

4.4.21 UtilityCard Class

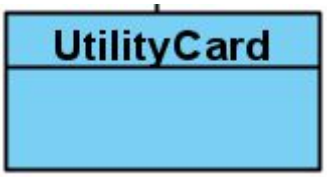

Figure 28: UtilityCard Class Diagram

This class is created for the utility card type.

4.4.22 Colors Enumeration Class

Figure 29: Colors Enumeration Class Diagram

This enumeration class is used to specify nine different colors for property cards and squares.

4.4.23 Outer Controller

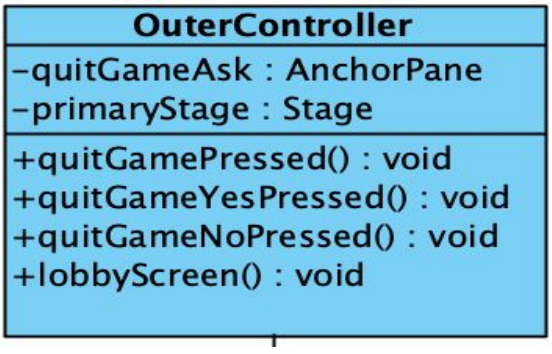

Figure 30: OuterController Class Diagram

This class is used to control the Main Menu screen of the game. Basically a controller class.

4.4.23.1 Attributes of OuterController Class

- private AnchorPane quitGameAsk: It is used to ask whether the player wants to quit the game or not.
- private Stage primaryStage: It is used to connect the entry screen and the game play section of the game.

4.4.23.2 Methods of OuterController Class

- public void quitGamePressed(): It is used when the quit game button is clicked.
- public void quitGameYesPressed(): It is used when the quit game question is answered as yes by the player.
- public void quitGameNoPressed(): It is used when the quit game question is answered as no by the player.
- public void lobbyScreen(): It is used to change the screen.

4.4.24 Middle Controller

Figure 31: MiddleController Class Diagram

This class is used to control the Game Settings/Lobby screen of the game. Basically a controller class.

4.4.24.1 Attributes of MiddleController Class

- private ImageView modeVanilla: It is used to switch the mode of the game to Vanilla.
- private ImageView modeBankman: It is used to switch the mode of the game to Bankman.
- private ImageView themeVanilla: It is used to switch to the Vanilla Theme of the game.
- private ImageView themeBilkent: It is used to switch to the Bilkent Theme of the game.
- private ImageView themeAnkara: It is used to switch to the Ankara Theme of the game.

4.4.24.2 Methods of MiddleController Class

- public void modeVanilla(): It is used to switch to Vanilla mode of the game.
- public void modeBankman(): It is used to switch to Bankman mode of the game.
- public void themeVanilla(): It is used to switch to the Vanilla theme of the game.
- public void themeBilkent(): It is used to switch to the Bilkent theme of the game.
- public void themeAnkara(): It is used to switch to the Ankara theme of the game.
- public void themeHalloween(): It is used to switch to the Halloween theme of the game.
- public void closeButtonPressed(): It is used to close the setting page.
- public void readyButtonPressed(): It is used to satisfy the condition of starting a game.

4.4.25 Inner Controller

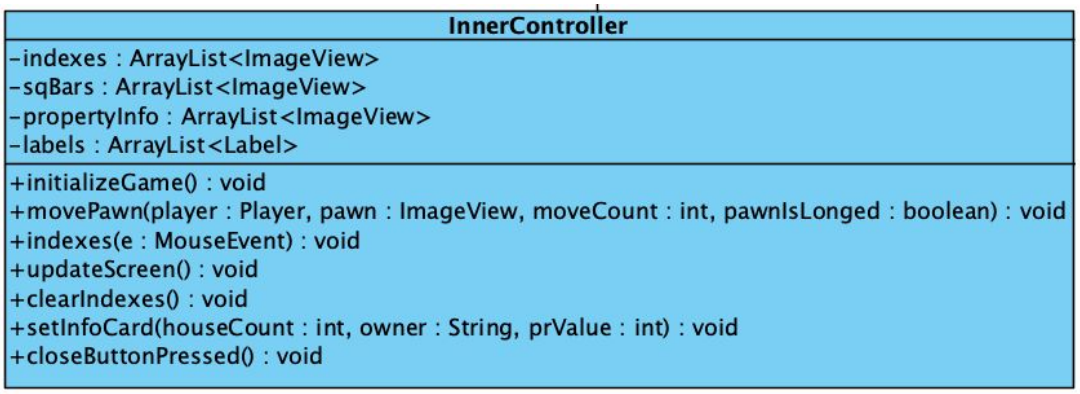

Figure 32: InnerController Class Diagram

This class is used to control the In-Game screen of the Game. Basically a controller class.

4.4.25.1 Attributes of InnerController Class

- private ArrayList<ImageView> indexes: It is used to list the squares on the board.
- private ArrayList<ImageView> sqBars: It is used to list the squares with either hotels or houses built on them.
- private ArrayList<ImageView> propertyInfo: It is used to list the information of properties.
- private ArrayList<Label> labels: It is used to list the labels.

4.4.25.2 Methods of InnerController Class

- public void initializeGame(): It is used to initialize the game.
- public void movePawn(player: Player, pawn: ImageView, moveCount: int, pawnsIsLonged: boolean): It is used to move the pawn from its current place.
- public void indexes(e: MouseEvent): It is used to hold the index of the square.
- public void updateScreen(): It is used to update the screen according to the changes.
- public void clearIndexes(): It is used to clear the indexes.
- public void setInfoCard(houseCount: int, owner: String, prValue: int): It is used to set info value.

- public void closeButtonPressed(): It is used after the close button is pressed and it closes the screen.

5. References

- Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design and Iterative Development,by Craig Larman, Prentice Hall, 2004, ISBN: 0-13-148906-2
- *Object-Oriented Software Engineering*, by Timothy C. Lethbridge and Robert Laganiere, McGraw-Hill, 2001, ISBN: 0-07-709761-0
- https://docs.oracle.com/