

CS 319 Term Project

Design Report Phase 2

**Team Name: Group 2H CA**

**Project Name: Settlers of Catan**

**Members:**

• Ravan Aliyev

• Berrak Taşkınsu

• Hazal Aksu

• Ayşe Nursu Savaşkan

# Table of Contents

[**Table of Contents**](#_2qz1xa5derst) **2**

[**1. Introduction**](#_7k570g3pcuzh) **4**

[1.1. Purpose of the System](#_ajp1zuv7h0o5) 4

[1.2. Design Goals](#_s9fr625xlzt6) 4

[**2. High-level Software Architecture**](#_vtienzxl8wpl) **5**

[2.1. Subsystem Decomposition (Packages)](#_ociqkg9m6pqb) 5

[2.2. Hardware/Software Mapping](#_2onl4e315yqu) 7

[2.3. Persistent Data Management](#_ufjkpygfq5if) 7

[2.4. Access Control and Security](#_5mmn8s2q2mam) 7

[**3. Subsystem Services**](#_8af88nn5wwaf) **8**

[3.1. GUI Package](#_rp95slfnee8b) 8

[3.1.1. View](#_9xw6x139bh40) 9

[3.1.2. MainMenuWindow](#_67k1dvoetlzh) 9

[3.1.3. HowToPlayWindow](#_4w9ybzuah8na) 10

[3.1.4. SettingsWindow](#_u8zeqwwsmv09) 12

[3.1.5. NewGameWindow](#_vn9cv0t8zusj) 13

[3.1.6. GameWindow](#_vn9cv0t8zusj) 15

[3.1.7. OfferTradeWindow](#_vn9cv0t8zusj) 16

[3.1.8. TradeWindow](#_bvy30o449258) 18

[3.1.9. HexButton](#_jsixetkdgmxt) 20

[3.1.10. CornerButton](#_jcflcun095ti) 20

[3.1.11. PathButton](#_xaobw7mje562) 20

[3.1.12. SettlersButton](#_druh13y7oy9z) 21

[3.1.13. BackButton](#_txkvykcompac) 21

[3.2. Game Package](#_qiudmga0m88k) 22

[Figure 19: GamePackage](#_sq45it56owlg) 22

[3.2.1. Controller](#_ys0ds225fvgx) 23

[3.2.2. Bank](#_rc48q94qetce) 25

[3.2.3. Player](#_ti3jkcu8zjrq) 27

[3.2.4. GameBoard](#_tcwiwnitbjdx) 29

[3.2.5. Card](#_tcwiwnitbjdx) 30

[3.2.6. DevelopmentCard](#_ohm3zdv6o2za) 30

[3.2.7. ResourceCard](#_uaw1hlsqf19) 30

[3.2.8. Hex](#_tcwiwnitbjdx) 31

[3.2.9. Path](#_tcwiwnitbjdx) 32

[3.3. Controller Package](#_dgpu909s9o6y) 33

[3.3.1. ViewControllerController](#_aogaqn3mxolz) 34

[3.3.2. ViewController](#_lu3bifr91uh0) 34

[3.3.4. MainMenuWindowController](#_gqr0gnkybohm) 34

[3.3.5. SettingsWindowController](#_p061x51gnvyb) 35

[3.3.6. HowToPlayWindowController](#_wr0zffp01vl) 35

[3.3.7. NewGameWindowController](#_9rt1dev2w10q) 35

[3.3.8. GameWindowController](#_4vjdzab8t94f) 35

[3.3.9. OfferTradeWindowController](#_75b7lu7ibxka) 36

[3.3.10. TradeWindowController](#_oxsalxu8dj8i) 36

[**4. Low-level Design**](#_8pux3sikokcb) **37**

[4.1 Object Design Trade-Offs](#_24eqxnq9yj8u) 37

[4.2. Final Object Design](#_wbudbtf15hmc) 38

[4.3. Packages](#_mj38yhvssvh6) 41

[4.4. Class Interface](#_ys1b8zhd7qaa) 41

[**5. Glossary & References**](#_5g3gms5qr2td) **43**

# 

# 1. Introduction

In this section purpose of the system and our main design goals are explained.

## 1.1. Purpose of the System

Settlers of Catan is a multiplayer board game. The purpose of our program is to create the digital version of an already existing physical game and to make the game more attractive by adding different features and creating a new mode to satisfy the users. In addition to the original version of the game, there is a new mode which is single player mode where user plays against the Artificial Intelligence. This game is planned to be platform-independent software, which allows this game to be played on Linux, Mac OS and Microsoft Windows operating systems. This software will be user-friendly, which allows the users to be able to learn how to play this game easily. This competitive game will challenge the players and entertain them by improving their analything thinking abilities and spatial reasoning aptitude. The main goal is getting ten points before other players by building cities, settlements and roads and by the help of the development cards.

## 

## 1.2. Design Goals

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.[1] These requirements are listed in functional requirements sections in the analysis report of this software and these requirements will be discussed below. Design goals are qualities or properties that software has a solution for these properties. They are found by analyzing the requirements and the information that we have about the game and will give us a direction while developing the software. Significant design goals for the digital version of our game are listed below:

***Performance Goals:*** Response time of the software will not take time more than 200 ms. Free space in memory for the software will be less than 0.4 GB .

***Maintenance Goals:*** The game should be implemented as a multi-platform software, which allows it to be played on Windows, MacOS and Linux operating systems in which JRE 8 is supported. Extensibility is also an important software designing criteria for maintenance, which we should pay attention while developing the software. This criteria increases the ability of the software for further extension and improvements. It also allows other developers to add new functions or modify the existing software too and this extensibility criteria provides for modifications without impairing the existing software. Project related documents will be open source too, and this will help other developers to redesign our system. To make it easy to understand our software by reading the codes, we have to pay attention to another maintenance criteria which is readability.

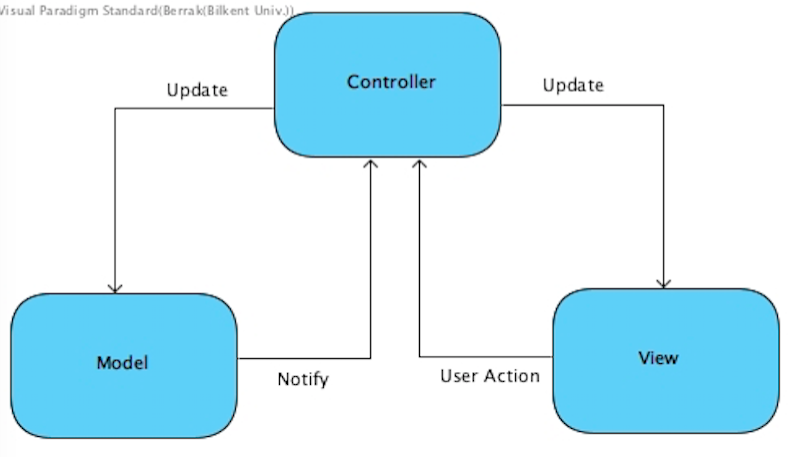
***Dependability Goals:*** Simplicity of our system allows it not to have any security risks, because it does not contain any personal information. Simplicity of our system lets it to have a high reliability.

***End User Goals:*** Usability is a criteria that shows the ease of use and learnability by users. Our system can be used by every user that has an ability to use a computer to achieve objectives in game with effectiveness, efficiency and satisfaction.

# 2. High-level Software Architecture

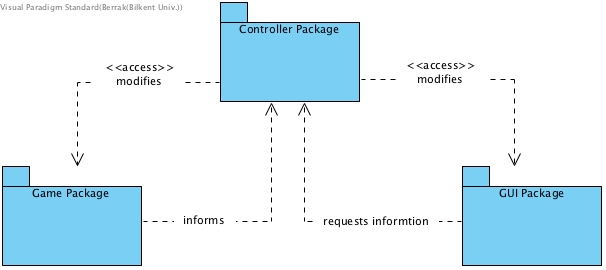
## 2.1. Subsystem Decomposition (Packages)

There are three packages in our software: Controller Package, GUI Package and Game Package. The GUI Package contains the classes subject to the user interface and the Game Package contains the model classes.

We used the MVC pattern in our project. The below figure is a visualization of MVC pattern.

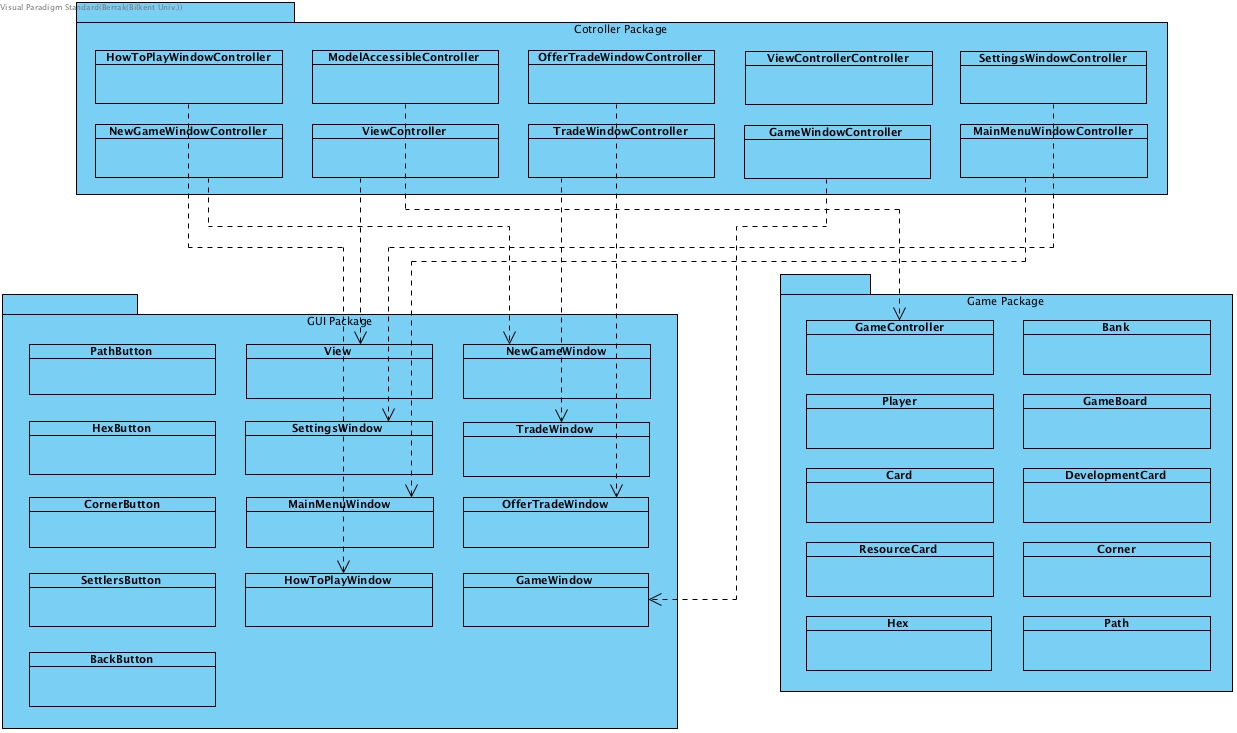
*Figure 1: MVC Pattern*

Additionally, the below figure represents the MVC pattern applied to our packages.



*Figure 2: Packages Overview*

The interactions between the model elements and the view elements occur through the Controller classes.



*Figure 3: Packages*

## 2.2. Hardware/Software Mapping

For the implementation of our software we do not use any external components, so there is not any distinction between user machines and server. That is why we do not do any hardware/software mapping.

## 2.3. Persistent Data Management

Only 3 data need to be stored for the Settlers of Catan game: saving settings, achievements and highscores. Our software does not need to save complex data since it is a simple game. Achievements and highscores will be given according to the players’ progresses. Those progresses and settings can be saved as a combination of strings and integers arrays. Because of the simplicity of this software, it does not require access to other applications and so, no application does need to be able to access data. Since this small amount of information needs to be stored, we will not use high level of data storage. Considering the points listed above, we think that using database to store the data of the game is not necessary and will be excessive. So, using .txt files instead of database will allow our program to work much faster.

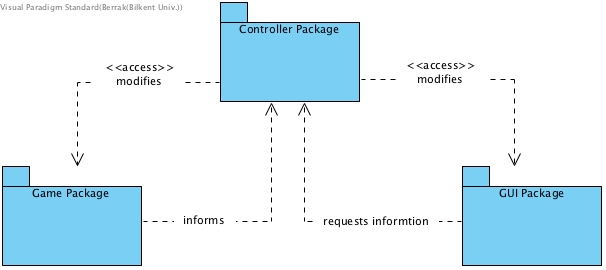
## 2.4. Access Control and Security

Subsystems of our software is shown at section 2.1 by explaining how and why they are chosen in such a way. Our game does not need to have access to the internet, which makes it secure, because it is not open to dangers coming from the internet. Single player mode will be played against Artificial Intelligence. Even multiplayer mode does not need internet access to the internet because it will be played on users’ local devices.

2.5. Boundary Conditions

This software does not need to be installed which means that it has not any executable .exe extension file. The game will be executed from the .jar file, enabling it to be portable among different operating systems and devices. Since it will be executed from .jar file internet connection is not needed to open the game. Playing the game on the local device and no need for the internet ensures that there will be no network or connection problem in the game. Even though there can be other failures too. For example, user terminates the game which can cause loss of the current game situation. This can cause potential data loss. Another problem can occur while reading files. Problem which occurs while reading files can cause to the game start without a sound or images. This problem can be fixed by modifying the corrupted file.

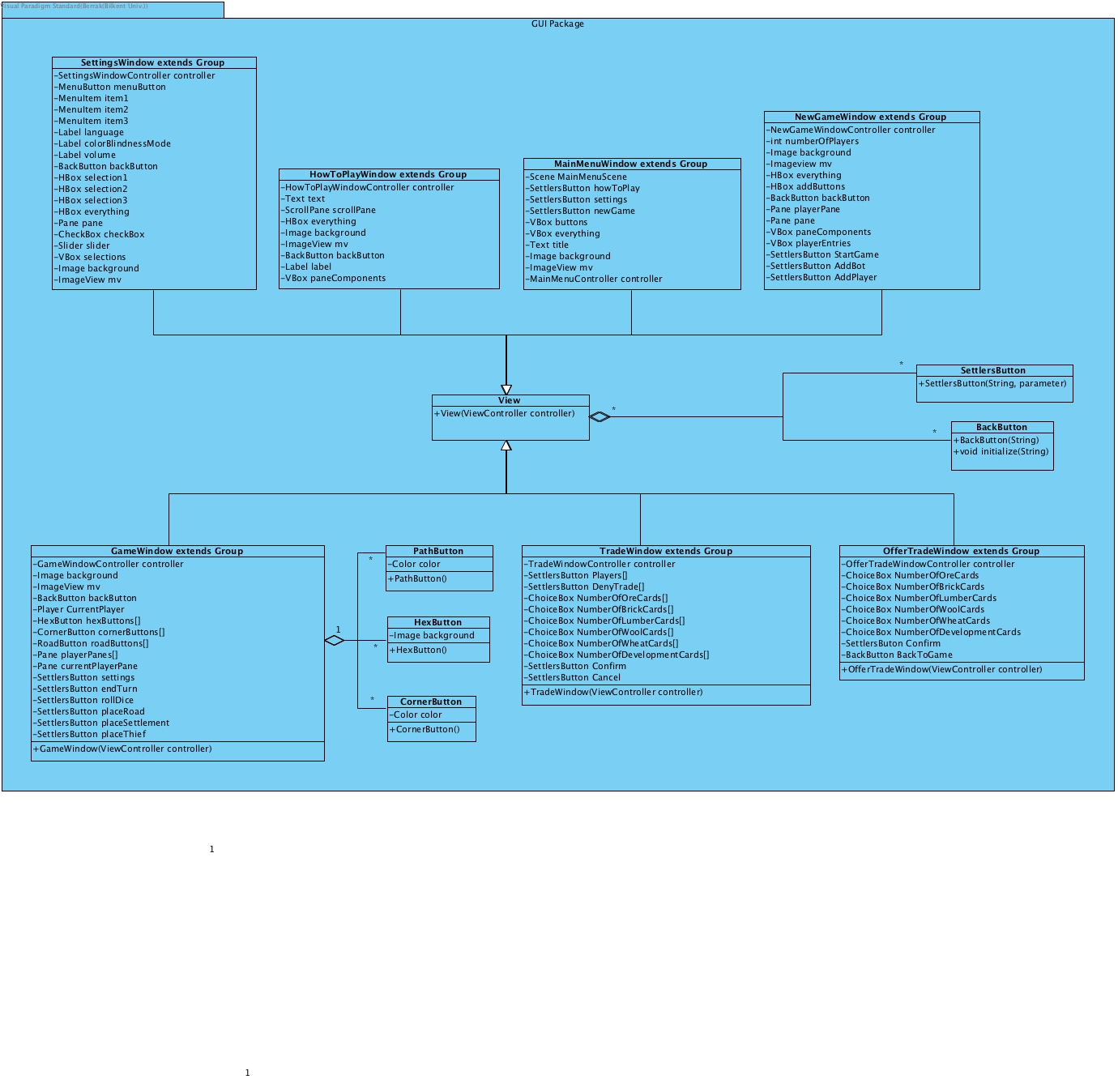
# 3. Subsystem Services



*Figure 4: Packages Overview*

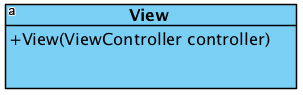
## 3.1. GUI Package

The UML Class Diagram for the GUI Package is provided below.



*Figure 5: GUI Package*

### 3.1.1. View

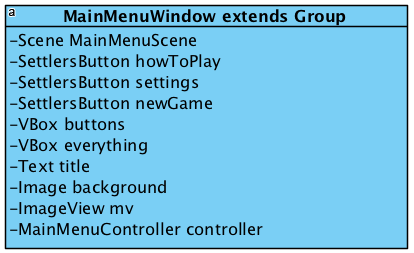


*Figure 6: MainMenuView*

**Constructors:**

* View(ViewController controller):

### 3.1.2. MainMenuWindow



*Figure 7: MainMenuWindow*

**Attributes:**

* private SettlersButton newGame: allows the user to go to the StartGamePanel. When clicked, MainMenuWindow will access the MainPanel via the getMainPanel() method it inherited from the abstract class SubPanel and the changeSubPanel() method of the MainPanel will be called to change the current panel into a StartGamePanel.
* private SettlersButton HowToPlay: allows the user to go to the HowToPlayPanel. When clicked, MainMenuPanel will access the MainPanel via the getMainPanel() method it inherited from the abstract class SubPanel and the changeSubPanel() method of the MainPanel will be called to change the current panel into an HowToPlayPanel.
* VBox Buttons: keeps all buttons
* VBox everything: keeps everything in the window
* Image background: keeps background image of the Main menu
* ImageView mv:
* MainMenuController controller: controller to control main menu
* Text title: to display title
* private SettlersButton settings: allows the user to go to the SettingsPanel. When clicked, MainMenuPanel will access the MainPanel via the getMainPanel() method it inherited from the abstract class SubPanel and the changeSubPanel() method of the MainPanel will be called to change the current panel into a SettingsPanel.

### 

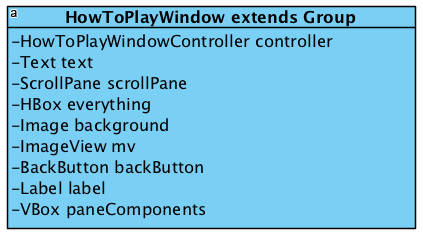
### 

### 

### 

### 

### 3.1.3. HowToPlayWindow

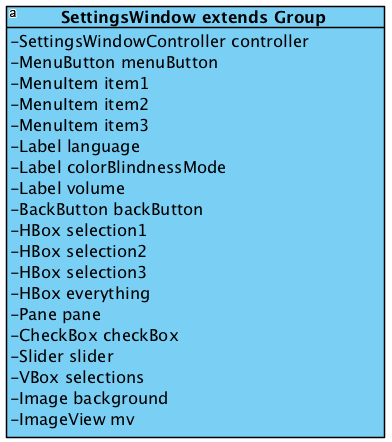


*Figure 8: HowToPlayWindow*

**Attributes:**

* HowToPlayWindowContoller controller: controller to control HowToPlayWindow
* Text text: needed to display the text
* ScrollPane scrollPane: to make it able to display whole text
* Hbox everything: keeps everything in How To Play window
* Image background: needed for background image
* ImageView mv
* BackButton backButton: allows the user to go to the MainMenuPanel. When clicked, HowToPlayWindow will access the MainPanel via the getMainPanel() method it inherited from the abstract class SubPanel and the changeSubPanel() method of the MainPanel will be called to change the current panel into a MainMenuPanel.
* Label label: To display title
* VBox paneComponents: keeps components of the How To play window

### 3.1.4. SettingsWindow

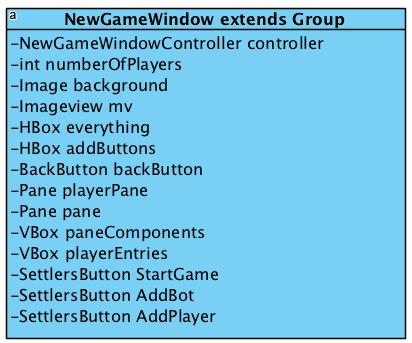


*Figure 9: SettingsWindow*

**Attributes:**

* SettingsWindowController controller: Controller to control the Settingswindow
* Menubutton menuButton: needed to go back to Main Menu
* MenuItem item1: displays menu item
* MenuItem item2: displays menu item
* MenuItem item3: displays menu item
* Label language: needed to indicate language selection part
* Label volume: needed to indicate volume setting part
* Label colorBlindnessMode: needed to indicate color blindness mode part
* HBox selection1: allows the user to select between English, Turkish, Azerbaijani and Russian.
* HBox selection2: allows the user to switch to the color blindness mode.
* HBox selection3: allows the user to set the volume of the game.
* Hbox everything: Keeps all HBox’s
* Pane pane
* CheckBox checkBox: to choose/unchoose color blindness mode
* Slider slider: to adjust volume of the background music
* VBox selections: to allow to select language
* Image background: needed for background image
* imageView mv:
* BackButton backButton: allows the user to go to the previous SubPanel, which can either be a MainMenuPanel or a GamePanel. When clicked, SettingsPanel will access the MainPanel via the getMainPanel() method it inherited from the abstract class SubPanel and the changeSubPanel() method of the MainPanel will be called to change the current panel depending on the attribute PreviousPanel.

### 3.1.5. NewGameWindow



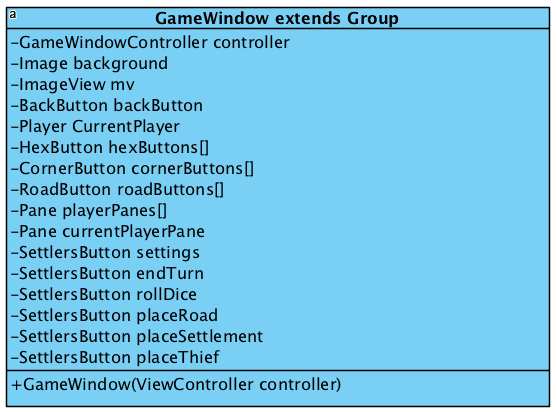
*Figure 10: NewGameWindow*

**Attributes:**

* NewGameWindowController controller: Controller to control NewGameWindow
* int numberOfPlayers: the number of human players.\*
* Image background: needed for background image
* imageview mv:
* HBox everything: Keeps all HBox’s
* HBox addButtons:
* BackButton backButton: allows the user to go to the MainMenuWindow.
* Pane playerPane:
* Pane pane:
* VBox paneComponents: keeps all components
* VBox playerEntries:
* SettlersButton StartGame: allows the user to go to the GameWindowl.
* SettlersButton AddBot: allows user to add new NPC
* SettlersButton addPlayer: allows users to add human players

\*The game allows 3 or 4 players, therefore the user must make a selection accordingly.

### 3.1.6. GameWindow



*Figure 11: GameWindow*

**Attributes:**

* GameWindowController controller: Controller to control GameWindow
* Image background: needed to display background image
* ImageView mv:
* BackButton backButton: direct the game to MainMenuWindow
* Player currentPlayer: shows the current player
* HexButton hexButtons[]: needed for creating the map of the game, located inside hexes
* CornerButton cornerButtons[]: needed for creating the map of the game, helps to locate cities and settlements
* RoadButton roadButtons[]: located between corners of the hexes, helps to locate roads
* Pane playerPanes[]: keeps panes for all players
* Pane currentPlayerPane: needed for the pane of the current player
* SettlersButton Settings: Opens SettingWindow, allowing players to change settings during game
* SettlersButton endTurn: this button ends the turn of the current player
* SettlersButton rollDice: needed to roll the dice
* SettlersButton placeRoad: needed to place road, player clicks this button before placing road
* SettlersButton placeSettlement: needed to place settlement, player clicks this button before placing settlement
* SettlersButton placeThief: needed to place thief, player clicks this button before placing thief

### 3.1.7. OfferTradeWindow

### 

*Figure 12: OfferTradeWindow*

**Attributes:**

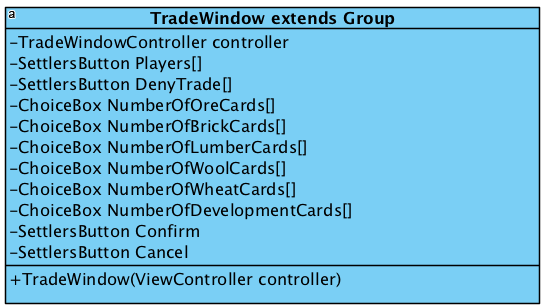
* OfferTradeWindowController controller: Controller to control OfferTradeWindow
* private ChoiceBox NumberOfOreCards: allows the user to select the desired number of ore cards.
* private ChoiceBox NumberOfBrickCards: allows the user to select the desired number of brick cards.
* private ChoiceBox NumberOfLumberCards: allows the user to select the desired number of lumber cards.
* private ChoiceBox NumberOfWoolCards: allows the user to select the desired number of wool cards.
* private ChoiceBox NumberOfWheatCards: allows the user to select the desired number of wheat cards.
* private ChoiceBox NumberOfDevelopmentCards: allows the user to select the desired number of development cards.
* SettlersButton Confirm: allows the user to confirm the trade request in terms of the desired cards. When clicked, OfferTradePanel will access the MainPanel via the getMainPanel( ) method it inherited from the abstract class SubPanel and the changeSubPanel( ) method of the MainPanel will be called to change the current panel into a TradePanel.
* BackButton BackToGame: allows the user to discard trade request. When clicked, OfferTradePanel will access the MainPanel via the getMainPanel( ) method it inherited from the abstract class SubPanel and the changeSubPanel( ) method of the MainPanel will be called to change the current panel into a GamePanel.

### 

### 

### 

### 3.1.8. TradeWindow

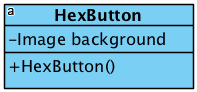
****

*Figure 13: TradeWindow*

**Attributes:**

* TradeWindowController controller: Controls the tradeWindow
* private SettlersButton Players[ ]: a list to store all Button’s for all players except for the player in turn. When clicked, the player will be selected by the user in turn.
* private Button DenyTrade[ ]: a list to store all Button’s for all players except for the player in turn. Available for a player if it is the trading turn of that player
* private ChoiceBox NumberOfOreCards[ ]: a list of all ChoiceBox’es to chose the number of desired ore cards, for all players except the one in turn. Available for a player if it is the trading turn of that player and the player has accepted the trade offer.
* private ChoiceBox NumberOfBrickCards[ ]: a list of all ChoiceBox’es to chose the number of desired brick cards, for all players except the one in turn. Available for a player if it is the trading turn of that player and the player has accepted the trade offer.
* private ChoiceBox NumberOfLumberCards[ ]: a list of all ChoiceBox’es to chose the number of desired lumber cards, for all players except the one in turn. Available for a player if it is the trading turn of that player and the player has accepted the trade offer.
* private ChoiceBox NumberOfWoolCards[ ]: a list of all ChoiceBox’es to chose the number of desired wool cards, for all players except the one in turn. Available for a player if it is the trading turn of that player and the player has accepted the trade offer.
* private ChoiceBox NumberOfWheatCards[ ]: a list of all ChoiceBox’es to chose the number of desired wheat cards, for all players except the one in turn. Available for a player if it is the trading turn of that player and the player has accepted the trade offer.
* private ChoiceBox NumberOfDevelopmentCards[ ]: a list of all ChoiceBox’es to chose the number of desired development cards, for all players except the one in turn. Available for a player if it is the trading turn of that player and the player has accepted the trade offer.
* private SettlersButton Confirm: allows a player to confirm what resources he/she wants in exchange for what the player in turn wants. When clicked, space allocated for the player will be highlighted in green, all functionality of the player will be lost and the trading turn will move on to the next player.
* private SettlersButton cancel: cancels the trade if the player does not want to trade

### 3.1.9. HexButton



*Figure 14: HexButton*

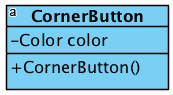
**Attributes:**

* Image backGround: background image for the HexButton’s

**Constructors:**

* HexButtons(): constructor for HexButton’s

### 3.1.10. CornerButton



*Figure 15: CornerButton*

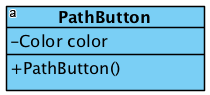
**Attributes:**

* Color color: keeps the color of the corner

**Constructors:**

* CornerButton(): constructor for CornerButton

### 3.1.11. PathButton



*Figure 16: PathButton*

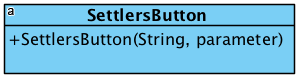
**Attributes:**

* Color color: keeps the color of PathButton

**Constructors:**

* PathButton():

### 3.1.12. SettlersButton

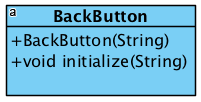


*Figure 17: SettlersButton*

**Constructors:**

* SettlersButton(String,parameter):

### 3.1.13. BackButton



*Figure 18: BackButton*

**Constructors:**

* Backbutton(String):

**Methods:**

* void initialize(String): initialize method helps the button to determine which window to go back to

## 3.2. Game Package

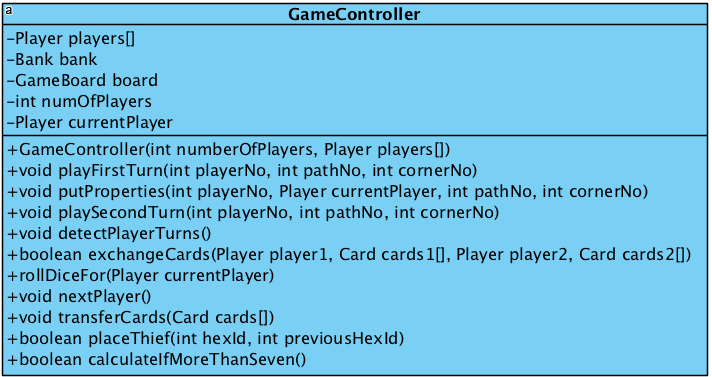
### 

### *Figure 19: GamePackage*

### 

### 

### 3.2.1. Controller



*Figure 20: Game*

**Attributes:**

* Player players[]: keeps all players
* Bank bank: object of the bank class, which keeps all resource and development cards
* GameBoard board: object of the Gameboard, one board object is needed for the game
* int numOfPlayers: shos number of the players
* Player currentPlayer: shows current player

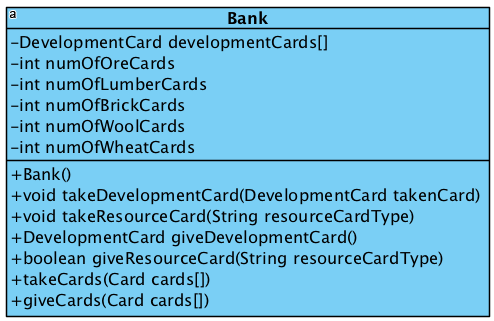
**Constructors:**

* GameController(int numberOfPlayers, Player players[]):

**Methods:**

* void playFirstTurn(int playerNo, int pathNo, int cornerNo): allows the players to play first turn, at this turn every player can place one settlement and one road without any charge
* void putProperties(int playerNo, Player currentPlayer, int pathno, int cornerNo):
* void playSecondturn(int playerNo, int pathNo, int cornerNo): allows the players to play second turn, at this turn every player can place one settlement and one road without any charge and they get resources from adjacent hexes after placing their settlements
* void detectPlayerTurns: determines the turns of the players
* boolean exchangeCards(Player1, Card cards1[], player2, Card cards2[]); needed to trade cards between 2 players
* rollDiceFor(Player currentPlayer): rolls dice when rollDice button is clicked and displays the result to everyone
* void nextPlayer(); changes the turn to the next player when the current player ends turn
* void transfercards(Card cards[]): transfers cards to the bank
* boolean placeThief(int hexid, int previousHexId): needed to move the thief to the new hex
* boolean calculateifMoreThanSeven: calculates the number of resource cards of each player, and returns true if there are anyone with more than seven resource cards

### 3.2.2. Bank



*Figure 21: Bank*

**Attributes:**

* int numOfOreCards: keeps number of Ore cards
* int numOfLumberCards: keeps number of Lumber cards
* int numOfBrickCards: keeps number of Brick cards
* int numOfWoolCards: keeps number of Wool cards
* int numOfWheatCards: keeps number of Wheat cards
* DevelopmentCard developmentCard[]; keeps all cards

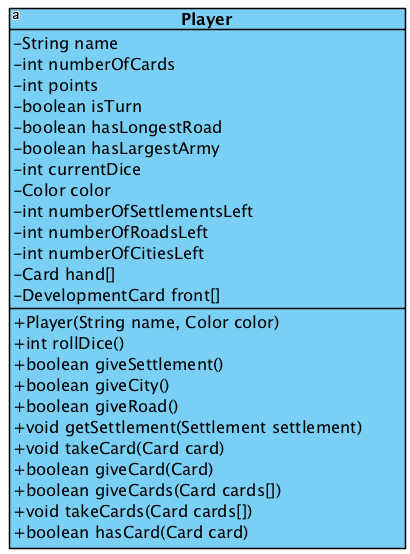
**Constructors:**

* Bank():

**Methods:**

* void takeDevelopmentCard(DevelopmentCard takencard): takes development cards and adds it to the developmentCards[]
* void takeResourceCard(String resourceCardType): takes resource card and increments according resource card number
* DevelopmentCard giveDevelopmentcard(): gives development card randomly and removes it from developmentCard[]
* boolean giveResourceCard(String resourceCardType): gives resource card according to the argument and decrements number of according resource card
* takeCards(Card cardds[]): takes cards from player and increments number of according cards
* giveCards(Card cards[]): gives cards to players and decrements number of according cards

### 3.2.3. Player



*Figure 22: Player*

**Attributes:**

* private String name: stores the names of a player
* int numberOfCards stores number of cards of the player
* int points: stores the sccore of the player
* boolean isTurn: shows whether it is that player’s turn or not
* boolean hasLongestRoad: shows whether that player has longest road or not
* boolean hasLargestArmy: shows whether that player has largest army or not
* int currentDice: shows the current dice situation
* Color color: stores the color of the player
* int numberOfSettlementsLeft: stores the number of the settlements left
* int numberOfRoadsLeft: stores the number of roads left
* int numberOfCitiesLeft: stores the number of cities left
* Card hand[]: stores the cards of the player
* DevelopmentCard front[]

**Constructors:**

* Player(String name, Color color): constructor for the Player class

**Methods:**

* int RollDice(): rolls the dice and returns sum of two dice
* boolean giveSettlement(): takes Settlement from the player by decrementing the settlement number
* boolean giveCity(): takes city from the player by decrementing the number of the cities
* boolean giveRoad(): takes a road from the player by decrementing its number
* void getSettlement(Settlement settlement): adds setllement by incrementing the settlements number
* void takeCard(Card card): add card from player
* void giveCard(Card card): takes card from the player
* void giveCards(Card cards[]): takes cards from the player
* boolean takeCards(Card cards[]): adds cards to the player

### 3.2.4. GameBoard

### 

*Figure 23: GameBoard*

**Attributes:**

* int numOfCitiesOnBoard[]: stores number of the cities on the board
* int numOfSettlementsOnBoard[]: stores the number of the settlements on the board
* int numOfRoadsOnBoard[]: stores the number of the roads on the board
* Hex hexes[]: stores the hexes which are forming the board
* Path paths[]: stores the paths which are placed along hexes
* Corner corners[]: stores the corners of the hexes

**Methods:**

* boolean addCity(int playerNo,int cornerNo): adds city to the map
* boolean addSettlement(int playerNo, int cornerNo): adds settlement to the map
* boolean addRoad(int playerNo, int pathNo): adds road to the map
* replaceThief(int hexNo): moves the thief to the new place

**Constructors:**

* GameBoard(int numOfPlayers):

### 3.2.5. Card

### 

Figure 24: Card

**Attributes:**

* String type: shows the type of the card

### 3.2.6. DevelopmentCard

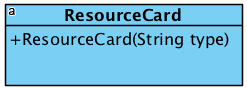


*Figure 24: DevelopmentCard*

**Constructors:**

* DevelopmentCard(String type): shows the type of the DevelopmentCard

### 3.2.7. ResourceCard

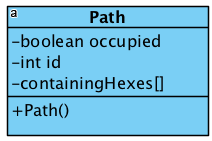
****

*Figure 26: ResourceCard*

**Constructors:**

* ResourceCard(String type): shows the type of the Resource card

### 3.2.8. Hex



*Figure 27: Road*

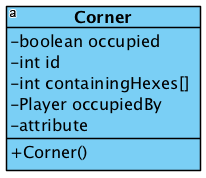
**Attributes:**

* boolean occupoied: shows whether it is occupied by some player or not
* int id: stores the id of the path
* containingHexes[]: shows the adjacent hexes

**Constructors:**

* Path():

3.2.9. Corner



*Figure 28: Corner*

**Attributes:**

* boolean occupied: shows whether it is occupied by some player or not
* int id: stores the id of the corner
* int containingHexes[]: stores the adjacent hexes
* Player occupiedBy: shows which player occupied that corner

**Constructors:**

* Corner(): constructor for the Corner class

### 3.2.9. Path

### 

*Figure 29: Path*

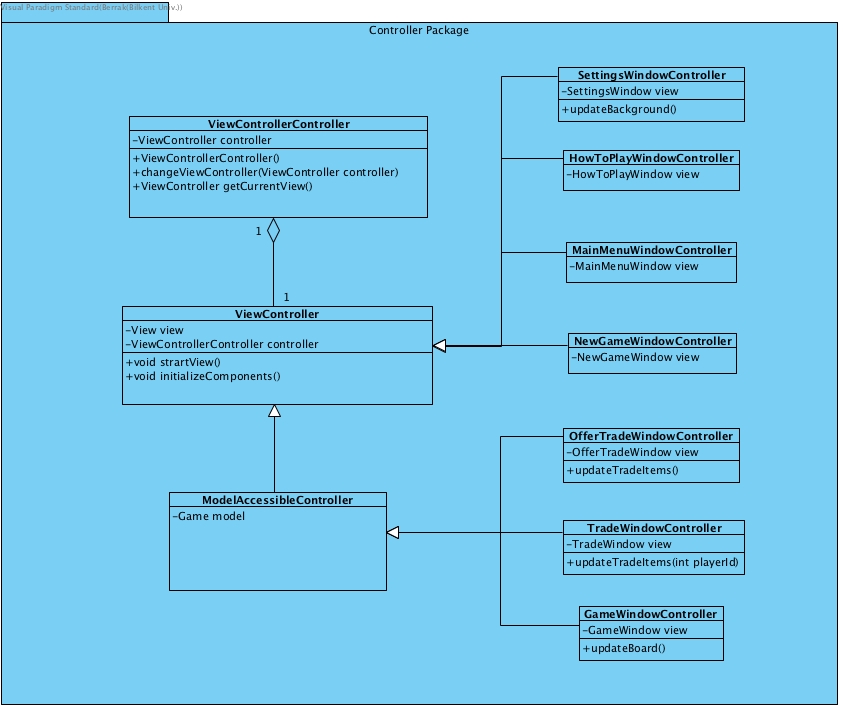
**Attributes:**

* boolean occupied: true if it is occupied by some player
* int id: stores the id of the player
* containingHexes[]: stores hexes which is adjacent to that path

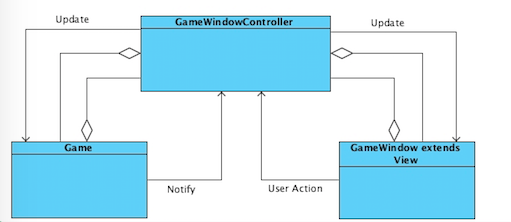
**Constructors:**

* Path(): constructor for the Path class

## 3.3. Controller Package



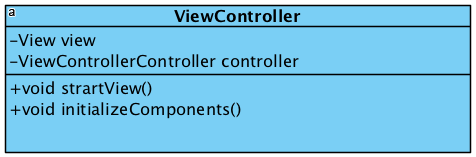
*Figure 30: Controller Package*



*Figure 31: MVC pattern with GameWindow*

### 3.3.1. ViewControllerController

### 3.3.2. ViewController



*Figure 32: ViewController*

**Attributes:**

* View view:
* ViewControllerController controller: controller for the ViewController

**Methods:**

* void startView(): starts view
* void intializeComponents(): initializes components of the ViewController

### 3.3.4. MainMenuWindowController

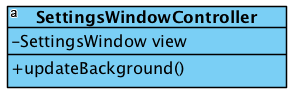
### 

*Figure 33: MainMenuWindowController*

**Attributes:**

* MainMenuWindow view: stores object of the MainMenuWindo class

### 3.3.5. SettingsWindowController



*Figure 34: SettingsWindowController*

**Attributes:**

* SettingsWindow view: stores object of the SettingsWindow

**Methods:**

* updateBackground(): updates background

### 3.3.6. HowToPlayWindowController

### 

*Figure 35: HowToPlayWindowController*

**Attributes:**

* HowToPlayWindow view: stores object of the HowToPlayWindow class

### 3.3.7. NewGameWindowController

### 

*Figure 36: NewGameWindowController*

**Attributes:**

* NewGameWindow view: stores the object of the NewGameWindow class

### 3.3.8. GameWindowController

### 

*Figure 37: GameWindowController*

**Attributes:**

* GameWindow view: stores the object of the GameWindow class

**Methods:**

* updateboard(): updates board after placing pieces on the board

### 3.3.9. OfferTradeWindowController

### 

*Figure 38: OfferTradeWindowController*

**Attributes:**

* OfferTradeWindow view: stores the object of the OfferTradeWindow class

**Methods:**

* updateTradeItems(): updates the window after every action

### 3.3.10. TradeWindowController

### 

*Figure 39: TradeWindowController*

**Attributes:**

* TradeWindow view: stores the object of the TradeWindow class

**Methods:**

* updateTradeItems(int playerId): updates trade window after every action

# 

# 

# 

# 

# 4. Low-level Design

Low-Level design of project will be described in this part.

## 4.1 Object Design Trade-Offs

It is not possible to perform some design goals at the same time. So, we have to choose which one of them to be prioritized over another. Our choices for design goals are listed below

*Functionality vs Usability*

In our game, we decided to prioritize usability over functionality. Since controls of our game is easy, usability is more important for our software. The game is controlled by mouse and this makes it easy to understand. “How To Play” screen increases our usability of our software.

*Efficiency vs Portability*

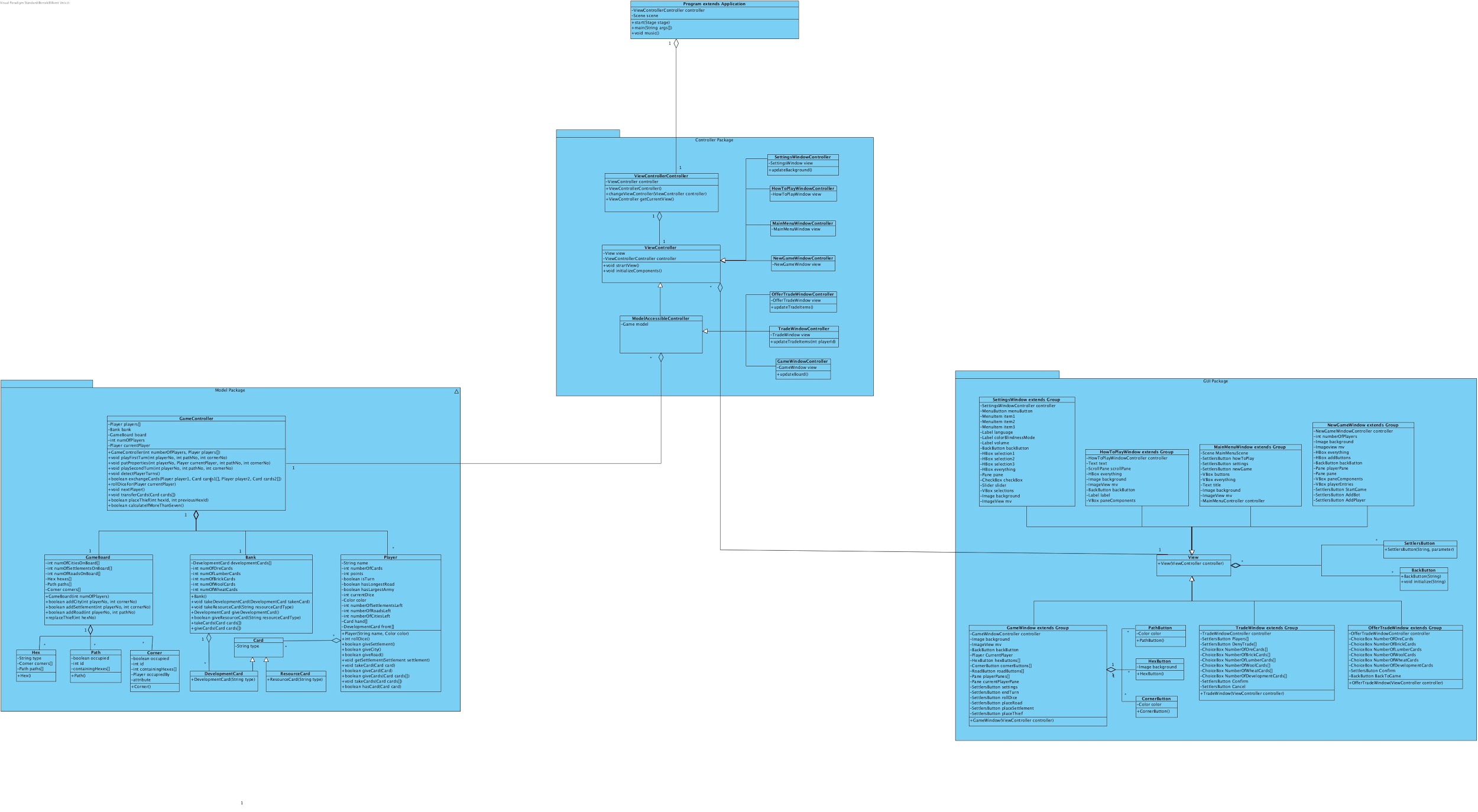
We chose portability over efficiency. Because we want to make our game enable to everyone we will design a platform independent software. Our software will be written in Java platform and it will be runnable on every operating systems and devices that supports JRE 8.

*Security vs Usability*

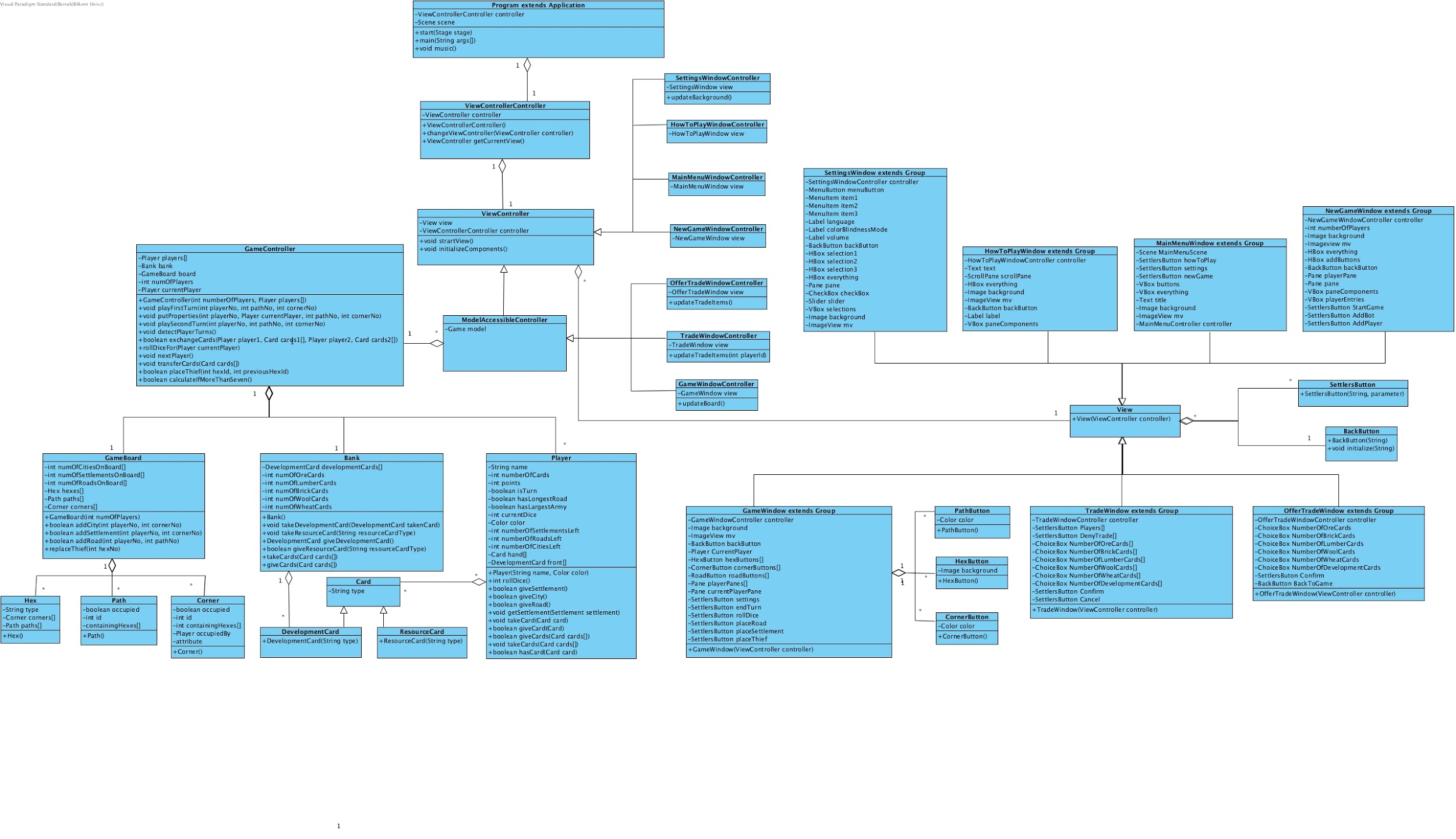
Simplicity of our game lets us not to be concerned about the security issues. We use just file system to store the needed data and we do not have to use database. Players do not need to login by using user/password system in our design of the game, because we prioritize usability and do not limit the user for security purposes.

## 4.2. Final Object Design

Final object design is provided below as a UML Class Diagram.



*Figure 40: Final Packages*



*Figure 41: Final Object Design*

## 4.3. Packages

We will be using Java to write our software. We will use Java’s Standard Library for the back-end part of our project and JavaFX to implement the GUI part. The packages that will be used are provided below.

Package javafx.\*: For the user interfaces, we will use java.fx.

Package java.io.\*: Provides for system input and output through data streams, serialization and the file system.

Package 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 Interface

Our software does not use any class interfaces.

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 5. Glossary & References

[1] <https://www.tutorialspoint.com/software_engineering/software_design_basics.htm>

Date accessed: 06.10.2019

[2] <http://tutorials.jenkov.com/javafx/index.html>

Date accessed: 06.10.2019

[3]<https://docs.oracle.com/javase/7/docs/api/index.html>

Date accessed: 07.10.2019