Bilkent University

Department of Computer Engineering

CS 319 - Object-Oriented Software Engineering

Design Report

Dirty Seven Card Game

Group 2-14

Ömer Mesud Toker

Nihad Azımlı

Metehan Kesekler

Özgür Taşoluk

Table of Contents

1 Introduction

1.1 Purpose of the System

Dirty Seven Card Game (Pis Yedili) is a well-known card game with different versions. However, in Google Play Store, there is not any likeable version of this game in terms of visually, usability and compensating different versions so it is decided that several version options will be included in this game with more user friendly features to reach more people and make them enjoy. Moreover, our application can be played in Turkish, English and Russian languages as well. The game will be an Android based application which can be played by all devices having Android 4.0 version or newer.

1.2 Design Goals

Before the composing the system it is crucial to identify the design goals of the system in order to clarify the qualities that our system should focus on. In this respect many of our design goals inherit from non-functional requirements of our system that are provided in analysis stage. Crucial design goals of our system are described below.

1.2.1 Performance

Response Time: For the games, it is vital that users' requests should be responded immediately in order not to distract the player's interest and entertainment. Our system will respond player's actions fast as the user wishes, while also displaying animations, effects smoothly for enthusiasm. Speed of game could be adjusted with configuration menu. There will 2 options for that: normal and fast. When human chooses fast option he will end up

game 1.5 times faster than he plays average of times. Moreover, both speeds can make the user enjoy from the game.

Memory: Memory usage is also an important factor for performance criteria and since the application uses simple graphics and saves only configurations data, it does not require huge amount of memory. Therefore, for the application it can be said that memory will not be problem.

1.2.2 Dependability

Robustness: In the application all game options and configurations are selected from a predetermined set of values. Moreover, moves of each players are according to the rules of the game which is known by the application. Therefore, those features prevent the user from choosing invalid input, so keep the system robust.

Reliability: Unexpected errors cause systems to become less reliable, but in the application, as the inputs given to the game are very expectable, these exceptional cases are really hard to occur. As the user chooses different options, the system will work smoothly. However in case some problems occur, our system can be shut down manually. Also data loss will not be a problem since the game does not have a save/load system except the configuration saving which is not a big issue.

1.2.3 Maintenance

Extensibility: In general, in the lifetime of game software, it is always important to add new components, features to the game in order to sustain the excitement and interest of the player. In this respect our design will be suitable to add new functionalities, entities

(i.e. new versions and rules for different users, new card styles, new features which enable the game played with more players and online etc.) easily to the existing system. It is believed the design we have currently provides this opportunity since it is tried to keep our classes separated and organized and that Android programming language (Java-based) is an Object Oriented architecture is very helpful for this aim.

Modifiability: In our system it would be easy to modify the existing functionalities of the system to make the game more functional or more playable. In order to achieve this we will minimize the coupling of the subsystems as much as possible, to avoid great impacts on system components by a desired change. Again, Object Oriented design helps for this aim as well.

Readability: Since this application is a group project, developers who work on this project have to comprehend each other's codes to implement nicely. Thus, source code must be neat and understandable and have necessary comments to improve the system.

Supportability and Adaptability: The application is to reach as many user as possible, so it is supportable by any Android devices which have Android version 4.0 or newer. If the user has a device which satisfying this condition, then s/he does not have to worry about the all features of the device.

1.2.4 End User

Usability:

Ease of Use: Since our system is a game, it should provide good entertainment for the player. In order to provide the entertainment player should not have a difficulty in using our

system. In this respect, system will provide player friendly interfaces for menus, by which player will easily find desired operations, navigate through menus and perform the desired operations. Also, it is determined that our system will perform actions according to tapping input from the user, like selecting the options or moving cards to in her/his hand. This makes it easy to use the system from the point of the player. Moreover, the game will have 3 language options which are Turkish, English and Russian to reach more people.

Ease of Learning: Since player is not ought to have knowledge about how the game is played, loss-win conditions of game, it is vital for the user to obtain information about the game concepts, for this purpose system will provide an instructive help document, by which he will be easily get warmed up to the game. The logic of the game is also very simple that user can easily understand intuitively or by reading the help document. This help menu will be always available for the user even when s/he is playing the game.

Cost: For now, it is planned that there will be no developing, deploying, upgrading, maintenance or administration costs associated with the application.

1.2.5 Trade-offs

Functionality vs. Usability: The application is intended to be user-friendly and easy to use for all groups of people. To achieve this usability aim, the application does not include all versions of the game played all over the world to not bother the user with complex and various menu options. It is limited to versions and rules which is played more generally. Moreover, to fulfil user-friendliness simple graphics and animations will be used.

Rapid Development vs. Dependability & Maintenance: Dependency and maintenance are key concepts. To keep the system robust and reliable, the application will

be minimized in terms of errors in the system which makes the code longer. Again, for extensibility, modifiability and readability goals, the system will be implemented cautiously. Therefore, to reach the dependability and maintenance goals, the application will be somewhat slower than rapid development.

Efficiency vs. Portability: Android programming applications helps the application to be portable, however; it has Android limitations. The application can be used only Android-based devices. Moreover, for the efficiency goal, the application can be used only devices which have Android version 4.0 or newer.

1.3 Overview

In this section, the purpose of the application is represented, which is basically entertaining the player as much as possible. To achieve this purposes, Design Goals part is defined. Desirable goals was performance, dependability, maintenance and usability. In this respect, it is an obligation to make some trade-offs to realize those goals. The system sacrificed from somehow functionality, rapid development property and portability to be more user-friendly, dependable, maintainable and efficient.

2 Subsystem Decomposition

We have following subsystems in our project:

1. **AppView subsystem:** This subsystem is game's user interface part. This part makes interface for human player.
2. **AppController subsystem:** This part of subsystem have very important role in design, since it makes major burden of code. Controller is responsible for controlling the

system. To make our design easier for both costumer and coder we divided our Controller into 3 parts:

a. **GameController subsystem:** This part is responsible of starting game and closing it. This controller is also managing things that happen during game related to gameplay
b. **SettingsController subsystem:** This part is responsible for things that happen in settings part of game. Opening settings and making changes belongs to this part of design
c. **ConfigController subsystem:** This part is for configuration part which is determine the gameplay type

3. **AppModels subsystem:** This subsystem gets value from Controller subsystem and displays it in value. As if it is making bridge role between Controller and View
   a. **GameModel subsystem:** This model subsystem is responsible for creating game objects initially when the game started. Its main job is forming the game model in the start of the game.
   b. **SettingsModel subsystem:** Settings model subsystem is responsible for retrieving settings from previous session and model settings when the game started and afterwards work responsibility changes to controller
   c. **ConfigModel subsystem:** This model subsystem is responsible for configurations of the game. Additionally, set them to the set configurations in the initial start of the game. When work of model done with configuration, additional work is done by controller subsystem.

2.1 AppView Subsystem

This subsystem is responsible for user interface of application. Human user interactions with application will be done by AppView subsystem.

Subsystem diagram of AppView subsystem is different than normal Java application, since we are using Android we will have android's ready user interface activity classes will be as MainActivity as main and GameActivity, HelpActivity, ConfigActivity, SettingsActivity, CreditsActivity.

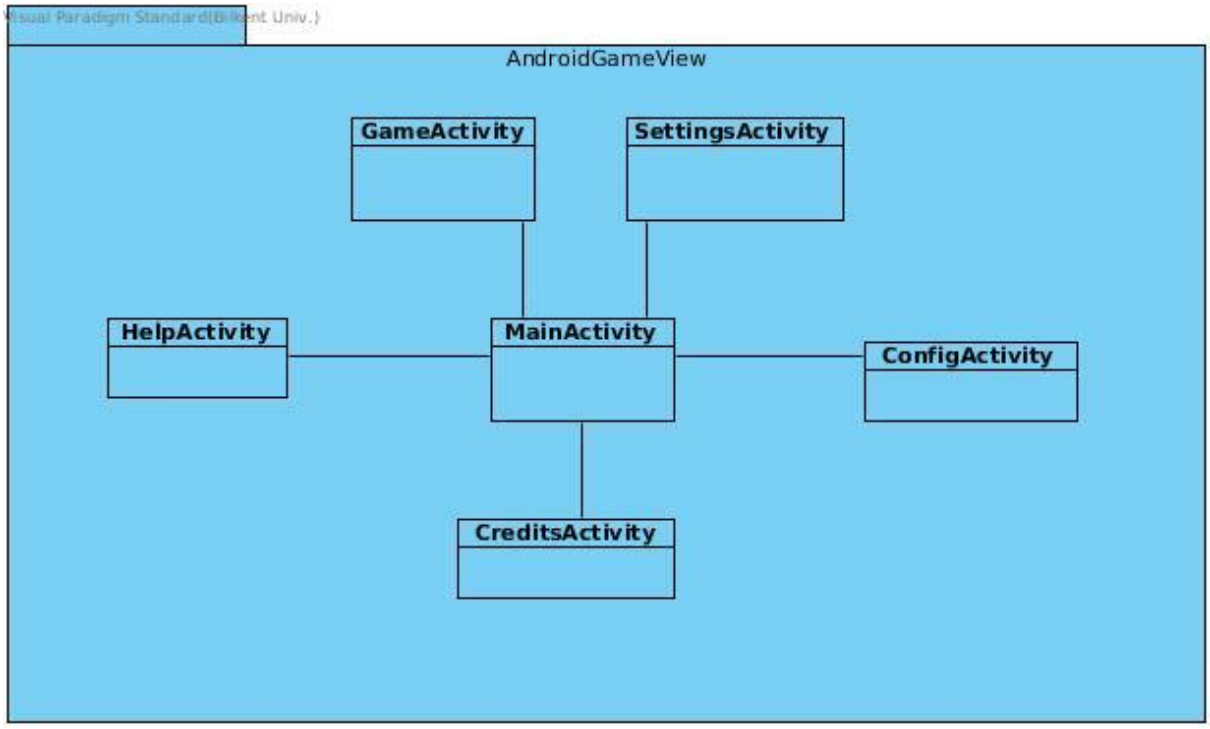

Figure 1: AndroidGameView

2.2 AppController Subsystem

This subsystem is responsible for handling the application. AppController subsystem consist of following parts.

2.2.1 GameController Subsystem

This subsystem will control during game and game characters. The subsystem diagram will be as above.

Character controller class will be head for players. HumanController and BotController will extend PlayerController class.

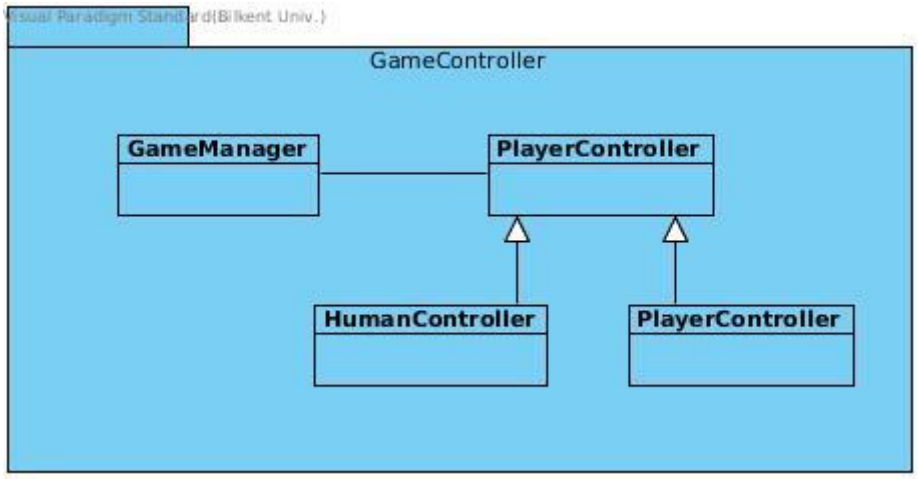

Figure 2: GameController

2.2.2 SettingsController Subsystem

This subsystem is relatively simpler than other subsystems. This part will control Settings part of game from the menu.

Figure 3: SettingsController

2.2.3 ConfigController Subsystem

Configuration part of the game will be controlled by this part. It is also simpler part of AppController.

Figure 4: ConfigController

2.3 AppModel Subsystem

2.3.1 GameModel Subsystem

This subsystem is manages logic and rules of application. This one is one of core AppModel subsystem.

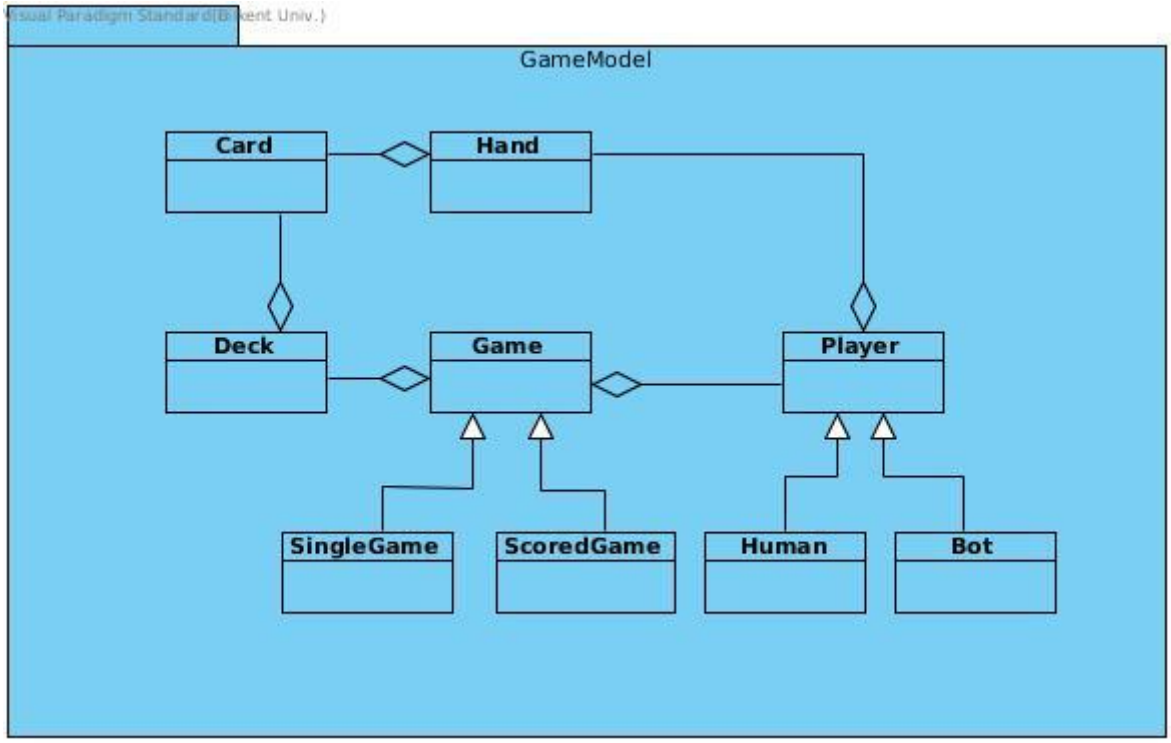

Figure 5: GameModel

2.3.2 SettingsModel Subsystem

Settings entity of application will be controlled by this model. This model subsystem only consists of a class.

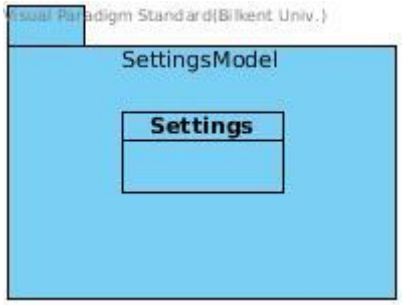

Figure 6: SettingsModel

2.3.3 ConfigModel Subsystem

This model will be responsible for configuration entity.

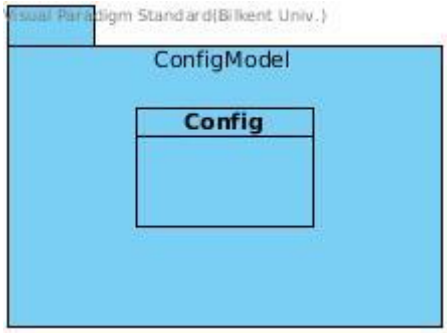

Figure 7: ConfigModel

2.4 Detailed Class Design

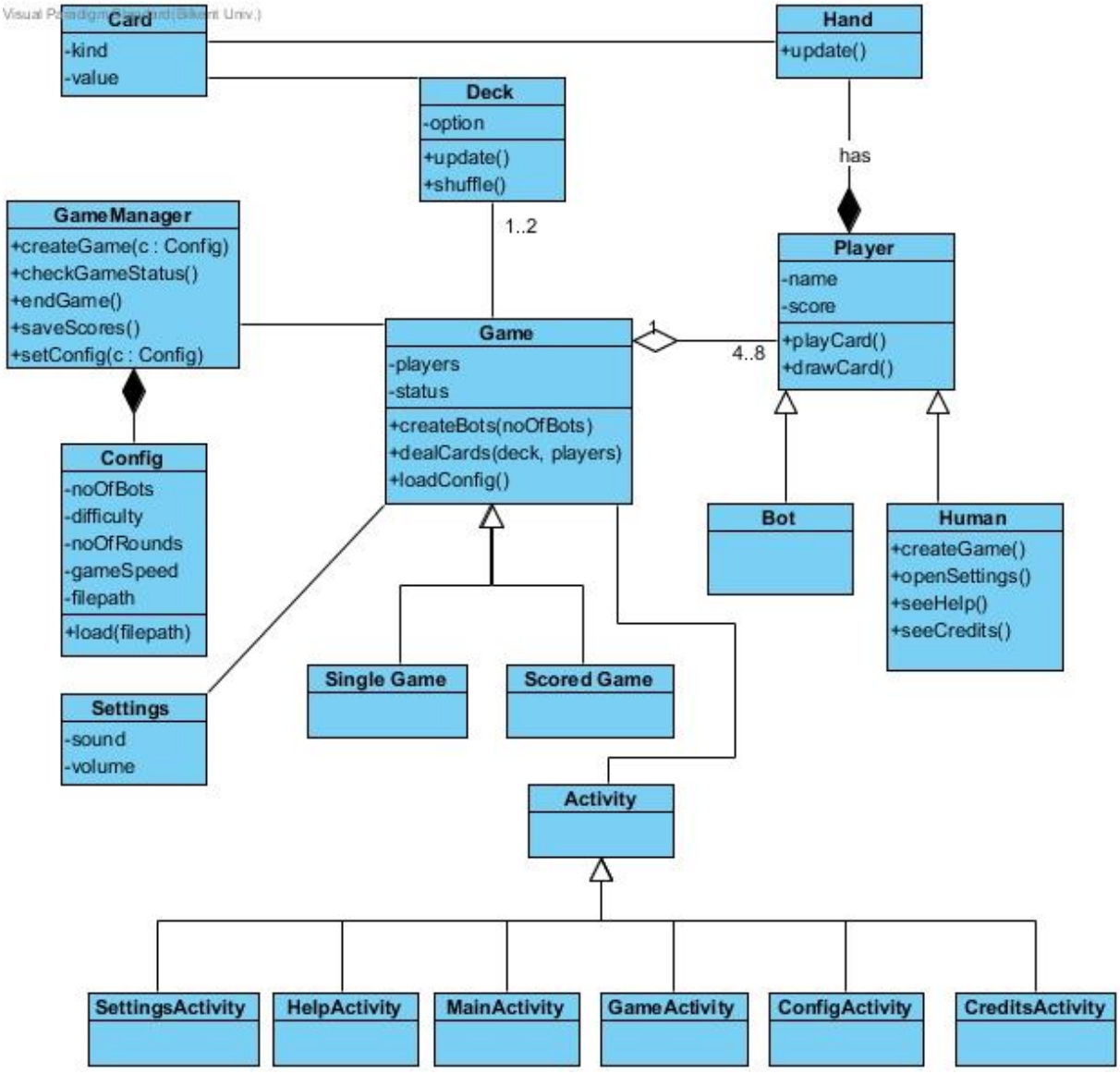

Figure 8: Game Logic Class Diagram

Player

Player is base of two classes which are Bot and Human class. However, play class also has its own properties and methods. They are described in the following lines:

Properties

Name: Player name property is one of the main properties for player such that every player has its own string type of name. It holds them until the end of game.

Score: Every player has its score end it is stored until the end of game. It is the factor to chose winner of the game

Methods

playCard(): This method will be called when player needs to place card to the table from his hand. It is one of the most common used methods during gameplay

drawCard(): This method will be called when any of the player needs to get card from deck on the table. This method will be invoked by any of the players.

Human

This class is designed to control humans during the game. This class has not got any property, however, it has several methods which are helping user to make the operations. This class inherits from Player class. These methods are stated as following:

createGame(): When human player creates a game this method is called. It creates new game and sets everything needed to start game state. This one is called by the human user

openSettings(): When human player opens the settings this method is called and load settings to the screen. This method also is invoked by the human user

seeHelp(): When human user wants to see help and click it this method makes help function to come to screen. This method can be invoked by human user

seeCredits(): When human user wants to see credits and click it this method makes credits to come to screen. This method can be invoked by human user

Bot

This class is designed to handle bots during the game. This class also inherits from Player class as Human class. However, this class has not got any property or methods inside.

Game

This class is responsible for the creation of the game according to the user's settings and configurations.

Properties

players: This array list holds the players and size of the list represnts the number of players in the game.

status: This boolean represents whether the game is finished or not.

Methods

loadConfig() : This method organizes the game according to user's configurations for the game.

createBots(noOfBots) : This method creates desired number of bots according to user's configuration for the game.

dealCards(deck, players) : This method distributes shuffled cards to the players of the game. Cards comes from deck.

SingleGame

This class is one of the sub-class of Game Class. Only difference between this class and its ancestor is loadConfig() method. noOfRounds will be 1 for sure for this game type.

ScoredGame

This class is one of the sub-class of Game Class. Only difference between this class and its ancestor is loadConfig() method. For this type of games, games are finished according to scores which is selected by the user as finishing score in the configuration.

Settings

This class is about the settings of the game which includes sound and volume.

Properties

sound : This attribute determines whether the sounds of the game is open or muted.

volume : This attribute determines the sound intensity.

Card

This class is responsible for the cards of the deck.

Properties

kind: This represents the kind of a card as Hearts, Diamonds, Clubs, Spades.

value: This represents the value of a card as A,K,Q,J,10,9,8,7,6,5,4,3,2.

Deck

This class is responsible for the deck of cards.

Properties

option: This is a boolean for number of decks(or cards) that the user wants to play with. (1 or 2, false or true respectively)

Methods

update(): This method updates the deck when there is a card drawn from it.

shuffle(): This method shuffles the deck at the beginning of each round and when there is no card left to draw.

Hand

This class is responsible for the hands of the players(Human and Bot).

Methods

update(): This method updates the hands of the players when there is a card played from it or drawn to it.

Game Manager

Game Manager is the main manager class of the Dirty Seven Card Game as it can be understood from the name of the class. It is connected with the Game entity and works together with it. This class is responsible for creating a game, checking the status of an ongoing game, ending a game and saving the scores of the players. It also has a Config object which holds game options.

This class has no properties, it is only responsible for main management of a game. Since it's connected with the Game and Config objects and those classes contain necessary properties, in GameManager class we didn't include redundant properties.

Methods

createGame(): This method is used for creating a new Game object with a given config.

checkGameStatus(): This function of the GameManager class checks the status of the Game every turn to see whether it changed and came to an end or otherwise. According to return value of this function game continues or ends by calling endGame() function.

endGame(): This method is called either when the Player wants to quit the game he/she plays or game finishes with regard to type of the Game.

saveScores(): This function is called when the Game ends normally, without the interruption of the Player. It saves the scores of the Player.

setConfig(): This method is called when the Player wants to change existing gameplay configuration. It takes a Config object and sets it as current configuration.

Config

This class is an entity class which is responsible for storing the information of gameplay relevant options. Since we keep saved gameplay configuration data in a file, this class uses those saved data.

Properties

noOfBots: This attribute keeps the value of how many computer controlled players will take place in the Game.

difficulty: This attribute is responsible for storing the hardness value of the Game.

noOfRounds: This attribute holds the value of how many rounds the game will have. This value is 1 for SingleGame game type and can be customized for ScoredGame game type.

gameSpeed: This property of the Config object contains the speed value of the Game.

filepath: This property is responsible for keeping the path of the config file which will be used when loading the data.

Methods

load(): This method is called when Config object is created. It reads the file which is located in the given path and loads the data to its properties.

MainActivity, GameActivity, HelpActivity, SettingsActivity, ConfigActivity, and CreditsActivity classes extend from Android SDK's Activity class. The classes are responsible for creating views for Main Menu screen, Game screen, Help screen, Settings screen, Set Config screen, Credits screen respectively. All those classes have necessary built-in Android SDK functions like onCreate, onDestroy, onResume. For interaction they include other built-in Android SDK methods like onClick for buttons etc. As properties, those classes have necessary view items like TextViews, EditTexts, Buttons etc.

3 Architectural Patterns

While designing the system of the Dirty Seven Card Game, as a group we decided to use the Model-View-Controller (MVC) architectural design pattern since it makes it easy to organize the system, to implement the code and it is a simple but strong pattern. We did not use other patterns and styles like Client-Server model, Repository pattern or Three-tier architecture because our system does not involve networking part, large amount of data, need for continuously updating-changing and complex processing the data or storing and accessing large persistent data. Dirty Seven Card Game application only keeps user's configuration settings as persistent data. In addition to this, since our application is a game it

will be an interactive application. Because of these reasons, choosing MVC design pattern would make system consistent and it is appropriate to use it.

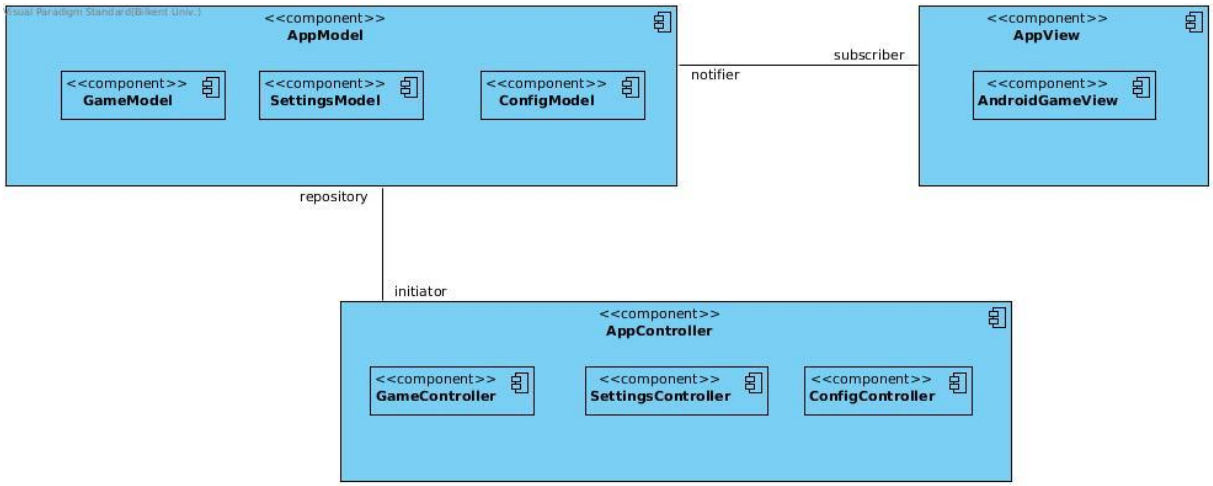

Figure 9

Dirty Seven Card Game does not consist of too many classes and it can be considered as simple, so MVC design pattern's effects on performance will not create problems for us and users. Model subsystems of our application are responsible for maintaining the domain knowledge, for instance properties of the active game, settings and configurations. Secondly, our View subsystem contains only Android Activity classes, which will be in charge of displaying the necessary views and getting input from user interactions. Last part of our design pattern is Controller part. It works just like it is defined in MVC definition. When it's necessary, it manipulates and/or updates the objects in Model subsystems according to user's interaction. We designed our Model subsystems to be independent of our Controller subsystems and View subsystem. At the very moment when an entity from Model subsystem gets updated by one of the Controller subsystems, the changes will be displayed by View subsystem with help of the relationship between View and Model classes.

4 Hardware-Software Mapping

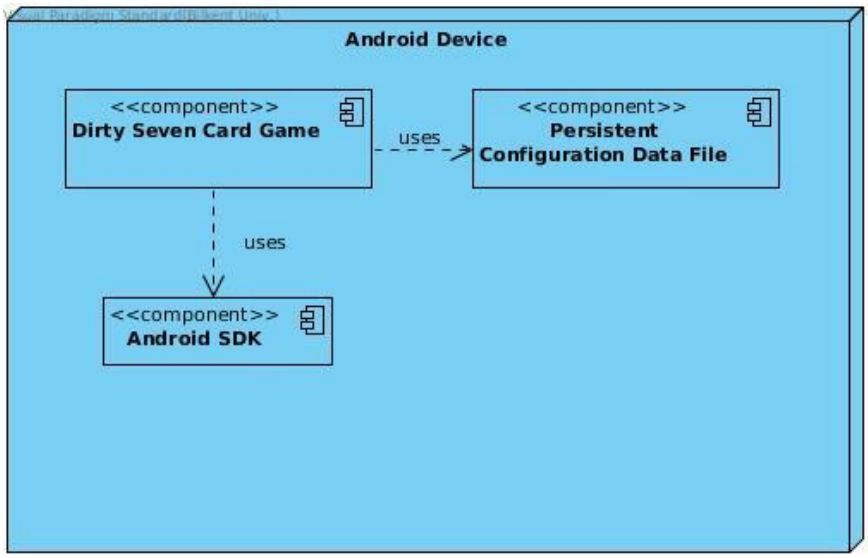

Figure 10

Our game will be Android based mobile application. So we will use Java Development Kit 1.8 which is up to date one. We will have support for phones with Android version of 5 or newer version. Application's hardware interaction will be with touch screen of mobile phones as it can be understood we just need mobile phone to operate through application. Everything that user done will be got by his touches to screen as configuration, settings opening different functionalities. Rather than traditional Java, Android has different design pattern. Android uses only Android SDK which consist of tools for developing Android Applications. Our application will not have connection with database. Also, only one player option will be possible for our Bad Sevens, there will be no multiplayer option with or without network. Bad Seven's graphic user interface will be done with Android Activity libraries. The reason behind using Android as development environment is because of as Android based on Java, so familiarity with java, also, design of GUI and overall application will be good. Additionally,

the highest number of overall usage of Android operating system makes also attractive to us for using Android. Finally result of our application will be consist of an executable file.

5 Addressing Key Concepts

5.1 Persistent Data Management

In Dirty Seven Card Game we store in game and not in game data. Not in game data means the data that will also be stored when the game is not running. These are the configuration data which the user can set his choices and they are stored in the disk and they will be accessed by the game when it's necessary. In game data means the data that will not be stored when the game is not running. In in game data there is data which will be changed in a round and there is data of the score table which will be changed after a round. Any change in not in game data which effects the gameplay will cause reset and create a new game so in game data will also be resetted. And there is data which cannot be changed by the user.

5.2 Access Control and Security

Dirty Seven Card Game is a single player offline game so there will not be a user authentication system. The persistent data that is inaccessible by the users will only be accessible by the program's controller classes and they will be read-only files. The game will not request access to any data that is not related with itself. Since this is a simple offline game that does not require user authentication, security issues will not be a problem.

5.3 Global Software Control

Dirty Seven Card Game uses event-driven control as its control flow mechanism. Until some external event or interruption occurs, the application will wait for such events. At the time an event occurs, for instance user presses a button on the screen, corresponding controller class is activated and that controller class manipulates the appropriate model. After controller class changing model, model object will notify necessary views and displays the desired view.

5.4 Boundary Conditions

Initialization:

Initialization of Dirty Seven Card Game will only require user to click the icon of the game from his/her Android device's menu.

Termination:

Dirty Seven Card Game can be terminated by pressing the X button which is located on the top right corner of the screen from the Main Menu. In a scenario where Player wants to quit the game from the Game screen, he/she will be returned to main menu after clicking X button and ending the ongoing game. After that it is the same process.

Error:

If persistent data file which we keep gameplay related configuration information is corrupted or distorted, game deletes that file and creates a new file to take the place of the previous one with default configuration values and starts the game with that configuration.

If game stops responding because of a device related issue, Player loses all current game related data.

6 Conclusion

To sum up all that mentioned above, design report made one step closer us to final stage. Because of concrete report we found concrete solutions to our problems. We made up our mind on making. For example, while making subsystem decomposition part we concluded our classes and design, also, it made things clearer. Furthermore, we had idea concrete idea about scope of project.

In design goals part, we get idea of some quality measures such as performance, durability, also, end user part.

In Subsystem Decomposition, we separated parts of system to subsystems. In this part of project class diagrams, deployment diagrams and so on made to give better understanding of diagram concepts. We divided our class diagrams into subsystems, and apparently MVC pattern looked good to us. Additionally, MVC pattern researching made and we found that the best architectural pattern that fits to our mobile application is MVC pattern so we decided to use it and made our subsystem decomposition part according to that. Furthermore, we used boundary case part to find exceptional point of project and it made more close to our project.

To conclude, we made good progress after making design report such that we already can imagine the project in our mind, Also, knowing subsystems and determining architectural pattern made our implementation part of project easier than it was before.