CS 319 - Object-Oriented Software Engineering

System Design Report

Jungle Pursuit

Fatma Begüm İlhan

Syed Sarjeel Yusuf

Sinan Öndül

Contents

Introduction **4**

Purpose of the system 4

Design Goals 4

End User Criteria: 4

Maintenance Criteria: 5

Performance Criteria: 6

Trade Offs: 6

Ease Of Use and Ease of Learning vs. Functionality: 6

Performance vs. Memory: 6

Definitions, acronyms, and abbreviations 8

Abbreviations: 8

References 8

Overview 8

Software Architecture **9**

Overview 9

Subsystem Decomposition 9

Architectural Styles 13

Layers 13

Model View Controller 14

Hardware / Software Mapping 14

Persistent Data Management 16

Access Control and Security 16

Boundary Conditions 17

Subsystem Services **18**

Design Patterns 18

Façade Design Pattern 18

Material Design 18

User Interface Subsystem 20

Main_Menu 20

Credits 21

Help 22

Game Manager Interface 23

Player Panel: 24

Introduction

Purpose of the system

Jungle Pursuit will be a mobile application that is based on Android so that user can control the game with the help of some features. That game also will have various images, animations and accordingly objects. One of the important features is that the only requirement for user to interact with the game is touching dice button to roll the dice. Since this game will be compatible with smartphones, in order to roll the dice, user can shake his/her own smartphones that is powered by the use of Android libraries.

Design Goals

Clarifying the design goals of system and so the game is the most important. We also should focus on what kind of qualities or features our system have. Accordingly, almost all design goals benefit from non-functional requirements. The most important part of the system which is design goals is clarified and explaining below.

End User Criteria:

Ease of Use: Since we are creating a game, our first aim is that while users are playing the game, they can have the entertainment. Even if user does not now how to play, she/he can play the game with the help of friendly and helpful interfaces for menus. This is because the game is mostly aimed at the younger demographics and hence ease of use is necessary to be considered. For the game, not only function but selection about the features of game also can be made by users. In addition to these explanations, our game will be touching dice button for simulating a computer by way of mouse and for tablets with the help of touching a point that points dices. They all are so easy that user can play the game.

Ease of Learning: As it is mentioned in purpose of system and ease of use, user does not have to have any knowledge about the game before playing. As getting information regarding the game before starting to play is substantial, he/she easily can get information

with the help of some information in “Help Window”. Therefore, the structure of game is so simple that user reads before playing the game and understand in an instant.

Maintenance Criteria:

Extendibility: Actually, for real life of software, there is always the thing that needs to be develop more and more to have sustainability. For that game, we are planning to add some new components and objects in order to increase the excitement and request of users to prefer and play that game.

Portability: As it is mentioned before, our system will be based on Android as a board game and so players can have that game with the help of their smartphones and tablets whenever they want to play. In other words, this feature makes our game portable. Since portability is crucial element for software to reach extensive area and users, we have decided to implement in Android.

Reliability: The system will be consistent in terms of conditions. Our game will be created using Android techniques and also we will have object-oriented system that is compatible with our android game so it has own features that players will become satisfied with its reliability.

Compatibility: As the game is being developed for android systems, it must be kept in mind that Google, who owns Android, releases several updates on a regular basis. This is manifest in the several Android APIs available in the market. Hence a major goal of the system will be to ensure that a wide range of APIs are supported by the game. This involves using several Android libraries that may be been since then deprecated and are no longer in use but are still essential for the fluent running of the system on older devices. This is because the game should be compatible with the wide range of Android devices in the market, owing to Android’s great popularity.

Performance Criteria:

Response Time: It is the fact that users should wait quick response while they are playing the game. Our system will also provide quick response to users' actions in order not to reduce user's attention to game. When the user click the button to roll the dices, dices will start rolling in an instant and user can also see the animation and images of how dices roll.

Trade Offs:

Ease Of Use and Ease of Learning vs. Functionality:

For our system, we think that user can play that game in order to have fun and learn that game with ease. According to that, our design and implementation will be so simple that usability and learning will have higher priority than functionality. In other words, the system will have simple interface like "Main Menu". For example, in Main Menu, users can make a selection in a simple way.

Performance vs. Memory:

For our system, our main aim is to make easy, useful and funny game with animations and some effects. We mostly focused on how to obtain high performance from the game. It must be kept in mind that the game is being developed for a wide range of mobile devices. Hence mobile devices on the lower end of the spectrum have limited resources as compared to mobile devices on the higher end of the price range. This limits the ability of the design regarding animations. For Example, the use of a gaming engine would prove to heavy for lower end mobile phones and may cause the system to result in an error that is 'out-of-memory-exception' error. The targeted device which will give us a good perception of low end devices is the Samsung S Duos which has RAM of just 768 MB[3]. We shall also test the application on a Samsung Galaxy Tab A which has a RAM of about 1.5G which can be seen is much larger than the S Duos. Hence the Galaxy Tab A can support advanced animations and allow the incorporation of heavy gaming engines. However, it is noted that the S Duos will experience great lag in trying to run the application and hence justifying our decision of not to use a gaming engine.

Definitions, acronyms, and abbreviations

Abbreviations:

MVC: [2] Model View Controller

SDK: [1] Software Development Kit

References

[1] https://developer.android.com/studio/index.html

[2] Object-Oriented Software Engineering, Using UML, Patterns, and Java, 3rd Edition*, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2010, ISBN-10: 0136066836.*

[3]"Managing your App's memory,". [Online]. https://developer.android.com/training/articles/memory.html. [Accessed: Oct. 1, 2016].

Overview

In this section, we explained purpose of the system and some features about game like how to play and ease of playing the game. With the help of those, players can have fun and play with ease. Also, we presented our design goals that are portability, ease of use, ease of learning, quick response. Additionally, we made some trade-offs to notice our aims in order to make the game more simpler and understandable.

Software Architecture

Overview

The section shall highlight Jungle Pursuit as maintainable subsystems. In dividing Jungle Pursuit into subsystems our aim is to reduce the coupling of different subsystems and also increase cohesion of the different subsystem components. Furthermore, it should be noted, that there is no need to define a MVC(Model View controller) architectural style on our system. This is because Android already handles the basic MVC structure where:

1. View = layout, resources and built-in classes like Button derived from android.view.View.
2. Controller = Activity
3. Model = the classes that implement the application logic

Subsystem Decomposition

In this section, the system is divided into relatively independent parts to clarify how it is organized. Since the decisions we made in identifying subsystems will affect significant features of our software system like performance, modifiability and extendibility; decomposition of relatively independent part is crucial in terms of meeting non-functional requirements and creating a high quality mobile software.

In Figure-1 system is separated into three subsystems which are focusing on different cases of software system. Names of these subsystems are User Interface, Game Management and Game Entities. This is to emulate the MVC model in android. They are working on completely different cases and they are connected each other in a way considering any change in future. For now it can be seen that the User Interface package involves a Main Menu which is connected to a screen manager. The Screen Manager can be thought of the Views of the layout and the imported using the 'setContentView'. Furthermore, the Game Manager package consists of an Input Manager that takes input

from the user and is connected to the Player Manager and Board Manager. These are responsible for controlling the appearance of the player and the board respectively. Finally in the Entity Manager, it can be seen how the entities are related.

As seen in Figure-2 Game Management and Game Entities subsystems are loosely coupled. Besides, only connection between User Interface subsystem and Game Management subsystem is also provided over Game Manager class. Therefore any change or error in User Interface subsystem will only affect Game Manager class which has a role as interface of control unit.

Furthermore, from Figure-2 we also see that the Player entity is controlled by the Player Manager while the Board is controlled by the Board Manager. The reason behind separation of the two manager classes was to create flexibility and have separate file managements for storing the information that would essentially go into manipulating the Player entity's appearance and the Board entity’s appearance. Another issue that had to be considered was that, the board entity would be updated continually throughout the game where as the player appearance is updated only once at the start of the game.

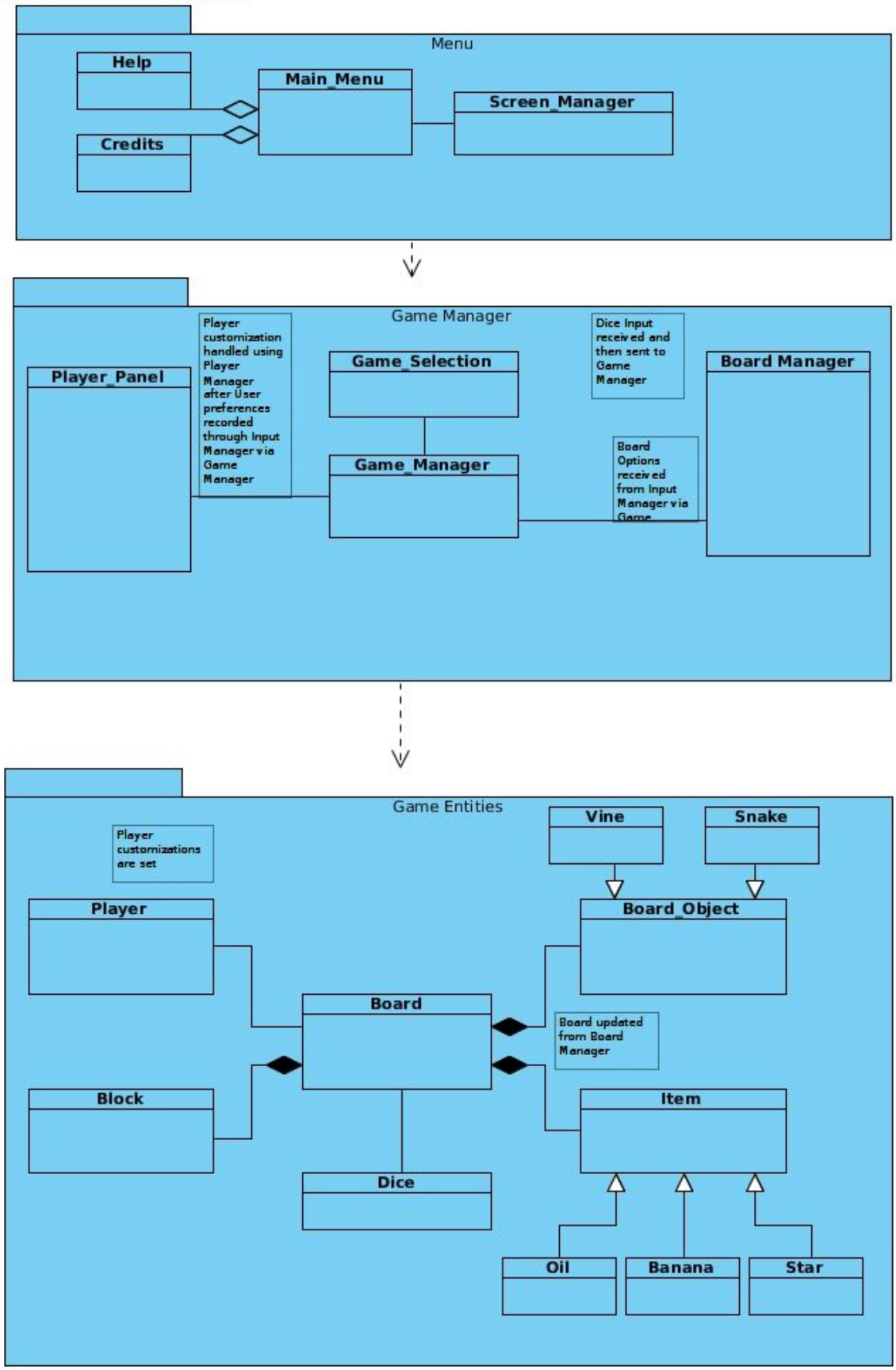

Figure – 1 (Basic Subsystem Decomposition)

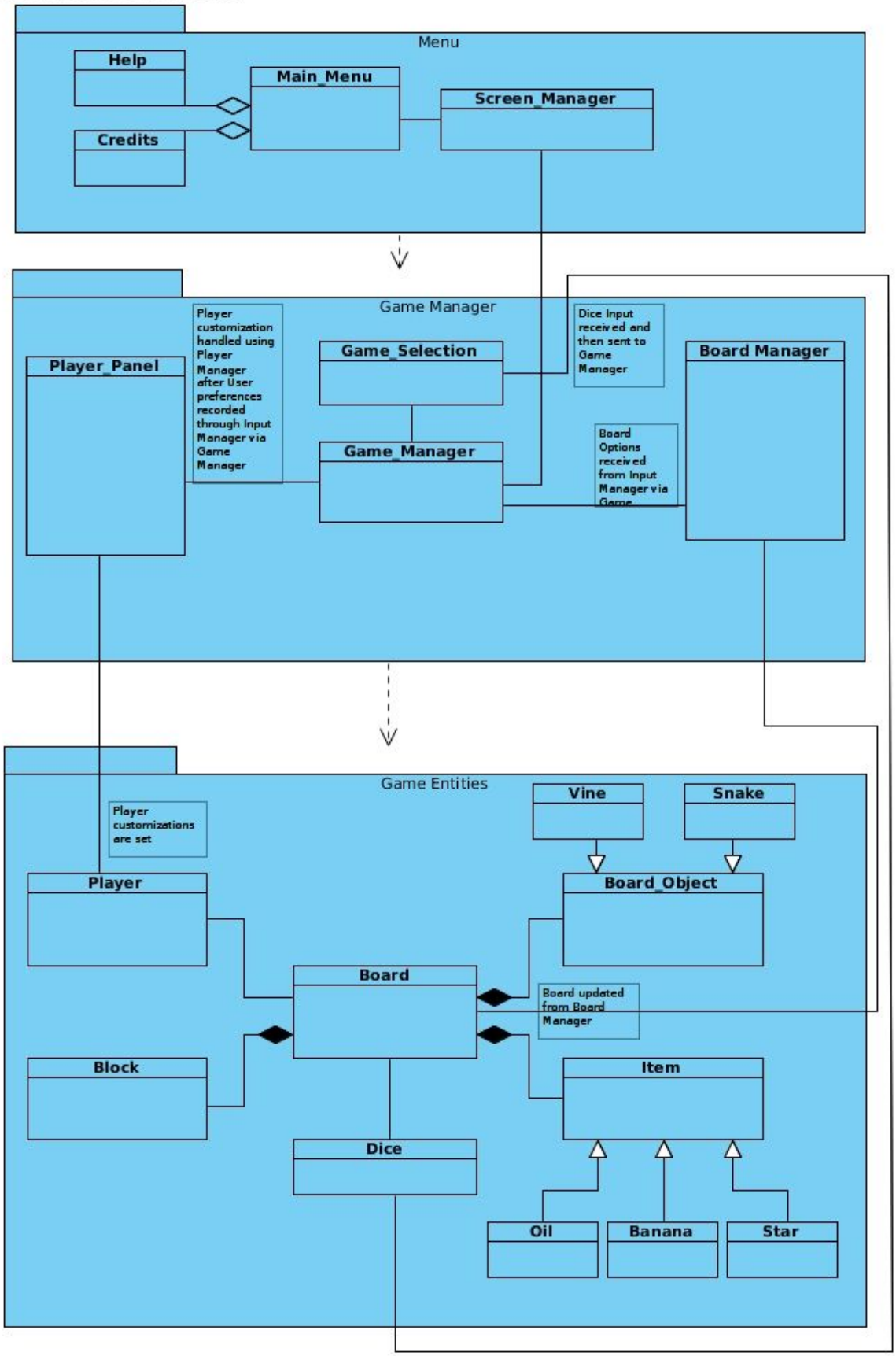

Figure-2(Detailed Subsystem Decomposition)

Architectural Styles

Layers

The system can be split into three layers with the User Interface being at the top and the Game Entities being at the bottom. To manipulate the Game Entities the user has to go through the Game Management layer and hence will need to use the User interface to do so. The User Interface will be directly visible to the user and in Android systems can be thought of as the Activity views that are created using the Layout XML files. Similarly the Game Management layer will involve the manipulation of variables within the Android Activities that the user is currently on and this will be done through the methods within the Activities. Finally the Activities will instantiate the class objects that represents the Game Entities. These Game Entities will be instantiated according to the inputs of the player before starting the game. For example, game will be initiated according to whether the player stated multiplayer or single player. Moreover, the game panel that will be shown in the user interface to display player information will also show the unique player name and the board shall display the player characters as chosen by the player. Hence it can be seen how the three layers will intercommunicate with one another.

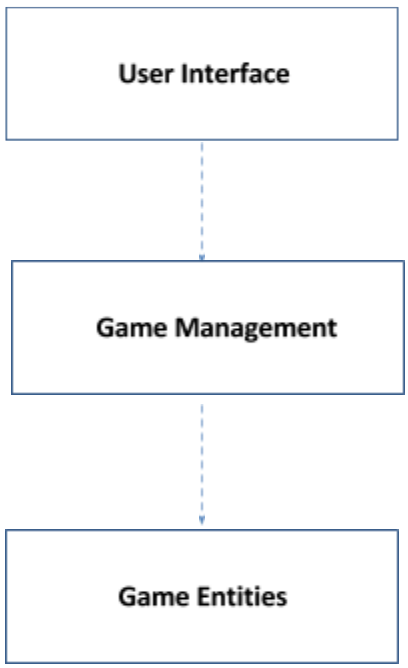

Figure-3(Layers of system)

Model View Controller

Even-though, as mentioned earlier, Android already implements the MVC architecture, Jungle Pursuit will be programmed by visualising the MVC architecture. Hence the splitting of the system into three major layers as seen in Figure-3. This is because by splitting the system into different layers we have managed to separate and isolate the domain knowledge from the user interface by adding a controller part between the Game Entities and User Interface. This has thus allowed the grouping of the domain game objects into the Entities which can only be accessed via the Game Management layer. This thus reduces the need for any gaming engine which would otherwise reduce the complexity of the system to a level that would be deemed inadequate to fit the requirements of the course. Hence by following an MVC style, the Jungle Pursuit will not need a game engine to do tasks such as update the board.

Hardware / Software Mapping

Jungle Pursuit will be implemented natively in Android, and thus be implemented in Java , therefore there will be a need for the JDK (8). Moreover, as one of the greater goals of the system is portability and extensibility, the Android API used is essential. As there is a large variety of phones out in the market, it is imperative that the system can be supported by many devices as possible to increase the user base. However, facilitating several old Android versions is problematic as programing a system that can do so would result in using several Android libraries that have since been deprecated. This can cause several User Interface problems as an essential library that may be needed by an older version API may not be suitable for complex tasks that are targeted at higher end APIs. Therefore, it is intended to set the minimum API of the system to 15 and the maximum API of the system to 21.

Another aspect to consider is the screen sizes of the different android devices that can range from mobile phones to large-screen tablets. A typical phone has a dp of 320. A tweener tablet such as the Streak has a dp of 480 and an average tab of 7” has a dp of 600 .

Hence with all these varying screen sizes, it has to be ensured that all the layouts are compatible with all the users. One of the first things to consider is font. Therefore sp shall be used instead of dp. Since sp is like dp but also 17 also scaled by the user's font size preference. Hence it is preferred to use this unit when specifying font sizes, so they will be adjusted for both the screen density and user's preference.

As hardware configuration is concerned, the system only needs an Android device that can support the stated APIs used. For input, the User is going to use the touch screen keyboard or an inbuilt keyboard to input Player Name. This is going to be the most complex input that the user will have to do. Other than that, he/she will only have to select options before starting the game, such as choosing the game type and character. Finally the player will only have to press a button to roll the dice and can also shake his phone to replicate how one would otherwise roll a dice in the physical world. However, for the user to be able to use this feature, his/her phone must have the correct sensors to allow the detection of the shake. Now days, most phones that fall within the stated API range do have an accelerometer which is responsible for detecting phone movement.

For storage issues, the game shall use files from the Assets manager of the Android system and also the internal storage of the device. The Assets Manager will be used to store data in .txt files that cannot be manipulated once created. It will store essential information about the game such as the instructions of how to play the game and basic block information. Internal storage will also be used, and hence the system will need to have certain PERMISSIONs to allow the game to create and delete files in the internal storage of the user’s device. These Android permissions are seen below:

```
<uses-permission android:name="android.permission.WRITE_INTERNAL_STORAGE" />
```

The permissions shall be included in the application's Manifest file which is responsible for describing the android resources that the application can utilise. It is in the application's root directory and provides essential information about the app to the Android system, which the system must have before it can run any of the app's code.

It it thus necessary that the user has enough storage capacity on his phone. However, the system will only be storing small .txt files that will mostly have information regarding the players and thus very minute amounts of memory will actually be needed.

Persistent Data Management

Since The game will only store basic information, mostly regarding the current game, a complex database system will not be required. Instead all information management can be conducted using .txt files in the internal storage of the device and the Assets manager of the Android application. Moreover, images will be stored, such as the ones that will be used to represent the players on the board. These images will have to be .jpg as it is the most suitable image type for Android applications. Moreover, there may be a need to store different sizes depending on the screen density. This can be seen below:

- xlarge (xhdpi): 640x960.
- large (hdpi): 480x800.
- medium (mdpi): 320x480.
- small (ldpi): 240x320.

Access Control and Security

Jungle Pursuit is a basic android game that will not need any networking functionalities. Hence, in terms of security, there is not much required to implement for the game to be successfully played. The most security that the use can have is ensuring that the device itself is secure. Moreover, the system will be easily downloadable from the Google App store which itself is a layer of security. Hence no special security functionalities in the form of passwords or access permissions will be necessary. This is also because the android APK will restrict the user from viewing any internal files such as images and those files in the Assets Manager.

Boundary Conditions

Initialization

jungle Pursuit will be downloadable, as intended, from Google's App store. This will involve the user finding the game on the App store by name and simply clicking the download option. The Game can also be directly downloaded by transferring the APK to the mobile and accessing it. This will however, need the user to change certain application security settings.

Termination

The game could be terminated by the user clicking on the home screen of his android device. However, it must be noted that this doe not exactly terminate the game but simply suspends it, allowing the user to go back to it. Thus the user will have to close the game by opening a list of the currently running apps and specifically closing Jungle Pursuit. Any Android application can be closed in such a manner. Nevertheless this is not the intended manner in which the game will be terminated. The user can go back to the main menu and select the quit option that will be available. This will terminate the app, and not keep it suspended.

Error

An error can occur due to lack of memory. It must be noted that several images such as those for the board objects and items are being used. Hence this could lead to a lack of phone memory that could occur in older device models. hence memory management will be a concern in the implementation of the system.

Subsystem Services

In this section we will provide the detailed information about the interfaces of our subsystems.

Design Patterns

Façade Design Pattern

Façade design pattern is a structural design pattern which proposes that developers can easily manage a subsystem from a façade class since the communication between outside of this subsystem is performed only by this class. This pattern, provides maintainability, reusability and extendibility since any change in the components of this subsystem can be reflected by making changes in the façade class.

In our design we used façade pattern in two subsystems: Game Management and Game Entities. In Game Management subsystem our façade class is GameManager which communicates with the components of the Game Management subsystem according to the requests of User Interface subsystem. For our Game Entities subsystem our façade class is Board class which handles the operations on entity objects of our system, according to the needs of Game Management subsystem.

Material Design

Furthermore, the user interface is designed using material design which is is a design language developed in 2014 by Google. Expanding upon the "card" motifs that debuted in Google Now, Material Design makes more liberal use of grid-based layouts, responsive animations and transitions, padding, and depth effects such as lighting and shadows. For Jungle pursuit, the color scheme of material design shall use shades of green which is to follow the jungle look that is maintained throughout the game.

Also it must be noted that Android performs necessary subsystem features for the user interface as it divides the layout structures in XML code and the functional classes in

Java, if the program is written natively in the Android SDK. However, for the ease of visualisation, figure 5 illustrates the subsystem structure of the user interface of Jungle Pursuit with screen manager which is actually managed by the XML layout files.

Green	
500	#4CAF50
50	#E8F5E9
100	#C8E6C9
200	#A5D6A7
300	#81C784
400	#66BB6A
500	#4CAF50
600	#43A047
700	#388E3C
800	#2E7D32
900	#1B5E20
A100	#B9F6CA
A200	#69F0AE
A400	#00E676
A700	#00C853

Light Green	
500	#8BC34A
50	#F1F8E9
100	#DCEDC8
200	#C5E1A5
300	#AED581
400	#9CCC65
500	#8BC34A
600	#7CB342
700	#689F38
800	#558B2F
900	#33691E
A100	#CCFF90
A200	#B2FF59
A400	#76FF03
A700	#64DD17

Figure-4 (Color Scheme)

User Interface Subsystem

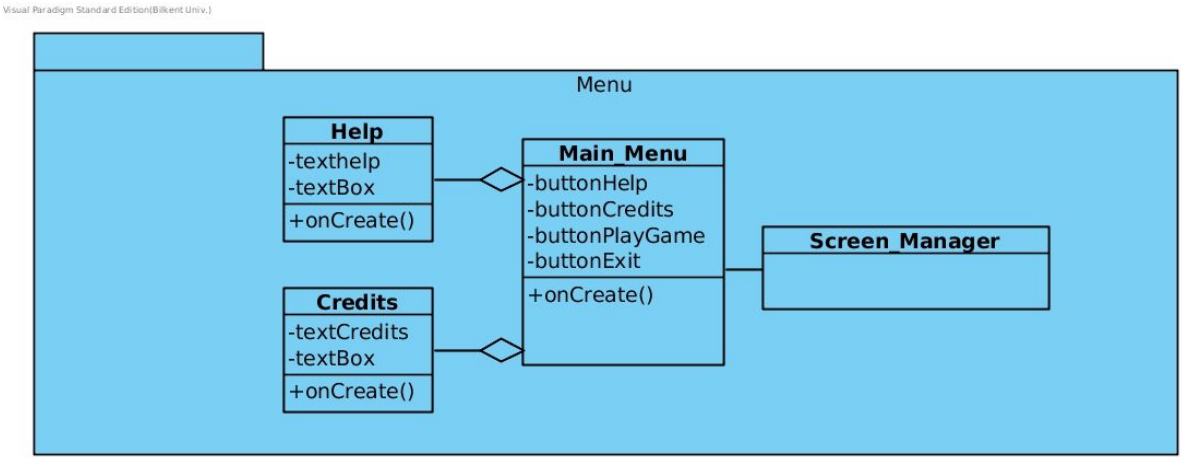

Figure-5 (Color Scheme)

From figure 5 it can be seen that screen manager does not have any attributes or methods because it is technically handled by the Android system and does not have to be defined by the developer. This is one advantage that the Android SDK provides and eases development of mobile applications. The screen is thus updated and the user interface is displayed by adding layouts to the view as will be seen in a method possessed by the GameManager class in the GameManager entity.

Main_Menu

This is the equivalent of the main function that will be called by the operating system. It is the main activity that the user will see first. It simply provides navigational options that allows the user to start a game, go to the credits menu, or the help menu to learn how to play the game. The user can also exit from the main activity which will kill the application running in the background. Moreover, because the activity will be displayed by the android system, it has its own layout file which shall be loaded upon its initialisation.

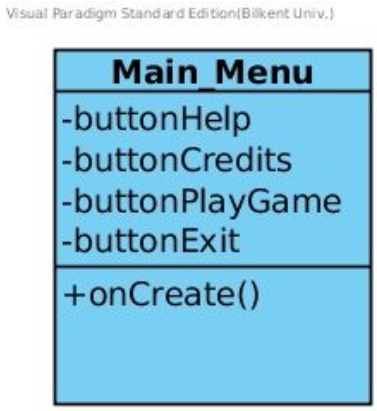

Figure 5.1 (Main Menu in Subsystem 1)

Attributes:

i. buttonExit ~ An Android button object that will be used to exit the program
ii. buttonPlayGame ~ An Android button object that will be used to start a new game
iii. buttonCredits ~ An Android button object that will be used to navigate to the Credits Activity
iv. buttonHelp ~ An Android button object that will be used to navigate to the Help Activity

Constructors:

i. onCreate() ~ This is the default constructor for the Activity as it will be displayed to the user and is partially handled by the Screen Manager. As the activity is loaded, the screen manager component of the Android System manages the layout of the screen by adding a Layout file from the Resources folder of the Android application system. Here the objects on the screen are also initialised. No other operations take place.

Credits

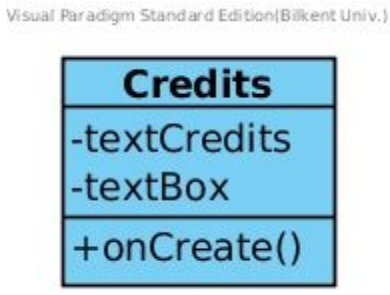

Figure 5.2 (Credits in Subsystem 1)

Attributes:

i. textCredits ~ Is a string attribute that will get its value from the Strings folder of the Android System where the information to be displayed will be stored and called when the activity is initialized. It will specifically get the Credits info string value.

ii. Textbox ~ This is an Android TextBox container object that will be used to output the textCredits that is obtained.

Constructors:

i. onCreate ~ Default constructor for the Activity and will allow the screen manager to load the layout on to the screen and then initialise the Android TextBox to the object on the screen according to the layout loaded. It will then also set the text of the TextBox to the textCredits.

Help

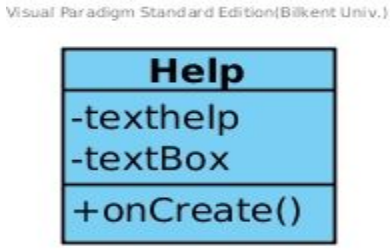

Figure 5.2 (Help in Subsystem 1)

Attributes:

i. textHelp ~ Is a string attribute that will get its value from the Strings folder of the Android System where the information to be displayed will be stored and called when the activity is initialized. It will specifically get the Help info string value.

ii. Textbox ~ This is an Android TextBox container object that will be used to output the textCredits that is obtained.

Constructors:

i. onCreate ~ Default constructor for the Activity and will allow the screen manager to load the layout on to the screen and then initialise the Android TextBox to the object on the screen according to the layout loaded. It will then also set the text of the TextBox to the textHelp.

Game Manager Interface

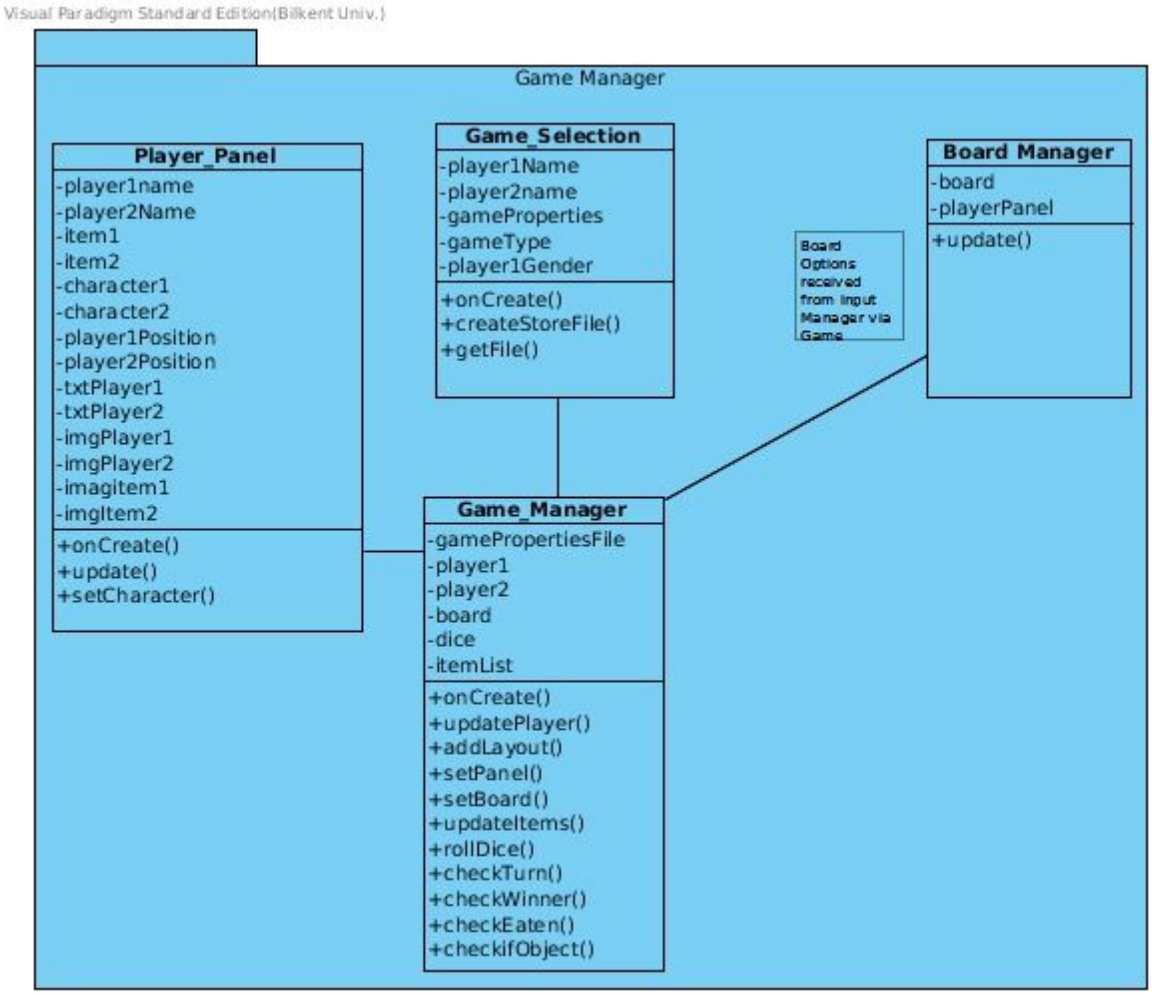

Figure 6 (Subsystem 2)

From figure 6 it can be seen that the game management class is responsible for controlling the players and deciding how and when to update the board of the game. It also controls what is displayed to the player regarding the player information and is responsible for updating the player information along with the board. Moreover, it must be noted that the Board Manager class is an entity that is controlled by the Android system when the application runs. It is not directly implemented but necessary for the need of the board graphics on the java canvas to be updated according the the actions of the user and where the players are currently on the board.

Another point to note is the Game Selection class. In reality this class will be implemented as two activities which will be seen in figure 6.2. This is because the player first chooses the

game type and then is taken to another activity where he then chooses the player properties. These Properties include the player names, and the character they have chosen. However, the game type along with the player names and their character selection in the internal storage of the Android system. .

Player Panel

Figure 6.1 (Player Panel in Subsystem 2)

Attributes:

i. player1name ~ The name of the first player input by user
ii. player2Name ~ The name of the second user input by user or can be Computer depending on game type.
iii. item1 ~ Integer value indicating which item is held by player 1. Value is -1 if no item held.
iv. item2 ~ Integer value indicating which item is held by player 2. Value is -1 if no item held.
v. character1 ~ Bitmap that holds the image of the player 1 depending on his character selection.
vi. character2 ~ Bitmap that holds the image of the player 1 depending on his character selection
vii. player1Position ~ Integer value which indicates the value of the block which player 1 is currently on.
viii. player2Position ~ Integer value which indicates the value of the block which player 2 is currently on.

ix. txtPlayer1 ~ Android TextBox object that displays the name of player 1.
x. txtPlayer2 ~ Android TextBox object that displays the name of player 2.
xi. imgPlayer1 ~ Android ImageBox object that displays the character of player 1.
xii. imgPlayer2 ~ Android ImageBox object that displays the character of player 2.
xiii. imagitem1 ~ Android ImageBox object that displays the item image of item 1.
xiv. imgItem2 ~ Android ImageBox object that displays the item image of tem 2.
xv. context ~ Context variable which will allow the panel to display the layout and get the images from the Drawable file for the Android system which stores all the images to be used.

Constructors:

i. onCreate ~ The default constructor for the game panel which will be displayed as a sub Layout by the Game Manager class on the activity. Since it will be an Activity in itself, it needs an onCreate constructor to be called when the Player Panel is displayed. It allows for the Layout file to be set from the Resources folder of the Android System. It then sets the Android object attributes to the objects on the screen layout. It also sets the images and relevant text onto these Android objects according to the Player objects passed as parameters to this activity.

Methods:

i. Update ~ Responsible for updating the item images displayed and the position of theplayer on the panel if necessary , after each roll.
ii. SetCharacters ~ Responsible for getting the correct character image for the each imgPlayer depending on the player selections made.

Game Selection

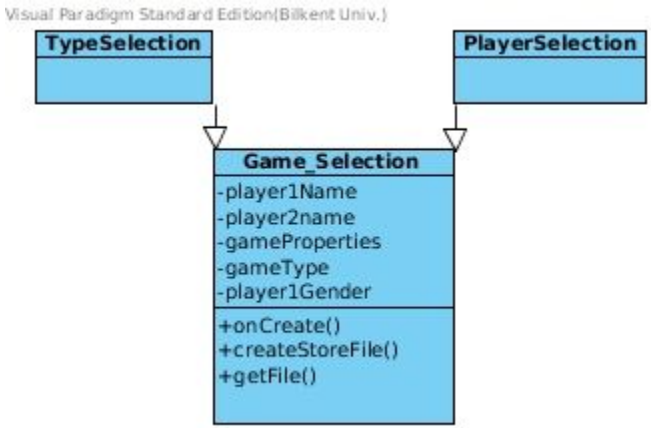

Figure 6.2 (Game Selection in Subsystem 2)

The Game Selection consists of two activities which run subsequently one after another. When the player selects to player the game he/she is directed to the Type Selection activity where the user selects if to play single player mode or multiplayer. After that the next Activity is loaded which allows him/her to choose the player names and characters depending on the game type selected in the previous Activity. Finally all this information is stored in a file in the internal storage of the Android System. Hence the attributes and methods can be combined in their description.

Attributes:

i. player1Name ~ String value that holds the name of player 1 when input into EditTextBox object which is an Android Object

ii. player2Name ~ String value that holds the name of player 2 when input into EditTextBox object which is an Android Object. If the game type is set to single player, then the player 2 name is set to 'Computer'.

iii. gameProperties ~ File object that is in .txt format and will hold all the information of the game.

iv. player1Gender ~ Boolean value that stores the gender of player 1. If it is male, then it can be said that player 2 will be female. Similarly if player 1 chooses female, then player 2 will have to be male.

v. gameType ~ Integer value that indicates if the game type is multiplayer or single player. If value is 1 then single player, and if the value is 2 then multiplayer.

Constructor:

i. onCreate ~ Since both the TypeSelection and PlayerSelection will be ndroid Activities, they will have to have the default onCreate constructor. As always the constructor will set the Android Objects to the matching layout objects being displayed.

Methods:

i. createStoreFile ~ Responsible for creating the file with all the relevant information of the game in it. It creates a .txt file and then stores it in the Android's internal memory which cannot be accessed by the user through his file manager on his phone. This is done at the end of the Player Selection Activity when the player presses the continue button.

ii. getFile ~ Returns the stored .txt file to whichever class class invokes the method. It returns the entire file from the internal memory of the Android System and not only parts. Parsing of the file can then be done in the the class that calls the method.

Game Manager

Figure 6.3 (Game Manager in Subsystem 2)

The Game Manager class is the main class that is responsible for managing the layout of the system and is also responsible for the setting up of the board and checking if additional player operations need to be performed after the roll of the dice.

Attributes:

i. gamePropertiesFile ~ File which is to hold the information regarding the game.
ii. player1 ~ Player object that holds player 1.
iii. player2 ~ Player object that holds player 2.
iv. board ~ Board object that holds an instance of the board object that is used.
v. dice ~ Dice object that holds an instance of the dice object that returns a random number from 1 to 6.
vi. itemList ~ An arraylist that holds a list of item objects which shall be placed across the board.

Constructor:

i. onCreate(): it will actually be the Game Manager activity that will be displayed to the user the duration of game play. Thus the default onCreate constructor is used which not only initialises the different layouts but also ensures that all the necessary game objects are initialised according to the preferences chosen by the player.

Method:

i. updatePlayer ~ Updates the player's position attribute.
ii. addLayout ~ Adds the game panel, board and the dice all into the screen.
iii. setPanel ~ Sets the player panel.
iv. setBoard ~ Draws the play board onto the Java canvas
v. updateItems ~ Toggles items on the board on and off depending on player movement.
vi. rollDice ~ Invokes the dice object initialized to return random number between 1 to 6
vii. checkTurn ~ Alternates the turns between the players.
viii. checkWinner ~ Checks if any player has reached the 100th block yet.
ix. checkEaten ~ Checks if one player has landed onto another player and moves the initial player on the block back three blocks.

x. checkifObject ~ Checks if the block on which the player has landed has the head of a snake or the bottom of a vine.

Board Manager

The Board Manager class is responsible for updating the board drawn on the canvas. This is done by methods within the Game Manager class but is handled separately by the Android System. hence can be considered as separate entity.

Game Entities

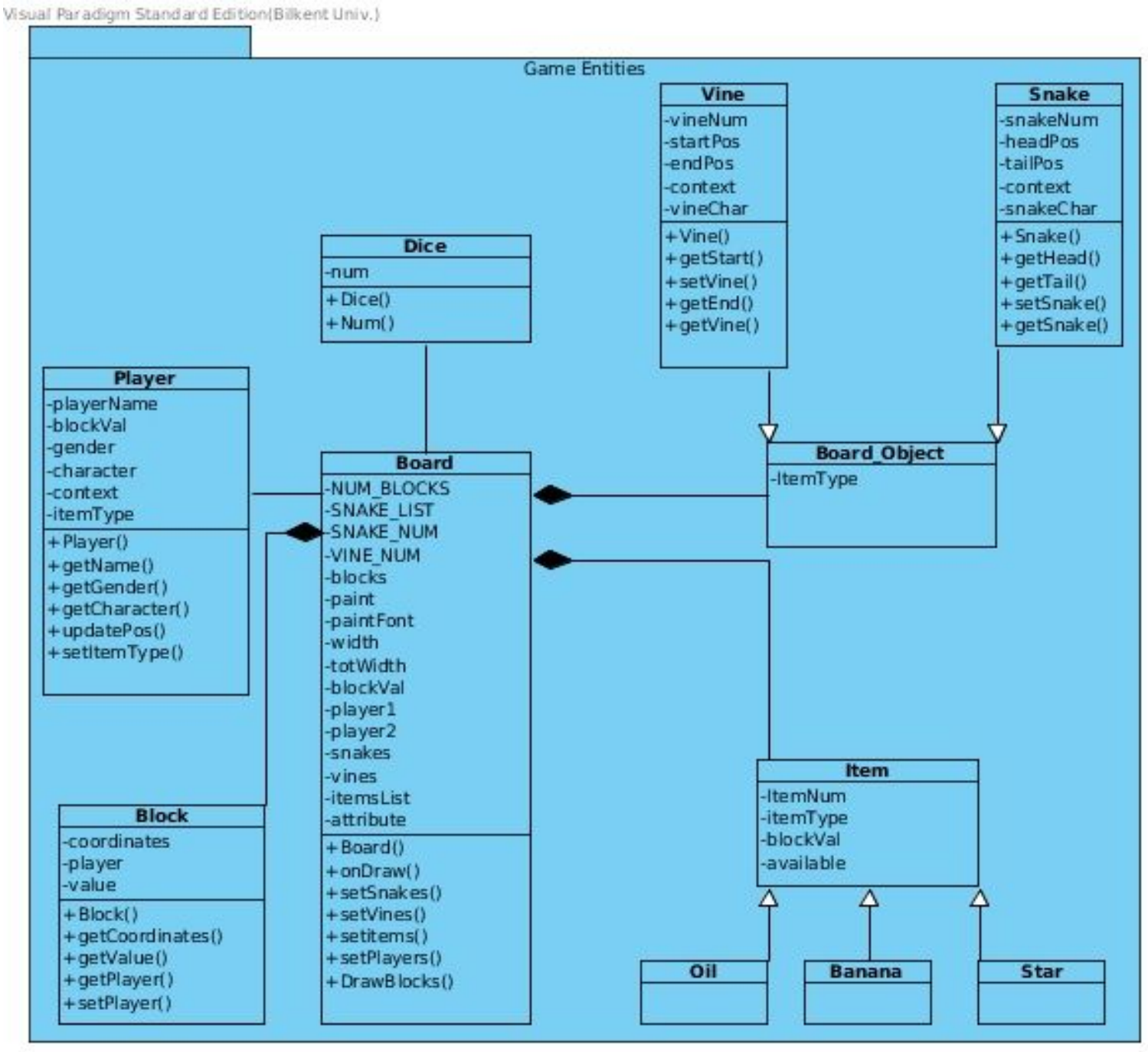

Figure 7 (Subsystem 3)

As can be seen from figure 7, all game entities are mainly connected to the Board entity. Furthermore, the board items are derived from the item class and the snake and vine board objects are different types of board objects.

Dice Class

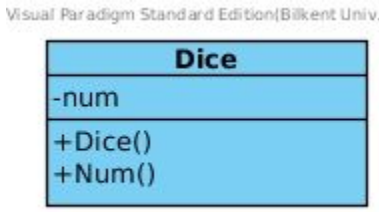

Figure 7.1 (Dice Class in Subsytem 3)

Attributes:

i. num ~ Integer value that holds the randomised number

Constructor:

i. Dice ~ It is a very simple and necessary constructor that sets the 'num' attribute that this class has, to 0.

Methods:

i. getNum ~ Returns a random number between 1 and 6, like a dice.

Snake Class

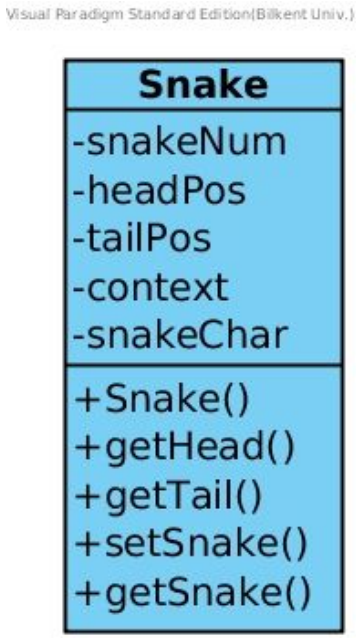

Figure 7.3 (Snake Class in Subsystem 3)

Attributes:

i. context ~ allows us to access the android system resources and will allow us to get the picture of the snake.
ii. snakeChar ~ Enables program to show snake picture.
iii. headPos ~ integer value indicating which block the snake's head is on.
iv. tailPos ~ integer value indicating which block the snake's tail is on.
v. snakeNum ~ The number of the snake out of the list of snakes.

Constructor:

i. Snake ~ creates context with given attributes.
ii.

Methods:

i. getHead ~ Returns the position of head of snake.
ii. getTail ~ Returns the position of tail of snake.
iii. setSnake ~ Gets the picture from the Resources file of android and sets the right picture of the snake according to where the snake points
iv. getSnake ~ Returns snake character.

Vine Class

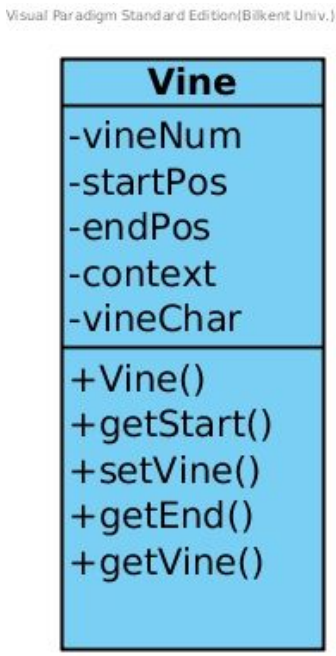

Figure 7.4 (Snake Class in Subsystem 3)

Attributes:

vi. context **~** allows us to access the android system resources and will allow us to get the picture of the snake.

vii. vineChar **~** Enables program to show vine picture.

viii. startPos ~ integer value indicating which block the vine's start is on.

ix. endPos ~ integer value indicating which block the vine's end is on.

x. snakeNum ~ The number of the snake out of the list of snakes.

Constructor:

iii. Vine ~ creates context with given attributes.

Methods:

v. getStart ~ Returns the position of start of vine.

vi. getEnd ~ Returns the position of end of vine.

vii. setVine ~ Gets the picture from the Resources file of android and sets the right picture of the vine according to where the vine points

viii. getVine ~ Returns vine character.

Player Class

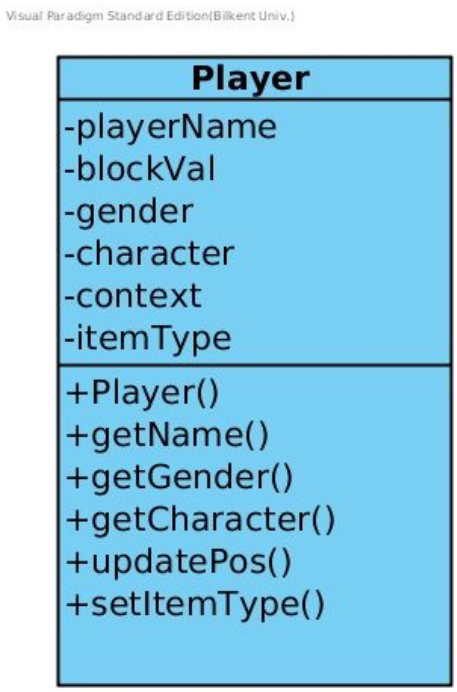

Figure 7.2 (Player Class in Subsystem 3)

Attributes:

i. playerName ~ String value of name of the player as set by the user.
ii. blockVal ~ The value of the block the player is currently on.
iii. gender ~ Boolean value of whether the character of the player is male or female.
iv. character ~ Bitmap value that holds the picture of the player that will be displayed on the board and depends on the gender value.
v. context ~ Context value that allows the class to access the Android System resources.
vi. itemType ~ Integer value indicating the item that the player may posses at the moment.

Constructor:

i. Player ~ Creates players' blocks and their context with given variables.

Methods:

i. getName ~ Returns player name
ii. getBlock ~ Returns block value
iii. boolean getGender ~ Returns players' gender
iv. getItemType ~ Returns item type that the player currently has
v. getCharacte ~ Returns character
vi. updatePos ~ updates the player's position
vii. setItemType ~ Sets the item as an integer, where all the integers correspond to an item

Block Class

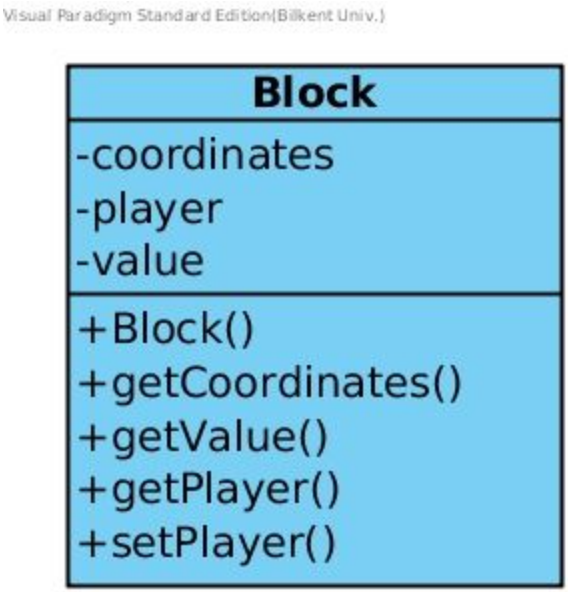

Figure 7.5 (Block Class in Subsystem 3)

Attributes:

i. coordinates ~ Rectangle is a shape in Java that is drawn and holds the x, and y coordinate of each block.
ii. player ~ Boolean value which indicates if there is a player on the block or not
iii. value ~ Value of the block

Constructor:

i. Block ~ Draws the block that is rectangle, with given attributes

Methods:

i. getCoordinates ~ Returns coordinates

ii. getValue ~ Returns value

iii. getPlayer ~ Return player

iv. setPlayer ~ Accepts player as true

Item Class

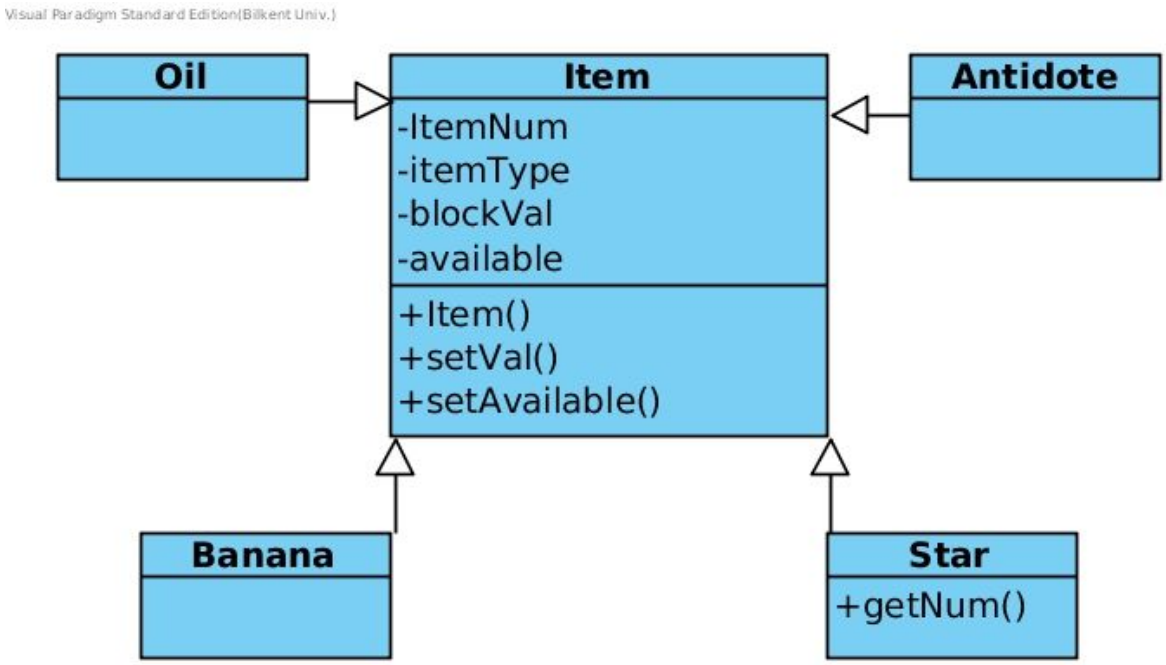

Figure 7.6 (Item Class in Subsystem 3)

As can be seen from figure 7.6, the item class has four sub classes which includes the items that can spawn and despawn on the object. They all inherit the properties of the item class including the methods. However the Star object returns a random number which sends the player either forward or backwards depending on the random number generated.

Attributes:

i. itemType ~ Integer value that indicates which item the object is
ii. itemVal ~ Integer value which indicates which item the object is in the list
iii. blockVal ~ Integer value that indicates which block the item is on
iv. available ~ Boolean value which indicates if the item is present on the board or not.

Constructor:

i. Item **~** Creates an instance of the item which then creates the item type.

Methods:

i. setVal ~ Sets the value of blockVal to set the position of the item on the board.
ii. setAvailable ~ Responsible for spawning and despawning the item on the board.
iii. getNum ~ Available exclusively for the Star item and is responsible for sending the player either forward or backwards depending on the random number generated which is in between -3 or 3.

Item Class

Figure 7.6 (Board Class in Subsystem 3)

Attributes:

i. NUM_BLOCKS ~ Fixed integer value that will be set to the number of blocks to be drawn.
ii. SNAKE_LIST ~ Fixed Array of snake head and tail positions.
iii. SNAKE_NUM ~ Fixed integer indicating number of snake objects on board.
iv. VINE_LIST ~ Fixed Array of vine start and end positions.
v. VINE_NUM ~ Fixed integer indicating number of vine objects on board.
vi. block ~ instance of block object drawn on board.
vii. paint ~ Java Paint to draw objects on canvas
viii. paintFont ~ Java PaintFont for the font of drawn object on the board.
ix. player 1 ~ Player object for player 1 on board.
x. player 2 ~ Player object for player 2 on board.
xi. snakes ~ Arraylist of snake objects.
xii. vines ~ Arraylist of vine objects.
xiii. itemList ~ Arraylist of item objects.

Constructor:

i. Board **~** Creates an instance of the board which then draws all the blocks and board objects on the canvas of the screen.

Methods:

i. onDraw ~ Draws the objects on the canvas including the snake character, the vine characters, the item images and the players characters, along with all the 100 blocks.
ii. setSnakes ~ Transposes the snake images onto the board.
iii. setVines ~ Transposes the vine images onto the board.
iv. setImages ~ Positions all the items in the item list onto the respective board blocks.
v. setPlayers ~ Updates the position of the players and redraws the player position onto the board after each dice roll.
vi. drawBlocks ~ Recursively draws each block onto the Java canvas.