CS 319 - Object-Oriented Software Engineering

System Design Report

Jungle Pursuit

Fatma Begüm İlhan

Syed Sarjeel Yusuf

Sinan Öndül

Contents

1. Introduction

    1.1 Purpose of the system

    1.2 Design Goals

        End User Criteria:

        Maintenance Criteria:

        Performance Criteria:

        Trade Offs:

            Ease Of Use and Ease of Learning vs. Functionality:

            Performance vs. Memory:

    1.3 Definitions, acronyms, and abbreviations

        Abbreviations:

    1.4. References

    1.5. Overview

2. Software Architecture

    2.1. Overview

    2.2. Subsystem Decomposition

    2.3. Architectural Styles

        2.3.1 Layers

        2.3.2 Model View Controller

    2.3. Hardware / Software Mapping

    2.4. Persistent Data Management

    2.5. Access Control and Security

    2.6. Boundary Conditions

3. Subsystem Services

    3.1. Design Patterns

        Façade Design Pattern:

1. Introduction

1.1 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.

1.2 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. 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 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.

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.

1.3 Definitions, acronyms, and abbreviations

Abbreviations:

MVC: [2] Model View Controller

SDK: [1] Software Development Kit

1.4. 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.*

1.5. 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.

2. Software Architecture

2.1. 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

2.2. 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.

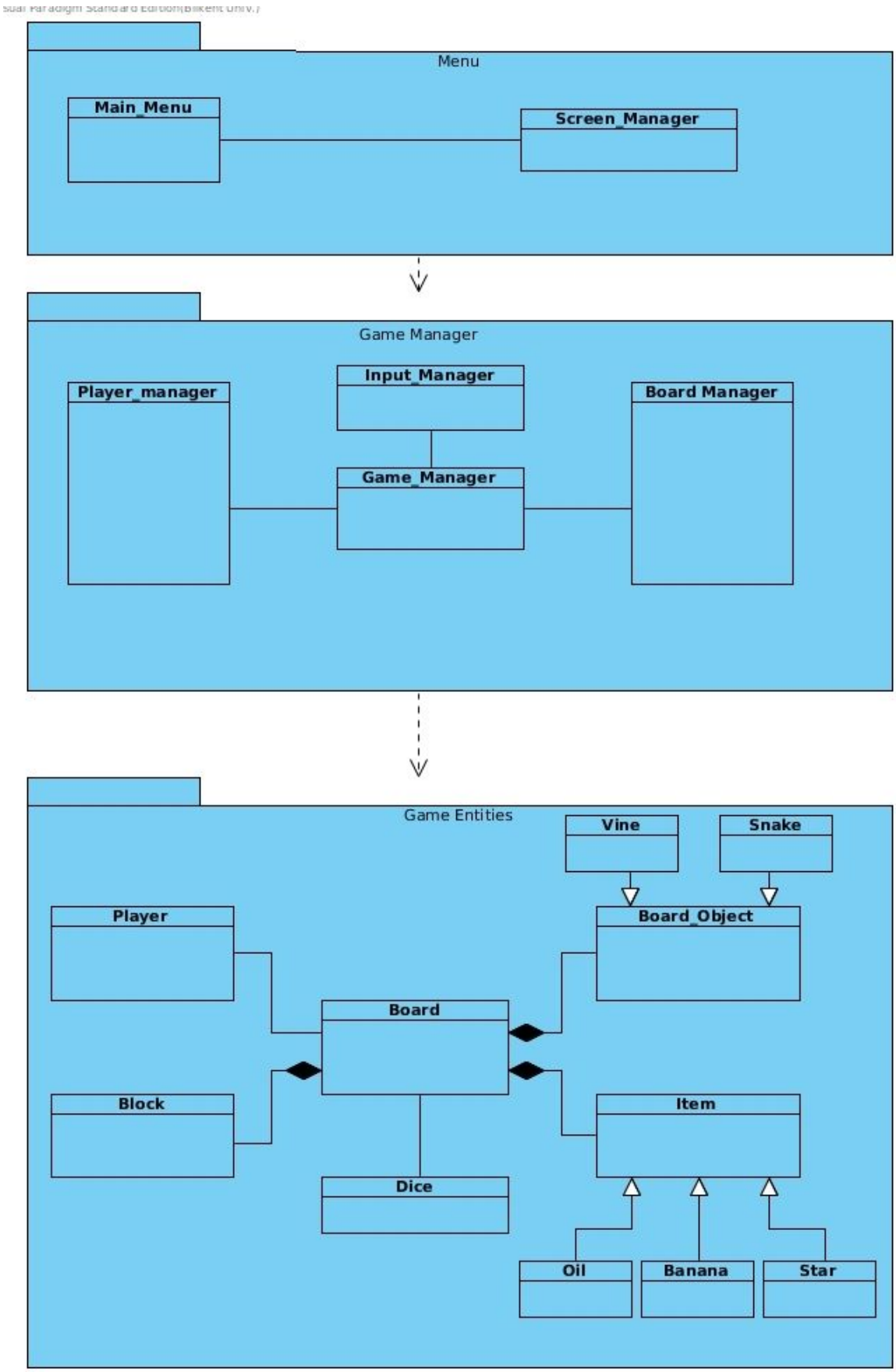

Figure – 1 (Basic Subsystem Decomposition)

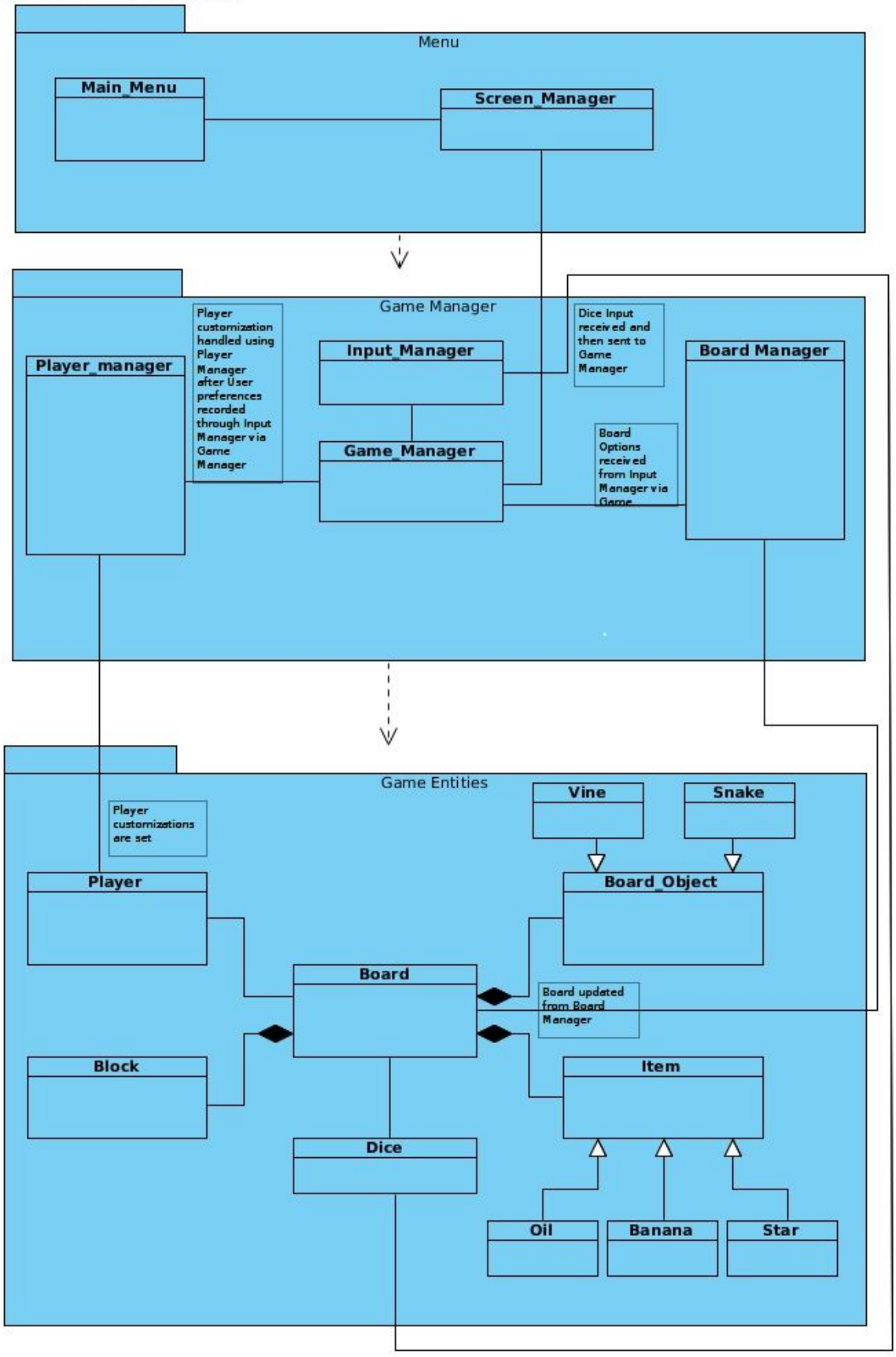

Figure-2(Detailed Subsystem Decomposition)

2.3. Architectural Styles

2.3.1 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.

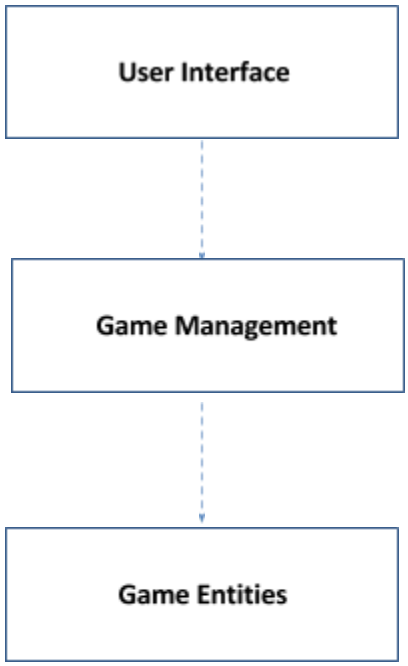

Figure-3(Layers of system)

2.3.2 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.

2.3. Hardware / Software Mapping

Crazy Ball 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.

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. 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.

2.4. 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.

2.5. 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.

2.6. 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.

3. Subsystem Services

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

3.1. 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 GameMap class which handles the operations on entity objects of our system, according to the needs of Game Management subsystem.

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.

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)

NB: Detailed Design to be continued.