1. Introduction

1.1 Purpose of the system

1.1.1 General information

Space Shooter game is a 2-D space game. Comparing to 3-D games including a real world environment and high quality graphics, Space Shooter game looks very easy. However, gameplay of Space Shooter gives challenging quests so that player wants to go further in game. It is also distinguishable among other space shooter game with its difficulty level and different level ups which are making gameplay more enjoyable. Space Shooter aims to improve rapid decision making ability, hand-eye coordination and reflexes, while having good time.

1.1.2 How the game works?

According to Bartle's Taxonomy, the game aims to fulfill the expectations of achievers, killers and explorers. In the game, there will be a spaceship which is brought to a corrupted area to complete a task. The game ends when the player kills all the enemies on the map. If player kills an enemy another one will be generated after 2 minutes which can change later on at implementation. Hence the player has 2 minutes to kill all the enemies. However player does not have that much power at the beginning of the game. Player needs to kill a lot of enemies or achieve quests to level up and improve the ship.

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.

End User Criteria:

Ease of Use: Since our system is a game, it should provide good entertainment for the player so that player shouldn't be get bored. 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 mouse input from the user, like clicking buttons and keyboard. This makes it easy to use the system from the point of the player.

Ease of Learning: Since player is not ought to have knowledge about how the game is played, loss-win conditions of game, level-up properties. It is important for the user to obtain information about the game concepts, for this purpose system will provide an help part, 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 or by reading the help document.

Maintenance Criteria:

Extendibility: In general, in the lifetime of game software, it is always important to add new components, quests, 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 number of enemies) easily to the existing system.

Portability: Portability is an important issue for a software, since it provides that the software can reach wide range of users. In this respect we are determined that the system will be implemented in Java, since its JVM provides platform independency, our system will satisfy the portability.

Modifiability: In our system it would be easy to modify the existing functionalities of the system. 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.

Performance Criteria:

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 almost immediate, while also displaying animations, effects smoothly for enthusiasm.

Trade Offs:

Ease Of Use and Ease of Learning vs. Functionality:

In our system we determined that player should be able to learn and use the system very easily. Therefore our design proposes that the priority of the usability is higher than functionality. In other words our system does not bother the user with complex functionalities or we do not make the user to be lost in many functionalities, in order to make our system easy to understand and use.

1.3 Definitions, acronyms, and abbreviations

Abbreviations:

MVC: Model View Controller

JDK: Java Development Kit

JVM: Java Virtual Machine

CP: Composite Pattern

FP: Façate Pattern

1.4. References

[1] https://en.wikipedia.org/wiki/Composite_pattern

[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] https://www.youtube.com/watch?v=JQJA5YjvHDU

[4] https://www.youtube.com/watch?v=YQ44hVeVdEw

1.5. Overview

The system which is going to be implemented is a two dimensional game which takes place at a space map near a station. The game will consist of enemies in the map and a main character which are all spaceships. The character will level up by playing the game properly which means killing enemies for experience points, collecting gift boxes and completing optional quests. Quests will be there for explaining the story, tutorial and leading the player through the game. The spaceship will be more powerful at each level and game will be over if the player kills all the enemies at the map. The game will end when player kills all the enemies on the map. Enemies will be regenerated after a while when they are killed. This will give the player a specific amount of time to kill all the enemies on the map.

2. Software Architecture

2.1 Overview

Since the system we are up to implement is a game, view part of the game is an important subject. Hence separating the view from the model is quite important for decomposing the system in a way that one person can comprehend the complexity of the subsystem. Hence Model View Controller (MVC) pattern is a good choice for the general structure of the project.

In the view part of the game which includes both sound and image viewing, Composite Pattern (CP) is going to be used for the collection of views.

The view and model parts will be controlled via controller object. This control is easy since there are only few keys the player can stroke. Hence controller does not need to increase the coupling between two packages. The view and model will be controlled through one Game Manager class which means it will have the Façade Pattern (FP).

2.2 Subsystem Decomposition

As it is stated above, the whole system will be decomposed into model, view and controller using MVC model. Since view part is too much for just one person to comprehend and handle, view part is also decomposed into other subsystems which are called engine, sound and user interface.

In Figure 1, the overall of all packages are shown. As seen view package is decomposed into three other packages. Engine part handles the view of objects on the game screen. Sound part handles sound mechanics of the game. User interface both shows the game screen to the user and other menu items like play game, help, credits and extra.

[figure 1]

To reduce coupling between classes, Game Control is connected to model via one Façade Class. Model class is extending classes from Engine package to create its Actors and ActorCollections for the game and implement the mechanics and logic upon them. This is where most of the coupling between two packages is happening.

2.3 Architectural Structure

2.3.1 Layers

There are three layers in the system. The first layer is the Actor level. Actor level includes Actor and ActorCollection which are Drawable GameObjects. The second layer consist of entities which are extending from Actor and ActorCollection to implement general behaviour of Actors. Third layer has GameScreen, User Interface and Game Control to interact with user. The layer system is shown in Figure 2.

[figure 2]

2.3.2 Model View Controller

As stated above, the system is going to be mainly structured using the MVC pattern. The model will not have a whole different class implementation but model classes will extend the view classes to implement the model upon view. Instead of nofiying the view from model classes, view classes will have a frame based structure in which controller class will call each view object's tick() method in each frame. View classes will have the data from model classes and update themselves at each frame.

2.4 Hardware/Software Mapping

The game does not require too much quality in hardware since it is a simple game. The system will be rendering only what should be seen on the screen. Hence this will optimize the rendering. The game also destroys unnecessary objects depending on the garbage collector of Java. The game requires keyboard and a mouse to play the game.

Game uses images to render the graphics. Hence this creates some delays at the first opening. However after the map is loaded or rendered, there is no delay on other objects.

2.5 Persistent Data Management

Since the game does not have too complex data to handle, the project will not include any database. The level of the player and enemies and quests are going to be stored in the memory directly. The images are going to be stored in the Images file and sounds are going to be stored in Sounds file.

2.6 Access Control and Security

This part is not applicable since the game does not reuire any log in and does not require any network connection. Security problem does not exist since the game does not apply any licensing and network connection.

2.7 Boundary Conditions

2.7.1 Initialization

The game does not have any installation. Hence the program will be executed via a .jar file.

2.7.2 Termination

The game can be closed with the “x” button on the game window. However the menu will stay if the game is closed. The program will be closed if the “x” button or “Quit” button on the menu frame is cliked.

2.7.3 Error

In an error situation, images and sounds may not be loaded. The mechanics of the game might not be working well. If the performance issues occurs, the game progress will be lost.