System design document for group 24

Version 0.2

2015-05-14

Rasmus Andersson
Oliver Tunberg

1 Introduction

1.1 Design Goals

The main goal of Dumbit is the combination of fast rated skill and roleplay. Gameplay will consist of quick paced combat sequences where experience and fast maneuvering dictates the outcome of the battle. Levels are layered with ever growing difficulty designed to make progressing harder then only defeating enemies. As important will be to build power by improving the properties of the playable character. Completing stages and explore the surroundings will earn the player much needed tools on the advancing front. The combination of two forces the the source of the design goal. A strong visual provided by the characters and the design theme of the game works as a powerful attraction to customers. New gamers are directly stimulated by the engaging initial combat system. Further on in the program, users will stay to complete the experience of making choices and enjoying a beautiful and exciting scenery.

As the controllers require a keyboard and a mouse cursor, Dumbit will be exclusive on pc devices only. A crucial part of the game is the efficient running. The program will be very light and easy to play. The game uses a quick start up and players will be able to instantaneously resume their latest progress. This makes the product excellent time filling enjoyment for a short period of time as for example when the user is waiting for something. As well as more intensive playing to fully appreciate the experience.

Many years of games have provided much valuable information about the sources of attraction with the customers. To fully utilize this opportunity Dumbit focuses a substantial amount of attention on modern and popular gaming elements. To reach the games full potential, users will be able to collect currency, compare and select a wide variety of weapons and items as well as make difficult choices concerning important consequences.

1.2 Definitions, acronyms and abbreviations ‘

GameObject - An object within the game which can be displayed during gameplay. GameObject can be divided into different categories with it's own special functionality and purpose.

Entity - A type of GameObject that has the ability to move dynamically in the gameworld.

Tile - A type of GameObject that is a static object on the screen and has a fixed position.

Sprite - A part sequence image that will be rendered on the screen.

SpriteSheet - A whole sequence of animated images that will be animated and rendered on to the screen.

WorldMap - the view where u get an overview of all the different levels and areas you are able to discover in the game.

2 System design

2.1 Overview

In this section we explain the overall design choices.

MVC - This design pattern is preferred in general in both game and software development. The pattern is superior in it's kind and the key is the separation of Controller, View and Model. This makes it possible to add multiply controllers and view but also to switch between them easily. This is well fitting into the need of changing the views in the game. With this pattern it makes it simple to change view between star-menu, worldmap, settings and the gameplay view.

Observer - The observer pattern including one observable class and some observer klass which receive a message when a certain event happens in the observable class. this is preventing unnecessary dependency between classes and to avoid separate calls to every single observer class. We are using this pattern to alert all the Gameobjects to update and render simultaneously.

Decorator - This pattern makes it possible to add new functionality to an object without changing the structure. the implementation of this pattern ll be inte the items and inventory system. with other words when u add a new item to your player, additional abilities ll be added to the player.

2.2 Software decomposition

2.2.1 General

Package diagram. For each package an UML class diagram in appendix

2.2.2 Decomposition into subsystems

2.2.3 Layering

2.2.4 Dependency analysis

In the Dumbit program the connection of delegation and inheritance is designed with balance in thought. While light dependency between classes are continuously strived for it has been beneficial for certain classes to maintain a higher surveillance over an area of other classes. An example the handler class which dictates the structure of the number of units currently occupies the active playground.

2.3 Concurrency issues

2.4 Persistent data management

2.5 Access control and security

2.6 Boundary conditions

3 References

APPENDIX