System design document for Proximity

Version: 3.0
Date: 18/5 2015
Author: Linda Evaldsson, Simon Gislén, Johan Swanberg, Hanna Römer

This version overrides all previous versions.

1. Introduction

1.1. Design goals

The design should be loosely coupled but the main focus is to have an advanced modular functionality. It must be easy to expand the program and add new components and functions. In addition, the design needs to be testable, that is the classes needs to be isolated and encapsulated.

1.2. Definitions, acronyms and abbreviations

- **Map**, the game board
- **Experience**, Points gained from killing enemies. When a certain amount of experience is collected the next level is reached.
- **Wave**, The enemies come in clusters, the current wave is the cluster the player is currently facing.
- **GUI,** graphical user interface.
- **MVC**, Mode-View-Controller, a way to organize the structure of an application to avoid mixing application code (controller), data (model) and GUI (view).
- **Passive MVC,** an alternative way of structuring MVC where the model don't tell the view to update, instead the controller tells the view to update.
- **Java**, a platform independent programming language.
- **Encapsulation**, hiding properties in objects from the "outside" to protect from manipulation and to bundle related methods.
- **Java math library,** The math library included with java.
- **Stan,** a program which generates uml from code.

2. System design

2.1. Overview

The application will be written in java and use passive MVC.

2.1.1. Game states

To differentiate between different views the game will have several game states that it can switch between. Playing a map is one game state and viewing the menu is another. This will make it easy to keep coupling low and connections between classes loose, since these states need a minimum of communication with one another.

2.1.2. Maps and play state

The application will feature different maps. The interface of the start menu will display a list of available maps. Each map will have a previous map that has to be won before the map is available. Selecting a certain map will initiate a new game state that will take the selected Map as an argument. This way we only need one play-state, since it can adapt to the chosen map.

2.1.3. Encapsulation

The application will avoid globally accessible variables and methods to make the interface for programming easier to understand, expand and change. Using encapsulation will make it easier if the project were to expand with new people working on it, it is easier to get a grip of a program with with encapsulation.

Encapsulation also prevents modification of data that should not be manipulated from outside the program.

2.1.4. Changing language support

In order to make the language easy to change all texts will be stored in a class with constants that can be accessed from the program. This will make it easier to change the language, although in the first version only english will be supported.

2.1.5. Point calculations

Since the game requires a smooth frame rate both speed and reliability is important. In the application the methods and implementations should be adapted to prioritize speed.

For example for calculating the distance between two points tan should be used instead of cos or sine since both cos and sine uses the hypotenuse for the calculation. The reason the hypotenuse calculation is slow is because it uses a square root calculation. Generally all implementations will avoid using square root calculations because no relatively efficient square root calculating method has been found.

The program should also aim to implement methods that ask for distance to the power of two, since that also avoids a square root calculation.

The java math library will be avoided since it prioritizes precision over speed. For example the method for calculating angles uses a mathematical formula to calculate and approximate an answer with 8 decimals precision. It is a lot faster to find an approximation in a lookup table, at the cost of precision .

2.1.6. Image cache

Another speed improving technique that should be used for images is cache. The Image class for representing images should contains a static hashmap that is used to save and lookup textures once they’ve been loaded into memory. This means that if the program

tries to create a texture that already exists, it will instead be given the reference to the already existing resource.

Since all creeps, towers and projectiles use the same image resource, and this image is never manipulated, the image data is stored as a static variable in the object class, which also means that the image never has to be disposed.

2.2. Software decomposition

2.2.1. General

The application is mainly divided into the following packages, se figure X (figure to be added).

- Models, where the data is stored, for example Creeps and Spells
- Viewers, where the data is rendered to the screen
- Utilities, helper classes and services that are very general and can be used in a lot of different scenarios
- Controllers, where input is handled and commands are given to the model classes

2.2.2. Decomposition into subsystems

No subsystems.

2.2.3. Layering

Image from stan will be added

2.2.4. Dependency analysis

Image from stan referenced here. Something will be written about circular dependencies here. This cannot be claimed until the program is done.

2.3. Concurrency issues

NA the application is single threaded.

2.4. Persistent data management

The application data is stored as a data file, containing a Hashmap object typed as <String, Double>. The object can read and write to different numbered files, which enables handling multiple save slots for different people. The save file is used to store:

- Experience, the amount of experience the player has
- Waves, how many waves the player has defeated on each map

2.5. Access control and security

NA

2.6. Boundary conditions

NA. Application launched and exited as normal desktop applications.

3. References

Model-View-Controller (2013) *I iOS Developer Library*
Hämtad 2015-05-06 från
https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

Basher, Khademul (2013) *MVC Patterns (Active and Passive Model) and its implementation using ASP.NET Web forms*
Hämtad från
http://www.codeproject.com/Articles/674959/MVC-Patterns-Active-and-Passive-Model-and-its

The role of Performance (2011) Uppsala universitet
Hämtad från
http://www.it.uu.se/edu/course/homepage/dark/ht11/dark6-performance.pdf

APPENDIX