System design document for Proximity

Version: 2.0
Date: 4/5 2015
Author: Linda Evaldsson, Simon Gislén, Johan Swanberg, Hanna Römer

This version overrides all previous versions.

1. Introduction

Introduction

1.1. Design goals

The design must 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

- **Experience**, Points gained from killing enemies. When a certain amount of experience is collected the next level is reached.
- **Level**, The players factions has a faction level. It is a display of how much experience the player has with that faction.
- **Wave**, The enemies come in clusters, the current wave is the cluster the player is currently facing.
- **Faction,** A theme with a spell-set that the player can chose from; different factions offer different spells. The player can level up their factions.
- **Tower,** A block that the user can place on the screen. The tower costs resources. Most towers fire projectiles at enemies or attack them in some other way.
- **Projectile,** Most towers can shoots projectiles that hit the enemies, these projectiles affect the creep(s) it hits, for example by freezing or devolving them.
- **Creep,** Enemies that follow a certain path in the game and tries to kill the player. These can be of different types and strengths.
- **Devolve,** When an enemy dies, it “reincarnates” as a smaller enemy type, if it’s the smallest type of enemy the enemy gets destroyed
- **Resource,** “Money” that the player can use to purchase towers, different towers require different resources.
    - Line, A type of resource
    - Point, A type of resource
    - Polygon, A more valuable type of resource
- **Upgrade,** The player can upgrade towers to make them better
- **Spell,** The player gets four spells from the faction and this spell can be used to affect the game in the players favor
- MVC,

2. System design

2.1. Overview

The application will 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

The application will feature different maps. The interface of the start menu will display a list of available maps. The maps will be reachable through connected nodes, each node will contain a map. Selecting a certain map will initiate a new game state.

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

2.3. Concurrency issues

2.4. Persistent data management

2.5. Access control and security

2.6. Boundary conditions

3. References

APPENDIX