System design document for Prjkté

Version:	1.0
Date:	2016-03-23

Autors:	Arvid "Alle" Björklund
	Axel "Mouseman" Bergrahm
	Frej "Täcket" Karlsson
	Ludvig "Frej" Ekman

This version overrides all previous versions.

1 Introduction

1.1 Design goals

The design must be loosely coupled to make it possible to switch different components like physics and graphics engine. This also makes it easier to maintain, since you will be able to rewrite a part of the program without breaking anything else. It should also be easy to write tests, since we are doing a test-driven development.

1.2 Definitions, acronyms and abbreviations

- Game loop: One iteration of the game were for example, a player moves or picks up an item.

- Game object: An object that appears on the screen, can be anything from a dead cat to a raging smurf.

- Character: extends from Game object. Is a controlable object that can deal damage and move.

- State: what current action a character is doing, i.e. punch, walk, jump.

2   System design

2.1   Overview

The game will use something similar to a MVC pattern. On the top we have a gameloop that constantly updates itself. The gameloop tells the Game class to update itself. We have several different controllers (scenes) that handels different usecases. For example selecting characters and fighting in a Arena. Arena also has a subcontroller handeling more complex usecases. Most models extends GameObject, this includes playable characters and items which have similar characteristics.

The view paints the different gameobjects and background (the 'Map').

2.1.1   Aggregates

The program consists of several different rootclasses that handels the communication. The communication between these all pass through the top class 'Game'.

Other controllers has their own models. Arena also has a separate subcontroller that has responsibility for controlling the characters and their movement.

2.1.2   Unique identifiers and global look-ups

Imgage files used for sprites and animations use their filepath as an identifier. Also concrete implementations of Characters and Maps have unique names since they are used as an identifier when using their coresponding factory classes.

2.1.3   Events paths

As earlier described the game is updated from a gameloop that updates itself. This loop then tells the game to update itself which in turn tells the active scenes controller to do the same. The Scenes check if relevant keyboardkeys are pressed via the Input class. After this it updates the models and tells the view to paint both the map and gameobjects present.

The models are updated both by setters and by updating the models certain steps, not caring what state the models currently are in.

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

References

Appendix