System design document for BoBo The Panda

Version: 1.2

Date 2014-05-24

Author Sebastian Sandberg, Elvira Jonsson, Victor Larsson, Oscar Muhr
//This version overrides all previous versions.

1 Introduction

1.1 Design goals

The design must be loosely coupled to make it easy to implement new objects in the model. The goal is to have a well structured Model-View-Controller pattern, where the graphics, input and logic are separated from eachother.

1.2 Definitions, acronyms and abbreviations

- GUI, Graphical User Interface
- Java, platform independent programming language
- JRE, Java Runtime Enviroment. Necessary software to be able to run Java applications.
- Slick2D, an external library for a 2d game engine
- Maven, tool to build the project, with necessary dependencies.
- MVC, Model-View-Controller. A designpattern.
- Tiled, Map creation tool
- TMX, Open XML-standard

- Level, what the game consists of, get to the next by grabbing the key and enetering the door
- Character, the player controlled game character
- User, the person playing the game

2 System design

2.1 Overview

For the project a well designed MVC structure was implemented and combined with clearly following the Open-Closed principle, this became the foundation of the program.

2.1.1 The Model Functionality

Level holds the objects of the gameplay functionality. The game menu is separated from the game play and to play you simply switch from the menu state to the game play state. In game menu it is the class Menu that hold all the objects related to the menu.

2.1.2 Collision

Collision between objects is detected by deciding wheater or not their hitboxes intersect. Handling the different cases of collision is solved by using the visitor pattern. The object affected by the collision is the visitor.

2.1.3 AbstractMapObjects

All interactive objects in the game are instaces of AbstractMapObject. Some doesn't affect themselves by collision. These objects are instances of fixed object and IVisitable. Objects who do get affected by collision implements both interfaces IVisitor and IVisitable.

2.1.4 Event handling

By using the popertychangesupport and the propertychangelistener libraries, the models where able to communicate with the views without needing strong coupling.

2.1.5 Menu

The menu listens to mouse events. The menu sends events to listeners in the event of proper mouse action.

2.2 Software decomposition

2.2.1 General

- **model,** the core of the game. Model in the MVC
- **view,** the graphical representation of the model. Contains logic from Slick2D
- **controller,** handles user input and modifies model. Contains logic from Slick2D
- **utilities,** i/o handling. Reads the tmx file and translates the data necessary for the model
- **main,** starts the game, initializes the necessary elements for gameplay

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

2.2.2 Decomposition into subsystems

There are no real subsystems in the program. However there is the utilities package that takes care of handling

2.2.3 Layering

See figure 1.

2.2.4 Dependency analysis

Dependencies are as shown in figure 1. There are no circular dependencies.

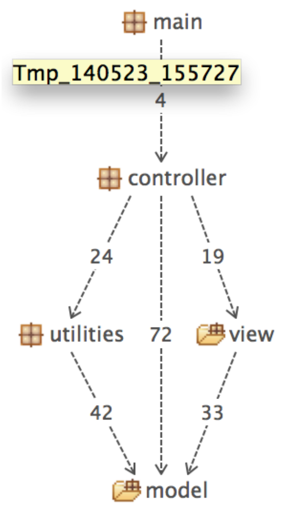

Figure 1. Layering and dependency analysis

2.3 Concurrency issues

NA. This is a single thread application. There is a possibility that more threads, background threads, could have been implemented to make response even better.

2.4 Persistent data management

NA. Terminating the programme will result in losing all data.

2.5 Access control and security

NA

2.6 Boundary conditions

This application requires Maven and JRE to run. Run from terminal using Maven exec-plugin or open project in Maven supported IDE, for example Netbeans. VM options needs to be configured before running. Set the library path; -Djava.library.path=/path/to/native/folder/in/project

3 References

http://en.wikipedia.org/wiki/Platform_game#Single_screen_movement
MVC, http://en.wikip edia.org/wiki/Mo del-view-controller

APPENDIX

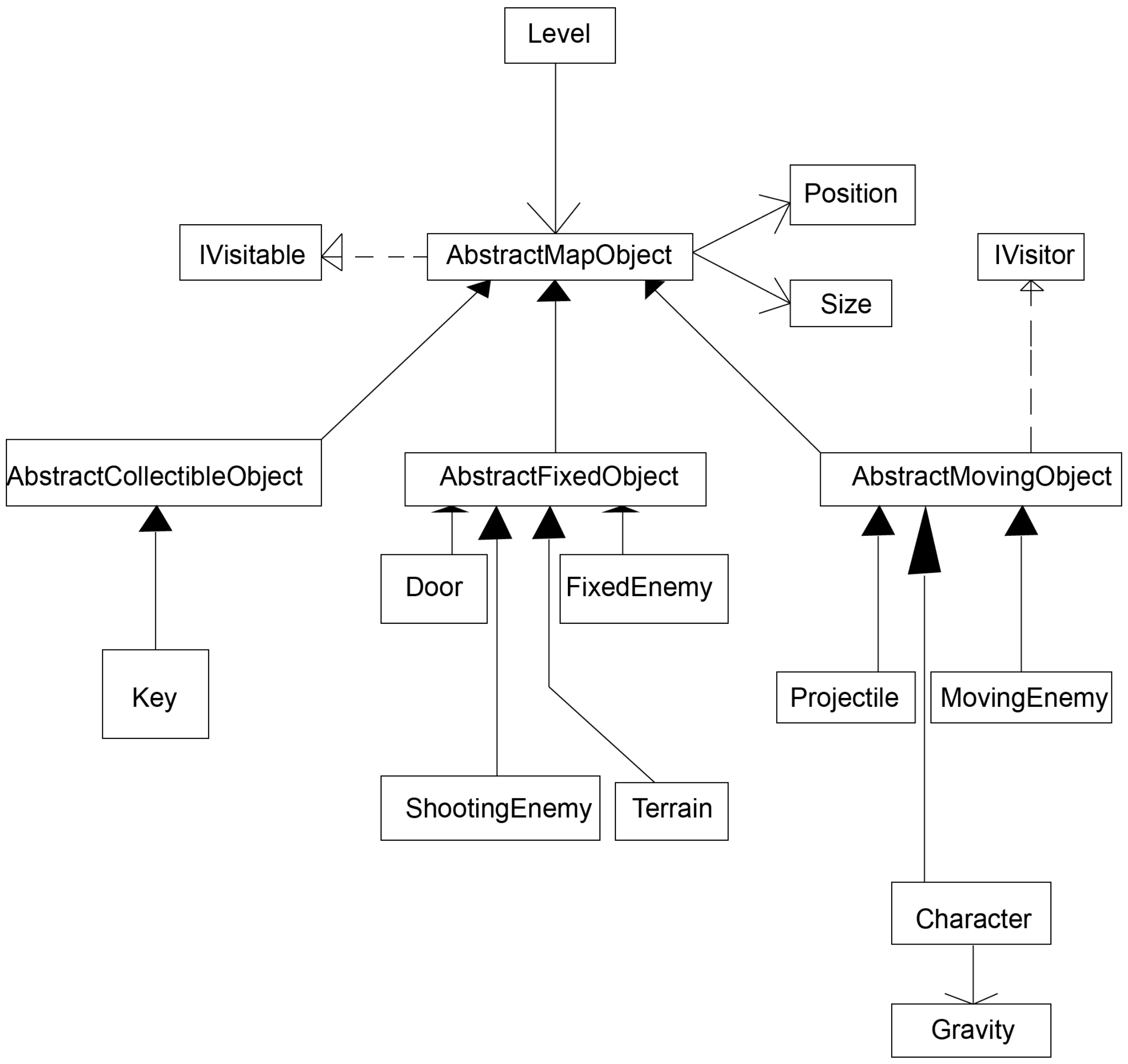

Figure 2. Class diagram for model package

Model classes and interfaces

- AbstractCollectibleObject, class for all collectibles in the game, currently just key

- AbstractFixedObject, all fixed objects in the map, aren’t visitors
- AbstractMapObject, all interactive objects in the map, are all visitable
- AbstractMovingObject, all moving objects, are all visitors
- Character, the game character, is controlled by user input
- Door, marks the end of the level. Can either be opened or closed
- FixedEnemy, non-movable deadly object
- IVisitor, interface for objects that affect themselves on collision
- IVisitable, interface for objects that are collidible
- Key, the only object currently collectible. Player needs key to be picked up to finish current level
- Level, contains all AbstractMapObjects, mainly to keep track of collisions
- MovingEnemy, a moving deadly object
- Position, a simple class for x and y positions
- Projectile, shootable projectiles that kills character
- ShootingEmeny, a fixed enemy that fires projectiles
- Size, a simple class for height and width of an object
- Terrain, the blocking terrain in the map, ground, platforms and so on

in model.menu

- AbstractMenuItem, all menu objects
- AbstractMenuButton, clickable menu item
- MenuItem, non-interactive menu object
- AudioButton, AbstractMenuButton to toggle sound
- StartButton, starts the game
- QuitButton, exits the game
- Menu, holds list of all menu items