TDA367

System design document for

‘Tis but a Scratch

André Samuelsson
Alma Ottedag
Anna Nylander
Ivar Josefsson

Table of Content

Version: 1.0
Date
Author
1 Introduction
1.1 Design goals
1.2 Definitions, acronyms and abbreviations
2 System design
2.1 Overview
2.2 Software decomposition
2.2.1 General
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

Version: 1.0

Date: ¾ -14

Author: Alma Ottedag, Ivar Josefsson

This version overrides all previous versions.

1 Introduction

1.1 Design goals

The goal is to have a loosely coupled, easily extended application. The model and controller should be minimally coupled to the graphical library, Slick2D. This will ensure the library is easily swapped.

1.2 Definitions, acronyms and abbreviations

The specific terms regarding the core “Tis but a scratch” game can be found in the references section.

- GUY - not a female
- Java - platform independent programming language.
- JRE - Java Runtime Enviroment, software needed to run Java applications
- Host - a computer that runs and controls the internal model of the game
- Frame - Rendering of one gamestate onto the screen, is done at approximately a rate of sixty per second.
- Update - One update of the internal gamemodel

2 System design

2.1 Overview

An MVC model will be used when implementing the application.

2.2 Software decomposition

2.2.1 General

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

The in-game application consists for four packages, the model, the controller, the view and the construction. Packages such as the model, view and controller adapt the MVC[1] model. Typically the main menu of the game will be separate from the in-game module and have a separate MVC counterpart. The construction package contains all classes and interfaces for creating game maps. It will have a separate class for creating maps using the Slick2 library, such that it can be swapped with any library as needed. See the appendix for a full view of the packages.

1

2.2.2 Decomposition into subsystems

Several subsystems will be used. Creating maps will be done through a seperate program called Tiled Map Editor. Graphics will be displayed using the graphics library Slick2d. For networking the kryonet library will be used.

2.2.3 Layering

See the UML-diagram in the appendix.

2.2.4 Dependency analysis

TODO: add dependency diagram.

2.3 Concurrency issues

The game will have to use multiple threads when multiplayer is implemented. This might raise some issues, however it is at the time unclear how extensive these issues will be. There will be one thread apart from the main thread which listens for new players. The concurrency issues which could arise would occur when the data is transferred to the main thread.

2.4 Persistent data management

The application does not save any in-game data to be loaded later. However the NPC-s including enemies will most likely be read from a file in later iterations for ease concerning especially creating entirely new enemy types.

2.5 Access control and security

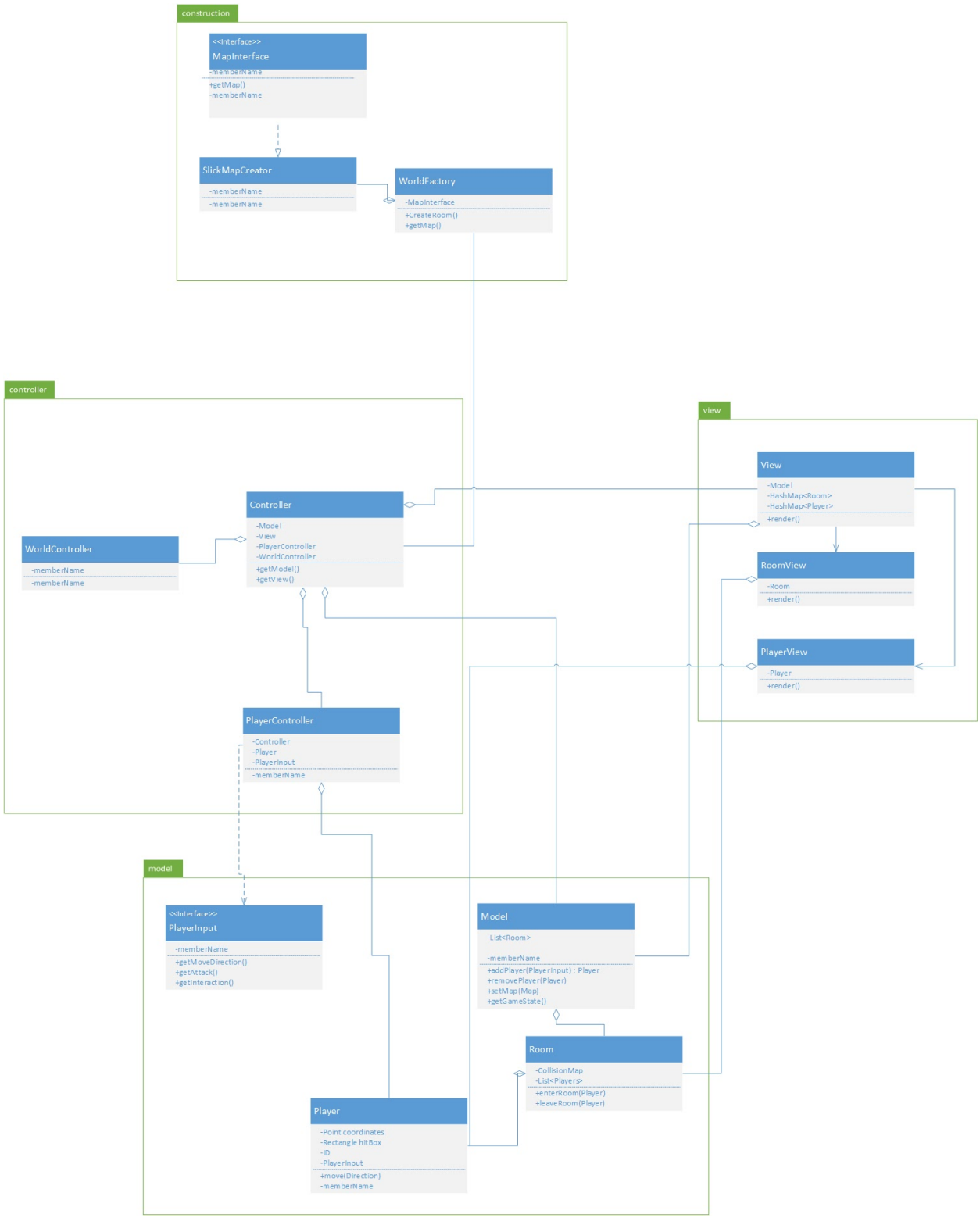
NA

2.6 Boundary conditions

NA - Game will be launched and terminated like a normal application

3 References

APPENDIX


construction
<<Interface>>
MapInterface
-memberName
+getMap()
-memberName
SlickMapCreator
-memberName
-memberName
WorldFactory
-MapInterface
+CreateRoom()
+getMap()
controller
WorldController
-memberName
-memberName
Controller
-Model
-View
-PlayerController
-WorldController
+getModel()
+getView()
PlayerController
-Controller
-Player
-PlayerInput
-memberName
view
View
-Model
-HashMap<Room>
-HashMap<Player>
+render()
RoomView
-Room
+render()
PlayerView
-Player
+render()
model
<<interface>>
PlayerInput
-memberName
+getMoveDirection()
+getAttack()
+getInteraction()
Model
-List<Room>
-memberName
+addPlayer(PlayerInput) : Player
+removePlayer(Player)
+setMap(Map)
+getGameState()
Room
-CollisionMap
-List<Players>
+enterRoom(Player)
+leaveRoom(Player)
Player
-Point coordinates
-Rectangle hitBox
-ID
-PlayerInput
+move(Direction)
-memberName

The uml diagram including packages.