# System design document for Robots Stole My Girlfriend (RSMG)

Table of Contents

**Version:** 1.1
**Date:** 2012-05-09
**Group:** 8
**Author:** Daniel Jonsson, Johan Grönvall, Johan Rignäs

This version overrides all previous versions.

# 1 Introduction

## 1.1 Design goals

The design is aimed to be testable and well structured. To detect errors we are going to use a lot of junit tests, especially for the model part. The Model is to be completely ignorant of the controller and GUI, so that it can be independently tested.

## 1.2 Definitions, acronyms and abbreviations

- RSMG: Project name, Robot Stole My Girlfriend
- Platformer: a game where the goal is to get from point A to point B by moving from platform to platform.
- GUI: Graphical user interface
- Java, platform independent programming language.
- Upgrade Point: a form of in-game currency used to purchase upgrades for a character
- MVC, a way to partition an application with a GUI into distinct parts avoiding a mixture of GUI-code, application code and data spread all over.

# 2 System design

## 2.1 Overview

The design will follow the MVC model.

### 2.1.1 Model functionality

The model is split into two major parts,Tiles and Interactive objects. Tiles are static and cannot be moved, they are used to describe the environment. Interactive objects are objects that can interact with the player in some way, these objects can be moved around and are mutable, they are used to describe enemies, items and the main character.

### 2.1.2 Graphics

We use the graphics library Slick, available at http://slick.cokeandcode.com/. Slick provides us with classes to easily create a game window, play audio, draw images and animations within the window and it also also has a easy way of creating and handling multiply states/views within the application.

Due to Slick's structure there is a really heavy connection between the controller and the view, and in our project those parts are actually in the same package.

### 2.1.3 Subsystem

See section 2.4

### 2.1.4 Event handling / Updating

Slick is structured in such a way that there is always a running loop that calls the update and render methods in the current state. Therefore does the application not use events, instead it checks in the render method for the current state if any relevant keys has been pressed (this is also true for menus and the level-selection-screen). This key check is run every loop.

As for the level state, where the model is used and drawn on the screen, Slick's loop will first call update in the view/controller. The update method will call update in the model, which then will move objects and do all calculations. Then will Slick's loop call the render method in the level state, which draws all objects from the model on the screen.

## 2.2 Software decomposition

### 2.2.1 General

The project is decomposed into the following modules:

- Main, the projects entry class
- Controller, handle user actions from GUI
- GUI, receives events from the user
- Model, store variables
- I/O, store saved data

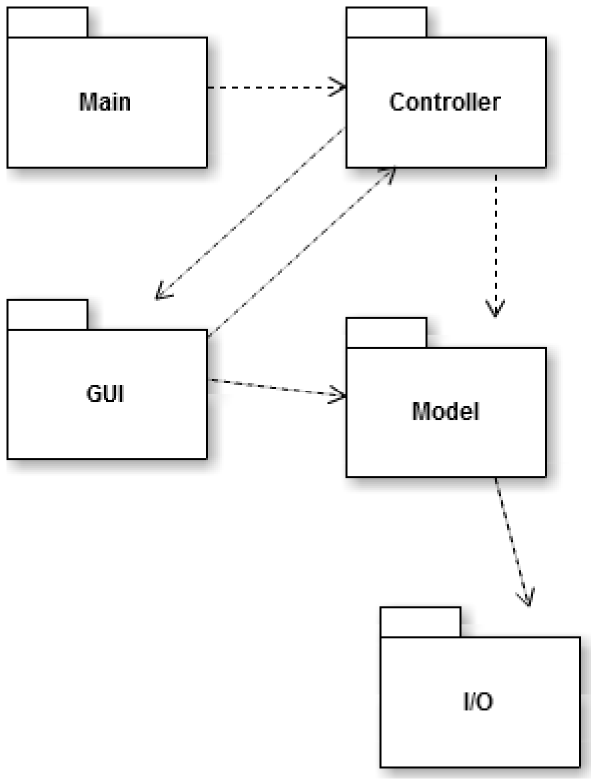

**Figure 1:** Package diagram

Seeing as how our application will be updating at a constant frequency, our controller and view are going to share a strong connection.
See appendix for UML class diagram.

### 2.2.2 Layering

### 2.2.3 Dependency analysis

We have avoided cyclic dependencies by putting both all interactive objects and weapons in the *model.object.unit* packet. This because each Wepaon will have a wielder and each of the LivingObjects will have a weapon.

## 2.3 Concurrency issues

NA. There is only one thread in the application. This is the AppGameContainer from Slick's library that starts the application and keeps it alive. Thereby we have no concurrency issues.

## 2.4 Persistent data management

Game configuration, game progress and levels are stored and read from XML files. To make this easier, we are using the library JDom. The levels can easily be created with our map editor. This map editor allows you to draw levels with our tiles and add in-game objects such as items and enemies. The map editor can be found on github, https://github.com/MaTachi/2D-Map-Editor. To make it even easier to add a level to the game, we have done it so that you only need to name the level file correctly and put it in the correct folder to make it appear in-game. No recompiling needed.

## 2.5 Access control and security

NA

## 2.6 Boundary conditions

NA

# 3 References

# APPENDIX

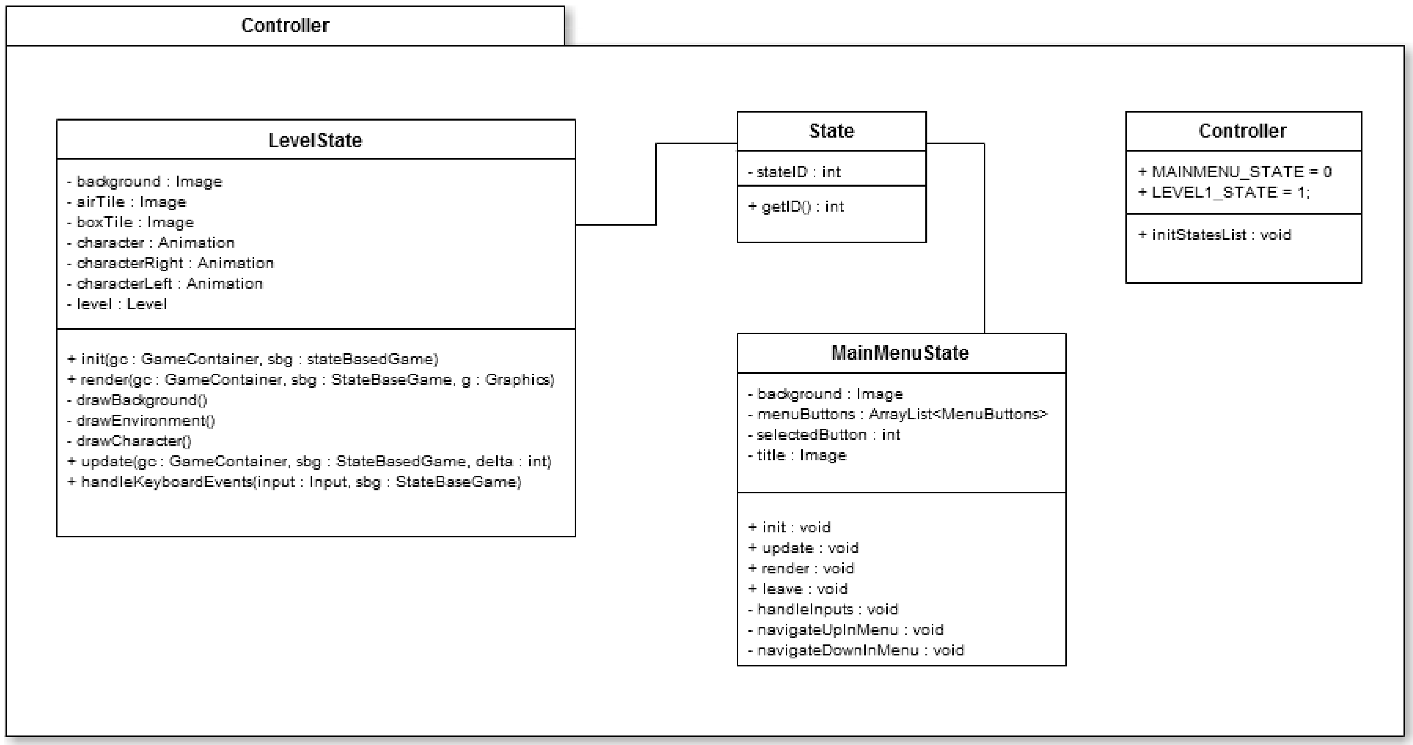

**Figure X:** Controller UML class diagram

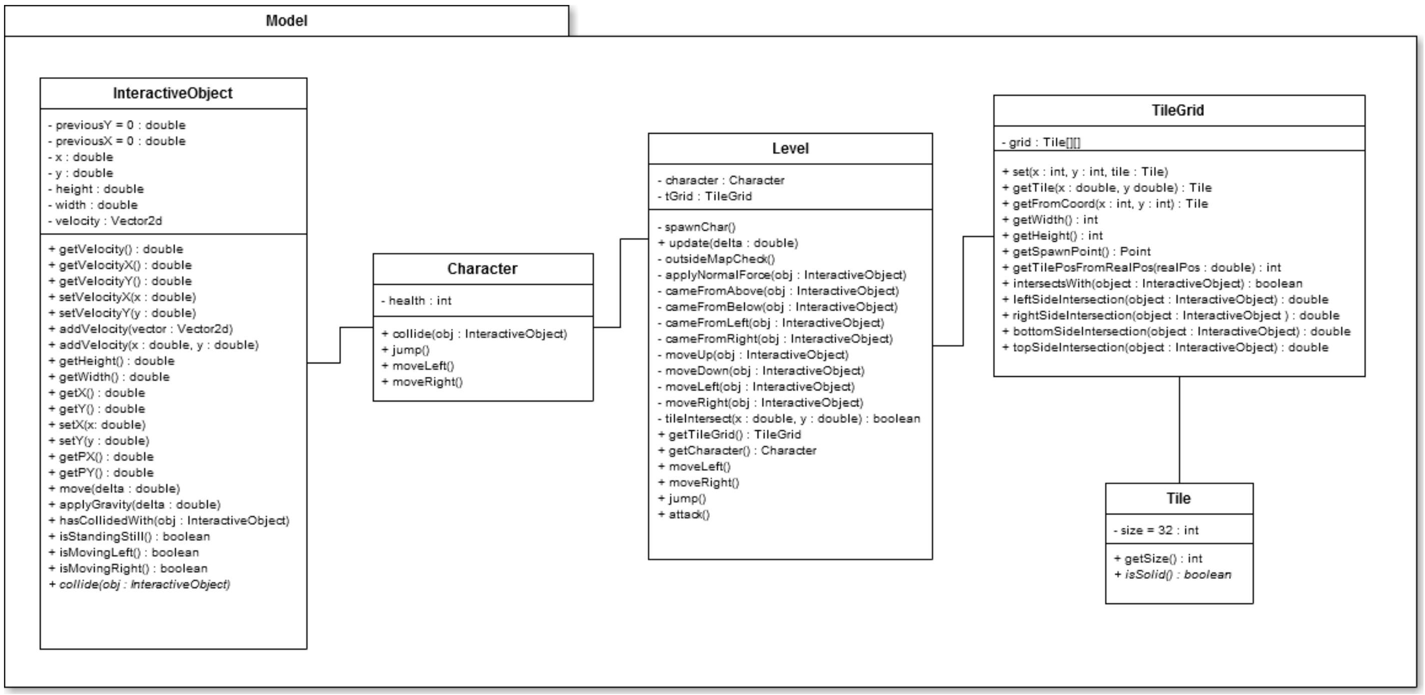
Model
InteractiveObject
- previousY = 0 : double
- previousX = 0 : double
- x : double
- y : double
- height : double
- width : double
- velocity : Vector2d
+ getVelocity() : double
+ getVelocityX() : double
+ getVelocityY() : double
+ setVelocityX(x : double)
+ setVelocityY(y : double)
+ addVelocity(vector : Vector2d)
+ addVelocity(x : double, y : double)
+ getHeight() : double
+ getWidth() : double
+ getX() : double
+ getY() : double
+ setX(x: double)
+ setY(y : double)
+ getPX() : double
+ getPY() : double
+ move(delta : double)
+ applyGravity(delta : double)
+ hasCollidedWith(obj : InteractiveObject)
+ isStandingStill() : boolean
+ isMovingLeft() : boolean
+ isMovingRight() : boolean
+ *collide(obj : InteractiveObject)*
Character
- health : int
+ collide(obj : InteractiveObject)
+ jump()
+ moveLeft()
+ moveRight()
Level
- character : Character
- tGrid : TileGrid
- spawnChar()
+ update(delta : double)
- outsideMapCheck()
- applyNormalForce(obj : InteractiveObject)
- cameFromAbove(obj : InteractiveObject)
- cameFromBelow(obj : InteractiveObject)
- cameFromLeft(obj : InteractiveObject)
- cameFromRight(obj : InteractiveObject)
- moveUp(obj : InteractiveObject)
- moveDown(obj : InteractiveObject)
- moveLeft(obj : InteractiveObject)
- moveRight(obj : InteractiveObject)
- tileIntersect(x : double, y : double) : boolean
+ getTileGrid() : TileGrid
+ getCharacter() : Character
+ moveLeft()
+ moveRight()
+ jump()
+ attack()
TileGrid
- grid : Tile[][]
+ set(x : int, y : int, tile : Tile)
+ getTile(x : double, y double) : Tile
+ getFromCoord(x : int, y : int) : Tile
+ getWidth() : int
+ getHeight() : int
+ getSpawnPoint() : Point
+ getTilePosFromRealPos(realPos : double) : int
+ intersectsWith(object : InteractiveObject) : boolean
+ leftSideIntersection(object : InteractiveObject) : double
+ rightSideIntersection(object : InteractiveObject ) : double
+ bottomSideIntersection(object : InteractiveObject) : double
+ topSideIntersection(object : InteractiveObject) : double
Tile
- size = 32 : int
+ getSize() : int
+ *isSolid() : boolean*

**Figure X:** Model UML class diagram

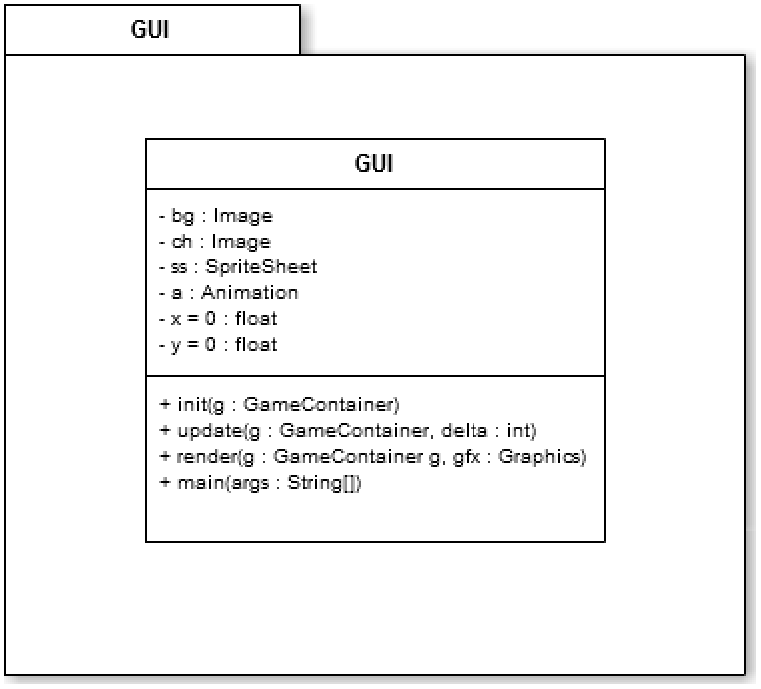

**Figure X:** GUI UML class diagram

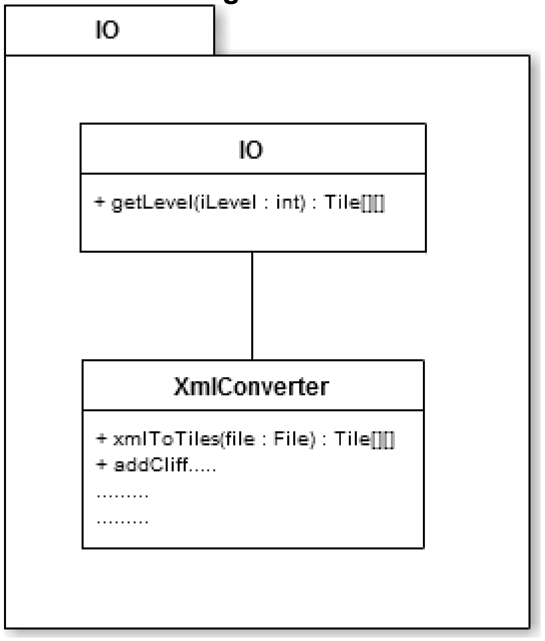
IO
IO
+ getLevel(iLevel : int) : Tile[][]
XmlConverter
+ xmlToTiles(file : File) : Tile[][]
+ addCliff.....
.........
.........

**Figure X:** IO UML class diagram