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

We have an io package that is responsible for all reading and writing to the hard drive. In this package can the classes CharacterProgress, Config and Levels be found. All these classes stores its data in XML documents that jDom interprets for us.

CharacterProgress stores what the player has unlocked, how many levels he has completed, number upgrade points and found weapons. This data is saved in a single file, and it can easily be reset to the default values, making it easy to start over in the game.

Config stores game configuration. That is if the game should be played in fullscreen, and if music and sound effects should be turned on. However, to make the fullscreen setting take effect the game has to be restarted.

Levels provides methods related to the level files. It can give a list with all available level numbers, tell if a level is a boss level and it can return a level as a jDom Document. The levels are a bit special because they aren't stored in a single file, in constrast to the character progress and the configuration. We have made it so you can easily add a new level to the game by using our map editor and saving it with the name LevelX.xml in the folder "res/data/level/". The jDom Document is interpreted by our LevelFactory class.

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.

Changing between is also handled by Slick. Slick has something called a *AppGameContainer*, which runs the game (a *StateBasedGame*) within itself. All states extends Slick's class *BasicGameState*. To every update and render method is a reference to the *StateBasedGame* sent. With a reference to the game you can easily call its method *enter()* that takes another state's ID as parameter.

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 and read data from hard drive

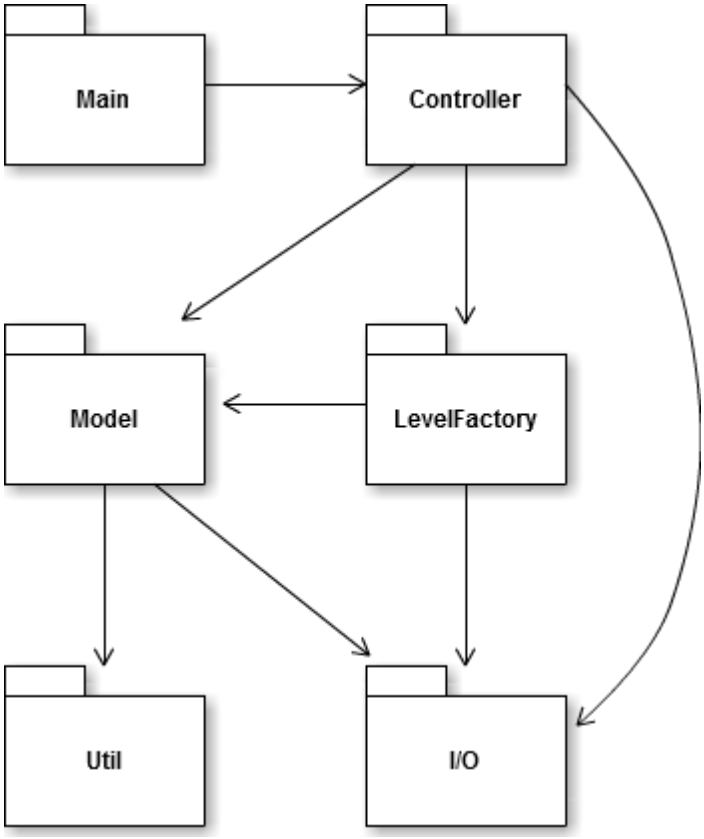

Figure 1a: Package diagram

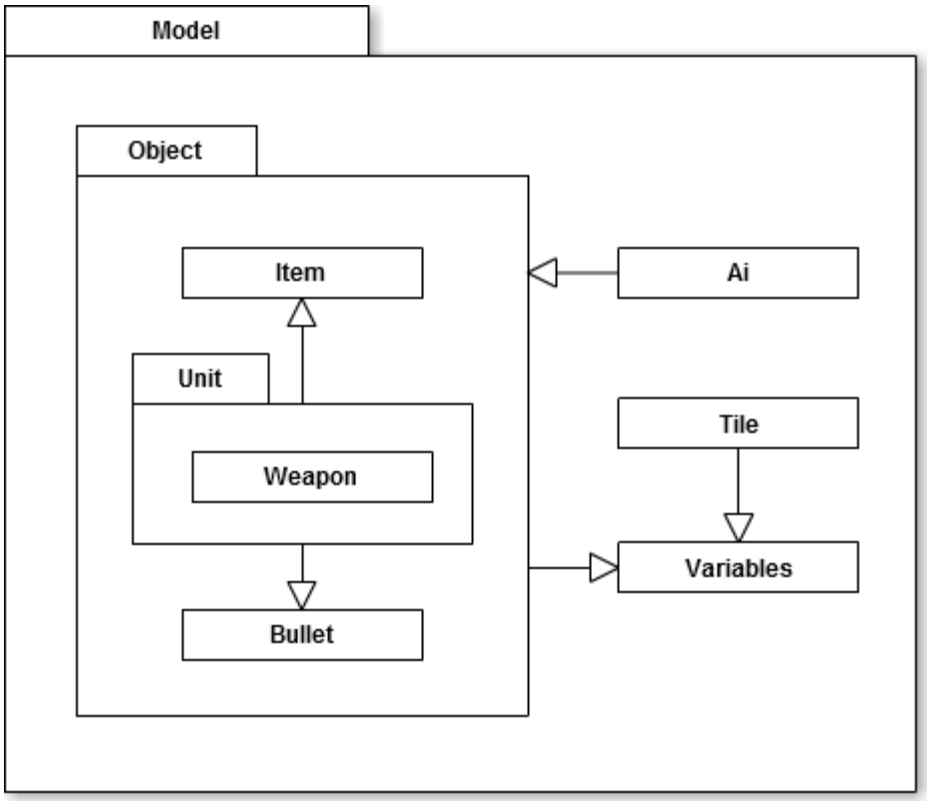

Figure 1b: Model package diagram

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. We did this becuase every weapon 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.

3 References

APPENDIX

Figure X: Controller UML class diagram

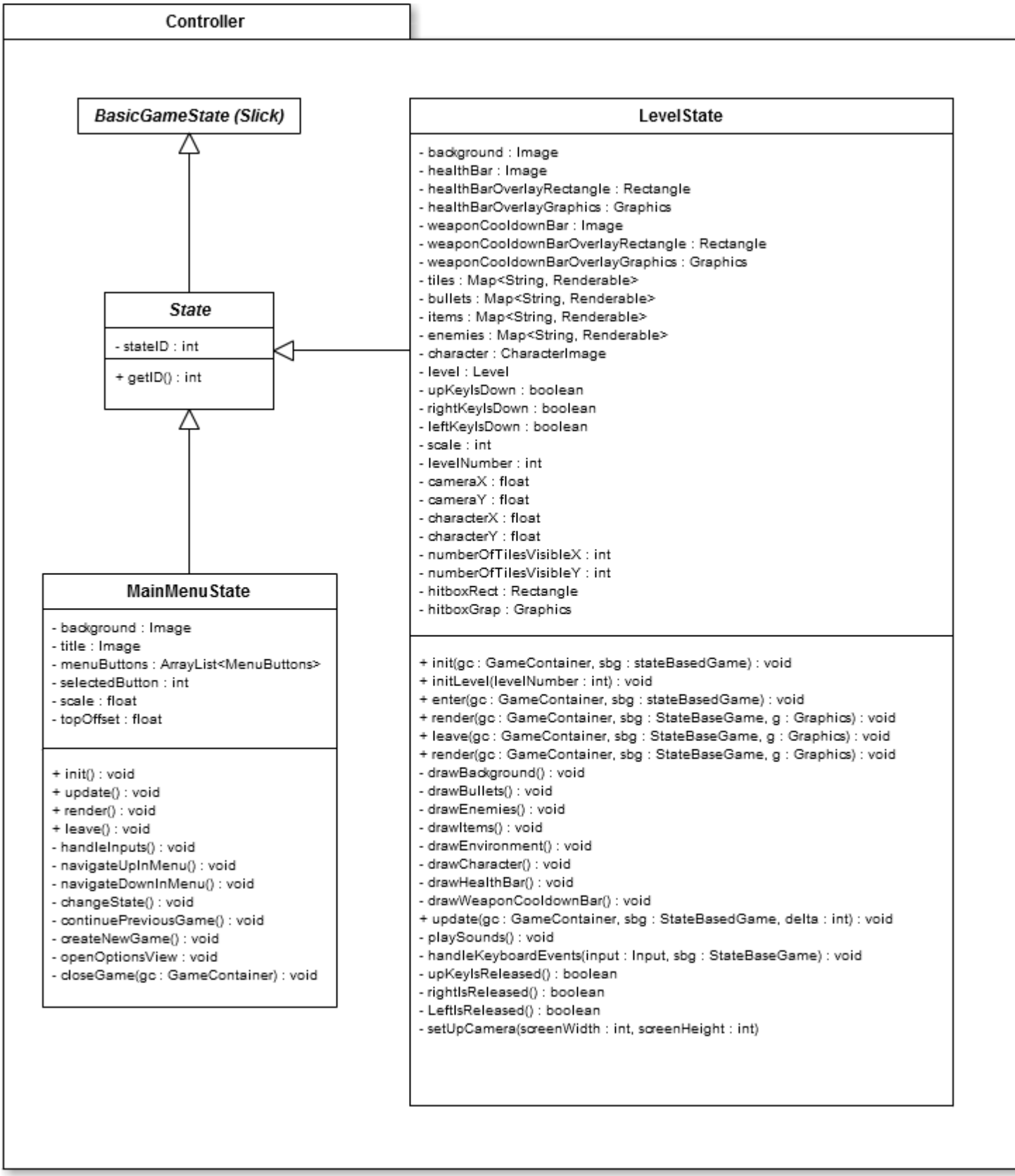
Controller
BasicGameState (Slick)
State
- stateID : int
+ getID() : int
LevelState
- background : Image
- healthBar : Image
- healthBarOverlayRectangle : Rectangle
- healthBarOverlayGraphics : Graphics
- weaponCooldownBar : Image
- weaponCooldownBarOverlayRectangle : Rectangle
- weaponCooldownBarOverlayGraphics : Graphics
- tiles : Map<String, Renderable>
- bullets : Map<String, Renderable>
- items : Map<String, Renderable>
- enemies : Map<String, Renderable>
- character : CharacterImage
- level : Level
- upKeyIsDown : boolean
- rightKeyIsDown : boolean
- leftKeyIsDown : boolean
- scale : int
- levelNumber : int
- cameraX : float
- cameraY : float
- characterX : float
- characterY : float
- numberOfTilesVisibleX : int
- numberOfTilesVisibleY : int
- hitboxRect : Rectangle
- hitboxGrap : Graphics
+ init(gc : GameContainer, sbg : stateBasedGame) : void
+ initLevel(levelNumber : int) : void
+ enter(gc : GameContainer, sbg : stateBasedGame) : void
+ render(gc : GameContainer, sbg : StateBaseGame, g : Graphics) : void
+ leave(gc : GameContainer, sbg : StateBaseGame, g : Graphics) : void
+ render(gc : GameContainer, sbg : StateBaseGame, g : Graphics) : void
- drawBackground() : void
- drawBullets() : void
- drawEnemies() : void
- drawItems() : void
- drawEnvironment() : void
- drawCharacter() : void
- drawHealthBar() : void
- drawWeaponCooldownBar() : void
+ update(gc : GameContainer, sbg : StateBasedGame, delta : int) : void
- playSounds() : void
- handleKeyboardEvents(input : Input, sbg : StateBaseGame) : void
- upKeyIsReleased() : boolean
- rightIsReleased() : boolean
- LeftIsReleased() : boolean
- setUpCamera(screenWidth : int, screenHeight : int)
MainMenuState
- background : Image
- title : Image
- menuButtons : ArrayList<MenuButtons>
- selectedButton : int
- scale : float
- topOffset : float
+ init() : void
+ update() : void
+ render() : void
+ leave() : void
- handleInputs() : void
- navigateUpInMenu() : void
- navigateDownInMenu() : void
- changeState() : void
- continuePreviousGame() : void
- createNewGame() : void
- openOptionsView : void
- closeGame(gc : GameContainer) : void

Figure X: IO UML class diagram

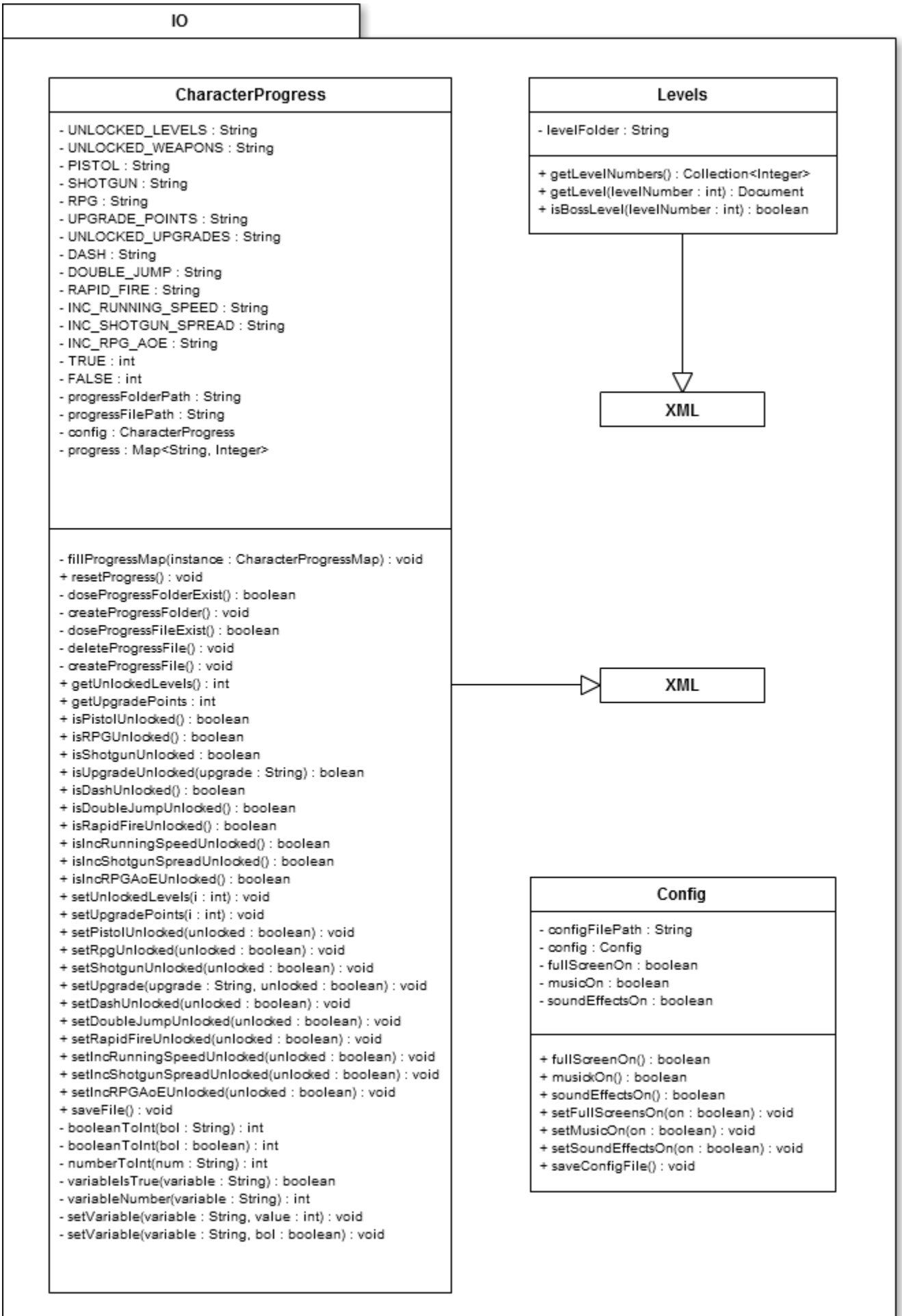
IO
CharacterProgress
- UNLOCKED_LEVELS : String
- UNLOCKED_WEAPONS : String
- PISTOL : String
- SHOTGUN : String
- RPG : String
- UPGRADE_POINTS : String
- UNLOCKED_UPGRADES : String
- DASH : String
- DOUBLE_JUMP : String
- RAPID_FIRE : String
- INC_RUNNING_SPEED : String
- INC_SHOTGUN_SPREAD : String
- INC_RPG_AOE : String
- TRUE : int
- FALSE : int
- progressFolderPath : String
- progressFilePath : String
- config : CharacterProgress
- progress : Map<String, Integer>
- fillProgressMap(instance : CharacterProgressMap) : void
+ resetProgress() : void
- doseProgressFolderExist() : boolean
- createProgressFolder() : void
- doseProgressFileExist() : boolean
- deleteProgressFile() : void
- createProgressFile() : void
+ getUnlockedLevels() : int
+ getUpgradePoints : int
+ isPistolUnlocked() : boolean
+ isRPGUnlocked() : boolean
+ isShotgunUnlocked : boolean
+ isUpgradeUnlocked(upgrade : String) : bolean
+ isDashUnlocked() : boolean
+ isDoubleJumpUnlocked() : boolean
+ isRapidFireUnlocked() : boolean
+ isIncRunningSpeedUnlocked() : boolean
+ isIncShotgunSpreadUnlocked() : boolean
+ isIncRPGAoEUnlocked() : boolean
+ setUnlockedLevels(i : int) : void
+ setUpgradePoints(i : int) : void
+ setPistolUnlocked(unlocked : boolean) : void
+ setRpgUnlocked(unlocked : boolean) : void
+ setShotgunUnlocked(unlocked : boolean) : void
+ setUpgrade(upgrade : String, unlocked : boolean) : void
+ setDashUnlocked(unlocked : boolean) : void
+ setDoubleJumpUnlocked(unlocked : boolean) : void
+ setRapidFireUnlocked(unlocked : boolean) : void
+ setIncRunningSpeedUnlocked(unlocked : boolean) : void
+ setIncShotgunSpreadUnlocked(unlocked : boolean) : void
+ setIncRPGAoEUnlocked(unlocked : boolean) : void
+ saveFile() : void
- booleanToInt(bol : String) : int
- booleanToInt(bol : boolean) : int
- numberToInt(num : String) : int
- variableIsTrue(variable : String) : boolean
- variableNumber(variable : String) : int
- setVariable(variable : String, value : int) : void
- setVariable(variable : String, bol : boolean) : void
Levels
- levelFolder : String
+ getLevelNumbers() : Collection<Integer>
+ getLevel(levelNumber : int) : Document
+ isBossLevel(levelNumber : int) : boolean
XML
XML
Config
- configFilePath : String
- config : Config
- fullScreenOn : boolean
- musicOn : boolean
- soundEffectsOn : boolean
+ fullScreenOn() : boolean
+ musickOn() : boolean
+ soundEffectsOn() : boolean
+ setFullScreensOn(on : boolean) : void
+ setMusicOn(on : boolean) : void
+ setSoundEffectsOn(on : boolean) : void
+ saveConfigFile() : void

Figure X: Model UML class diagram