System design document for Life of a Goblin

Version: 2.0

Date: 2015-05-31

Author: Anton Strandman, Fredrik Bengtsson, Jesper Olsson, Ulrika Uddeborg

This version overrides all previous versions.

1 Introduction

Life of a Goblin (referred to as the 'system') will be a standalone side-scroller computer game with 3-D graphics intended for laptops and desktop computers. The system is intended for entertainment purposes only. The system will be an OS independent application written in Java. The potential users are less experienced to semi-experienced gamers.

1.1 Design goals

The system will be built in Java, using tools provided by the game engine jMonkeyEngine 3 (jME3). A "Skinny controller, fat model" type of Model-View-Control (MVC) will be used in the implementation, i.e. the logic will be located in the model. The model will be decoupled from the jME3 framework (see Image 1).

The jME3 game engine will provide the (MVC) view. The movement of the visual representation of the model will be managed outside of the model by the jME3's built-in physics engine. jME3 will also provide some parts of the controller, such as an input manager for handling user input.

The jME3 objects will be encapsulated from the model, as described in Image 1, in order to make the system logic as engine independent as possible. Purpose-built interfaces will be used in the model to connect the jME3 objects to the model, through polymorphism.

The classes in the model will implement interfaces and extend abstract classes to detail its functionality. The concrete classes will not contain any functionality not provided by interfaces or abstract classes that it implements or extends . The benefits of this design is that two different objects, such as a checkpoint and a boss may both implement the same interfaces for ending a game level, or a specific minion may have the same functionality as a spawnpoint. These abilities may be added without adding any new functionality apart from the model itself.

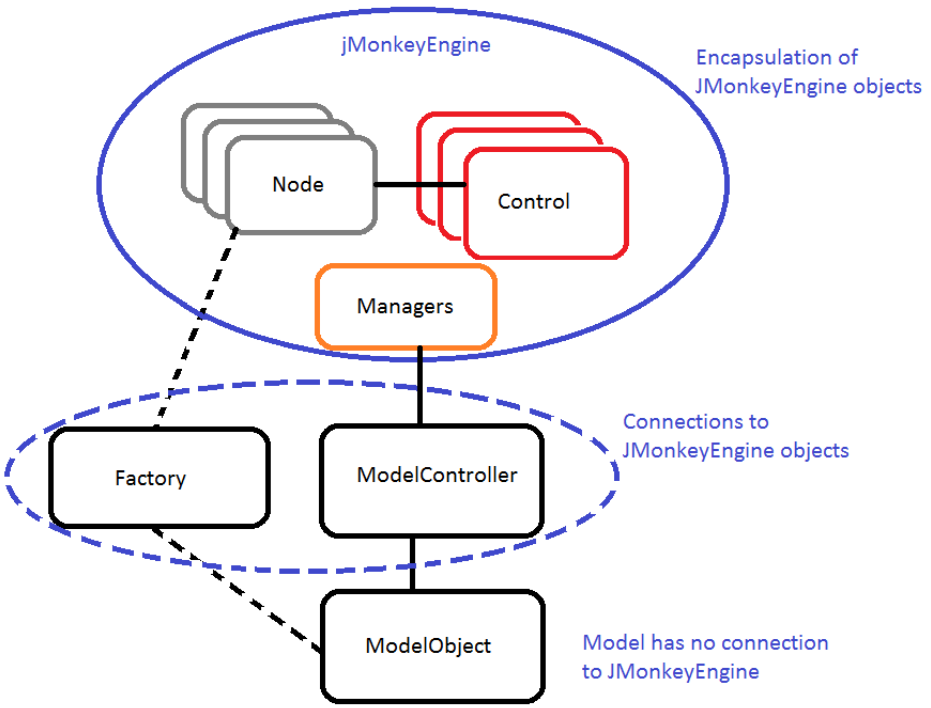

Image 1. High level system design.

The system will be designed according to the Open-Closed Principle (OCP). The system should be fully runnable on Windows, Mac and Linux. The system should contain multiple game levels and up to four playable characters. The system should be able to locally save settings and game progress. The system should provide options for changing settings and managing saved games.

1.2 Definitions, acronyms and abbreviations

Checkpoint	A point in the game where progress is saved.
GUI	Graphical User Interface.
HUD	The part of the GUI displaying player status and possibly information about the level.
(Game) level	A course which the player will have to traverse.
NPC	Non playable (computer controlled) characters.
Player	The character directly controlled by the user and the main focus of the game.

Side-scroller	A 2D game in which the player traverse a linear level, typically from left to right.

2 External tools

This section describes which external tools have been used in this project and what the tools are used for.

2.1 jMonkeyEngine 3

jMonkeyEngine 3 (jME3) is an open-source game engine for Java

2.1.1 jMonkeyEngine SDK

The jMonkeyEngine SDK is a IDE with the jMonkeyEngine 3 libraries built in and with additional tools for game development. Scene Composer is one of the tools available in the jMonkeyEngine SDK.

2.1.2 Scene Composer

Scene Composer has been used to create the visual representation of the game levels. The Scene Composer tool is not compatible with the Maven project structure and any scene file will have to be created in a separate jME3 project. Any new or modified scene files will have to be manually imported into the Maven project by hand.

As scenes will be created outside of the main project it will not be able to access any of the classes or methods in the game, thus objects will have to be referenced through the userData key “nodeType”. The factories in the main project will translate this nodeType data into the corresponding game object at runtime.

2.2 Apache Maven

Apache Maven is a system used for managing Java libraries for the project and to build jar files.

2.3 Git

Git is a revision control system. The repository of this project can be found at https://github.com/Mrmiffo/LifeOfAGoblin.

2.4 STAN

STAN is a system used for analysing package structures.

2.5 FindBugs

FindBugs is an open-source system used for finding bugs in Java code.

2.6 JUnit

JUnit is a framework used for creating tests in Java.

3 System design

This section will describe the design of the system.

3.1 Overview

The system will be a standalone desktop application, which will be fully runnable on Windows, Mac and Linux. The system will be distributed via a downloadable jar file. An Internet connection will not be required for neither installing nor running the system. The system is intended to run in only one instance, but there will be no limits as to how many instances may be run at the same time.

The system will take user input only via standard PC input devices (keyboard and mouse). If the input is linked to an action (hotkey), the action will be performed if possible. The user will get feedback through sound and GUI.

The model will not be completely decoupled from the engine as jME3 will manage the movement of objects in the system through the jME3 built-in physics engine.

3.2 Software decomposition

The system will be composed of four distinct components: the model, the view, the controller and various utility classes. The model will contain the system data and logic, the view will display the visual representation of the model, and the controllers will connect the view to the model. The utility classes provide subsystems, including helper classes and converters.

3.2.1 General

This section describes the overall package structure of the system. Classes and interfaces which are extended or implemented by classes in two or more packages are located in the package one level above those packages.

In order to decouple the model from the jME3 framework, all classes using jME3 code related classes are contained in the jME3 package or the utils package. These classes may retrieve information from the model and call methods in the model. The model will never call the jME3 package (see Image 2), which contains the view and controller (see Image 3).

The utils package (see Image 2) is used to provide subsystems, including but not limited to helper classes and classes for converting model data to jME3 data.

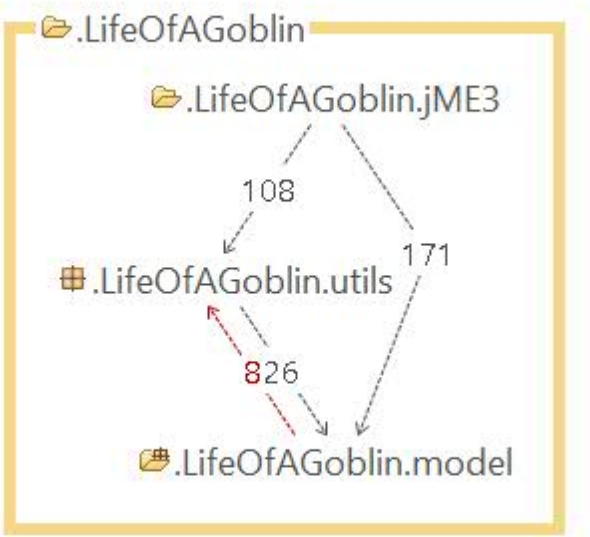

Image 2. The top-level packages

The jME3 package contains all the classes which extend or implement jME3 classes and interfaces. This package contains the game (contains main, SimpleApplication class and AppStates), view, controllers and factories (see Image 3).

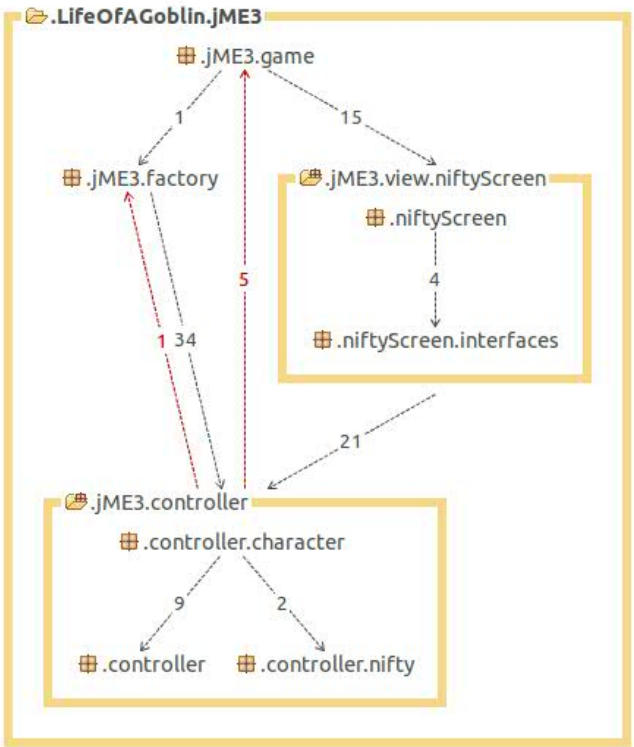

Image 3. The jME3 package

The model contains three distinct parts: characters, objects and profiles (see Image 4). These classes provide the data and logic for the system.

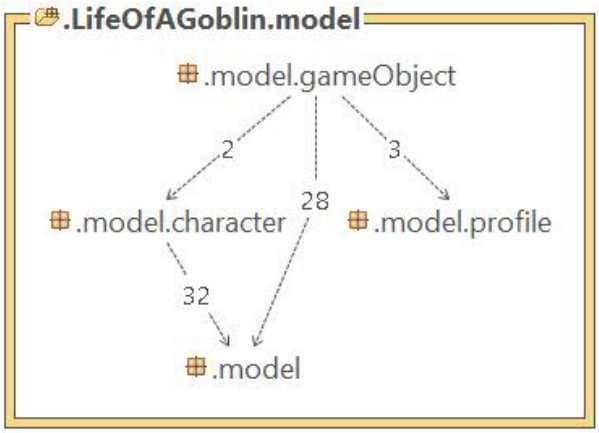

Image 4. The model package

3.2.2 Decomposition into subsystems

The classes in the jME3 package are considered jME3 objects as they implement or extend interfaces and classes from the jME3 framework. The jME3 package contains the jME3 controls and the GUI components, as well as factories for creating and painting objects to display in the game world.

The controls are attached to the jME3 node objects and use the custom ModelControl, which contains an instance of the model that the node represents, to access and modify the system model objects.

The classes in the utils package may contain jME3 code and objects, but will never implement nor extend jME3 classes and interfaces, and are therefore not considered jME3 objects.

jME3 provides an input manager class, which will be used as is through a wrapper class.The input-command mapping will however be managed by the system model.

3.2.3 Layering

This section details how the model will be decoupled from the jME3 framework.

The concrete classes of the model will extend abstract classes and implement interfaces to detail its functionality. No concrete class will contain any functionality not provided by the interfaces or abstract classes it extends and implements.

The factory will use abstraction levels of the model to determine the functionality available to a jME3 node.

The jME3 controls will use abstraction levels of the model to determine the actions to use in the model.

Wrapper classes will be used to extract data from the model and provide it to engine in a jME3 friendly way.

3.2.4 Dependency analysis

This sections will detail the dependency issue indicated as red arrows in Image 2-4.

The model uses some of the helper classes in the utils package to store data and activate hotkeys(see Image 2). These solutions are acceptable, from a decoupled model point-of-view, as they do not depend on either the control or view.

The jME3 MainMenuControl uses methods in the game package to shut down the application and to start a game, as this is the only part of the system which has access to the system instance.

The SpawnControl uses the NodeFactory to create jME3 nodes for the spawnable objects to add to the game world (see Image 3).

These were the most logical solutions to be found with the decided design solution.

3.3 Concurrency issues

Threading issues need to be taken into consideration when managing app states and jME3 controls. The jME3 framework is not completely thread safe, which may cause concurrency issues when using out of the box solutions. The model has not been created to utilize the synchronization functionality in java for thread safety. Synchronization could cause delay as a frame update might freeze while waiting to be processed. Instead other safeguards has been put in place, such as the player invulnerability timer, to reduce threading issues.

3.4 Persistent data management

The user progress and settings will be stored in the profile class which in turn will be saved automatically to the user hard drive. For this purpose the $home/LifeOfAGoblin/savedFiles folder will be created when the system is first launched.

3.5 Access control and security

No access control or security will be implemented in the system. The system will not store or require any personal information that require safekeeping.

4 Open issues

This section describes issues that are currently unresolved due to lack of time or which are not possible to resolve.

4.1 NiftyGUI

The construction of the NiftyGUI components is not optimal. NiftyGUI is built using XML code through a Java builder. This causes some parts of the code to be “dodgy”, according to FindBugs.

The HUD is currently static and is updated by a jME3 control, attached to the player node. This is not an ideal solution, instead the HUD should be maintained by the GameAppState.

4.2 Factories

The design goal of the factories is to have the factories only look for abstractions of the objects to decide how to create a jME3 node. This is currently not the case in all instances. The factories currently do not check for interfaces. This is a work in progress and a draft for connecting interfaces to jME3 controls has been created (see appendix 1)

4.2.1 Scene Composer

The creation of objects in Scene Composer is not optimal and the NodeIdentifier is hard coded to the strings from the nodes created in Scene Composer. Although with the current project design (Maven) this is the only way, that was found, to create game levels.

4.2.2 Excess collisions

GhostControl is added to some nodes which doesn’t need these. This results in duplicated collision events. This could potentially decrease performance.

4.3 Content

More content should be created. Only two game levels exist, with limited content in each. Models are missing textures and materials. The player character graphical model contains an invisible physical shape that cause bad framerates when starting the game, as it collides with the capsule shape of the character.

Pausing is possible, but does not display a menu.

Error messages are not displayed in the GUI, only in the command line.

Feedback for updating progress should be added.

4.4 Unit testing

Not all test cases have been written for all classes.

4.5 Profile

It is currently not possible to change the profile name from the Profile menu. The profile class provides this functionality, but it has not been added to the GUI.

It is currently not possible to start a game based on profile progress.

4.6 Input

Each instance contains anonymous inner classes for the default hotkeys which should be made static.

The SettingsMenu screen has a bug which is triggered when an action has two hotkeys and only one of them is modified. This causes both hotkeys to be replaced with only the new one. The root cause for this bug is in the settings menu controller and how it stores the hotkey before it is saved to the profile.

4.7 Checkpoint

When an NPC collide with a checkpoint bad framerates occur. This is most likely due to a faulty part of the collision listener implementation.

5 References

Appendix 1: Interface controller cheat sheet DRAFT