SYSTEM DESIGN DOCUMENT FOR LEGEND OF CHALMERS

Version: 2.0
Date: 2015-05-29
Author: Kevin Hoogendijk, Maxim Goretskyy, Alexander Håkansson, Alexander Karlsson

This version overrides all previous versions.

1 Introduction

1.1 Design goals

The goal is to have a very loosely coupled application where the model is free from any type of graphical framework. The application should be designed so that any module can be easily swapped and so it becomes easy to extend it with new features.

2 Definitions, acronyms and abbreviations

- Java - Multiplatform programming language
- libGDX - Framework for game development in Java.
- MVC - Model View Controller, design pattern to divide modules in three parts: model, view and controller.
- Passive MVC - Passive Model View Controller. The model is passive and does not notify the view when it has finished updating itself. Instead the view gets notification from controller and then the view gets the information from model and updates itself.
- Stan - Tool for structure analysis.
- Map - Representing the inner world of the game. Represents a coordinate system in this application.
- Minigame - A simple game inside the main application.
- Player - The character controlled by the human using the game
- Hec - Higher Education Credits
- LoC - Legends of Chalmers
- NPC - Non-Playable-Character
- Dialog - Conversation between two entities.
- Id - identifier
- Factory-design - A design pattern which allows safe creation of objects by setting all of the variables needed before creating the object
- Event/Listener/Observation design

3 System design

The application uses the graphical game framework libGDX for representing the model. This is done using a passive variant of the Model-View-Controller (MVC) design pattern.

3.1 Overview

The backend's functionality lies within GameModel class where all information needed for the game can be retrieved. GameModel is the top of the model package and has references to both the Player and the GameMap, where the GameMap in turn hold references to all NPCs and items. Both the view and the controller have references to the model.

LocMain is the head of the whole application and extends the libGDX class "Game" which is needed to start the application. Since LocMain is responsible for which libGDX screen is showing it also switches between the main game screen and minigame screens when minigames are started or ended.

3.1.1 Map

The map is simply a 2D coordinate system with tiles on it. Each coordinate has one tile. There are different types of tiles and the map has access to all of them. Standard tiles have nothing, ItemTiles have items on them and MinigameTiles trigger minigames when stepped on. The map is also divided into multiple layers where each layer has its own set of tiles. The purpose of the different layers is for separating i.e. tiles just for collision checking from the rest of the tiles. It also enables for easy switching layers "on" or "off" which is useful in the view for not displaying a certain layer.

3.1.2 NPC

NPC's are created using the NPCFactory class. There are three types of NPC's: StandardNPC, ItemNPC and MinigameNPC. The StandardNPC is a normal NPC with no extra features. The ItemNPC has an inventory which is given to the player when spoken to. The MinigameNPC can start a minigame when the player talks with it. LocMain uses the "NPCs.loc" file to read all the NPC's and their properties when created. LocMain then builds the NPC's by handing all that information to the NPCFactory. Inside the file you can specify the position, name, dialog and id number of an NPC. LocMain uses a utility for the actual loading of NPC's. MinigameNPC's must have the same id as its minigame to be associated with it.

3.1.3 Minigame

Each minigame has its own MVC design and can be used as a stand-alone application. The MinigameHandler class helps minigames notify the main game when the minigame is finished so the main game can set its own screen and controllers back. Different minigames are started by matching the id of the minigame with the id of a MinigameNPC at the end of the dialog or if the player steps on a MinigameTile.

3.2 Software decomposition

3.2.1 **General**

The application is divided into the following modules:

- controller - main game controller. Controller part in MVC.
- models - backend. Model part in MVC.
- view - main GUI for the main game. View part in MVC.

- minigame - minigames with its own MVC.
- utilities - service classes, file handling.
- LocMain - The head of the application, the libGDX "Game" needed to start the application.

UML diagrams representing the different modules in the application and how they fit together can be found in the appendix under references.

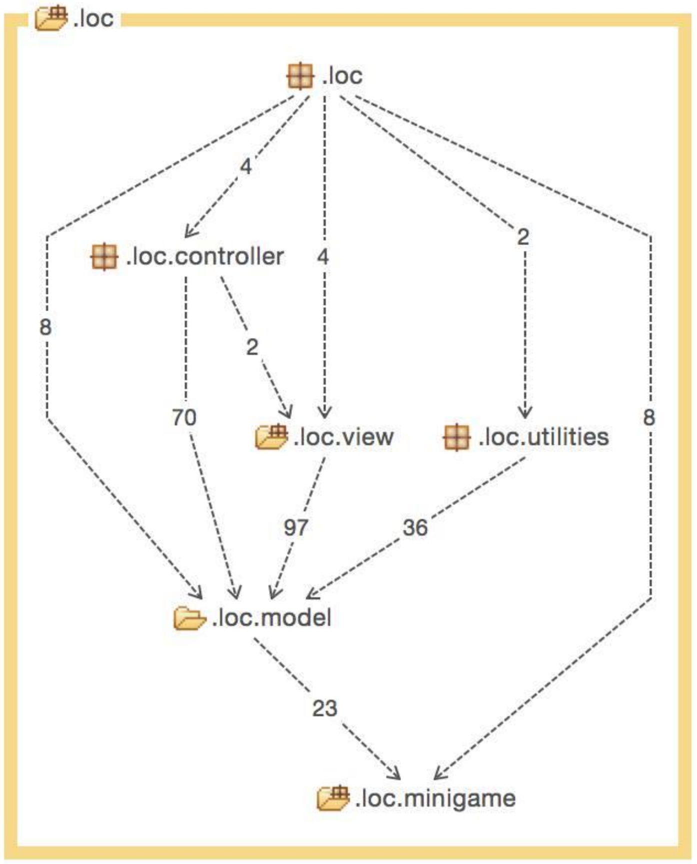
3.2.2 Decomposition into subsystems

Minigames have their own model, view and controller. These minigames are started by using a MinigameHandler class which relays the communication between the main game to the different minigames. This is also used to actually switch the view and controller between the minigames and the main game.

3.2.3 Layering

See the diagrams in the appendix.

3.2.4 Dependency analysis



AbstractItem has a reference to IGameModel because an item should be able to modify anything it wants inside the IGameModel. This could be seen as bad practice while it is an intended design choice of the application.

libGDX uses Viewport to display a portion of a screen, this forces us to use the libGDX methods associated with this thus creating violating dependencies.

3.3 Concurrency issues
The application should not have any concurrency issues since it will not use networking and all graphical rendering is handled by the libGDX framework. The application is intended for single-threaded use and is designed as such.

3.4 Persistent data management
NPC's, positions, ids and names will be specified in a "NPCs.loc" textfile. This allows for adding or removing NPC's without changing the source code of the application.

NPC's dialogs are specified in "Dialogs.loc". See Appendix for correct format. The dialogs will be loaded into the application when it starts.

3.5 Access control and security
There are no security issues since there is no account handling or login information involved in the application

3.6 Boundary conditions
N/A - Game will be launched and terminated like a normal application

4 References

1. Model-View-Controller see http://st-www.cs.illinois.edu/users/smarch/st- docs/mvc.html
2. libGDX see https://github.com/libgdx/libgdx/wiki

4.1 APPENDIX
4.2 File specification document for The Legend of Chalmers
4.2.1 Map
The map is created using Tiled Map Editor, the following layer names are used:

- ground - layer where the player walks and the items are located
- collision - layer used for collidable tiles where a player cannot walk
- building - layer used for buildings
- buildingRoof - layer that should be hidden when player is located beneath it
- groundDetail - layer used for graphical ground decoration

4.2.2 NPC
The NPCs are specified in the NPCs.loc file located in the Assets folder.

4.2.2.1 File format
Each row specifies one NPC and should have the following format:

```
id:name:GENDER:posX:posY:DIRECTION:ItemID:
```

4.2.2.2 ID

The ID's are used as unique identifiers for each NPC. The following ID's should be used for each type of NPC:

0-999 Standard NPC.

1000-1999 Item NPC, each ID corresponds to one or more items (See "ItemID" below)

2000-2999 Minigame NPC, each ID corresponds to a minigame (See "Minigames" below)

If the same id is specified in the file twice a random ID in the range 9000-9999 will be assigned, however this might cause issues when trying to identify the NPC.

4.2.2.3 Minigames

The following table shows which minigames are assigned to which ID's

ID	Minigame
2000	BeerChug
2001	Caps
2002	Cortège

4.2.2.4 Name

The name can be any type of string, it is however recommended to use a memorable short word.

4.2.2.5 Gender

This field specifies the gender of the NPC and could be either MALE, FEMALE or OTHER. Please note that you must only use capital letters.

4.2.2.6 Position

The x and y coordinates specified corresponds to a coordinate on the map. If the position is invalid the NPC will be placed randomly on the map.

4.2.2.7 Direction

This field specifies the direction the NPC is facing and could be either NORTH, WEST, SOUTH or EAST. Please note that you must only use capital letters.

4.2.2.8 ItemID

This field is optional, but is not implemented in the current version (1.0) of Loc.

4.2.3 Dialogs

The dialogs are specified in the Dialogs.loc file located in the assets folder.

4.2.3.1 File format

Each row specifies one dialog and should have the following format:

id:yesOption/okOption:message1:message2:...:messageN:

4.2.3.2 ID

The dialog id corresponds to the id of the NPC that has the dialog. If no dialog is specified for an NPC a dialog in the range 9000-9999 will be randomized.

4.2.3.3 YesOption/okOption

Write “yes” if the dialog has a yes/no option, this should be used by any item- and minigame NPC.

Write “ok” if the dialog only needs ok approval, these are generally used by standard NPCs.

If the format of this field i invalid an ok dialog will be created.

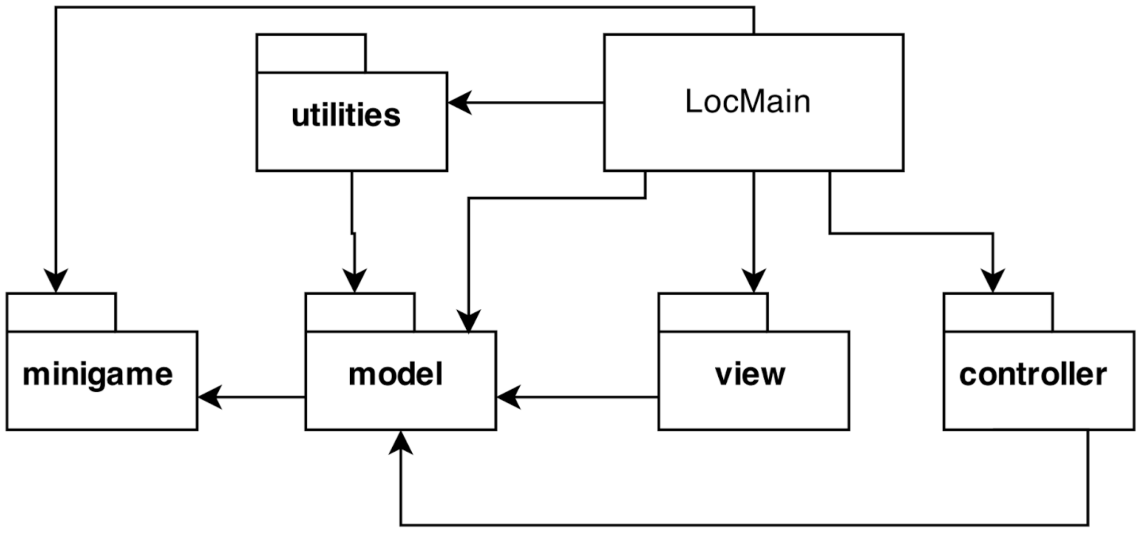
4.2.3.4 Messages

Write messages separated by colons. Each message will create a new message window when displayed in the game.

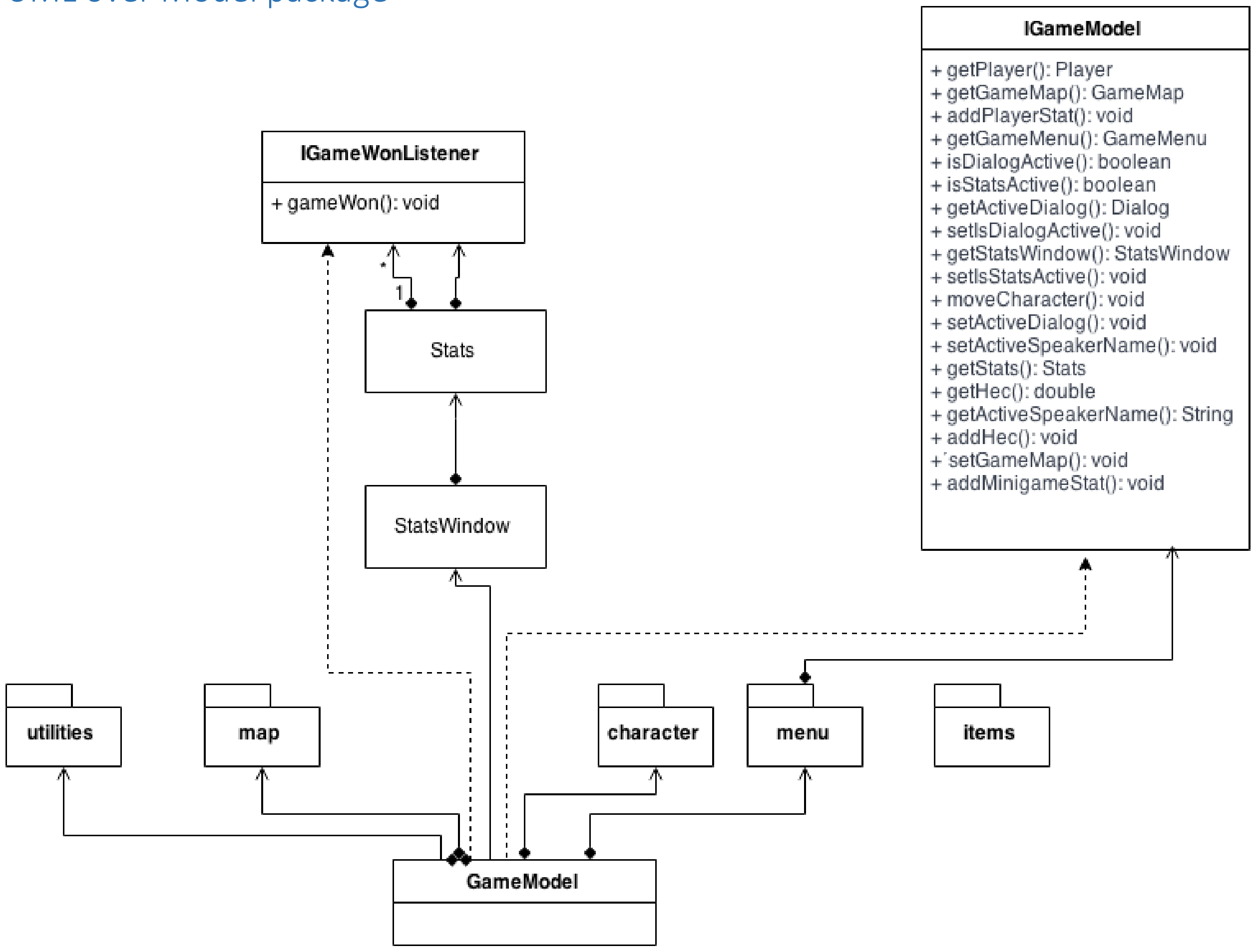
4.3 UML Diagrams

The following simplified diagrams shows the structure of the code

4.3.1 UML over module structure



4.3.2 UML over Model package


IGameModel
+ getPlayer(): Player
+ getGameMap(): GameMap
+ addPlayerStat(): void
+ getGameMenu(): GameMenu
+ isDialogActive(): boolean
+ isStatsActive(): boolean
+ getActiveDialog(): Dialog
+ setIsDialogActive(): void
+ getStatsWindow(): StatsWindow
+ setIsStatsActive(): void
+ moveCharacter(): void
+ setActiveDialog(): void
+ setActiveSpeakerName(): void
+ getStats(): Stats
+ getHec(): double
+ getActiveSpeakerName(): String
+ addHec(): void
+´setGameMap(): void
+ addMinigameStat(): void
IGameWonListener
+ gameWon(): void
*
1
Stats
StatsWindow
utilities
map
character
menu
items
GameModel