System design document for Vanaheim

Version: 1.0

Date: 2013-05-02

Author: Daniel Jonsson, Gabriel Ekblad and Richard Karlsson

This version overrides all previous versions.

1 Introduction

Vanahem is a speed type RPG computer game. You play as a character in a real world composed by obstacles, enemies and NPC:s,

1.1 Design goals

Our goals when designing the game were to make it easy to expand, add functionality and additional feautures. We also wanted a nice pure structure easy to understand.
The whole vanaheim project is incomplete as it is at the current state. It is however a very extendable game which got room for alot more features.

1.2 Definitions, acronyms and abbreviations

Sprite = A graphic component that can be moved around the screen
NPC = Non-player character.
GUI = Graphical user interface
MVC = Model View Controller. Design pattern.
RPG = Role-playing game.

2 System design

2.1 Overview

Our application uses a pure MVC model.

2.1.1 GUI component

The graphical user interface (GUI) uses the Slick2D library. It is decomposed into different states. A game can be in:

- LoadingState. This is the first state shown when starting the application. The user is presented with a loading screen which contains information about which file is being loaded and a progress bar to easily track the loading process.
- MenuState. If the loading was successful, the application enters this state. This state displays a welcome screen to the user.
- ExploreState. The main state for the game. Here the user can navigate around the world and explore different areas.
- FightState. The application enters this state if a fight is triggered. When a fight ends, the user is either redirected to the menu state or the explore state depending if he/she lost the fight or not.

2.1.2 Event handling

We do not use a lot of event handling in our application. Most of the time, the GUI asks the model for information via the controller. The only thing the GUI listens for is messages from the message buffer. Whenever a new message is added to the message buffer, it is fired to the GUI using the Observer pattern.

2.2 Software decomposition

2.2.1 General

The application is decomposed into the following models:

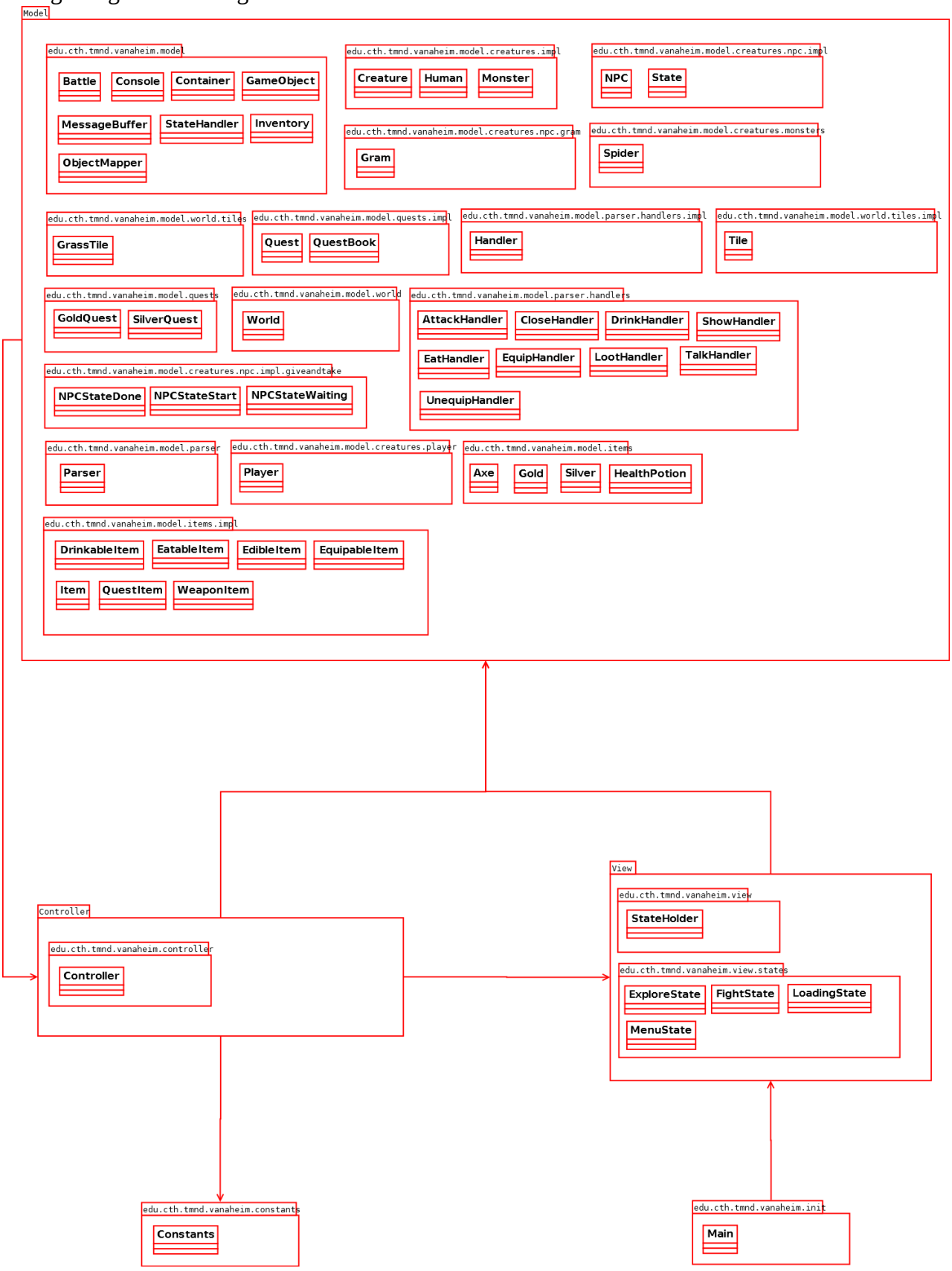
- view, the view part of the MVC where different view states are located. This handles all the drawing onto the canvas and takes all the logic from the controller
- controller, where the controller of the MVC model is located. This handles all the communication between the view and the model.
- Model, where all the model classes are located. This is where the game logic is located.

2.2.2 Decomposition into subsystems

The only subsystem we use are the Slick2D system which in itself handles states and the communication between the abstract methods of slick with the lwjgl library.

2.2.3 Layering

Package diagram in the figure below.



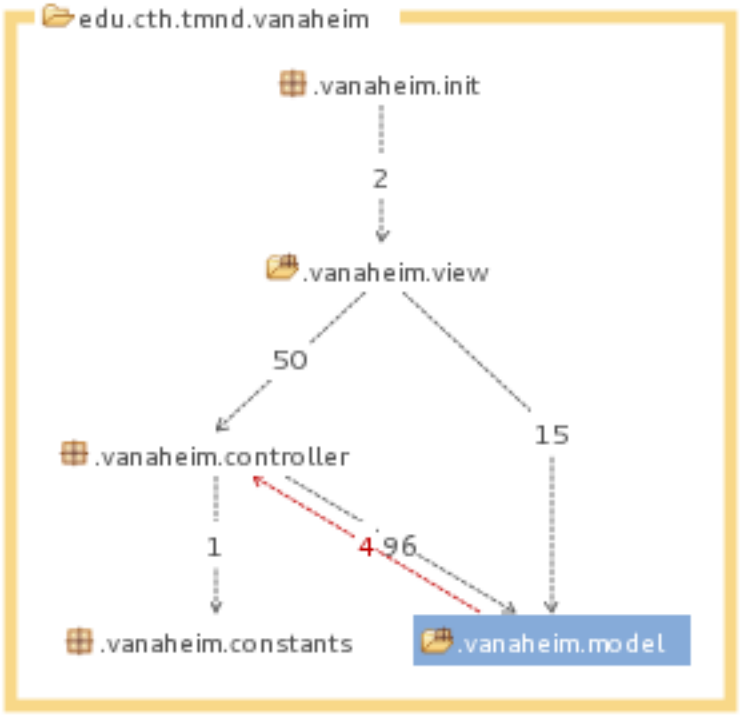
The *.impl subpackages in each package contains in almost all cases base classes , abstract class and/or interfaces.

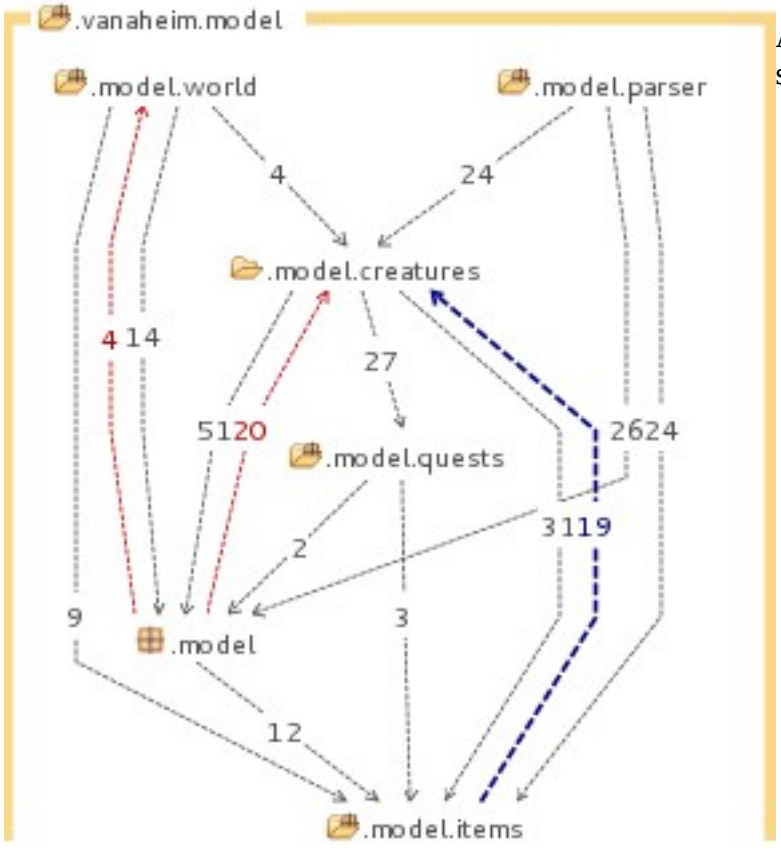
Each package is designed in such a way that the packages should be self-depent in every way.

2.2.4 Dependency analysis

The figure below describes the general dependency structure. There are however some things that need to be described here. First of all. The dependency between the model and the view is to handle various objects coming from the controller, for example Items.

The circular dependency from the controller to the model is a necessary solution for the parser. The loot handler needs to get the coordinate from the player to access a tile. When these kind of design flaws comes up one need to be careful with the usage of the dependency. In this case we will only access information, and not alter any information.





A more robust view of the model is shown in the figure below.

What can be seen in the figure is that there are three circular dependencies. The one between items and creatures are the fact that we chose to add both the creature who used the item and the creature to use the item on. The solution we had before this was to have an owner for each item. That solution was revamped due to the fact that an item shouldn't know of its owner. The item should however be dependent on how to use itself when a creature is using it.

The circular dependency between the model and the world is in fact a design choice. The problem lies in the fact that a battle knows of the tile the battle is located on. This makes it easy and simple to interact with the tile when for example dropping items on the tile when the battle has ended.

The last dependency is an effect of the previously mentioned dependency.

When it comes to making games, it is a principal to not use the MVC model due to the fact of the unnecessary communication between the controller and the model. A better design choice when making a game is the Entity/Component System which composites objects with components instead of inheritence.

2.3 Concurrency issues

Since Slick2D is single threaded and advises developers to stick to single threaded, our application is mostly single threaded. The only multithreading we have is in the fight state where a timer runs in its own thread. This can cause concurrency issues but they are very rare.

2.4 Persistent data management
Commands is stored in a seperate text file which got the following structure:
command : HandlerPrefix
where the command can be compositioned by wildcards and static command text. Wildcards in this case stands for an arbitrary object. E.g.
Drink * : Drink

This says more precisely ”drink [an object] and send the objects to the DrinkHandler.

When it comes to persistent game information like dialog or various creatures behaviours we have agreed on having this information in the respective classes.
We could have used scripts like Lua or something similiar to store this information, but we found no use in it really.

Maps of the world is stored in tmx files which are generated by Slick2D's own map maker.

Music files are stored in ordinary *.ogg files which are the only format the Slick2D library can handle.

2.5 Access control and security

N/A

2.6 Boundary conditions

NA. The application is launched and exited as a normal desktop application.

3 References

1. MVC, see http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
2. Slick2D, see http://www.slick2d.org/

APPENDIX

gitinspector summarized.txt – Summarizes the gitinspector statistics and explains aliases.