<World Engine>

Analysis and Design Document

Student: Stefanescu Marian

**Group:30233**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <20/05/2017> | <1.0> | Initial class design and data structures | <Stefanescu Marian> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

# Project Specification

# World Engine is an online browser game, that puts the user in a medieval setting and offers the opportunity to build an advanced village, using local resources that can be found around it. A major difference between World Engine and other browser based games will be in the interactivity: W.E. will allow the player to choose the location of his village building in a dynamic fashion and control a primary player, or a hero.

# Elaboration – Iteration 1.1

# Domain Model

Though the domain model can be become very complicated for my specific project, I’ve chosen to illustrate the bare bones structure of my application. As can be seen in the image below, the simplest structure of my application would imply having a User that has three primary options regarding the general playing atmosphere.

In “World Engine” each user will have multiple villages and a Map, were he will be able to see the other players and the opportunities the land has to offer in terms of resources. Because this will be an adversarial game, each user will have his own army, with different troops, user to user attacks being possible.

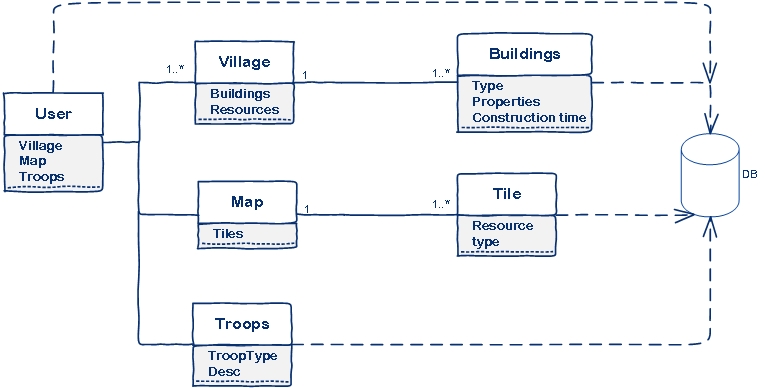


Fig. Barebones domain model of the World Engine

The village will have a collection of buildings, and each map will be a collection of tiles. I’ve chosen tiles instead of a standalone image that I would break in different pieces, because I want to generate the map randomly, and also give a 3D feel by using the isometric projection.

# Architectural Design

## Conceptual Architecture

The system will use an MVC driven architecture, because this architecture generally works very well with Web Applications. In this sense, the peculiarities of the application will reside mostly in the Controller and Model sections, but especially the Controller, as all the Business Logic resides there.

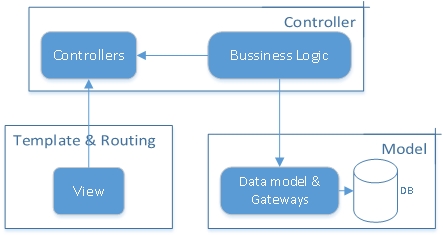


Fig. MVC driven architecture

## Package Design

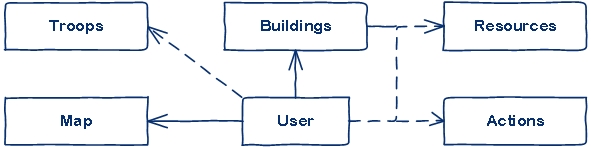


Fig. Major packages

It is important to note that this package diagram was extended in the second version, following the general MVC architecture. The main point here is the image showcases an actual implementation that wasn’t changed but rather extended.

Though at first glance, some packages might be taken by mistake as classes, in this project this is not the case, as each package has behind a couple of classes that are very specific to it. For example, the **Buildings** package contains all the subclasses with all the type of Buildings. I’ve chosen a package for each of them because in time, surely this packages should become increasingly large.

The **Actions** package encompasses all the relations that take place between the users. In the image above, all the packages, except the **Actions** one, have a direct or indirect relationship with the user. This makes very decoupled, except the pivotal point represented by the **User** (which in the application is only a class).

## Component and Deployment Diagrams

# C:\Users\maria\AppData\Local\Microsoft\Windows\INetCache\Content.Word\component.jpg

Fig. Component Diagram

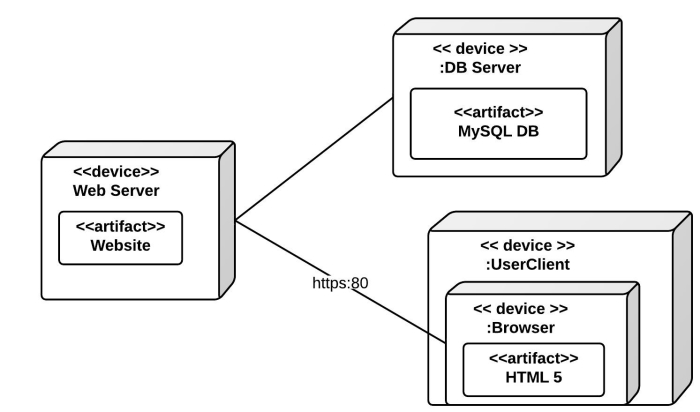


Fig. Deployment diagram

# Elaboration – Iteration 1.2

# Design Model

## Class Design

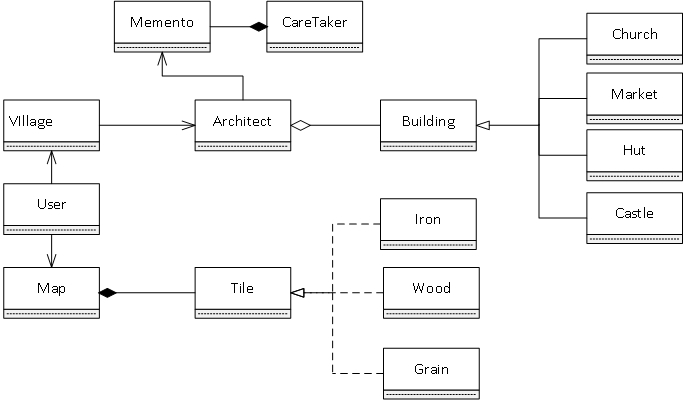


Fig. General user-centered UML

The software pattern used in this diagram are the recurrent Builder Patterns applied at the level of Building and Tile. The central role of this pattern is to increase the relative complexity of each object instance, which happens in our case, starting from less specialized versions, that are described by the superclasses, Building and Tile and going to the specialized versions described in the subclasses.

The Memento pattern is used at the level of the Architect class, because we want for a specific time, to be able to return to the previous state, before the initiation of a building queue(because the user may want to cancel that action, even though the construction process started at a specific time in the past and still hasn’t ended.

The general class diagram follow a Top Down approach, starting from the User, which is the central class, containing indirectly all the logic behind each player’s instance of the game. The idea is to make the User as an exterior interface towards more advanced inter user actions, that will create the gameplay scenario.

# Data Model

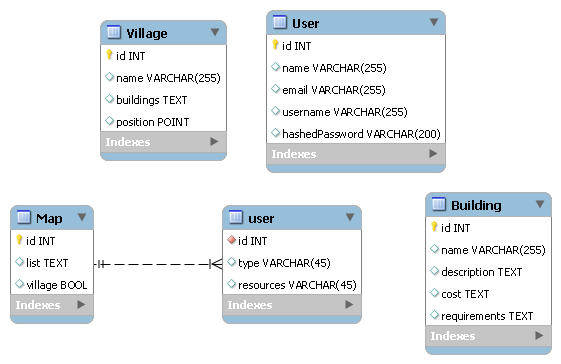


Fig. MySQL data model

# Unit Testing

The methods extensively used were incremental tests made during the various implementation stages and functional testing of the system, were the system was treated from the perspective of a normal user, without taking into account the implementations made under the hood. The idea behind this was to offer a fully functional end version of the game, so that bugs would have ever existed, they would have been constrained in the backend.

# Elaboration – Iteration 2

# Architectural Design Refinement

No architectural refinements were made, as the general structured maintained a good general structure. The difference was done by adding two news packages, View and Controller, which linked with the current packages that defined the Bussiness Logic.

# Design Model Refinement

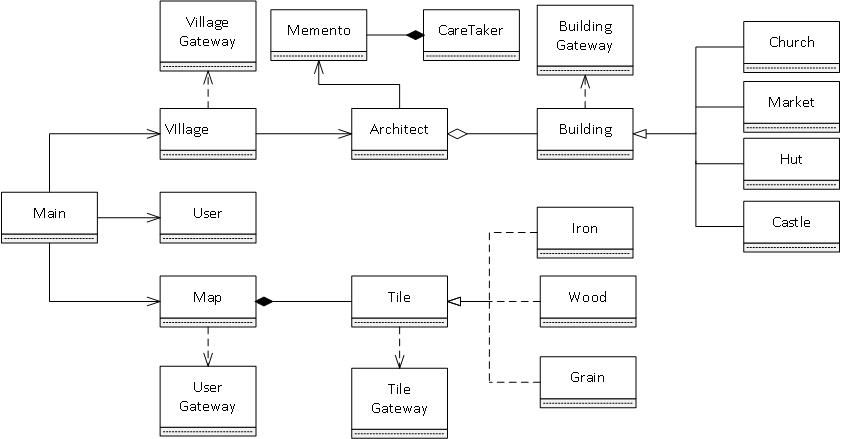


Fig. Updated UML

# Construction and Transition

# System Testing

|  |  |  |  |
| --- | --- | --- | --- |
| # | Test Case Objective | Test Case Description | Expected result |
| 1 | To check if the login works properly | Checked by entering various email addresses, that were not in the DB. Tried SQL injection and wrong passwords. | To obtain error messages at each step of the test cases |
| 2 | To check if the buildings can be dragged and dropped in the construction zone | The buildings available in the Village perspective should offer the possibility to be dragged and dropped in the construction area, and for the buildings to stop appearing in the unconstructed queue. | To be able to drag all the buildings in the construction area, and not be able to put them outside of it. |
| 3 | To check if the construction queue works | The user would put in the time queue multiple buildings to be constructed, depending on the resources it has and then cancelling a random number of them, repeating the test multiple times. | To come back at the initial state, prior to the construct action. |

# Future improvements

The system should allow in the future a single player mode for each user and also a Hero

mode, were each user will build a hero that would be used in various kind of actions in the village and outside. A major difference in comparison with games from the same genre would be to have an animated character, that would implement simple movements (walking, running, fighting etc.). The end goal would be to move the game away from the static point and click approach and to create a specific style of it’s own.

# Bibliography

* <http://www.guru99.com/integration-testing.html>
* <https://en.wikipedia.org/wiki/Architectural_pattern>
* <http://www.scaledagileframework.com/domain-modeling/>
* <https://www.lynda.com/Java-tutorials/Introduction-GRASP-principles/96949/106106-4.html>
* <http://www.uml-diagrams.org/component-diagrams.html>