Paul Hunter, Mark Roller, Justin Guze, Scott Low, Allen Chang and Mikko Sanchez
August 3rd, 2012

Group 3’s Milestone 4 Report

Class Model Evolution

Throughout the development process, the class model for our implementation of Almost Civilization evolved in several ways. In the early stages, we simply created a diagram based on our initial assumptions of what we would need in order to make a functional game. As time progressed, however, it became apparent that our class model would not be able to accurately reflect the interactions between classes that were necessary to complete our game. As a result, during each milestone, a major refactoring of the class diagram was done to ensure that it provided us with a cohesive description of the system as a whole. Furthermore, based on feedback we received after Milestone 3, our class model underwent a major structural refactoring in order to ensure readability and a strong reflection of the structure in the system. This was deemed to be crucial, in case if we decided to pick the project up later, we wanted to be able to recall what we had done based on the class diagram alone, as the system itself is large and convoluted. What follows is a description of how our class model for ACiv evolved from milestone to milestone, and how we ultimately arrived at the class model we have today from initial concepts over a month ago.

Milestone 1 - 2:
Our initial class model diagram consisted of the base functionality we thought would be required for a successful ACiv game. It took into account the features required to have a playable game such as maps, units, unit movement, and unit combat. The most obvious design pattern that we felt applied to the project and implemented was the MVC pattern. With this pattern in mind we created the World object which acted as the model for the game and held the game’s current state. The GameEngine was the ‘controller’ of the game and modified the World based on the users input. Finally, the MapView acted as the ‘view’ of our MVC based system and displayed the World’s state to the user. Also, basic objects such as Cities, Players, and Units were hashed out during Milestone 1. Specific tasks that the GameEngine would perform were placed into separate Utility and Manager classes, which were determined at the time to be SpriteUtils, TerrainManager, CombatManager, and NavigationUtils. These classes were designed for handling most, if not all the functionality of the game. Another decision at that time was for every Unit to be represented as an enumeration value instead of using inheritance which would have seen us creating roughly fifty different unit subclasses.

As we moved to Milestone 2, our class diagram grew greatly in size. Classes that we had not yet thought of began to appear as we began implementation of more features. Maintaining the original architectural design in Milestone 1 also proved to be challenging, and as a result the MVC pattern that was in use was no longer a true MVC pattern. Listening for input such as clicks or keyboard presses was handled by the MapView, which passed it on to the GameEngine to determine how to process the input. Therefore the view handled both input and output, and talked to the GameEngine to talk to the World. More managers were created to handle use cases such as unit creation and city creation in an to attempt to stop the explosion of code that was being written within the GameEngine.

Milestone 2 - 3:
In Milestone 3, we implemented navigation/pathfinding, city creation, unit creation, combat, and extra functionality such as resource bonuses on terrain, saving the game, a minimap, and an enhanced user interface to setup and start a game. This extra functionality was not accounted in our initial class diagram architecture, and thus we had to add more classes and adjust the design to account for it. Most extra use cases were handled by existing managers or by creating more methods in the GameEngine, which led to GameEngine becoming 'all-in-one' class at the end of this milestone. This increase in classes led to a fairly unreadable class model which in turn prompted us to package classes to encapsulate functionality, this helped us to greatly clean up our class diagram and make it more understandable.

Milestone 3 - 4:
In Milestone 4, nothing was required to be added in order to meet the requirements for unit attack/defence and movement over terrain since these were already implemented in Milestone 3. As a result, extra functionality was implemented instead. A research tree was created to upgrade a civilizations technology and introduce new units that the play was about to create. Features regarding cities were also implemented as such as food supply, gold production, terrain bonuses leading to increases in food and production of units as well as science. In addition, we also implemented some 'fun' into the system in the form of cheats and our own custom units for each team member. In terms of new classes, the city class was already created, so new methods were added within the city class to implement the extra functionality. The new research tree required several new classes, such as a TreeNode abstract class and others consisted of concrete classes.

Sequence Diagrams vs. Implementation

Throughout each implementation of the functionality for our tasks scenarios, the main aspects of our sequence diagrams that changed were elements that needed to be expanded upon in order to implement new features within our project. For instance, when we first implemented unit movement, it was a basic teleport to any place on the map; we then had to change this to incorporate pathfinding, and again to include terrain restrictions. Over this process, the main idea behind unit movement did not change, however, sequence diagram slowly evolved and became more complex in order to incorporate the new and changing elements. However, other tasks underwent several changes if not a large refactor as they became cumbersome and difficult to expand upon.

Below are several paragraphs to explain the general changes that happened with our sequence diagrams throughout the iterations of our project.

Milestone 1:
For this milestone we included three new sequence diagrams that were to be implemented for milestone 2. These included: displaying a map of size 10 by 10, moving the view of the map north by five units, and moving a player's character west by five units (ignoring terrain restrictions). No changes were made to existing sequence diagrams as we did not have any at that time.

Milestone 2:
For this milestone, we included three new sequence diagrams that were to be implemented for milestone 3. These were done for use cases involving city creation, unit creation, and unit movement. The particular use cases were the creation of a unit, creation of a city using a settler, and to move a land unit over ocean tiles. Not only did we create several new sequence diagrams, but we also made changes to the milestone 1 diagrams. However, this was not to an extent that deserves notoriety; it was a result of not changing those aspects of the project when we implemented new features in this milestone.

Milestone 3:
For this milestone we included three new sequence diagrams that were to be implemented for milestone 4. These were done for use cases involving unit attack and defense as well as unit movement over terrain. The particular use cases were movement that took into account terrain cost, and combat between two fighting units. Not only did we create several new sequence diagrams, but we also changed our existing ones when our implementation of what they represented needed to be changed. Below is a list of the sequence diagrams that were changed and what was changed.

> *Create a Warrior:*
> In Milestone 2's sequence diagram, we had the game engine dealing with the selection of which unit to produce and we assumed this is where we would also control the turns remaining to create that unit for a player. This was changed in milestone 3 to be controlled by the city itself, particularly by keeping track of production turns left in the city class. We refactored the system so that the UnitManager created the unit instead of GameEngine.
>
> *Move Land Unit Over Ocean Tile:*
> The only change here was that we changed the name of the function called from NavigationUtils to findPath instead of getRoute. This was done because we needed to update the unit movement sequence diagram to include terrain restrictions and pathfinding. FindPath was made to include all of this within NavigationUtils and completely replaced getRoute. Note, these changes also were applied to the unit movement diagram.

Note that no other changes were made to our sequence diagrams for this milestone and the changes that did occur were mainly to allow us to implement new features within our program.

Milestone 4:
For this milestone we did not include any new sequence diagrams as we are not implement anymore aspects of the game. We also only needed minor name changes on milestone 3's sequence diagrams. Minor changes were only needed as we had already implemented most of the tasks described in the milestone 3 use cases before the diagrams needed to be created.

Overall, as we modified our code to incorporate new features, the changes that were made to the sequence diagrams were minor. Thus we feel that we followed our sequence diagrams quite well when implementing new features in all of the milestones.

UML and Design/Implementation

By being restricted to use UML notation to design class diagrams, it forced us to determine what an object would do before attempting to implement it. As such, we didn't end up with many pointless classes, or classes with too much functionality in one place. By designing the classes before implementation, it gave us a good overview what the project would look like before it was coded.

At one point, our class diagram became quite cluttered due to classes having vague association relationships and having classes all sitting in the same place. This reflected the code as well which had classes sitting in the same folder. This led us to package classes that had similar functionality with associations. It also cleaned up our class diagram.

In the class diagram we had various classes defined a having a composition relationship with each other. We noticed that there really was only one needed in the implementation, and led to using Singletons for these classes rather than pointless instantiation.

Architecture Recommendations

Given the opportunity to start the project from scratch, we would change a few aspects of the architecture of our Almost-Civilization project. First would be "MVC-esk" pattern used for the main game window would be changed to a 'true' MVC pattern. At this point in time, aspects of the data, such as players and which should be kept in the model (which in our project is the World), and aspects of the controller are spread throughout the code behind of UI elements and within the GameEngine.

Secondly, The game managers, such as the TerrainManager, UnitManager, and the like were never thought out as being Singletons, although in retrospect we would have liked (and most likely will refactor) them to be Singleton, as there is only ever one in use. This would also help to decouple some class references within these managers and to these managers.

As well, we regret holding a reference to the World object within the GameEngine as we ran into issues standardizing the ways that people and the code they wrote were accessing the World and its fields. We agreed that it would have been nice to encapsulate the World in into another manager style singleton, as we only have one world in memory during main game play.

We also found that GameEngine was rather 'fat' as we worked towards Milestone three. Much of the functionality within GameEngine could have been better encapsulated within managers in charge of things like turn management and unit movement. In addition, GameEngine could have been converted to a singleton as well as we only have one in use at a time, and shouldn't have more than one.

Lastly CityManager was not fully utilized as it was originally intended. Much of the functionality of cities leaked itself into the City class, which has in turn become slightly bloated.