Computing Project: ATHENA

Kieran Armstrong, Alastair Campbell, Grant Thorburn

May 5, 2014

1 Group-Based Project Plan

1.1 Introduction to Project

Our group project, code-named ATHENA, was to create a video game. We discussed some ideas and eventually agreed to create a science-fiction themed game in the RPG genre with a turn-based combat system. We chose this because we thought that it would be an interesting setting and theme for the game and also due to our belief that it would be less complex to implement compared to a more fast-paced action-oriented game.

Once we had an idea of what we wanted to achieve, we approached the project with an approach which was particularly planning-heavy, creating a very detailed project plan with accompanying charts and diagrams, alongside a broad plan of what we intended our game to be like - level design, story elements, characters, etc. We then fleshed out these basic ideas until we had, essentially, the elements of our game in terms of UI design, character sprites, the lines spoken by characters and when they would be spoken, along with a chronology of the story taking place. The game engine of our project was the biggest element of the game and received the most work on it as a result.

We were aware of the scale of our project so we knew we had to keep strictly to schedules. Our project was originally planned out task by task, with an estimate of how long we thought each task would take. Resources were then assigned to the tasks appropriately. Other than human resources, we used several tools (including Microsoft Visual Studio, Microsoft Word, Microsoft Visio, Microsoft Project and Adobe Photoshop) to create or manage our project.

1.2 Gantt/PERT Chart and Critical Path

2 Group-Based Development

2.1 Project Planning and Approach

Once the project plan was created, it was largely ignored. This was probably due to the fact that it was so prescriptive about what must be done and when it must be done that it was overburdening. The project was managed on an ad-hoc basis, with work being where it was needed rather than in accordance with the plan. Due to this, certain tasks fell behind and others received the majority of the work. For example, the game engine got the largest amount of work done on it, whereas sprites, level design, sounds, etc. were nearly completely neglected. I believe that a more lightweight, agile project plan to start with would have benefit the project more than the heavyweight plan we had, allowing us to attempt to follow the plan more rather than blindly working on whatever suited at the time.

2.2 Design of Deliverable

We adopted a mostly ad-hoc approach to the design of our game. Due to time constraints, it was decided that the time taken to plan the class hierarchy would be better spent implementing the classes themselves. This approach was very successful and led to a very powerful and robust, but complex, game engine. The engine itself was designed to be highly object-oriented and was written in C# on the XNA 4.0 Framework, which turned out to be the biggest flaw in the project. XNA 4.0 was officially dropped by Microsoft and was not supported in the latest versions of Visual Studio, requiring several workarounds to be deployed to be able to use certain workflows. The project was later switched to a free implementation of XNA 4.0 called MonoGame, but this was not without flaws (for example, it was impossible to compile fonts into sprites using MonoGame). However, most elements of our game, such as the story, user interface and sprites were planned and designed prior to implementation.

We quickly adopted a version control system (VCS) for managing our project code and files. Initially, we used `git` to manage our source code and project files. This was later switched to Microsoft's built-in Team Foundation software for better integration with the Visual Studio environment. The Team Foundation Version Control was used to control the source code version while `git` was still used to maintain shared copies of the Project file, UI plans, floorplans, diagrams, etc.

We designed our story in Microsoft OneNote. We wrote some ideas of what we wanted our story to be like, then we wrote a plot overview and finally wrote the story using a storyboard and a set of spoken dialogue for each character.

Similarly, we planned our user interface and character designs in Adobe Photoshop. We sketched out how we wanted our characters to look, then we refined these sketches until we had a complete sprite for each character.

The same method was used for level design. Simple sketches of how the levels should look were created, and then the designs were digitized using Microsoft Visio.

The game engine used a slightly different design method. Rather than planning progressively how we wanted it to be, we had a very strong idea of what would be needed, allowing us to forego the more formal design process used for other game elements. We created a very strong object-oriented structure in Microsoft Visual Studio itself, rather than planning it using a modelling system such as UML first. This approach was chosen due to the fact that it would take far longer to do in UML first and we had very limited time for this very ambitious project.

2.3 Development of Deliverable

2.3.1 Description of Deliverable

The prototype is mostly comprised of the game engine. Our engine is a very strong class-based inheritance structure featuring objects to draw many types of entity and handles them in a very reliable way, using interfaces to ensure compliance. Our prototype solution also features a test suite to attempt code regressions. To run our prototype, you require a recent version (2010 or later) of Microsoft Visual Studio. To compile the prototype you will need a copy of the latest build of MonoGame (version 3.2 at time of writing)

2.3.2 Deliverable Features

Our prototype is very unfinished. The features that we have implemented are:

- Class-based inheritance structure
- Drawing standalone sprites or sprites out of a spritesheet
- Drawing UI elements (such as buttons) with labels
- Actor (such as player, enemy, etc.) movement with collision detection
- Advanced XML-based level loading system
- XML-based tileset system
- Unfinished system for handling RPG elements, such as experience, health and attributes
- A camera system that tweens to the focused object and allows zoom, rotation, etc.
- Several helper classes which add to the framework (for handling graphics, fonts, etc.)
- Code reflection in game objects (For example, triggers written in C# which trigger when a player walks on them)
- Limited user-interface components - buttons can be displayed, but they are only for show currently.

Features which we have not yet implemented, but plan to:

- Actual gameplay - levels need to be encoded into XML and the unfinished combat system needs finished.
- User Interface - More elements with interactivity (using code reflection like trigger tiles)
- Sounds - These aren't yet implemented but there is some groundwork done in this already such as the ResourceManager helper class.

2.4 Implementation & Evaluation

2.4.1 Implementation Models and Approaches

Our initial plan was to develop our game engine in parts for example, player movement and collision detection. We then realised that some parts relied on other parts being completed, for example Collision Detection needed Level Loader and Player Movement. So our approach changed, we started focusing on individual parts rather than trying to make everything work at once.

Our group utilised an Agile Methodology, agile methodolgy promotes adaptive planning, iterative development and a project which is able to adapt to change rapidly. this benefitted our project as much of the development was ad-hoc and unstructured. Some flaws with this methodology are that sometimes you make mistakes and time is wasted but the advantages of wasting less time planning development as opposed to actaully developing far out weighs the flaws.

Class diagrams are very useful tool during development, we decided not to use class diagrams. We thought that if we never implemented class diagrams that it would reduce development time. this did have the desired effect however if we implemented class diagram the development time may have been reduced. The output however had the same effect the only flaw was some time may have been lost.

At the start of the project we implemented planning first approach, this wasted alot of time as we never followed the plan. We believe that if we utilised an agile methodology from the begining we would could've put the time that was wasted to a better cause.

2.4.2 Key areas addressed

We split our project implementation into five main parts - the game engine, the story script, the graphical design, the sound design and the level design.

Game Engine This involved planning and writing classes to work together as part of our game to support all of the necessary gameplay features. The key features of the Game Engine are:

- Programming Movement Logic - player movement and npc movement
- Programming RPG Elements - this includes - levelling, character stats etc.
- Programming Collision Detection - players interaction with level
- Programming Graphic Handler - this includes Textures, Sprite Sheets etc.
- Programming Story Handler - handles story elements
- Programming Level generator - converts XML documents into levels

- Programming Encounters - triggers battles encounters, triggers story encounters.
- Programming Battle System - this includes damage recieved/taken, item usage etc.
- Programming Sound Handler - this includes background music, battle music and sound effects

Story Script This involved writing the story, dialogue and how the story would progress. during the planning we decided how our story would progress. We also decided on character names and what characteristics they had. We also decided that we wanted our game to have multiple ending depending on user interaction with our game so we thought of 3 endings: good, evil and neutral.

Graphical Design This involved creating graphic assets such as sprites and UI elements. We firstly created sketches of what we wanted our characters and levels to look like, and then we created prototypes in Photoshop, which then became actual game assets.

Sound Design This involved choosing appropriate sounds and background music. We chose appropriate sounds for our game and then we saved these as .mp3 for import to our game.

Level Design This involved designing levels for the game. We sketched how we wanted our levels to look and then we digitized them in Publisher. These levels can then be written in the XML format for our level import tool.

2.4.3 Evaluation

This development will be evaluated by looking at the work done and based on how much of the work was completed. The game engine was the furthest into development. player movement, collision detection and drawing onto the screen were all implemented, the only things that weren't fully operational was the user interface, story elements and sound.

Although the game was never finish we done testing of the engine. We asked a third party to perform gameplay test of the prototype. where collision detection, full player movement, random encounter triggers and test sprites were drawn to the screen. They were also shown the game script and some of the graphic concept.

We also applied unit testing, unit testing is performing test against self contained units such as classes and functions. We used unit test to prevent code regression (i.e change code breaking existing functionality) so when the solution was compiled, if there was any code regression the tests would fail.

Another technique for evaluating software is to use various testing techniques, such as white box/black box testing, top-down and bottom-up testing, etc.