Animageddon

Final Report

Team E3

Barnaby Colby
Christopher Thorne
Tom Whale
Richard Maskell
Arun Bahra

Contents

Introduction

The Game

Animageddon is a based on the popular concept of “Capture The Flag”. The idea of this game mode is to capture your enemy team’s flag and return it to your own base. This has been implemented in many popular games such as Call of Duty, Halo and Gears of War. In these games, the CTF game mode forms only a small portion of the game, whereas Animageddon is focused entirely on this concept.

The game has a top-down view and uses animals for the players character. It incorporates both cooperative and competitive aspects by using 2 teams with multiple members on each. The user can decide on whether to play with other users via the internet and / or local network; or to use the built in AI players which aim to be as valuable as a real user. This feature is used in order to balance the teams, improving gameplay quality, or to allow a user to enjoy a single player experience.

The idea of using animals as characters is based off the popularity and success of recent games with animals, such as Angry Birds. Furthermore, it allowed us to incorporate animal classes where each one had a specific skill set that the player could intuitively understand from the type of animal. For example, an Elephant has high damage and health, and low speed due to it being strong and large. Alternatively, a monkey has low health and damage, but is extremely fast. To continue with the collaborative aspect of the game, this enables team players to employ tactics and strategies using different classes. For example, the monkey may be used to capture the flag because of its speed, whereas the elephant could be used to defend the flag holder, due to its high health and damage.

Report Overview

The rest of this document delves further into the details of the game; from the technical aspects, such as the classes and algorithms used, through to the ways in which we managed the project, and any issues we encountered during development.

Software Design

Game Model

Animageddon uses the Entity part of the well known and widely used Entity Component system explained in Bilas (2002). Instead of using the Composition Over Inheritance design pattern that is part of the Entity Component System, we decided to use standard inheritance to achieve the same goal. The Composition Over Inheritance pattern provides the additional benefits that the initial design of the entities is simplified, because the behaviours can be added very easily to each entity, and modification is easier when requirements change. However the entities often must define many ‘forwarding methods’ which can be a lot of work, and can make the code messy and bloated. Whilst both inheritance and composition are both valid techniques, we decided to use the inheritance method to ensure that our code was as clean as possible.

Data Structures

Maps

The map files are stored within the “*maps/*” directory as XML files. The name of the file relates to the name of the map and is unique. The structure of these files is defined and explained in the file *“maps/template.xml”.*

Navigation Mesh

The navigation mesh for each map is stored in the file with the same name as the map it corresponds to, but has the file extension ‘.navmesh’. The structure of a navigation mesh file is defined and explained in the file *“maps/template.navmesh”*.

Main Classes

The GameWindow class, which contains the main method for the animageddon client, and the DedicatedServer class, which contains the main method for the animageddon dedicated server both play a similar role. Both classes initialise the relevant World objects and continually call the update methods to update the state of each of them. The DedicatedServer also listens for new connections and the GameWindow class also causes the GUI to be rendered.

The World classes instantiated by the GameWindow and DedicatedServer classes both inherit from the abstract World class in the shared package. This World class describes a basic interface for updating a game world and getting the other objects relating to the map and the entities, as well as some basic functionality shared by both the ServerWorld and ClientWorld classes.

The Entity class provides basic functionality for objects that can be placed on the map, such as obstacles, players and flags. The classes for these entities inherit from the abstract Entity class and provide information about how the entity should be updated and rendered as well as any other entity specific functionality.

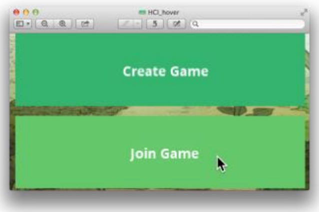
Human Computer Interaction

Design Process

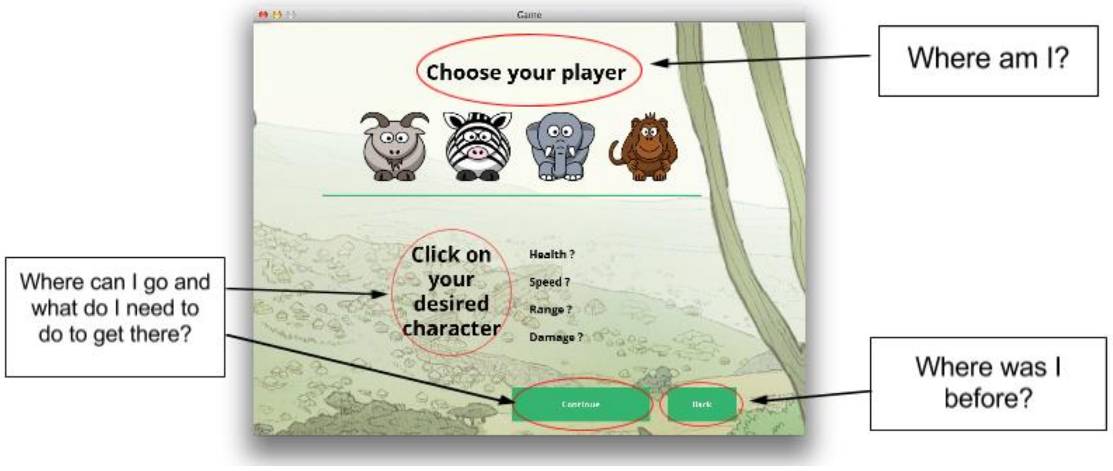
The human computer interaction in our game used a combination of both system centered design and user centered design. Due to the time constraints that we faced, we needed to figure out what we could build to make the game as interactive as possible with the tools we had at our disposal. This included how we were going to design the menus and how easily the users could interact with the game.

Screen Design and Interaction

Menu navigation and character selection plays an import role in intialising the games configuration, and so we needed to design the screen layout to minimise confusion and maximise clarity. This was partly achieved through the design of the GUI.

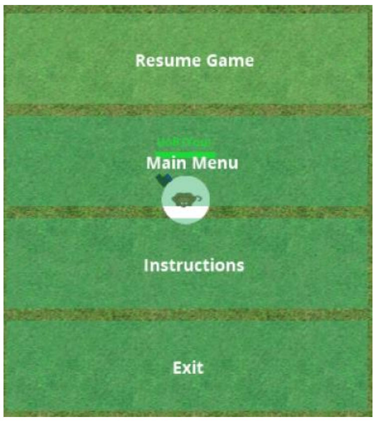


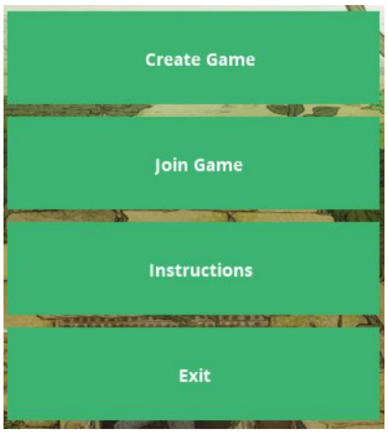
The contrasting colours used in the GUI, allow the user to see exactly what component (button, character) they are selecting. Not only does this add to the aesthetics of the game, but it minimises the learning process that the user must go through in order to operate the game. For example, the image on the right shows a button changing colour when the mouse is hovered over it. In this case, the user knows that if they click the mouse button then they will join a game. This is an important, as it brings predictability into the design, whilst also offering constant user feedback. Another trait which our design takes into consideration is observability. This lets the user know where they are in navigation terms.



The labels above show that the GUI has been designed to reduce ambiguity. This is so that the user knows what functional goals must be achieved in order to progress to the next stage of the games configuration.

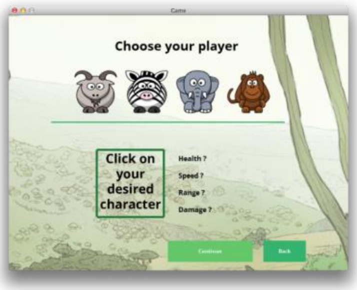
An important feature of HCI, which we have taken into consideration, is consistency. Each menu is implemented using the same template which means no matter where the user is in the game, they will already be familiar with this layout, and know how to get to the desired destination. This reduces the chance of error (navigating to the incorrect goal) and erases the need to memorize certain interaction sequences. For example, the in game menu and main have the same layout but different functional goals. If these menus were visually different then the user would have to think differently about how to navigate them.



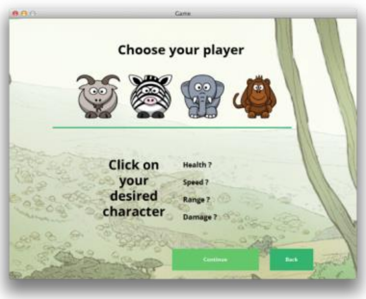


Error Recoverability

The interface has been designed to increase the recoverability when a configuration error occurs. For example, if the user attempts to go to the lobby without selecting a player, then they are informed. It is then made clear to the user how they should proceed. Again, the GUI plays an important role. As you can see in the image below, a border is drawn around the instruction, emphasizing to the user the action required in order to continue.



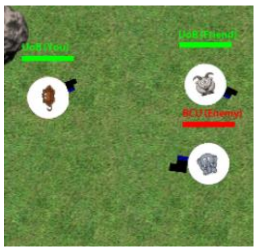
The menu to the right shows the interface after an error has occurred, it makes clear that this is the case and emphasises the steps that should be taken to proceed.



Gameplay

The most important aspect of Animageddon is the gameplay. The in game surroundings have been made as user friendly as possible; with emphasis put on character position and character identification. This ensures that the user has an enjoyable experience and makes it easy for them to understand the environment.

When the game is started the user can immediately identify their character because of the label positioned above their character. This, coupled with a health bar, means that no further explanation is required. During intense gameplay, it may become difficult to distinguish your character quite so easily and this feature attempts to mitigate this possibility. Using colours that contrast the background, the label and health bar are never unrecognisable, increasing the clarity aspect mentioned earlier. Enemy players use the same images as friendly team mates which creates the possibility that users could get confused as to which team a player is on. The enemy identifiers and health bars are contrasted from the friendly ones using red to avoid any confusion. The images below show the features mentioned in action.

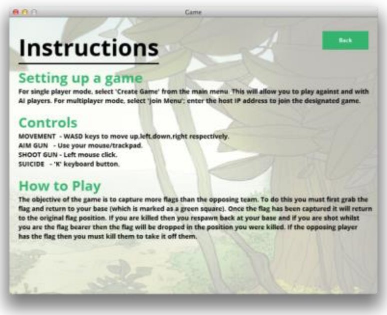


Furthermore, the direction in which the player is facing changes at a fast pace. The player body and gun rotate in accordance to keyboard and mouse input, making the direction they are heading in much clearer. This needs to be accurately represented as any discrepancy could cause the user frustration. We also had to take into consideration the feedback for hit detection. The easiest and most obvious solution is to reduce the proportion of green to red in the health bar of the damaged player for every successful hit.

Sound played a part in notifying the user when they had died and also when they were firing their gun. Two distinct sounds were used so that it was clear what action had occurred. When used with the diminishing health bar it added predictability to the situation, e.g. the user knows they are about to die.

Instruction Page

The last major feature we incorporated was an instruction page. The position of said page had to be correct so that the users could easily locate and access it. We decided it was important to display it on the main menu and on the in game menu so that the user could have access at two vital points; before game starts and during the game. The design of this page conforms with our consistent theme of clear, engaging colours set on a transparent background.



Instructions

Setting up a game

For single player mode, select 'Create Game' from the main menu. This will allow you to play against and with AI players. For multiplayer mode, select 'Join Menu'; enter the host IP address to join the designated game.

Controls

MOVEMENT - WASD keys to move up,left,down,right respectively.
AIM GUN - Use your mouse/trackpad.
SHOOT GUN - Left mouse click.
SUICIDE - 'K' keyboard button.

How to Play

The objective of the game is to capture more flags than the opposing team. To do this you must first grab the flag and return to your base (which is marked as a green square). Once the flag has been captured it will return to the original flag position. If you are killed then you respawn back at your base and if you are shot whilst you are the flag bearer then the flag will be dropped in the position you were killed. If the opposing player has the flag then you must kill them to take it off them.

GUI

This section contains information about the graphical user interface system, including our design choices and how we implemented its features.

Choice of Design

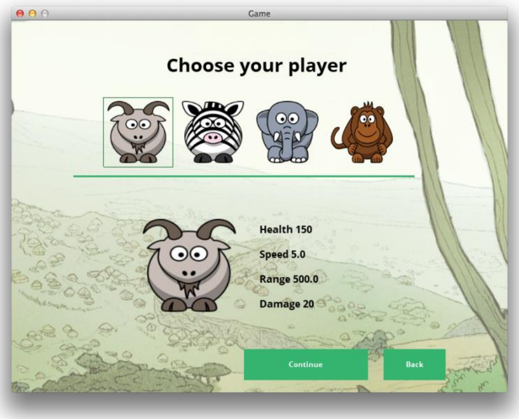
The game library we were using (Lightweight Java Game Library) does not include any dedicated GUI functionality. It is also incompatible with the standard Java GUI library (Swing). A different approach had to be considered. The choices were limited so the decision was made to build the user interface from scratch, modelling it’s design on the Swing architecture.

Structure

The core class of the GUI system is the abstract Component class. Every component has an x coordinate, y coordinate, width and height. Different components, such as Buttons and Labels, inherit from this class, and extend it with their own functionality. This design allowed us to define components that take the functionality of one component and add to it to suit the need of different requirements. An example of this can be seen in the Button and TexturedButton classes; TexturedButton contains all the features of Button but it allows the additional feature of binding images to it.

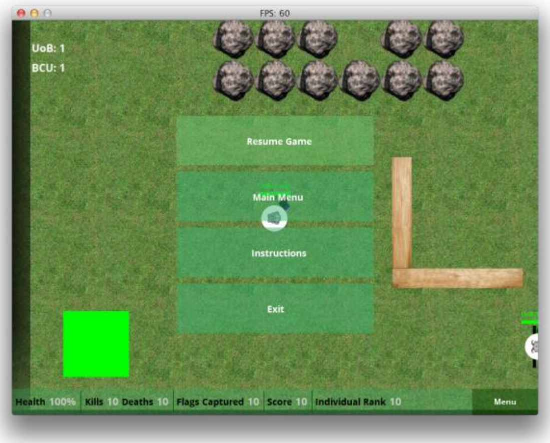
Flexibility

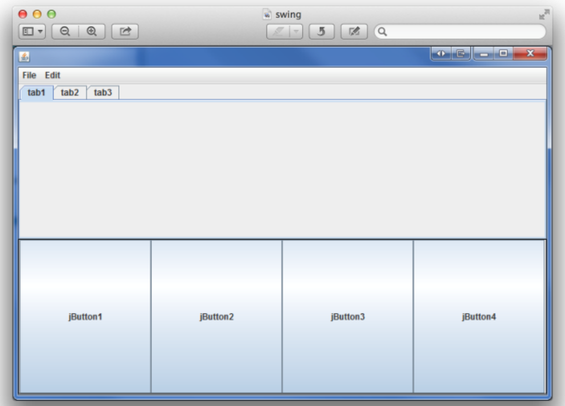
The advantage of our approach is that it provided flexibility and allowed for changing requirements. A menu template was created in order to make the process of creating the menus much easier. The menu template is implemented as an abstract class that contains methods that assist in the design and implementation of customizable menus. This template enabled the menus to have a consistent look and feel, with the option to adapt the layout for different needs. The image to the right was created using this template and has been adapted to meet the requirements for player selection.



Professional Edge

Through the use of custom made GUI elements, a theme was created that has given Animageddon a professional look. Aesthetically, the GUI components are much more appealing, when compared to the default look of the Swing library, as shown below. Due to the extensibility of Animageddon's design, the theme can easily be tailored to meet different requirements. The use of transparency and hover have made the navigation of the game menus both engaging and simple.


FPS: 60
UoB: 1
BCU: 1
Resume Game
Main Menu
Instructions
Exit
Health 100% Kills 10 Deaths 10 Flags Captured 10 Score 10 Individual Rank 10
Menu



Input

Due to the fact that custom GUI libraries were being used, the mouse listeners used for the game had to be built from scratch, in order to allow user input to be registered. This was achieved by creating an Interface containing the listener methods (mouseClicked/mouseEntered). The Component class contains an 'inside' method that takes two coordinates and returns true if they are within the component. Calling this method provided the ability to tell whether the mouse button is inside a particular component. This enables the ability to iterate through the listeners, and, depending on whether the mouse is over the component or the mouse has actually been clicked perform the appropriate actions.

View Switching

There are many menus that are included in the game, a method to switch between them, without any drop in performance, was required. Animageddon does this by using a View class that the MenuTemplate class inherits from. This view class contains two methods, render and update, both of which are called in the game loop. Each menu defines these two methods, so that when the view is switched, the process of updating and rendering the current view remains the same.

Artificial Intelligence

This section contains information about the Artificial Intelligence used within the Animageddon game. In particular; why it is important to the user, what techniques were used, and why.

Importance of AI

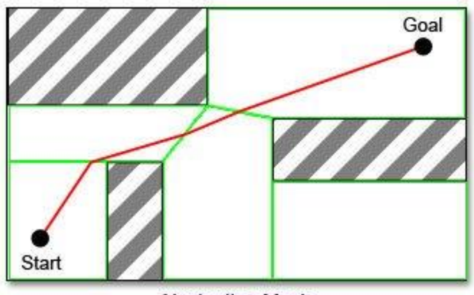
The artificial intelligence of a game is usually a very important factor in its success, or failure, because it can provide both cooperative and competitive aspects to the game. It can also increase the pace at which the game is played, maximising the players interest as well as enjoyment. However, if implemented poorly, it could affect these factors in reverse. Possibly causing the player to lose interest, perhaps because the AI provides no real threat, or causing the player to become frustrated due to the stupidity of their AI team mates. In an ideal world, the AI algorithms used should cause the character to be as equal to the skill and intelligence of a human player whilst keeping the performance of the game as optimal as possible. If the character out-performed or under-performed when compared to a human player then some of the negative connotations of bad AI, mentioned above, may come into play.

Techniques used

In order to achieve the intelligent, high performance AI described above, many hours of research into the techniques and algorithms used in industry were required. Two popular methods for pathfinding were the grid-based method and the navigational mesh method, both of which allowed the character to navigate around obstacles.

The first, grid-based pathfinding, involves splitting the map into equal sized squares that represent the walkable area of the map. Decreasing the size of the squares can cause an increase in the accuracy of the represented area. For example, if a gap between the edge of the grid and an obstacle was too small to be covered by a square, decreasing the size of the squares would allow the size of the gap to be reduced or removed. The limitation of this reduction is that the pathfinding algorithm would take longer to run due to the increased number of nodes in the search tree. The path found by the grid-based technique would also be quite jagged because where the path changes direction, the angle would be a sharp 90° or 270°.

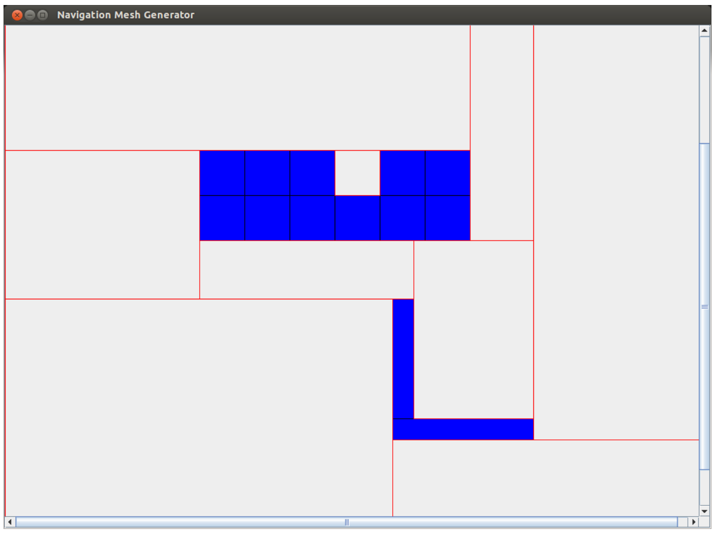
The second alternative to the grid-based technique is the Navigation Mesh technique described in Cui and Shi (2012). It represents the walkable area of the map as a series of convex polygons. This reduces the pathfinding problem to finding a path between the polygons, as it is possible to travel via any two points in a convex polygon via a simple straight line. The convex polygons can represent a much larger area of the map and allow the walkable area represented to be as accurate as possible. It is also a lot faster than the grid-based method because there will be fewer nodes in the search space. In order to turn the path across the polygons into a complete path for the map, a varying amount of computation is required, depending on the compromise made between the perfection of the path found and the time taken to find it. The options include travelling via the midpoints of the edges connecting each polygon and employing the Funnel algorithm to find the optimal path through them.



Navigation Mesh

Navigation Mesh

Implementing a Navigation Mesh system within Animageddon was a fairly simple task due to the fact that all of the obstacles in the game are rectangles, which means that the mesh can be made entirely from rectangles, avoiding a lot of complicated maths. However, some work was required in order to make the pathfinding algorithms run in a timely manner, if the algorithm takes 10 seconds to run, the goal position is likely to have changed, and the path would have to be recalculated. In order to make the game perform adequately at run-time, a Navigation Mesh Generator tool was created, it makes the modification, or addition, of maps a large amount easier. By calculating the edges connecting neighbouring polygons and their midpoints in the generator, the games performance stays relatively high. The navigation mesh technique also provides the advantage that the pathfinding algorithm only needs to be run if the start and goal points are in different polygons, the AI can simply calculate the straight line to the goal point, which can save precious CPU cycles.



Finding a Polygon Path

Animageddon uses an implementation of the A* pathfinding algorithm to find a route from the start polygon to the goal polygon. The implementation calculates the g value, the actual cost to get to each node, by calculating the length of the path between the edge midpoints. It calculates the h value, the heuristic estimate of the future-path cost, by working out the straight line distance from the last edge midpoint in the path to the goal directly.

Finding a Complete Path

Once the polygon path has been found, it needs to be converted into a set of points that the AI character can follow. There are many ways to do this; including travelling via the polygon centers (where possible), travelling via the midpoints of the edges connecting the polygons, and travelling via the set of polygon vertices that form the shortest path from the start to the goal. Due to the edge midpoint methods ease of implementation and the time restraints of the project, this method was chosen. It forms acceptable paths and saves the computation time spent finding the shortest path using the Funnel algorithm explained in Mononen (2010).

Networking

Overview

Animageddon uses networking for multiplayer gameplay. Since the game is real-time, all clients need to be synchronised and should ideally have exactly the same information about the game world and its entities as all the other clients. For example, when an entity's position changes, this should be relayed to all clients with a minimal time delay. In practise, there are delays in networking this information and a lot of time was spent on minimising the effects of this.

Implementation

Networking is done using the TCP protocol. This ensures game information is transmitted correctly and in the correct order. UDP was considered for its speed benefits but with the time provided, it was considered too risky and it was unlikely that a robust application protocol could be created without having to sacrifice a significant amount of time on other features.
All game entities in the world are networked. By default, basic properties such as the position are transmitted. Entities can specify more properties that require networking using the NetworkedEntityField class. These fields are transmitted to clients whenever their value is changed.

Problems Encountered

Keeping all clients synchronised is far from simple when network delay is introduced. Consider a user pressing the right arrow key. Feedback should be immediate for the user and so their screen will display the player moving to the right on this event for as long as the key is down. This key press is also transmitted to the server, but let there be a 500ms delay. By the time the server receives this key press information, the user will have already simulated 500ms of movement in the right direction and its position will be different to what the server has stored. The server begins moving the player in the right direction but it is slightly behind the position the user is seeing. Furthermore, let there be a 500ms delay between the transmission of this data from the server to the other clients. The other clients are now seeing a different position to the server as well as the user controlling the player. This caused issues when implementing the code for shooting and visible moving bullets had to be sacrificed for instant shoot damage as a result. Well known techniques explained in Gambetta (2014) including client-side prediction and interpolation were required to counter other issues brought about by network delay.

Software Engineering

Potential Models

There are many Software Engineering models that could have been employed in order to develop the Animageddon project. Choosing the right method was an important decision because the wrong method could seriously affect the outcome. For instance, a method that doesn't allow the requirements to change flexibly, such as a waterfall model, may mean that the submitted product is incomplete or far too unambitious.

Agile Model

The Agile Software Engineering Model was used for the Animageddon project. This was chosen because of the flexibility it allows for the evolution of requirements. It was likely that this flexibility would come in use in the case that the original requirements could not have been met by the deadline and the requirements had to be shortened, or if the original requirements were met before the deadline and the project requirements could be extended. Because nobody in the team had worked on a team-oriented software project before it was unlikely that the amount of time taken to fulfil the original requirements would accurately match the amount of time allocated for the project.

Object Oriented Design

The Animageddon software was written in the Java programming language; a language heavily focused on object-oriented design. The Entity and World classes make strong use of inheritance and polymorphism to save time and duplicated code. Allowing an entity to inherit the functionality in the Entity class saves having to write the functionality multiple times, which can prevent the opportunity for bugs to arise as well as having to make the same modifications to code in multiple places. Abstraction allows multiple classes to be dealt with in the same manner, despite having different implementations of the same methods. A good example of this is the update and render methods of the entities. Each entity must be updated and rendered in a different way. By making them all implement the abstract update and render methods in the Entity class, a list of entities can be updated and rendered in just a few lines of code.

By keeping the code modular, it is clear to see what each class does and how it should be interacted with. It also causes the code to be extensible, enabling future additions to the game. Moreover, it allows each member to work on their own code without needing to worry about the functionality of other peoples code.

Risk Management

Risks were identified on a weekly basis in order to ensure that any potential risks could be prevented and any that had already occurred could be mitigated as soon as possible. One of the ways these risks were identified was by calculating Animageddon's critical path. In other words, the sequence of tasks that, if not completed in immediate sequence, would delay the entire project. This method identified that the Artificial Intelligence was at risk of not being completed by the project deadline, mitigations were easily put in place by assigning more resources to this task.

Effectiveness

The Agile model that was employed turned out to be very effective. It allowed the requirements to be changed when it was realised that there was insufficient time to complete some of the more ambitious features first described in the specifications document, such as the animated movement of the animals. It also provided the ability to add small additional features that were not included in the initial specification, such as sound.

Evaluation

Overall the project was a positive experience for the team. Weekly meetings and constant communication via social media helped the productivity of the team significantly. The entire team got on with each other and no big disagreements or disputes occurred at any point.

By keeping the code as clean as possible and making refactoring changes to improve code quality as we went along, the addition of new features, or modifications of old ones, was generally a simple process. Writing documentation at the same time as writing code meant that it was easy to interact with the code written by others.

An important factor of the projects success was that all the code for the project was kept integrated with the other code throughout the entire project. This avoided any issues with code integration where the two pieces of code used two different interfaces, and generally kept the code quite clean.

The majority of tasks in the projects critical path were completed well within time. However, despite the Artificial Intelligence being identified as a risk early on, we failed to mitigate the risk as soon as possible. Unfortunately, this meant that it had to be completed within a short amount of time and could not be finished to the highest possible standard. More planning at the start of the project and proper assignment of tasks to team members could have prevented this risk from evolving into an issue. The complexity of networking was also overlooked in the design stages and it may have been wiser to have started development on it earlier on in the development schedule to allow for more time to perfect it.

Team Work

Organization

As a group, we chose to give each team member equal responsibility and a project manager was not allocated. This worked fairly well because all team members had their own motivation to work on the project and were capable enough to not require leadership. It was decided that if a disagreement between team members were to arise, a group vote would be taken. For a final decision to be made, three out of the five members must consent to it. This ensured quick, fair and effective decision making.

Problems

The flat organisational structure described above worked really well for us and we didn't have any significant issues within the team. Minor issues arose when we had conflicts in our working schedule and therefore we could not meet at a time that suited us all. However, we worked around this via other means of communication such as Facebook and Skype. This allowed us to discuss our work and solve scheduling clashes so that we were not wasting time waiting for everyone to be free.

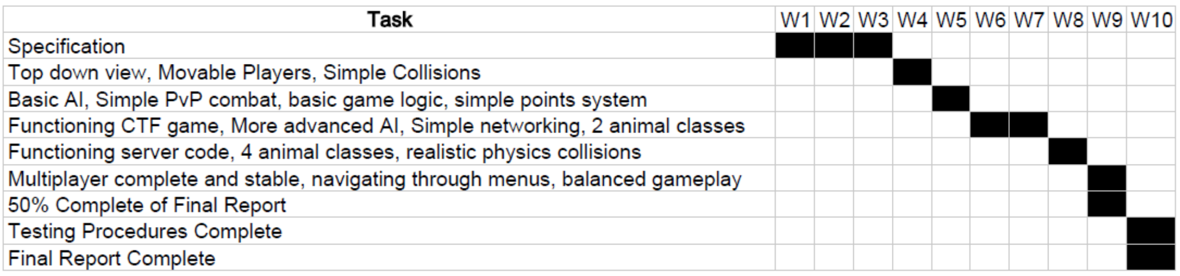
Teamwork System

The teamwork system was used entirely for the logging of how many hours we were doing on our project. This let each member check that enough work was being performed by each person. It also allowed us to see what we were spending most of our time on and if the time was used as efficiently as possible. For example, if one member saw that they had not enough work to do to use their time effectively, as a team we could decide how they can extend their work to meet the requirements, or assign them different tasks so that they are working towards a functional goal.

Work Breakdown

Barnaby Colby	• Loading map from file • AI
Arun Bahra	• GUI system • HCI aspects
Richard Maskell	• Player shooting and health • Local player input • Game Sound • JUnit testing • Test report
Tom Whale	• Game graphics • GUI features such as keyboard event listener • JUnit tests • Player classes

Gantt Chart

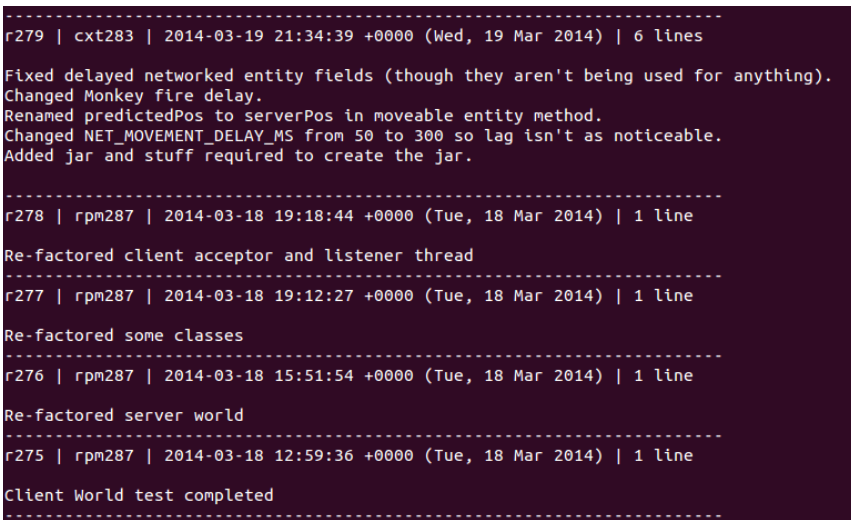


Task	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
Specification	■	■	■							
Top down view, Movable Players, Simple Collisions				■						
Basic AI, Simple PvP combat, basic game logic, simple points system					■					
Functioning CTF game, More advanced AI, Simple networking, 2 animal classes						■	■			
Functioning server code, 4 animal classes, realistic physics collisions								■		
Multiplayer complete and stable, navigating through menus, balanced gameplay									■	
50% Complete of Final Report									■	
Testing Procedures Complete										■
Final Report Complete										■

This schedule was designed at the start of the project and was referenced to throughout the development process to make sure the project was on track. We slightly deviated from the original testing time allocated and decided to begin testing procedures as soon as we could.

SVN Activity

We made over 330 commits to the svn repository during the 11 weeks we spent on the project. Most of these were fairly small commits, for example, fixing small bugs or documentation mistakes. Committing in small frequent chunks meant that we were able to keep each others code integrated with minimal effort. We had to make sure that the code we committed was working code to avoid causing any problems for our team-mates. For the Artificial Intelligence part of the game, this meant that a large commit was necessary when it was completed so that the game was still playable for the rest of the team. A small subsection of the svn logs are shown below, to allow the reader an idea of the kind of commits that were made.



```
------------------------------------------------------------------------
r279 | cxt283 | 2014-03-19 21:34:39 +0000 (Wed, 19 Mar 2014) | 6 lines

Fixed delayed networked entity fields (though they aren't being used for anything).
Changed Monkey fire delay.
Renamed predictedPos to serverPos in moveable entity method.
Changed NET_MOVEMENT_DELAY_MS from 50 to 300 so lag isn't as noticeable.
Added jar and stuff required to create the jar.

------------------------------------------------------------------------
r278 | rpm287 | 2014-03-18 19:18:44 +0000 (Tue, 18 Mar 2014) | 1 line

Re-factored client acceptor and listener thread
------------------------------------------------------------------------
r277 | rpm287 | 2014-03-18 19:12:27 +0000 (Tue, 18 Mar 2014) | 1 line

Re-factored some classes
------------------------------------------------------------------------
r276 | rpm287 | 2014-03-18 15:51:54 +0000 (Tue, 18 Mar 2014) | 1 line

Re-factored server world
------------------------------------------------------------------------
r275 | rpm287 | 2014-03-18 12:59:36 +0000 (Tue, 18 Mar 2014) | 1 line

Client World test completed
------------------------------------------------------------------------
```

Summary

Software Design

The early software design phase of any project is an important stage in its overall success. A good design ensures that the process of adding basic functionality and additional features is as simple as it can be. This can save precious time and can help reduce the potential for bugs because the code is clearer and more understandable. Animageddon took careful steps to ensure that this stage was as good as it could be by choosing the same design patterns chosen in industry and carefully researching the benefits they offer. This set the Animageddon project up for success from early on.

Game Design and HCI

By carefully designing the game and ensuring that it was easy for a user to interact with it, Animageddon has, again, set itself up for success. Menus that are hard to navigate, and gameplay that is difficult to see what is going on can easily cause the user frustration. Failure to hook the user quickly can result in failure of a project. Once again, the research and careful design taken to create Animageddon has ensured that any source of frustration due to game design or HCI has been reduced or removed.

GUI, AI and Networking

The game uses sophisticated and industry standard techniques, practices and algorithms in order to achieve a high performing and intelligent piece of software. This enables the user to enjoy a game that runs at high speed, is fun to play and can provide an intriguing challenge to the user. Keeping the user interested is one of the most important features a game can offer, both performance and difficulty need to be perfected in order to give the user the best possible experience.

Software Engineering

The methods chosen for the Software Engineering process of Animageddon allow the game to evolve as a product. Separating concerns by using modular code allowed us to complete the project faster, integrating the features as we went along. Our extensible design choices mean that the game can easily be extended to improve the users experience at minimal cost to the developers, whether this be time or money.

Team Work

The team has collaborated well together and have been able to utilise each other’s strengths in order to overcome the problems faced and improve the final product. By splitting the work up, each team member has been able to contribute effectively, without knowing much about how some of the other code works. Animageddon has benefited from this, and is a strong piece of software on all fronts.

Individual Reflection

Thomas Whale

The Java Team Project has been a fantastic experience for me. By having no previous experience in game programming or design, or in-fact intense playing, this was a full learning curve. I have always been interested in dabbling in certain features of game design, but never knew how to begin. Working within this team made me learn areas that I was not always familiar with and let me strengthen areas that I was.

I found that the team project really stressed how important communication and discussion is within a project environment. The use of various media channels (including social media & systems of teamwork) complimented well with face-to-face meetings; and this team project helped myself understand this.

Furthermore, by never being part of a group project before (in or out of uni), the use of a version control system allowed me to fully understand SVN. It also made me realise the full potential of these systems.

In addition, I have never had experience in game programming or openGL prior to this project - causing this to be a real learning curve and letting me understand from the ground up how a game is built. Prior to this project also, I had never experienced using JUnit testing - which made self-teaching myself these methods highly rewarding and engaging. It also showed me how important test-driven coding is.

I also found it extremely interesting to see how others worked, planned their time and coded during the entire project which helped me improve my working patterns and become more productive. The whole experience gave me a real insight to working in industry as a team - having weekly meetings with a set schedule, using official tools such a teamwork to log hours.

In conclusion, the last 11 weeks have taught me alot from programming through to project management, and is definitely something that can not be replicated without real team members.

Christopher Thorne

Working on a game in a team has been a very beneficial experience for me. As someone with previous experience in game development, I commonly find myself abandoning game projects that I develop on my own due to lack of motivation or lack of interest in implementing certain features. I have always wondered if things would be different if I were in a team environment, surrounded by other passionate developers working with the same goal in mind of creating a good product and this project has taught me that indeed, developing in a team is an entirely different experience to developing individually.

Projects can typically be divided into a number of modules that provide specific functionality. These modules are linked together to do some useful task. Some examples of these modules in our game are networking, physics, AI and GUI. Personally, I am most interested in networking and physics and have experience in these fields. Being in a team gave me the option of focusing solely on these modules whilst my team members worked on the other modules that were not so suitable for me. Not only did this result in the project actually being completed, it also led to more robust networking and physics code as I did not have to spend time worrying about other parts of the system beyond how they interact with my code. Making sure the modules interacted with each other correctly was more difficult than it would have been had I been working alone, however, this did not cause any major problems due to robust integration testing carried out by another team member.

Another part of working in a team I found particularly interesting was the software engineering process. Talking my ideas through with other developers and explaining the reasoning behind my design decisions made it much easier to spot any problems or things I had missed. Normally these things would not become clear until late into the implementation phase, at which point it is not always easy to adjust the design. Similarly, explaining problems I found in my teammate’s designs without coming off as too critical or aggressive was an interesting challenge.

Overall I am satisfied with how the project turned out and would happily work with my team members again.

Barnaby Colby

This project has allowed to me to gain valuable insights and experience that will become useful throughout my future career. Some of the initial discussions about the structure and class design of Animageddon enabled me to see how some of my initial thoughts could be improved by combining or replacing them with other team members ideas. When I struggled to choose between implementation options I found that the other team members were useful resources to help make the choice. I believe that the game has turned out better through this process than if I was to design it by myself.

Whilst I have used version control systems before, such as Git, it was definitely useful to see how they can be used in a team-oriented manner, allowing everybody else to work on different parts of the project simultaneously without causing them problems by modifying the code I was working on, and allowing easy integration of others work even before completion of my own.

It was an interesting experience to not know how some of the code for the game worked, but still being able to interact with it by looking at the documentation. When working by yourself, documentation can be pretty useless as you already know how to interact with the code you wrote. I can now see how powerful documenting code can be, the small amount of effort required can pay dividends later on.

Interacting with our tutor in weekly meetings helped me to see how identifying risks and achievements can help keep the project on track, I also learnt some basic project management techniques in order to identify these risks.

I also noticed how valuable code testing can be. It can easily save hours of tedious debugging and frustration when finding subtle errors. The test-driven approach can also lead to simpler code, by writing tests one by one and modifying the code until it passes it, instead of going straight to the sometimes more obvious and complicated solution. This reduces the likelihood of bugs.

Arun Bahra

My overall experience on this assignment has been very positive. I believe that I have been able to contribute effectively throughout the duration of the project and I have gained valuable knowledge for future team projects.

The most interesting part for me was the integration of the teams code throughout development. To see how people with similar skill sets design and implement their work was very thought provoking because it made me think how I can complement what they had done. In addition to this, the opportunity to use version control has been very useful as it is used in industry by most companies.

The use of an agile design methodology has given me an insight into how changing requirements can affect the design process. It has also taught me that in a team you don’t have code ownership; even though you designed it, the code belongs to the team and it can be altered for the benefit of the team.

As my role was to implement the GUI, the use of openGL has been invaluable as it is a widely used in the community. Although it was difficult to use in the beginning, I have gained a lot of experience in the way the methods and features should be implemented and have a better understanding of common errors.

I thought my team members were excellent and would definitely work with them again. We communicated with each other about any issues we had and dealt with them appropriately.
In today’s industry, emphasis is put heavily on working in a team and the competencies that you have used. I believe this module has been very worthwhile in gaining experience that can be applied in real world situations.

Richard Maskell

I believe this project to have been a valuable experience in the dynamics of developing software as part of a team and particularly how this can change from one team to the next, having developed software in a team before. This stressed to me the importance of adapting to new teammates and methods of working as well as being open to new ideas.

I also liked the degree of freedom we were given over our project and how we went about doing it and I feel that as a team we developed an efficient method of programming with everyone working well together.

Furthermore, I am glad that I got to use tools different to those typically used in other Java projects such as the LWJGL graphics library and I feel this has broadened my knowledge base, for which I am grateful.

Finally, I enjoyed getting to use JUnit tests in a realistic environment which emphasized the ease and flexibility of such tests which I plan on using in future projects.

References

Gambetta (2014), “Fast-Paced Multiplayer”, http://www.gabrielgambetta.com/fast_paced_multiplayer.html [accessed 28 -

Cui and Shi (2012), “An Overview of Pathfinding In Navigation Mesh”, http://paper.ijcsns.org/07_book/201212/20121208.pdf [accessed 28 March 2014]

Mononen (2010), “Simple Stupid Funnel Algorithm”, http://digestingduck.blogspot.co.uk/2010/03/simple-stupid-funnel-algorithm.html [accessed 28 March 2014]

Bilas (2002), “A Data-Driven Game Object System”, http://gamedevs.org/uploads/data-driven-game-object-system.pdf [accessed 28 March 2014]

Bibliography

Aspinall (2007), “Interface Design Rules”, http://homepages.inf.ed.ac.uk/da/teach/HCI/rules.pdf [accessed 28 March 2014]