Solent University

SCHOOL OF MEDIA ARTS AND TECHNOLOGY

Computer Games (Software Development)

2018

Rowan Strafford

“Animal Behaviours Driven by Learning”

Supervisor	:	Ken Pitts
Date of submission	:	May 2018

1. Abstract

The concept behind this project was to design and implement a Behaviour Tree to simulate actions and behaviours that an animal would carry out. The tree would be powered by a Machine Learning algorithm allowing for an adaptive output for each individual entity and for allowing animals to interact with each other. DirectX 11 has been used to allow for future development to be made in 3D and for its current graphical 2D capabilities.

The end result is a simulation that allows entities to perform actions driven by a Behaviour Tree whilst providing the ability to create and place entities at runtime.

Contents

Table of Figures

Table of tables

2. Progress Report

2.1. Application Introduction

I will be creating an application used to demonstrate how animals behave when interacting with other animals. I will be using a modified A* pathfinding (Patel, 2017) algorithm for the movement within the program, machine learning for the adaptive side and behaviour trees to connect everything up. Animals will be placed in a level by the user where they will interact with other animals as well as objects. Animals will hunt, flee, eat, remain idle or carry out any further actions I decide to implement. I will be using C++ with DirectX11 (Microsoft, 2018) in Visual Studio 2017 (Microsoft, 2018) to program this application.

2.1.1. Subject Overview & Scope

I am aiming to implement a fully working application that allows animals to be displayed, act out their lives and to learn and adapt from behaviour they perform. An example by Dmitry Shesterkin's (Shesterkin, n.d.) shows a similar concept to this using bird flocking. Although this version of bird flocking may not use machine learning or specifically a behaviour tree, it does provide a good practice in how entities can swarm together in movement, a concept I am planning on implementing.

The project I am working on will allow me to demonstrate how AI can be implemented to give entities life, seemingly. The solutions and concepts that I will work on along the way will have the ability to be implemented into other projects if the outcome of this project fits the program.

The scope of my application would be defined by having various different entities representing animals, each of which interacting with the environment in its own way. These entities have the ability to come into contact with other entities, at which point certain actions will be triggered based on the individuals visual perception. Using a view cone, each entity will scan and read the environment, making decisions based on size, distance and previous knowledge of the perceived data. The user can interact with the environment by adding animals at a chosen location.

2.1.2. Project Aims

- I want the entities actions to change over time, driven by learning.
- I want the application to run at a high number of frames per second, with finished graphics and with a user friendly interface.
- I want all entities to derive appropriately from base classes to allow for any future features to be implemented with ease.

2.1.3. Project Objectives

I have a few objectives that will need to be worked through in order to have a working application, these objectives include:

- Create simple level with obstacles to use.
- Create the base classes and create a simple animal in the level.
- Begin implementing a behaviour tree to control an animal.
- Implement basic movement, moving onto the A* pathfinding algorithm.
- Create the user interface allowing for animals to be placed.
- Incorporate the senses starting with sight.
- Allow animals to have different behaviours to one another.
- Implement animal learning.

2.2. Research

2.2.1. Animals in Nature

Sandeep V. Kharkar states that “You must realize that you are trying to imitate nature in your game and recreate the experience the player has had with real animals” (Kharkar, 2002). I observed how birds flock together and move in a given direction as well as how they gain their food by diving down into water below one by one. This information is relevant to an animal that has the ability of flight but behaviours such as hunting are transferable amongst species.

Kharker (Kharkar, 2002) also describes the three categories of animals within games and how they can be used to create a realistic and atmospheric effect using animals alone. The first is Ambient Animals, which are entities that are used to enhance the environment without coming into contact or effecting how the game is played. An animal that would fall into this category would be a bird flying overhead. Secondary and Primary Animals are the last two categories of animals. These entities interact with the player or other entities with the only difference being that secondary animals not having as much interaction as the primary ones. A Primary Animal may be the sole focus of the game which in turn might interact with a smaller animal that does not interact with the player. An example of this would be a Lion hunting Rabbit, where the player can kill the lion but not the rabbit.

This information taken from AI Game Programming Wisdom is useful for understanding how animals might be used to give a more life-like feel to a game but the actual implementation of these features such as the three different categories may not be appropriate for an application like mine, where there is no interaction between a specific entity and the user, as the user will not have a character to play as.

2.2.2. A* Pathfinding Algorithm

This algorithm focuses on moving an object from one node to another with the ability to navigate obstacles and take all paths into account. It is a very widely used algorithm and is featured in a number of different games in a number of different ways.

Mathews (Mathews, 2002) says “A* traverses the map by creating nodes that correspond to the various positions it explores”. The algorithm uses three main attributes which help calculate where in the map can be traversed. Mathews (Mathews, 2002) describes these as:

G – The cost to get from the starting node to this node.

H – The estimated cost to get from this node to the goal.

F – The sum of G and H.

The diagram below, see Figure 1, shows an example of how the A* pathfinding algorithm has been implemented, with the large cantered number representing the f value, the top left representing the g cost and the top right representing the h cost. The algorithm starts by following the node with the lowest f value until it either arrives at its destination of reaches a point where it can no longer move. If there are multiple nodes, each of which has the same f value, then the node with the lowest h cost will be chosen. Nodes will continue to be explored, choosing the node with the lowest f value, not necessarily following the nodes in a line, meaning that nodes uncovered earlier can have their surrounding nodes revisited and explored.

This algorithm is very efficient and can be modified to include features such as smoothing and grouping, however the algorithm has issues dealing with optimisation and remaining memory efficient. I am relying on my ability to simplify the algorithm in terms of memory use to allow each entity to work with the A* pathfinding algorithm.

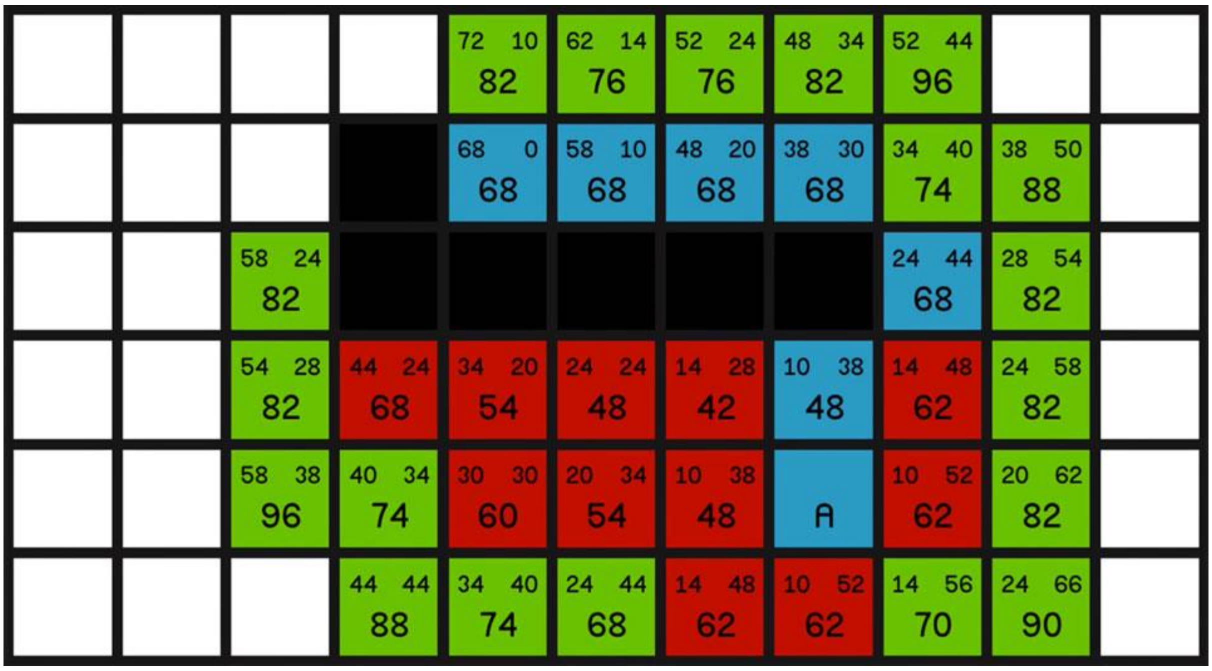


Figure 1: A Pathfinding example (Strømsvik, 2018)*

The need for using A* within my project is key as all entities will need to know where in the map they can travel. This will also link in nicely with incorporating a grid as certain areas of ground within the map will be edible to some animals. An example of this will be a Rabbit who needs to eat, a grass tile would be ideal in this situation, therefore if the Rabbit knew it needed food it could make its way to the nearest tile occupied with grass.

2.2.3. Behaviour Tree

Behaviour trees are “Data-driven structures that can be easily represented in the form of a tree or graph” according to Sergio Ocio Barriales (Rabbin, 2015). This perfectly sums up how I want to approach my project, with a structure that can be easily understood and mechanics that are driven from data.

Behaviour Trees use a set of nodes, including leaf nodes and branch nodes, to allow the application to flow between different actions. These nodes can be structured in a way that allows the developer to have full control over what actions are released at what stage, if they are reached at all. Simpsons

(Simpson, 2014) describes three main nodes, the composite, decorator and leaf. See descriptions below and Figure 2.

Composite – “A composite node is a node that can have one or more children”.

Decorator – “A decorator node, like a composite node, can have a child node. Unlike a composite node, they can specifically only have a single child”.

Leaf – “Leafs are however the most powerful of node types, as these will be defined and implemented by your game to do the game specific or character specific tests or actions required to make your tree actually do useful stuff”.

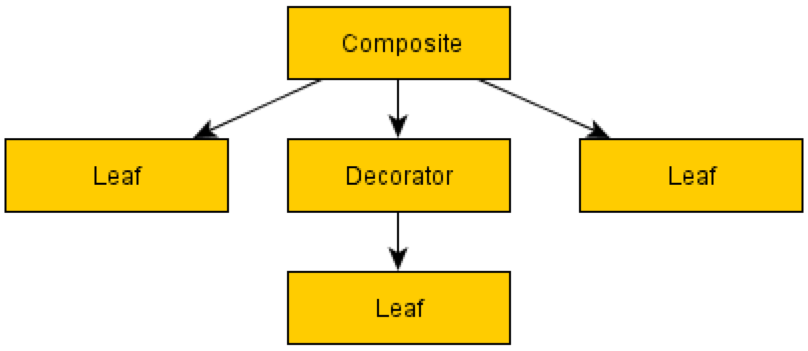


Figure 2: Behaviour Tree Representation (Simpson, 2014)

These nodes work together with the code by allowing only one node to be used at once as the code traverses the tree. A node will return a status value when it is updated, either Success, failure or running. The program will use this information to allow for further exploration of the behaviour tree. This return value will be used alongside the learning concept that I will implement to allow for nodes to have success and failure values change over time. An example of where I will use this is having an animal searching for food, such as a Cat, an obligate carnivore. The Cat may eat grass resulting in the animal being sick, therefore next time this area of the behaviour tree is run, the option to eat grass will not be chosen, as this resulted in a failure value on the previous cycle.

A key issue when dealing with complex Behaviour Trees, such as mine will be, is keeping it manageable. To tackle this problem I will allow all leaf nodes to be reused for different tasks of a similar nature such as moving towards a given location instead of moving towards a tree. In Game AI pro 2 (Rabbin, 2015) it states that,

> The most important node of the example tree is the selector. Selectors allow us to represent conditionals in BT; translating it to what we could do in a programming language such as C++, selectors are the BT version of an “if-then-else” construction.

This statement leads me to believe that using a Behaviour Tree will be very beneficial in regards to my project as it will allow me to have constant checks represented as a selector to navigate the tree. A Behaviour Tree is also a concept that could have learning built into the core of it, allowing me to implement learning within a behaviour tree, reducing the complexity of the project.

2.3. Project learning solutions

Here will be determines whether or not implementing a learning algorithm will be an appropriate decision though three different solutions.

2.3.1. Solution 1 – Q-Learning

Q-Learning is a technique used to allow for unmanaged learning until an optimum solution is achieved. The rewards and processes that are chosen to be carried out determine how an entity will adapt giving the ability for chosen entities to learn at a slower rate than others.

Although a fully implemented Q-Learning algorithm will provide evidence of learning as my application is ran, the time it would take before any significant changes were made to the entities would be very large. This is due to how the algorithm uses continuous rewards as opposed to choosing when the entity has done something correct. Q-Learning also has issues when dealing with a large number of learning objects, as each object will have its own calculations to work through after each reward.

John Manslow (Manslow, 2002) discusses the potential issues with the Reinforcement Learning algorithm Q-Learning stating that

> Reinforcement learning generally adapts only very slowly, and hence often requires very large numbers (e.g., tens or hundreds of thousands) of performance evaluations. This problem is often overcome by evaluating agents against each other.

This argument tells me that Q-learning alone will not be a sufficient algorithm to use as it will take too long for the learning side to adapt, to the point where I would need to speed the application up to increase the number of learning iterations. Instead I will research alternate solutions to Reinforcement Learning that focus on the principles instead of a hard coded algorithm.

2.3.2. Solution 2 – Neural Networks

A neural network is an approach on machine learning used to tackle complex tasks similar to those in real life. This is why neural networks have been designed from the human brain, based of neurons and their synapses. This algorithm uses a network constructed of these neurons and their connections, which can be weighted to give certain advantages to situations.

This concept uses multiple layers including an input and output layer with a hidden layer in-between. The input layer deals with what will be read into the algorithm such as line of sight or what is heard by an entity. This information will be used in the hidden layer to determine the outputs at which point one will be chosen and carried out based on how the algorithm is implemented.

As a neural network works by adapting the play style as mistakes are made, this approach may not be credible for this project as this may result in animals getting themselves into situations that they cannot get out of. This idea is not a problem if only a few animals are being effected this way but if every animal dies early on from lack of learning important information then this will be a problem. A solution to this would be to allow animals to learn from witnessing other animals behaviour, therefore animals in a pack will learn from the actions of others in their group and adapt this way.

2.3.3. Reinforcement Learning principle

This learning concept bases its workings on adapting behaviours in order to improve performances using chosen factors to influence an agents actions. An agent, in my case an Entity, will have basic actions is can carry out and have knowledge of. These actions will lead them into situations that demand a decision to be made, such as hide if an enemy is in sight or flee if an enemy is attacking. (Palmer, n.d.)

Four fundamental parts are used within each agent to help it adapt to the environment and discover its capabilities. These include:

- **Learning element** – The section of the agent that alters the behaviours and allows for improvements to be made.
- **Performance element** – The area that deals with making decisions, usually determined from what the agent has seen or has gained knowledge of.
- **Curiosity element** – As stated, this element gives an agent the effect of having a curious mind by allowing the agent to make decisions that may not be safe or suitable. These decisions stop an entity from performing basic actions and getting stuck doing the same tasks more than once.
- **Performance analyser** – Using the performance element, future decisions are taken into account based on how the agent dealt with the current situation, altering the performance element as it does so.

This learning method has been used commonly in situations where trial and error is a big feature, however for this to become a solution for my project I will have to adapt the way it is constructed to allow agents to learn using more trial and less error, as error may result in premature entity death. Basic instincts built around sight and hearing can be used to resolve this issue by allowing entities to make decisions using these instincts alongside the four elements that make up Reinforcement Learning.

Reinforcement Learning would be an appropriate solution to use as it allows decisions to be made by an agent using error correction, a trait that animals also follow according to Yael Niv's The Neuroscience of Reinforcement Learning. (Niv, 2009) He also states that "Animals will sometimes work for food they don't want", another principle that can be implemented using this learning concept named Devaluation. An issue that may arise could be how the behaviours and actions of an agent are executed, in a way that they are repetitive and predictable. This can be caused by an agent finding a set of actions that keep it safe and then only performing these to stay alive. This is where alterations to Reinforcement Learning may have to be made.

2.3.4. Chosen Solution

Q-Learning will not be used to tackle the learning within the project as entities will need to be rewarded at specific moments when a task is either completed or failed. By rewarding the entities when they have done something correct they will adapt to repeat these processes whereas if an entity does not complete its task, such as hunting, it will evaluate why and improve on this. As the application will support the ability to add a large number of entities to the scene, Q-Learning would become very memory intensive as more entities were added.

Using a neural network is an appropriate response to this learning solution as it would allow me to incorporate advanced life-like behaviours with smooth progressive learning, however a modified version of this algorithm will have to be used to reduce the time it would take for each entity to learn.

The more realistic approach will be to use my own modified version of Reinforcement Learning built into a behaviour tree. The principles taken from this concept such as the curiosity element etc. would be what determines how leaf nodes are selected. Over time attributes such as speed and bravery would increase or decrease leading to different nodes being selected. An example can be seen in Figure 3.

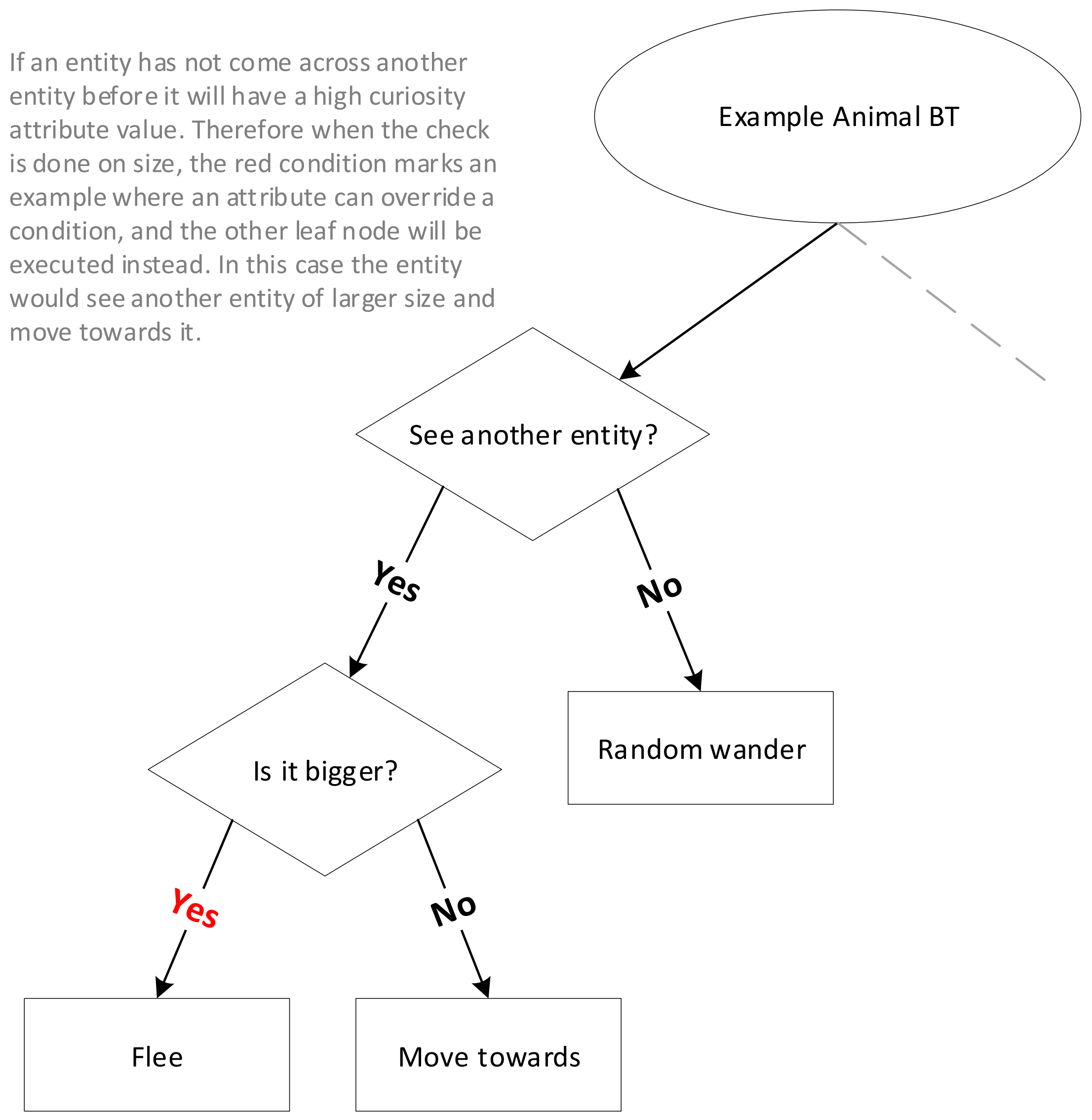

If an entity has not come across another entity before it will have a high curiosity attribute value. Therefore when the check is done on size, the red condition marks an example where an attribute can override a condition, and the other leaf node will be executed instead. In this case the entity would see another entity of larger size and move towards it.
Example Animal BT
See another entity?
Yes
No
Is it bigger?
Random wander
Yes
No
Flee
Move towards

Figure 3: Behaviour Tree Example

2.4. Project Tool Solutions

The research and chosen solution for the project tools can be found in Appendix 2.

2.5. Project Specification

- The display will consist of two individual windows, the User Interface using the Win32 API and the graphical window using DirectX11.
- The camera will be in an orthographic view, consisting of different layers to allow certain elements to be rendered on top.
- The A* pathfinding algorithm will be implemented to deal with navigation.
- Entities will learn based on win and loss cases, with decisions being made through a behaviour tree.
- Each entity will perform its own actions without the use of a manager.
- The agent learning will be based on Reinforcement learning.

2.6. Risk Assessment

Here is the risk assessment graph that details possible risks that may arise during the entirety of this project. An impact of each risk has been included alongside the probability and a brief description of how these risks can be mitigated.

Risk	Impact	Probability	Preventative measures
Becoming ill	High	Low	Eat well and rest for long periods.
Hardware failure and corruption of files	High	Low	Back up all work regularly using GitHub.
Lack of experience in machine learning	Medium	High	Read into all avenues of machine learning specifying in Reinforcement learning.
Lack of web access	Medium	High	Obtain books and download papers relating to the project for when there is no internet.
Excessive work load	Medium	Medium	Have work to fall back on if the project becomes too much to handle.
Unexpected travel	High	Low	Follow the Ghant chart and insert times where I will be away.
Bereavement	High	Low	Keep on top of the workload.
Failure to implement main mechanic	High	Low	Plan how the mechanic will be implemented and make sure that the workload isn't too large.
Implementing unnecessary features	Medium	Low	Stay on track with the Work Breakdown Structure and Ghant chart so only the required work is completed.
Insufficient testing	Medium	Low	A test plan will be used to ensure that no areas of the application are left untested.

Table 1: Risk Assessment Table

2.7. Project Management Methodology

Due to the nature of this project I will be using a Prescriptive Framework, a hybrid between a waterfall and agile approach. This consists of four main phases, the definition and progress reports, the project mid-point and the final product. The phases can be broken down as follows:

Definition: This describes the direction of the project and defines the scope in a clear and detailed way.

Progress: This includes all necessary planning and research that will be carried out before the implementation is underway.

Project Mid-Point: This marks the start of the final implementation, where an application should have a clear end goal. A prototype should have been completed by this point.

Final application: This is where the final documentation is done for the project and a fully polished application is produced.

This approach fits in extremely well with the style of implementation I am going for as I will be able to fix issues as they present themselves along the way while implementing smaller added features if the work load is not too heavy.

Another methodology that I have worked in before is the SCRUM method, which focuses on setting the workload into smaller manageable sections called Sprints. (scrumalliance, n.d.) Once a sprint is complete the next is tackled and so on until the project is complete. This principle is an appropriate one but is more focused around teams where there is more than one member. For me to work on a methodology that is based around team working would not be a sensible decision for this project and with the time frame I have been given, an agile/waterfall hybrid would appear appropriate. See Figure 4.

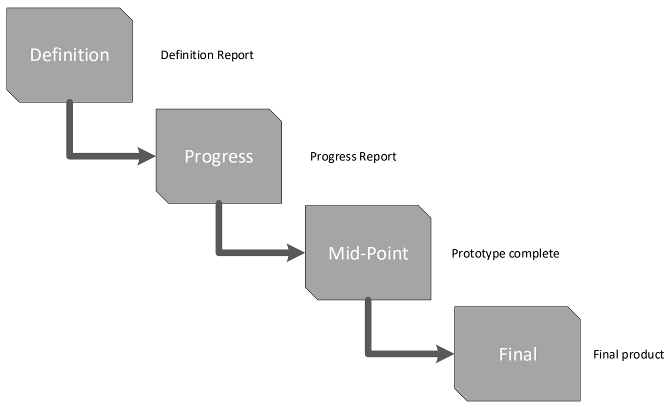


Figure 4: Prescriptive Diagram

2.8. Work Breakdown Structure

This Work Breakdown Structure diagram illustrates how and in which order certain tasks will be tackled. A section on planning has been included to remind me of what is needed to be done. Although the tasks could appear vague to an outside user, within this context I have complete understanding of what each task means. See Figure 5. If this diagram is difficult to view, there is a larger version within the Resources folder of this project, named WBS.vsdx.

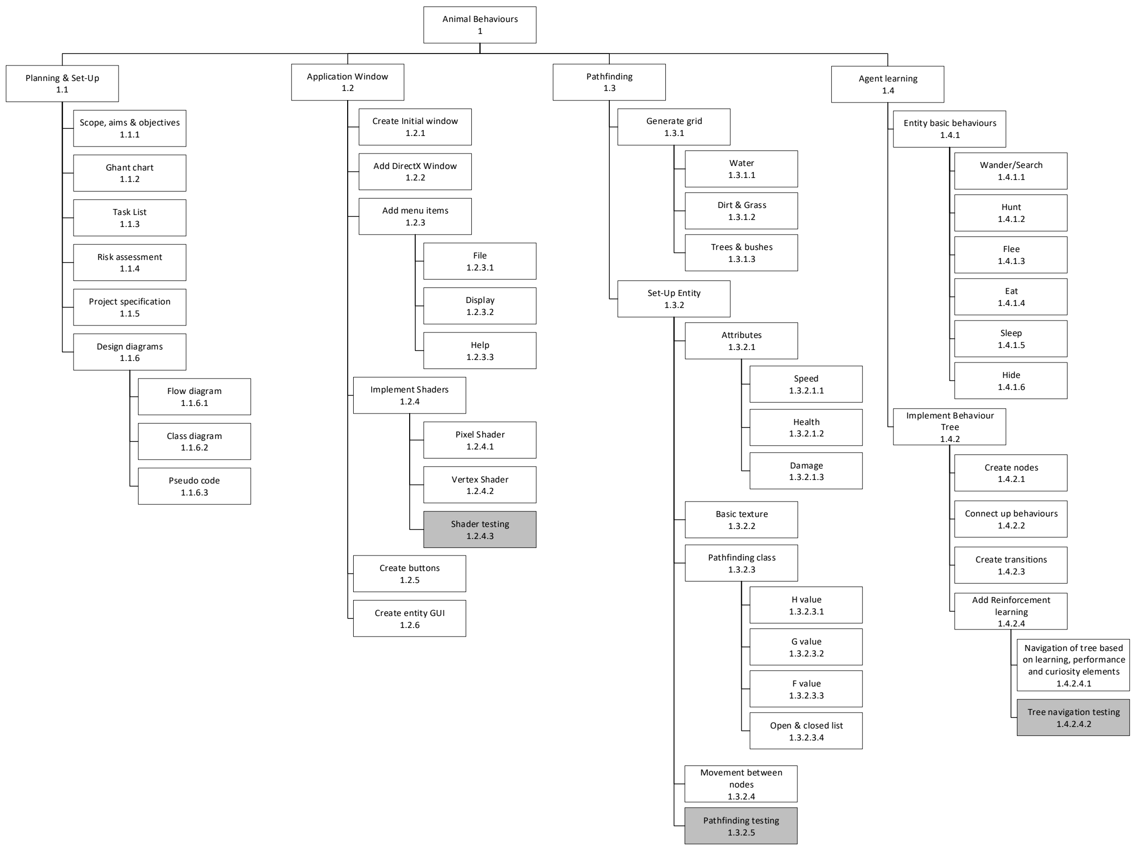
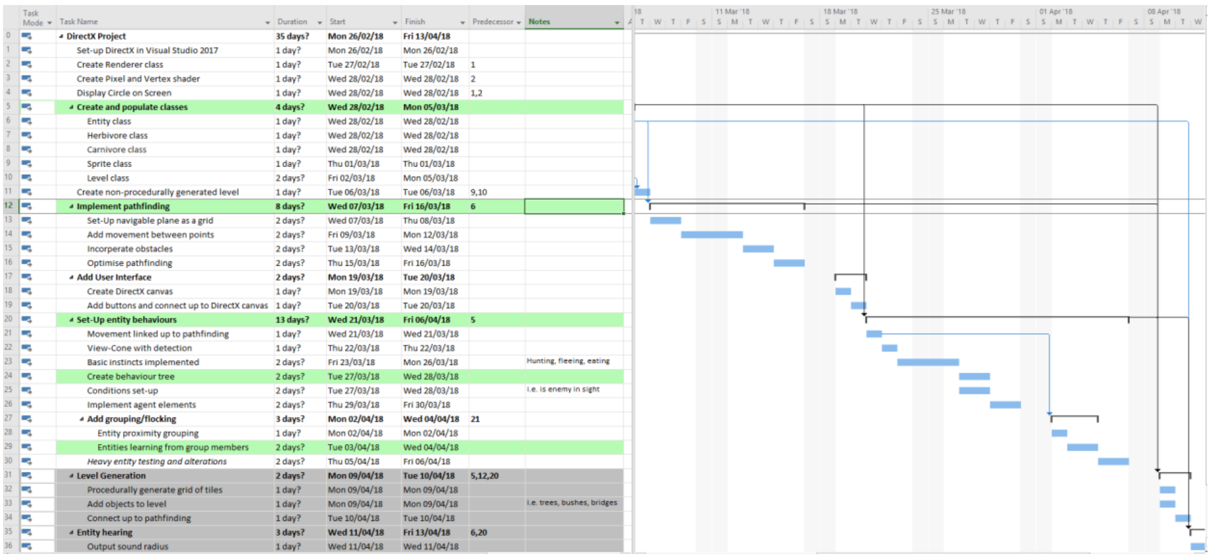


Figure 5: WBS diagram

2.9. Ghant Chart/Task List

Here contains the Ghant Chart, See Figure 6, covering the workload, spanning from the 26th of February to the 6th of April. This includes any extra work that could be done, marked with a grey overlay. I have set milestones which will act as my phases marked in green. These lay out key stages within the implementation that are vital to the progression of the project. The full Ghant chart can be found in the Resources folder of this project, named Ghant Chart.mpp.



	Task Mode	Task Name	Duration	Start	Finish	Predecessor	Notes
0		**DirectX Project**	**35 days?**	**Mon 26/02/18**	**Fri 13/04/18**		
1		Set-up DirectX in Visual Studio 2017	1 day?	Mon 26/02/18	Mon 26/02/18		
2		Create Renderer class	1 day?	Tue 27/02/18	Tue 27/02/18	1	
3		Create Pixel and Vertex shader	1 day?	Wed 28/02/18	Wed 28/02/18	2	
4		Display Circle on Screen	1 day?	Wed 28/02/18	Wed 28/02/18	1,2	
5		**Create and populate classes**	**4 days?**	**Wed 28/02/18**	**Mon 05/03/18**		
6		Entity class	1 day?	Wed 28/02/18	Wed 28/02/18		
7		Herbivore class	1 day?	Wed 28/02/18	Wed 28/02/18		
8		Carnivore class	1 day?	Wed 28/02/18	Wed 28/02/18		
9		Sprite class	1 day?	Thu 01/03/18	Thu 01/03/18		
10		Level class	2 days?	Fri 02/03/18	Mon 05/03/18		
11		Create non-procedurally generated level	1 day?	Tue 06/03/18	Tue 06/03/18	9,10	
12		**Implement pathfinding**	**8 days?**	**Wed 07/03/18**	**Fri 16/03/18**	**6**	
13		Set-Up navigable plane as a grid	2 days?	Wed 07/03/18	Thu 08/03/18		
14		Add movement between points	2 days?	Fri 09/03/18	Mon 12/03/18		
15		Incorperate obstacles	2 days?	Tue 13/03/18	Wed 14/03/18		
16		Optimise pathfinding	2 days?	Thu 15/03/18	Fri 16/03/18		
17		**Add User Interface**	**2 days?**	**Mon 19/03/18**	**Tue 20/03/18**		
18		Create DirectX canvas	1 day?	Mon 19/03/18	Mon 19/03/18		
19		Add buttons and connect up to DirectX canvas	1 day?	Tue 20/03/18	Tue 20/03/18		
20		**Set-Up entity behaviours**	**13 days?**	**Wed 21/03/18**	**Fri 06/04/18**	**5**	
21		Movement linked up to pathfinding	1 day?	Wed 21/03/18	Wed 21/03/18		
22		View-Cone with detection	1 day?	Thu 22/03/18	Thu 22/03/18		
23		Basic instincts implemented	2 days?	Fri 23/03/18	Mon 26/03/18		Hunting, fleeing, eating
24		Create behaviour tree	2 days?	Tue 27/03/18	Wed 28/03/18		
25		Conditions set-up	2 days?	Tue 27/03/18	Wed 28/03/18		i.e. is enemy in sight
26		Implement agent elements	2 days?	Thu 29/03/18	Fri 30/03/18		
27		**Add grouping/flocking**	**3 days?**	**Mon 02/04/18**	**Wed 04/04/18**	**21**	
28		Entity proximity grouping	1 day?	Mon 02/04/18	Mon 02/04/18		
29		Entities learning from group members	2 days?	Tue 03/04/18	Wed 04/04/18		
30		*Heavy entity testing and alterations*	2 days?	Thu 05/04/18	Fri 06/04/18		
31		**Level Generation**	**2 days?**	**Mon 09/04/18**	**Tue 10/04/18**	**5,12,20**	
32		Procedurally generate grid of tiles	1 day?	Mon 09/04/18	Mon 09/04/18		
33		Add objects to level	1 day?	Mon 09/04/18	Mon 09/04/18		i.e. trees, bushes, bridges
34		Connect up to pathfinding	1 day?	Tue 10/04/18	Tue 10/04/18		
35		**Entity hearing**	**3 days?**	**Wed 11/04/18**	**Fri 13/04/18**	**6,20**	
36		Output sound radius	1 day?	Wed 11/04/18	Wed 11/04/18		

Figure 6: Ghant Chart

I will be using this Ghant chart as my task list to eliminate the chances of having unsynchronised data between different pieces of software. I will be using Microsoft Project Professionals built in task percentage feature that allows tasks to be marked as underway. See figure 7 & 8.

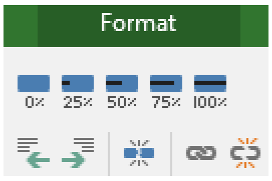


Figure 7: Project Professional Percentage

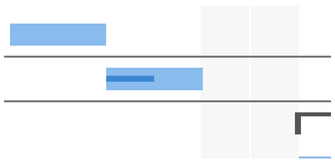


Figure 8: Project Professional 50% Example

2.10. Product Design

2.10.1. Flow diagram

I have designed a game flow diagram to outline how the application will run in terms of processes and decisions. This diagram, see Figure 9, contains a game loop in which the processes relating to placing and viewing an entity are inside.

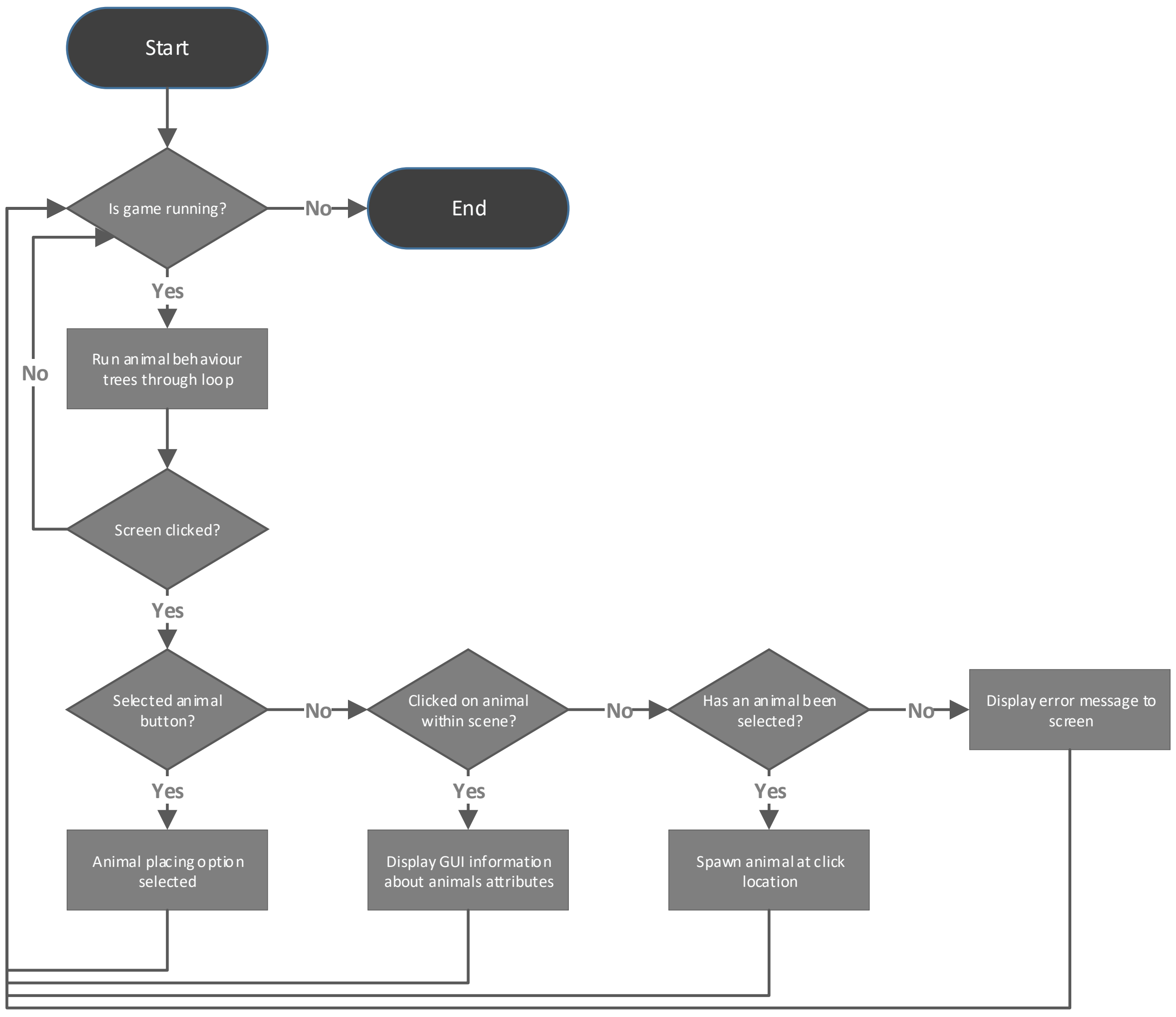

Start
Is game running?
No
End
Yes
Run animal behaviour trees through loop
No
Screen clicked?
Yes
Selected animal button?
No
Clicked on animal within scene?
No
Has an animal been selected?
No
Display error message to screen
Yes
Animal placing option selected
Yes
Display GUI information about animals attributes
Yes
Spawn animal at click location

Figure 9: Game Flow Diagram

2.10.2 Pseudo Code

The pseudo code here describes how an entity will learn after playing out an action from the behaviour tree, see Figure 10.

```
Start
IF actionComplete THEN
    IF Outcome == positiveResult THEN
        Increase attribute values relating to result
    ELSE IF Outcome == negativeResult THEN
        Decrease attribute values relating to result
ELSE
Carry out action
END
```

Figure 10: Pseudo Code

2.10.3 Visual Prototype

See below the prototype for the user Interface and windows within the application. See Figure 11.

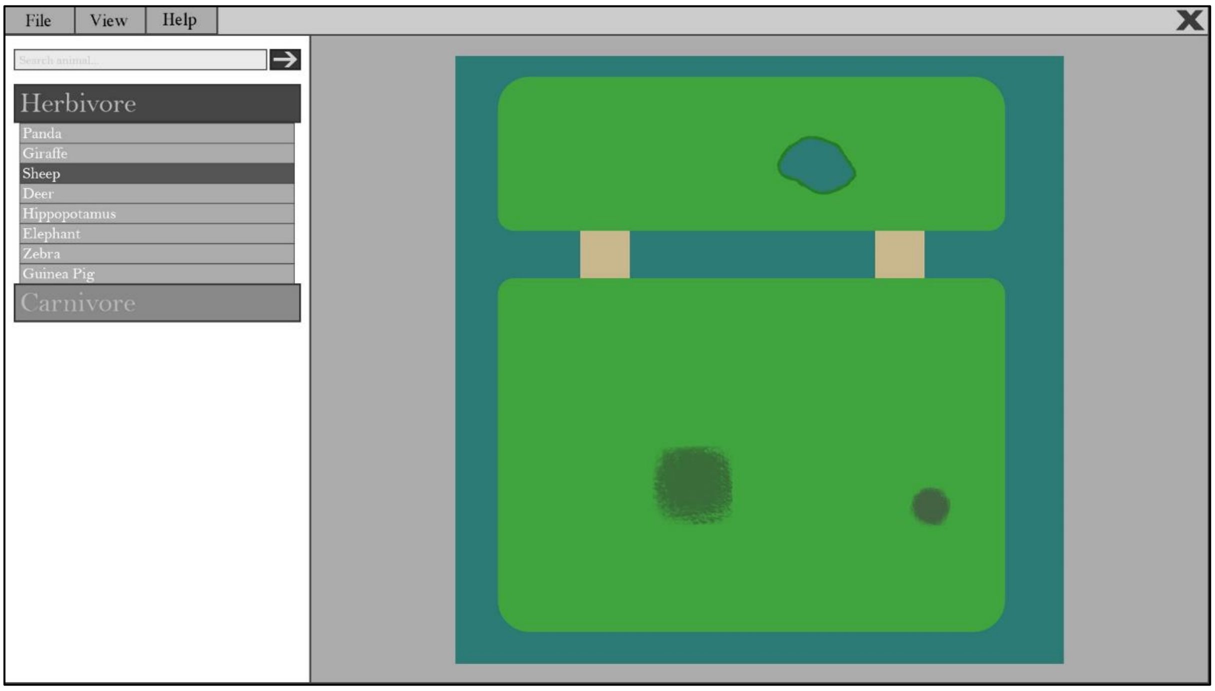


Figure 11: Visual Prototype

2.10.4. Class Diagram

I have designed a class diagram to allow me to visualise what classes will need to be implemented and how they will interact with each other. The relationships of the classes have been included in the diagram to demonstrate the complexity between the classes. See figure 11.

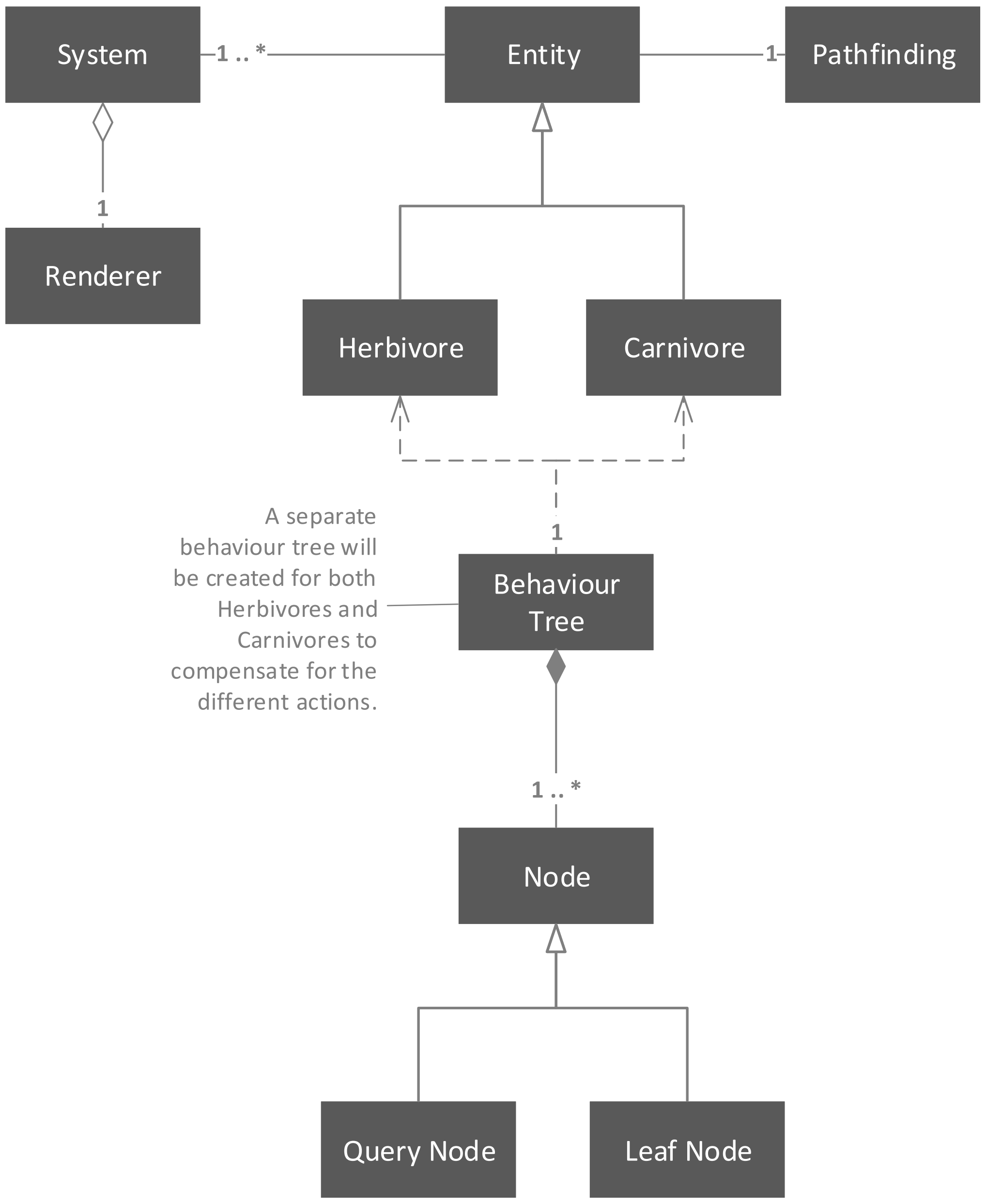

System
1 .. *
Entity
1
Pathfinding
1
Renderer
Herbivore
Carnivore
1
A separate behaviour tree will be created for both Herbivores and Carnivores to compensate for the different actions.
Behaviour Tree
1 .. *
Node
Query Node
Leaf Node

Figure 12: Class Diagram

2.10.5. Behaviour Tree Diagram

During my research I designed an example Behaviour Tree that visually explains how the tree will run, see Figure 2.

3. Final Report

3.1. Entity flocking

3.1.1. Implementing Cohesion, Separation and Alignment

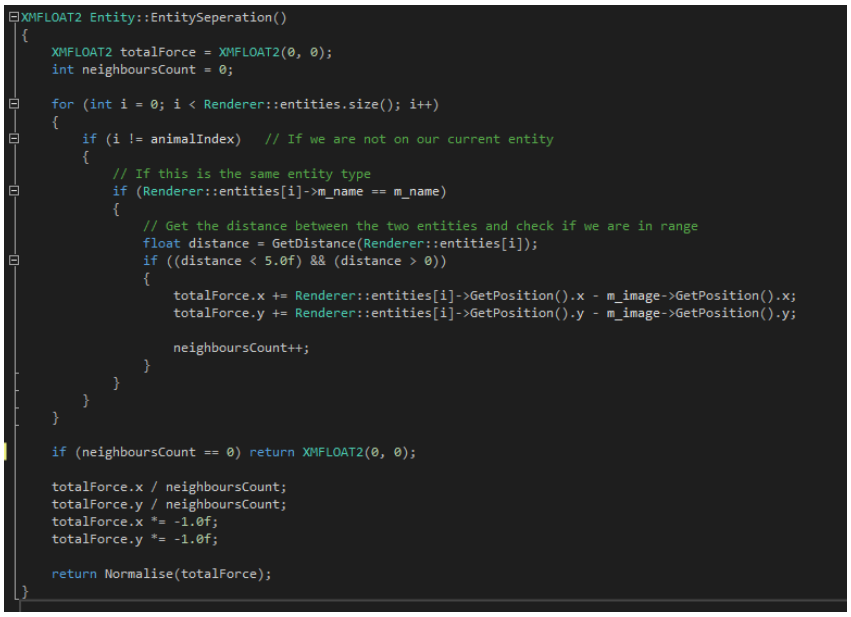
The implementation of Cohesion, Separation and Alignment upon the entities movement is a feature that allows the entities to display a sense of intelligence. The three sections implemented as functions work together to modify an entities velocity represented as an XMFLOAT2 (Microsoft, 2018). The three different functions work as follows:

- Cohesion: Returns a vector towards the centre position of the surrounding entities.

- Separation: Returns a vector away from the centre position of the surrounding entities.

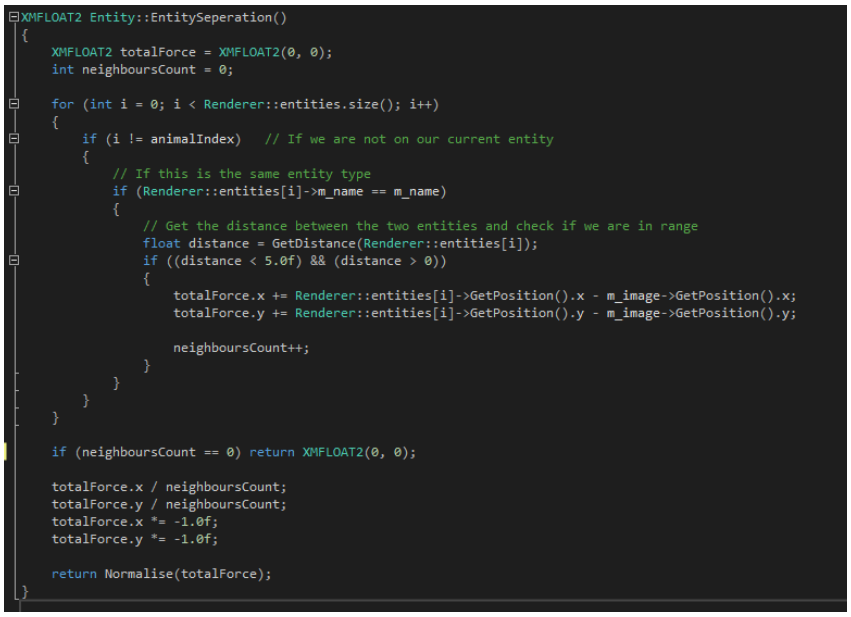
- Alignment: Returns a vector of the direction average from the surrounding entities including itself.

The implementation of these elements was based on the same principle of looping through each element and performing a distance check. Entities within a certain distance of the entity they are checking would have their position added to a force value. This value would then be divided by the number of entities found to return the average position (See



```
XMFLOAT2 Entity::EntitySeperation()
{
    XMFLOAT2 totalForce = XMFLOAT2(0, 0);
    int neighboursCount = 0;

    for (int i = 0; i < Renderer::entities.size(); i++)
    {
        if (i != animalIndex)   // If we are not on our current entity
        {
            // If this is the same entity type
            if (Renderer::entities[i]->m_name == m_name)
            {
                // Get the distance between the two entities and check if we are in range
                float distance = GetDistance(Renderer::entities[i]);
                if ((distance < 5.0f) && (distance > 0))
                {
                    totalForce.x += Renderer::entities[i]->GetPosition().x - m_image->GetPosition().x;
                    totalForce.y += Renderer::entities[i]->GetPosition().y - m_image->GetPosition().y;

                    neighboursCount++;
                }
            }
        }
    }

    if (neighboursCount == 0) return XMFLOAT2(0, 0);

    totalForce.x / neighboursCount;
    totalForce.y / neighboursCount;
    totalForce.x *= -1.0f;
    totalForce.y *= -1.0f;

    return Normalise(totalForce);
}
```

Figure 13 for an example of Entity Separation).

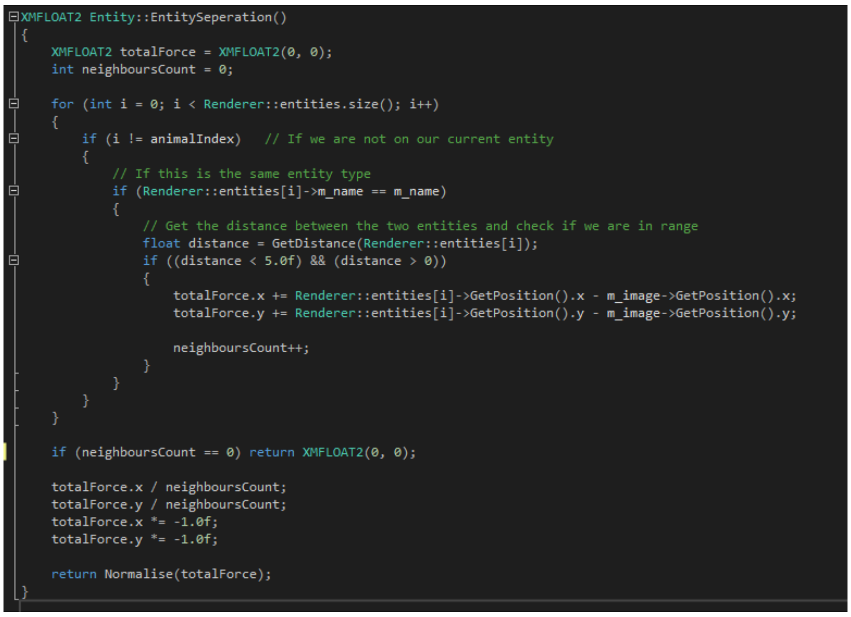


```
XMFLOAT2 Entity::EntitySeperation()
{
    XMFLOAT2 totalForce = XMFLOAT2(0, 0);
    int neighboursCount = 0;

    for (int i = 0; i < Renderer::entities.size(); i++)
    {
        if (i != animalIndex)   // If we are not on our current entity
        {
            // If this is the same entity type
            if (Renderer::entities[i]->m_name == m_name)
            {
                // Get the distance between the two entities and check if we are in range
                float distance = GetDistance(Renderer::entities[i]);
                if ((distance < 5.0f) && (distance > 0))
                {
                    totalForce.x += Renderer::entities[i]->GetPosition().x - m_image->GetPosition().x;
                    totalForce.y += Renderer::entities[i]->GetPosition().y - m_image->GetPosition().y;

                    neighboursCount++;
                }
            }
        }
    }

    if (neighboursCount == 0) return XMFLOAT2(0, 0);

    totalForce.x / neighboursCount;
    totalForce.y / neighboursCount;
    totalForce.x *= -1.0f;
    totalForce.y *= -1.0f;

    return Normalise(totalForce);
}
```

Figure 13 : Flocking Separation Code Example

3.1.2. Changing the Values

An On-Screen set of text visualising the values of Cohesion, Separation and Alignment was added to see the outcome of altering the values. The results can



```
XMFLOAT2 Entity::EntitySeperation()
{
    XMFLOAT2 totalForce = XMFLOAT2(0, 0);
    int neighboursCount = 0;

    for (int i = 0; i < Renderer::entities.size(); i++)
    {
        if (i != animalIndex)   // If we are not on our current entity
        {
            // If this is the same entity type
            if (Renderer::entities[i]->m_name == m_name)
            {
                // Get the distance between the two entities and check if we are in range
                float distance = GetDistance(Renderer::entities[i]);
                if ((distance < 5.0f) && (distance > 0))
                {
                    totalForce.x += Renderer::entities[i]->GetPosition().x - m_image->GetPosition().x;
                    totalForce.y += Renderer::entities[i]->GetPosition().y - m_image->GetPosition().y;

                    neighboursCount++;
                }
            }
        }
    }

    if (neighboursCount == 0) return XMFLOAT2(0, 0);

    totalForce.x / neighboursCount;
    totalForce.y / neighboursCount;
    totalForce.x *= -1.0f;
    totalForce.y *= -1.0f;

    return Normalise(totalForce);
}
```

be seen in

Figure 13 and Appendix C, C.2.

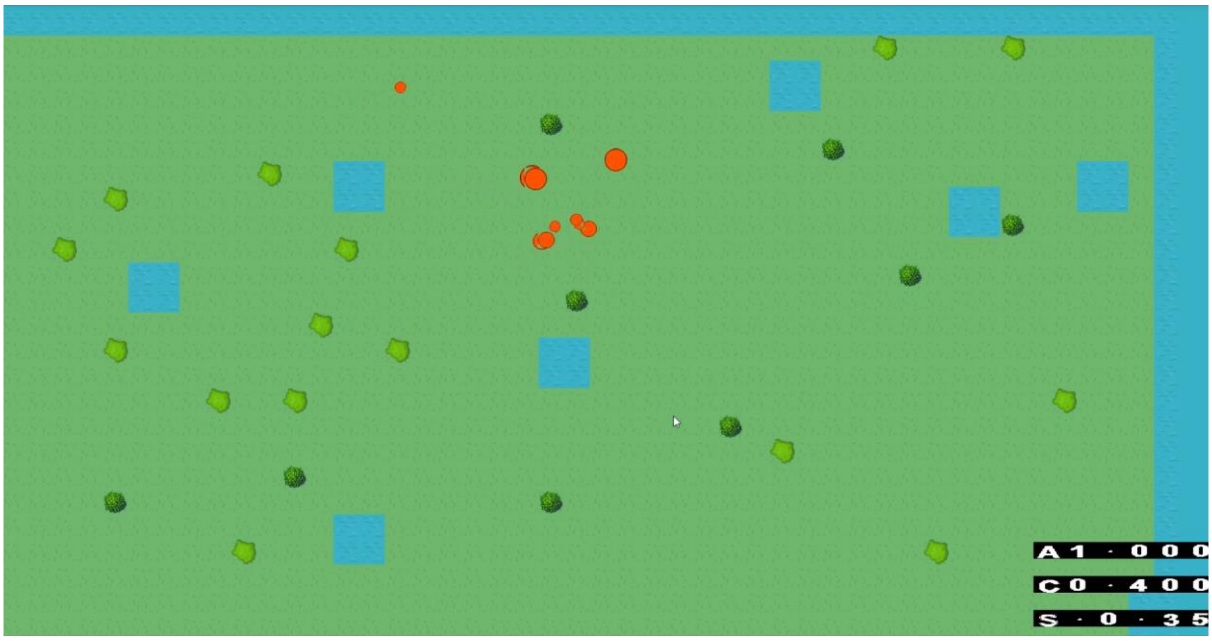


Figure 14 : Flocking evidence: https://youtu.be/8PBZQDXJMd4

3.1.3. Issues with Overlap

As shown in Figure 15 the entities would not abide by their separation value and would begin to overlap with one another. Two attempts to fix this issue will be discussed including setting up circle collisions for the entities and increasing the separation value calculated for the flocking.

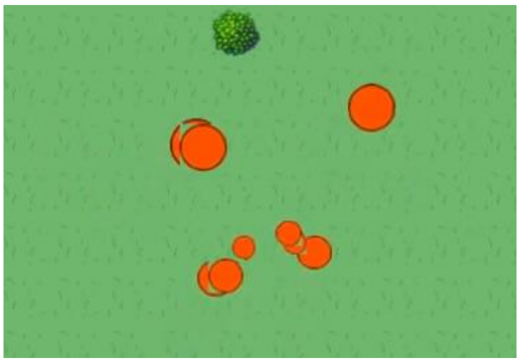
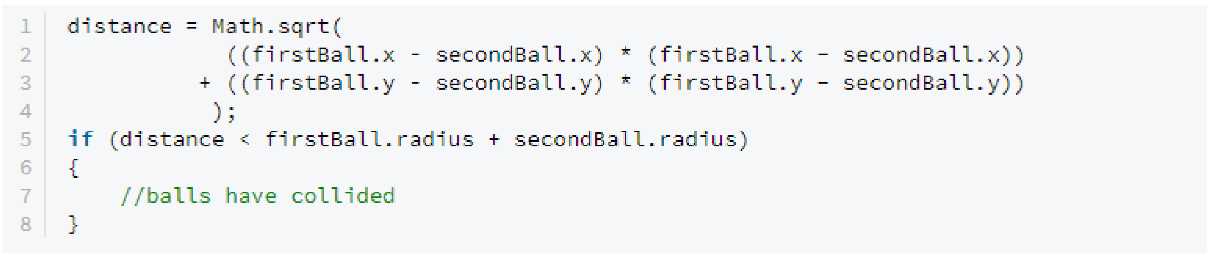


Figure 15: Overlap of entities

An approach to using circle collisions is a valid one with a simple implementation. The equation needed for this function was discovered from Darran Jamiesons's article When Worlds Collide: Simulating Circle-Circle Collisions (Jamieson, 2012). The equation can be seen in Figure 16 and details when a collision will occur. Once implemented and tested (See Figure 17 & Figure 18) within the application it was realised that when a collision had occurred, an action would have to be performed, such as stopping the entity moving altogether or moving the entity back to the previous position before the collision occurred. This however led to more issues where an entity would become stuck once colliding with another entity.



```
distance = Math.sqrt(
            ((firstBall.x - secondBall.x) * (firstBall.x - secondBall.x))
          + ((firstBall.y - secondBall.y) * (firstBall.y - secondBall.y))
           );
if (distance < firstBall.radius + secondBall.radius)
{
    //balls have collided
}
```

Figure 16 : Circle Collisions example

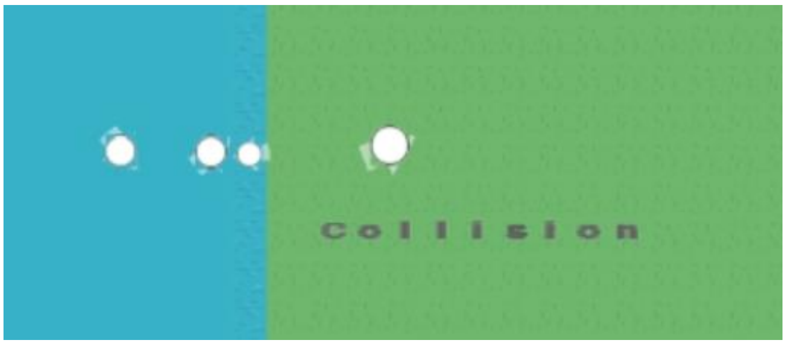
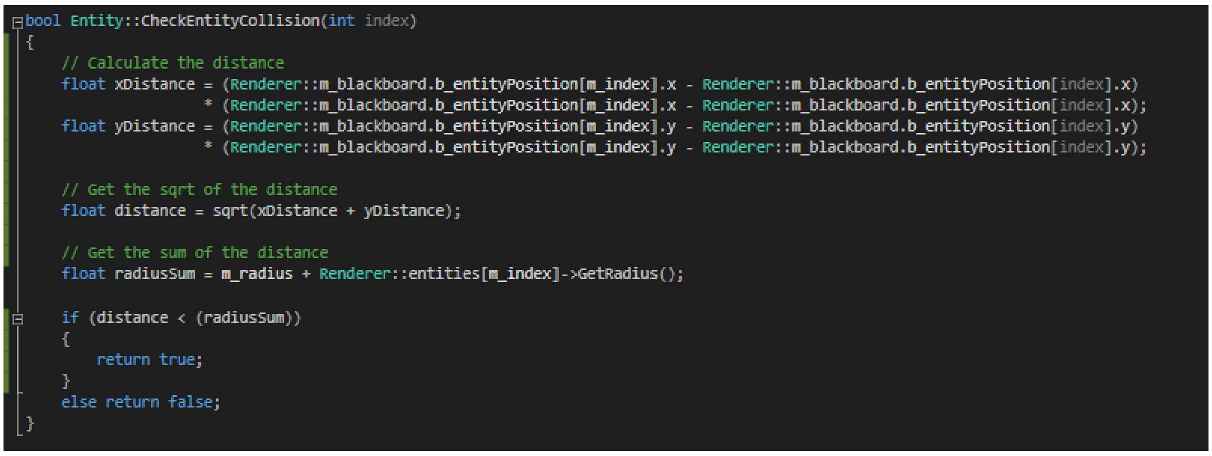


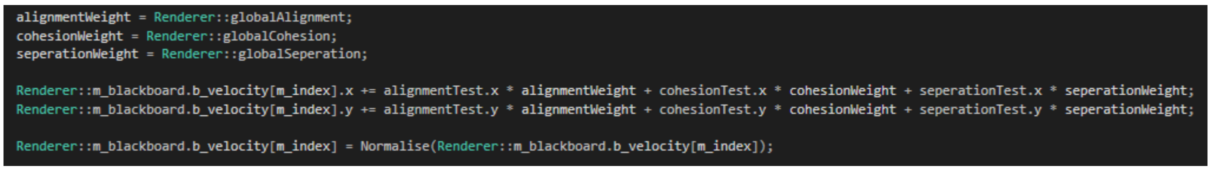
Figure 17 : Collision Evidence - https://youtu.be/RYdqpfktV-U



```
bool Entity::CheckEntityCollision(int index)
{
    // Calculate the distance
    float xDistance = (Renderer::m_blackboard.b_entityPosition[m_index].x - Renderer::m_blackboard.b_entityPosition[index].x)
                    * (Renderer::m_blackboard.b_entityPosition[m_index].x - Renderer::m_blackboard.b_entityPosition[index].x);
    float yDistance = (Renderer::m_blackboard.b_entityPosition[m_index].y - Renderer::m_blackboard.b_entityPosition[index].y)
                    * (Renderer::m_blackboard.b_entityPosition[m_index].y - Renderer::m_blackboard.b_entityPosition[index].y);

    // Get the sqrt of the distance
    float distance = sqrt(xDistance + yDistance);

    // Get the sum of the distance
    float radiusSum = m_radius + Renderer::entities[m_index]->GetRadius();

    if (distance < (radiusSum))
    {
        return true;
    }
    else return false;
}
```

Figure 18 : Circle Collision Check Code

The solution of increasing the separation value within each entities flocking calculations was an idea with a clean outcome that required a small amount of extra information. Three static variables were created that control the multipliers for Cohesion, Separation and Alignment. These values can be multiplied with the individual values to give importance to specified attributes. In this case the Separation value was set to a higher float than the others (See Figure 19). This outcome gave a positive result with entities moving away from each other at close range.



```
alignmentWeight = Renderer::globalAlignment;
cohesionWeight = Renderer::globalCohesion;
seperationWeight = Renderer::globalSeperation;

Renderer::m_blackboard.b_velocity[m_index].x += alignmentTest.x * alignmentWeight + cohesionTest.x * cohesionWeight + seperationTest.x * seperationWeight;
Renderer::m_blackboard.b_velocity[m_index].y += alignmentTest.y * alignmentWeight + cohesionTest.y * cohesionWeight + seperationTest.y * seperationWeight;

Renderer::m_blackboard.b_velocity[m_index] = Normalise(Renderer::m_blackboard.b_velocity[m_index]);
```

Figure 19 : Evidence of flocking multipliers

3.2. Behaviour Tree Implementation

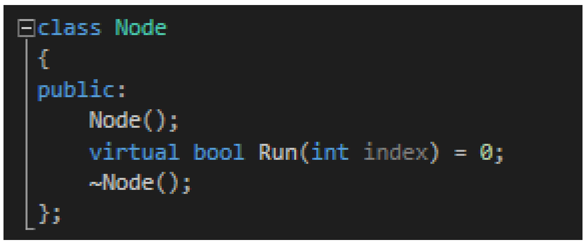
The Behaviour Tree is the most essential aspect of this project in relation to how the mechanics were implemented. Without the use of a Behaviour Tree the entities would be in relation to one another through the use of simple checks within each entities Update function. The Behaviour Tree solution is a much cleaner one.

Chris Simpson’s article ‘Behaviour trees for AI: How they work’ (Simpson, 2014) outlines every possible piece of information that was needed for the understanding of how a Behaviour Tree could be designed, what they are made up of and how it can be executed. The article also discusses the different node types that can be used within a behaviour tree, many of which were used in this project.

3.2.1. Setting up the Nodes

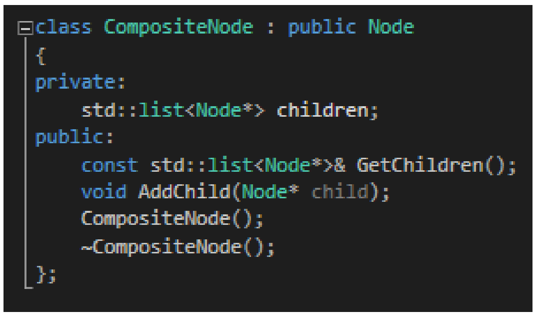
When referring to a node in the context of the project a class is what is being discussed (See Figure 20 and Figure 21). A class named Node was created with an overloaded function called Run. This function takes one integer parameter representing the index of the entity and returns the data type Boolean. The function remains empty within this base class but populated with code in the derived classes. Alongside this base node class there are others including:

- **CompositeNode** – contains a collection of child nodes, there are two types:
    - ***Sequence*** – All children must return true for this node to return true
    - ***Selector*** – Any child can return true for this node to return true
- **Inverter -** Inverts the result of the return value
- **Repeater (Repeat until fail)** – Runs its child node until it returns false

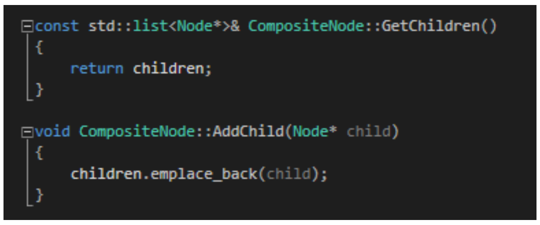


```
class Node
{
public:
    Node();
    virtual bool Run(int index) = 0;
    ~Node();
};
```

Figure 20 : Node class



```
class CompositeNode : public Node
{
private:
    std::list<Node*> children;
public:
    const std::list<Node*>& GetChildren();
    void AddChild(Node* child);
    CompositeNode();
    ~CompositeNode();
};
```



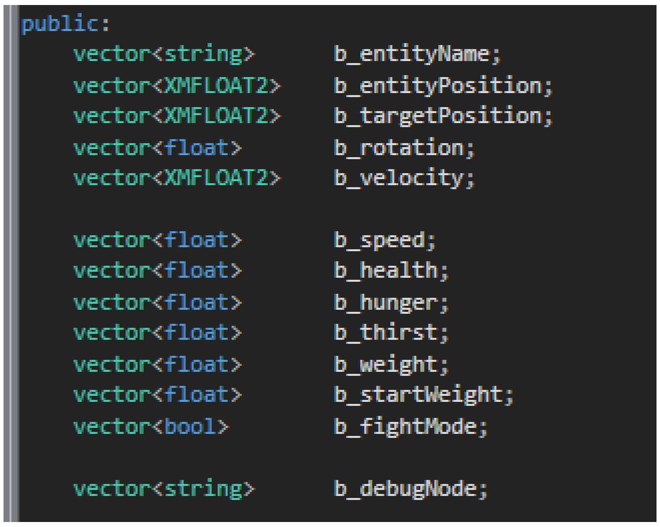
```
const std::list<Node*>& CompositeNode::GetChildren()
{
    return children;
}

void CompositeNode::AddChild(Node* child)
{
    children.emplace_back(child);
}
```

Figure 21 : Composite Class

The leaf nodes that represent the actions of the program such as hunting or sleeping will be defined by one of these nodes. Once a node has been set up and populated with code, that will adapt or check the behaviour of the entity, it can have its parent set or in this case have the parent set the child. The node also contains an AddChild function which parents a child to a node allowing the program to traverse the tree using recursion. To understand how the program traverses the tree please see the videos in Apendix C, elements C.8 & C.9.

3.2.2. Blackboard

A blackboard simply consists of data that needs to be accessed from different locations (Champandard, 2007). A static blackboard was created to store all of the information needed for the entities to access. Information such as the entities position, target position and health are stored and then accessed through the Renderer (See Figure 22). This is the reasoning behind making the blackboard static.



```
public:
    vector<string>      b_entityName;
    vector<XMFLOAT2>    b_entityPosition;
    vector<XMFLOAT2>    b_targetPosition;
    vector<float>       b_rotation;
    vector<XMFLOAT2>    b_velocity;

    vector<float>       b_speed;
    vector<float>       b_health;
    vector<float>       b_hunger;
    vector<float>       b_thirst;
    vector<float>       b_weight;
    vector<float>       b_startWeight;
    vector<bool>        b_fightMode;

    vector<string>      b_debugNode;
```

Figure 22 : Blackboard header file

The information stored here can be written to and read from at any location within the Behaviour Tree nodes, as long as the Renderer header file is included. To keep all of the data consistent with the entities they respond to, an index was added to the overwritten Run function within the base class Node. When the Behaviour Tree is run, the entities personal index is passed in, allowing the Behaviour Tree to access the blackboards vectors using this index. Once an entity is killed, they are not removed from the vector of entities as they will continue to be part of the level.

3.2.3. Creating a simple Behaviour

To allow for the concepts to be tested and tried a simple Behaviour Tree was set up containing no more than a few nodes representing a drink behaviour. Firstly a sequence node was created and set as the child of the root node, of type selector (See Figure 23).

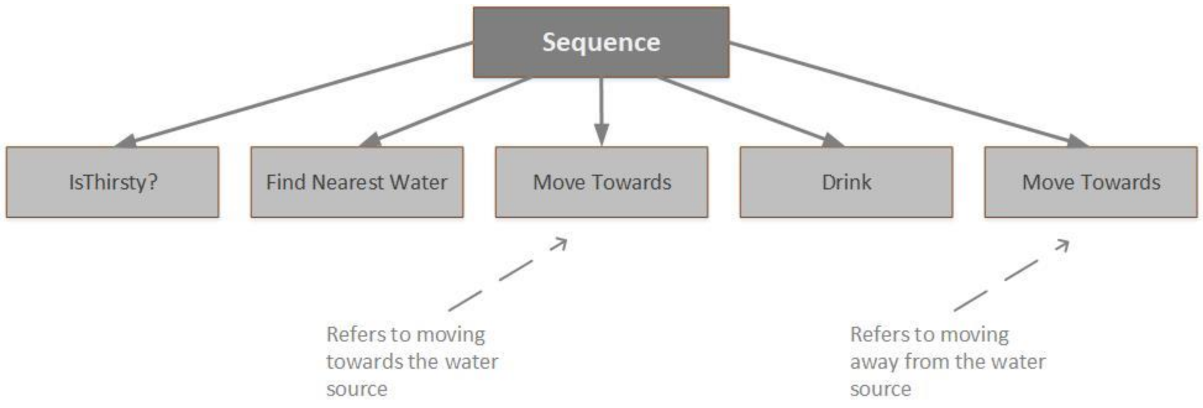
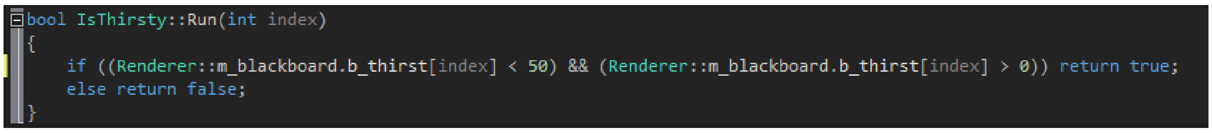

Sequence
IsThirsty?
Find Nearest Water
Move Towards
Drink
Move Towards
Refers to moving towards the water source
Refers to moving away from the water source

Figure 23 : Simple Behaviour Tree

Four leaf nodes were then created each dealing with a different task including:

3.2.3.1. IsThirsty

Checks if the entity is in need of water.



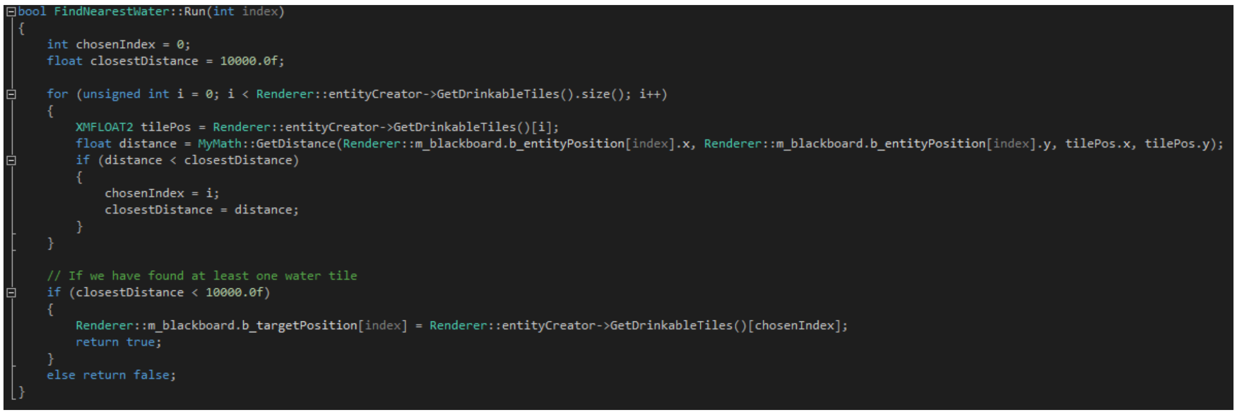
```
bool IsThirsty::Run(int index)
{
    if ((Renderer::m_blackboard.b_thirst[index] < 50) && (Renderer::m_blackboard.b_thirst[index] > 0)) return true;
    else return false;
}
```

Figure 24 : IsThirsty Code

The blackboard is accessed and a simple check is done to see if the entities thirst value is less than half of the starting value and greater than 0. If this is correct then the function will return true and the program will carry on.

3.2.3.2. FindNearestWater

Finds the nearest tile represented by water.



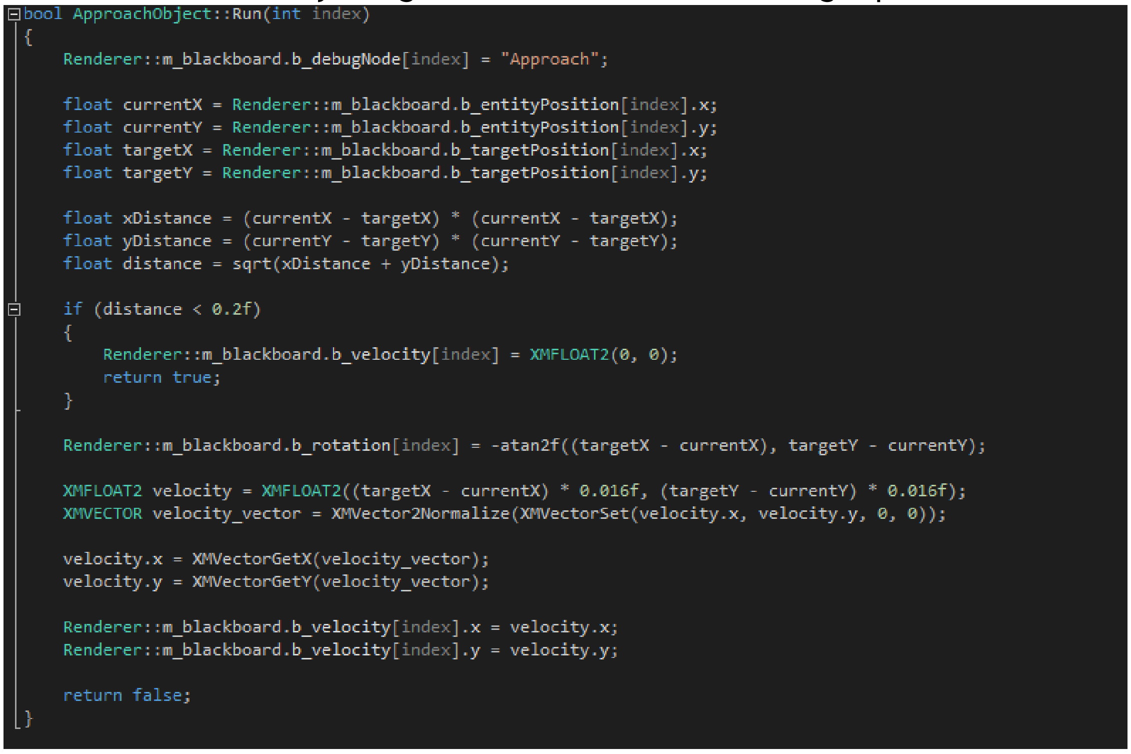
```cpp
bool FindNearestWater::Run(int index)
{
    int chosenIndex = 0;
    float closestDistance = 10000.0f;

    for (unsigned int i = 0; i < Renderer::entityCreator->GetDrinkableTiles().size(); i++)
    {
        XMFLOAT2 tilePos = Renderer::entityCreator->GetDrinkableTiles()[i];
        float distance = MyMath::GetDistance(Renderer::m_blackboard.b_entityPosition[index].x, Renderer::m_blackboard.b_entityPosition[index].y, tilePos.x, tilePos.y);
        if (distance < closestDistance)
        {
            chosenIndex = i;
            closestDistance = distance;
        }
    }

    // If we have found at least one water tile
    if (closestDistance < 10000.0f)
    {
        Renderer::m_blackboard.b_targetPosition[index] = Renderer::entityCreator->GetDrinkableTiles()[chosenIndex];
        return true;
    }
    else return false;
}
```

Figure 25 : FindNearestWater Code

This node loops through each drinkable tile, determined through the EntityCreator, calculating the distance between the entity and the tile. If the current tile is closer than the closest stored tile then the value is overwritten and the new tile becomes the closest. If a tile is found then the target position of the entity is set to the position of the tile and the function returns true. If no tile is found the function will return false and the sequence will end.

3.2.3.3. MoveTowards

Calculates the velocity using the entities current and target position.



```cpp
bool ApproachObject::Run(int index)
{
    Renderer::m_blackboard.b_debugNode[index] = "Approach";

    float currentX = Renderer::m_blackboard.b_entityPosition[index].x;
    float currentY = Renderer::m_blackboard.b_entityPosition[index].y;
    float targetX = Renderer::m_blackboard.b_targetPosition[index].x;
    float targetY = Renderer::m_blackboard.b_targetPosition[index].y;

    float xDistance = (currentX - targetX) * (currentX - targetX);
    float yDistance = (currentY - targetY) * (currentY - targetY);
    float distance = sqrt(xDistance + yDistance);

    if (distance < 0.2f)
    {
        Renderer::m_blackboard.b_velocity[index] = XMFLOAT2(0, 0);
        return true;
    }

    Renderer::m_blackboard.b_rotation[index] = -atan2f((targetX - currentX), targetY - currentY);

    XMFLOAT2 velocity = XMFLOAT2((targetX - currentX) * 0.016f, (targetY - currentY) * 0.016f);
    XMVECTOR velocity_vector = XMVector2Normalize(XMVectorSet(velocity.x, velocity.y, 0, 0));

    velocity.x = XMVectorGetX(velocity_vector);
    velocity.y = XMVectorGetY(velocity_vector);

    Renderer::m_blackboard.b_velocity[index].x = velocity.x;
    Renderer::m_blackboard.b_velocity[index].y = velocity.y;

    return false;
}
```

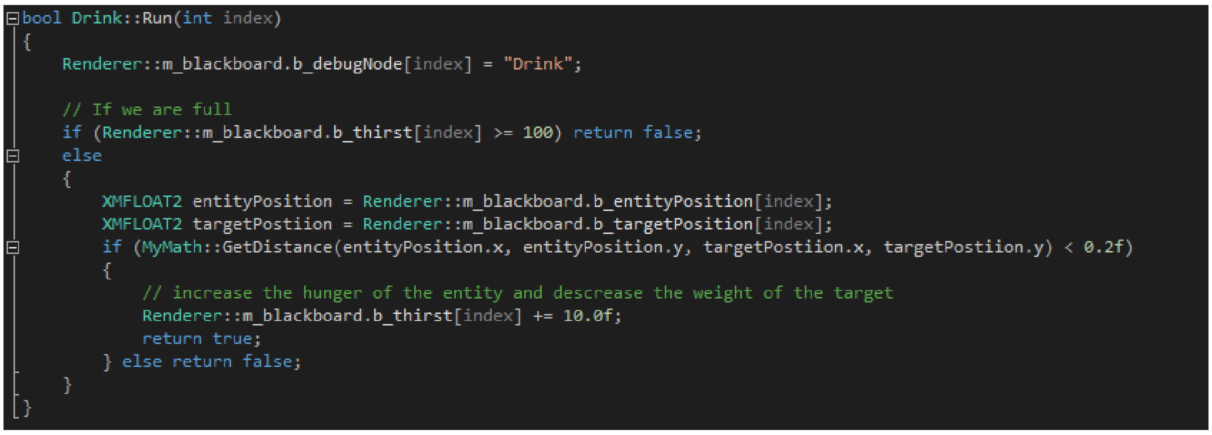
Figure 26 : MoveTowards Code

This function simply calculates the distance between the entity and the target, setting the velocity to zero and returning true if the entity is within a certain range. If the entity is outside of this range then a new velocity is determined using the -atan2f function.

XMFLOAT2 and XMVECTOR have been used to convert the velocity into a format that can be worked with, a generic floating point value. This function will only return true once the entity has reached its target position.

3.2.3.4. Drink

Decreases the amount of thirst that the entity has.



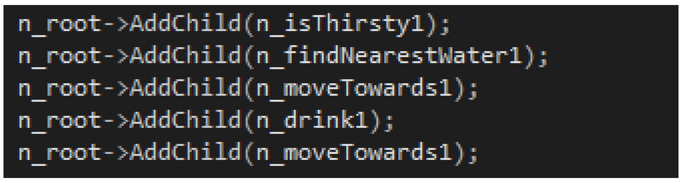
```
bool Drink::Run(int index)
{
    Renderer::m_blackboard.b_debugNode[index] = "Drink";

    // If we are full
    if (Renderer::m_blackboard.b_thirst[index] >= 100) return false;
    else
    {
        XMFLOAT2 entityPosition = Renderer::m_blackboard.b_entityPosition[index];
        XMFLOAT2 targetPostiion = Renderer::m_blackboard.b_targetPosition[index];
        if (MyMath::GetDistance(entityPosition.x, entityPosition.y, targetPostiion.x, targetPostiion.y) < 0.2f)
        {
            // increase the hunger of the entity and descrease the weight of the target
            Renderer::m_blackboard.b_thirst[index] += 10.0f;
            return true;
        } else return false;
    }
}
```

Figure 27 : Drink Code

This nodes function consists of a distance check, with the entities b_thirst value increasing if the entity is within the specific range. If the entities thirst is full or they are not in distance of another entity, a return value only put in as a safety measure, then the return value will be false.

3.2.3.5. Linking the tree

Once all four nodes had been created the parents could be set and the tree could be traversed. As the MoveTowards node had already been created, there was no need to declare a second version, instead the same class was used when adding the child to the root node (See Figure 28). This is another advantage to using the principles of Encapsulation by allowing nodes to be created and declared once with the ability to be reused as many times as needed, severely reducing the amount of code that is used.



```
n_root->AddChild(n_isThirsty1);
n_root->AddChild(n_findNearestWater1);
n_root->AddChild(n_moveTowards1);
n_root->AddChild(n_drink1);
n_root->AddChild(n_moveTowards1);
```

Figure 28: Simple behaviour Tree Parenting

3.2.4. Expanding on the Behaviour

3.2.4.1. Adding to the Behaviour Tree

An effective approach, which worked well for the design of the Behaviour Tree, was to create and add nodes along the way, therefore a node could be created and populated with code, added to the tree immediately and tested. The opposite approach would have been to design the entire behaviour tree at the start, following every node throughout the implementation until the entire tree is created. This could however had led to issues such as finding a flaw in the hierarchy of nodes at a late stage of the project or creating nodes that go unused or are unnecessary or overcomplicated. By designing a small section of the tree, creating the nodes and then testing them, the functionality of the Behaviour Tree could be readjusted and adapted during the implementation instead of having to deal with an enormous array of different functioning nodes at the end of the project.

3.2.4.2. Nested Sequence and Selector Nodes

A Behaviour Tree would be an unnecessary approach to this project if nodes were not used as a way of parenting other nodes. By creating sequence and selector nodes with children added to them, the tree has the ability to traverse a far more complex situation. This allows nodes to be created and arranged in a format that best suits a situation. The advancements on the simple Behaviour Tree can be found in Figure 29, where a sequence node has been added with functionality represented within its children.

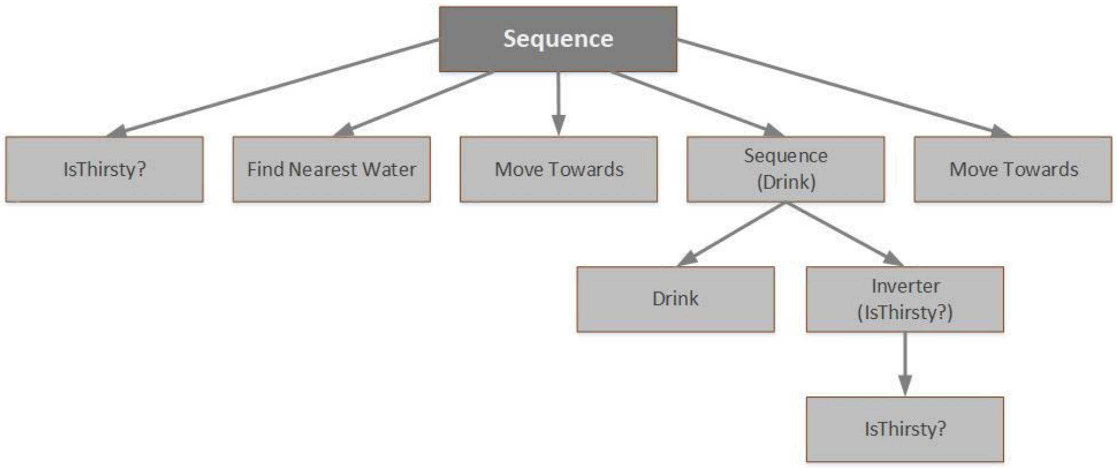
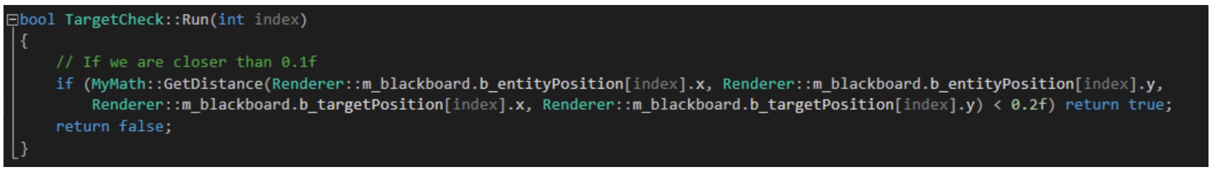


Figure 29 : Nested Sequence Node

This new approach allows for the entity to check if they are thirsty, find the nearest water source, move towards that water source and then check whether they are still thirsty before moving away. Beforehand using the simple Behaviour Tree example (See Figure 23) the entity would move towards the water and move away after entering the drink node once. By using a nested Sequence node with the parent set to the root, the child nodes will need to both return true for the entity to then move onto the second instance of the MoveTowards node. This is the reasoning behind using an Inverter node to control the return value of the nested IsThirsty node.

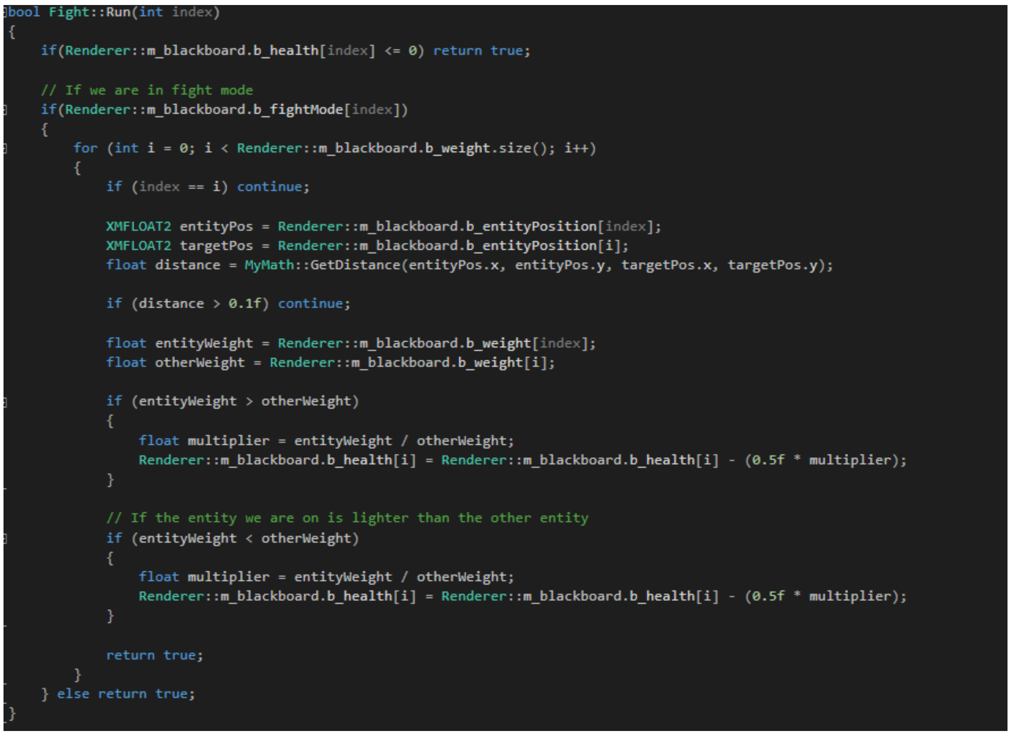
3.2.4.3. Distance check

The distance between two points is a calculation that was carried out multiple times throughout the Behaviour Tree, in different nodes. This reused section of code goes against everything that Encapsulation defines so a node was created to deal with this calculation alone (See Figure 30). This node simply returns true if the entity is within the desired distance of the target and returns false if not. By creating this separate node I can modify the way that the Behaviour Tree runs by placing these nodes before other nodes when a distance check is needed. The node can also be set as a child to an Inverter node, allowing for the functionality within this class to be completely inverted wherever needed.



```
bool TargetCheck::Run(int index)
{
    // If we are closer than 0.1f
    if (MyMath::GetDistance(Renderer::m_blackboard.b_entityPosition[index].x, Renderer::m_blackboard.b_entityPosition[index].y,
        Renderer::m_blackboard.b_targetPosition[index].x, Renderer::m_blackboard.b_targetPosition[index].y) < 0.2f) return true;
    return false;
}
```

Figure 30 : TargetCheck node class



```
bool Fight::Run(int index)
{
    if(Renderer::m_blackboard.b_health[index] <= 0) return true;

    // If we are in fight mode
    if(Renderer::m_blackboard.b_fightMode[index])
    {
        for (int i = 0; i < Renderer::m_blackboard.b_weight.size(); i++)
        {
            if (index == i) continue;

            XMFLOAT2 entityPos = Renderer::m_blackboard.b_entityPosition[index];
            XMFLOAT2 targetPos = Renderer::m_blackboard.b_entityPosition[i];
            float distance = MyMath::GetDistance(entityPos.x, entityPos.y, targetPos.x, targetPos.y);

            if (distance > 0.1f) continue;

            float entityWeight = Renderer::m_blackboard.b_weight[index];
            float otherWeight = Renderer::m_blackboard.b_weight[i];

            if (entityWeight > otherWeight)
            {
                float multiplier = entityWeight / otherWeight;
                Renderer::m_blackboard.b_health[i] = Renderer::m_blackboard.b_health[i] - (0.5f * multiplier);
            }

            // If the entity we are on is lighter than the other entity
            if (entityWeight < otherWeight)
            {
                float multiplier = entityWeight / otherWeight;
                Renderer::m_blackboard.b_health[i] = Renderer::m_blackboard.b_health[i] - (0.5f * multiplier);
            }

            return true;
        }
    } else return true;
}
```

Figure 31 : Fight node code

3.2.5. The Repeater Node

The largest obstacle that arose during the implementation of the behaviour tree revolved around an unknown outcome from the behaviour tree itself. In this case, the entities would approach the closest source of water at which point they would enter the Drink leaf node stage, filling up the entities value storing its thirst, represented as a floating-point value (See Figure 32Figure 32).

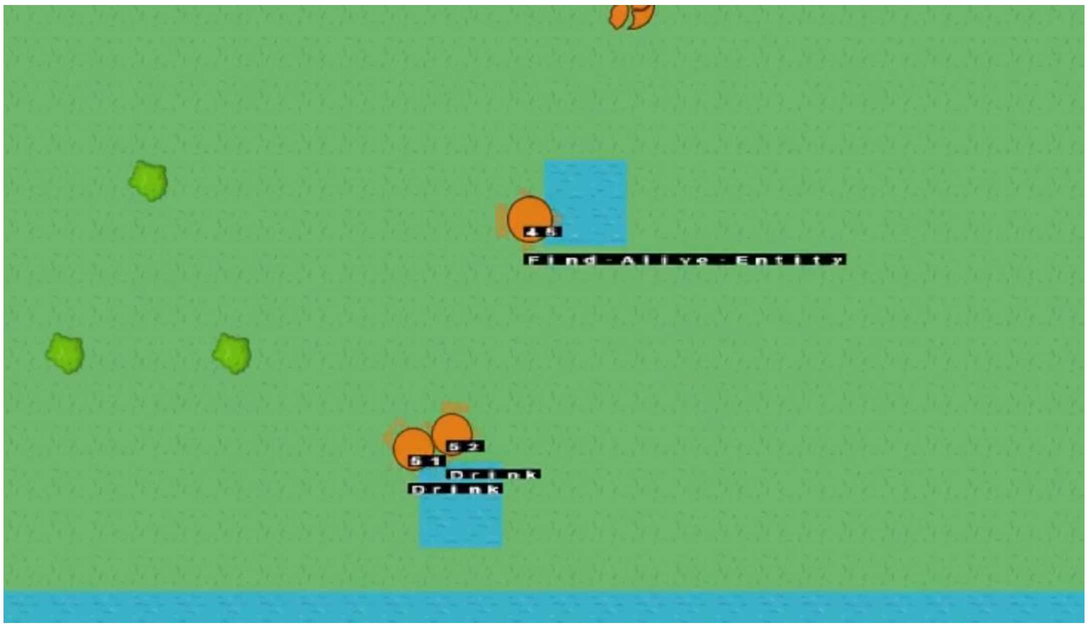


Figure 32 : Drinking Error - https://youtu.be/Gp7591nWsGA

The issue was that the entities would only enter the Drink node once and then not revisit it again, taking control of the behaviour tree within a loop, until the entity was killed. To fix this issue, a repeater node was first added to the application, derived from the previously defined class DecoratorNode. This node was an attempt at fixing the issue.

The Repeater node simply runs its child nodes until they return a fail at which moment the behaviour tree can continue. Although a Repeater node was not the only solution to this problem, it proved to work substantially better than expected by allowing the entities to complete important tasks such as drinking and eating without the behaviour tree becoming interrupted, which was the cause of the issue initially. This argument could also be seen as a negative approach by forcing entities to complete tasks without regard for others. An example of where this could influence the entities behaviour poorly is with an entity being attacked whilst eating. In a real example, the prey would forget all regard for eating and flee from the predator so to simulate this an extra section was added to the behaviour tree based around a Repeater node (See Figure 33Figure 33).

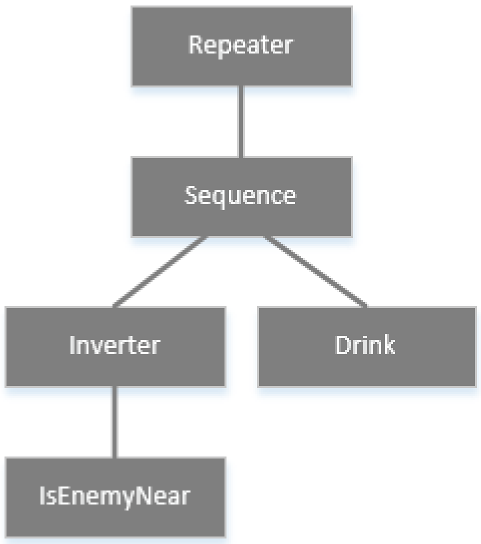
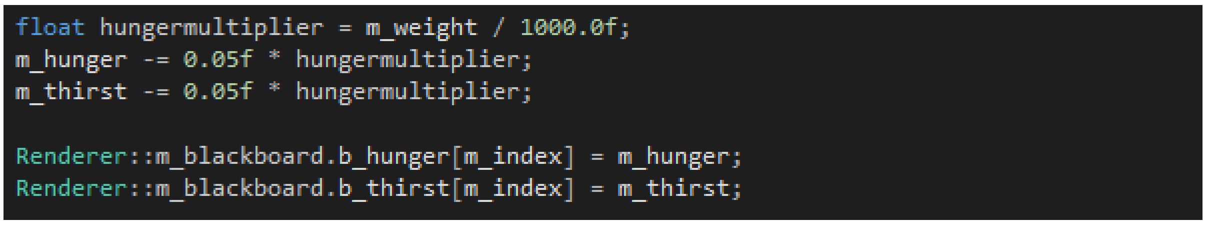


Figure 33 : Repeater Drink node section

Unfortunately after adapting the new system using a Repeater node, it was found that the node would continue to run within its loop without any regard for the rest of the program. This caused the application to not run the draw functions, meaning that entities taking a longer time to drink would become stationary on the screen, updating all of the data once the entity had finished drinking. Although this system is not being used for this scenario, creating a repeater node was an appropriate move, allowing for other situations to use it such as classes that do not rely on the rendering of graphics every frame.

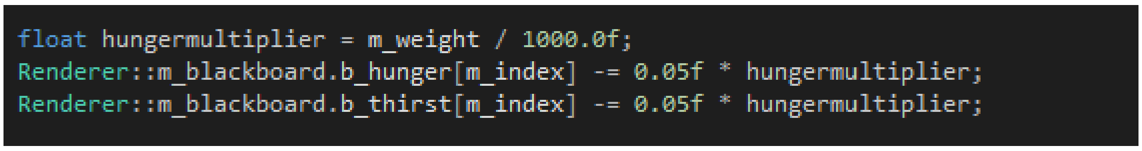
My initial analysis of the issue proved incorrect once the code was revisited multiple times. A problem originally believed to be related to a fault within the MoveTowards function was replaced by a fault relating to the entities member variables. Each frame, the floating point variables m_hunger and m_thirst were decremented depending on their weight and set to the values stored within the blackboard. This caused an inconsistency between two different variables being used where the blackboards values were being set from the member variables and not the blackboard itself (See Figure 34).



```
float hungermultiplier = m_weight / 1000.0f;
m_hunger -= 0.05f * hungermultiplier;
m_thirst -= 0.05f * hungermultiplier;

Renderer::m_blackboard.b_hunger[m_index] = m_hunger;
Renderer::m_blackboard.b_thirst[m_index] = m_thirst;
```

Figure 34 : Drink issue (before)

The fix to this issue involved no extra lines of code but simply to decrement the hunger and thirst values from within the blackboard (See Appendix C, Video C.10). This solution also led to a cleaner section of code with less calculations present. To conclude, the result of having this issue has led me to create a new node, which if not used in this project, can be used in others. The knowledge of this node and how it can be used is also one that I will continue to expand on.



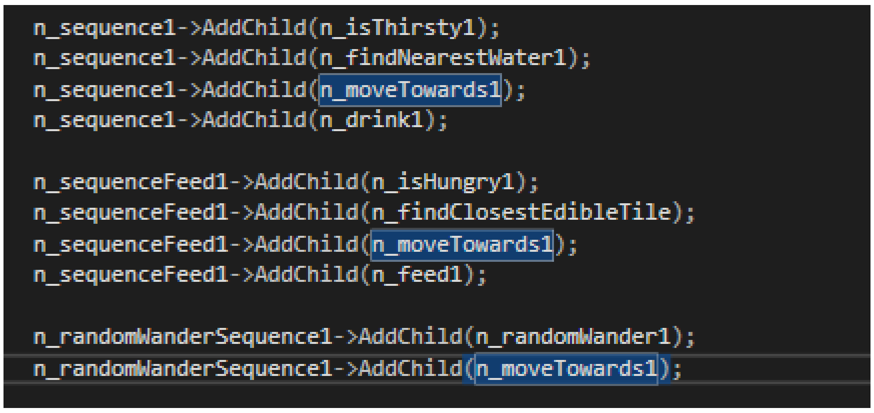
```
float hungermultiplier = m_weight / 1000.0f;
Renderer::m_blackboard.b_hunger[m_index] -= 0.05f * hungermultiplier;
Renderer::m_blackboard.b_thirst[m_index] -= 0.05f * hungermultiplier;
```

Figure 35 : Drink issue (after)

3.2.6. Encapsulation

The concept of Encapsulation is one that the Behaviour Tree was designed from very carefully with every new node being created with this idea in mind. Each node only tackles a specific task with no extra unnecessary features included alongside it. This is so the nodes inside the Behaviour Tree can be created and understood without need to alter other nodes or code.

This concept also allows for nodes to be reused multiple times while only needing to create and declare one class. An example of this can be found in Figure 36. If Encapsulation was not used the nodes may appear unorganised and non-reusable.



```
n_sequence1->AddChild(n_isThirsty1);
n_sequence1->AddChild(n_findNearestWater1);
n_sequence1->AddChild(n_moveTowards1);
n_sequence1->AddChild(n_drink1);

n_sequenceFeed1->AddChild(n_isHungry1);
n_sequenceFeed1->AddChild(n_findClosestEdibleTile);
n_sequenceFeed1->AddChild(n_moveTowards1);
n_sequenceFeed1->AddChild(n_feed1);

n_randomWanderSequence1->AddChild(n_randomWander1);
n_randomWanderSequence1->AddChild(n_moveTowards1);
```

Figure 36 : Reused MoveTowards Node Example

3.2.7. What was learnt

Throughout the design and implementation of the behaviour tree many skills have been learnt from discovering what a Blackboard is and how it can be used to writing reusable and recursive classes. Before this project began I had a very narrow idea of what behaviour trees were and how they could be used but after creating my own and incorporating it into a DirectX project, I feel extremely comfortable creating and writing nodes for Behaviour Trees within more than just one language.

I have learnt how to create a node class with an overwritten function that also allows for derived classes to be created based from it. I have also learnt how nodes can be created used in a certain sequence to perform specific tasks.

The completed Behaviour Tree can be found in Appendix D.

3.2.8. Flocking with Behaviour Trees

Once the Behaviour Tree had been set up it seemed relevant to incorporate the previously working flocking system into this however the flocking had been set up to work with no target location whereas the Behaviour Tree nodes dealing with movement defined a target location to move towards. This led to problems finding a middle ground where the entities could calculate their velocity based on their target position as well as move around in proximity with the velocity determined from the Cohesion, Separation and Alignment functions. Figure 37 provided evidence of this.

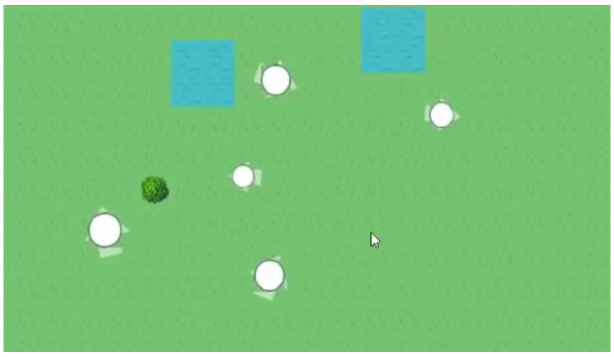


Figure 37 : Flocking with BT - https://youtu.be/NVwycLixm6I

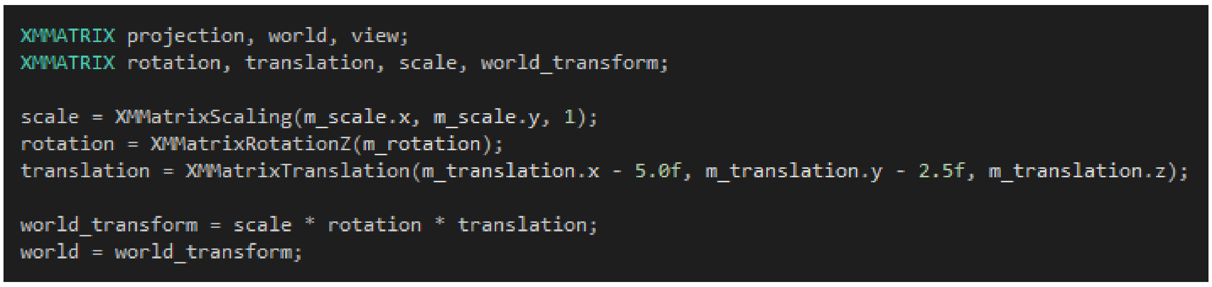
A solution to this issue would require a large amount of time to resolve by finding a way to incorporate the flocking functions into the Behaviour Tree. Although this would take time it would not be a daunting task as the functionality of the functions have already been set-up to return a velocity modifier. It will be in the best interest of this project to allow more time to be focused on the completing of the Behaviour Tree and less time focusing on fixing this feature, as implementing a flocking system has already been demonstrated within this project.

3.3. Initial Implementation

3.3.1. Creating entities

The principles of Inheritance were used when designing and implementing the entity class. A base class was set up containing information such as health, speed and size with two child classes derived from this representing a carnivore and herbivore. These derived classes were used to layout the extra functionality of the entities for both carnivores and herbivores by overriding certain functions to tackle different tasks. For example, the carnivore will find food by potentially killing another entity and then eating it whereas as a herbivore will feed on the bushes and grass and not hunt at all, unless specifically specified.

The entity class also includes an Image object which allows the entity to be rendered onto the screen as well as any other objects that use the Image class. The Image uses the World View Projection matrix to render the entity in its current position by initially accessing its position through a function call from the Entity to the Image (See Figure 38). Later in the project it was decided that a blackboard would be used instead so now the Image object receives its position from the blackboard.

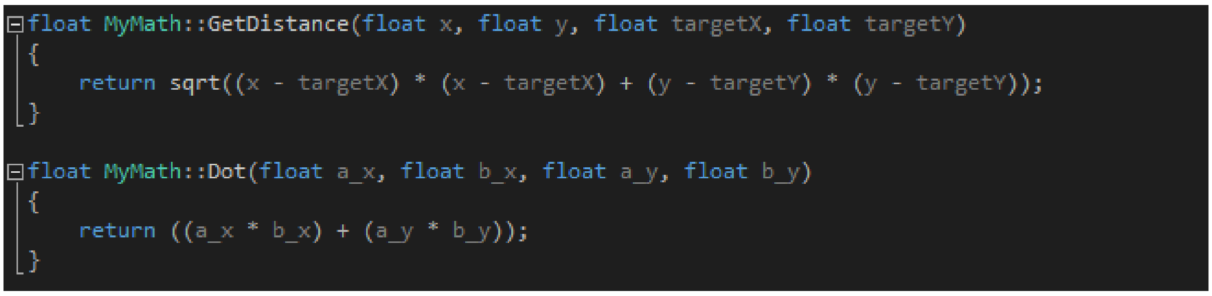


```
XMMATRIX projection, world, view;
XMMATRIX rotation, translation, scale, world_transform;

scale = XMMatrixScaling(m_scale.x, m_scale.y, 1);
rotation = XMMatrixRotationZ(m_rotation);
translation = XMMatrixTranslation(m_translation.x - 5.0f, m_translation.y - 2.5f, m_translation.z);

world_transform = scale * rotation * translation;
world = world_transform;
```

Figure 38 : WVP Matrix Code

3.3.2. MyMath class

Part-way through the development of the project, soon after the rendering classes had been set up, a MyMath class was created. This class was populated with static functions to deal with maths functions such as getting the distance between two points or finding the dot product of two vectors.



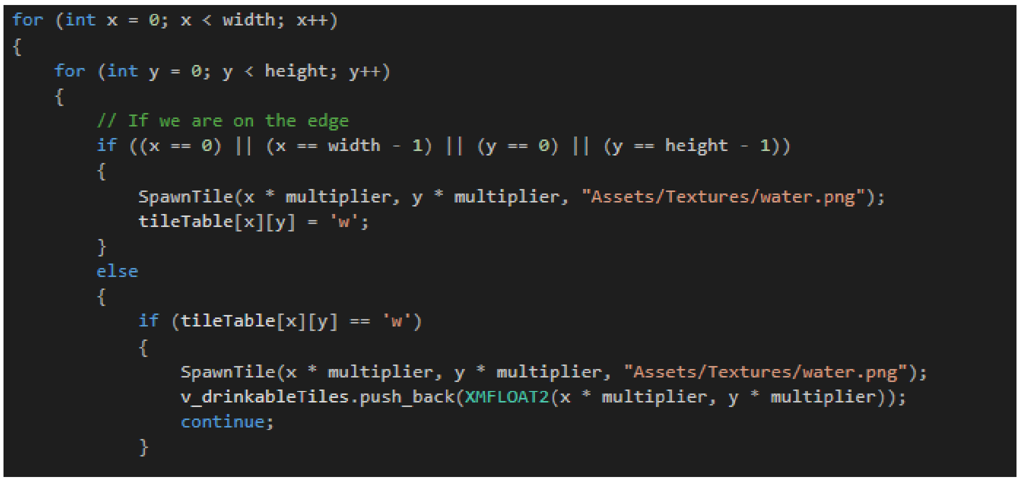
```
float MyMath::GetDistance(float x, float y, float targetX, float targetY)
{
    return sqrt((x - targetX) * (x - targetX) + (y - targetY) * (y - targetY));
}

float MyMath::Dot(float a_x, float b_x, float a_y, float b_y)
{
    return ((a_x * b_x) + (a_y * b_y));
}
```

Figure 39 : MyMath class

3.3.3. Procedural Generation

To simulate a different experience every time the program is run, Procedural Generation is used. As the view only consists of a few unique tiles including grass, hedge, water and trees, setting up the program to redesign the level every time the application is opened was very straight forward.

Firstly, the width and height of the tile grid is defined so the program knows how many tiles to generate. A nested for loop is added which creates a new tile for every element within the two dimensional loop. To allow for a variety of tiles, a random number is generated up to one hundred and used as a percentage. If this value falls between two specified numbers, a tile of certain type will be spawned using the index of the loop as its spawn position (See Figure 40). A video evidencing this generation can be found in Apendix C, C.11.

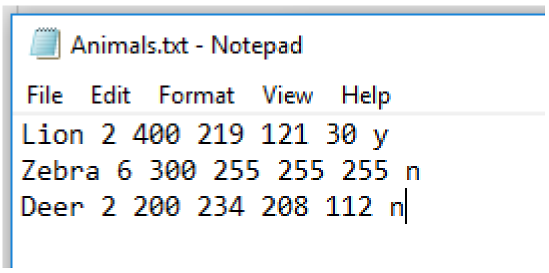


```
for (int x = 0; x < width; x++)
{
    for (int y = 0; y < height; y++)
    {
        // If we are on the edge
        if ((x == 0) || (x == width - 1) || (y == 0) || (y == height - 1))
        {
            SpawnTile(x * multiplier, y * multiplier, "Assets/Textures/water.png");
            tileTable[x][y] = 'w';
        }
        else
        {
            if (tileTable[x][y] == 'w')
            {
                SpawnTile(x * multiplier, y * multiplier, "Assets/Textures/water.png");
                v_drinkableTiles.push_back(XMFLOAT2(x * multiplier, y * multiplier));
                continue;
            }
```

Figure 40 : Procedural Generation Code

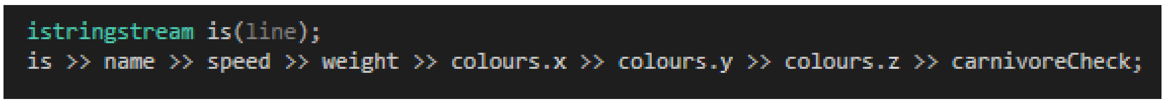
3.3.4. Creating personal animals

When creating the buttons to deal with spawning entities a text file importer was quickly added to accompany this. Firstly the text file storing information relating to the entities is read in as lines (See Figure 41). The data is split up and then placed into variables using the istringstream input (Microsoft, 2018) (See Figure 42).



```
Lion 2 400 219 121 30 y
Zebra 6 300 255 255 255 n
Deer 2 200 234 208 112 n
```

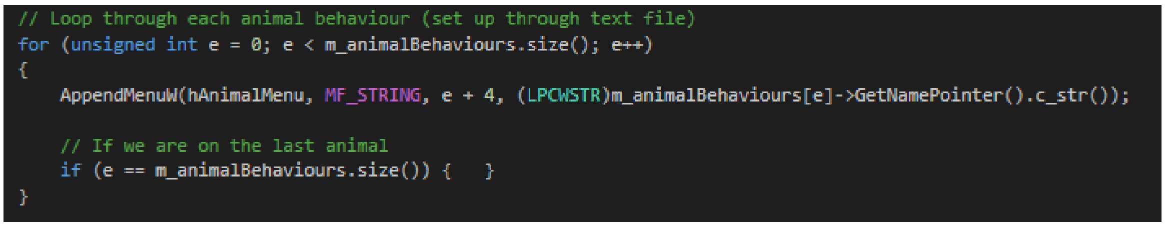
Figure 41 : Text file example (y & n representing either carnivore or herbivore)



```
istringstream is(line);
is >> name >> speed >> weight >> colours.x >> colours.y >> colours.z >> carnivoreCheck;
```

Figure 42 : istringstream example

Once the text file has all of the entities included that the user wants, with all of the fields entered represented in the order of Name, Speed, Weight, R, G, B and the animal type, it can be saved (See Figure 43. Once in the program the button list will change depending on the entered entities, with all of the attributes set to the personalised integers. See Figure Figure 49 for evidence on the buttons)



```
// Loop through each animal behaviour (set up through text file)
for (unsigned int e = 0; e < m_animalBehaviours.size(); e++)
{
    AppendMenuW(hAnimalMenu, MF_STRING, e + 4, (LPCWSTR)m_animalBehaviours[e]->GetNamePointer().c_str());

    // If we are on the last animal
    if (e == m_animalBehaviours.size()) {    }
}
```

Figure 43 : Button code

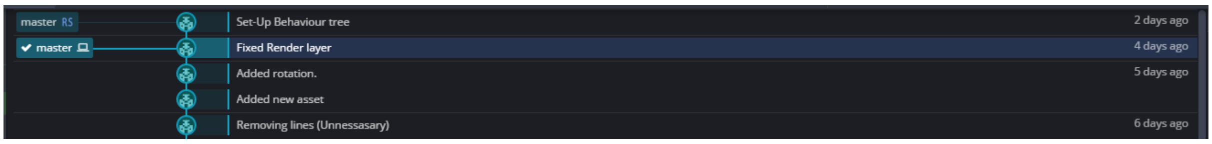
3.1. Project Management and planning

The methodology styles declared within the progress report were followed through for the entirety of the project however some content displayed within the Ghant chart and Work Breakdown Structure diagram went left unimplemented for various reasons.

3.1.1. GitHub/Gitkraken

Using the Git repository was a task that was carried out multiple times a week through the process of pushing and pulling the project code. This tool was only useful in allowing me to revert my work back to a previous date but also allowed me to visualise the changes in the workflow by what tasks had been completed in which order.

Using GitHub also resolved the issue of depending on a portable hard drive to carry the project from one location to the over. Instead the work could be carried out at home, pushed to the Git Repository and then pulled from another location such as University. Using GitHub allowed for a constant backup to be made during the entire length of the project.

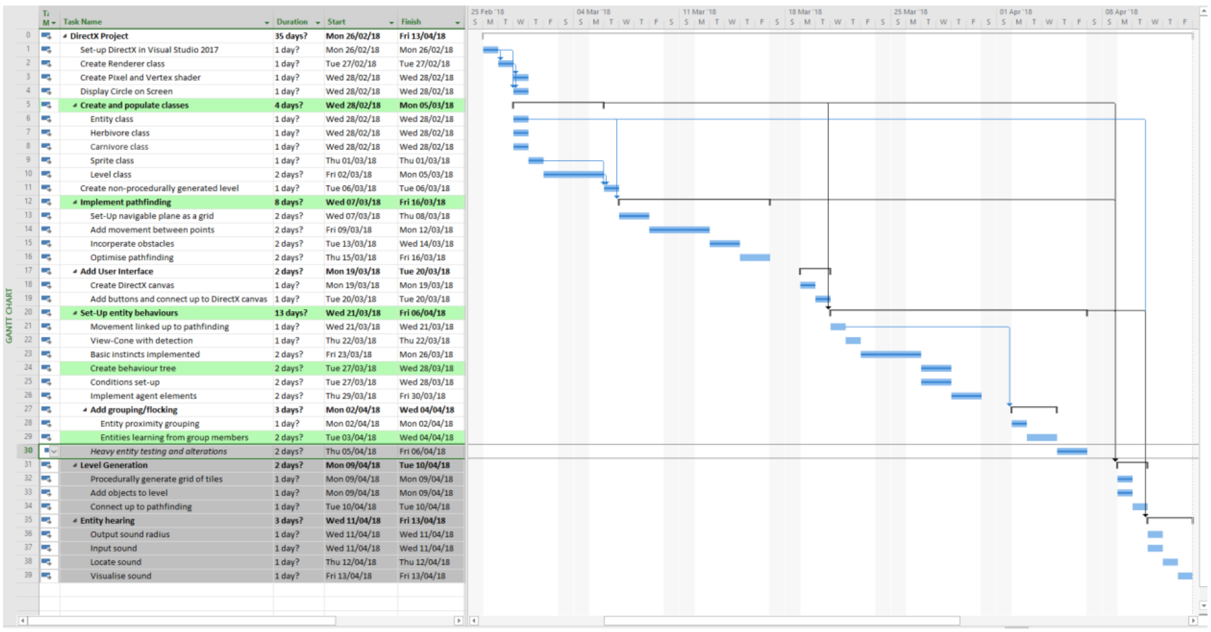


```
master RS     Set-Up Behaviour tree                 2 days ago
✔ master      Fixed Render layer                    4 days ago
              Added rotation.                       5 days ago
              Added new asset
              Removing lines (Unnessasary)          6 days ago
```

Figure 44 : GitKraken Example

3.1.2. Gantt Chart

The Gantt Chart acted as a list of all the tasks that would need to be implemented into the project. The chart allowed for all of the tasks to contain a duration, start date and end date, meaning I knew exactly when I was either ahead or behind of schedule. The calendar section of the Gantt chart displayed the tasks as a solid line from the start to end date, allowing me to see how large each task was in comparison to the rest (See Figure 45).



GANTT CHART

	Ta M	Task Name	Duration	Start	Finish
0		**DirectX Project**	**35 days?**	**Mon 26/02/18**	**Fri 13/04/18**
1		Set-up DirectX in Visual Studio 2017	1 day?	Mon 26/02/18	Mon 26/02/18
2		Create Renderer class	1 day?	Tue 27/02/18	Tue 27/02/18
3		Create Pixel and Vertex shader	1 day?	Wed 28/02/18	Wed 28/02/18
4		Display Circle on Screen	1 day?	Wed 28/02/18	Wed 28/02/18
5		**Create and populate classes**	**4 days?**	**Wed 28/02/18**	**Mon 05/03/18**
6		Entity class	1 day?	Wed 28/02/18	Wed 28/02/18
7		Herbivore class	1 day?	Wed 28/02/18	Wed 28/02/18
8		Carnivore class	1 day?	Wed 28/02/18	Wed 28/02/18
9		Sprite class	1 day?	Thu 01/03/18	Thu 01/03/18
10		Level class	2 days?	Fri 02/03/18	Mon 05/03/18
11		Create non-procedurally generated level	1 day?	Tue 06/03/18	Tue 06/03/18
12		**Implement pathfinding**	**8 days?**	**Wed 07/03/18**	**Fri 16/03/18**
13		Set-Up navigable plane as a grid	2 days?	Wed 07/03/18	Thu 08/03/18
14		Add movement between points	2 days?	Fri 09/03/18	Mon 12/03/18
15		Incorperate obstacles	2 days?	Tue 13/03/18	Wed 14/03/18
16		Optimise pathfinding	2 days?	Thu 15/03/18	Fri 16/03/18
17		**Add User Interface**	**2 days?**	**Mon 19/03/18**	**Tue 20/03/18**
18		Create DirectX canvas	1 day?	Mon 19/03/18	Mon 19/03/18
19		Add buttons and connect up to DirectX canvas	1 day?	Tue 20/03/18	Tue 20/03/18
20		**Set-Up entity behaviours**	**13 days?**	**Wed 21/03/18**	**Fri 06/04/18**
21		Movement linked up to pathfinding	1 day?	Wed 21/03/18	Wed 21/03/18
22		View-Cone with detection	1 day?	Thu 22/03/18	Thu 22/03/18
23		Basic instincts implemented	2 days?	Fri 23/03/18	Mon 26/03/18
24		Create behaviour tree	2 days?	Tue 27/03/18	Wed 28/03/18
25		Conditions set-up	2 days?	Tue 27/03/18	Wed 28/03/18
26		Implement agent elements	2 days?	Thu 29/03/18	Fri 30/03/18
27		**Add grouping/flocking**	**3 days?**	**Mon 02/04/18**	**Wed 04/04/18**
28		Entity proximity grouping	1 day?	Mon 02/04/18	Mon 02/04/18
29		Entities learning from group members	2 days?	Tue 03/04/18	Wed 04/04/18
30		*Heavy entity testing and alterations*	2 days?	Thu 05/04/18	Fri 06/04/18
31		**Level Generation**	**2 days?**	**Mon 09/04/18**	**Tue 10/04/18**
32		Procedurally generate grid of tiles	1 day?	Mon 09/04/18	Mon 09/04/18
33		Add objects to level	1 day?	Mon 09/04/18	Mon 09/04/18
34		Connect up to pathfinding	1 day?	Tue 10/04/18	Tue 10/04/18
35		**Entity hearing**	**3 days?**	**Wed 11/04/18**	**Fri 13/04/18**
36		Output sound radius	1 day?	Wed 11/04/18	Wed 11/04/18
37		Input sound	1 day?	Wed 11/04/18	Wed 11/04/18
38		Locate sound	1 day?	Thu 12/04/18	Thu 12/04/18
39		Visualise sound	1 day?	Fri 13/04/18	Fri 13/04/18

Figure 45 : Gantt Chart Finished

The Gantt chart also provided the feature of being able to display how complete a task was (in percent), with a darker line through the main one (See Figure 46). This meant I could focus on a section of a larger task, mark it as a certain percentage completed and then when going back to the work on another day, I would know where roughly I led off.

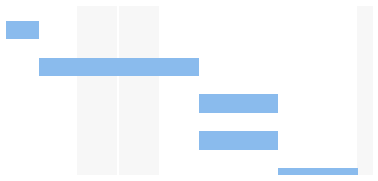
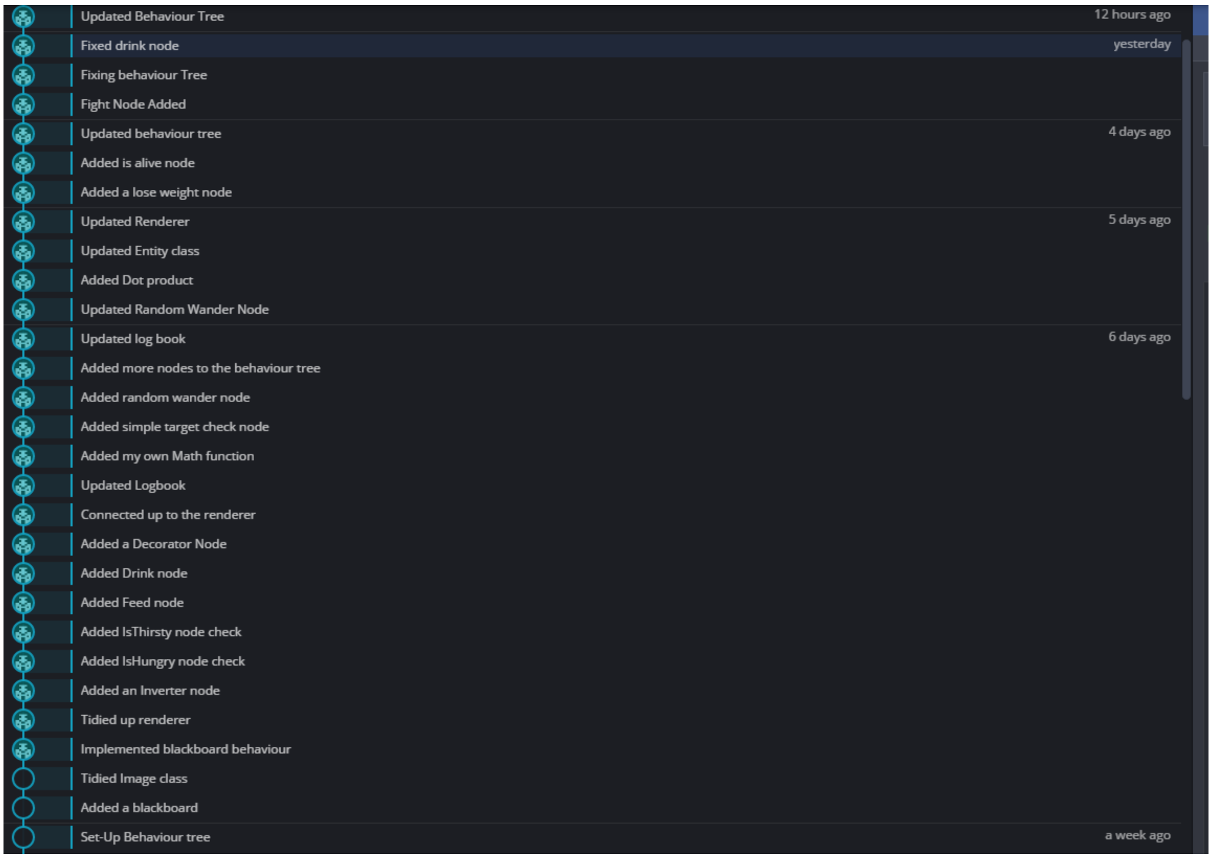


Figure 46 : Gantt timeline

There were a great number of tasks that, when comparing the initial predicted length to the actual length, were far off. An example of this relates to the implementation of the Behaviour Tree. As seen in the Gantt chart in Figure UP, the predicted length of time to complete the fully working Behaviour Tree was a total of six days, assuming that the behaviours had been set up prior. This task however took almost two weeks, more than double the length than first expected (See Figure 47Figure 44).



```
Updated Behaviour Tree                          12 hours ago
Fixed drink node                                yesterday
Fixing behaviour Tree
Fight Node Added
Updated behaviour tree                          4 days ago
Added is alive node
Added a lose weight node
Updated Renderer                                5 days ago
Updated Entity class
Added Dot product
Updated Random Wander Node
Updated log book                                6 days ago
Added more nodes to the behaviour tree
Added random wander node
Added simple target check node
Added my own Math function
Updated Logbook
Connected up to the renderer
Added a Decorator Node
Added Drink node
Added Feed node
Added IsThirsty node check
Added IsHungry node check
Added an Inverter node
Tidied up renderer
Implemented blackboard behaviour
Tidied Image class
Added a blackboard
Set-Up Behaviour tree                           a week ago
```

Figure 47 : GitKraken BT Commits

A distinctive example of how the Gantt chart was not followed to the exact tasks would be the rearrangement of the Procedurally Generated aspects. After testing simple movements of the entities it was realised that an algorithm such as the A* Pathfinding would not be a viable solution to the navigation of entities upon the level due to the high range of memory allocation that would be needed. Instead, a simple flocking algorithm was used which gave a much more fluid and natural looking results. Using the A* Pathfinding solution would also result in every entity having complete knowledge of each tile and how to get there but in a real example, an animal would not have that knowledge.

3.1.3. Work Breakdown Structure

The Work Breakdown Structure (WBS) diagram was an extremely useful tool alongside the Gantt Chart. The WBS diagram allowed the tasks that needed to be completed to be shown in a format that was ordered and clear, whilst allowing the tasks to have some sense of leeway in regard to implementation. The aspect of the WBS diagram that kept the implementation of the project moving was the process of categorising the tasks into sections that could not be tackled until the previous were completed. For example, the Pathfinding section could not be started until the previous Create Window group of tasks had been completed, as the pathfinding could not be visualised without the implementation of a window (See Figure 48).

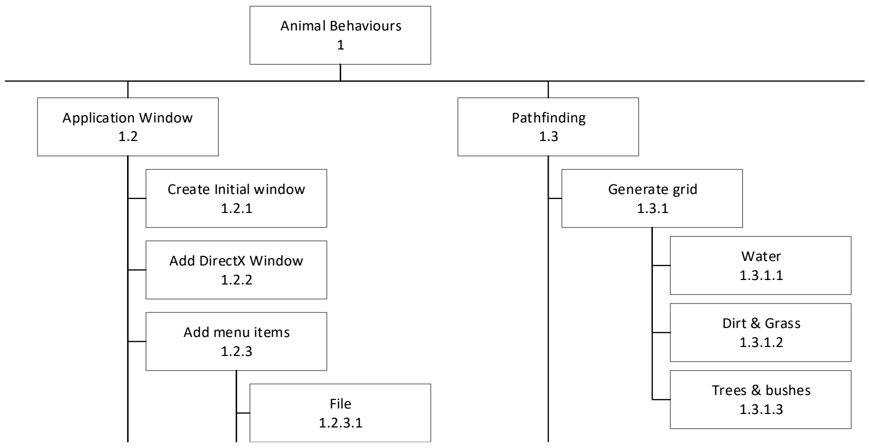


Figure 48 : WBS diagram snippet

Although the pathfinding section of the WBS diagram contained a large number of tasks, the majority of them were not completed due to the fact that a pathfinding algorithm such as A* would not be able to handle a project with this scope. I anticipated this becoming an issue beforehand and so went on to implementing a procedurally generated scene alongside the entity and its attributes (see figure 5 for WBS diagram). The only tasks that went unused were those of the A* algorithm itself and instead a simple chase and flee concept was incorporated which mimics animal movement in a more realistic way than the A* would.

3.1.4. Visual Prototype Changes

My visual prototype included an on-screen user interface which included a drop down containing various different buttons. After initially trying to implement this feature I discovered difficulties getting the text to display perfectly over the button. This was due to working with two different positioning coordinates, the button texture using world position and the text using screen position from -1 to 1.

As I wanted to spend more time on the functionality of the entities I decided to use the win32 HMenu which allowed me to create menu items on the Windows Menu bar without any issues. The set-up was extremely easy and as it uses a docking system entity buttons could be added into separate folders of Carnivore and Herbivore without any extra problems (See Figure 49).

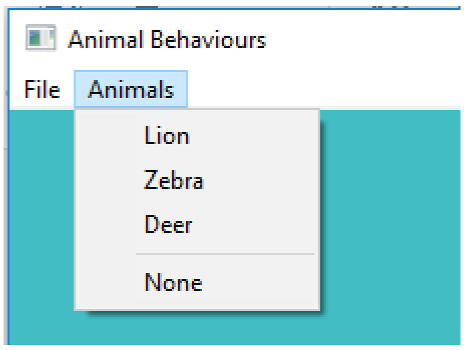


Figure 49 : Menu Bar

3.2. Issues along the way

As the project led me into an area that I was unfamiliar with I came across many problems that either slowed the rate of my work down or left me unsure for days.

3.2.1. Lack of documentation

As DirectX 11 has such a broad amount of functionality that can be implemented it was difficult to find solutions to many issues that presented themselves. Many fixes that could be found online were either too specific to one task or only dealt with a solution tailored for heavy 3D use and not something that could be easily converted into the 2D scene.

An example of an issue that I ran into was during implementation of the on-screen entities. A 4D Matrix was used to represent the World, View and Projection of each entity, with the final dimension used for translation of the matrix. Once set up it was easy to translate and rotate the entities however after adding text to the screen I found it difficult to keep the text in the correct position (See Figure 50).

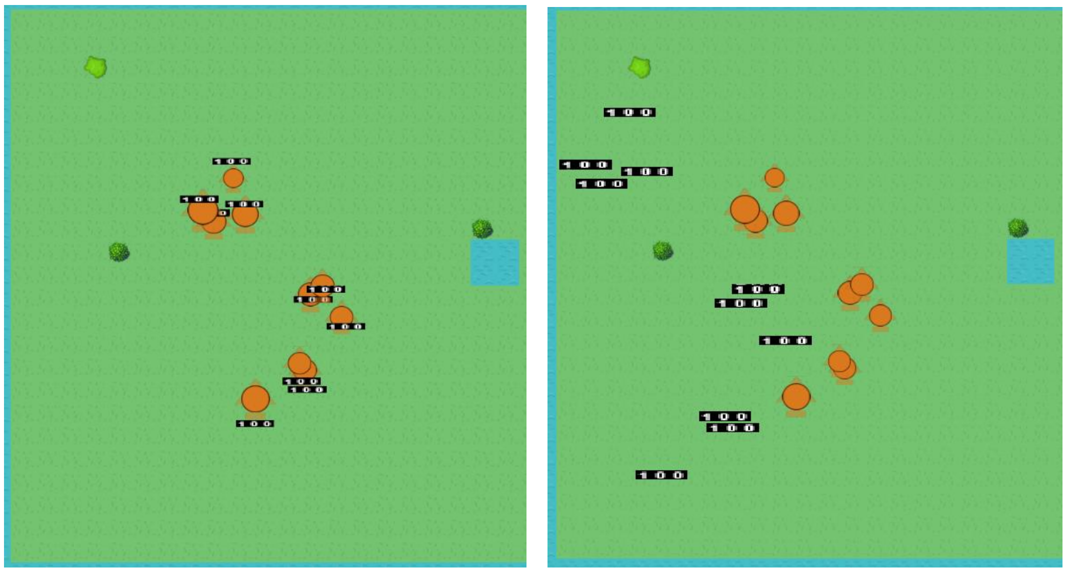


Figure 50 : Text Displayed Fixed and Unfixed

This issue was caused by the use of two different coordinate system, as mentioned earlier in section 3.1.4. In this case the text had no trouble translating correctly to the entities at the starting position, but as the orthographic camera size value changes the text becomes offset. It was very difficult to find resources online that not only explained how the translation between world coordinates to screen coordinates worked but did this in way that could be translated into a DirectX format. The fact that DirectX has many different ways to render a single texture did not also aid in my solution. From looking back through an old project I found the solution to this issue which involves using a second matrix to change the value of each element depending on the desired outcome.

3.2.2. Project Duration

I found it extremely difficult to work towards my goal of implementing a fully working behaviour tree which controls entities within a scene through the use of adaptive learning. The task in itself was far too large to be implemented and then polished ready to be used so a smaller section of this overall task was chosen. By the end of the project a Behaviour Tree has been implemented for entities that fall under both Carnivores and Herbivores.

3.5. Machine Learning

Learning was a concept that from the start seemed difficult and time consuming but only from tackling this project front hand could that be truly understood to its full extend. Although the ambition was high and the outcome provided nothing, in terms of learning, this has still been an experience where learning has taken place.

From the research carried out in section 2.3, two main learning concepts were defined as well as a third set of information relating to reinforcement learning which was the approach that was going to be tackled. After defining the window and providing the screen with graphics, entities had to be set up, with basic behaviours to allow them to move for the learning algorithm to then pick up from. An algorithm defined by learning needs to have a substantial amount of data and information to work from, so in my case there were not enough triggers to activate a detailed learning implementation.

In hindsight it may have been a better decision to create the application using Unity 3D as to allow myself to fully focus on the Behaviour Tree. This mind-set however is based around spending an enormous amount of time on one

Behaviour Tree whereas the intentions I had were to create an application that performed the actions that were needed of them in a basic level to then expand on in further dimensions or different languages later on.

4. Testing

An archive of videos have been recorded during the implementation of this project. A Log Book has also been written that documents various tasks that were completed or tackled with evidence and a brief description of what is entailed. The videos can be found in Appendix section C and the Log Book in the Resources folder.

5. Reflections and Conclusions

5.1. Evaluation

The project aims specified within the Progress Report have been tackled throughout this project however my first goal of implementing learning into the entities fell short. Although this concept was left unimplemented a lot was learnt about this topic during the research phase as a huge amount of information was needed before tackling Machine Learning.

I do feel confident however with the implementation of the other aims including having the application running at a high frame rate and allowing the entities to consist of classes where future development can be done. I am certainly pleased with how the creation and development of the Behaviour Tree allowed me to create these classes with the future in mind, setting up the nodes to allow continues tasks to be added and adding them to the Behaviour Tree.

The majority of the objectives were completed for this project including creating a simple level to allowing animals to have different behaviours from one another. I did not however include a few of these points due to time constraints and the length of the project. I do however plan on evolving the project to include features such as these within the near future.

5.2. Reflection

Incorporating a Behaviour Tree into this project was a road that was not fully defined at the start, but is a route I am glad I went down. Adding this mechanic allowed for a much simpler way of controlling AI entities which could be expanded or contracted with ease. The knowledge alone gained from this project has allowed me to feel comfortable in the development of a Behaviour Tree, regardless of dimension or programming language. I also underestimated the ease of creating and adding nodes to the overall tree and how briefly the nodes could be set-up.

If I had tackled this project again I would have set up the MoveTowards and RandomWander nodes differently by having them set the position of the entities directly instead of modifying their velocity and then moving them every frame within the update loop. By implementing these classes the way I did, issues arose over entities moving past a desired location and incorporating the flocking into the Behaviour Tree movement.

The largest obstacle that was overcome was understanding how a Behaviour Tree would be implemented into the program. Initially each node was going to take in a variable that would be modified or used to an extend but after discussing the principles with Mark Bennett it was decided that a blackboard should be used. This concept appeared much more logical than any other found so this is what was used.

5.3. Future Development

If this project were to be tackled again, I would consider expanding the behaviour tree to take in weights, allowing for a much wider range of actions to be carried out. This is however a task that could be completed in my own time.

If the Behaviour Tree was made slightly simpler I would consider revising the learning concept initially discussed and implement an extremely basic form of learning. This would however require far more research and testing before the learning could be added to the Behaviour Tree.

As only sight has been used for this project to allow entities to know when others are nearby, I would like to add the sense of sound and a view cone to represent sight. These are features that were not relevant or important to the workings of the Behaviour Tree so were left out for this project however with no time limit these concepts could be implemented to give a far more realistic outcome.

Bibliography

Beal, V. (2005, 03 25). *Webopedia*. Retrieved from All about DIrectX: https://www.webopedia.com/DidYouKnow/Hardware_Software/directx.asp

Champandard, A. J. (2007, July 18). *Using a Static Blackboard to Store World Knowledge*. Retrieved from AIGameDev: A blackboard simply consists of data

Jamieson, D. (2012, September 27). *When Worlds Collide: Simulating Circle-Circle Collisions*. Retrieved from envatotuts+: https://gamedevelopment.tutsplus.com/tutorials/when-worlds-collide-simulating-circle-circle-collisions--gamedev-769

Kharkar, S. V. (2002). *AI Game Programming Wisdom*. Hingham, Massachusetts: Jenifer Niles.

Manslow, J. (2002). Learning and Adaptation. In S. Rabin, *AI Game Programming Wisdom*. Hingham, Massachusetrts: Jenifer Niles.

Mathews, J. (2002). In S. Rabin, *AI Game Programming Wisdom*. Hingham, Massachusetts: Jenifer Niles.

Microsoft. (2018). *<sstream> typedefs*. Retrieved from Microsoft: https://msdn.microsoft.com/library/d102edd2-ecea-4a35-a398-cf96e58dd422.aspx#istringstream

Microsoft. (2018). *DirectX Graphics and Gaming*. Retrieved from Microsoft: https://msdn.microsoft.com/en-us/library/windows/desktop/ee663274(v=vs.85).aspx

Microsoft. (2018). *Visual Studio*. Retrieved from Microsoft: https://www.visualstudio.com/

Microsoft. (2018). *XMFLOAT2 structure*. Retrieved from Microsoft: https://msdn.microsoft.com/en-us/library/microsoft.directx_sdk.reference.xmfloat2(v=vs.85).aspx

Niv, Y. (2009). *The Neuroscience of Reinforcement Learning*. Retrieved from princeton: http://www.princeton.edu/~yael/ICMLTutorial.pdf

OpenGL. (2018). *The Industry's Foundation for High Performance Graphics*. Retrieved from OpenGL: https://www.opengl.org/

Palmer, N. (n.d.). *Machine Learning in Games Development*. Retrieved from ai-depot: http://ai-depot.com/GameAI/Learning.html

Patel, A. (2017). *Introduction to A* from Amit's thoughts on pathfinding*. Retrieved from theory-stanford: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

Rabbin, S. (2015). *Game AI pro 2 collected wisdom of game AI professionals*. CRC Press.

scrumalliance. (n.d.). *Learn About Scrum*. Retrieved from scrumalliance: https://www.scrumalliance.org/why-scrum

Shesterkin, D. (n.d.). *BirdFlock*. Retrieved from GitHub: http://black-square.github.io/BirdFlock/

Simpson, C. (2014, 07 17). *Gamasutra*. Retrieved from Behavior trees for AI: How they work: https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

Smarty. (2012, May 29). *Using HMENU*. Retrieved from cplusplus: http://www.cplusplus.com/articles/LhvU7k9E/

Strømsvik, Ø. (2018). *A* pathfinding*. Retrieved from Twiik: http://twiik.net/resources/a-pathfinding

Unity. (2014). *Unity*. Retrieved from Collider 2D: https://unity3d.com/learn/tutorials/topics/2d-game-creation/collider-2d

Appendix

Appendix A – Research Literature (Progress Report)

Source 1 – Predator & Prey: Adaptations - Royal Saskatchewan Museum

This booklet contains some very detailed sections covering multiple different topics relating to hunting, hiding and anything else I might need to know when implementing a feature based around animals. This source even provides information on how the senses are used to utilise any actions carried out by the animals. While talking in a vague sense, the text also provides examples of specific animals which I can use as a template to get started.

Source 2 – Artificial Intelligence for Games Second Edition – Ian Millington, John Funge

This book provides information covering a huge range of topics including movement, flocking and processer issues.

Source 3 – AI for Game Developers – David M. Bourg & Glenn Seeman

The content majority within this book relates very closely to the movement of entities within a game world. A chapter that stands out to me is the second, Chasing and Evading, which goes into detail about implementing basic chasing and evading features alongside using line of sight to move entities towards others in their facing direction.

Source 4 – Game AI Pro 2 – Steve Rabin

This book is an excellent resource as it includes a chapter on Theta* Pathfinding, a pathfinding concept that focuses on smoother movement and adjusting to terrain.

Source 5 – AI Game Programming Wisdom – Steve Rabin

This book includes an area on Squad Movement which will provide information on how to develop a system where multiple entities can work together.

Appendix B - Project tool solutions (Progress Report)

There are three solutions to take into account before deciding how to move forward.

Solution 1 – Game Engine built application

A game engine is a plausible approach to developing my application in the case that entities can be set up extremely easy and within a short space of time. By using an engine such as Unity or Unreal Engine these entities can have collisions added to them as part of the engines package (Unity, 2014). This feature provides both positive and negative solutions to the development of the application by allowing new features to be added with ease but in doing so restricting the control of the code and what it does.

Using a game engine would allow for the application to be downloaded by others through the engines marketplace, giving the opportunity for this project to be monetized upon. Both Unity and Unreal Engine support cross platform development which would allow for the application to be viewed on more than just a PC.

Solution 2 – SDL built application

SDL is a great library used in the development of graphical applications. It can be used to develop 2D games with audio, input as well as file I/O abstraction, a key feature used in allowing entities to remember data from machine learning. SDL can also support 3D development but requires an external API such as OpenGL (OpenGL, 2018) or Direct3D. This factor has quite a large impact on the development of my project as I want the ability to further develop the application to allow for 3D graphics, without the use of an external API.

Solution 3 – DirectX built application

DirectX is a collection of APIs used in the implementation of graphics applications. It can be used to render both 2D textures and 3D models equipped with textures as well as working with audio and input. DirectX11 provides all the necessary ways of creating a 2D visual application using some more advanced features such as multi-threading, which will help in the rendering of a smooth program.

Although DirectX is far more complicated to understand and learn than an API such as SDL, it will allow me to develop a system that includes more powerful features without the need to incorporate any other layers or added software. It may be more difficult to work on but it will provide the user with a far more sophisticated program written with detailed and well-constructed code. Vangie Beal (Beal, 2005) discusses the components that DirectX has to offer, with the DirectSound component gaining my attention. (Beal, 2005) states that “DirectSound enables the playing of sounds with very low latency and gives applications a high level of control over hardware resources.” As I am aiming to implement sound at a later stage of the application this statement confirms that DirectSound will be the best choice when tackling sound.

Chosen Solution

Although using an engine would allow me to create a working application within the time limits of this project, I want to be able to show the skills I have learnt through my code and completed application and using an engine will not provide this. SDL is an appropriate solution for my project as it will allow me to create an application that completes all of the aims I have set, however I will be using DirectX11 as my development tool as this API is more powerful in terms of what it can do and what it can process. DirectX is also more likely to be used in the creation of games so this will provide me with a path into the industry. See Table 2: Tool Solution Table for breakdown table.

Graphics API	Will need extra API’s	Unnecessary features	Single Platform dependant	Prior knowledge of solution
Unity 5	✗	✓	✗	✓
Unreal Engine 4	✗	✓	✗	✗
SDL	✓	✗	✗	✓
DirectX 11	✗	✗	✗	✓

Table 2: Tool Solution Table

Appendix C

C.1: Rotation Testing - https://youtu.be/e8ZsoHAvkUE

C.2: Flocking - https://youtu.be/8PBZQDXJMd4

C.3: Behaviour Tree working - https://youtu.be/rvm5EnW0JMA

C.4: Behaviour Tree Random Wander - https://youtu.be/omFjLbqWdGQ

C.5: Approach Loop - https://youtu.be/DHjeSbSwxoo

C.6: Drinking Error - https://youtu.be/Gp7591nWsGA

C.7: Drinking Fixed - https://youtu.be/acvhF_Xz2hY

C.8: Simple Behaviour Tree (Not Thirsty) - https://youtu.be/2JXlMl2m7tc

C.9: Simple Behaviour Tree (Thirsty) - https://youtu.be/UZ3Wmk9ydHU

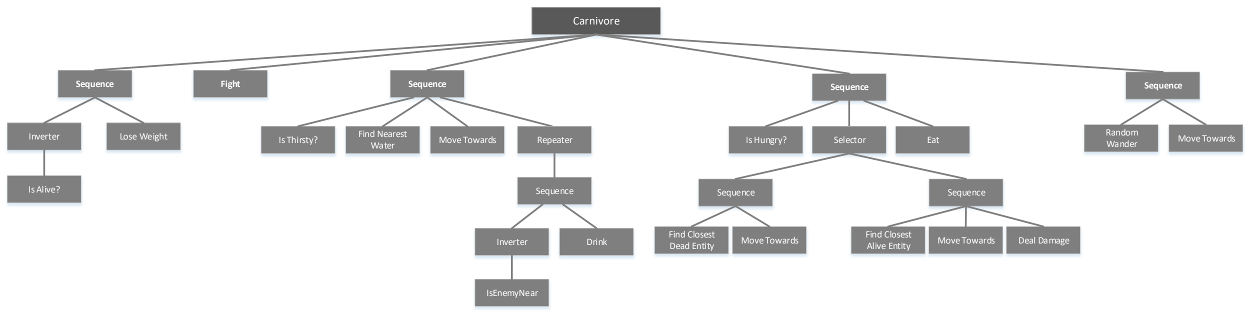
C.10: Thirst Error fix - https://youtu.be/rXimONZ8uvg

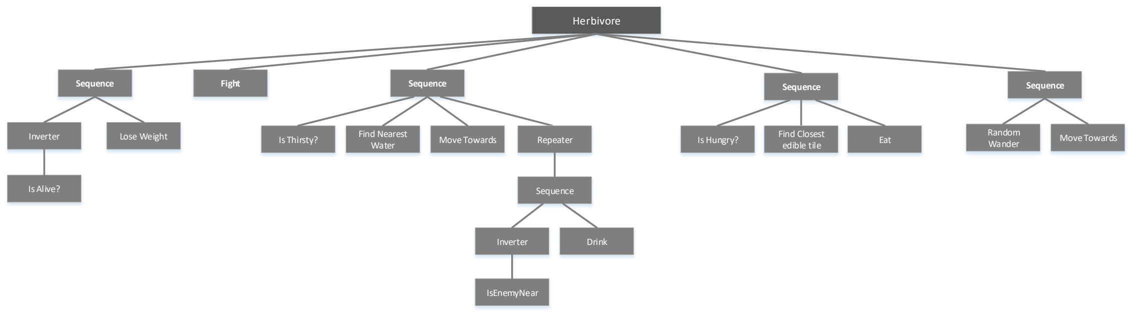
C.11: Circle Collisions - https://www.youtube.com/watch?v=RYdqpfktV-U

C.12: Flocking with BT - https://www.youtube.com/watch?v=NVwycLixm6I

Appendix D

D.1: Complete Behaviour Trees


Carnivore
Sequence
Fight
Sequence
Sequence
Sequence
Inverter
Lose Weight
Is Alive?
Is Thirsty?
Find Nearest Water
Move Towards
Repeater
Sequence
Inverter
Drink
IsEnemyNear
Is Hungry?
Selector
Eat
Sequence
Find Closest Dead Entity
Move Towards
Sequence
Find Closest Alive Entity
Move Towards
Deal Damage
Random Wander
Move Towards


Herbivore
Sequence
Fight
Sequence
Sequence
Sequence
Inverter
Lose Weight
Is Alive?
Is Thirsty?
Find Nearest Water
Move Towards
Repeater
Sequence
Inverter
Drink
IsEnemyNear
Is Hungry?
Find Closest edible tile
Eat
Random Wander
Move Towards