Garden Battle

An AI-focused game using Hierarchical State Machines combined with Behaviour Trees

Arturo Barbosa Utrera

Final Degree Work
Bachelor's Degree in
Video Game Design and Development
Universitat Jaume I

May 12, 2021

Supervised by: Angel Pascual del Pobil Ferré

Acknowledgments

I would like to thank Víctor Manuel Jiménez Pelayo and José Ribelles Miguel for being very good professors and for inspiring me to continue and finish this journey that has been the degree.

Abstract

This document is the result of my Final Degree Project from the Bachelor's Degree in Video Game Design and Development. The game I made, Garden Battle, is an AI-focused one, where I used some popular AI techniques like Hierarchical State Machines and Behaviour Trees in order to make a battle simulator game. Unity is the engine that I chose, and, I used some other tools and programs for other areas such as art, project organization, etc.

Contents

CHAPTER 1

INTRODUCTION

Contents

This chapter reflects how the project started, the motivations, the objectives, and the initial state from which I started.

1.1 Work Motivation

After finishing the majority of the Bachelor's Degree, I wanted to make a game that used some of the most popular AI techniques on the market. For that reason, I thought on a game that its major focus was the AI. Garden Battle is a game where the players just have to select the number of characters. The main interest of the game is to see how the bugs kill each other. On the other hand, I wanted to make my own art, low poly assets with straightforward animations, by this way, I could prove to myself that I can make a complete game on my own. I also wanted to make SOLID code[1], which was expandable and readable, in order to improve my skills with C#.

1.2 Objectives

So, the objectives were clear:

- Implement AI Techniques in order to create interesting interactions between non-playable agents.

- Organize the code for creating a readable, maintainable and extensible project using SOLID principles.
- Design an interesting game focused on AI with almost no interaction of the player.
- Create simple low poly assets with animations, having some style. Although this objective was no present on my initial project proposal, it was also an important objective for me.

However one of the most important goals was to actually finish the game, it is very easy to quit projects just for lack of motivation, this time I wanted to finish a complete game.

1.3 Environment and Initial State

The project was created from scratch. I just had the general idea of the game, some bugs scattered across a map, fighting each other, and having their own strategies, to destroy their bases.

Regarding the technology, I had been using Unity for 3 years. My knowledge of the engine was good enough to make a project like this, I had used tools like the Navmesh[2] or Cinemachine[3] that I would use within the game. However, I had almost no experience with Blender and modeling in general, and even less with animating.

About the code, I had some experience programming different little games on Unity, I had a good knowledge about object-oriented programming, and, some basic notions about SOLID principles.

I also had experience with project organization applications like Trello or HacknPlan, and with version control softwares[4] like Github or BitBucket.

CHAPTER 2

Planning and resources evaluation

Contents

In this chapter, I describe all the pre-planning of the game, a general idea of the cost of the project.

2.1 Planning

In order to organize the project, I made a chart with all the tasks and their estimated time to complete them (see Figure 2.1) It was a bit generalist, but enough to get an idea of the time needed for each area. They were not ordered either because I could move from one to another without altering the rest.

sorted

2.1.1 Write the final memory

This task was the first one that I introduced, that was an obvious one, that would take a lot of time, and it was mandatory for this Final Degree Project.

2.1.2 Select the most adequate AI method for implementation

This was one of the most important task of the project. I had to analyze and choose those AI techniques that fit on my type of game in order to implement them.

Tasks	Time
Write final memory	50h
Select the most adequate AI method for implementation.	15h
Design the behaviour of the characters	15h
Model the characters	30h
Make characters animations	30h
Make Some particles for the animations	15h
Set the environment	15h
Implement the AI Technique selected	50h
Implement individual characters behaviours	70h
Implement UI	10h

Figure 2.1: Planning with all the tasks

2.1.3 Design the behaviour of the characters

One focused on design, I needed to think about the design of all the characters, their behaviours, strengths, and weaknesses. A design that was more or less interesting and distinct between the different bugs.

2.1.4 Model the characters

Another important task, I had to learn the basics of Blender and low poly creation. Furthermore, I had to try to make characters with a bit of style.

2.1.5 Make characters animations

I also had to learn how to make animations on Blender and try to make them interesting, so this also was a difficult task for me.

2.1.6 Make some particles for the animations

For this task, I also had to learn, this time the Unity's Particle System[5] which is integrated into the engine. although the VFX(visual effects) are an important part of the game feeling, this was a task I could probably reject if I run out of time as it is not as crucial as the others.

2.1.7 Set the environment

I had to think about what type of environment I wanted for my game, and, if it would be procedural or a unique pre-made one. This was an important task, as the terrain is where all the action happens. Furthermore, I had to make a camera that was easy to control for showing the game.

2.1.8 Implement the AI technique selected

Once the selection was made I had to implement it. This was one of the largest tasks as build an AI system from scratch takes a lot of hours for planning it and make it, thinking about its reusability in order to be able to easily adapt it to different NPCs.

2.1.9 Implement individual characters behaviours

With the AI system implemented, I had to use it in order to make all the behaviours of the different characters. This was the other big task that would take a lot of time and the most important one, as the result of this task is what players would see in the game.

2.1.10 Implement UI

The last task. I thought about making a main menu, where you could select the number of characters, and, an in-game UI(User interface) for changing the game speed of the game and show the characters and base health. Part of this task could also be rejected if I run out of time as it was not a really important one either.

2.2 Resource Evaluation

For this project, I wanted all the resources used to be free. So all the programs I used in this project are free of charge. Unity, Blender, Krita, Trello, Github, Diagrams, and Overleaf are free tools. As for the hardware, I planned to use my PC, which was capable of running all those programs properly, so it was good enough for this project.

CHAPTER 3

System Analysis and Design

Contents

This chapter presents the requirements analysis, design and architecture previous to the development in order to facilitate the work.

3.1 Requirement Analysis

For the requirement analysis, I divided them in two parts, functional and non-functional.

3.1.1 Functional Requirements

A functional requirement defines a function of the system that is going to be developed. This function is described as a set of inputs, its behavior, and its outputs [6].

In general, the functional requirements refer to what the system should do. I detailed the more important ones on the next charts. These are not all the specific functional requirements but the more general that describe the overall game.

3.1.2 Non-functional Requirements

Non-functional requirements impose conditions on the design or implementation (e.g., to meet performance, safety, or reliability constraints) [7].

Input:	Number of characters selected
Output:	Characters added to the game
Players can select how many characters want from each type of bug. The system must save that number and add those characters to the game	

Table 3.1: Functional requirement «MENU1. Entering Amount»

Input:	Play button pressed
Output:	Generation of a game with all the characters selected before
Once the play button is pressed, the game must start and generate a game with all the characters selected on the main menu	

Table 3.2: Functional requirement «MENU2. Start Game»

Input:	Environment
Output:	A behaviour to follow
NPCs must choose the correct behaviour after analyzing their environment	

Table 3.3: Functional requirement «NPC1. Behaviour decision»

Input:	Keyboard keys or mouse buttons and pointer
Output:	Camera movement accordingly to the input
Players can control the camera with keyboard or mouse, move it, rotate it, and, zoom in and out	

Table 3.4: Functional requirement «CAMERA1. Camera Movement»

Input:	Keyboard keys or game buttons
Output:	Game speed
Players can change the game speed of the game in order to end the game faster	

Table 3.5: Functional requirement «GSPEED1. Game Speed»

In general, non-functional requirements define how a system is supposed to be. For this project, I planned these ones:

- The code must follow the SOLID principles in order to create an expandable, reusable and readable code that could be move to another project without major problems.
- The game must be able to work on a stable frame rate.

3.2 Characters Design

In this project, all the design is centered on the characters , so I thought about 5 type of characters and I detailed the behaviour of each one.

3.2.1 General behaviour

- Every insect has its base, except the worm.
- They regenerate health at their base.
- Bases generate a new unit each certain time.
- The insect whose base is the only one left standing wins.

3.2.2 Ant behaviour

- If there are ants in the game, there are sources of resources scattered all over the map that the ant can take to become stronger.
- Resources are regenerated after a period of time.
- At the beginning they are dedicated to gathering resources until they are strong.
- When they are strong enough, they will attack the rival bases.
- If they are weak and enter in combat, they will flee back to their base.
- If they are low on health, they will return to their base.
- The ants will take turns eating and protecting the base.
- If their base is attacked, they return to defend it.

3.2.3 Wasp behaviour

- They patrol since the beginning.
- They attack whatever they see.
- If they are low on health, they will return to their base.
- If their base is attacked, they return to defend it.

3.2.4 Spiders behaviour

- They set traps in the field and return to base.
- They leave the base to check the traps and kill instantly the insects that have fallen into them.
- Their traps immobilizes enemies for a certain period of time.
- When there are few enemies, they will attack the bases.
- If their base is attacked, they return to defend it.
- If they are low on health, they will return to their base.

3.2.5 Mosquito behaviour

- They attack from a distance.
- They stay defending next to the base until they have a good number of troops, then they start to patrol like the wasps.
- If their base is attacked, they return to defend it.
- If they are low on health, they will return to their base.

3.2.6 Worm behaviour

- They have no base.
- They can not win the game.
- They remain underground.
- They go out to attack, to bugs if there are any or to bases if there are no bugs left.
- Their attack is in area

3.2.7 Characters stats

Another important part of the characters are the stats, I wanted that each character had a different attack damage so I made a table for this.(see Figure 3.1)

Insect	Attack Damage
Ant	From low to very high(Depending on resources)
Wasp	High
Spider	Low
Mosquito	Medium
Worm	Very high

Figure 3.1: Table with the characters stats

3.3 Concept Art

In order to model each character I needed a reference for each one, so I searched some images or 3D models that could be useful to me for making the assets.(see Figures 3.2, 3.3, 3.4, 3.5, 3.6) I also searched images for the terrain.(see Figures 3.7, 3.8)

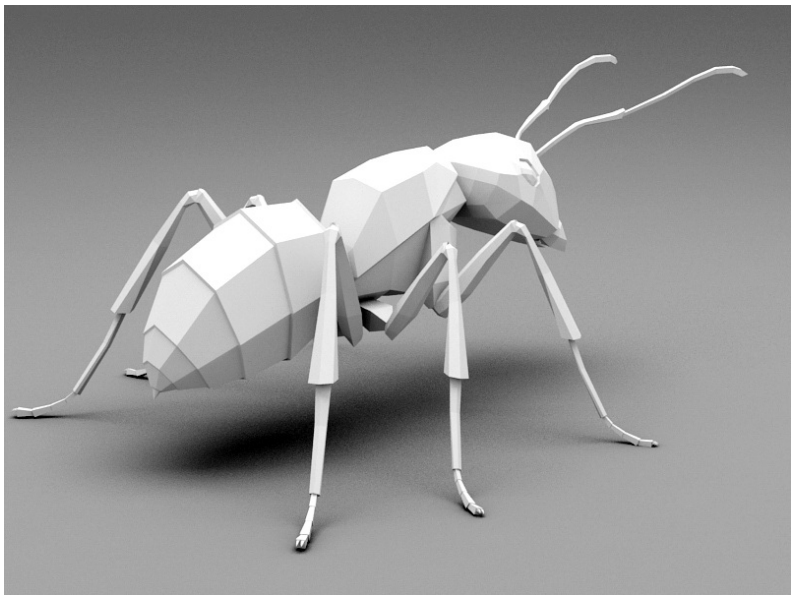

Figure 3.2: Ant concept

Figure 3.3: Wasp concept

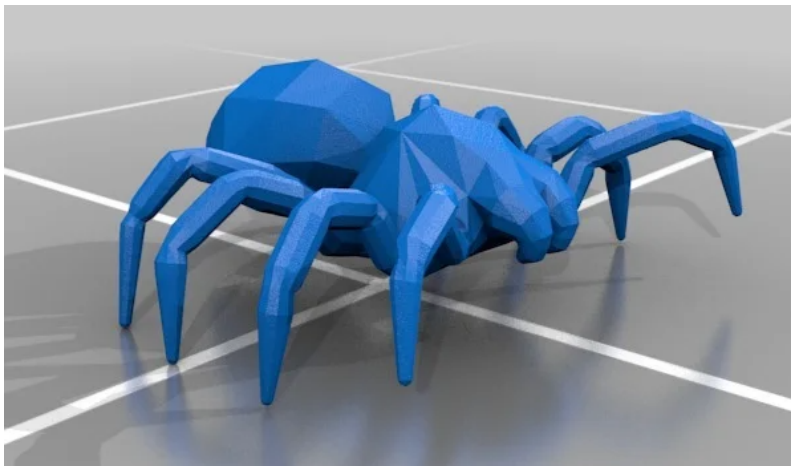

Figure 3.4: Spider concept

Figure 3.5: Mosquito concept

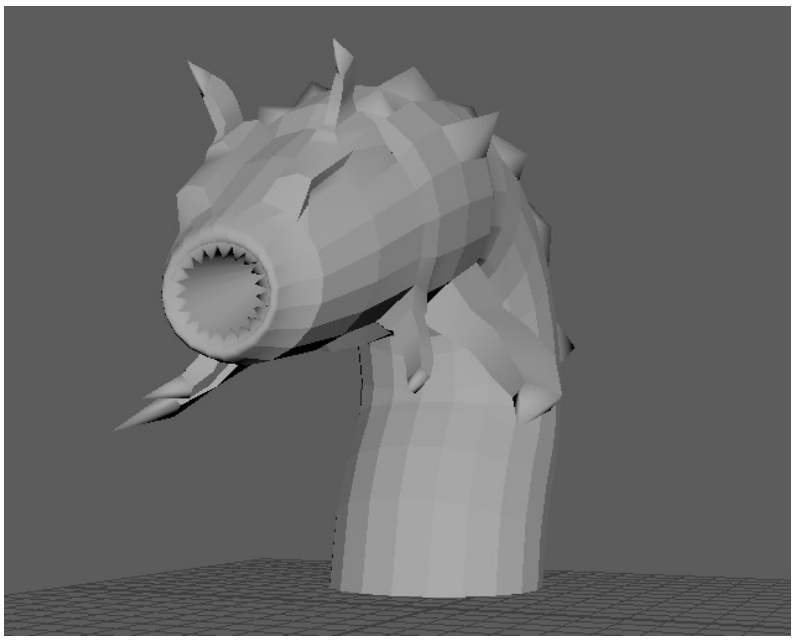

Figure 3.6: Worm concept

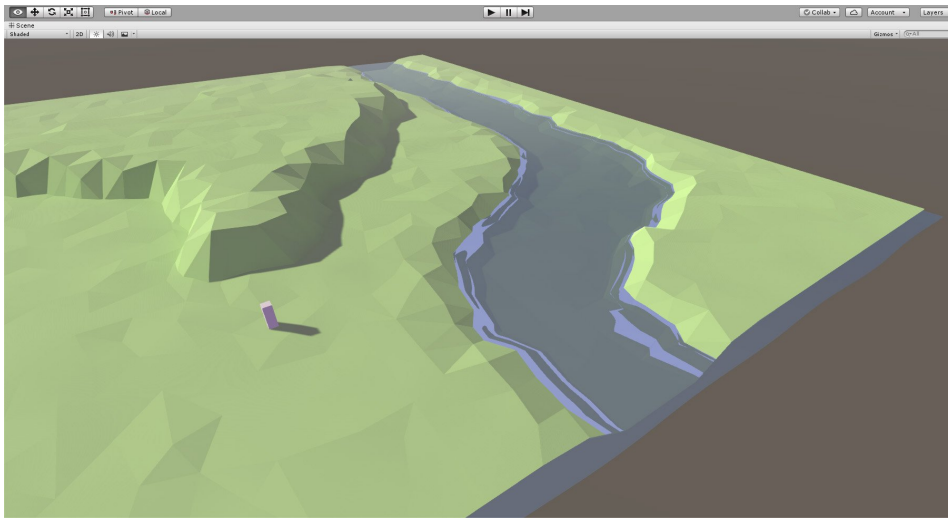

Figure 3.7: Terrain concept 1

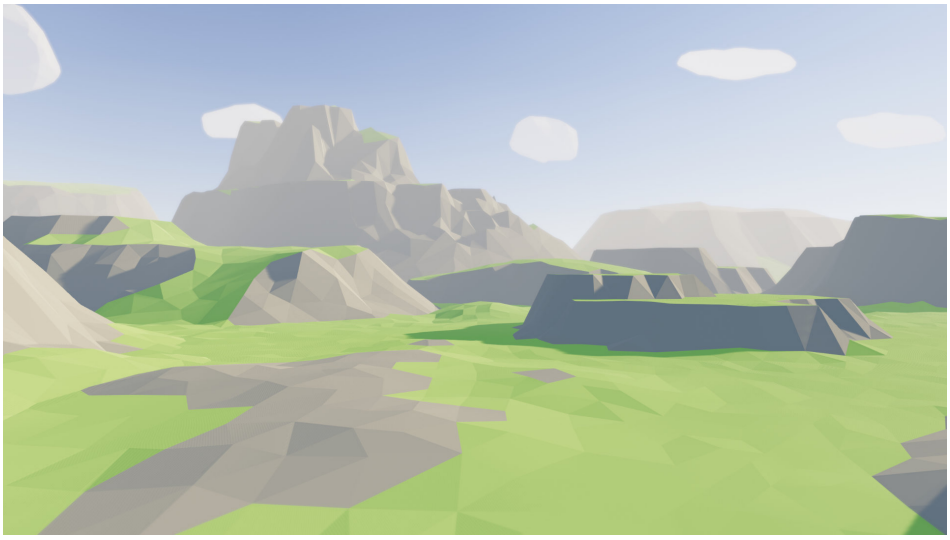

Figure 3.8: Terrain concept 2

Chapter 4

Work Development and Results

Contents

In this chapter, I describe all the work done during the development of the game. All the steps I took and the difficulties I encountered along the way. I also point out the discrepancies between the initial analysis and planning and the final result.

4.1 Work Development

4.1.1 Project Organization

First, I decided which applications I should use to organize the project. On one hand, I chose Trello to write the tasks and divide them into groups, to do, done, bugs, and, extra things to do if I had extra time before or after the delivery of this report(see Figure 4.1). On the other hand, I decided to use Github to manage the version control of the project since it is the program I had more experience with.

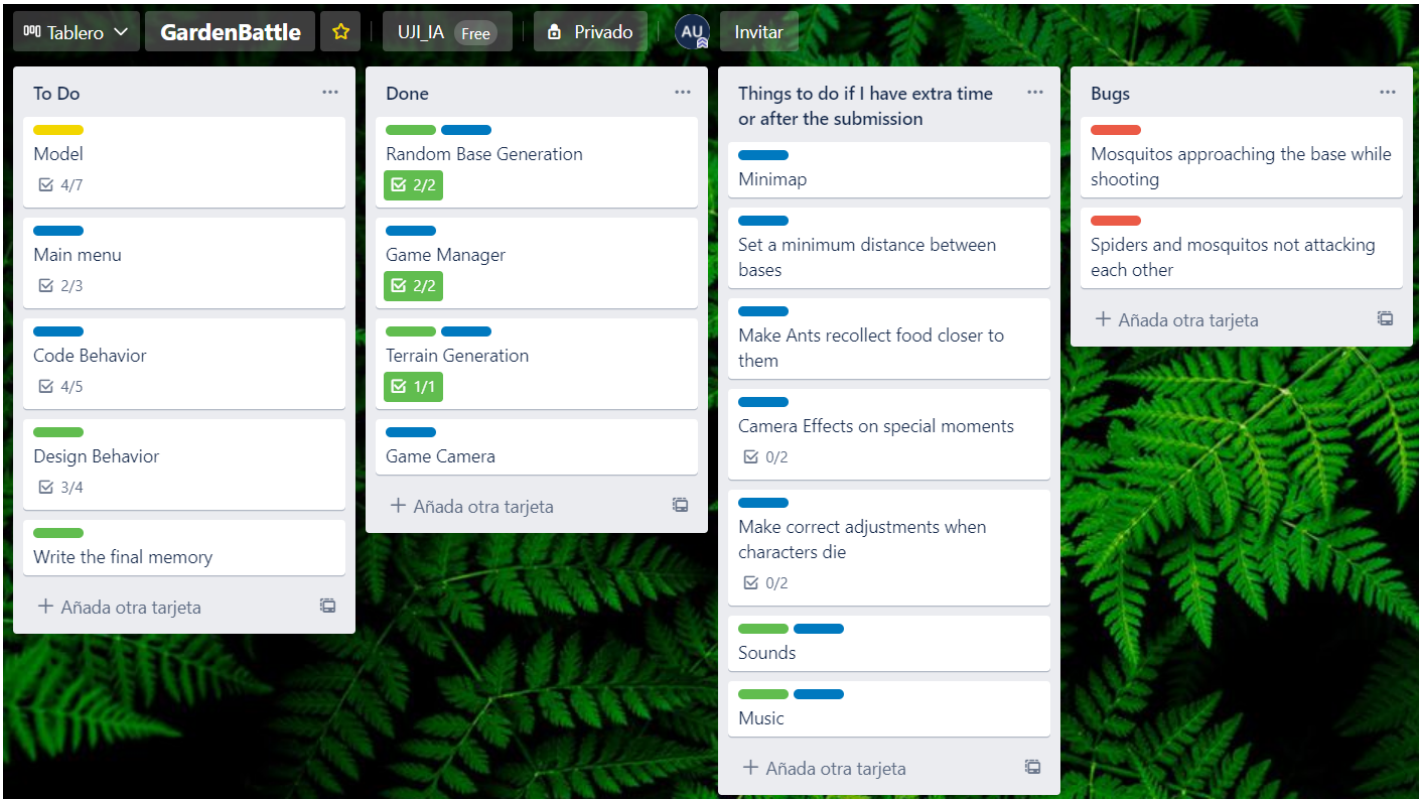

Figure 4.1: The status of the trello at the moment of writing this memory

4.1.2 AI system selection

The first step was to decide what type of AI technique I wanted to implement in my game. After a long time analyzing different methods, I finally arrived at the conclusion that I wanted to use a system that mixed Hierarchical State Machines[8] with Behaviour Trees[9]. This way, I could have the best of both worlds, the code partitioning of the State Machines and the ease of making transitions and actions with Behaviour Trees. In addition, these two systems could easily be made expandable and reusable, so I could improve them in the future or use them in other projects.

4.1.3 AI system implementation

Hierarchical State Machines

For the implementation of the Hierarchical State Machine, I divided the code into some classes. HierarchicalStateMachine, SubMachineState, State, Transition and Condition. The complete state machine of a character is composed of some, or all, of them. The HierarchicalStateMachine is always present, is the upper layer of the machine, inside has the rest of layers, and it is responsible for executing the current layer. SubMachineS-

tate is the second layer, it is a specific type of State that can have other sub machine states and states inside it, and can execute code and behaviours too. The last layer would be the State, which has no other states within it, only code to execute and do behaviours.

For traveling between states, there are transitions and conditions. When the character is in a state or submachine state, some transitions are checked through conditions in order to see if it must change its state into another one. The state machines must check the transitions of its layer and the upper ones,(see Figures 4.2, 4.3, 4.4). Furthermore, all states have functions that are executed when entering, exiting or updating them.

I also did a StateSelector class for deciding what state is chosen first when entering a submachine state.

Finally, I created the code in a way that all the states, machines, transitions, etc. could be edited on the Unity Editor, within the inspector.

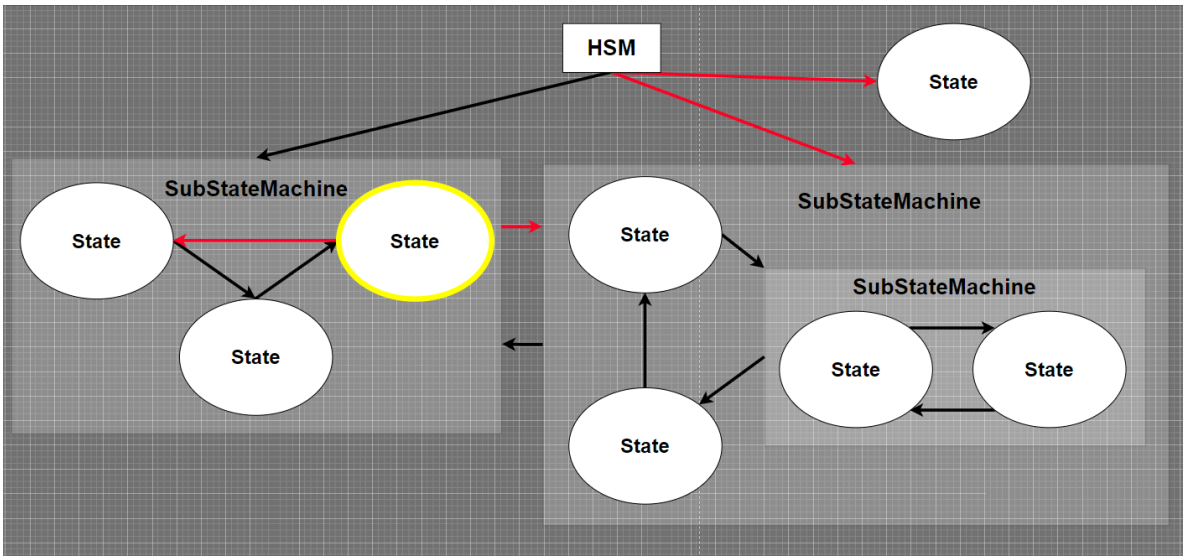

Figure 4.2: Example 1 of an state machine. The state with yellow border is the current one, and the red arrows are the transitions that must be checked

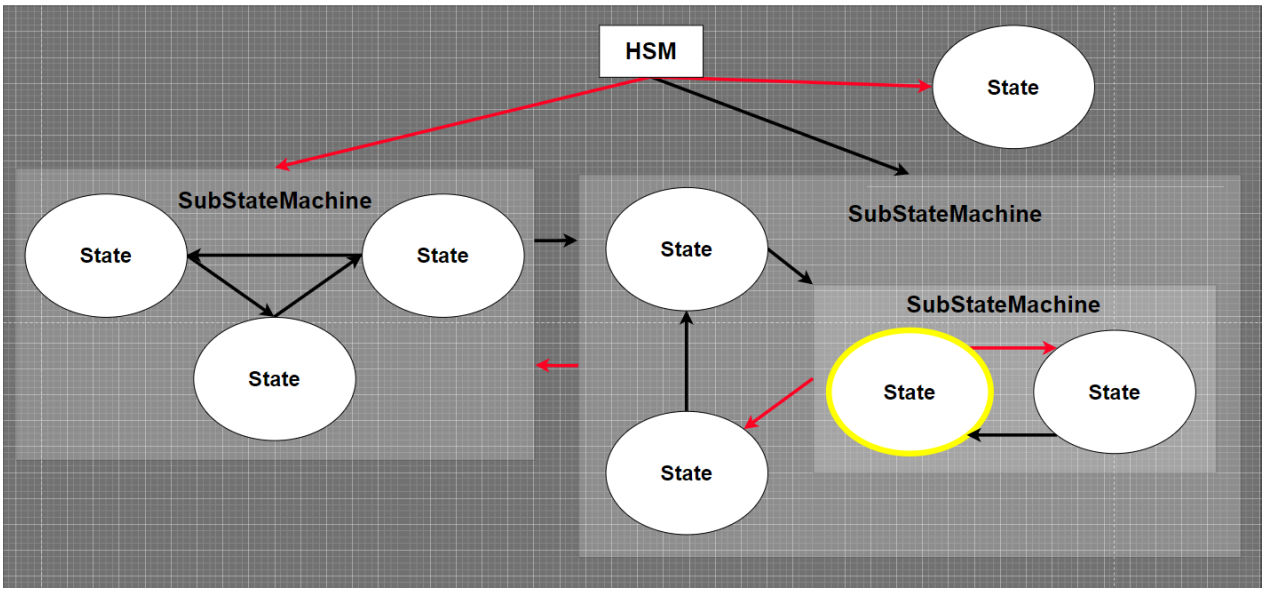

Figure 4.3: Example 2 of an state machine. The state with yellow border is the current one, and the red arrows are the transitions that must be checked

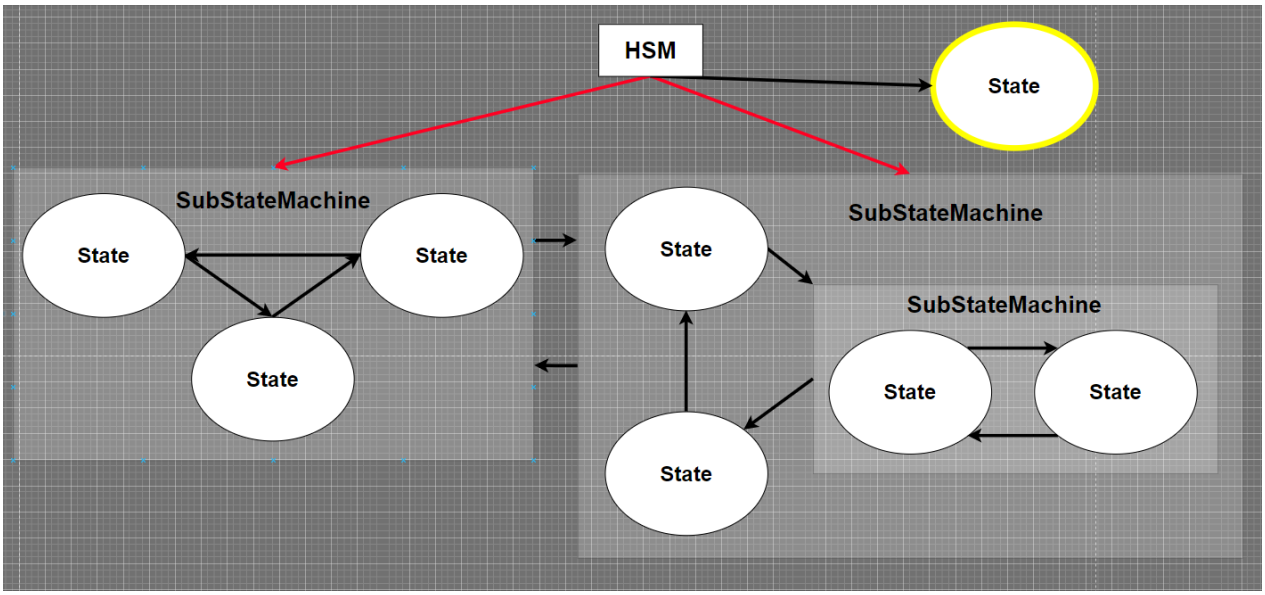

Figure 4.4: Example 3 of an state machine. The state with yellow border is the current one, and the red arrows are the transitions that must be checked

Behaviour Trees

For the behaviour trees, I decided not to implement all the parts that are usually made. I only implemented sequencers, selectors, random selectors, and, the blackboard. So I left out decorators, random sequencers, and, parallel tasks. This is a bit simple behaviour tree but enough for the project.(see Figure 4.5)

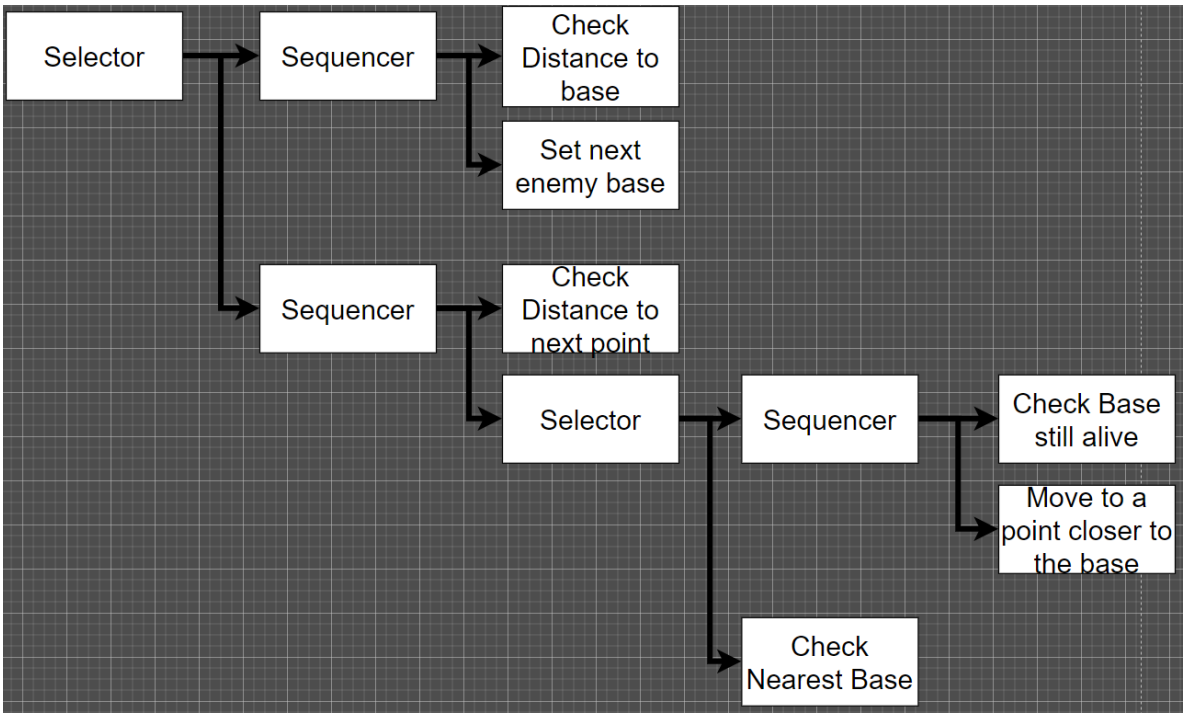

Figure 4.5: An example of a behaviour tree

These behaviour trees were integrated into the state machines, so the transitions and the actions of the states could be made through behaviour trees.

Regarding the blackboard, there are two types of it, the blackboard that is specific to each character and saves its data, and, the global blackboard, which stores the data shared for all the characters of the same type. This last blackboard is also used for the GameGlobalBlackboard, which saves data about the whole game, and can be accessed by any character.

For the behaviour trees, I also wrote the code for being completely edited from the Inspector of the Unity Editor. So it is possible to create a full hierarchical state machine, with behaviour trees in it, from outside the code.(see Figure 4.6)

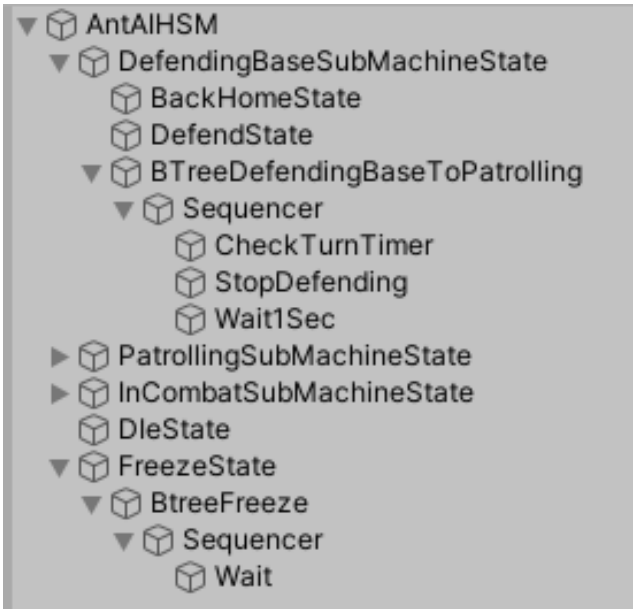

Figure 4.6: Example of a complete hierarchical state machine, with beaviour trees in it, made on the Unity Editor

4.2 Characters Modeling

As I said before, I had not too much experience with modeling and Blender, so I had to watch some tutorials to learn about the program and low poly creation. The videos from Imphenzia helped me a lot[10].

Firstly, I wanted to create only a character for testing the AI system that I implemented. The ant was the first one that I made. The auto-mirror option for modeling both sides of the body at once helped me a lot in creating all characters. The methodology for creating every character was the same. I created the model, then I painted the polygons from a simple color palette, and finally, I made the bones and animate them.

I created the ant, and, later, I gradually created the rest. In the end, I succeeded in creating characters with a certain style that all seem to belong to the same game.(see Figures 4.7, 4.8, 4.9, 4.10)

Figure 4.7: Final model of the ant

Figure 4.8: Final model of the wasp

Figure 4.9: Final model of the spider

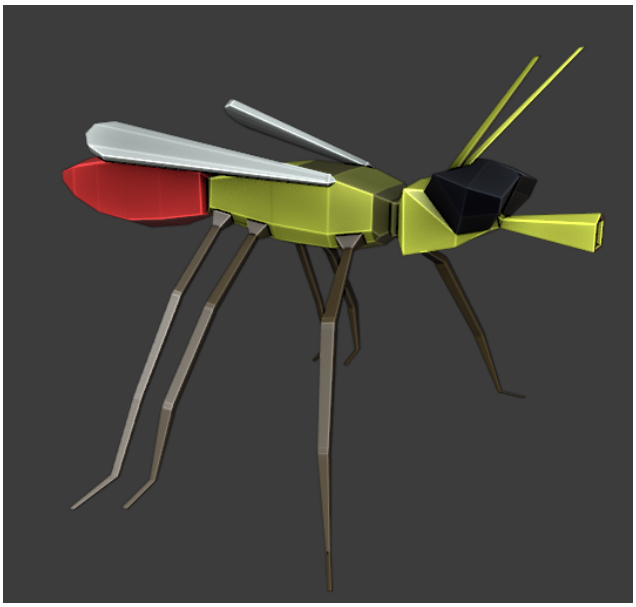

Figure 4.10: Final model of the mosquito

4.3 Terrain Generation

The terrain generation is the only part of the game that is not original. I decided that I wanted to do a procedural terrain for each game, and, it was some time-consuming creating one procedural system from scratch. So, I followed the videos from Sebastian Lague to create a procedural mesh from a noise texture[11]. I did not implement all the things that his tutorial has, but I created the necessary ones for my project. The terrain generates procedurally and is possible to change some parameters like color, height, size, and others related to the noise map, lacunarity, persistence, octaves, etc.

I have managed to integrate this system into my game, and I have been able to generate a simple but beautiful terrain.(see figure 4.11). Furthermore, as the characters have to navigate through this environment, I had to download additional NavMesh packages from unity[12] in order to generate a real-time navigation mesh from that generated terrain.

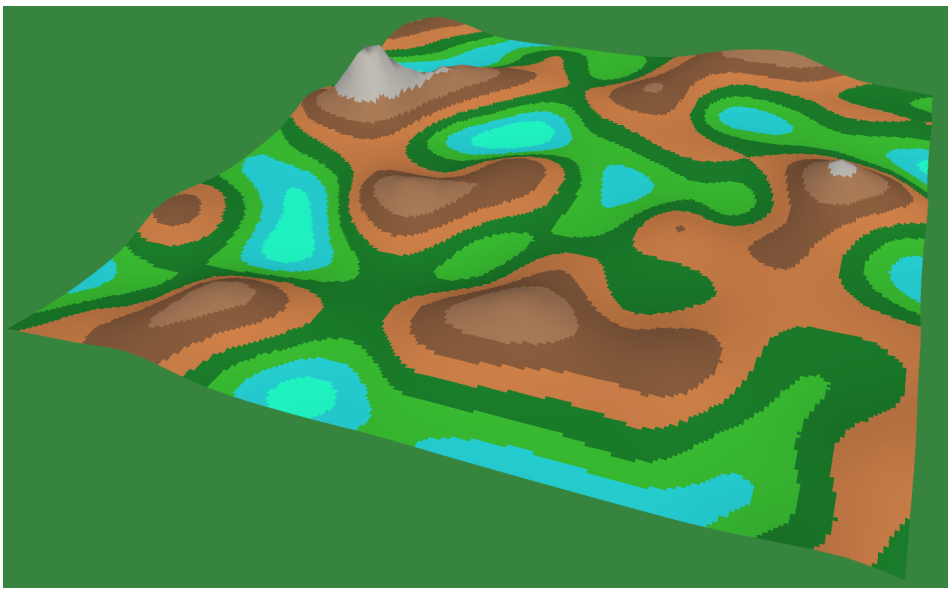

Figure 4.11: An example of a procedurally generated terrain

4.4 Base Generation and character spawning

As I said before, each character type has a base from which it is spawned. So, after creating the terrain, I had to randomly spawn the bases around the map, and, then spawn the characters around that base. The number of characters is chosen by the player, so I had to spawn the characters in a way that was able to handle any amount of them,

regardless of where the base is placed.

I managed to do a spawning system that is capable to react to any number of characters and can even handle the amount when the base is placed on the boundary of the terrain. For this purpose, the bugs are generated in rings, that grow as they move away from the base. If some bugs can not be spawned because there is no terrain to spawn, then those characters are saved and the generator is able to place them somewhere they fit.

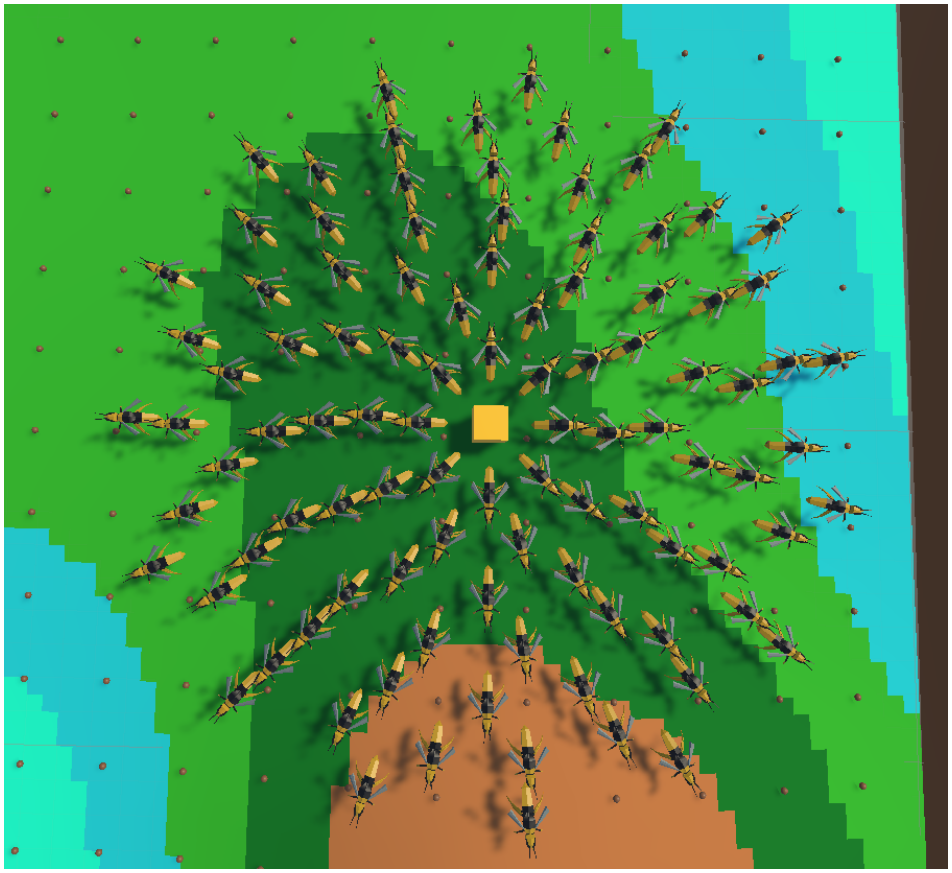

Figure 4.12: An example of a normal spawn generation

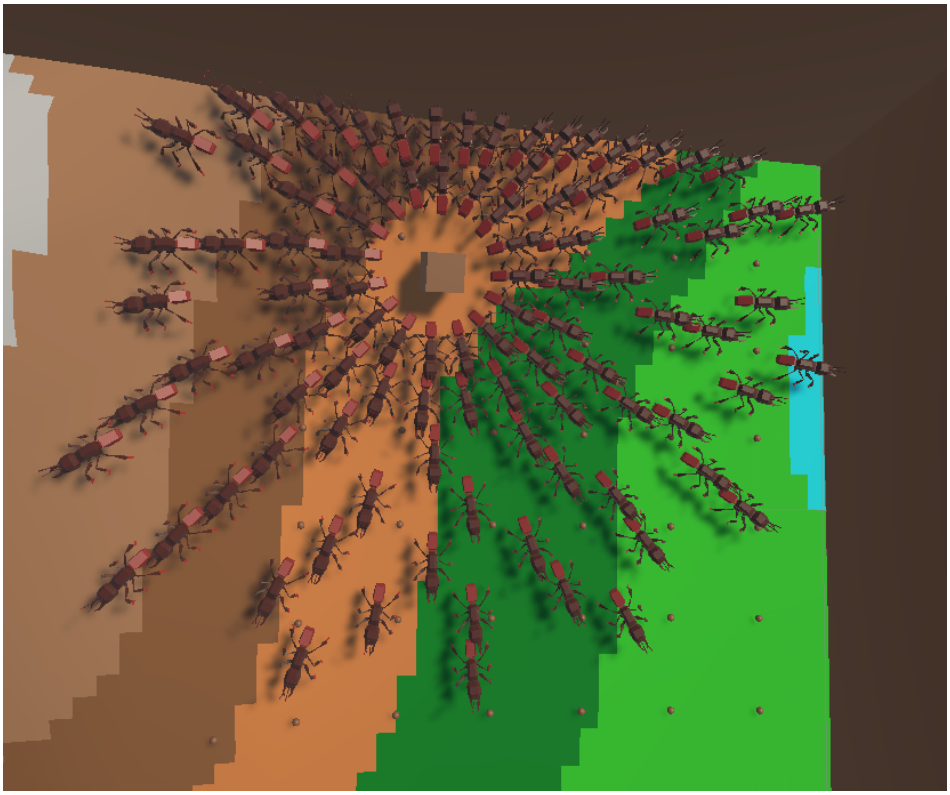

Figure 4.13: An example of a spawn generation when the base is at the edge of the map

4.5 Food generation

As I stated in the analysis, the ants are dedicated to eating food in the early moments of the game. So, if players choose to have ants within the game, the food has to be spawned around the map. Moreover, I needed the food to be ordered from distance to the base. For this reason, I created a FoodGenerator class that spawns the food along the map, and a FoodManagement class that stores all the food and sorts it by distance to the base to give the ants the closest ones.

4.6 bugs behaviour

4.6.1 Ant behaviour

Finally, it was time to use the AI system that I created for implementing the behaviour of the Ant, the first character that I modeled. Before starting to code, I did some diagrams, within the diagrams webpage[13], to facilitate the programming of this bug. On these diagrams, I described the state machine, the transitions, the actions, and, the elements that would be saved on the blackboard. It was quite time-consuming to do all

the actions from the scratch and trying to make the system work, but after all I managed to do it, and the ant was working more or less as expected. I just needed another type of bug to confront them.

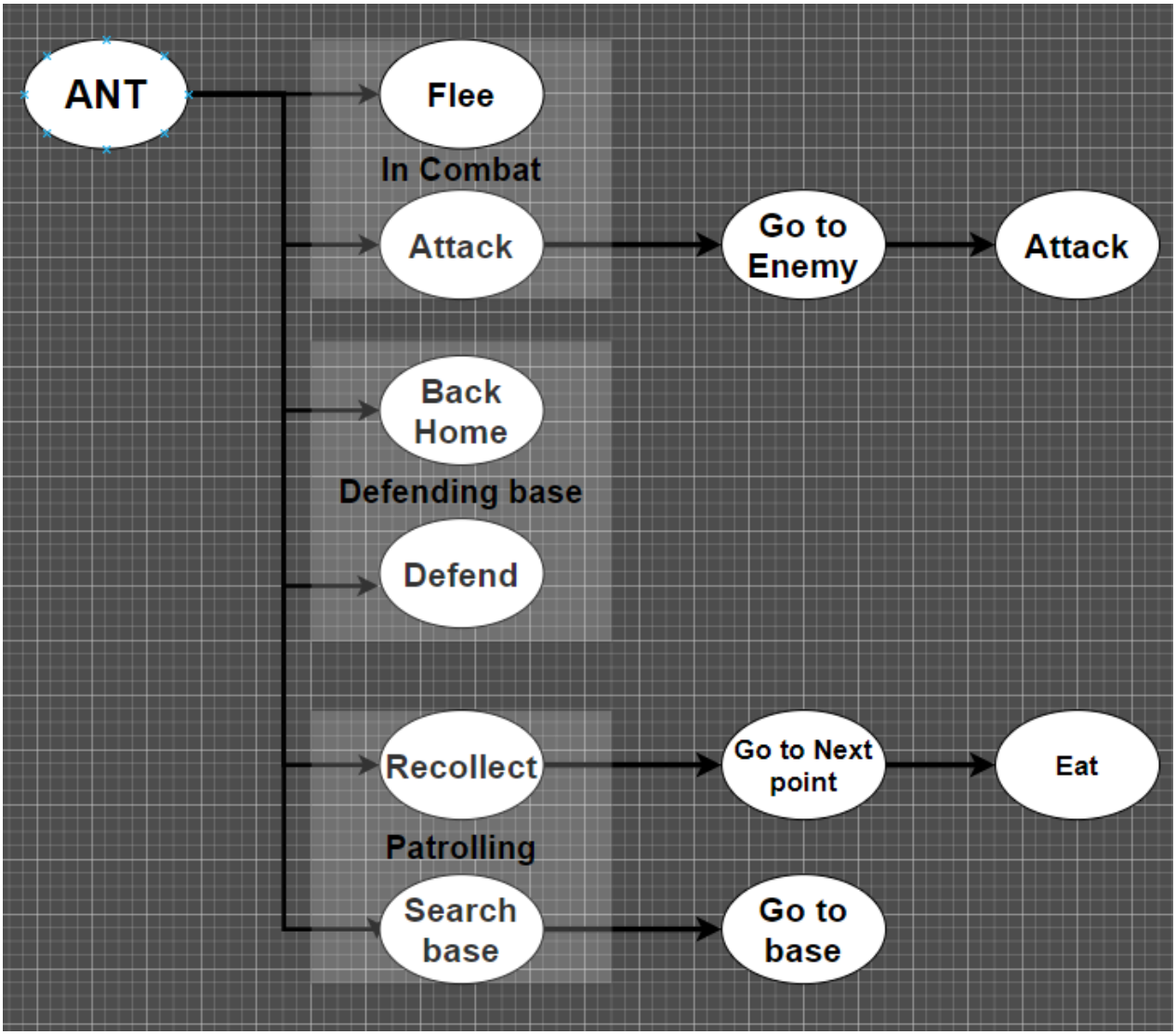

Figure 4.14: The first sketch of the state machine of the Ant

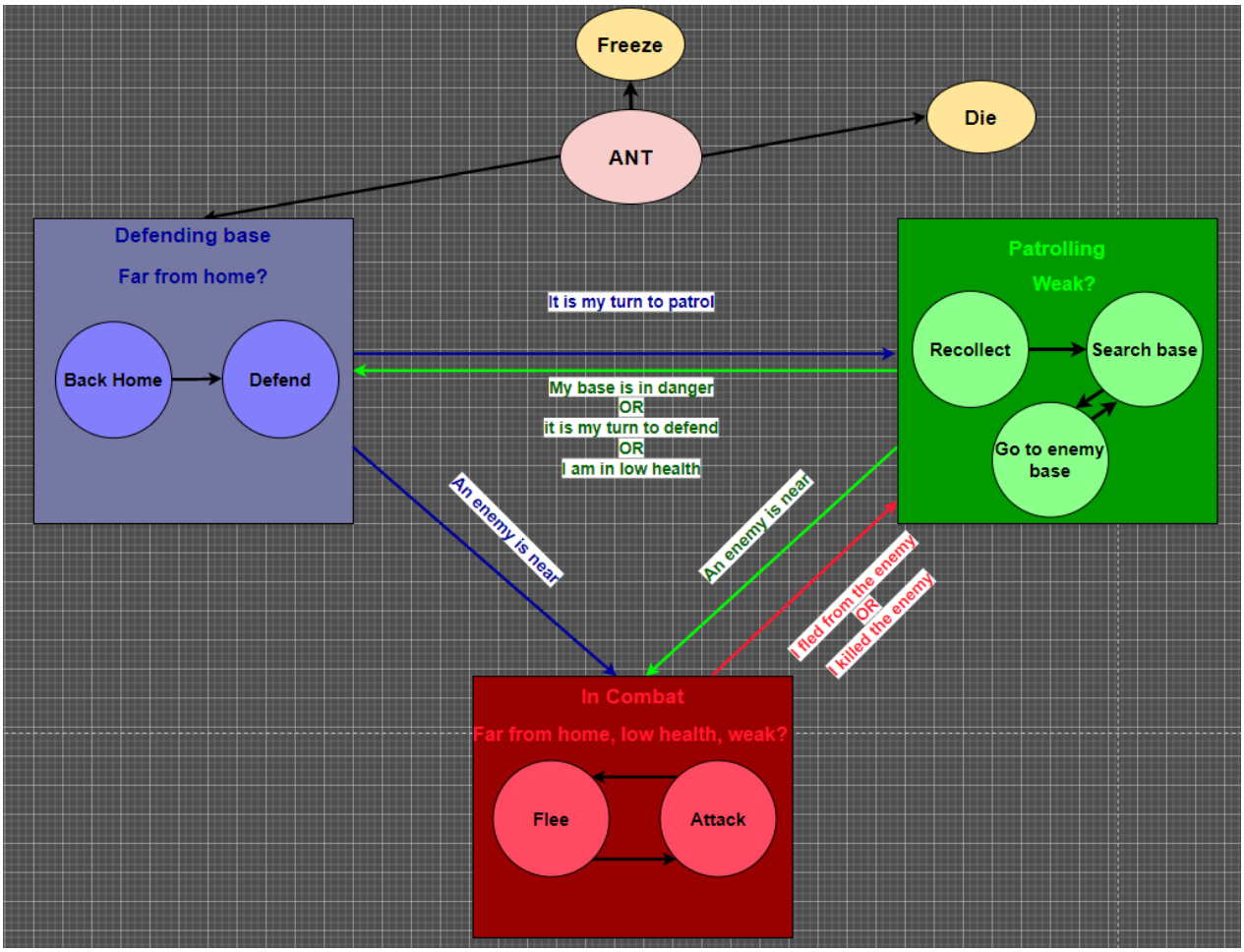

Figure 4.15: The final state machine of the ant

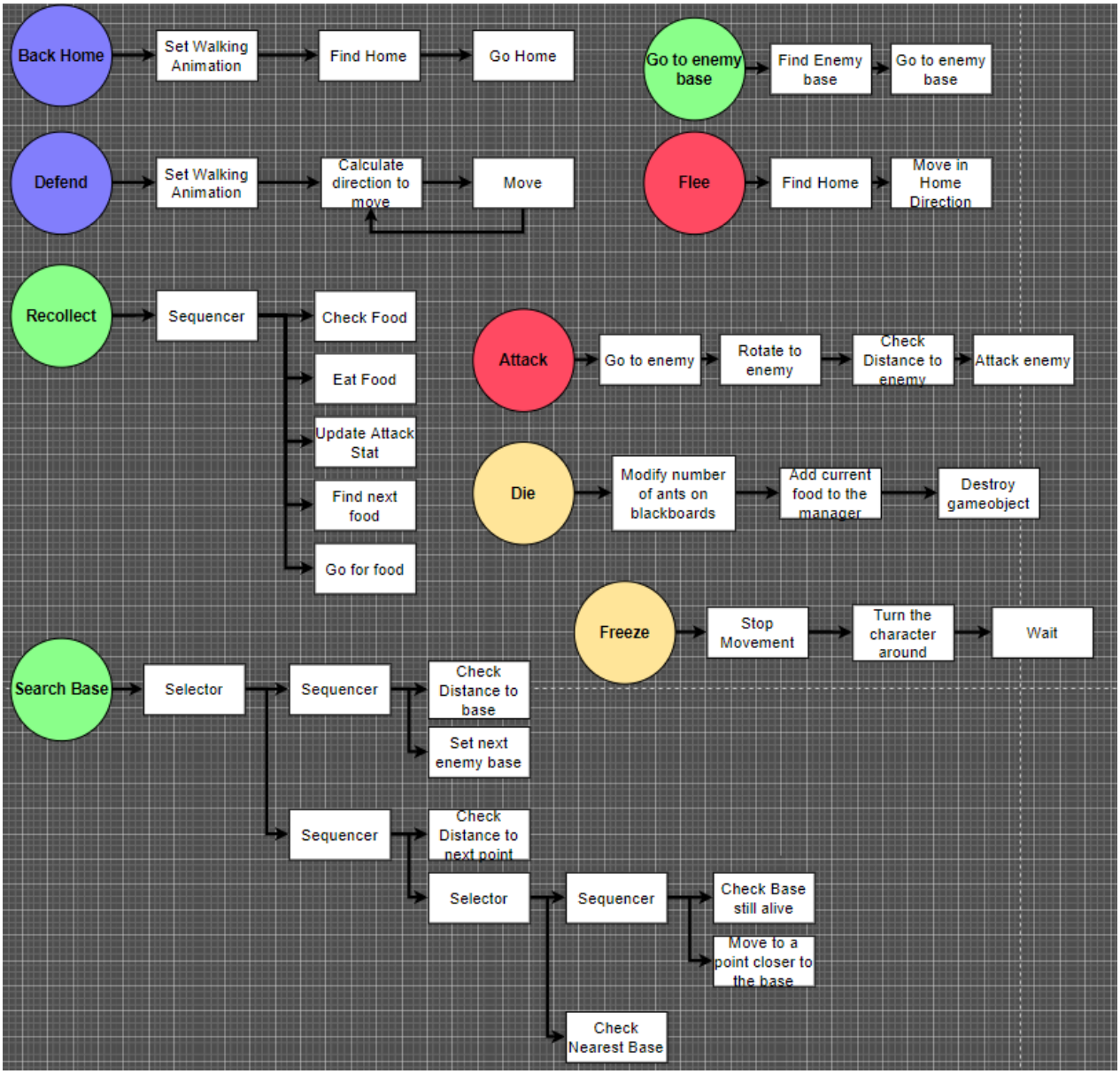

Figure 4.16: The explanation of the actions of the Ant

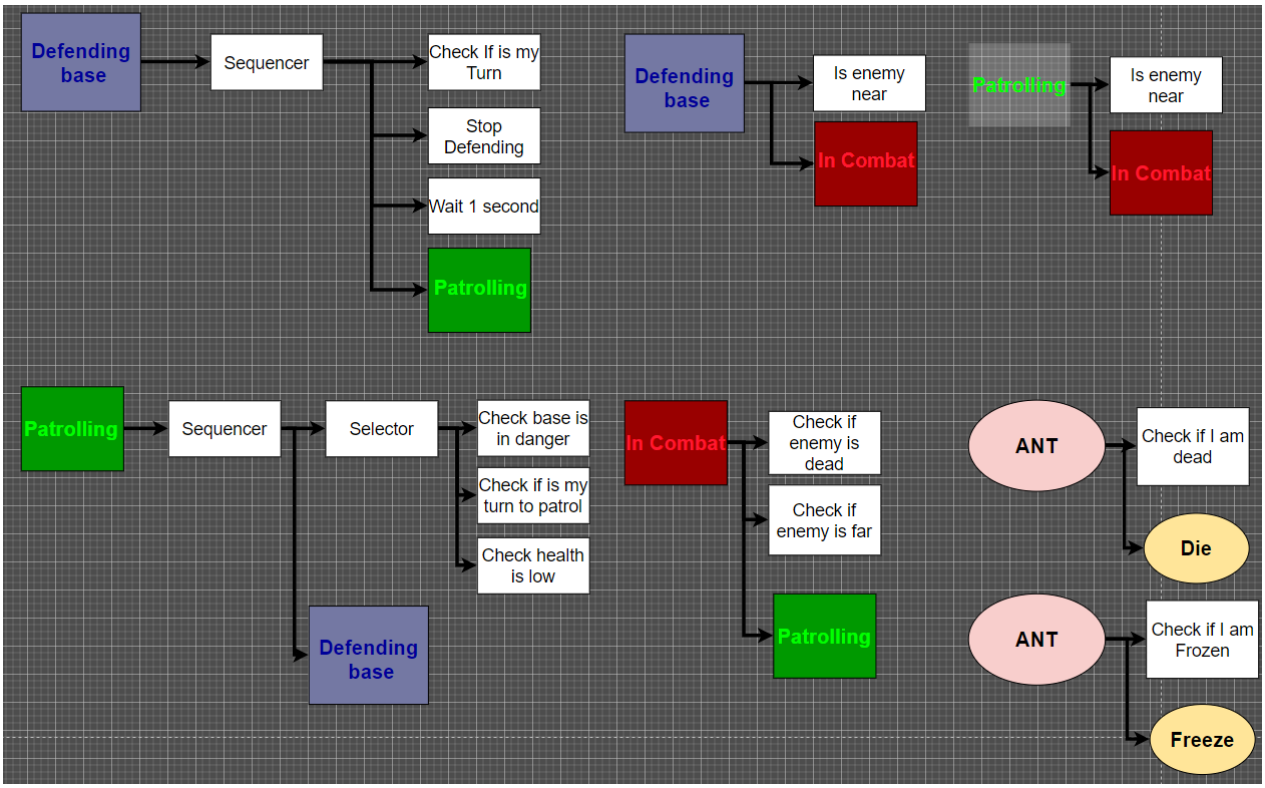

Figure 4.17: The explanation of the high level transitions of the Ant

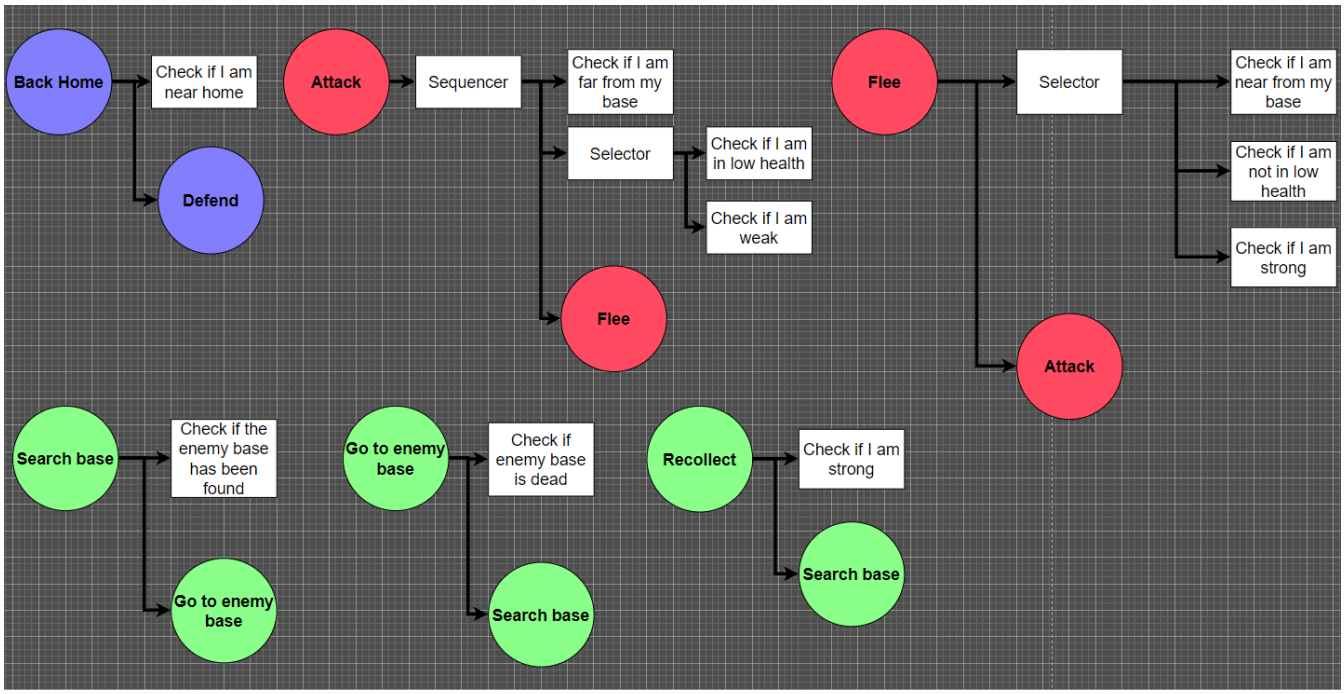

Figure 4.18: The explanation of the low level transitions of the Ant

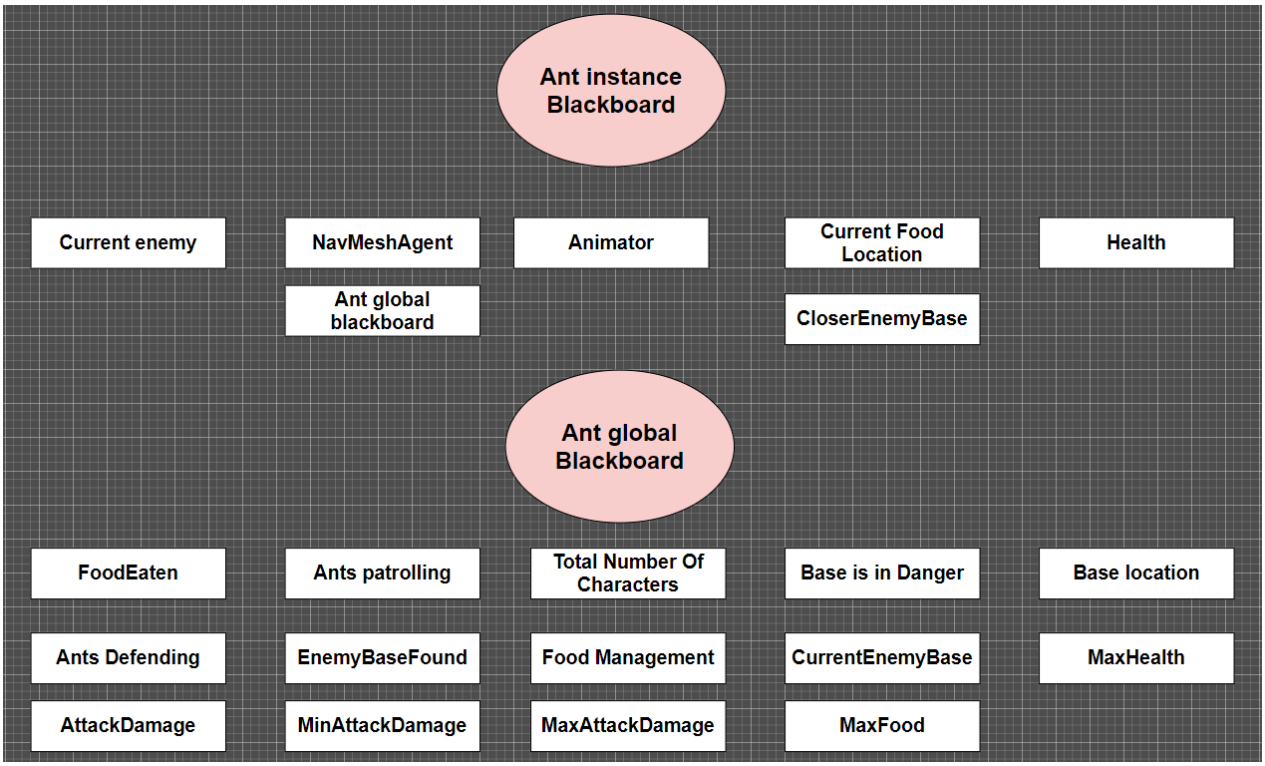

Figure 4.19: The elements inside the blackboard of the Ant

4.6.2 Wasp behaviour

This was the easiest character to do since its behaviour is a simplified version of the ant. So I just had to do minor tweaks to the ant's code to make this work. Furthermore, the diagrams of this and the rest were also easier to make and I did not even have to make them all as I did with the ant(see Appendix A).

4.6.3 Mosquito behaviour

This one was a bit more difficult to adapt as the mosquito attacks from distance. So, I had to alter a bit the code for attacking that I created for the ants in order to make them throw projectiles. In addition, since creating projectiles is something that wastes memory and can create memory allocations, I had to create an Object Pooler[14]. This way, I can spawn multiple projectiles by reusing them, activating them when necessary and deactivating them when they hit the enemy, instead of instantiating and destroying them.

4.6.4 Spider behaviour

The spider was the second most difficult. Spiders spend the early game setting traps for the enemy, and the way that they go, set a trap, return to base, wait, and, then go for another trap, made that I had to make little adjustments on the actions that I had for creating new ones that fit this behaviour.

For setting the traps, I did an approach similar to the food generation. First, I generate all the points where the traps can be placed, then I store them in a class as I did with the food to sort them by distance to the base and give these positions to the spiders. Furthermore, I had to model a spider web to place over those traps.(see Figure 4.20)

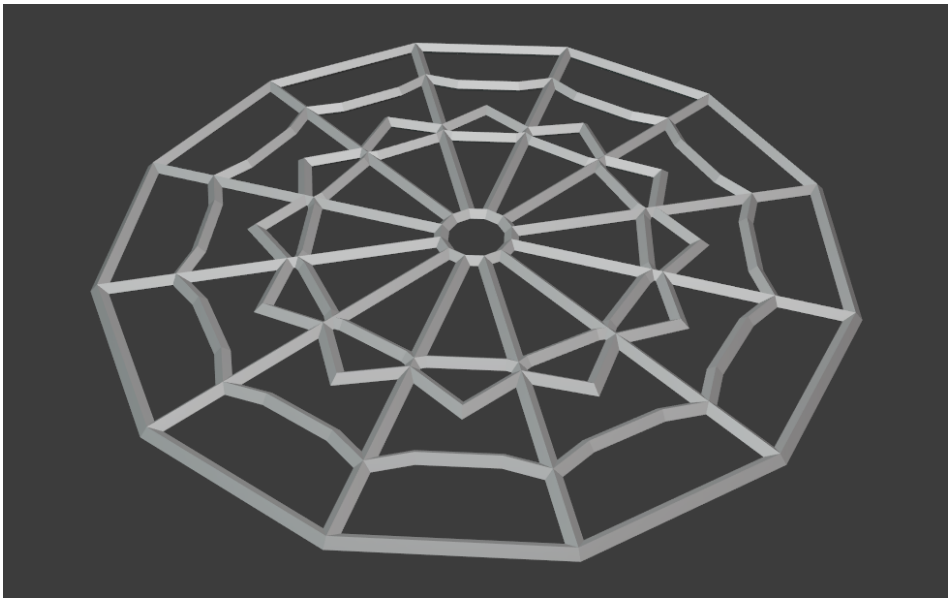

Figure 4.20: The spider web model

4.7 Main menu

With the behaviours of all the characters done, I decided to continue with the main menu. I set up a scene with all the bugs in the center. I positioned them around a circle and created two cameras. When players enter the game, a screen with a top view of the bugs is displayed, and they have the option to play a game.(see Figure 4.21) When they press the button to play, the camera makes a transition into the other one, I made that transition with the Cinemachine package. After the transition, the second camera is positioned in front of the first character, and then, through Unity's input field, players can type the number of characters they want from that bug.(see Figure 4.22) They can also select the other bugs by pressing the arrows of the screen or the A and D keys of the keyboard. I limited the maximum number of characters to 100 per type in order to make sure that the game works at a stable frame rate. Finally, they can press the start game option to create a game or the back option to return to the first camera.

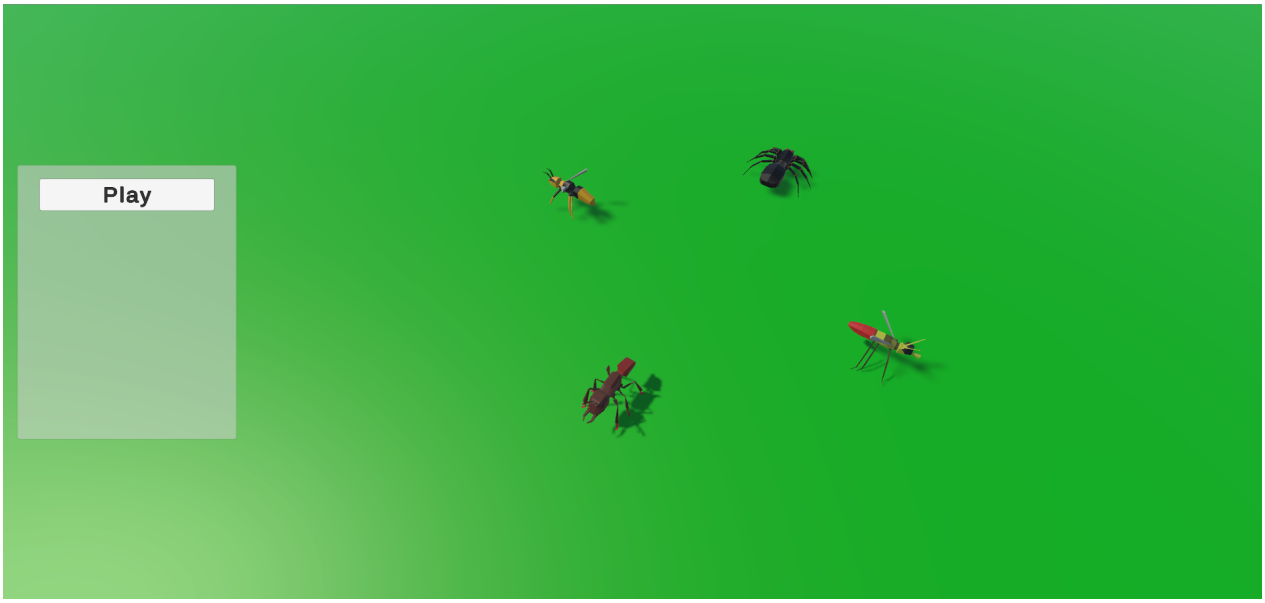

Figure 4.21: The main menu when entering the game

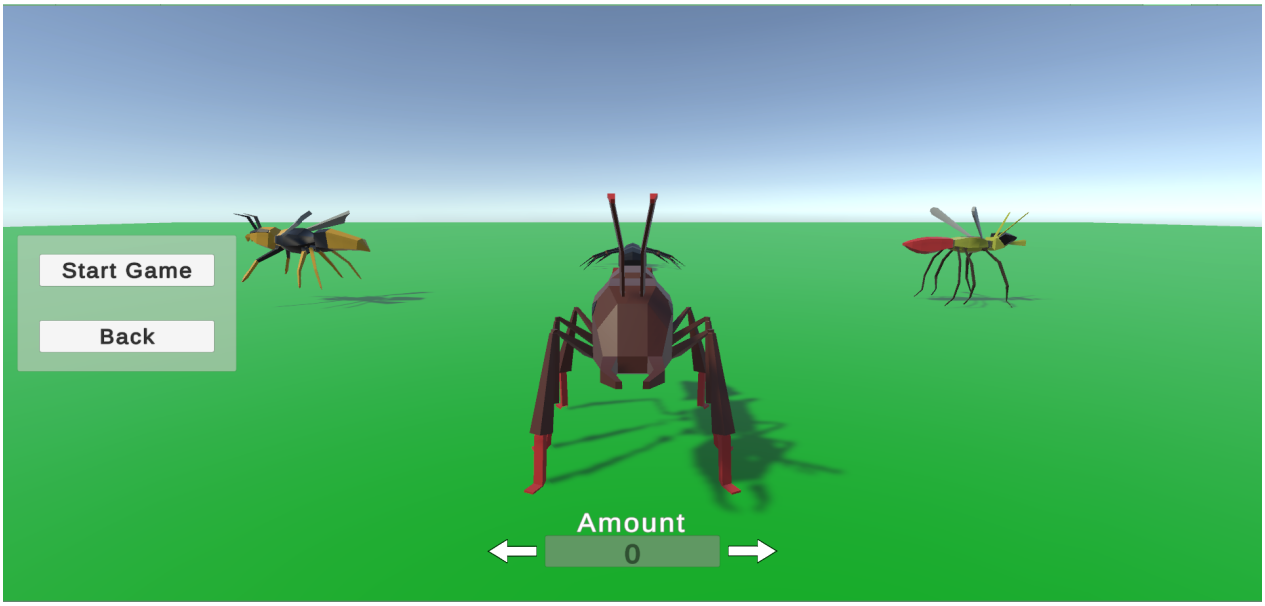

Figure 4.22: The main menu when selecting characters

4.8 Camera

The camera was one of the last things that I implemented. I had to think about a camera that fit in my game. So, I decided to make an RTS-like camera. Players can zoom in/out, move, and, rotate the camera, from a top view. In addition, I made The controls for being manageable with keyboard and mouse. You can zoom in/out with LShift/LCtrl or the mouse wheel, move the camera with W, A, S, and, D or pointing with the mouse at the borders of the screen, and, rotate with Q and R or dragging the mouse while the right click is pressed.

4.9 Game speed

Finally, the last thing that I implemented in the game was the game speed selector. For that, I created a little UI on Krita, with 4 states, 1X, 2X, 4X, and 8X, so players can choose the speed of the game that they want(see Figure 4.23). The different speeds can be selected by clicking with the mouse or by pressing the key numbers 1, 2, 3, and 4.

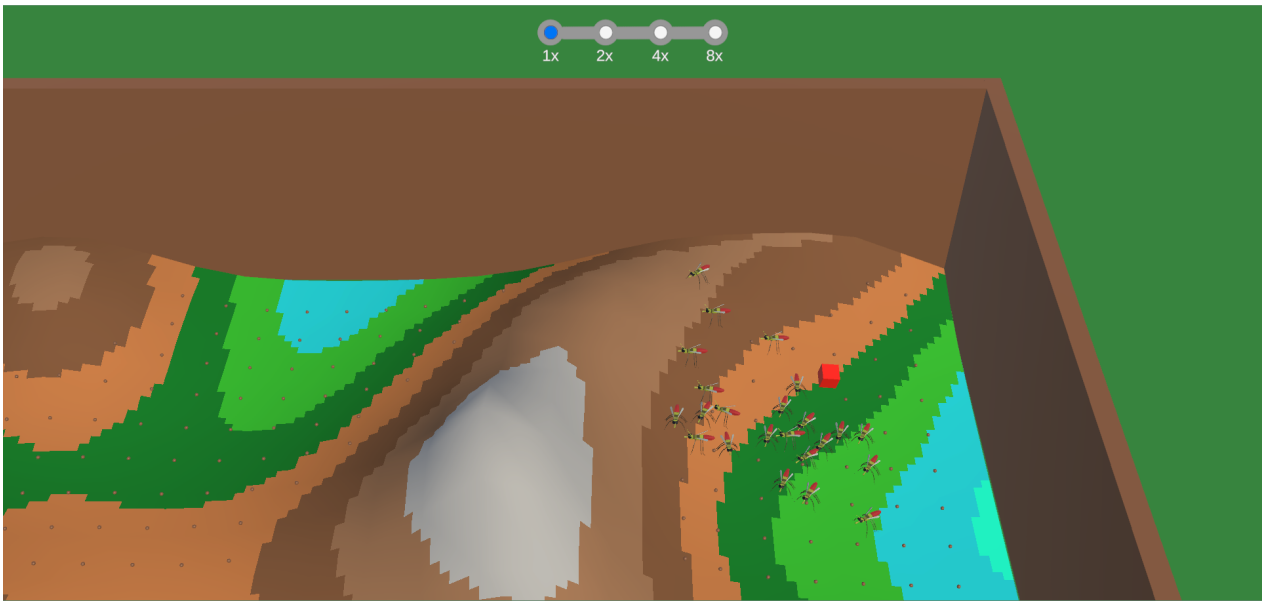

Figure 4.23: The game with the game speed UI implemented

4.10 Results

So, after all the work, this is the result. Garden battle has become a game. A battle simulator game where you can choose between 4 different character types and make them fight on a procedurally created terrain. However, there are some things of the original idea that I could not implement. As you may have noticed, the worm has not been done,

I run out of time, so there are some things that I could not finish. Since the worm was a character so different from the rest, it was very time-consuming to do it and I decided to leave it out. The particles for the animations were another feature that I proposed, but I had to leave it too, it was not as important as the rest, so I decided to spend that time in the others. Another thing that I said was the generation of bugs in the middle of the game. I discarded this one because, in the end, I did not like the idea, I did not know how to spawn them naturally in the middle of a game and I did not want to spend too much time on this.

Apart from this, the objectives of the project are completed. I implemented an AI technique that was suitable for my game. The code is somehow readable, maintainable, and extensible. Although there are some parts that were done in a hurry and are a bit messier, the core part, which is the hierarchical state machine and the behavior trees, is completely SOLID and I can reuse them in other projects or improve them easily. About the design, I think that the gameplay is interesting enough to play some games. And, finally, I think that the art of the game has turned out well, the characters have their own style that they all share and is beautiful enough.

Here, I show a video with a gameplay of the game: `https://youtu.be/7hXsPfDce14`

Chapter 5

Conclusions and Future Work

Contents

In this chapter I give my conclusions about the project and take a little glimpse to the future.

5.1 Conclusions

All in all, this has been a project where I have learned a lot. In this project, I worked on many aspects that I saw in the grade. Coding, modeling, design, AI, etc. It was a big project, the first one that I finished alone. One of the more important things that I have learned is that I am able to stay in a project and be constant, working every day for improving the game day by day. Maybe the game could have been better, but I think that I worked as much as I could and I am happy with the result. Overall, it was a good experience to do this project and is good closure for the grade, with a little more polish, this game is a good project for my portfolio.

5.2 Future work

As I showed on the image of my Trello board, I thought on task to do even after the submission. My intention is to keep improving the game. I would like to add the worm as I said, and do some more tweaks and features to the game. For example, I would like to do special camera movements when a base is destroyed, remarking that something

important happened and pointing it out. Furthermore, since the AI techniques I implemented are very reusable, I probably will use them in other projects and maybe I will improve them. Finally, since this was a way to see that I am able to work constantly on a project, I am sure that from now on I will finish more games.

Bibliography

Here are all the links for extra information about some technical aspects of the report, as well as links of tutorials or programs that I used.

[1] Wikipedia. Solid. https://es.wikipedia.org/wiki/SOLID.

[2] Unity Docs. Navigation system in unity. https://docs.unity3d.com/Manual/nav-NavigationSystem.html.

[3] Unity. Cinemachine. https://unity.com/es/unity/features/editor/art-and-design/cinemachine.

[4] Wikipedia. Version control. https://en.wikipedia.org/wiki/Version$_c ontrol$.

[5] Unity Docs. Unity's particles system. https://docs.unity3d.com/ScriptReference/ParticleSystem.html.

[6] Wikipedia. Functional requirements. https://en.wikipedia.org/wiki/Functional$_r equirement$.

[7] Wikipedia. Non-functional requirements. http://en.wikipedia.org/wiki/Non-functional_requirement.

[8] Miro Samek. Introduction to hierarchical state machines. https://barrgroup.com/embedded-systems/how-to/introduction-hierarchical-state-machines.

[9] Chris Simpson. Behavior trees for ai: How they work. https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior$_t rees_f or_A I_H ow_t hey_w ork.php$.

[10] Imphenzia. Easy low poly character modeling in blender 2.9x. https://www.youtube.com/watch?v=eBOcbYHexAMt=674sab$_c hannel = Imphenzia$.

[11] Sebastian Lague. Procedural terrain generation. https://www.youtube.com/watch?v=wbpMiKiSKm8list=PLFt$_A vWsXl0eBW2EiBtl_s xmDtSgZBxB$3.

[12] turadr. Navmeshcomponents. https://github.com/Unity-Technologies/NavMeshComponents.

[13] Diagrams. Diagrams. https://www.diagrams.net/.

[14] Wikipedia. Object pool pattern. https://en.wikipedia.org/wiki/Object_pool_pattern.

Appendix A

Extra Diagrams

This appendix is just to include some other diagrams I made for the rest of the characters besides the ant. I didn't do as many as for the ant because they were very similar, but at least the state machines were important.

Link: `https://drive.google.com/drive/folders/1JO2dwd_ERg1du5nOpnRJ5HrF_KtATwtX?usp=sharing`

Appendix B

Source code

In this appendix I show some parts of the game code. I only show the most relevant aspects of the project, but all the code is public here: `https://github.com/arturo101199/GardenBattle`

B.1 Hierarchical State Machine

Here there are some classes related to the hierarchical state machines. HierarchicalStateMachine and State classes are the most important ones since they are the core of the machines. But I also show the Transition and Condition classes that are also very important to make the machines work.

```
using UnityEngine;

public class HierarchicalStateMachine : MonoBehaviour, IParentState
{
    [SerializeField] protected StateSelector initialStateSelector;
    [SerializeField] protected Transition[] transitions;
    [SerializeField] AIHandlerHSMs AIHandlerHSMs = null;

    State activeState;

    public void SetCurrentState(State state)
    {
        activeState = state;
    }

    private void Start()
    {
        activeState = initialStateSelector.SelectNode();
        activeState.OnStateEnter();
        activeState.SetParentState(this);
        AIHandlerHSMs.AddMachine(this);
    }

    public void UpdateMachine()
    {
        foreach (Transition transition in transitions)
        {
            if (transition.isTriggered() && transition.TargetNode != activeState)
            {
                transition.MakeHierachicalStateMachineTransition(this, activeState);
                return;
            }
        }
        activeState.OnStateUpdate();
    }

    private void OnDestroy()
    {
        AIHandlerHSMs.RemoveMachine(this);
    }
}
```

Figure B.1: HierarchicalStateMachine Class

```
public class State : MonoBehaviour
{
    [SerializeField] protected Transition[] transitions;
    protected IParentState parentState;
    protected Blackboard blackboard;

    protected virtual void Awake()
    {
        blackboard = GetComponentInParent<Blackboard>();
    }

    public virtual void OnStateEnter()
    {
        transform.root.name = name;
    }
    public virtual void OnStateExit()
    {

    }
    public virtual void OnStateUpdate()
    {
        if (checkTransitions()) return;
        makeUpdate();
    }

    protected virtual bool checkTransitions()
    {
        foreach (Transition transition in transitions)
        {
            if (transition.isTriggered())
            {
                transition.MakeStateTransition(this, parentState);
                return true;
            }
        }
        return false;
    }

    protected virtual void makeUpdate()
    {

    }

    public virtual void SetParentState(IParentState state)
    {
        parentState = state;
    }
}
```

Figure B.2: State Class

```
public class Transition
{
    [SerializeField] State targetNode = null;
    [SerializeField] Condition condition = null;

    public State TargetNode { get => targetNode; }

    public bool isTriggered()
    {
        if(condition != null)
            return condition.EvaluateCondition();
        return false;
    }

    public void MakeStateTransition(State currentState, IParentState parentState)
    {
        MakeBaseTransition(currentState, parentState);
    }

    public void MakeSubMachineStateTransition(SubMachineState currentState, IParentState parentState)
    {
        MakeBaseTransition(currentState, parentState);
        //currentState.SetCurrentState(targetNode);
    }
    public void MakeHierachicalStateMachineTransition(HierarchicalStateMachine currentState, State activeState)
    {
        activeState.OnStateExit();
        currentState.SetCurrentState(targetNode);
        targetNode.SetParentState(currentState);
        targetNode.OnStateEnter();
    }

    void MakeBaseTransition(State currentState, IParentState parentState)
    {
        currentState.OnStateExit();
        parentState.SetCurrentState(targetNode);
        targetNode.SetParentState(parentState);
        targetNode.OnStateEnter();
    }
}
```

Figure B.3: Transition Class

```
public abstract class Condition : MonoBehaviour
{
    public abstract bool EvaluateCondition();
}
```

Figure B.4: Condition Class

B.2 Behaviour Tree

For the behaviour trees I also show here the more relevant parts, in this case, the BTree class, which is the one that handles all the tree. The BNode class which is the template for all the nodes included in the tree. And, the Blackboard class, fundamental element for this project, where I save and share all the data between nodes and states.

```
public enum NodeState { FAIL = 0, RUNNING = 1, SUCCESS = 2 }

public class BTree : MonoBehaviour
{
    [SerializeField] BNode initialNode = null;
    BNode[] allNodes = new BNode[] { };
    Blackboard blackboard;

    private void Awake()
    {
        allNodes = GetComponentsInChildren<BNode>();
        blackboard = GetComponentInParent<Blackboard>();
        setBlackboard();
    }

    void setBlackboard()
    {
        foreach (BNode node in allNodes)
        {
            node.SetBlackboard(blackboard);
        }
    }

    public NodeState EvaluateTree()
    {
        NodeState state = initialNode.Evaluate();
        if (state != NodeState.RUNNING) ResetNodes();
        return state;
    }

    public void ResetNodes()
    {
        foreach (BNode node in allNodes)
        {
            node.OnTreeEnded();
        }
    }
}
```

Figure B.5: Btree Class

```
public abstract class BNode : MonoBehaviour, INode
{
    protected Blackboard blackboard;

    public abstract NodeState Evaluate();
    public abstract void OnTreeEnded();

    public void SetBlackboard(Blackboard blackboard)
    {
        this.blackboard = blackboard;
    }
}
```

Figure B.6: BNode Class

```
public class Blackboard : MonoBehaviour
{
    Dictionary<string, object> blackBoard = new Dictionary<string, object>();

    public void AddKeyValue(string key, object value)
    {
        blackBoard.Add(key, value);
    }

    public void UpdateValue(string key, object value)
    {
        blackBoard[key] = value;
    }

    public object GetValue(string key)
    {
        object value;
        bool found = blackBoard.TryGetValue(key, out value);
        if (!found)
        {
            Debug.LogError("Not found " + key);
        }
        return value;
    }

    public void ResetBlackBoard()
    {
        blackBoard.Clear();
        InitializeBlackboard();
    }

    public virtual void InitializeBlackboard() { }
}
```

Figure B.7: Blackboard Class