DH2323 - Project Report

Andersson, Frida[1] | 970303-5742 | friande@kth.se
Wallenthin, Anders[1] | 921104-5035 | wallenth@kth.se

[1] School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology

Abstract :

This report covers the work process of how we implemented Project Invasive, our final project within the course of DH2323 Computer Graphics and Interaction. It highlights our own personal background and lack of knowledge in the scope of developing a project of this caliber, how we implemented certain functionalities and the struggles that we faced, and suggestions for future work that could enhance the immersiveness of the game MVP that our project resulted in.

Index Terms: Unity, Pathfinding, Game programming.

1 Introduction

During the course DH2323 Computer Graphics and Interaction, we have dived into the realm of interactive programming and animation with Unity. Through several lectures and labs, we explored different aspects and systems. Inspired by Lab 3 - Animation track, we chose to expand the lab into a more fully fledged game experience, ending up with a minimum viable product (MVP) named Project Invasive as our final project for the course. Working on this MVP, we have learned more about how Unity works and how to build an interactive experience, utilizing different systems and learning new ones to further aid our vision of what we wanted the MVP to be.

1.1 Paper Organisation

In chapter 2, we set the stage for our position regarding working in Unity and what our starting point was regarding the course labs and seeking systems to implement in our application. Chapter 3 outlines what and how we implemented each key aspect of the game. In chapter 4, we reflect on where the project could go if one were to improve on our MVP. Lastly, we end with chapter 5 with some closing remarks, reflecting upon the project.

2 Background

To preface this report, we would like to highlight our personal background and knowledge within this subject, as neither of us has any prior knowledge of interactive game programming. This is why we decided to extend Lab 3 and modify it, as it felt like an achievable task to take on due to our knowledge base. This is also the reason why some of the solutions that we implemented have easier solutions integrated into Unity that we did not have prior knowledge of when implementing them, such as the health bars, which were made using a Slider instead of Billboards.

When deciding how to expand Lab 3, we decided to go down a less war-coded route and therefore decided to create a game experience with flowers and bugs, instead of tanks and bullets. We also decided on the technical aspect of implementing a more sophisticated pathfinding algorithm, as the tank movement in Lab 3 was rudimentary.

We researched several different pathfinding algorithms Rafiq et al. (2020), but ultimately settled on the A* (A star) algorithm. One of the main reasons we chose A* is that it is a popular algorithm used in many games, and its simplicity made it feel achievable within our knowledge base. Other algorithms we considered were Dijkstra's algorithm (W3schools, 2025) and the Greedy Best-First search algorithm (GeeksforGeeks, 2022).

3 Impelementation and Results

Starting off, we wanted to run the project through Unity Cloud to facilitate easier collaboration and sharing of project files among team members. Unknown to us, this proved non-trivial to set up. After wrangling with DevOps, Unity organizations, permissions, and several other settings we had never encountered before (having worked with Git repositories multiple times in mind), we resolved this issue and moved on to building the actual application. Visual examples for many of our implementations can be found in the project repository README.md. The version that we now call our MVP provides an idea of what a more involved experience could look like if it were allowed for further development (see some of our ideas for this in Chapter 4).

3.1 Player Character

Our player character (PC), similarly to how it's set up in Lab 3, is controlled by using WASD keys for movement, the mouse to aim, and left-click to spray (shoot in Lab 3). Spraying detects if something is hit by a raycast that scans the enemy layer. If the ray intersects an object with this layer tag, it is considered a hit. This proved to cause some minor issues with our 2D NPC models, which will explained further in 3.2.

Relevant components of the Player Character are the Capsule that runs the movement script and the spray bottle that runs the hit ray script. The Player Capsule also serves as a lock on target for the main camera, which follows at a fixed distance and turns in response to player input.

3.2 Non-Playable Characters

Bugs are the adversarial agents in this experience, trying to eat the flowers that spawn across the map. They are confined to the navigation grid and move across it using our A* implementation (explored further in 3.3). When a Bug spawns, it requests a path to the nearest flower. Once the path is calculated, the Bug moves towards the targeted flower and proceeds to "eat" it upon arrival. Should a flower disappear before arrival, a new nearest flower is picked, and the Bug continues towards this flower instead.

If the player hits a Bug with a ray from the sprayer, it receives damage to its health attribute. This is then communicated visually via a healthbar graphic (described further in 3.6). Once health reaches zero, it despawns. If there are fewer than the predefined number of NPCs spawned, a new one will spawn with a delay.

All of the NPCs are represented by 2D sprites that are constantly pointed in the same orientation as the main camera, to avoid having them "flatten" if the player is oriented any other direction than looking directly at them. This caused the hitray from the player to initially miss NPCs' hitboxes, which was easily fixed by angling the hitray down a few degrees (a visual of this can be found in the Blog.md).

3.3 *Pathfinding*

Our implementation of A* starts off with generating a grid upon which the navigation can be drawn. This is done with the GridManager.cs script, which establishes both grid resolution and size. For this implementation, we choose to stick to a square grid. Each tile is represented by a Node class, which can be found in Node.cs.

Calculating heuristics for deciding the best path is done through *fCost*, which is defined as $fCost => gCost + hCost$. *gCost* represents the cost of moving from a known starting cell to a specific cell. If a Bug starts at a random cell and wants to move to any adjacent cell, the gCost is calculated as $0 + d$, where 0 is the gCost for the starting cell and d is the Euclidean distance from center to center between the two cells. *hCost* represents a cell's distance from the target cell (sort of an inverse of *gCost*). It utilizes the same Euclidean distance to calculate this.

3.4 *Obstacles*

To have something in the way of a trivial straight-line pathfinding for the NPCs to handle, we put in some obstacles in the shape of walls that would have to be navigated around. After the navigation grid is loaded, a set number of obstacles spawn in orientations of 0° or 90°, and of two different sizes. For each wall, the corresponding tiles it occupies are marked as unwalkable, which in turn signals to the pathfinder component that those specific tiles aren't valid targets for seeking the shortest path.

3.5 *Flowers*

The final piece of our game loop is the flowers that spawn after obstacles have been placed, so that NPCs have something to target, and the player has something to (try to) protect. They spawn according to an arbitrary set of scatter and amount. When an NPC interacts with a flower, it decreases its health until it reaches zero and despawns.

3.6 *Healthbar*

To communicate both the items (flowers) and the NPCs' (bugs) health, a slider graphic is placed on top of the relevant models. This slider represents the GameObject's health and serves as the primary indicator to the player on how close a GameObject is to being despawned. This is implemented as a script that is placed on the GameObject that needs it.

4 Future Work

When narrowing down the specifications for the MVP for this project, there were a fair few ideas that we had to deprioritize for the scope of this project. In the following section, we will present some of these ideas, as they are future implementations that could be done to enhance the game experience of Project Invasive.

4.1 *Game design*

Originally, we had ideas on how to further gamify this project. Some ideas that we had were to add a time element, such as having to destroy a certain number of bugs in a set time, or having waves of enemies spawn and the player having to protect the flowers, similar to the Bullet-Heaven style of Swarm in the League of Legends universe (Games, 2025). Developing a simpler version of what Riot Games did in Swarm, therefore, feels like a possible extension of the project.

Other ideas that we had were to add power-ups that spawn randomly or after a certain stage. The power-ups could be faster player movement, more damage on the enemies at each spray, or slower enemy movement. Furthermore, we had the idea of implementing regenerating HP of the flowers. We also had the idea of implementing that if the player sprays the enemy a certain number of times within a set time limit, the enemy would back away from its current target flower and move on to another one. This, together with the regeneration of HP for the flowers, would allow for more strategy from the player, and thus enhance the game experience. Another implementation that could be done is seed spawning, which allows the player to plant new flowers and then regenerate some global HP as the player is aiming to protect their flowers.

4.2 *Systems*

As previously stated, we are new to Unity and game-making. This has led us to create systems that have been shown to be built into Unity. Two examples of this concern are navigation, Health Bars, and NPCs.

4.2.1 Navigation grid Our implementation of a navigational grid works for all intents and purposes, but is bound to a square grid, and all cells are of equal size. Unity's built-in system for Navmesh (Unity, 2025b) would eliminate this and also allow for more dynamic declaration of navigational space in 3D, whilst our solution is somewhat bound to a 2D plane as implemented. So if a more dynamic environment is sought after, say hills and valleys, or caves for that matter, we suggest replacing our GridManagers grid with Unity's built-in Navmesh for a more rigid and future-proof solution, and solve A* navigation based on this instead.

4.2.2 2D elements in 3D space Another similar case can be found with our solution for the healthbar and NPCs. Again, Unity has systems for the purpose of rendering 2D elements in 3D space, Billboards (Unity, 2025a), which could have replaced multiple of the systems we put in place for healthbars and NPCs to work as intended relative our main camera. Replacing our systems with billboards would, in a similar fashion to NavMesh, streamline and better utilize Unity's built-in systems, which would most likely be more rigorous toward expanding the MVP to a full game.

4.3 *Animation*

There is a fair few modifications that could be done to improve the animation of our MVP, but we will highlight two aspects that could be implemented to drastically improve the immersive experience of the game.

4.3.1 NPCs Within the scope of this MVP, we were satisfied with having 2D elements in a 3D space for our enemies, as it served well enough to convey the visual aspect of how different parts of the game interacted with one another. One implementation that could further enhance the game experience would be to implement 3D models that could potentially be used to animate the legs as the NPCs walk over the map, or animate the mouth to depict eating on the flowers, etc.

4.3.2 Player character Currently, the Player "character" (if one can even call it that) is just a capsule with the spray bottle attached to it. This has been sufficient for the scope of the MVP, but not such a visual treat regarding the immersiveness of the game. Therefore, we see a possibility of future work, which is to implement a character of some caliber. Here there is big possibilities within the animation aspect of the development, as the legs, head, arms, etc., could be animated to move in different ways depending on what the user input is.

4.4 Environment

Regarding the physical game space, we see some possibilities for improvement. This would not only give the environment a more dynamic feel but also potentially introduce new possible interactions for the game. These design decisions could work in conjunction with power-ups and similar items mentioned 4.1.

Moving from a strict 2D plane for the game to a 3D environment could immerse the player further into the designed experience of defending their garden against the insect onslaught. Some ideas we had during the project are, in no particular order:

Buildings Large or small obstacles for PC and NPCs to navigate around. It/They would also aid in giving a sense of place for the player.

Foliage could act as obstacles for PC by blocking passing through or giving PC a speed penalty, whilst some NPCs might be allowed to crawl through bushes or under trees.

Bodies of water In a similar fashion to foliage, ponds could be available, moving space to flying NPCs but block or give a speed penalty to PC.

Paths Since the planned environment for the game is some sort of outdoors (garden, forest, etc.), paths could be a natural way to add some way for the player to get some speed boosts. Dirt paths could give a small boost, while gravel or stone paths could give increments of larger boosts to level the playing field between PC and NPCs.

5 Conclusion

Using Lab 3 as a starting point was a smart decision, helping us with code examples and providing easy points to improve/change. Even so, we fell for our own non-understanding regarding what Unity as a development platform could provide through things like NavMesh or Billboard. On the other hand, we had to develop these systems ourselves, learn about basic functionalities which we would not have explored in the same way if we had used Unity's systems from the get-go. Therefore, we find our process valuable in and of itself when considering the new understanding and knowledge we gained along the way.

There are a few points that could be done to make this project more immersive, as explained in 4, but for the scope of what we set out to do with respect to the MVP, we are more than pleased with the results. We have learned a lot about how to write scripts for programs in Unity, how to create a game environment from scratch, and how many functionalities that are built into Unity, which could be used for potential future projects in animation and game programming.

References

Games, Riot (2025). *Minions, Servers, and Progression – The Tech Behind Swarm*. Riot Games. URL: `https://www.riotgames.com/en/news/the-tech-behind-swarm` (visited on 09/22/2025).

GeeksforGeeks (Dec. 2022). *Greedy Best first search algorithm*. GeeksforGeeks. URL: `https://www.geeksforgeeks.org/dsa/greedy-best-first-search-algorithm/`.

Rafiq, Abdul, Tuty Asmawaty Abdul Kadir, and Siti Normaziah Ihsan (June 2020). "Pathfinding Algorithms in Game Development". In: *IOP Conference Series: Materials Science and Engineering* 769, p. 012021. DOI: `10.1088/1757-899x/769/1/012021`.

Unity (2025a). *Unity Documentation - Billboard*. Unity Documentation. URL: `https://docs.unity3d.com/6000.2/Documentation/Manual/class-BillboardRenderer.html` (visited on 09/25/2025).

— (2025b). *Unity Documentation - NavMesh*. Unity Documentation. URL: `https://docs.unity3d.com/6000.2/Documentation/ScriptReference/AI.NavMesh.html` (visited on 09/25/2025).

W3schools (2025). *DSA Dijkstra's Algorithm*. www.w3schools.com. URL: `https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php`.