3.6 - Unity game report

Jokūbas Rusakevičius Koen Haverkort Matthijs Romeijnders Yuqi Liu
Sivan Duijn

July 1, 2022

1 Introduction

The goal of this project is to develop a game with 50k to 100k or more agents in Unity3D using the uCrowds plugin. Processing and visualizing such a high number of agents brings with it a couple of challenges. Because uCrowds handles the processing of the agents, the challenge of rendering is the one we need to solve.

uCrowds brings its own challenges with it. While uCrowds is a good program for crowd simulations, it is not necessarily designed for games. In this paper, we will use uCrowds in ways it might not be envisioned to be used, and explain how we worked around certain limitations.

Our paper is structured as follows. In Section 2, we describe the design of the game. In Section 3, we present the problems encountered during development and the ways they were solved. In Section 4, we discussed what we were unable to implement. Section 5 provides feedback on uCrowds, and Section 6 concludes the paper.

2 Design

2.1 Game idea

The first step in the game design was figuring out which type of game best suits the challenge of 50k to 100k agents. We felt that it had to make sense that there are so many agents in the game. We quickly arrived at a tower defense/storm the castle type game, because most of our other ideas did not fit the challenge as well.

2.2 Design choices

We decided on making a tower defense game where ants storm your house. Contrary to most tower defense games, the path that agents take is not fixed. We thought that the uCrowds crowd simulation had to be the core of the gameplay mechanics. With these ideas we came to the conclusion that the player would not mainly place towers to defend against the ants, but instead the player would have to delay the crowd by placing obstacles that interact with the ECM.

The win condition of the game is that the player needs to *delay* the ants for as long as possible until bug spray arrives by parcel. Again here we deviate from the traditional tower defense mechanics by not killing the enemies, but managing their movement.

In our game, to hide the renders of all of the agents for performance gains, we wanted to use a heatmap to show the density of agents in a fixed area size. This also made the crowd much easier to manage, because dense areas are highlighted and movement is more apparent rather than when you use 50k very small models of ants.

Another way for the player to delay the ants is by using “distractions”. Distractions act as a temporary exit to attract a maximum specified number of agents.

Finally we thought about adding weapons that could eliminate ants in large numbers. This would mean that the player needs to manage their time between strategically placing obstacles and killing ants yourself. This eventually did not make it into the game.

The obstacles that the player can use to manipulate the crowd are distributed by a random resource system. The player gets a steady amount of resources per second and can spend them on obstacles and distractions. The obstacles come in different shapes and sizes.

3 Problems & solutions

3.1 Agent profile configuration

One of the first things we did was looking into uCrowds code to see how we could change how the agents behaved. This was not only because we really wanted fast and dumb ants, but also because we felt that this was a good place to start when learning about the uCrowds codebase. We looked into the documentation provided with the repository and from there it was not too difficult how to change profile characteristics from within the Unity editor.

3.2 Obstacle placement

One challenge that took a little more work, was obstacle placement. Not because the integration with uCrowds was so difficult, but because this was going to be one of the main mechanics of the game, and so it had to be nice to use. We went through several iterations of how the placing actually works, going from a drag & drop method to a click-drag-click method. We enabled rotation with "q" & "e" while the obstacle is being dragged by following the intersect of the mouse with the "walkable area".

We completely separated the uCrowds from the placing mechanic. Using a placeholder object to drag or place the obstacle, then destroying the placeholder and then spawning an actual uCrowds obstacle where the placeholder was placed.

We eventually made multiple different models to give the player more than one option to to manipulate the ant crowd strategically and tactfully.

3.3 Distractions

Doing distractions how we originally wanted was the biggest practical challenge. The original idea was that we wanted a placeable distraction, that interacted with the crowd simulation in real time. Our first attempt was to try to make some waypoints that would be added to agent routes in real time but we could not make it work properly. Because we did not find a way to add waypoints in real time.

The next best thing was to dynamically add an `ExitBlock` to the scene. We were aware that this would never allow for it to have agents change routes, but we thought we could at least have a new `FlowGroup` made, which exits at this distraction. This would function in a similar way as we originally wanted it, it just cannot change agents routes in real time so the strategic use is limited. You are just removing 1000 agents from the level.

The placing of the distraction works in the same way as the obstacles. How it works is the following: First we instantiate an `ExitBlock`, for which we call `AddToSimulator()` to build the `LinkedFeature`. Then we use a changed version of `AddLogicObject()` in the `SimulatorStarter` object, which loops over all new `ExitBlocks`, and all `EntryBlocks` and connects the linked features of new `ExitBlocks` with all `EntryBlocks`. After this we call the same code that is called in the `SimulatorStarter.Update()` function.

Doing it this way meant that you can only place one distraction per level, which is a bug. We have not figured out yet why this is the case. This feature is more of a proof of concept kind of feature in that sense. For the distraction model we originally wanted to use a cake but we could not find a nice cake model or texture for free in the asset store, so we just use a small pink cube.

It must be said that for this kind of implementation, the provided documentation was not enough. In some parts it was even out of date. We felt that if we get an assignment to work with this software like this,

we should be provided with the full, most up to date documentation. It would have saved us a lot of time with this feature.

3.4 Heatmap

One challenge in using a crowd simulation with 50k to 100k agents is the visualization of the crowd. Often, different levels of detail (LOD) are used to dynamically change the model of each agent as the zoom level of the camera changes. Completely zoomed in, the 3D models of the agents is at it highest resolution, with the most vertices. When the camera zooms out, the models transition to lower polygon models (lower levels of detail). Given the LODs are configured correctly, rendering a large amount of agents should be do-able. It is however possible that the lowest LOD model is still too heavy to render at a scale of a large crowd simulation.

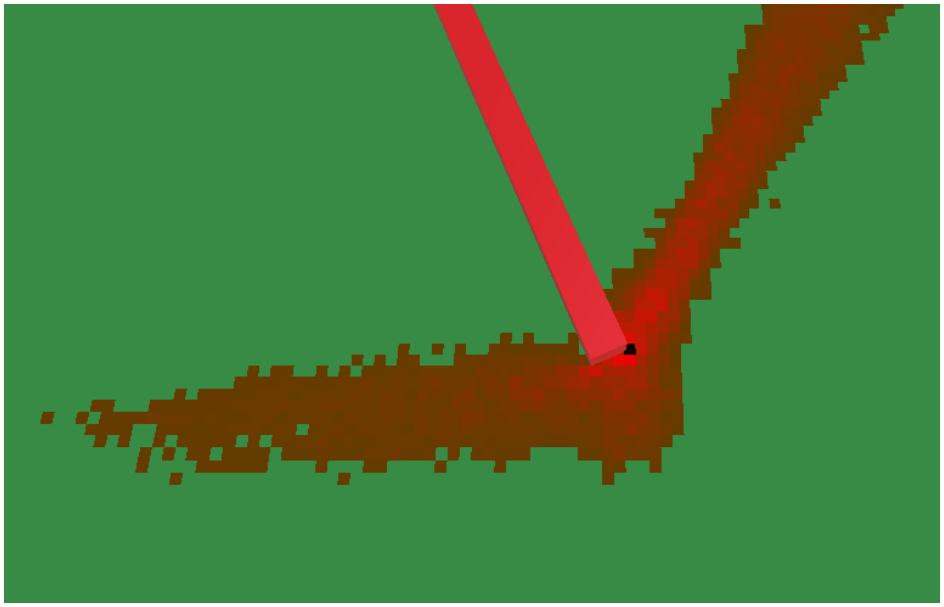

Figure 1: Only the heatmap is shown at a low zoom level

For *The AntGame*, we wanted to make sure the game runs at a good framerate (preferably 60 frames per second) on most commodity hardware. More importantly, we wanted the large number of ants to be manageable by the player. If there are 50k to 100k ants on-screen, it will become hard for the player to understand what is going on at a glance. In our game, it is really important that this is possible, as time is of the essence. To do this, we decided to combine the LOD system with a heatmap: at the lowest zoom level (zoomed out the furthest), the game does not show ants. Instead, colored tiles are shown, each color indicating the number of ants that are present in that tile (see Figure 1). As the camera zooms in, the full resolution 3D model of the ants will be shown on top of this heatmap (see Figure 2), and at the highest zoom level (zoomed in the furthest), only ants are shown (see Figure 3).

As seen in Figure 1, the colors of the heatmap range from brown to red and eventually black. Brown indicates the lowest range of ants, when there is at least one present in that area. As more ants are present, the color slowly changes to a brighter and brighter red. Eventually, the tile will turn black to indicate the highest number of ants possible.

In testing the game ourselves, we found that the heatmap creates easily distinguishable areas where the player can focus their attention. These areas really grab the players attention, thus bring the information we intended them to have. Also, when zooming in we saw that the visualization of the individual ants was chaotic, making it hard to see what is going on.

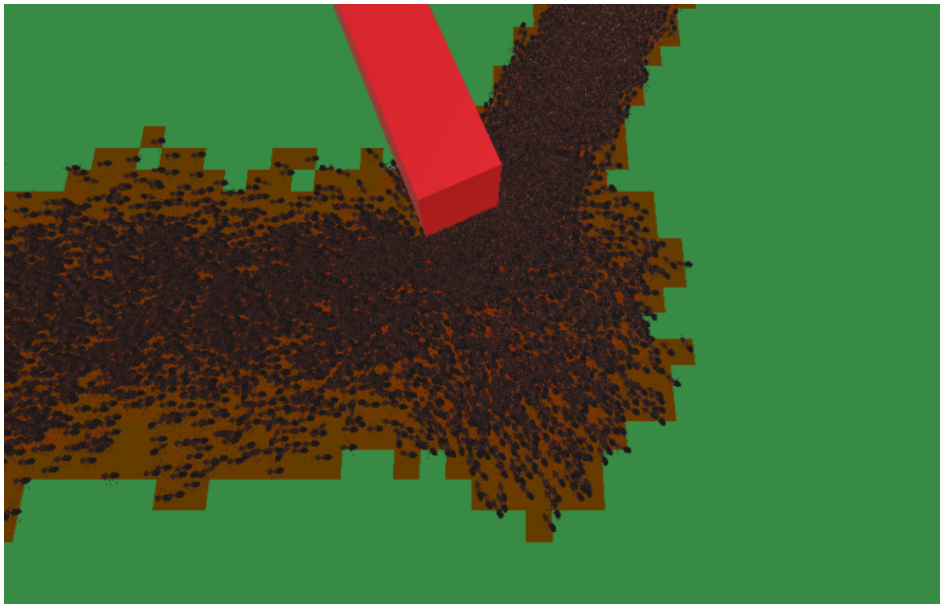

Figure 2: The heatmap is combined with 3D models of individual ants at a medium zoom level

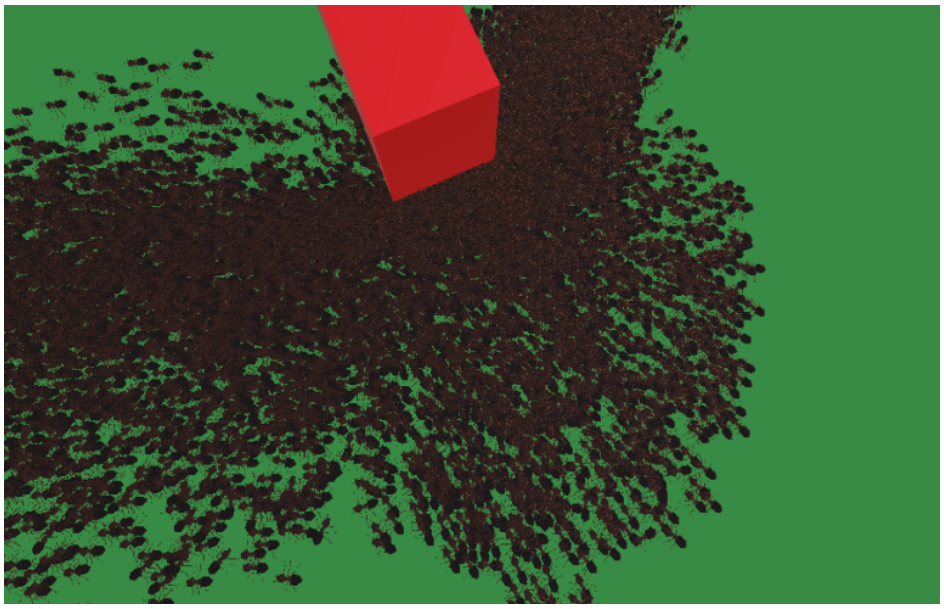

Figure 3: Only 3D models of individual ants are visualized at a high zoom level

3.4.1 Performance

We tested the performance on a computer with a Ryzen 5 2600X, 16GB of DDR4 RAM @3000MHz, and a GeForce GTX 1070 8GB. At around 25000 agents and fully zoomed out (thus only the heatmap), the game runs at around 100 frames per seconds. Zooming in on an area with a lot of agents, the game slows down to around 20 frames per second. While there are less agents on screen when zoomed in, this only proves how much more performant the heatmap is: it runs at a five times higher frame rate while visualizing more agents. Scaling this up to 50k or 100k agents, the difference in frame rate is only higher, as we found that

the game still ran smoothly at 100k agents in a previously playtest.

3.5 Health points

The goal destination for the ants was set by using an `ExitBlock` script example from the uCrowds plugin. This destination was set on the house. To get the current health points for the house we want to see how many ants have reached the exit block attached to the house. We assumed this functionality would already exists on the `ExitBlock` component, something like `ExitBlock.GetCurrentCapacity()`. Unfortunately this was not the case.

We solved it by asking for support in the uCrowds Discord channel. After some messages back and forth, we got a useful answer to attach a `Counter` component to the same area. This worked wonderfully because we can get the total number of agents that reached the area with `Counter.GetTotalAgentNumber()`.

3.6 Checking if the house is reachable

As mentioned earlier, we tried to build a tower defence game with 50k to 100k agents (ants). To prevent the ants from reaching the house, the player can place obstacles and distractions to delay the ants. This way, the environment for the path planning can change at any moment, when the player places a new obstacle. As a tower defense game, if the ants can not reach the end (house), the game will lose its meaning. Therefore, we have to check, when placing an obstacle, whether there is a global path from the start to the goal (house) location, obviously, this is an online problem. The environment of this path planning problem is continuous and not composed with grids. uCrowds own algorithm for global planning of paths is to use A^* on the Explicit Corridor Map (ECM). Apparently, this method does solve the online problem of checking if the goal is still reachable. We tried to solve this problem using the RTT-Connect algorithm. However, as the RTT-Connect algorithm itself does not guarantee convergence, the performance of the algorithm is very unstable and does not solve this problem.

4 Future work

In the end we managed implement almost all of the must haves from our MoSCoW list, except the ability to somehow kill the ants. Initially we thought that this would make the game more fun and less difficult. But, after playing the current version, we discovered that this feature is actually not that crucial, since you can delay the ants quite alright by placing the obstacles in a smart way. It will be a little stressful and require some spatial planning skills, but that is where the fun starts. But still, the ability to kill the ants by, for example, purchasing a tower that shoots the ants, could be added in the future.

Some features that could be implemented in the future are heterogeneous terrain types where ants move slower through a mud pool, more complicated levels with more scenery and ants coming from multiple directions.

Another feature we really would have liked to make was that when ants are stuck, they will eat obstacles. This would add a whole new dimension to the game, because the player is then incentivized to not block off the ants, but to make them walk a certain long route.

5 Feedback

First of all it was already stated but we really would have benefited from full access to the documentation of uCrowds. From the get-go it was hard to understand how the structure of the `Exit-` and `EntryBlocks` worked with the `simulationStarter` and how different functions really interacted. It was hard to really connect exits to entries in code in real time, because they have to be added to and objective list, and the linked features need connecting.

It has to also be said that without justification, it is really weird that so many classes have such long similar variable names: `ContainsObjectives`, `ContainsObjectivesBlock`, `ContainsFlowgroup`, `ContainsFlowgroupBlock` just to name a few. It was very difficult at first to figure out which object has access to which attribute so that you do not know how to use `GetComponent<>` to get what you need.

6 Conclusion

This project taught us how to work with the uCrowds engine, and also taught us loads about game design and working in this kind of context in a group of five. We also all learned more about working with Unity and C# in general. This project was a good exercise in planning with a group.

Our project was a big success in the sense that we were able to easily pass the requirement of having 50k to 100k agents in a game on commodity hardware. With a good commodity PC (PC: Ryzen 5600x, GPU: GeForce RTX 3070Ti, 16GB of DDR4 RAM @3600MHz) we were able to push towards 120k to 150k agents while still maintaining good frame rates.

We came to the conclusion that uCrowds software is not best suited to gaming in general, at least in this current implementation. Even though the uCrowds software plugin with Unity works pretty well, as a developer we did not have access to the tools you need to bring ideas to fruition. Especially because of the way uCrowds is built and the amount of documentation we have access to.

7 Acknowledgements

To build *The AntGame* we used a couple of assets we did not make ourselves, but were either generously provided under a free license, or were bought:

- The 3D model for the ants is *Ant (Anim_run)*, provided by ZERG_LURKER under the TurboSquid 3D Model License on TurboSquid.com.
- For the breezeblock texture we used Rossendy & Brito's Concrete barricades asset.
- For the garden models we used the polygon garden asset pack from Alebrijes Studio in the Unity Asset Store.
- The background music of the game "Weird Village" is part of the *Soft RPG Music Pack*, which is provided by Ezdeha for free on the Unity Asset Store.

Links

- The source code can be found on GitHub.
- The standalone build version of the game does not work properly unfortunately. In the built version, the uCrowds plugin keeps throwing errors and the ants are invisible. We were unable to resolve this issue in time, however the game still runs fine in the Unity editor. The partially working standalone build can still be downloaded here.