NPC Population Control

Maria Barbulescu
mb4767

1. Synopsis

Non-playable characters (NPCs) are characters in video games that cannot be controlled by the player. Controlling the NPC population in video games is essential to a positive, immersive gaming experience for players. Developers often have to make sure that the virtual world they are creating is balanced, which means the player has to be surrounded by the right amount of characters - too many NPCs can make the game feel overcrowded, while too few can make it seem empty and lifeless. The goal of this project was to test different methods for controlling the amount of NPCs in the game.

2. Method

a) *External work*

The starting point and inspiration for this project was [1]. In this paper, the author uses fuzzy logic to implement a machine learning algorithm for population control. The game this algorithm was tested on was a version of the Anthill game, where the NPCs and the player are ants. The player can gather food, explore the hill or kill the NPCs, while the NPCs can only gather food or explore. There is a certain balance between NPCs that are gatherers and ones that are explorers that is set by the developer. There is a global satiation to keep track of. As the ants gather food, the satiation goes up, which leads to the colony reproducing (more NPCs appear in the game). As the satiation goes down, NPCs are killed, disappearing from the game. The goal of [1] is to keep the NPC population steady no matter what the player decides to do, and it does so by regulating the food on the anthill.

b) *Tools and APIs*

I used Unity 3D (v2020.3.32) as an engine for developing a simplified version of the Anthill game. I also used Unity's machine learning toolkit, ML Agents, for implementing the machine learning models.

c) *Initial set-up*

The game consists of a simple grid to represent the anthill. The player is represented by an orange cube, ant NPCs are represented by blue cubes, and food is represented by green spheres (see Figure 1 below). The player can move on the grid, gather food, or kill other ants. All NPC ants are gatherers.

d) *Prey-predator model set-up*

When trying to extend the project to account for prey-predator models, I introduced anteaters as another type of NPCs. These were represented by red cubes.

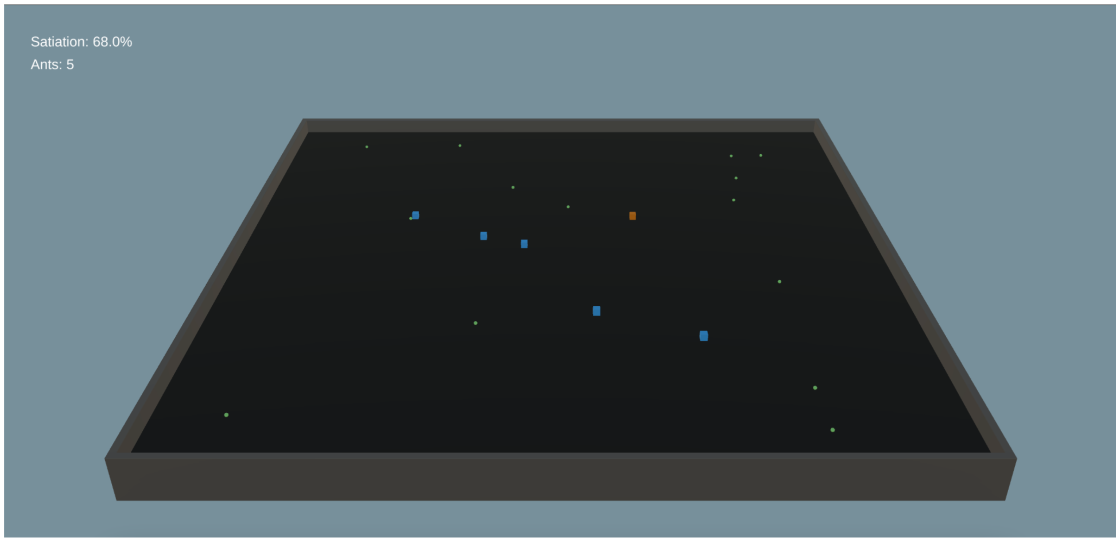


Figure 1: Initial set-up

e) Machine learning set-up

For the first part of the project, I tested population control techniques described by straight-forward mathematical models (mostly using thresholds). I then used machine learning to compare results. The machine learning set-up is very similar to the one described above, with the exception that NPCs are now also considered machine learning agents and their behaviors are altered accordingly (more on this in section 3).

3. Research Questions

a) Is machine learning needed for NPC population control?

This question is rather general, and the answer largely depends on the type of game we are focusing on. For the scope of this project, we will consider games such as Anthill - not very complex, low-budget, indie games.

In the first part of the project, I focused on developing colony dynamics using only threshold-based mechanisms. I only used the initial set-up from section 2, and built the following model. The NPCs choose a random food object and move towards it. If the food is grabbed by another and (NPC or player) in the meantime, the NPC will choose another random food object. As mentioned in the previous section, all NPC ants are gatherers in this small-scale set-up (this means that all NPC ants are tasked with gathering food).

With each food object that is gathered by an ant, the satiation is increased by **1 - 0.3429 * 0.75 * N**, where N is the number of ants in the scene. This factor is taken straight from [1], and it was developed after experimenting with different values (they describe 0.3429 as the average number of food objects gathered in a minute, and 0.75 as a factor ensuring that the colony will grow as long as there is no food shortage and the player does not interfere with the other NPCs). Even if the set-up is different, causing the values to be different (0.3429 does not relate very well to my set-up, as in one minute there is more food gathered), I decided to start experimenting with the given values. I also kept the following thresholds from [1]: when the global satiation is above 75%, a new ant NPC is generated, when it is under 25%, a random NPC is killed. I determined experimentally that it was best to only do this action once every 20 seconds, to allow the new habitat to achieve a balance.

[1] used machine learning for food generation, and here is where our approaches strongly diverge. I implemented the following thresholds for automatic food generation (here, **n** is the current number of ants and **target** is the target number of ants):

- If **n < target / 1.5**: create 3 * n food objects
- Else if **n < target**: create n food objects
- Else if **n > target * 1.5**: destroy a fourth of the food objects

These thresholds were determined experimentally. I tried several values and these ones performed best, but there are many values at which the model performed well enough. This is a good sign, as this means this could easily be implemented to fit more types of games.

The results from this method are surprisingly good. Because the game is simplified, I decided to aim to keep a population of 5 ants. I tested on 3 types of players: the aggressive player, who kills as many ants as possible, the gatherer, who just gathers food, and a mix of the two. When testing with an aggressive player, the ants in the colony were killed very fast, resulting in an NPC population of 0. The food level quickly rose, and new ants started to appear and gather food. It took 4.5 minutes for the NPC ants number to get back to 5, given the aggressive approach of the player. Testing with a gatherer, the number of NPCs quickly peaked at 9 ants. Then, the food became more and more sparse, and the satiation went down, killing NPCs in the process. It took the model 7 minutes to reach back to 5 ants. With a balanced player, the game always stays at around 5 NPCs, which is ideal.

In [1], the system took 20 minutes to rebuild a colony from zero NPCs, and 40 minutes to go from a high number of NPCs to the targeted amount. While our results are much better, it is good to keep in mind that we tested on a smaller scale (5 ants compared to 20 ants in [1]). The number of minuted may not be as telling as to the success of this method on bigger games, but it can still answer the question on whether machine learning is really needed for population control. In short, it is not needed, as we have shown that good results can still be achieved with experimenting with different thresholds.

b) How does this model extend to prey-predator models?

I wanted to expand the above model to account for dynamics between prey and predators. I introduced anteaters as predators in the game and I decided to account for keeping the population of the prey at a steady amount (of 5 ants).

When developing the predator behavior, I decided to let the predators chase the prey, as they would in real life. Then, what would vary in the predators' behavior would be how close they can get to the prey: if they are close enough to eat the ant, they kill it. If not, they simply chase it. How close they can get is determined by the population of NPC ants (since that is the number that we would like to be steady during the game). Here is the new model used (here, **n** is the current number of ants and **target** is the target number of ants):

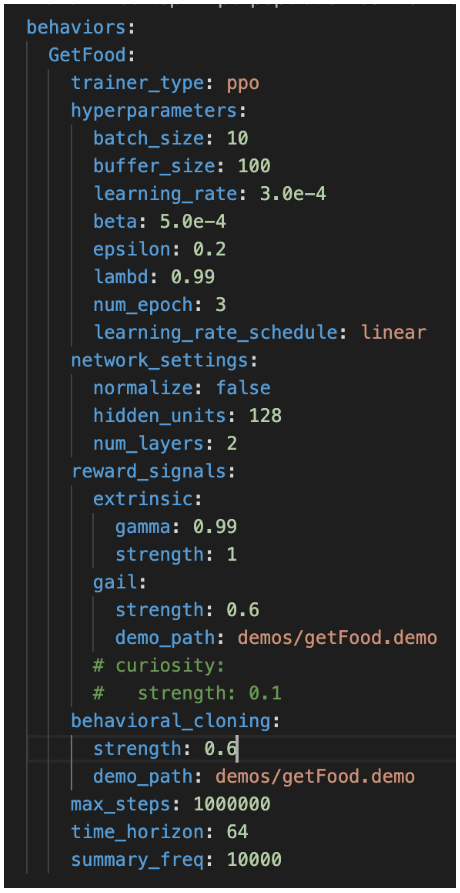
- If **n < target / 1.5**: create 3 * n food objects, anteaters chase the NPC from far away (cannot reach it)
- If **n < target**: create n food objects, anteaters chase the NPC from closer (but still cannot quite reach it)
- If **n > target * 1.5**: destroy a fourth of the food objects, anteaters follow NPC very closely
- If **n > target * 2**: anteaters follow very closely and can reach the ants (and kill them)

I decided to implement the model in this way in order to give a realistic feel to the game, where the predator always chases the prey but can only catch it if it fits the population control premise. This model behaved really well. Testing similarly to a), the aggressive playing mode needed the same amount of time (4.5 minutes) to get back to the target amount, the gatherer took significantly less (5 minutes as opposed to 7, thanks to the predators), and the balanced player was more or less balanced at 5 ant NPCs.

This model actually helped with the problem of population control when used in this way.

c) *How do machine learning models impact population control techniques?*

A big downside to the model in b) is that it does not follow a very realistic hypothesis. The predators always chase the prey, but they should be always to reach it more or less randomly, and not only when it is beneficial to the task of population control. To improve on this, I decided to use reinforcement learning to train the NPC ants to gather food and avoid the predators. I worked on this feature for a long time, but unfortunately could not implement it succesfully. I used the ML Agents API provided by Unity to train my NPCs, but after more than 5 hours of training, the results were still not good enough to be used in the actual implementation of the game. I tried 7 different set-ups, some of them including a combination of imitation learning and reinforcement learning. You can see the set-up that worked best in the figure below.



```
behaviors:
  GetFood:
    trainer_type: ppo
    hyperparameters:
      batch_size: 10
      buffer_size: 100
      learning_rate: 3.0e-4
      beta: 5.0e-4
      epsilon: 0.2
      lambd: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1
      gail:
        strength: 0.6
        demo_path: demos/getFood.demo
      # curiosity:
      #   strength: 0.1
    behavioral_cloning:
      strength: 0.6
      demo_path: demos/getFood.demo
    max_steps: 1000000
    time_horizon: 64
    summary_freq: 10000
```

Figure 2: Parameters of the learning algorithm, including RL and IL

For the reinforcement learning algorithm, I used a reward of +1 for an ant that gathered a food object and -1 for an ant that fell off the board. I also used a survival reward of -0.001 at every step, so that the agent would be compelled to end the episode faster.

I trained several environments in parallel to reduce computation time. In the figure below, you can observe the status of training at different moments in time. A red game field means the last episode was unsuccessful (NPC did not reach the food) and a green one means the last episode was successfully completed (NPC reached the food). Even if there is a clear increase in performance, the final neural network did not perform well during my experiments, which is why I decided to move on with the project without using it. I believe that more computational power/more time could have been very beneficial for this task.

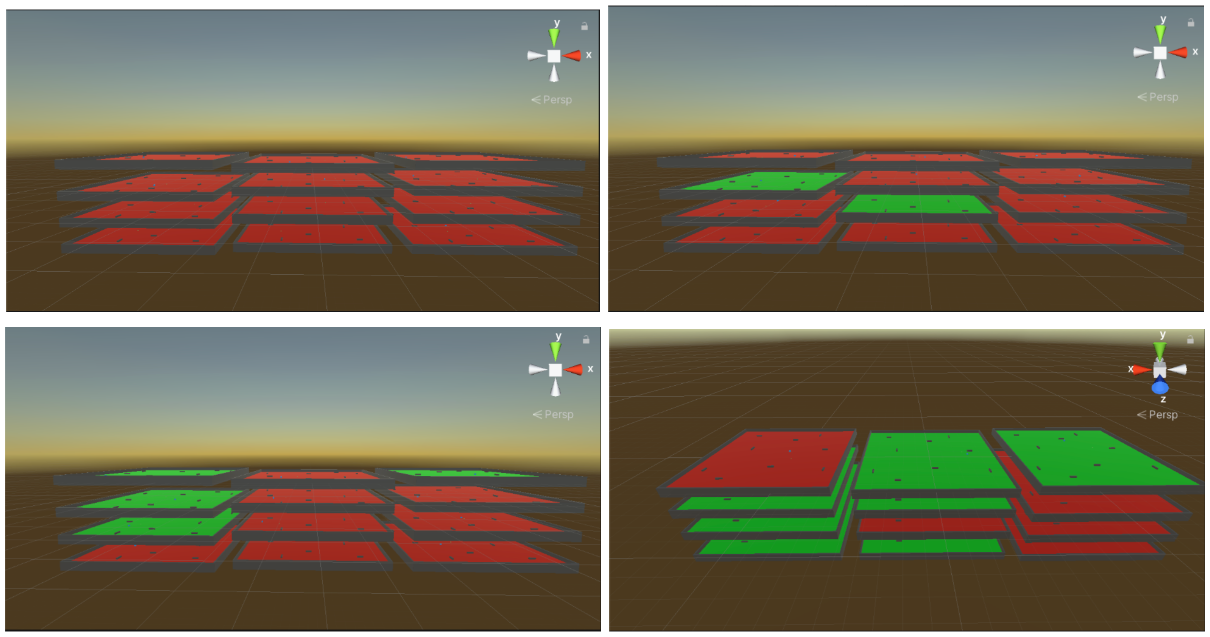


Figure 3: Parallel learning of game environments at different stages of learning

4. Deliverables

You can find my repo at https://github.com/mariabrbz/population-control-npc.

5. Self-Evaluation

I found this project very educational. I never used the Unity ML Agents API, and this was also my first time implementing a reinforcement learning algorithm and an imitation learning algorithm. The learning curve was quite steep, but I am very happy that I experimented with these technologies, as I really felt like I grew as a developer. Using C# was also challenging coming from a background of Python and JavaScript.

What I found interesting and educational also intersects with what I found challenging. The learning part of the machine learning algorithms was very time consuming and took its toll on my machine. Having to run a model for hours without a satisfactory result can be frustrating, especially since there is only so much a basic laptop can do at once. I also felt like I did not have enough time to finish everything that I set out to do.

Overall, I am satisfied with my project. I feel like for all the different elements that it encompasses (building game from scratch, experimenting with the different thresholds and models) it is a very complex project, and that it has some nice results.

6. Course Evaluation

I had a great time attending the course this semester. I loved how many different things I learnt from different fields (some that I never even thought of). If there is one thing that I would change, it would be a soft introduction to these topics. Reading a paper about a very abstract topic can be hard if you have no relevant background, so I felt the need at times for a quick introduction to the field by the students presenting on a given day.

7. Tools & References

References:

[1] Boiński, T. “ANFIS-Based NPC Population Control in Video Games.”, Gdansk University of Technology, mostwiedzy.pl, 2021, https://mostwiedzy.pl/pl/publication/anfis-based-npc-population-control-in-video-games,155118-1

Tools:

[2] Unity Engine: https://unity3d.com/get-unity/download
[3] Unity ML Agents: https://github.com/Unity-Technologies/ml-agents