Emergent Gameplay System

By Ben Zoiss
Advisor: Andy Harris, ajharris@iupui.edu

Introduction

My capstone project is focused on creating emergent gameplay in a three dimensional environment. Emergent gameplay occurs when there are interactions between different agents and/or the player that are not planned to happen. To accomplish this I decided to create a game that simulates a very simple ecosystem. This ecosystem has procedurally generated terrain, an observer, and multiple agents. I implemented this in Godot using their C# API.

Terrain Generation

To create the terrain I decided to use Open Simplex Noise. This allowed me to create a heightmap that represented the general look of the terrain. When I was trying to use the values from Open Simplex Noise to create the terrain I ran into a challenge. I was under the impression that I had to create the terrain mesh by hand at the low level. I spent a lot of time trying to work with this but I ended up finding a better way to do this. To actually create the terrain I used a plane that was subdivided to have thousands of vertices. I then copied that plane into an editor object and changed the height of each vertex. I then deleted the old plane and applied the new one. I also applied a custom shader to the terrain to show height differences.

Observer

I then needed to create an observer that can watch interactions happen and explore the terrain. To accomplish this I created a simple first-person controller. This was all I needed for this but it is possible to extend this simple observer into a fully implemented player or character.

Agents

After I got the terrain and observer working properly I was finally able to implement agents. All agents inherit from the Kinematic Body class from Godot. I also wanted there to be a general base agent but C# does not support multiple inheritance. I ended up with two options from there. Either I had to have my base agent be an abstract class and inherit from Kinematic Body or create an interface for the base agent and each agent will extend the base. I ended up using the second option. At the moment there are three agents; grass, rabbits, and wolves.

All three agents can die and reproduce depending on internal values. Each has a unique behavior but the general idea for the behavior is the same. All agents are trying to survive and reproduce. The grass does not move and only gains the values to reproduce after random time intervals. The rabbits wander around the terrain until they get hungry, when they get hungry they will find the closest grass and eat it. If they don't eat in time they will lose health. Wolves wander around a "home" area until they get hungry, they will then run after the nearest rabbit to eat and go back to their "home" area.

Improvements

While I am content with what I have accomplished in this project I know there are many ways I can improve it. The biggest way would be performance-wise. Right now each agent has multiple timers which is causing the performance to take a big hit when there are a lot of agents. After discussing this with Andy, I would implement a global timer that sends pulses to each agent and after a certain amount of pulses then the agent would do what it needs to. This would get rid of the thousands of timers that currently exist at runtime and reduce it to one. Because Godot has a feature called signalling I would be able to do this easily. When the global timer times out I would emit a signal that each agent would pick up and act accordingly.

Conclusion

Over the course of this project I learned how to navigate the godot environment and use the C# language which I had never used before. I also became familiar with how noise generation works and the basics of shaders. Finally, I learned how to implement many different game objects that interact with each other. Given the time I would improve the timing mechanisms to be more efficient so this could run for a long time and hopefully run to an equilibrium.