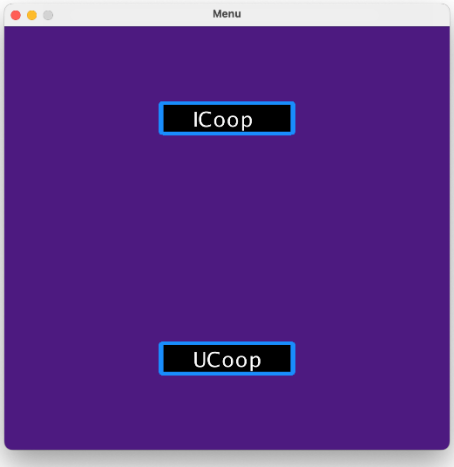
Entry for 'Best Game of CS107'

We decided to pursue two different avenues of extension for this project. The first was focused on enriching the game with new features and content, while the second aimed to explore the idea of bringing autonomous learning agents to the game to see how well they could perform. Thus, when opening the game, you are shown a menu wherein you select a mode, either "**I**Coop" meaning you play the game, or "**U**Coop" meaning you watch the genetic simulation evolve to play the game to the best of its ability. The two different branches of extension are described in more detail separately below.



Part 1:

Log Monster:

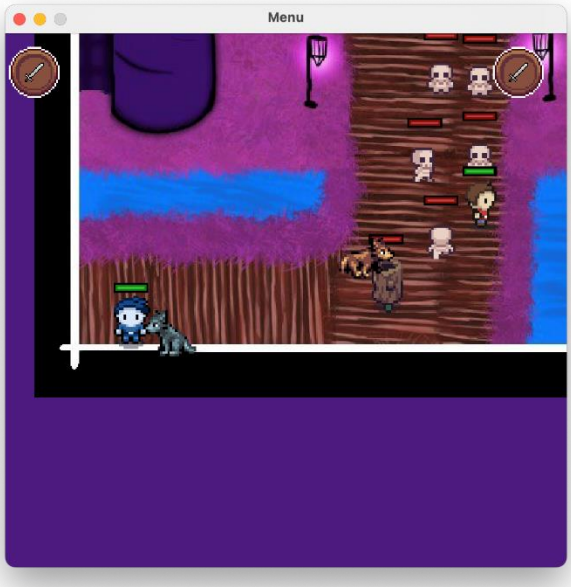
The first of the additions in this first part is a new Foe, the LogMonster. This foe is of particular interest due to his mechanics. He starts off in a sleeping state until the player enters his field of view (an approximately circular area). Then, he wakes up (with a waking animation and state), and enters his attack state, in which he will chase the player, and when they are directly in front of him at an appropriate distance, he fires off a GreenOrb (a new type of Projectile we introduced). The special behaviour of this orb leads us nicely into the next part of the extension – it places grass on the cell just in front of its collision spot with the player.

Grass:

Grass is essentially a decorative actor, as it is completely passable, but simply displays a cutting animation when the player walks through it. An important feature of the grass (as it is an actor that will more likely than not be placed en masse) is that the placement of grass can be done at initialisation automatically if the correct colour is drawn onto the behaviour map of an area (as was done for rocks and obstacles), to which end we added a new GRASS type to the ICoopCellType enum.

Skeletons:

Skeletons are meant to be inconvenient rather than difficult to deal with enemies. They chase the enemy and attempt to deal damage via collisions. Another interesting feature of skeletons is that they override the Foe super class to implement their own death animation.

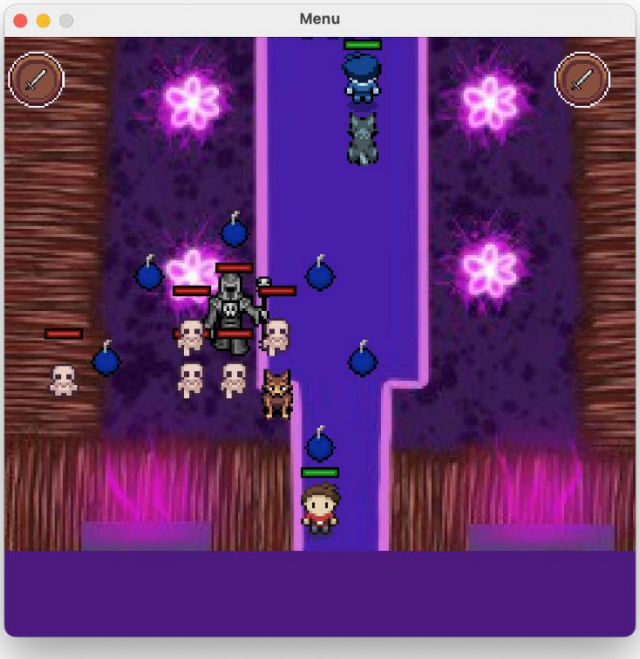


Pets:

We also added in pets, of which there exists a sole type, the Wolf. These are elemental entities that will attempt to follow a player of the correct element.

Boss Fight:

The elements of this part of the extension are explored in the two provided extension areas, SanctumEntrance and Sanctum, and it is only natural that the Sanctum is home to a boss fight. Hence, we created a Dark Lord actor with some unique and powerful attacks and a great deal of health. His attacks are determined by the distance the player (his target) is from him. Depending on this distance, he will either spawn skeletons around him, spawn a ring of bombs, or shoot 3 flame projectiles. If the player defeats the boss, the logical signal of the Sanctum area (determined by the death of the boss) is now on, and a teleporter appears, which takes the players back to spawn.



Part 2:

The evolving of a population is managed by the GeneticGame class, which contains a list of hundreds of instances of the UCoop game all of which are updated until they are complete (as determined by a function of the games, essentially, a certain amount of time elapsed since last meaningful action or player dead), and, of course, only one of them is drawn at a time.

Once all simulations are complete, the GeneticGame ranks the population by fitness (as determined by each game, helped by functions from the players). Then, a certain number of the top performers are chosen as a genetic base for the next generation, where the new instances are drawn from a weighted probability distribution and then mutated. The top performer reserves the sole right to completely preserve his genes without mutation into the next generation.

In terms of the actual implementation of each player's genome, this is represented as a Neural Network, where the weights and biases are the genes of the player. The forward propagation of this network is then used to decide on the player's action. With a certain (very high) probability, this will simply be the action that has the highest activation in the output layer, but rarely, the action is instead picked using the last layer activations as a probability distribution. Mutation of the genes is done by adding small perturbations to the weights and biases with a low probability.

The fitness of a player is determined by his collection of reward objects populated on the map of an area upon initialisation (like grass, rocks and obstacles) in a path drawn on the area behaviour. This system was chosen as the progression of this game is very hard to understand a priori without a basic understanding of how games work.

The fitness of a simulation is then the sum of the fitnesses of its players plus a bonus for completing any areas.

Due to time constraints, we were unfortunately unable to adequately search the hyperparameter space for the best choices of constants and network size, but the best performance we saw looked promising, even for suboptimal hyperparameters, as the agents were quite easily able to learn to beat the OrbWay area (consistently collecting the Orbs), and, with a bit more struggle, were able to then leave the area upon completion. A video of the agents exhibiting this functionality can be found at this link.