Spike: Task 10

Title: Tactical Steering

Author: Sam Huffer, 101633177

# Goals / Deliverables

* A hunter-prey simulation involving multiple agents, with the prey hiding behind obstacles to evade the hunter. The simulation must:
  + Include objects (simple circles) for the prey to hide behind.
  + Distinguish between the hunter and prey’s appearance and abilities.
  + Indicate suitable hiding spots with an “x” or similar.
  + Prey agents must pick a good hiding spot using tactical evaluation.
* Instructions on how to operate the code in the spike report.

# Technologies, Tools, and Resources Used

* Command prompt (for executing and testing the code)
* SublimeText (for editing code)
* Learning materials on Canvas (for instructions and sample code)

# Tasks Undertaken

* I copied the Autonomous Steering project from task 9 into the task 10 folder, and then went through the code to comment out any code or methods that seemed like they would not be necessary for this variant of the project. I didn’t delete the code outright in case I needed it later.
* First, I developed the code for the base requirements of the task:
  + I added the code for generating obstacles and rearranged the code to just spawn one hunter and one evader, the former being made to wander randomly with the wander() method developed in previous tasks.
  + Next task was to adapt the lecture’s notes on calculating hiding spots based on the hunter’s position and implementing that code in the agent class, as well as the code to select the best hiding spot. At this point, I just had the agent arrive at the closest hiding spot. Each spot’s position was marked by an “X”, which was red unless it was the agent’s chosen hiding spot, which was orange.
* The coding of the base requirements was fairly straightforward. The bulk of the time spent on this task was taken up by coding the extensions:
  + First was the ability to avoid objects. To implement this, I took from the lecture notes the ideas of the three feeler lines and the detection box out in front of the agent, but as I couldn’t quite figure out how to implement a few methods detailed in either concept’s code, adapted them to instead use the agent’s position and a point in front of it and an avoidance radius equal to 1.5 times the agent’s radius to check if another agent or an obstacle was getting too close and force needed to be applied to avoid it. These circles are rendered around the agent in blue, turning red if they detect an object to avoid. Later in this coding phase, I added some code such that if an agent would collide with an object, it simply didn’t move to its new position.
  + Next I developed the code for the agent to process several tactical considerations when choosing a hiding spot besides the agent’s proximity to the hiding spot, such as the hunter’s proximity to it, and if traveling as the crow flies to it would lead it to cross the hunter’s field of view. The field of view I implemented, rendered in white, used a point projected out in front of the hunter, and two extra points flanking it on either side, borrowing the offset pursuit’s concept. I also limited the depth of the field of view if an obstacle (and later agents too) would obstruct the field of view.
  + Multiple hunters and prey, spawn them, hiding spot class, hiding spot generation in world
  + Hunter hunts prey, wandering if they’re not close enough or fov markers aren’t in prey’s avoidance radius, pursuing if they are. Gets close enough, eats them.

# Code Snippets

# Instructions for Operating the Code

* E: create a new evader agent.
* H: create a new hunter agent.
* O: create a new obstacle.
* R: reposition all obstacles in random but valid positions. Obstacles are automatically repositioned when the window changes size.
* P: pause or un-pause the game.
* Escape: exit the game.
* F: toggle friction method on and off.
* I: toggle the display of agents’ force, velocity and net desired change in position.

# What I Found Out