Rutgers - Computer Graphics Course - Assignment B1

Unity Navigation Basics

1. This is a **GROUP** assignment.

2. Set up your own **private** repository on GitHub with all members. If you are unfamiliar with Git, please refer to the attached tutorial. Each group project will be set up on a different repository, and your group will submit a log of your commits using the command below. **Your commit messages should be descriptive.**

   ```
   git log --pretty=format:'%h was %an, %ar, message: %s' > log.txt
   ```

3. [5 points] In this part, you must create a simple crowd simulator, where you can select a set of agents using the mouse’s left click and make them navigate to a desired location using the mouse’s right click. Your mouse should also control a free-look camera with WASD, Space (higher), and Shift (lower). Consider going through this tutorial and/or reading this documentation if you are having trouble.
   a. Design a relatively complex environment using different obstacle configurations. Your environment must include:
      i. Different obstacles
      ii. A simple maze
      iii. Several rooms
      iv. Bottleneck areas
      v. Three disjoint levels (in terms of height) with unidirectional off-mesh links. All 3 heights should be walkable by the agent.
      vi. Add connections between the disjoint levels in the form of stairs, bridges, or jumping blocks.
   b. Bake a navigation mesh for the environment.
   c. Create an Agent prefab with a NavMeshAgent component.
      i. The Agent can be a simple capsule with a capsule collider and a rigid body.
      ii. You must instantiate multiple instances of the Agent prefab to create a crowd of agents in the scene (at least 5).
   d. You can borrow a simple mouse script to select agents. If you are implementing your own, you should use raycasts from the Camera.
   e. Use NavMeshObstacle to create obstacles in the environment which can be moved around with the arrow keys along the XZ-plane. These obstacles must carve the navigation mesh (check the carve checkbox in NavMeshObstacle).

4. [5 points] When multiple agents are selected and directed to navigate to the same location, they should not push each other, but stop naturally.
   a. Place dynamic obstacles in the environment which move without controls (e.g., move along fixed waypoints).

   b. Add at least three different weighted planes which agents will **clearly** choose/avoid depending on their values when planning. Think about how you can make this clearly evident.

5. [5 points] Write the following into a small report.
   a. Describe your braking mechanism for agents (4.).
   b. Describe a way for implementing how an agent can avoid obstacles without carving.
   c. Explain the difference in behavior between carving and non-carving options for a NavMeshObstacle. When and why should you use carving? What is the issue with making all obstacles carving? What is the issue with making all obstacles non-carving?

6. Submit the following in Sakai for grading:
   a. Your Unity project in a zip file which contains the "**Assets/Scripts/**" folder, which includes all of your C# scripts.
   b. A zipped WebGL build.
   c. A document (text or pdf) containing:
      i. The text response to (5.).
      ii. A description of the extra credit attempts.
   d. Your log.txt file containing a log of your last commit.

7. Extra credit opportunities:
   a. [1-5 points] Create an adversarial agent which all normal agents avoid. The more apparent their avoidance, the more points will be given. You should be able to select and move adversarial agents just like your normal agents (using the mouse's left and right clicks). You are not permitted to use a large-radius collider to push normal agents away.
   b. [10 points] Extract the navigation mesh into a graph and implement your own navigation from scratch using A*. You must be able to navigate to positions that are not points on the triangulation, and your paths must be as straight as possible.
   c. [15 points] Unity's navigation system has a certain limitation for large-scale pathfinding computations (i.e., for many agents). Precisely describe what this problem is and create a high-performance navigation system which can compute the paths for many agents going to the same destination, and compute the paths for many agents going to unique destinations.