Michael Zhang, Ryan Pollack, Jay Shah, Mukund Chanchlani
NetID: mbz27, rep127, jas1085, mac817

Unity Navigation Basics

Controls / Information

Space - to move up
Left-shift - to move down
WASD - to move
Left-alt - to make your cursor appear/disappear
Left-click - to select/deselect the agents/obstacles you want to control (they will turn red when selected)
Right-click - to put down the destination marker
Arrow keys - to control movable obstacles (movable obstacles are colored grey)
Click on the run button to make the agents path to the destination marker
Yellow plane has a cost of 3
Green plane has a cost of 4
Blue plane has a cost of 5
Note: Do not move the destination marker when the agents are moving it will cause the agents bump into each other

Section 5 Report

5a)

A high-level description of how our agents' braking mechanism works is that the agent will detect if other agents are stationary at the destination point. It will try to avoid bumping into the stationary agents by pathing to open areas with no obstacles. Then, it will resume its path to the destination (this process will repeat if there are other stationary agents in the way).

How we implement this relatively crude braking mechanism was we had a small cube collider that was always facing towards the front of the agent and it will detect if there were any stationary agents that were colliding with it. When it detects a stationary agent it will send out raycasts that span from -125 degrees to +125 degrees relative to the agent's forward vector. These raycasts will detect if there are any obstacles that are in the direction that the ray is facing up to a specified limit. We implemented an algorithm that picks the largest span of rays that detected no obstacles and tells the agent to move towards the middle of the span of rays. If the front of the agent is completely blocked off it will simply move backward. The agent will keep doing this until it reaches a position that is within a radius around the target destination.

5b)

One way to implement an agent to avoid obstacles without carving is to simply use the implementation of our braking mechanism, but it would account for obstacles instead of stationary agents. When the agent detects an obstacle in front, it will send out raycasts to determine the vector direction/groups of vectors that do not have obstacles that will block the agent and move towards that vector. Then the agent will continue towards the destination. In addition, the agent can also calculate the best vector to take that will get the agent closer to the destination faster (taking into account the obstacle in front of it), and it can choose vectors that are close to this vector to move towards.

5c)

Michael Zhang, Ryan Pollack, Jay Shah, Mukund Chanchlani
NetID: mbz27, rep127, jas1085, mac817

The difference between carving and non-carving options for the NavMeshObstacle script is that when enabling carving will have additional options for the movement threshold (the distance an obstacle needs to move in order to have the navmesh to update), the time to stationary (the time the navmeshobstacle needs to wait until the obstacle is treated as stationary), and the carve only stationary option (the navemeshobstacle will only carve when it is stationary). Both carving and non-carving have options to define the size of the obstacle.

You should use carving whenever you want to create an obstacle that you want the agent to avoid. Specifically speaking, you should use carving whenever you have a complex obstacle like a maze to path through since a non-carved might trap an agent in a dead-end. But for simple structures like a straight wall, it would be beneficial to have it non-carved and have a basic user implemented obstacle avoidance script (like the one in 5b) since it will save the cost of calculating the mesh at startup.

Some issues with making all obstacles carving are that it will increase the startup time for the program since it needs to calculate the carve into the mesh, and if there are a lot of dynamic obstacles, it will have increased performance cost than have static navigation obstacles since it needs to calculate the carve dynamic as opposed to calculating the mesh on startup.

Some issues with making all obstacles non-carving are that it will make it more difficult for the agent to avoid obstacles, and the agent might get trapped in a dead-end since it does not know how to path past obstacles that are not carved into the navmesh.

Michael Zhang, Ryan Pollack, Jay Shah, Mukund Chanchlani
NetID: mbz27, rep127, jas1085, mac817

Implementations and Ideas (Braking mechanism) - Dev Notes

In implementing this braking mechanism we had many versions and ideas here are a few of them.

1 - One idea we had was to disable the navmeshagent script and enable the navmeshobstacle script whenever the agent is stationary and is near the target destination. Even though this method does indeed work very well but when disabling the navmeshobstacle script and reenabling the navmeshagent script it will cause the agents to warp to different locations (that are close to its original position). This implementation in my opinion was way too clunky and we felt like it was not in the spirit of what the instructions wanted us to implement.

2 - So another idea we had was to have the agents stop in a radius around the target point whenever it detects a stationary agent in the way. Implemented the agent such that it would send out a raycast to test if there was another agent in front of them whenever it was within a general radius of the target destination and it would simply stop. But this solution had problems when agents were clumped up and especially when the agents formed a line (Since the furthest agent would simply push the other agents out of the way until it reached the radius around the target destination).

[Current implementation of the braking mechanism]
3 - My third and last attempt at implementing the braking mechanism is the one we stuck with which was described in 5a) in the report. We called this implementation crude since there is possibly a situation where the agent might not be able to find a position to stop and it will be suck going in circles trying to find a place to stop around the target destination. If we had more time we would try to polish up this implementation such that it would assign the agents a specific spot around the target destination which will be determined by the positions of the agent in relation to other agents and obstacles that are in the way.