RUTGERS

Rutgers - Computer Graphics Course - Assignment B1

Unity: Navigation Basics

This is a group assignment.

Setup Git Repo for Unity

First, you must set up a *NEW* git repo on which your whole team can collaborate. The purpose of having a git repo is mostly for you to be able to work on your projects together, without necessarily having to meet in person. Using git, specially when working on a group, will make creating a project not only easier, but also much more efficient. So get start using your git repos, and feel the magic of git! (Refer to “Git Introductory Tutorial” available on sakai resources to learn about git.)

NOTE

1. This git repository is independent of the one you are using for the SteerSuite (A) assignments.
2. You will create one git repository to be used for all Unity (B) assignments. Each group will thus maintain 2 git repositories, for SteerSuite (A) and Unity (B) assignments respectively.

Follow these instructions and setup your group git repo for Unity projects

1. From the link below, “fork” the template structure of unity projects into your group account on GitHub (refer to A1 instruction if you have not created a group account yet): https://github.com/Rutgers-CG/Unity
2. Now each member of your group can clone the repo and start using it. Read here for setting up line ending in your specific platform (win, linux, mac) so members on different platforms can work together:

https://help.github.com/articles/dealing-with-line-endings/

3. Change README.md file to your personal information. Add your team logo and website (see part 4 for team website creation info). Update README.md as you develope each unity project. Of course remember to push to repo after you make changes, and always to pull before you start changing/adding new things.
4. Make sure not to change the implemented structure of file system.
5. Look inside .gitignore text file: this is where you can specify what files should be ignored by git when it pushes/pulls/clones. Do not remove/change anything on the file. Read about .gitignore and its syntax here: https://git-scm.com/docs/gitignore

Simple Navigation [5 points]

In this part you must create a simple crowd simulator (navigation system), where you can select a single or a set of agents (using a mouse interface) and can click on a desired location in the environment to have them navigate towards.

Useful materials and readings:

1. Unity Navigation Tutorials: http://unity3d.com/learn/tutorials/topics/navigation
2. Unity Navigation documentation: http://docs.unity3d.com/Manual/Navigation.html

Your simulation must comply with the following **requirements**:

1. Implement a FREE LOOK camera script. Preferably, you will control all the interfaces I/O from this script. I strongly suggest you against including any IO handling logic within your agents prefab. Agents only hold state (attributes) and actions (affordances - i.e. GoTo())
2. Design a *relatively complex* **environment** using different obstacle configurations. Your environment must include:
    a. Different obstacles
    b. A simple maze field
    c. Several Rooms

    d. Bottleneck Areas
    e. Three (3) different height with **1-way off-mesh links. All three heights should be walkable for the agents.**
    f. A connector between unconnected planes such as stairs, bridges or even more fun, jumping blocks.
3. Compute a navigation mesh for the whole environment. Such planes can be simple “Planes”, cubes or whichever primitive of your choice.
4. Create an "Agent" prefab with NavMeshAgent component to allow agents to navigate in the environment while using the navigation mesh.
    a. The Agent can be a simple capsule with a capsule collider and a rigidbody.
    b. You must instantiate multiple instances of the Agent prefab to create a crowd of agents in your scene. At least 5.
5. You can follow this game example to create your own or borrow a simple mouse script to select agents and specify which destination to navigate towards. The example shows you how to use rays to detect a mouse click position in your scene.
6. Setup a "Director" script, in addition to the IO handler one, that keeps track of all the selected characters and sends messages to each character's component to move to the specified destination.
7. Use NavMeshObstacle to create obstacles in the environment which can be moved around. Obstacles should be selectable, and movable using arrow keys. The obstacles must carve the navigation mesh (check the carve checkbox in NavMeshObstacle).

Refined Navigation [5 points]

1. When multiple **agents** are selected and directed to the same location, these **should not “bunch up”** but stop before colliding and pushing each other.
2. Place **dynamic obstacles** in the environment, which move during the simulation. You do not have to spend time refining their logic, simple control points will do.
3. Add at least three **different weights planes** which agents will choose/avoid depending on their values when planning navigation.

Report [5 points]

1. Describe your “breaking” mechanism to detect that the location is already occupied by some other agent when moving in crowds.

2. Describe a way for implementing how an agent can avoid obstacles with not-carving option. (hint: a way was discussed in details in class).

3. Explain the difference (in behavior) between carving and not carving option for a NavMeshObstacle? When and why should the carve checkbox be active/deactive? Describe the problem with these two situations:
   if we make all obstacles carving.
   if we make all obstacles not carving.

Extra Credit

Add some “Nazgûl” agents! These agents can be selected and moved like others, but the other normal human agents should always avoid them at any cost, by a rather far distance! So for example if one of them enters a room, all human agents in that room should run away to other places, and when human agents are navigating, they should avoid these agents! Remember: Nazgul agents must be able to move seamlessly, like any other agent. Mention in your documentation how you implemented this part. (hint: it is possible to use both NavMeshAgent and NavMeshObstacle on same agent, but it is tricky to handle that, very tricky! some good/smart scripting is needed.)

Any other **idea**, worth of extra credit at the grader’s discretion, **you** might **come up with** working in your simulation. A description and implementation of these ideas must be described in your report.

Submission

1. A web playable version of your game which includes all the required features described in this instruction. It should be an stand alone “WEB APPLICATION” which can be executed using a browser (e.g. Firefox). Do not submit windows or mac builds.
2. A graphics.cs website update containing a description of this assignment, your extra efforts, and a video demo of your game, showcasing all the features. Points will be deducted if the update is not sufficient. (All videos must be uploaded to our youtube channel).
3. A text document containing your report.