UUCS Crowd Simulation Master Class

November 16th, 2018

Authors: Dionysi Alexandridis, Simon Dirks, Wouter van Toll, Yiran Zhao

Introduction

In this assignment, you will use the UUCS crowd simulation plug-in to set up a simple crowd simulation in the Unity game engine. When finished, you will have a working simulation with a crowd of characters (agents) realistically avoiding obstacles and moving towards a common goal.

Those of you that have never worked with Unity before should be able to fully setup the basic simulation, without writing any code. For those of you with more experience, there are bonus exercises ranging from easy to more advanced.

NB The manual, tutorial movies and API reference can be found here: https://ucrowds.com/documentation/unity3d/.

Exploring the project

The project contains a number of directories. Below are the directories that you will be using in this assignment. It is important not to change any files in any of the other directories since this may break the plugin.

Demo: This directory contains three scenarios involving important features of UUSC Plugin. You may want to modify these scenes in order to set up a crowd simulation.

Materials: This directory contains material objects that you can add as a component to objects on the scene to give them a certain color. This is useful to make a distinction between the different objects.

Prefabs: This directory will store any re-usable Unity prefabs that you create. We will introduce prefabs later in this tutorial.

Scripts: This directory contains all C# scripts of the UUCS plug-in. After you have finished the basic tutorial, which does not require any coding, you can try the advanced assignments in which you can use scripts from this directory (or add scripts of your own).

Warning: Please do not adjust ANY of the child directories, since this may break the plugin!

Part A - Setting up a basic simulation

Introduction

The center of the simulation is the Environment object. This object handles the setup and running of the crowd simulation. The UUCS Crowd Simulation plugin currently supports *one* Environment. In the Unity scene, the Environment is an *empty GameObject* since it only serves as a container for its children, and is not actually an object in the simulation itself.

Step 1: Creating an Environment

We will create an Environment by using the GUI. Create an empty scene, and then open the UUCS Simulation Setup window, click on "Window", and select "UUCS Simulation Setup".

First, you need to input your license information and user information in the window. You can change the password through the API function "UUCS_ResetPassword".

After setting up your login information, you will see the window as shown in Figure 1.

Set the name of the Environment to something of your liking and press the 'Create' button. We will not modify any of the other properties for this exercise. If you are interested to see what these other properties do, please check the documentation at https://ucrowds.com/documentation/unity3d/manual/cs-environment.

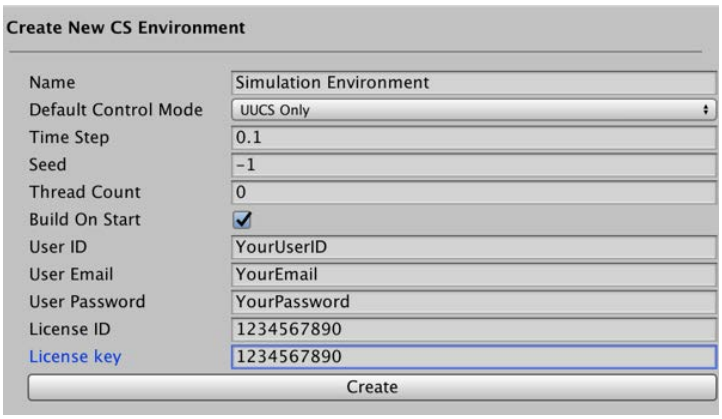


FIGURE 1

After having pressed the 'Create' button, you will see an Environment object in the hierarchy. The UUCS GUI window has also changed, because it has detected an active Environment object (see Figure 2).

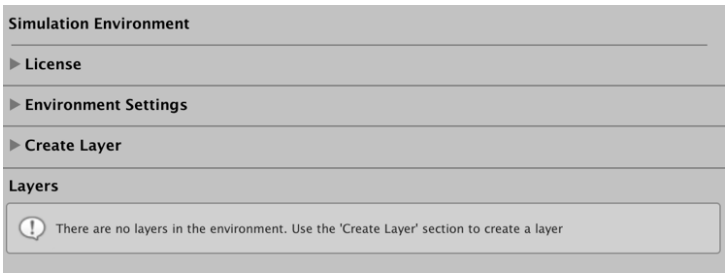


FIGURE 2

You can always adjust the properties we just set under the *Environment Settings* section.

Step 2: Creating and modelling a Layer

The next step in setting up a crowd simulation is creating a Layer object. Just like the Environment, a Layer is an empty *GameObject*. It will be used as a container for the *Characters, CharacterGroups, WalkableAreas, SpawnAreas, GoalAreas, Obstacles*, *RemovableObstacles and ActivityAreas* in the simulation. We will refer to these objects as *layer objects* from here on.

An Environment can contain multiple Layers, and each Layer is an independent simulation in a way (unless they are connected). Layer objects only interact with layer objects of the **same** layer. For example, a character on one layer will not avoid any obstacles placed on other layers (unless they are connected).

To create a Layer, open the *Create Layer* section in the UUCS Window (see Figure 3). Give the Layer a name and click the 'Create' button. The hierarchy will then show that a Layer has been created as a child of the Environment object (see Figure 4). The UUCS Window has also changed: within the *Layers* section, we are now presented with the options to create and modify layer objects.

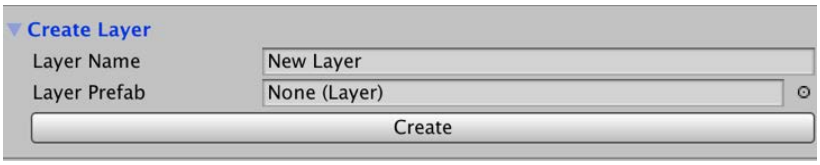


FIGURE 3

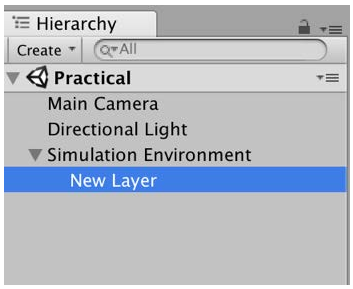


FIGURE 4

Step 3: Creating and modelling a Walkable Area

To indicate the areas on which our characters will be allowed to walk, we need to define *walkable areas* in our Layer. Click the Areas tab in the UUCS window. As you can see, there are no Walkable Areas in our Layer yet (see Figure 5).

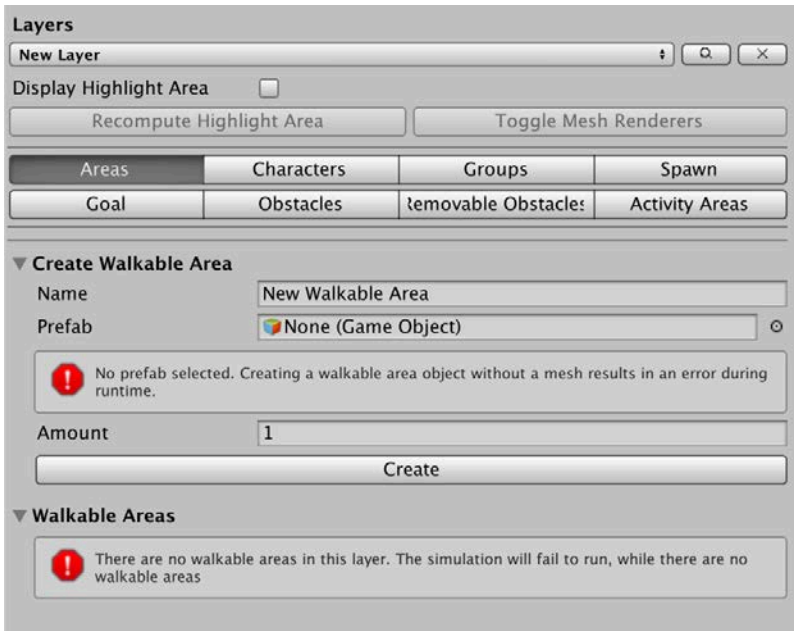


FIGURE 5

Add a new plane to the Layer, by right-clicking the Layer in the hierarchy and choosing 3D object → Plane. Make sure this plane is a child of the Layer in the hierarchy.

To resize the plane, click the Scale icon (shortcut: R, see Figure 6), and drag the handles of the plane. You can also drag a Material onto this plane (such as the pre-made walkable area material) to give it some color.

FIGURE 6

There are two ways to convert this plane into a Walkable Area object. The first way is by selecting the plane, and adding a "Walkable Area" script component to it via the Unity inspector (see Figure 7). The second way is by dragging the plane into the prefab field in the Create Walkable Area section of the UUCS window (see Figure 8), and clicking 'Create'. To ensure you have created your walkable area correctly, make sure that it is listed under the Walkable Areas menu in the UUCS window (see Figure 9).

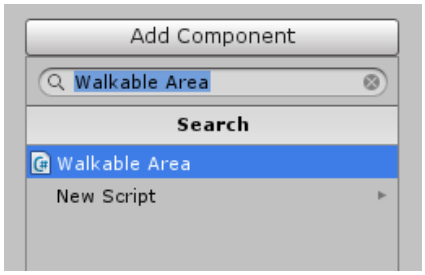


FIGURE 7

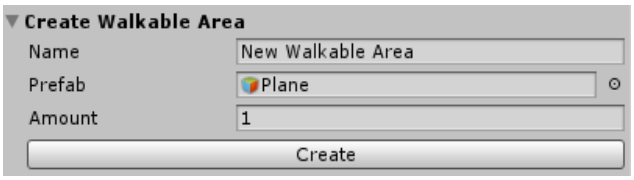


FIGURE 8

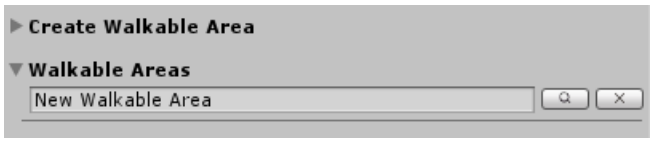


FIGURE 9

Step 4: Adding obstacles

An obstacle indicates an area in which characters are not allowed to walk. They can be added on top of walkable areas to 'override' their meaning: for example, a walkable area could be the floor of a building, and obstacles can represent the objects on this floor that need to be avoided.

We will now create our first obstacle. Add a new Cube object to the Layer, and resize it so that it blocks a part of the walkable area (plane). Again, you can drag a Material on top of this cube to give it a color.

To convert this cube to an obstacle, click the Obstacles tab in the UUCS window, and drag the Cube to the Prefab field under the Create Obstacle menu (see Figure 10). Click 'Create', and when you select the cube, you will see a green outline indicating the obstacle's bounds (see Figure 11). You should also see your newly created obstacle listed under the Obstacles menu in the UUCS window (see Figure 12).

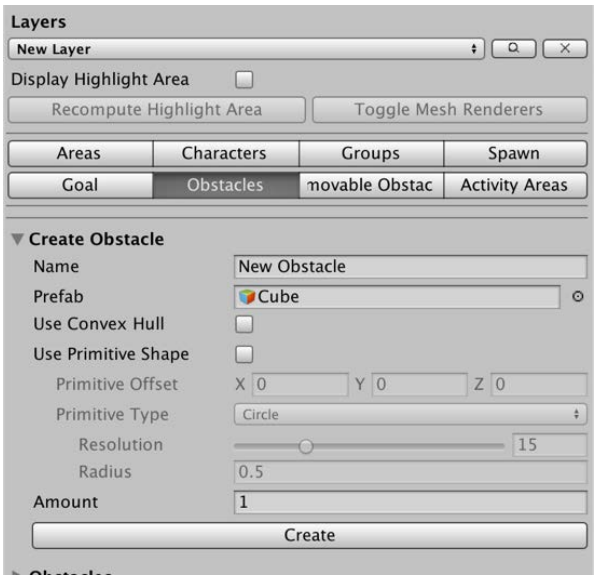


FIGURE 10

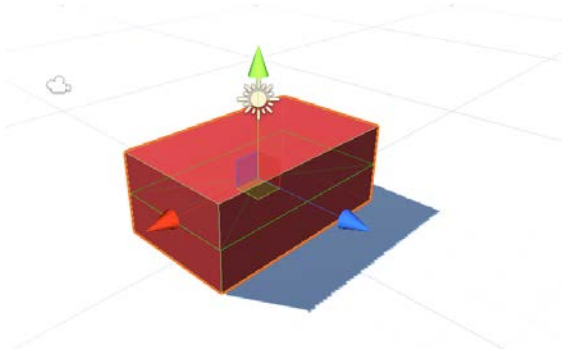


FIGURE 11

FIGURE 12

We have now defined a basic environment in which characters will be walking. Feel free to add more walkable areas and obstacles to the layer in the same way, to make the environment more interesting.

Step 5: Adding a character

It is time to add a character to our simulation. First, create a 3D capsule in our Layer, and move it (shortcut: W) so that it stands on a walkable area and does not collide with any obstacle. Set the Y position to 1 (see Figure 13) to see the entire capsule instead of just the upper half.

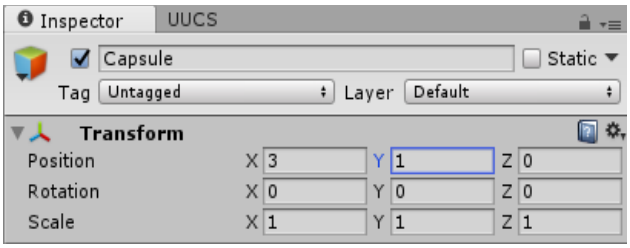


FIGURE 13

Next, we will convert this capsule to a UUCS character. Just like with the obstacles and walkable areas, we can either drag the capsule to the Create Character Prefab field, or add a "Character" script component to it via the inspector. Use either way and leave the settings at default. If done correctly, the character should be listed under the Characters menu in the UUCS window.

Step 6: Setting the camera and previewing our scene

Click on the Main Camera in the hierarchy. You will see a Camera Preview in the lower right of your Scene view. Make sure that the entire walkable area is visible in the Camera Preview (for a simple top-down view: Y position = 20, X rotation = 90). Now, click the Play button in Unity to start our game.

In the background, the UUCS plug-in will have created a *navigation mesh* based on all walkable areas and obstacles. The plug-in can use this navigation mesh to let characters find paths through the environment, which they can then traverse over time while avoiding other characters. Thus, the navigation mesh will be the basis for our crowd simulation.

However, in the current situation, you will see that our character is not moving at all. This is because the character does not have a goal yet, so it has not computed any path. To fix this problem, we can use goal areas.

Step 7: Adding goal areas

Click the Play button again to go back to the Scene view. We will add our first goal area to the environment. In the UUCS window, under Goal → Create Goal Area, click 'Create'. You will see a flag icon appear in our scene, which belongs to a new Goal Area object in our Layer. Click this flag to edit the goal area's bounds and move it around. Make sure that the goal area lies on walkable areas and does not lie in any obstacles.

Next, we will assign this goal area to our character. Click our character, and (in the inspector), drag our Goal Area from the hierarchy into the 'Goal Area' field of the Character script (see Figure 14).

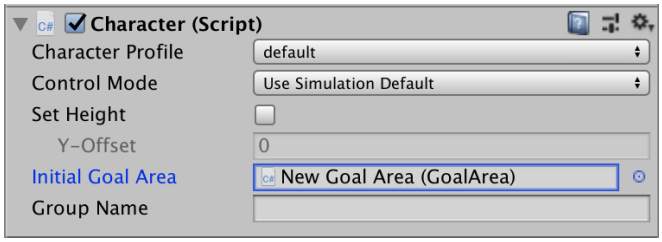


FIGURE 14

Click Play to start the game again. If you have performed all steps correctly, you should see your capsule moving towards the goal area while avoiding obstacles. In the background, the UUCS plug-in has chosen a (random) point in the goal area as the character's goal position. The plug-in has computed an obstacle-free path for the character to that goal position, and the character is now traversing this path.

Part B - Creating a crowd

Introduction

In the previous part, we have manually created a single character. In this part, you will learn how to generate larger crowds.

Step 1: Saving the character prefab

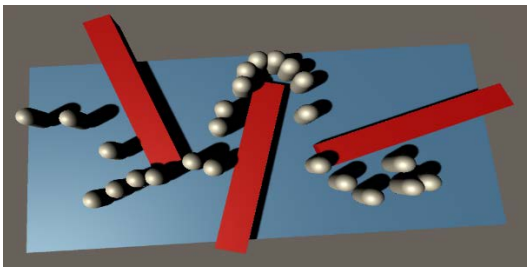
It is useful to save our previously created character as a prefab. This allows us to re-use the capsule as a blueprint for all characters. If you do not know what a prefab is, please check http://bit.ly/unity-prefabs.

To convert the character to a prefab, simply drag our Capsule character from the hierarchy to the Prefabs folder. Now that we have saved our capsule as a prefab, we can safely delete it from our scene by selecting it and pressing Delete. We have now removed this specific character, but we will soon add a whole crowd of characters instead.

Step 2: Creating a spawn area

Click the Spawn tab in the UUCS window, and expand the Create Spawn Area menu. Drag our newly created character prefab into the Character Prefab field. Drag our Goal Area from the hierarchy into the Goal Area field. Spawn Interval is the number of seconds between each generation of characters. Let us set the spawn interval to 2 to increase the rate at which the characters will be spawned. Spawn Count is the number of characters spawned in each generation. Let us set the Spawn Count to 3.

Click 'Create', and a Spawn icon appears. Drag the spawn area to the desired location and press Play. Now, every two seconds, three new characters are spawn in the spawn area and they are moved to the goal area. An example screenshot is shown below.



That's it! You have created a simple crowd simulation by:

- defining the environment with walkable areas and obstacles;
- creating a prefab for a character in the simulation;
- using spawn and goal areas to automatically generate characters with goals.

The UUCS plug-in can do more, though. If you have time left, try out some of our bonus exercises.

Part C - Bonus exercises

If you have finished the above tutorial, you can take a look at these bonus exercises to get a better feeling of what the UUCS plug-in can do. Of course, feel free to ask us any questions.

You can start with Exercise 1, which does not require any coding. For the students who already have worked with Unity3D or have some experience in writing code, we suggest to try out Exercise 2 and onward. These exercises require you to write some code of your own. They are meant for students who already have some experience with writing code in an object-oriented language such as C#. For these exercises, you will have to look at the scripts in the *Assets->Scripts->Framework* directory. It is likely that you will write your own scripts that inherit from the

scripts that already exist. For a full documentation of the plug-in's API, please check: https://ucrowds.com/documentation/unity3d/reference.

Exercise 1: Other Obstacles (easy)

We have used cubes as obstacles up until now, but you are not restricted to using cubes. You can download any 3D model in the Unity asset store and use this model as an obstacle. Then you could use a Convex Hull Mesh as the object bounds.

Next, try to rotate the obstacle. You will see that the obstacle's bounds have become bigger than it needs to be. Try changing this by using the 'Primitive Shape' option instead of the Convex Hull Mesh. Tweak its parameters until you are satisfied with the result.

Exercise 2: Click Interaction (intermediate)

Add a script that lets you left-click on a character to select it, and right-click on a position to set that position as the selected character's goal. Hints:

- Keep track of the character that has currently been selected.
- In your script's *Update()* function, use Unity's *Raycast* to check if a character has been clicked.
- Check the UUCS API for information on how to set a specific goal for a character, without requiring the use of goal areas.

Exercise 3: Goal Areas (intermediate)

Add more goal areas to the environment. Change the simulation such that a character receives a new goal position (in a different goal area) as soon as it has reached its current goal. Hints:

- Make a new C# file (script) and place this within the *Scripts* folder;
- Inherit from the character class;
- Overwrite the *Update* method (which is a default method for Unity's *MonoBehavior* objects), and call the base functionality.
- Check if the current position (nearly) matches the set goal position of the character.
- Reassign a goal position if the character has reached its goal position.

Exercise 5: Removable Obstacles (hard)

This bonus exercise is meant for students who already have some experience with writing code in an object-oriented language. Our goal is to use *RemovableObstacles* in our scene. While regular *Obstacle* objects are static and cannot be removed, *RemovableObstacles* can be removed during runtime as they cause the navigation mesh to be dynamically recomputed.

The *RemovableObstacle* object has two methods that are important for this exercise:

- *TryAdd()*: this method tries to add the removable obstacle to the environment.
- *TryRemove()*: this method tries to remove the *RemovableObstacle* from the simulation.

You are entirely free to implement your own triggers for when you want to add or remove a *RemovableObstacle* from the scene. To give you some inspiration, we will give you some suggestions on what you could do:

- Inherit from the *RemovableObstacle* class. In the update method, add some triggers for removing the obstacle from the simulation;
- Add a trigger to a goal area, where a random *RemovableObstacle* is placed somewhere in the scene if *n* characters reached their goal;
- Every *n* seconds, add/ remove a *RemovableObstacle* from the scene.