UNIVERSITY OF
WOLVERHAMPTON
KNOWLEDGE ▪ INNOVATION ▪ ENTERPRISE

Unity Game Engine Activity

Student Workbook

Activity Title:
Ant Colony Optimisation Example.

Activity Description:
In this student activity, you will develop an Ant Colony Optimisation example using Unity. We will code all the elements in this example from scratch.

Please Read:

- This student activity will check that you have understood and can apply the knowledge you have gained during the talks / presentation.
- Please read each activity task carefully and when required please type out all code. Attempting to copy and paste code *may* cause errors during compilation. Also writing out the code helps you understand how to program C# and Unity.

Document Version:
V4.0.

Game Engine Version:
This student workbook was checked using **Unity 2022.-.--f1**.

1. Start the Unity Hub and Create a New Project

Start the **Unity Hub** by double clicking on the “Unity Hub” shortcut in the Windows start menu. The Unity Hub icon looks like the screenshot below.

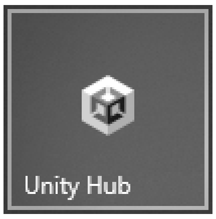


Image description: *The Unity hub Windows icon.*

In the Unity Hub, go to the Projects tab and click the (blue) New Project button

The Unity Hub window should look like the screenshot below. The blue New Project button is on the top right of the window.

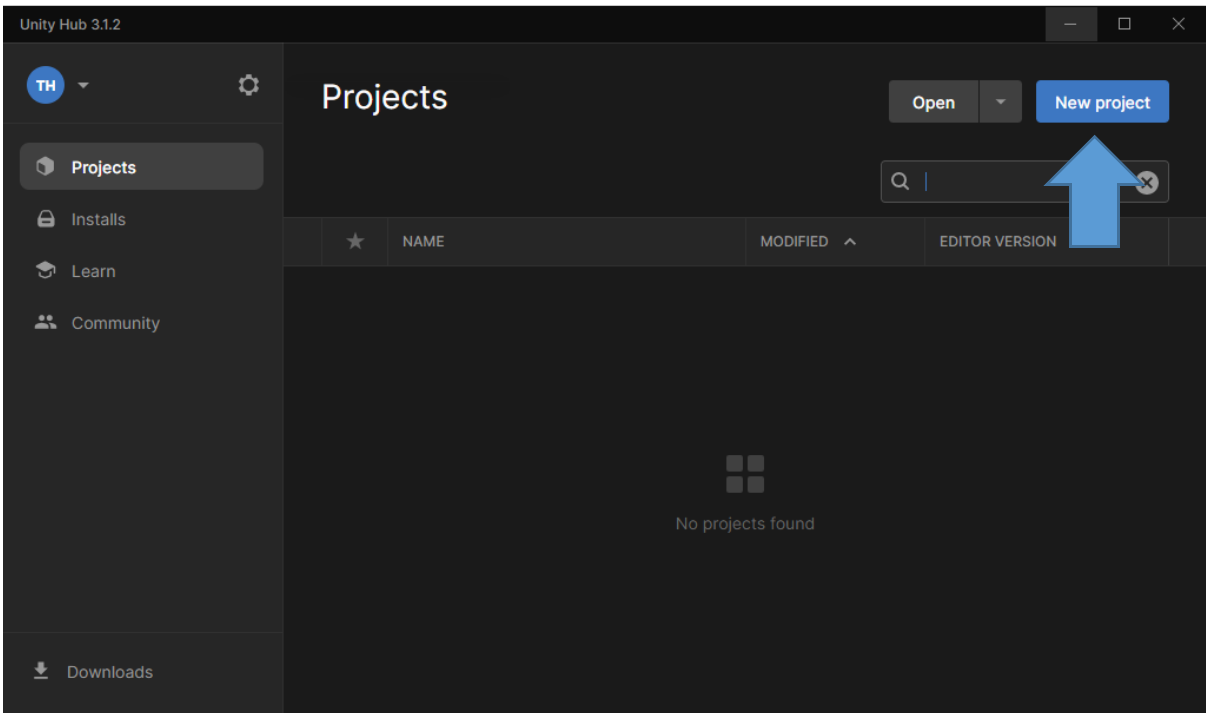
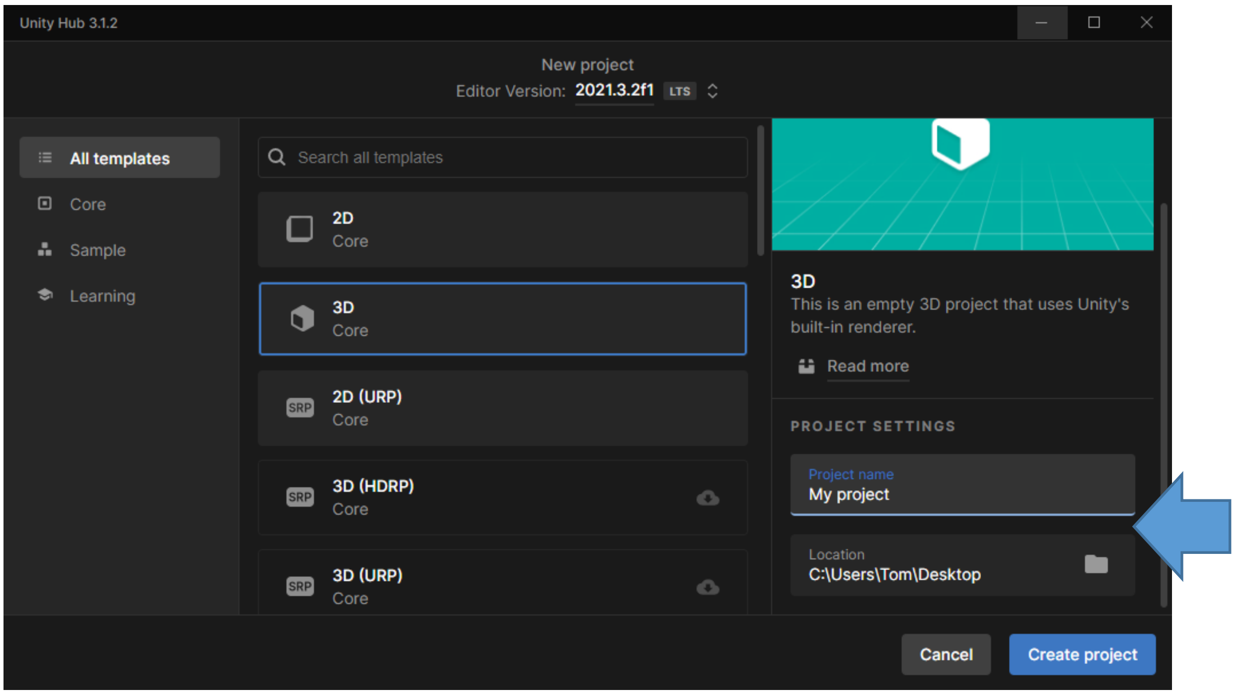


Image description: *The Unity hub with an arrow pointing to the New Project button.*

When you click the New Project button a "New project" screen will appear.

The window looks like the screenshot below.



3D
This is an empty 3D project that uses Unity's built-in renderer.

Project name
My project

Location
C:\Users\Tom\Desktop

Image description: *The Unity hub "New project" screen.*

Set the following settings in the Unity hub "New project" screen.

Template:	Select the 3D Core template. This is a basic empty 3D template.
Project Name:	Give your project the name **ACO_Example**.
Location:	Select a location to store your project. Below is some additional information about where to store your projects.

Useful Project Save Location Information:

- Any save location should be fine if you have enough disk space. Your will probably find project load and compile times are quicker if you save your projects to your computer's hard drive.
- **Don't forget to back-up your Unity projects.** I would recommend keeping at least one USB memory stick or USB hard drive for backup of all your university work. It would be better if you could maintain two backup devices.

Make sure you are creating a project in the correct version of the game engine. You can see the game engine version next to the "Editor Version:" text at the top of the Unity hub "New project" screen. **See the title page of this document for the version of the game engine we are using.**

We are now ready to create the project. <u>Click the "Create project" button.</u>

2. Importing Assets into your Project

We will now import some assets that we will use in this workbook.

Unity has an Asset Store that is home to thousands of free and priced assets. We will use **free** assets on the Unity Asset Store to help us create games or interactive 3D applications.

We are going to add the following free assets to our project:
- POLYGON Starter Pack - Low Poly 3D Art by Synty
- Starter Assets - First Person Character Controller

For quickest import, add the assets in the order above.

Before you can add the assets to your Unity project you need to first add them to your Unity account.

Step 1: Adding Assets to Your Unity Account

If needed, add the Unity assets to your Unity account. See the "Introduction to Unity" workbook for detailed guidance on how to add assets to your Unity account via the Unity asset store.

In a web browser, go to the web site: https://assetstore.unity.com/

Log into your Unity account. If you do not have a Unity account, you need to create one.

Search for the each of the assets listed above, select the asset and click the "Add to My Assets" button (it is a big blue button near the top right of the web page). Make sure you have logged into your Unity account at this point

When you have added the assets to your account, you can close the web browser.

Step 2: Importing Assets into Your Unity Project

In Unity you can import assets that have been added to your account. If you have not added the assets to your account, you need to add them before you proceed. See the previous step for more information.

In the Unity Editor, go to the Window menu and select Package Manager.

Select "My Assets" from the drop-down menu.

You may need to sign in with your Unity account to view your assets. If you are not signed-in, you should have the option to sign-in on the left panel of the Package Manager. Sign-in if needed.

A list of all your assets should appear.

Import all the assets asset pack(s) outlined above.

Follow these steps to import each asset pack:

- To import an asset pack, select it in the list.
- Next, you may need to click the **Download** button to download the asset to your computer.
- When the assets have been downloaded an Import button will appear. **Click the <u>import</u> button to import the asset.**
- **When you click the Import button a small "Import Unity Package" window will appear.**
- **Simply, click the <u>Import</u> button on the "Import Unity Package" window.**
- When you click the Import button the assets will be added to your project.

When you have imported all the assets, close the Package Manager window.

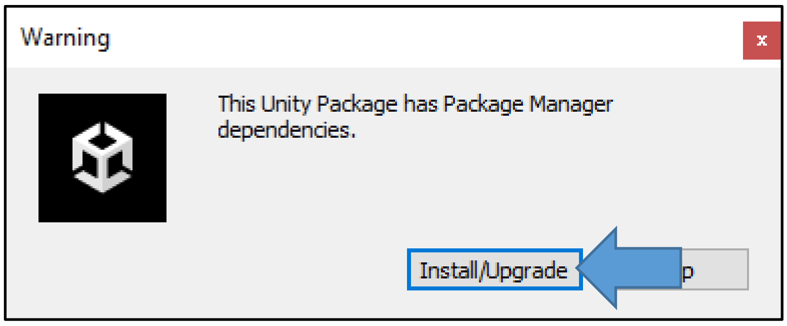
<u>Step 2.1: Importing - Starter Assets - First Person Character Controller</u>

Remember, the **<u>First Person Character Controller</u>** has some additional import steps.

Select the **Starter Assets - First Person Character Controller** from the list. If you have a lot of assets, you may need to click the **Load Next** button at the bottom of the assets list.

When you click the Import button, you will see this message.

<u>Click the "Install/Upgrade" button.</u> See the screenshot below for an example.

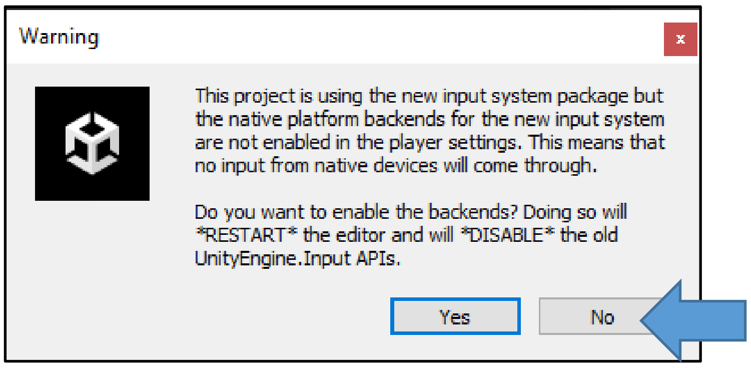

Warning

This Unity Package has Package Manager dependencies.

Image description: *When you click the Import button, you will see this message. Click the "Install/Upgrade" button.*

Wait for the package dependencies to be installed / upgraded.

Next, the message in the screenshot below will be displayed. <u>Click No</u>.


Warning

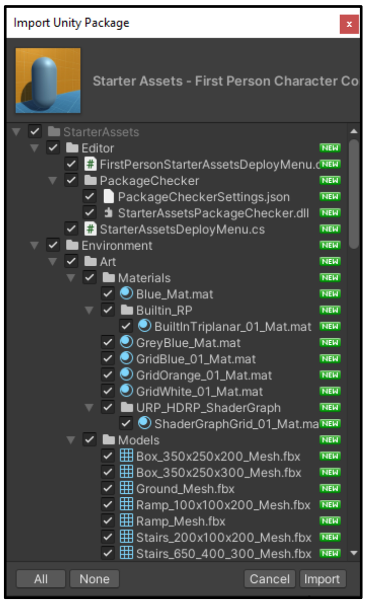
This project is using the new input system package but the native platform backends for the new input system are not enabled in the player settings. This means that no input from native devices will come through.

Do you want to enable the backends? Doing so will *RESTART* the editor and will *DISABLE* the old UnityEngine.Input APIs.

Image description: *A warning message stating that the assets being imported use the new input system package. Click no.*

Wait while more importing happens.

Then, a small "Import Unity Package" window will appear. See the screenshot below for an example.


Import Unity Package

Starter Assets - First Person Character Co

```
StarterAssets
  Editor                                        NEW
    FirstPersonStarterAssetsDeployMenu.c        NEW
    PackageChecker                              NEW
      PackageCheckerSettings.json               NEW
      StarterAssetsPackageChecker.dll           NEW
    StarterAssetsDeployMenu.cs                  NEW
  Environment                                   NEW
    Art                                         NEW
      Materials                                 NEW
        Blue_Mat.mat                            NEW
        Builtin_RP                              NEW
          BuiltInTriplanar_01_Mat.mat           NEW
        GreyBlue_Mat.mat                        NEW
        GridBlue_01_Mat.mat                     NEW
        GridOrange_01_Mat.mat                   NEW
        GridWhite_01_Mat.mat                    NEW
        URP_HDRP_ShaderGraph                    NEW
          ShaderGraphGrid_01_Mat.ma             NEW
      Models                                    NEW
        Box_350x250x200_Mesh.fbx                NEW
        Box_350x250x300_Mesh.fbx                NEW
        Ground_Mesh.fbx                         NEW
        Ramp_100x100x200_Mesh.fbx               NEW
        Ramp_Mesh.fbx                           NEW
        Stairs_200x100x200_Mesh.fbx             NEW
        Stairs_650_400_300_Mesh.fbx             NEW
```

Image description: *The "Import Unity Package" window. Click Import.*

Simply, click the <u>Import</u> button on the "Import Unity Package" window.

When you click the Import button the assets will be added to your project. Wait while this happens.

<u>When everything has finished importing you should close the Package Manager.</u>

In the Unity Editor, go to the Edit drop down menu and select <u>Project Settings</u>. See the screenshot below for an example.

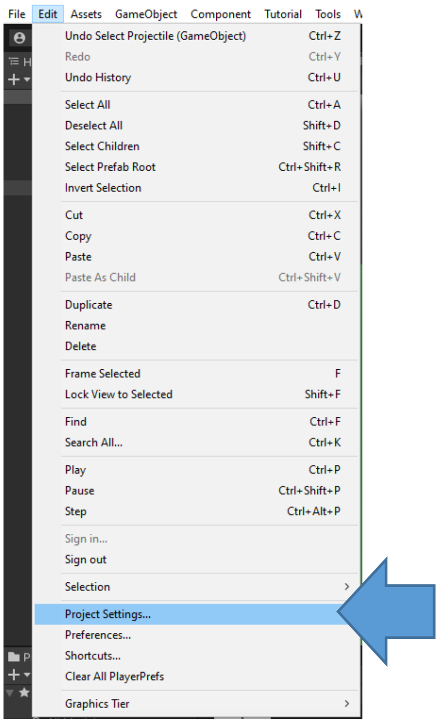
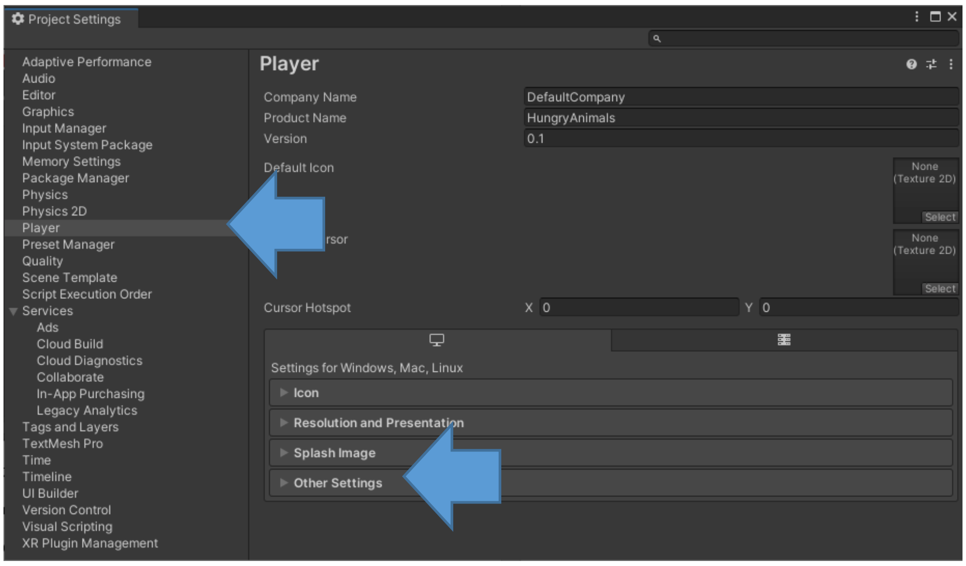


Image description: *Open the Project Settings window to set Unity project input type.*

Select Player from the list on the left and expand the "Other Settings" option. See the screenshot below for an example.

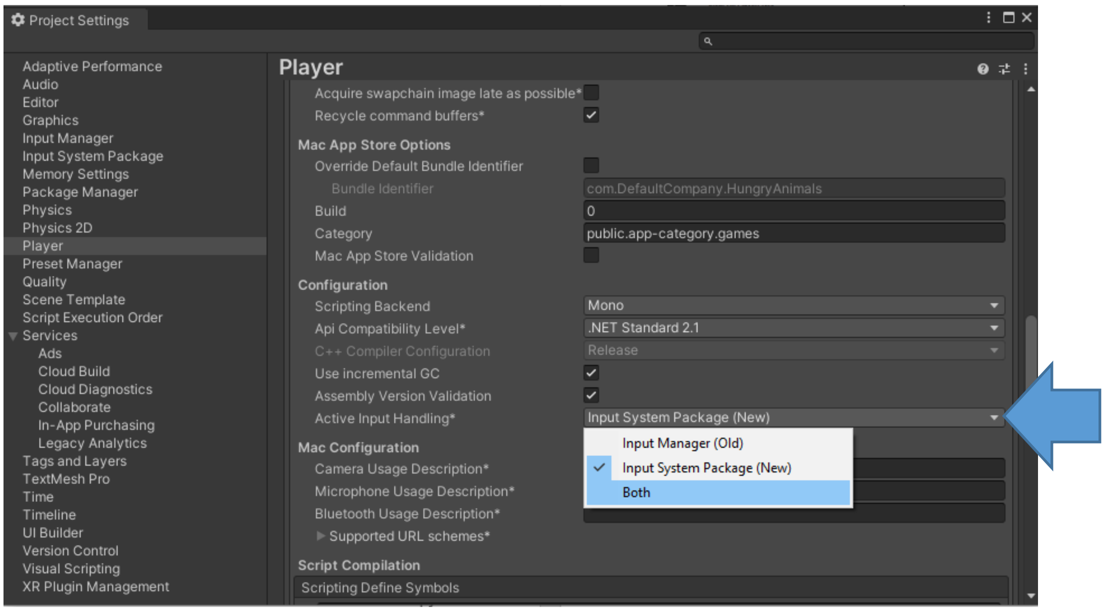


Company Name DefaultCompany
Product Name HungryAnimals
Version 0.1
Cursor Hotspot X 0 Y 0

Image description: *Select Player from the list on the left and expand the "Other Settings" option.*

Scroll down "Other Settings" until you find the "Active Input Handling" option. Click the drop-down menu and select <u>Both</u>. See the screenshot below for an example.

Note: there are two input systems in Unity. An old one and a new one. In our workbooks we will use both because the old input system is very quick to get up and running.



Player

Acquire swapchain image late as possible*
Recycle command buffers* ✓

Mac App Store Options
Override Default Bundle Identifier
Bundle Identifier: com.DefaultCompany.HungryAnimals
Build: 0
Category: public.app-category.games
Mac App Store Validation

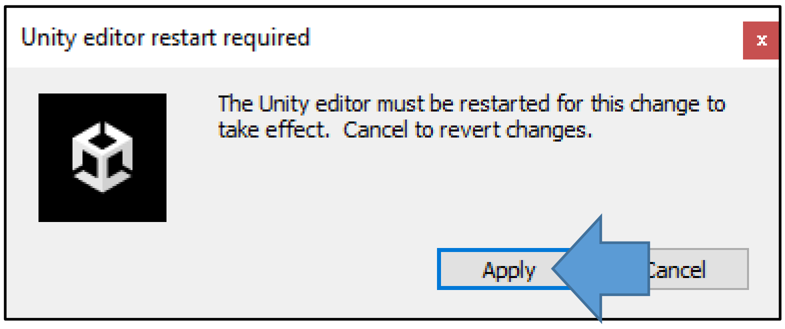
Configuration
Scripting Backend: Mono
Api Compatibility Level*: .NET Standard 2.1
C++ Compiler Configuration: Release
Use incremental GC ✓
Assembly Version Validation ✓
Active Input Handling*: Input System Package (New)
- Input Manager (Old)
- ✓ Input System Package (New)
- Both

Mac Configuration
Camera Usage Description*
Microphone Usage Description*
Bluetooth Usage Description*
Supported URL schemes*

Script Compilation
Scripting Define Symbols

Image description: *Scroll down "Other Settings" until you find the "Active Input Handling" option. Click the drop-down menu and select* *<u>Both</u>*.

A "Unity editor restart required" message box will apear. Click Apply.


Unity editor restart required

The Unity editor must be restarted for this change to take effect. Cancel to revert changes.

Image description: *A warning message stating that Unity must restart. <u>Click Apply</u>.*

Note, at this point Unity will restart. This is fine. If needed, click Save to save your scene. <u>Unity will now restart.</u>

When Unity restarts, close the "Project Settings" window (if needed).

When Unity has restarted we can continue to import asset packs (if needed).

3. Set External Script Editor and Regenerate Project files (*If needed*)

When our Unity projects include code, we might need to set the external script editor to Visual Studio and click the Regenerate Project files button.

This is important if you are opening a Unity project on a computer for the first time and it contains code. Regenerating project files ensures that the Visual Studio intellisense will work properly.

Therefore, it is a good idea to check the external script editor is set to Visual Studio and click the Regenerate Project files button every-time you load your Unity project on a new computer.

Tip: At home you probably do not need to do this if you are using the same computer all the time. Things should just work. You only need to do this if you are opening your Unity project on a new computer.

Follow the steps below to set Visual Studio as the external script editor and regenerate project files.

In the Unity Editor, click the **Edit** drop-down menu, and select **Preferences.** It is the 5th option from the bottom of the menu. See the screenshot below for an example.

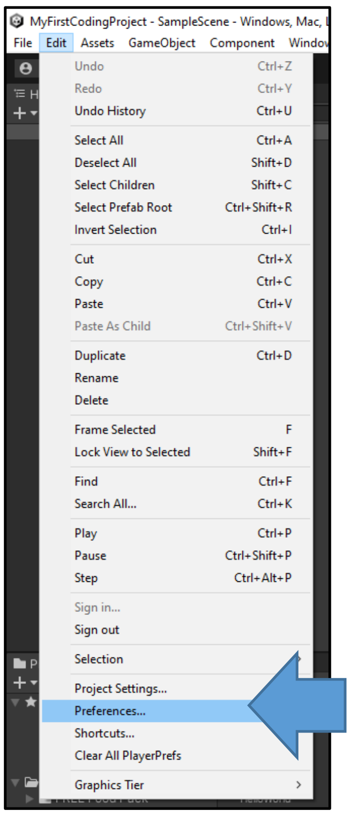


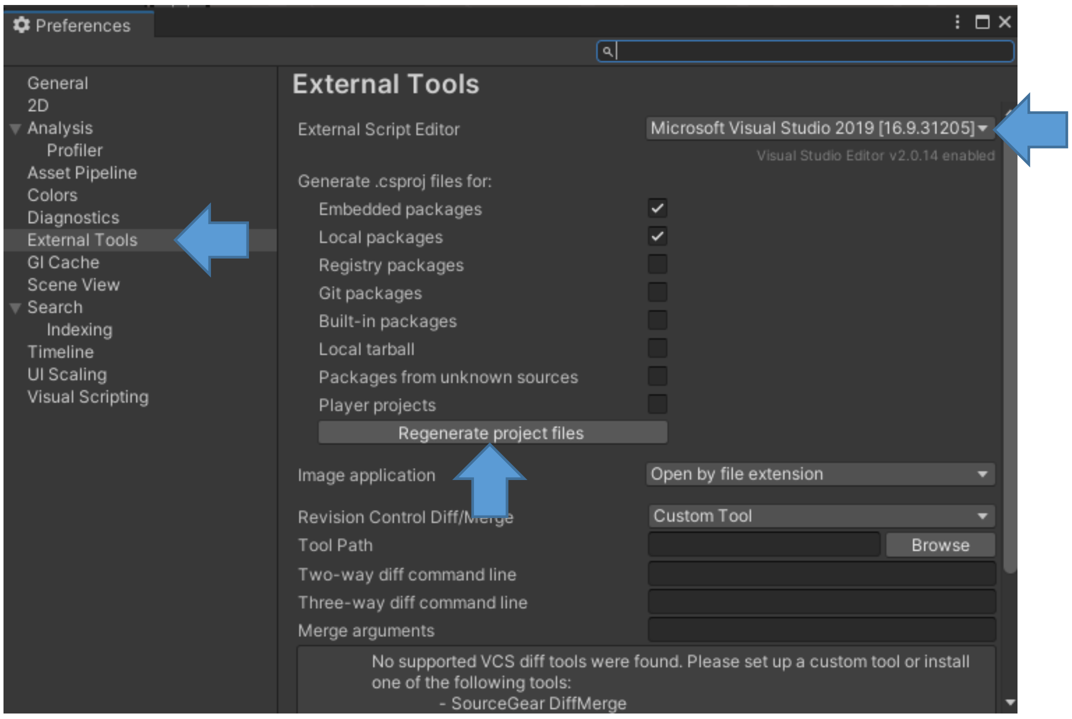
Image description: *Open the Preferences window to set Visual Studio as the external script editor and regenerate project files.*

A preferences window will load.

Select **External Tools** from the list on the left.

Then for the **External Script Editor** option and select the correct version of **Visual Studio**. If Visual Studio is already set, you do not need to do anything.

Then click the **Regenerate Project files** button.



External Tools

External Script Editor: Microsoft Visual Studio 2019 [16.9.31205]
Visual Studio Editor v2.0.14 enabled

Generate .csproj files for:
- Embedded packages ✓
- Local packages ✓
- Registry packages
- Git packages
- Built-in packages
- Local tarball
- Packages from unknown sources
- Player projects

Image application: Open by file extension
Revision Control Diff/Merge: Custom Tool
Tool Path
Two-way diff command line
Three-way diff command line
Merge arguments

No supported VCS diff tools were found. Please set up a custom tool or install one of the following tools:
- SourceGear DiffMerge

Image description: *The Preferences window. If needed, set the external script editor to Visual Studio. Then, click the regenerate project files button.*

You can now close the preferences window.

4. Creating a Simple Scene with Primitive Shapes

In this section we will create a simple scene. This illustrates how you can create a simple scene in Unity using primitive shapes. This is an alternative to terrains or could be used in conjunction with terrains.

The first thing we will do is create a simple ground or floor to walk on.

Go to the Hierarchy Window and click the plus (+) menu. Select 3D Object -> Cube. See the screenshot below for an example.

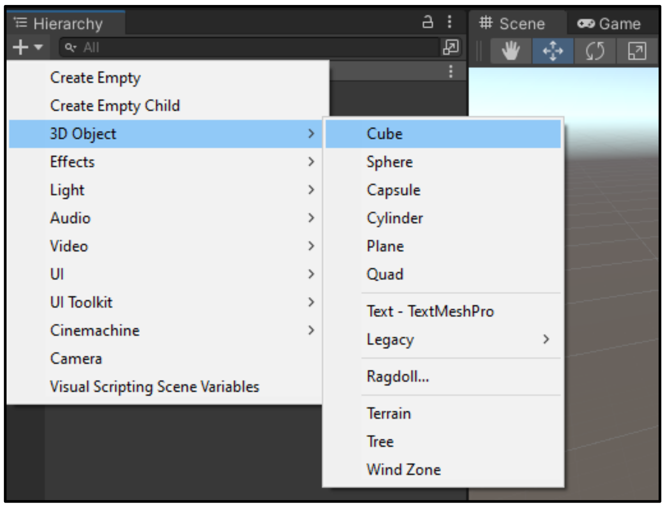


Image description: *The Hierarchy Window and the plus (+) menu. Cube has been selected in the menu.*

A cube should be created in your scene.

At this point the cube should be selected in the hierarchy window and you should be able to enter a name for the cube. Give the cube the name **Ground**.

- **Tip:** If the cube is not selected in the hierarchy window. For example, you might have clicked on something else. Go to the Hierarchy Window and slowly single click twice on the Cube text to rename it. Or you can right click and select Rename.

Select the ground cube in the hierarchy window.

Go to the inspector. Scale the cube so that it looks more like a floor. I used the following settings:

X:	100
Y:	1
Z:	100

This creates a relatively small environment. You can use larger values for the x and z axes to create a bigger environment.

Your cube should now look like the screenshot below.

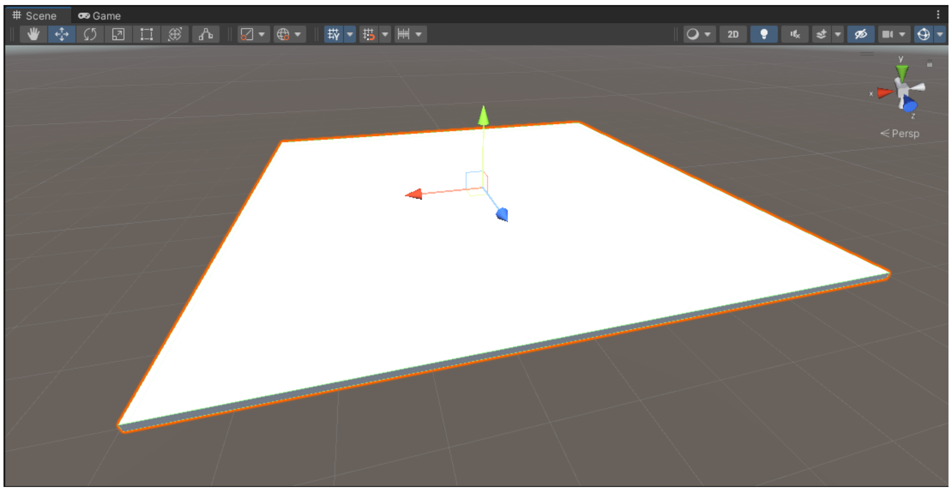


Image description: *The ground cube in the scene view.*

Next, we will give the ground some colour using a material.

Go to the Project Window. Select the Assets folder. Right click in the righthand panel of the Assets folder and select Create -> Folder.

When the folder is created give it the name **Materials**.

- **Tip:** It is a good idea to organise your assets in folders. You can make the folder structure simple or complicated. You should do what works for your project.

Double click on the Materials folder to open it.

Right click in the Materials folder and go to Create -> Material.

Give the Material the name **GrassColour**.

Select the material in the Project window. You should be able to see its properties in the inspector.

Go to the inspector, click on the block next to the Albedo property at top of the Inspector.

- **Information:** The Albedo property, pronounced, al-bee-dow, is the colour or texture map for the object.

See the screenshot below for an example.

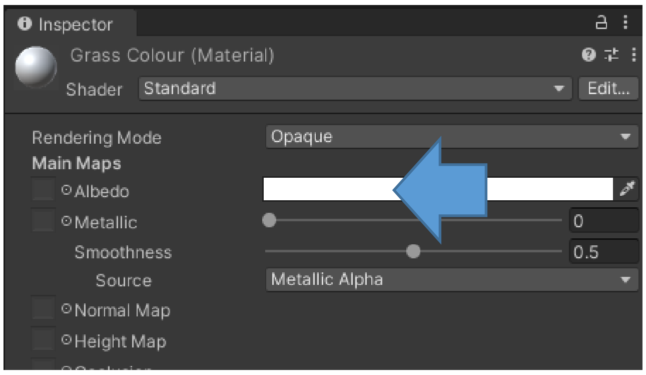


Image description: *The inspector window for the material. Click on the white box next to the Albedo property.*

Clicking on the block next to the Albedo property at top of the Inspector will open a colour picker window.

Select a green colour. I entered the hexadecimal value: 228C22.

See the screenshot below for an example.

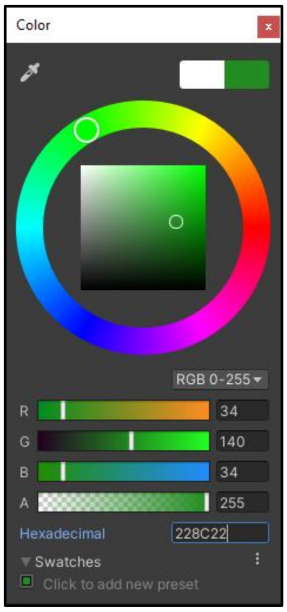


Image description: *The material Albedo property colour picker.*
Close the colour picker window.

Next, we need to assign the material to the ground cube in the scene. To do this, drag the **GrassColour** material from the Project window and drop it onto the name of the ground object in the Hierarchy OR drag it onto the ground in the scene view.

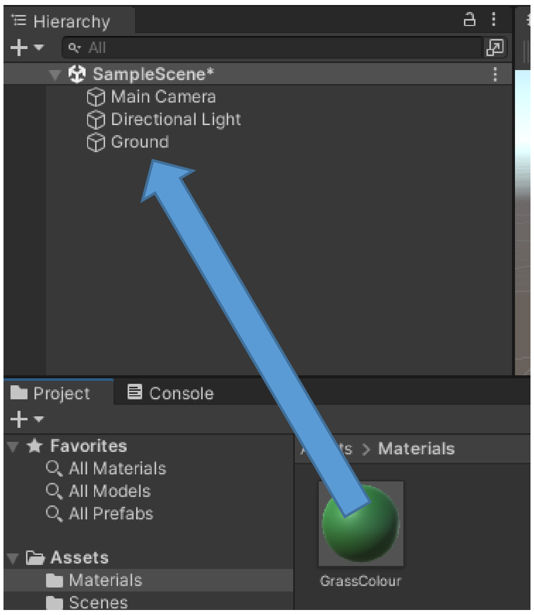


Image description: *Drag the material from the Project window to the ground GameObject.*

Your updated ground should now look like it has a green grass colour applied to it. See the screenshot below for an example.

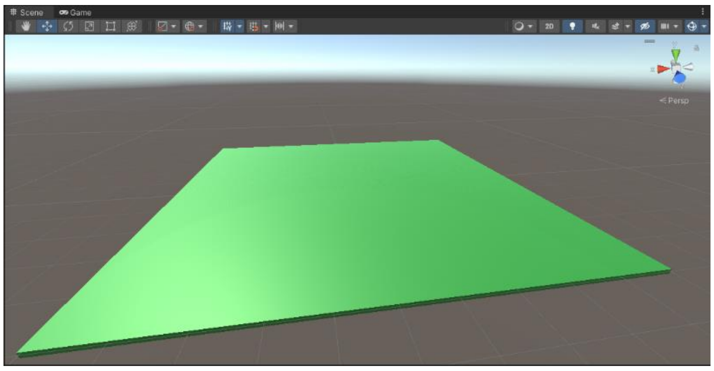


Image description: *The ground cube in the scene view. A green material has been applied to it.*

Save your scene. Go to the File menu and find the save option.

5. Adding a First-Person Camera to the Environment

We will now add a first-person camera to our scene. This camera will allow a player or user to move around our scene.

By default, Unity includes a camera in our scene. Go to the hierarchy, find the Main camera, right click on it, and select delete from the menu. See the screenshot below for an example.

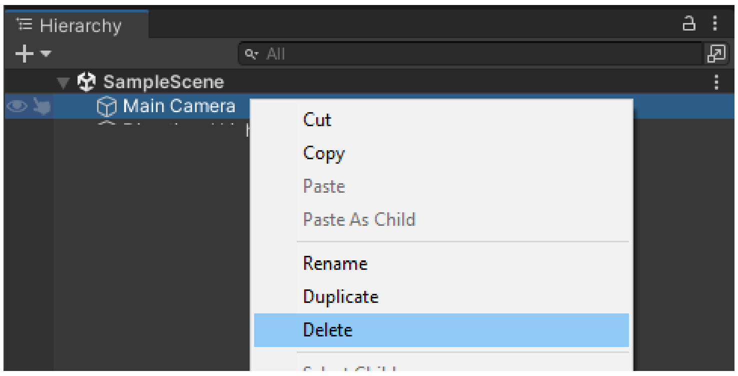


Image description: *Example of how to delete the "Main Camera" from the scene.*

Once you have deleted the main camera from the scene you can add a first-person camera.

Go to the Project window.

Navigate to the folder: StarterAssets -> FirstPersonController -> Prefabs.

<u>You should find that some of the assets in this folder are coloured magenta (pink). This is because they are setup to use the universal render pipeline, instead of the built-in render pipeline, which this project is using.</u>

First, we need to change the **<u>PlayerCapsule</u> prefab** so that is used the built-in render pipeline.

<u>Double click</u> on the **<u>PlayerCapsule</u> prefab**. This will open the prefab editor. See the screenshot below for an example.

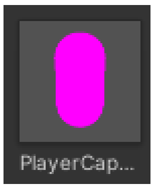


Image description: *some of the assets in the StarterAssets -> FirstPersonController -> Prefabs folder are coloured magenta (pink). This is because they are setup to use the universal render pipeline, instead of the built-in render pipeline, which this project is using. Double click on the PlayerCapsule prefab.* This will open the prefab editor.

The prefab editor should open. See the screenshot below for an example.

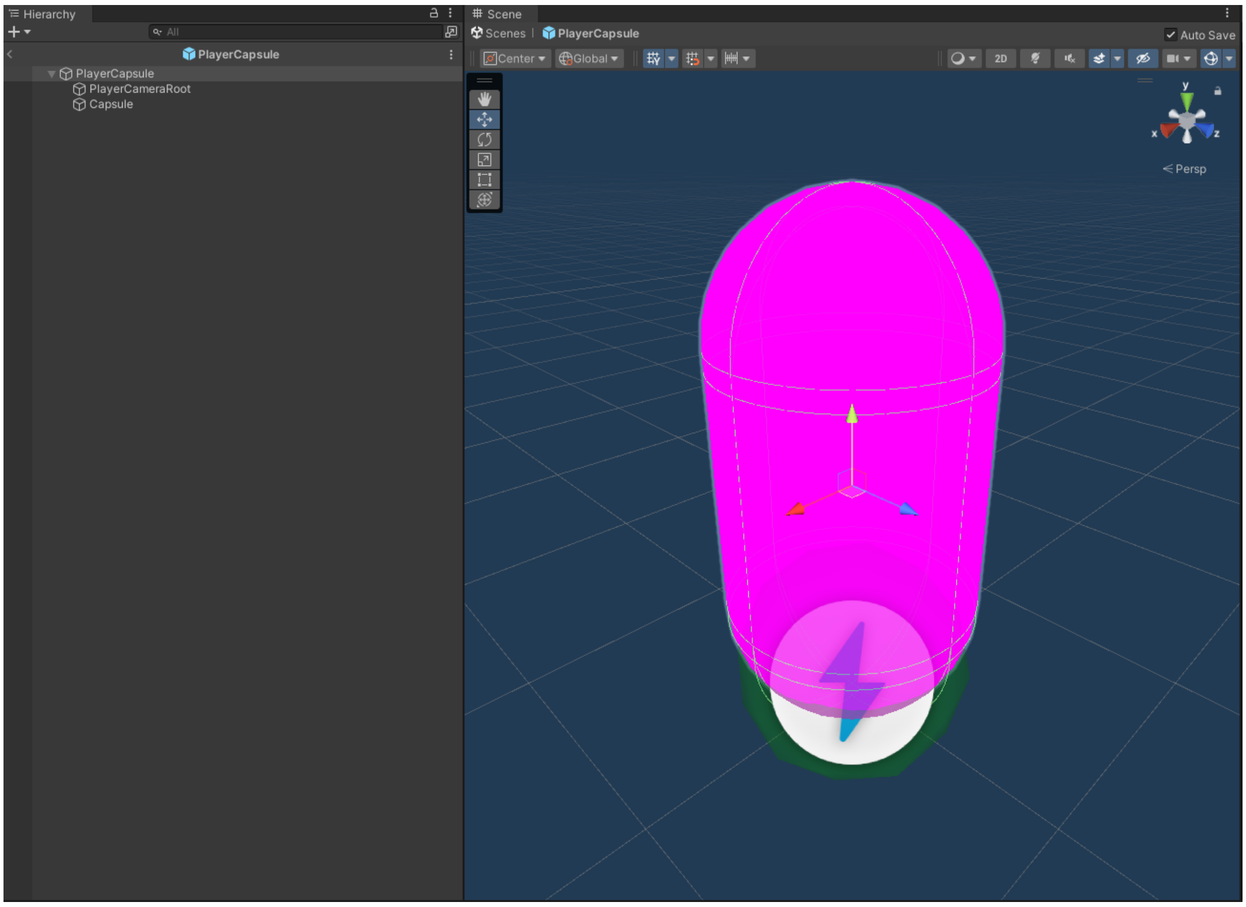


Image description: *The prefab editor with the PlayerCapsule prefab loaded.*

Click the Capsule gameObject in the Hierarchy.

Go to the Inspector and find the material for the game object. See the screenshot below for an example.

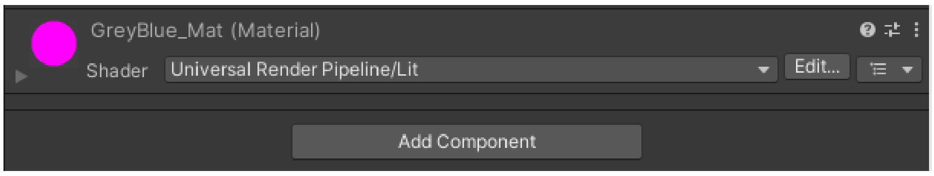

GreyBlue_Mat (Material)
Shader Universal Render Pipeline/Lit

Image description: *Go to the Inspector and find the material for the game object.*

Click the drop-down menu next to the word "Shader". Select "<u>Standard</u>" from the drop-down menu.

The capsule should now **<u>not</u>** be coloured magenta (pink).

Click the back arrow near the top of the Hierarchy to move back to your scene. See the screenshot below for an example.

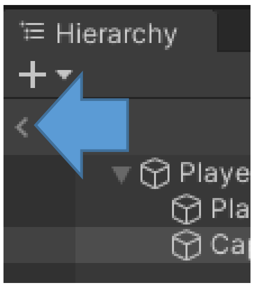


Image description: *Click the back arrow near the top of the Hierarchy to move back to your scene.*

Save your scene.

Drag the PlayerCapsule prefab into the scene and position it somewhere on the ground.

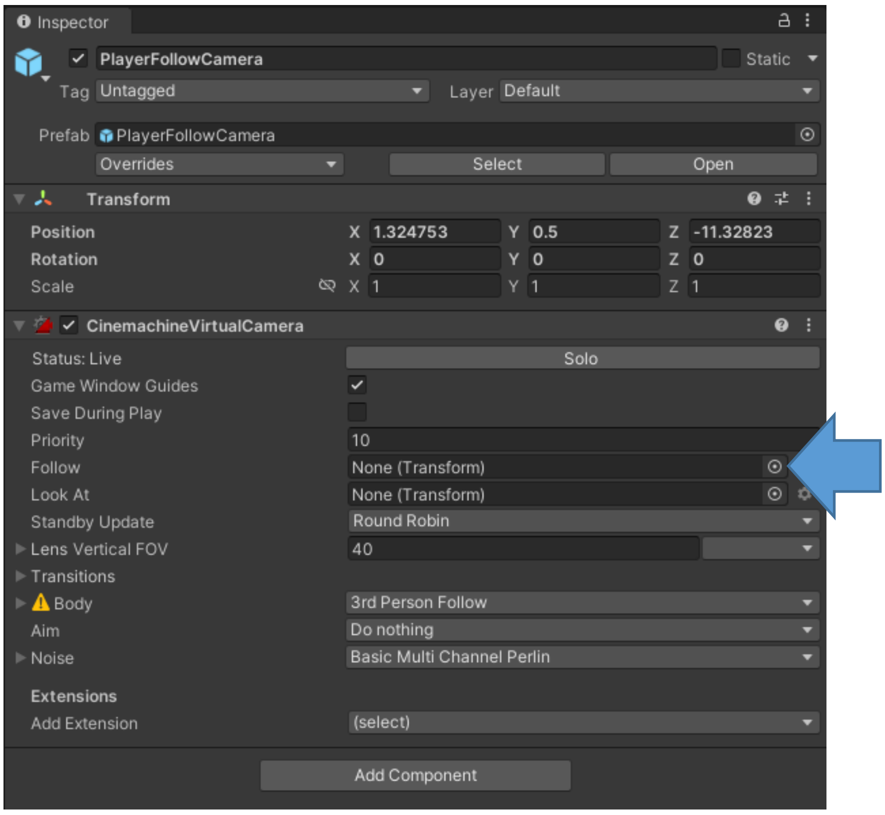
Drag the PlayerFollowCamera prefab into the scene. This prefab can be positioned anywhere; however, I would recommend you position it near the PlayerCapsule prefab.

Drag the MainCamera prefab into the scene. This prefab should automatically position itself at the PlayerFollowCamera prefab's position.

Next, go to the hierarchy, and select the PlayerFollowCamera prefab.

Go to the inspector and find the **CinemachineVirtualCamera** component.

Find the **Follow** property and click the bullet icon next to "None (Transform)". See the screenshot below for an example.


Inspector

PlayerFollowCamera Static
Tag Untagged Layer Default
Prefab PlayerFollowCamera
Overrides Select Open

Transform
Position X 1.324753 Y 0.5 Z -11.32823
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

CinemachineVirtualCamera
Status: Live Solo
Game Window Guides ✓
Save During Play
Priority 10
Follow None (Transform)
Look At None (Transform)
Standby Update Round Robin
Lens Vertical FOV 40
Transitions
Body 3rd Person Follow
Aim Do nothing
Noise Basic Multi Channel Perlin

Extensions
Add Extension (select)

Add Component

Image description: *Go to the hierarchy and select the PlayerFollowCamera prefab. Go to the inspector and find the CinemachineVirtualCamera component. Find the follow property and click the bullet icon next to “None (Transform)”.*

A “Select Transform” window will appear.

Double click on **<u>PlayerCameraRoot</u>** (in the Scene tab) to select it.

The follow property of the CinemachineVirtualCamera component should now have the value “PlayerCameraRoot (Transform)”.

<u>The first-person camera should now be setup.</u>

Save the scene.

🎮 **We are now ready to playtest the scene.**

Press the play button to playtest your scene. See the screenshot below for an example.

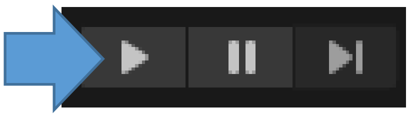


Image description: *The game / scene play controls on the main toolbar. Click the play button.*

Play Mode is a realistic test of your game.

- Note, when in Play mode you can adjust GameObjects via the Scene window. However, all adjustments made to GameObjects will be temporary and undone when play mode is stopped.

If the game view is behind the scenes view, click the Game view tab to select the Game view window.

You should be able to walk around the environment.

The default controls:

- Arrow keys or WASD to move. Spacebar to jump. Hold down shirt to sprint.
- Mouse look.

To stop playtesting the game, click the play button again.

See the screenshot below for an example. Note, in my setup I have the Scene and Game view side-by-side.

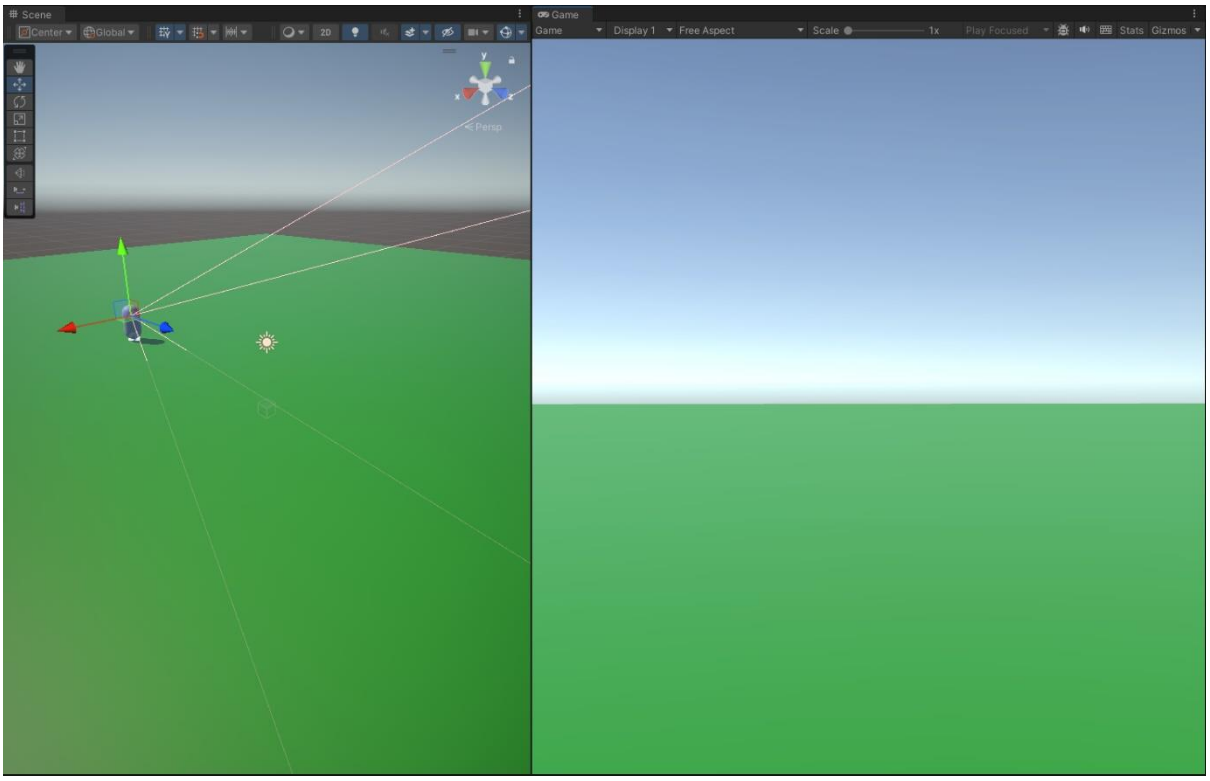


Image description: *A playtest of the scene using the first-person controller. Note, in my setup I have the Scene and Game view side-by-side.*

6. Creating a Simple Waypoint Graph – Visual Representation

Next, we will create some simple waypoints in Unity. We will do this in code. These waypoints will allow us to create a waypoint graph in the Unity editor and provide a visual representation of the waypoint.

We will then use the visual waypoints to create a waypoint graph in code. The code waypoint graph will be used by an Ant Colony Optimisation algorithm to generate a sequence of nodes to visit. The sequence will be the shortest route between all the nodes.

In this example each node will be accessible via a straight-line from every other node. However, note each straight-line connection could be adjusted to be a path of nodes generated by the A* algorithm.

We will start by creating a C# class that represents a connection in your visual representation of our waypoint graph. The class will only be used for the visual representation. We will create a connection class for our code representation of the waypoint graph later.

We will now create a C# script to represent a waypoint node.

Create a C# script. Go to the project window.

Select the Assets folder. Right click in the righthand panel of the Assets folder and select Create -> Folder.

When the folder is created give it the name **Scripts**.

Double click on the Scripts folder to open it.

Right click in the Scripts folder and go to Create -> C# Scripts.

When the script has been created give it the name **VisGraphConnection**.

Update the code in the script file to match the code below.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[System.Serializable]
public class VisGraphConnection
{
    // The to node for this connection.
    [SerializeField]
    private GameObject toNode;
    public GameObject ToNode
    {
        get { return toNode; }
    }
}
```

In the code above we add the [System.Serializable] attribute so that our VisGraphConnection class is visible in the inspector.

We then add one variable that stores the connection “to node”. This is the node the connection goes to. The from node will be the waypoint itself.

Save the script in Visual Studio and go back to Unity.

Next, we will create a C# script to represent a waypoint node.

Go to the project window.

Go to the Assets -> Scripts folder.

Right click in the Scripts folder and go to Create -> C# Scripts.

When the script has been created give it the name **VisGraphWaypointManager**.

Update the code in the script file to match the code below.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

// Used to display text above the node.
using UnityEditor;

public class VisGraphWaypointManager : MonoBehaviour
{
    // Allow you to set the waypoint text colour.
    [SerializeField]
    private enum waypointTextColour { Blue, Cyan, Yellow };
#pragma warning disable
    [SerializeField]
    private waypointTextColour WaypointTextColour = waypointTextColour.Blue;
#pragma warning restore

    // List of all connections from this node.
    [SerializeField]
    public List<VisGraphConnection> connections = new List<VisGraphConnection>();
    public List<VisGraphConnection> Connections
    {
        get { return connections; }
    }

    // Allow you to set a waypoint as a start or goal.
    public enum waypointPropsList { Standard, Start, Goal };
#pragma warning disable
    [SerializeField]
    private waypointPropsList waypointType = waypointPropsList.Standard;
#pragma warning restore
    public waypointPropsList WaypointType
    {
        get { return waypointType; }
    }

    // Controls if the node type is displayed in the Unity editor.
    private const bool displayType = false;

    // Used to determine if the waypoint is selected.
    private bool ObjectSelected = false;

    // Text displayed above the node.
    private const bool displayText = true;
    private string infoText = "";
    private Color infoTextColor;

    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
    }
```

```
// Draws debug objects in the editor and during editor play (if option set).
void OnDrawGizmos()
{
    // Text displayed above the waypoint.
    infoText = "";
    if (displayType)
    {
        #pragma warning disable
        infoText = "Type: " + WaypointType.ToString() + " / ";
        #pragma warning restore
    }
    infoText += gameObject.name + "\n Connections: " + Connections.Count;

    switch (WaypointTextColour)
    {
        case waypointTextColour.Blue:
            infoTextColor = Color.blue;
            break;
        case waypointTextColour.Cyan:
            infoTextColor = Color.cyan;
            break;
        case waypointTextColour.Yellow:
            infoTextColor = Color.yellow;
            break;
    }

    DrawWaypointAndConnections(ObjectSelected);

    if (displayText)
    {
        GUIStyle style = new GUIStyle();
        style.normal.textColor = infoTextColor;
        Handles.Label(transform.position + Vector3.up * 1, infoText, style);
    }
    ObjectSelected = false;
}

// Draws debug objects when an object is selected.
void OnDrawGizmosSelected()
{
    ObjectSelected = true;
}
```

```csharp
// Draws debug objects for the waypoint and connections.
private void DrawWaypointAndConnections(bool ObjectSelected)
{
    Color WaypointColor = Color.yellow;
    Color ArrowHeadColor = Color.blue;
    if (ObjectSelected)
    {
        WaypointColor = Color.red;
        ArrowHeadColor = Color.magenta;
    }

    // Draw a yellow sphere at the transform's position
    Gizmos.color = WaypointColor;
    Gizmos.DrawSphere(transform.position, 0.2f);

    // Draw all the connections.
    for (int i = 0; i < Connections.Count; i++)
    {
        if (Connections[i].ToNode != null)
        {
            if(Connections[i].ToNode.Equals(gameObject))
            {
                infoText = "WARNING - Connection to SELF at element: " + i;
                infoTextColor = Color.red;
            }

            Vector3 direction = Connections[i].ToNode.transform.position - transform.position;
            DrawConnection(i, transform.position, direction, ArrowHeadColor);

            if(ObjectSelected)
            {
                // Draw spheres along the line.
                Gizmos.color = ArrowHeadColor;

                float dist = direction.magnitude;
                float pos = dist * 0.1f;
                Gizmos.DrawSphere(transform.position +
                    (direction.normalized * pos), 0.3f);
                pos = dist * 0.2f;
                Gizmos.DrawSphere(transform.position +
                    (direction.normalized * pos), 0.3f);
                pos = dist * 0.3f;
                Gizmos.DrawSphere(transform.position +
                    (direction.normalized * pos), 0.3f);
            }
        }
        else
        {
            infoText = "WARNING - Connection is missing at element: " + i;
            infoTextColor = Color.red;
        }
    }
}
```

```
    // This arrow method is based on the example here: https://gist.github.com/MatthewMaker/5293052
    public void DrawConnection(float ConnectionsIndex, Vector3 pos, Vector3 direction,
        Color ArrowHeadColor, float arrowHeadLength = 0.5f, float arrowHeadAngle = 40.0f)
    {
        Debug.DrawRay(pos, direction, Color.blue);

        Vector3 right = Quaternion.LookRotation(direction) *
            Quaternion.Euler(0, 180 + arrowHeadAngle, 0) * new Vector3(0, 0, 1);

        Vector3 left = Quaternion.LookRotation(direction) *
            Quaternion.Euler(0, 180 - arrowHeadAngle, 0) * new Vector3(0, 0, 1);

        Debug.DrawRay(pos + direction.normalized +
            (direction.normalized * (0.1f * ConnectionsIndex)),
            right * arrowHeadLength, ArrowHeadColor);
        Debug.DrawRay(pos + direction.normalized +
            (direction.normalized * (0.1f * ConnectionsIndex)),
            left * arrowHeadLength, ArrowHeadColor);

    }
}
```

In the code above we include a class level variable to store a list of connections (VisGraphConnection objects) for the waypoint node. Remember, this class is designed to help use visually create a waypoint graph. We create a code representation of the waypoint graph used by the A* algorithm later.

There is also a class level variable to set the waypoint type in the editor. This might be useful if you wan to set waypoint nodes as goals or starts.

The rest of the code in the class is designed to display the position of the waypoint node and the connections for the node in the Unity editor or during editor play (if the option is set).

The code draws a line between nodes if there is a connection. It also draws an arrow hear near a node indicating there is a connection in a direction. The code also displays the number of connections above a node.

There are three helper visuals to help you determine if you have issues with a node. Firstly, the text above the node will display a warning if you have added a connection element, but not included a connection. It will warn you about the last missing connection is the list. The text above the node will also display a warning if you have added a connection to itself (e.g., a connection to the current waypoint node). It will warn you about the last connection to itself connection is the list.

Secondly, a node will add an arrowhead for each connection. Therefore, if you see multiple arrow heads on a single connection is means you have connected to another waypoint node twice. This might be intentional; however, it is useful to for you to know if this has happened.

Save the script in Visual Studio and go back to Unity.

Next, we need to attach the <u>VisGraphWaypointManager</u> script to an empty game object and create a prefab that we can use to create waypoint nodes.

Go to the Hierarchy Window and click the plus (+) menu. Select 3D Object -> Create Empty. See the screenshot below for an example.

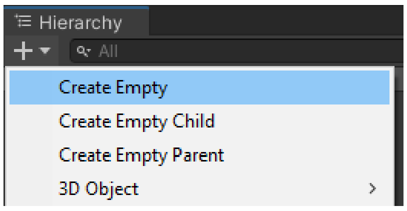
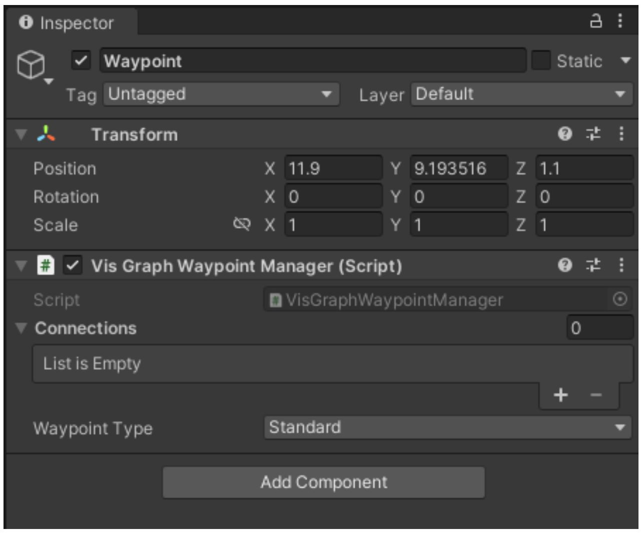


Image description: *The Hierarchy Window and the plus (+) menu. Create Empty has been selected in the menu.*

Call the empty GameObject **Waypoint**.

Make sure the empty **GameObject** called Waypoint is selected in the hierarchy.

Go to the inspector and click the **add component** button. Search for the **<u>VisGraphWaypointManager</u>** script. Select the **<u>VisGraphWaypointManager</u>** script from the list to add the component. See the screenshot below for an example.



Inspector
Waypoint — Static
Tag: Untagged — Layer: Default

Transform
Position: X 11.9, Y 9.193516, Z 1.1
Rotation: X 0, Y 0, Z 0
Scale: X 1, Y 1, Z 1

Vis Graph Waypoint Manager (Script)
Script: VisGraphWaypointManager
Connections: 0
List is Empty
Waypoint Type: Standard

Add Component

Image description: *The inspector window for the empty GameObject called Waypoint. The VisGraphWaypointManager script has been added as a component.*

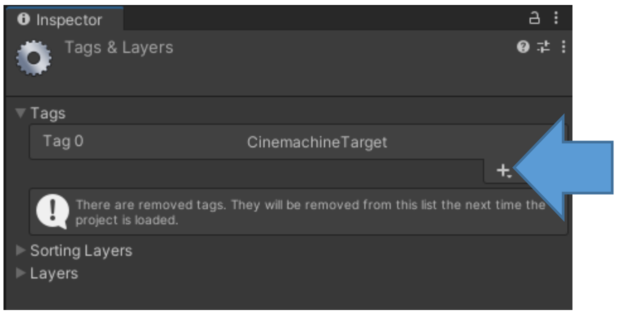
<u>Next, we will give the waypoint a tag of Waypoint.</u>

Make sure the empty GameObject called Waypoint is still selected in the hierarchy.

Go to the Inspector. Click the tag dropdown menu and select “Add Tag…”.

The inspector will change to a “Tags & Layers” window.

Click the plus (+) icon in the Tags section. See the screenshot below for an example.



Inspector
Tags & Layers

Tags
Tag 0 — CinemachineTarget

There are removed tags. They will be removed from this list the next time the project is loaded.

Sorting Layers
Layers

Image description: *When you click the “Add Tag…” option the Inspector will change to a “Tags & Layers” list. Click the plus (+) icon in the Tags section.*

Enter the name **Waypoint** and **click save**.

“Tags & Layers” window should now have the **Waypoint** tag in it.

Select the empty GameObject called Waypoint again in the Hierarchy.

The inspector should now have the empty GameObject called Waypoint properties again.

Go to the Inspector. Click the tag dropdown menu and select the **Waypoint** tag.

We want the waypoint to be stored in the asset folder and instantiated by a virtual world designer when they create the waypoint map. Therefore, we need to store the object as a prefab.

Go to the Project Window. Select the Assets folder. Right click in the righthand panel of the Assets folder and select Create -> Folder.

When the folder is created give it the name **Prefabs**.

Next, drag the empty GameObject called Waypoint from the hierarchy and drop it into the Prefabs folder in the project window.

Save the scene. Go to the File menu and find the save option.

See the screenshot below for an example.

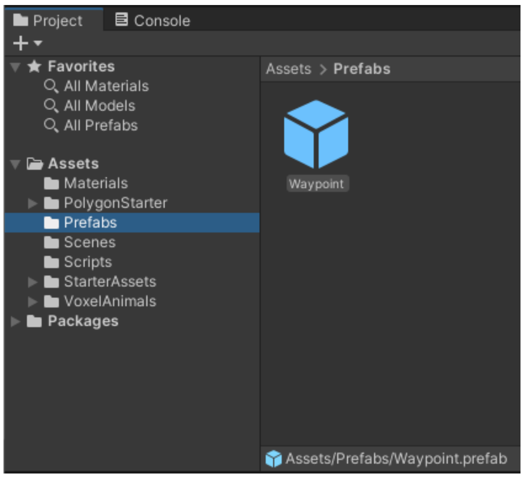


Image description: *The project window. The Prefabs folder has been opened. The empty GameObject called Waypoint has been dragged into the Prefabs folder. A prefab has been created and stored in the Prefabs folder.*

7. Adding the Waypoint Nodes to our Level and Setting Properties

Next, we will add the nodes to our level and set the connections for each node. We will visually create a waypoint graph in a Unity scene.

Waypoint graph nodes have connections that identify the links between each graph node. We will add waypoint nodes and set the connections

In general, you need to add 10 nodes to your level and set the connections so that that <u>every</u> node <u>connects to every other node</u> in your waypoint graph. Your waypoint graph should look like the screenshot below. As you can see, every waypoint node in the graph below connects to every other waypoint node.

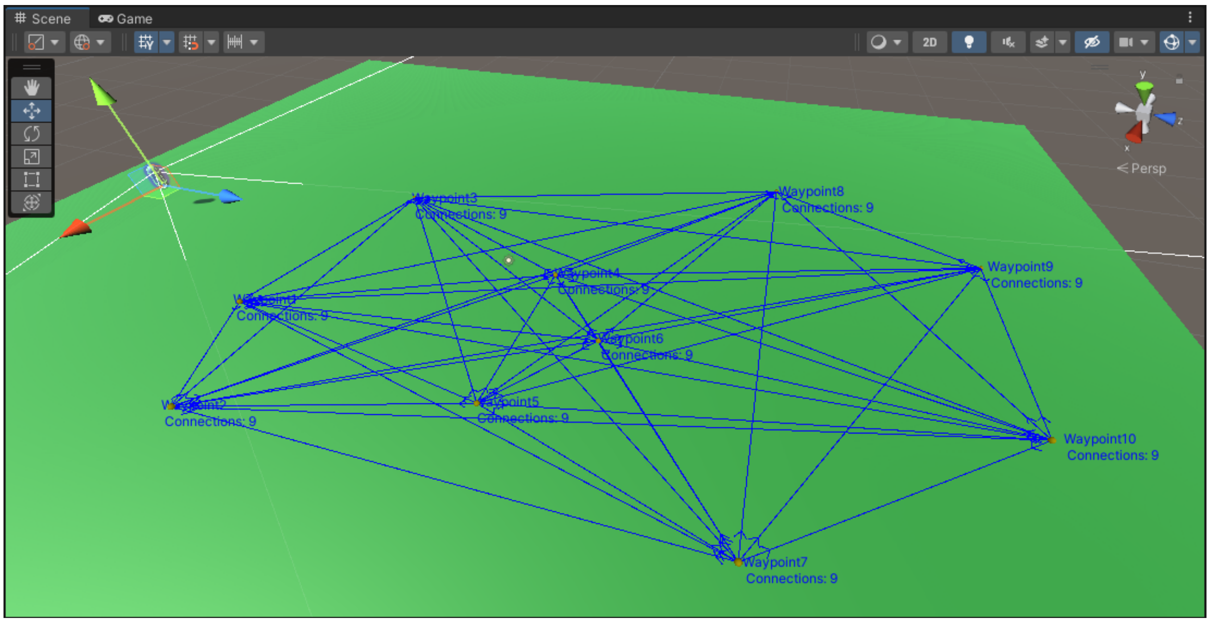


Image description: *The scene window. I have added 10 waypoint nodes to the scene and set the connections so that that <u>every</u> node connects to every other node in your waypoint graph.*

I have set the y position of all my waypoint nodes to 1.

Save you level.

You can playtest the level and visualise the waypoint graph during playtesting. To do this go to the Game Window and click the Gizmos button in the window menu bar. You should now see the Gizmo lines and spheres drawn in play view. See the screenshot below for an example.

🎮 **<u>Playtest</u> - Playtest the scene.** You should be able to move around the level using the W, A, S, and D keys and use the mouse to look around. You should see the waypoint graph drawn in your level. See the screenshot below.

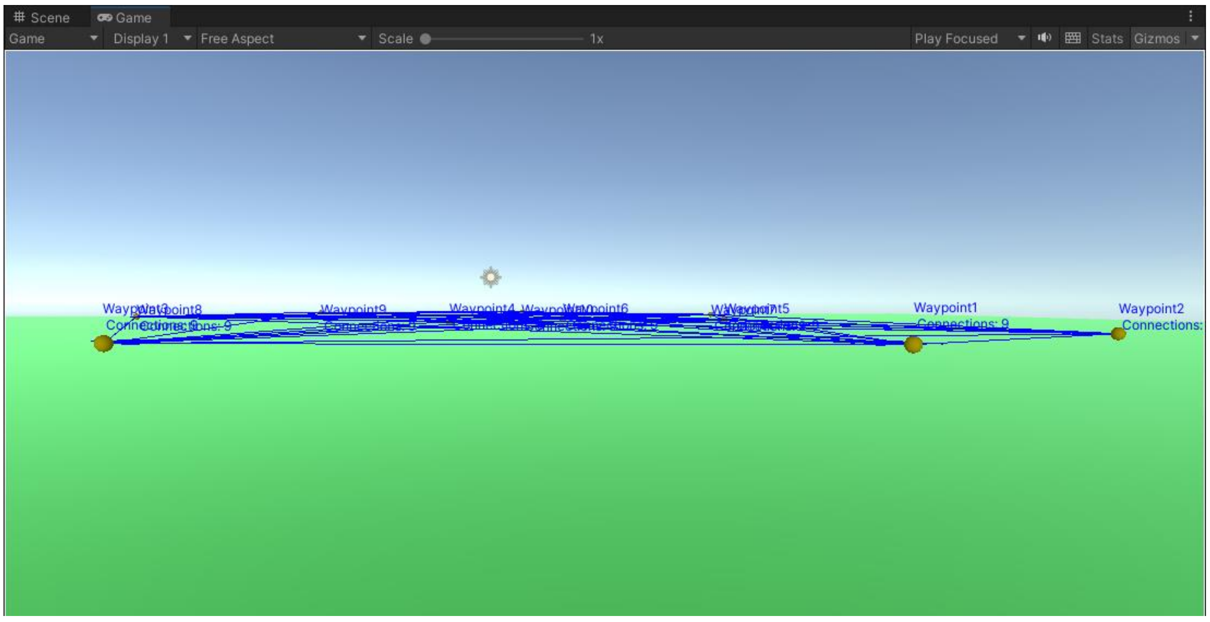


Image description: *The game window. I have clicked the Gizmos button in the window menu bar to ensure the waypoint graph is visible.*

Stop playtesting the level.

In the next sections we will create the classes needed to implement the Ant Colony Optimisation algorithm. However, first we need to set some goal locations.

8. Waypoint Nodes – Setting Goal Locations

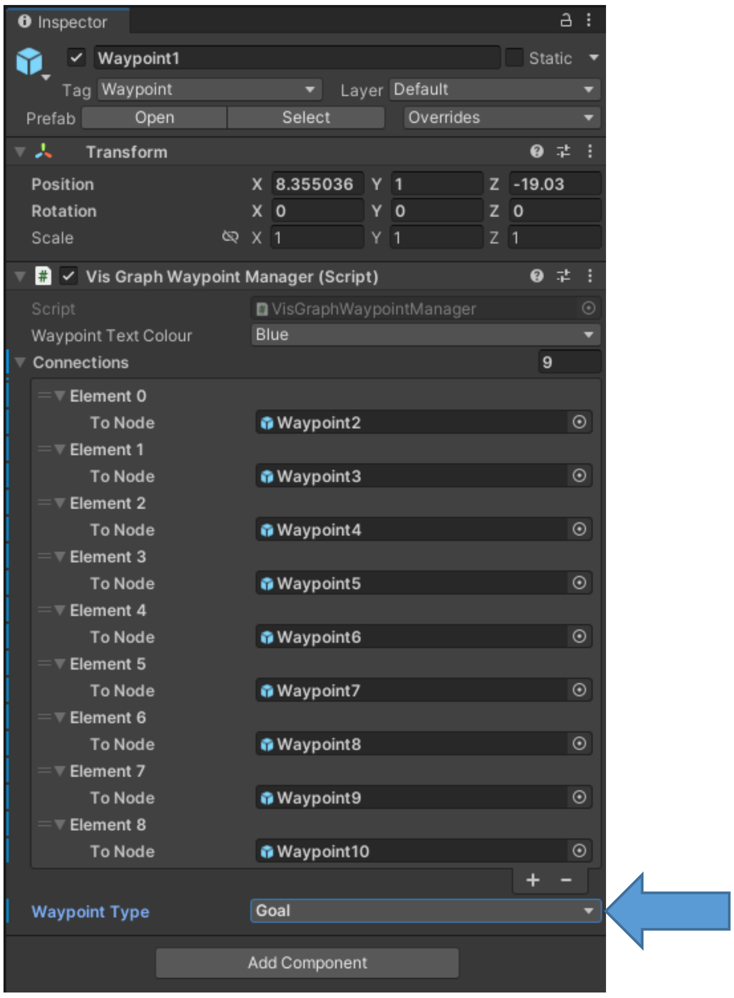
Next, we will set <u>all</u> the nodes in the waypoint graph to be <u>goal</u> locations. You do not have to set every node in your waypoint graph for Ant Colony Optimisation as a goal location; however, for this example we will.

Go to the Hierarchy window in the Unity Editor. Select a node.

Go to the Inspector and set the Waypoint Type property to Goal.

Repeat the process for <u>all the nodes</u> in your waypoint graph.

Below is a screenshot of a Waypoint nodes properties. Note the Waypoint Type has been set to goal.



```
Inspector
✓ Waypoint1                                   Static
Tag  Waypoint                  Layer  Default
Prefab  Open        Select          Overrides
Transform
Position      X 8.355036   Y 1   Z -19.03
Rotation      X 0          Y 0   Z 0
Scale         X 1          Y 1   Z 1
✓ Vis Graph Waypoint Manager (Script)
Script                  VisGraphWaypointManager
Waypoint Text Colour    Blue
Connections                                9
  Element 0
    To Node             Waypoint2
  Element 1
    To Node             Waypoint3
  Element 2
    To Node             Waypoint4
  Element 3
    To Node             Waypoint5
  Element 4
    To Node             Waypoint6
  Element 5
    To Node             Waypoint7
  Element 6
    To Node             Waypoint8
  Element 7
    To Node             Waypoint9
  Element 8
    To Node             Waypoint10
                                        + -
Waypoint Type           Goal
Add Component
```

***Image description:** The inspector window. I have set the Waypoint Type to Goal.*

9. Ant Colony Optimisation – Overview – PLEASE READ

So far in this workbook we have created a simple environment and a simple waypoint graph where each node is connected to every other node. Let's now review why we have done this and review the Ant Colony Optimisation algorithm.

We explain the algorithm here and we also give an overview of the algorithm code.

Ant Colony Optimisation (ACO) searches for an optimal path in a graph. It does this based on behaviour of ants seeking a path between their colony and a source of food [1].

Here is an overview of the concept [1]:

- Ants navigate from nest to food source.
- Ant leaves a pheromone as they travel down a path.
- More pheromone on path increases probability of path being followed.
- Shortest path is discovered via pheromone trails.

Here is an overview of the ACO process [1]:

- Pheromone/trail accumulated on path segments (e.g., connections in our waypoint graph).
- Path selected at random based on amount of pheromone present on possible paths from starting node.
    - I.e., If there is more pheromone along a path the ants are more likely to choose that path.
- Ant reaches next node, selects next path.
- Continues until reaches starting node again.
- Finished tour is a solution.
- Tour is analysed for optimality.

Let's now review or world representation. We have created a waypoint graph, so we have a simplified representation of our environment. We have connected every node to every other node so that we have a waypoint graph that represents the travelling salesperson problem (TCP). The Ant Colony Optimisation algorithm can be used to solve the travelling salesperson problem.

What is the travelling salesman / salesperson problem (TCP)?
"The travelling salesman problem (TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?""
Source: https://en.wikipedia.org/wiki/Travelling_salesman_problem

In other words, a salesman must start from one place, travel to all other cities just once and return to the original city. The travelling salesman problem is all about finding the least cost (i.e. shortest distance) path with these conditions.

In our waypoint graph the waypoint nodes represent cities in the travelling salesman problem and the connections represent connections between cities.

The Ant Colony Optimisation algorithm can be used to solve the travelling salesman problem. In other words, the Ant Colony Optimisation algorithm can generate a route between all the cities.

Overview of the Ant Colony Optimisation algorithm for the travelling salesman problem.

If there are *n* cities, then *l* distinct ants start from any of these *n* cities randomly.

These ants have two distinct advantages over real ants:

1. They have memory and do not visit the cities they have visited.
2. They know the distance of the cities and tend to choose the nearest city if the pheromones on paths are the same.

Also, if the distance of two paths is the same, they tend to choose the path with more pheromone, like real ants.

Let ant *k* be sitting at city *i* then the probability the ant chooses to travel to city *j* is p_{ij}^k. The section below outlines how p_{ij}^k is determined.

Formula for the probability of the available paths

This is given by a formula where $allowed_k$ is the set of cities not yet visited by ant *k* and $j \in allowed_k$. Basically, this restricts the calculation to allowed cities - that is those not visited before.

∈ means that it is an element in the set of…

Here is the equation for selecting a path:

$$p_{ij}^k = (\tau_{ij})^\alpha \cdot (\eta_{ij})^\beta \,/\, \{ \sum_{s} (\tau_{is})^\alpha \cdot (\eta_{is})^\beta \}$$

Where $s \in allowed_k$.

If $s \in allowed_k$ is not true $p_{ij}^k = 0$. That is, the probability of moving to a city that has already been visited is zero.

Where τ_{ij} is the intensity of the pheromone for connection i,j.

And α is a parameter to regulate the influence of τ_{ij} that takes a value between $0 \leq \alpha \leq 1$ and reflects how long the pheromone lasts.

Where η_{ij} is the visibility of city *j* from city *i* and equals $1/d_{ij}$ where d_{ij} is the distance between city *i* and *j*. So nearer cities have a higher value of η_{ij} and so those cities are more likely to be chosen. And β is a parameter to regulate the influence of η_{ij} that takes a value between $0 \leq \beta \leq 1$.

Here is an overview of the equation above in words:

1. Sum the product of the pheromone level and the visibility factor on all allowed paths from the city where the ant *k* is…
2. Calculate the product of the pheromone level and the visibility factor of the proposed path.
3. Define the probability of choosing the proposed path as the value found in (2) divided by the value found in (1).
4. **Travel down the path with the largest probability - or have a random choice if there are paths with equal probabilities.**

The equation above can be coded in the following way. I have split the equation into two key steps. First, we sum the product of the pheromone level and the visibility factor on all allowed paths from the city where the ant *k* is. Then we calculate the product of the pheromone level and the visibility factor of the proposed path and then divided by the value found in in the first step.

```
float TotalPheromoneAndVisibility = 0;

// Loop through the paths not visited and calculate total pheromone & visibility.
foreach (Connection aConnection in ConnectionsFromNodeAndNotVisited)
{
      TotalPheromoneAndVisibility += (Mathf.Pow(aConnection.GetPheromoneLevel(),
      Alpha) * Mathf.Pow((1 / aConnection.GetDistance()), Beta));
}
```

```
float PathProbability = (Mathf.Pow(aConnection.GetPheromoneLevel(), Alpha) *
                                    Mathf.Pow((1 / aConnection.GetDistance()), Beta));
PathProbability = PathProbability / TotalPheromoneAndVisibility;
```

<u>Pheromone Update</u>

The pheromone level of each connection is updated at the end of each iteration of *l* number of ants.

The pheromone update formula is as follows:

$\tau_{ij}(t + 1) = (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}$

Where:

$\tau_{i,j}$ is the amount of pheromone on a given edge i, j.

t is the iteration counter.

ρ where $0 \le \rho \le 1$ is the evaporation factor of the pheromone (i.e. it is the rate of pheromone evaporation).

$\Delta\tau_{i,j}$ is the amount of pheromone deposited. It is determined using the following:

$\Delta\tau ij = \sum_{k=1}^{l} \Delta T_{i,j}^{k}$ recall that the number of ants = *l*

$$\Delta T_{i,j}^{k} = \begin{cases} Q/L_k \text{ if ant } k \text{ travels on edge (i, j)} \\ 0 \text{ otherwise} \end{cases}$$

Where L_k is the cost, typically length of the tour for ant *k*.

Q is a constant, it should be ≤ 1. Ants whose route is longest, the value of L_k, have a smaller effect on the value of $\Delta\tau_{ij}$.

$\Delta\tau_{ij}$ is the total increase of trail level on the edge (i, j) by all ants.

The equation above can be coded in the following way.

```
// Update pheromone by formula Δτij.
// Loop through the paths and check if visited destination already.
foreach (Connection aConnection in Connections)
{
        float Sum = 0;
        foreach (Ant TmpAnt in Ants)
        {
                List<Connection> TmpAntConnections = TmpAnt.GetConnections();
                foreach (Connection tmpConnection in TmpAntConnections)
                {
                        if (aConnection.Equals(tmpConnection))
                        {
                                Sum += Q / TmpAnt.GetAntTourLength();
                        }
                }
        }

        float NewPheromoneLevel = (1 - EvaporationFactor) * aConnection.GetPheromoneLevel() + Sum;

        aConnection.SetPheromoneLevel(NewPheromoneLevel);

        // Reset path probability.
        aConnection.SetPathProbability(0);
}
```

Pseudo code

Below is pseudo code for the whole ACO algorithm.

```
Get the coordinates of the cities
For t = 1 to iteration_threshold
        For k = 1 to l                 (l = number of ants)
                For move_count = 1 to n
                        Let ant move based on the probability $p_{ij}^k$
                Loop
                Calculate $L_k$
        Loop
        Update pheromone by formula $\Delta\tau_{ij}$
Loop
```

iteration_threshold is a constant value. The number of ants is also a constant value.

Further Links

Below are some links for some further research about Ant Colony Optimisation:

- Video - https://www.youtube.com/watch?v=wfD5xlEcmuQ.
- Video - https://www.youtube.com/watch?v=qfeymoF8pb4

References

[1] Ant colony Optimization Algorithms: Introduction and Beyond. Anirudh Shekhawat, Pratik Poddar, Dinesh Boswal. <http://mat.uab.cat/~alseda/MasterOpt/ACO_Intro.pdf>.

Now we have reviewed the theory of Ant Colony Optimisation we can continue with developing the algorithm.

10. Ant Colony Optimisation - ACOConnection

Next, we will add a new C# class to the project. This class will represent connections in our waypoint map. The Ant Colony Optimisation algorithm will use the connections to determine routes through the waypoint map. In this example the routes are straight-line paths between nodes; **however, this class could store a list of nodes that had been generated by a pathfinding algorithm, such as A* (*This note is useful for the module assignment*).**

Go to the Scripts folder in the Project Window.

Double click on the Scripts folder to open it.

Right click in the Scripts folder and go to Create -> C# Scripts.

When the script has been created give it the name **ACOConnection**.

Update the code in the script file to match the code below. This class does not need to use **MonoBehaviour** so the inheritance has been removed.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ACOConnection
{
    private float distance = 0;
    public float Distance
    {
        get { return distance; }
    }

    private float pheromoneLevel;
    public float PheromoneLevel
    {
        set { pheromoneLevel = value; }
        get { return pheromoneLevel; }
    }

    private float pathProbability;
    public float PathProbability
    {
        set { pathProbability = value; }
        get { return pathProbability; }
    }

    private GameObject fromNode;
    public GameObject FromNode
    {
        get { return fromNode; }
    }

    private GameObject toNode;
    public GameObject ToNode
    {
        get { return toNode; }
    }

    // Default constructor.
    public ACOConnection()
    {

    }
    public void SetConnection(GameObject FromNode, GameObject ToNode, float
DefaultPheromoneLevel)
    {
        this.fromNode = FromNode;
        this.toNode = ToNode;

        distance = Vector3.Distance(
                  FromNode.transform.position, ToNode.transform.position);

        PheromoneLevel = DefaultPheromoneLevel;

        PathProbability = 0;
    }

}
```

Save the script and go back to Unity.

11. Ant Colony Optimisation - ACOAnt

Next, we will add a new C# class to the project. This class will represent an ant in our Ant Colony Optimisation algorithm. It will store the length of the tour and the connections travelled.

Go to the Scripts folder in the Project Window.

Double click on the Scripts folder to open it.

Right click in the Scripts folder and go to Create -> C# Scripts.

When the script has been created give it the name **ACOAnt**.

Update the code in the script file to match the code below. This class does not need to use **MonoBehaviour** so the inheritance has been removed.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ACOAnt
{
    private float antTourLength = 0;
    public float AntTourLength
    {
        set { antTourLength = value; }
        get { return antTourLength; }
    }

    private List<ACOConnection> antTravelledConnections = new
List<ACOConnection>();
    public List<ACOConnection> AntTravelledConnections
    {
        get { return antTravelledConnections; }
    }

    private GameObject startNode;
    public GameObject StartNode
    {
        set { startNode = value; }
        get { return startNode; }
    }

    public ACOAnt()
    {

    }

    public void AddAntTourLength(float AntTourLength)
    {
        this.AntTourLength += AntTourLength;
    }

    public void AddTravelledConnection(ACOConnection aConnection)
    {
        antTravelledConnections.Add(aConnection);
    }

}
```

Save the script and go back to Unity.

12. Ant Colony Optimisation - ACOCON

Next, we will add a new C# class to the project. This class implement the Ant Colony Optimisation. This class should be used by other classes to control the Ant Colony Optimisation process.

Go to the Scripts folder in the Project Window.

Double click on the Scripts folder to open it.

Right click in the Scripts folder and go to Create -> C# Scripts.

When the script has been created give it the name **ACOCON**.

Update the code in the script file to match the code below. This class does not need to use **MonoBehaviour** so the inheritance has been removed.

Note – your algorithm is large, so take you time typing it out.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ACOCON
{
    private float defaultPheromone = 1.0f;
    public float DefaultPheromone
    {
        get { return defaultPheromone; }
    }

    private float Alpha = 1.0f;

    private float Beta = 0.0001f;

    // where 0 ≤ EvaporationFactor ≤ 1 is the evaporation factor of the pheromone.
    private float EvaporationFactor = 0.5f;

    // Q is a constant, it should be ≤ 1.
    private float Q = 0.0006f;

    // Ants of agents moving through the graph. This class stores properties: Total distance and
connections used.
    private List<ACOAnt> Ants = new List<ACOAnt>();

    // The generated route.
    private List<ACOConnection> MyRoute = new List<ACOConnection>();

    public ACOCON()
    {

    }
    /*  IterationThreshold = Max number of iterations.
     TotalNumAnts = Total number of ants in the simulation.
     Connections = Connections between nodes.
     WaypointNodes = All the waypoint nodes in the waypoint graph used by the ACO algorithm.
  */
    public List<ACOConnection> ACO(int IterationThreshold, int TotalNumAnts,
        GameObject[] WaypointNodes, List<ACOConnection> Connections,
        GameObject StartNode, int MaxPathLength)
    {
        if (StartNode == null)
        {
            Debug.Log("No Start node.");
            return null;
        }

        // The node the ant is currently at.
        GameObject currentNode;

        // A list of all visited nodes.
        List<GameObject> VisitedNodes = new List<GameObject>();

        for (int i = 0; i < IterationThreshold; i++)
        {
            // Clear ants from previous iterations.
            Ants.Clear();

            for (int i2 = 0; i2 < TotalNumAnts; i2++)
            {
                ACOAnt aAnt = new ACOAnt();

                // Randomly choose start node.
                currentNode = WaypointNodes[Random.Range(0, WaypointNodes.Length)];
                aAnt.StartNode = currentNode;

                VisitedNodes.Clear();

                // Keep moving through the nodes until visited them all.
                // Keep looping until the number of nodes visited equals the number of nodes.
                while (VisitedNodes.Count < WaypointNodes.Length)
                {
                    // Get all connections from node.
                    List<ACOConnection> ConnectionsFromNodeAndNotVisited =
                        AllConnectionsFromNodeAndNotVisited(currentNode, Connections, VisitedNodes);

                    // Sum the product of the pheromone level and the visibility
                    // factor on all allowed paths.
                    float TotalPheromoneAndVisibility =
                        CalculateTotalPheromoneAndVisibility(ConnectionsFromNodeAndNotVisited);

                    // Calculate the product of the pheromone level and the visibility
                    // factor of the proposed path.
                    // Loop through the paths and check if visited destination already.
                    foreach (ACOConnection aConnection in ConnectionsFromNodeAndNotVisited)
                    {
                        // Not visited the path before.
                        float PathProbability = (Mathf.Pow(aConnection.PheromoneLevel, Alpha) *
                            Mathf.Pow((1 / aConnection.Distance), Beta));
                        PathProbability = PathProbability / TotalPheromoneAndVisibility;

                        // Set path probability. Path probability is reset
                        // to zero at the end of each run.
                        aConnection.PathProbability = PathProbability;
                    }
                    // Travel down the path with the largest probability - or have a random
                    // choice if there are paths with equal probabilities.
                    // Loop through the paths and check if visited destination already.
                    ACOConnection largestProbability = null;
                    if (ConnectionsFromNodeAndNotVisited.Count > 0)
                    {
                        largestProbability = ConnectionsFromNodeAndNotVisited[0];
                        for (int i3 = 1; i3 < ConnectionsFromNodeAndNotVisited.Count; i3++)
                        {
                            if (ConnectionsFromNodeAndNotVisited[i3].PathProbability >
largestProbability.PathProbability)
                            {
                                largestProbability = ConnectionsFromNodeAndNotVisited[i3];
                            }
                            else if (ConnectionsFromNodeAndNotVisited[i3].PathProbability ==
largestProbability.PathProbability)
                            {
                                // Currently, 100% of the time chooses shortest connection if
probabilities are the same.
                                if (ConnectionsFromNodeAndNotVisited[i3].Distance <
largestProbability.Distance)
                                {
                                    largestProbability = ConnectionsFromNodeAndNotVisited[i3];
                                }
                            }
                        }
                    }

                    // largestProbability contains the path to move down.
                    VisitedNodes.Add(currentNode);

                    if (largestProbability != null)
                    {
                        currentNode = largestProbability.ToNode;

                        aAnt.AddTravelledConnection(largestProbability);
                        aAnt.AddAntTourLength(largestProbability.Distance);
                    }

                } //~END: While loop.

                // Add a connection from the current node back to the start node for this tour.
                foreach (ACOConnection aConnection in Connections)
                {
                    if (aConnection.FromNode.Equals(currentNode))
                    {
                        if (aConnection.ToNode.Equals(aAnt.StartNode))
                        {
                            aAnt.AddTravelledConnection(aConnection);
                            aAnt.AddAntTourLength(aConnection.Distance);
                        }
                    }
                }

                Ants.Add(aAnt);
            }
            // Update pheromone by formula Delta_Tau_ij.
            // Loop through the paths and check if visited destination already.
            foreach (ACOConnection aConnection in Connections)
            {
                float Sum = 0;
                foreach (ACOAnt TmpAnt in Ants)
                {
                    List<ACOConnection> TmpAntConnections = TmpAnt.AntTravelledConnections;
                    foreach (ACOConnection tmpConnection in TmpAntConnections)
                    {
                        if (aConnection.Equals(tmpConnection))
                        {
                            Sum += Q / TmpAnt.AntTourLength;
                        }
                    }

                }

                float NewPheromoneLevel = (1 - EvaporationFactor) * aConnection.PheromoneLevel + Sum;

                aConnection.PheromoneLevel = NewPheromoneLevel;

                // Reset path probability.
                aConnection.PathProbability = 0;
            }

        }

        // Output connections and Pheromone to the log.
        LogAnts();
        LogRoute(StartNode, MaxPathLength, WaypointNodes, Connections);
        LogConnections(Connections);

        MyRoute = GenerateRoute(StartNode, MaxPathLength, Connections);
        return MyRoute;
    }
    // Return all Connections from a node.
    private List<ACOConnection> AllConnectionsFromNode(GameObject FromNode, List<ACOConnection>
Connections)
    {
        List<ACOConnection> ConnectionsFromNode = new List<ACOConnection>();

        foreach (ACOConnection aConnection in Connections)
        {
            if (aConnection.FromNode == FromNode)
            {
                ConnectionsFromNode.Add(aConnection);
            }
        }
        return ConnectionsFromNode;
    }

    // Return all Connections from a node that have not been visited.
    private List<ACOConnection> AllConnectionsFromNodeAndNotVisited
        (GameObject FromNode, List<ACOConnection> Connections, List<GameObject> VisitedList)
    {
        List<ACOConnection> ConnectionsFromNode = new List<ACOConnection>();

        foreach (ACOConnection aConnection in Connections)
        {
            if (aConnection.FromNode == FromNode)
            {
                if (!VisitedList.Contains(aConnection.ToNode))
                {
                    ConnectionsFromNode.Add(aConnection);
                }
            }
        }

        return ConnectionsFromNode;
    }

    // Sum the product of the pheromone level and the visibility factor on all allowed paths.
    private float CalculateTotalPheromoneAndVisibility(List<ACOConnection>
ConnectionsFromNodeAndNotVisited)
    {
        float TotalPheromoneAndVisibility = 0;

        // Loop through the paths not visited and calculate total pheromone & visibility.

        foreach (ACOConnection aConnection in ConnectionsFromNodeAndNotVisited)
        {
            TotalPheromoneAndVisibility +=
                (Mathf.Pow(aConnection.PheromoneLevel, Alpha) * Mathf.Pow((1 / aConnection.Distance),
Beta));
        }

        return TotalPheromoneAndVisibility;
    }
    public List<ACOConnection> GenerateRoute(GameObject StartNode, int MaxPath, List<ACOConnection>
Connections)
    {
        GameObject CurrentNode = StartNode;
        List<ACOConnection> Route = new List<ACOConnection>();

        ACOConnection HighestPheromoneConnection = null;
        int PathCount = 1;

        while (CurrentNode != null)
        {
            List<ACOConnection> AllFromConnections = AllConnectionsFromNode(CurrentNode, Connections);

            if (AllFromConnections.Count > 0)
            {
                HighestPheromoneConnection = AllFromConnections[0];

                foreach (ACOConnection aConnection in AllFromConnections)
                {
                    if (aConnection.PheromoneLevel > HighestPheromoneConnection.PheromoneLevel)
                    {
                        HighestPheromoneConnection = aConnection;
                    }

                }

                Route.Add(HighestPheromoneConnection);
                CurrentNode = HighestPheromoneConnection.ToNode;
            }
            else
            {
                CurrentNode = null;
            }

            // If the current node is the start node at this point then we have looped through the path
and should stop.
            if (CurrentNode.Equals(StartNode))
            {
                CurrentNode = null;
            }

            // If the path count is greater than a max we should stop.
            if (PathCount > MaxPath)
            {
                CurrentNode = null;
            }
            PathCount++;
        }

        return Route;
    }
```

```
// Log Connections.
private void LogConnections(List<ACOConnection> Connections)
{
    foreach (ACOConnection aConnection in Connections)
    {
        Debug.Log(">" + aConnection.FromNode.name + " | ---> " +
            aConnection.ToNode.name + " = " + aConnection.PheromoneLevel);
    }
}

// Log Route
private void LogRoute(GameObject StartNode, int MaxPath,
    GameObject[] WaypointNodes, List<ACOConnection> Connections)
{
    GameObject CurrentNode = null;
    foreach (GameObject GameObjectNode in WaypointNodes)
    {
        if (GameObjectNode.Equals(StartNode))
        {
            CurrentNode = GameObjectNode;
        }
    }

    ACOConnection HighestPheromoneConnection = null;
    string Output = "Route (Q: " + Q + ", Alpha: " + Alpha + ", Beta: " +
                        Beta + ", EvaporationFactor: " +
                        EvaporationFactor + ", DefaultPheromone: " + DefaultPheromone + "):\n";
    int PathCount = 1;
    while (CurrentNode != null)
    {
        List<ACOConnection> AllFromConnections = AllConnectionsFromNode(CurrentNode, Connections);

        if (AllFromConnections.Count > 0)
        {
            HighestPheromoneConnection = AllFromConnections[0];

            foreach (ACOConnection aConnection in AllFromConnections)
            {
                if (aConnection.PheromoneLevel > HighestPheromoneConnection.PheromoneLevel)
                {
                    HighestPheromoneConnection = aConnection;
                }

            }

            CurrentNode = HighestPheromoneConnection.ToNode;

            Output += "| FROM: " + HighestPheromoneConnection.FromNode.name + ", TO: " +
                HighestPheromoneConnection.ToNode.name +
                " (Pheromone Level: " + HighestPheromoneConnection.PheromoneLevel + ") | \n";
        }
        else
        {
            CurrentNode = null;
        }
```

```
            // If the current node is the start node at this point then we have looped
            // through the path and should stop.
            if (CurrentNode.Equals(StartNode))
            {
                CurrentNode = null;

                Output += "HOME (Total Nodes:" + WaypointNodes.Length +
                    ", Nodes in Route: " + PathCount + ").\n";
            }

            // If the path count is greater than a max we should stop.
            if (PathCount > MaxPath)
            {
                CurrentNode = null;

                Output += "MAX PATH (Total Nodes:" + WaypointNodes.Length +
                    ", Nodes in Route: " + PathCount + ").\n";
            }
            PathCount++;
        }

        Debug.Log(Output);
    }

    // Log Route
    private void LogAnts()
    {
        string Output = " Last Ant Tour Info (Q: " + Q + ", Alpha: " + Alpha + ", Beta: " + Beta + ",
EvaporationFactor: " +
                                        EvaporationFactor + ", DefaultPheromone: " + DefaultPheromone +
"):\n";

        for (int i = 0; i < Ants.Count; i++)
        {
            Output += "Ant " + i + " - Start Node: " + Ants[i].StartNode.name +
                " | Tour Length: " + Ants[i].AntTourLength + "\n";
        }

        Debug.Log(Output);
    }

}
```

Save the script and go back to Unity.

13. Ant Colony Optimisation - ACOTester

Next, we will add a new C# class to the project. This class implement a tester for the Ant Colony Optimisation algorithms. This class is an example of how you can use the Ant Colony Optimisation algorithms to generate a route between waypoint nodes.

Go to the Scripts folder in the Project Window.

Double click on the Scripts folder to open it.

Right click in the Scripts folder and go to Create -> C# Scripts.

When the script has been created give it the name **ACOTester**.

Update the code in the script file to match the code below. This class does not need to use **MonoBehaviour** so the inheritance has been removed.

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

public class ACOTester : MonoBehaviour
{
    // The ACO Controller.
    private ACOCON MyACOCON = new ACOCON();

    // Array of possible waypoints.
    private List<GameObject> Waypoints = new List<GameObject>();

    // Connections between nodes.
    private List<ACOConnection> Connections = new List<ACOConnection>();

    // The route generated by the ACO algorithm.
    private List<ACOConnection> MyACORoute = new List<ACOConnection>();

    // Debug line offset.
    private Vector3 OffSet = new Vector3(0, 0.5f, 0);

    // The Start node for any created route.
    [SerializeField] private GameObject StartNode;

    // The max length of a path created by the ACO.
    [SerializeField] private int MaxPathLength;

    // Start is called before the first frame update
    void Start()
    {
        if (StartNode == null)
        {
            Debug.Log("No start waypoint node.");
            return;
        }
        VisGraphWaypointManager tmpWpM = StartNode.GetComponent<VisGraphWaypointManager>();
        if (tmpWpM == null)
        {
            Debug.Log("Start node is not a waypoint.");
            return;
        }

        // Find all the waypoints in the level.
        GameObject[] GameObjectsWithWaypointTag;
        GameObjectsWithWaypointTag = GameObject.FindGameObjectsWithTag("Waypoint");

        foreach (GameObject waypoint in GameObjectsWithWaypointTag)
        {
            VisGraphWaypointManager tmpWaypointCon = waypoint.GetComponent<VisGraphWaypointManager>();
            if (tmpWaypointCon)
            {
                if (tmpWaypointCon.WaypointType == VisGraphWaypointManager.waypointPropsList.Goal)
                {
                    // We are creating a waypoint map of only the goal nodes. We want out ACO algorithm
to create the shortest path between the goal nodes.
                    Waypoints.Add(waypoint);
                }
            }
        }
        // Go through the waypoints and create connections.
        foreach (GameObject waypoint in Waypoints)
        {
            VisGraphWaypointManager tmpWaypointCon = waypoint.GetComponent<VisGraphWaypointManager>();
            // Loop through a waypoints connections.
            foreach (VisGraphConnection aVisGraphConnection in tmpWaypointCon.Connections)
            {
                ACOConnection aConnection = new ACOConnection();
                aConnection.SetConnection(waypoint, aVisGraphConnection.ToNode,
MyACOCON.DefaultPheromone);
                Connections.Add(aConnection);
            }
        }
        if (Connections.Count <= 1)
        {
            Debug.Log("Warning, you have set 1 or 0 goal nodes. You need at least 2. However, more is
expected.");
            return;
        }

        MyACORoute = MyACOCON.ACO(50, 25, Waypoints.ToArray(), Connections, StartNode, MaxPathLength);
        if (MyACORoute.Count == 0)
        {
            Debug.Log("Warning, ACO did not return a path. Please check all logs.");
        }
    }

    // Draws debug objects in the editor and during editor play (if option set).
    void OnDrawGizmos()
    {
        // Draw path.
        if (MyACORoute.Count > 0)
        {
            foreach (ACOConnection aConnection in MyACORoute)
            {
                Gizmos.color = Color.white;
                Gizmos.DrawLine((aConnection.FromNode.transform.position + OffSet),
                    (aConnection.ToNode.transform.position + OffSet));
            }
        }
    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Save the script and go back to Unity.

14. Adding the ACOTester to Our Scene

Once you have completed your **ACOTester** class we can add it to the scene.

Go to the Hierarchy window, click the Create button and add a cube to the scene. We will attach our **ACOTester** script to this cube.

We will add a vehicle asset and attach the **ACOTester** script to it.

In the Unity Editor, go to the **project window**.

Go to the folder: PolygonStarter -> Prefabs.

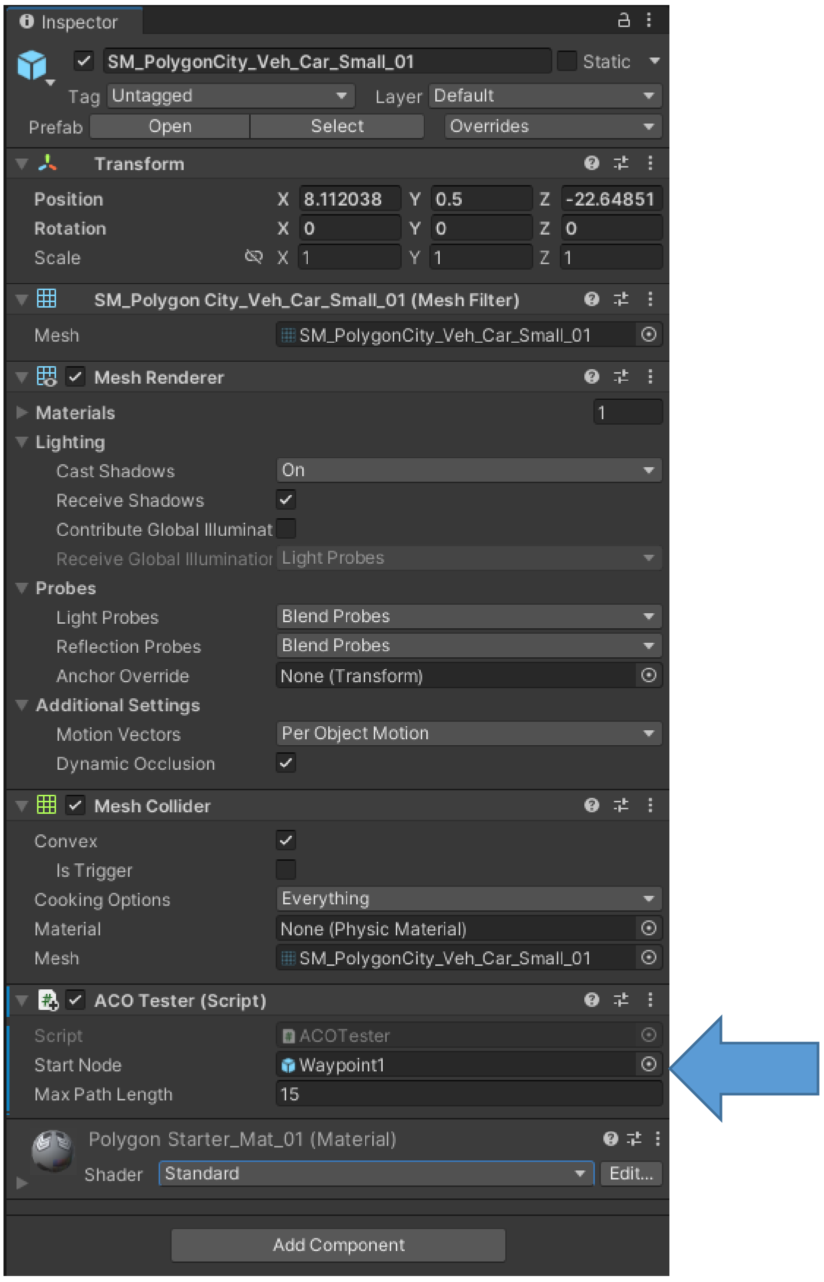
Select the **SM_PolygonCity_Veh_Car_Small_01** asset and drag it into the scene. Drag the vehicle onto the ground. Position the vehicle so that it is next to the waypoint node you would like your path to start at.

Select the vehicle in the Hierarchy window, go to the Inspector and add the **ACOTester** script.

Once the ACOTester script has been added you should also set the start node and Max Path Length for the ACO algorithm.

Start Node:	The start node should be any of the goal nodes (i.e., any node in the waypoint graph used by the ACO algorithm).
Max Path Length:	This should be slightly higher than the number of goal nodes in the waypoint graph. For example, if you have 10 nodes you could set this value to 15. This variable stops the algorithm generating a path between the goal nodes when the value it reached. For example, if you set it to 15 it will stop generating a route when 15 nodes have been reached. This variable stops the algorithm looping forever if the ACO algorithm has not generated a valid route between all the goal nodes.

Your updated Hierarchy and Inspector window should look like the screenshot below.


Inspector

SM_PolygonCity_Veh_Car_Small_01 Static
Tag Untagged Layer Default
Prefab Open Select Overrides

Transform
Position X 8.112038 Y 0.5 Z -22.64851
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

SM_Polygon City_Veh_Car_Small_01 (Mesh Filter)
Mesh SM_PolygonCity_Veh_Car_Small_01

Mesh Renderer
Materials 1
Lighting
Cast Shadows On
Receive Shadows ✓
Contribute Global Illuminat
Receive Global Illuminatior Light Probes
Probes
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)
Additional Settings
Motion Vectors Per Object Motion
Dynamic Occlusion ✓

Mesh Collider
Convex ✓
Is Trigger
Cooking Options Everything
Material None (Physic Material)
Mesh SM_PolygonCity_Veh_Car_Small_01

ACO Tester (Script)
Script ACOTester
Start Node Waypoint1
Max Path Length 15

Polygon Starter_Mat_01 (Material)
Shader Standard Edit...

Add Component

Image description: *The inspector window. I have set the Start Node to a Waypoint and Mad Path Length to 15.*

Save you game scene.

15. Playtesting the ACOTester

Now play test your scene.

When you play test your scene you should see your waypoint map.

Depending on the size of your waypoint graph it may take several seconds to generate a path through all your goal nodes. Don't worry about this.

When a route has been generated you should also see white lines slightly above the waypoint map that represent the route generated by the ACO algorithm. Your scene should look like the screenshot below.

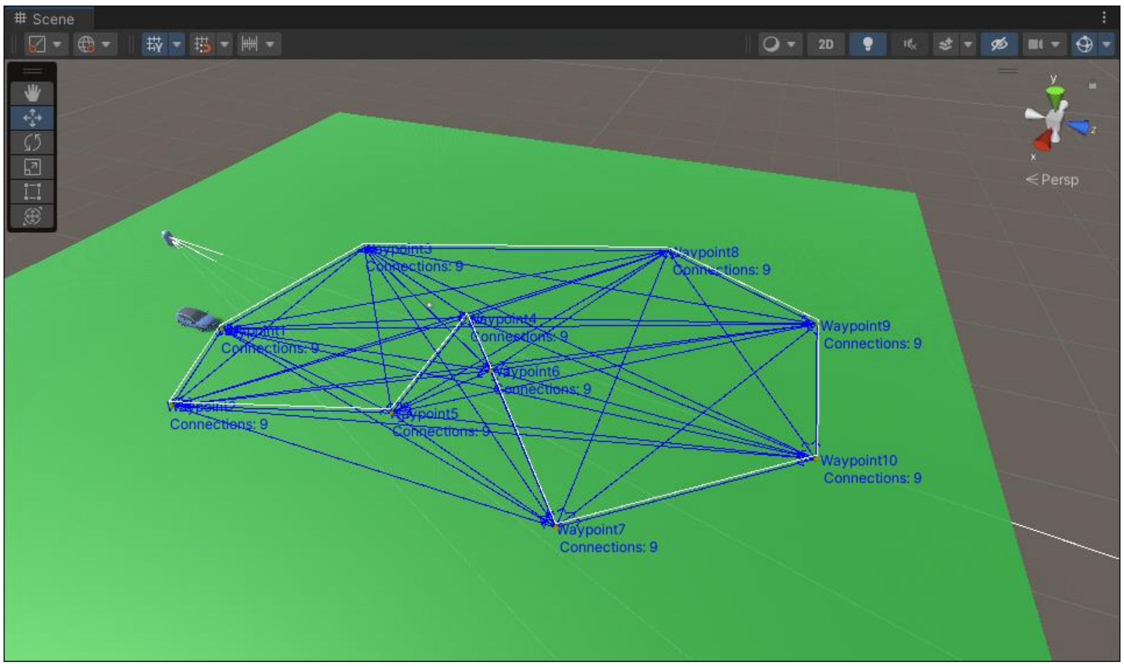


Image description: *The scene window. A route has been generated by the ACO algorithm.*

In the screenshot above every node is a goal node. The ACO algorithm must generate a route between every node in the graph. The white line represents the path the ACO algorithm has generated. As you can see the algorithm has generated a route between all nodes.

16. An Extension to Ant Colony Optimisation: Rank-based Ant System (ASrank)

In this task we will apply a simple extension to the standard ACO algorithm to improve its performance. Rank-based ant system ranks the ants according to their solution length. Only a fixed number of the best ants in an iteration are allowed to update the pheromone trails.

This is a straightforward extension to the standard ACO algorithm and **may** improve results.

Go to the Unity Editor.

Go to the Scripts folder in the Project Window.

Double click on the Scripts folder to open it.

Open the C# Script: **ACOCON**.

Add this line of code to the using section of the class:

```
using System.Linq;
```

Next, we need to change the "Update pheromone by formula Δτij" part of code in the ACO method.

Find this code in the **ACOCON** script:

```
        // Update pheromone by formula Δτij.
        // Loop through the paths and check if visited destination already.
        foreach (ACOConnection aConnection in Connections)
        {
            float Sum = 0;
            foreach (ACOAnt TmpAnt in Ants)
            {
                List<ACOConnection> TmpAntConnections = TmpAnt.AntTravelledConnections;
                foreach (ACOConnection tmpConnection in TmpAntConnections)
                {
                    if (aConnection.Equals(tmpConnection))
                    {
                        Sum += Q / TmpAnt.AntTourLength;
                    }
                }

            }
```

Change the code above to the following code. You will need to add two new lines of code before the foreach and change one variable name. See the highlighted code below.

```
        // Find the best ants.
        List<ACOAnt> SortedAnts = Ants.OrderBy(x => x.AntTourLength).ToList();
        List<ACOAnt> BestAnts = SortedAnts.GetRange(0, 3);

        // Update pheromone by formula Δτij.
        // Loop through the paths and check if visited destination already.
        foreach (ACOConnection aConnection in Connections)
        {
            float Sum = 0;
            foreach (ACOAnt TmpAnt in BestAnts)
            {
                List<ACOConnection> TmpAntConnections = TmpAnt.AntTravelledConnections;
                foreach (ACOConnection tmpConnection in TmpAntConnections)
                {
                    if (aConnection.Equals(tmpConnection))
                    {
                        Sum += Q / TmpAnt.AntTourLength;
                    }
                }

            }
```

The code above changes the ACOCON script so that only the best (i.e., shortest) 3 ant tours are used to update the pheromone in connections each iteration. This ensures only the best tours in an iteration effect the pheromone levels of connections.

You could also improve the implementation above by replacing value 3 with a variable that is defined and set at the class level. Then you can more easily adjust the best number of ants.

Save the script and go back to Unity.

Now play test your scene.

When you play test your scene you should see your waypoint map.

Depending on the size of your waypoint graph it may take several seconds to generate a path through all your goal nodes. Don't worry about this.

When a route has been generated you should also see white lines slightly above the waypoint map that represent the route generated by the ACO algorithm. Your scene should look like the screenshot below.

This updated version of the algorithm <u>may</u> improve your result.

Experiment with the number of best ants that get to update the pheromone in connections. See what effect it has.

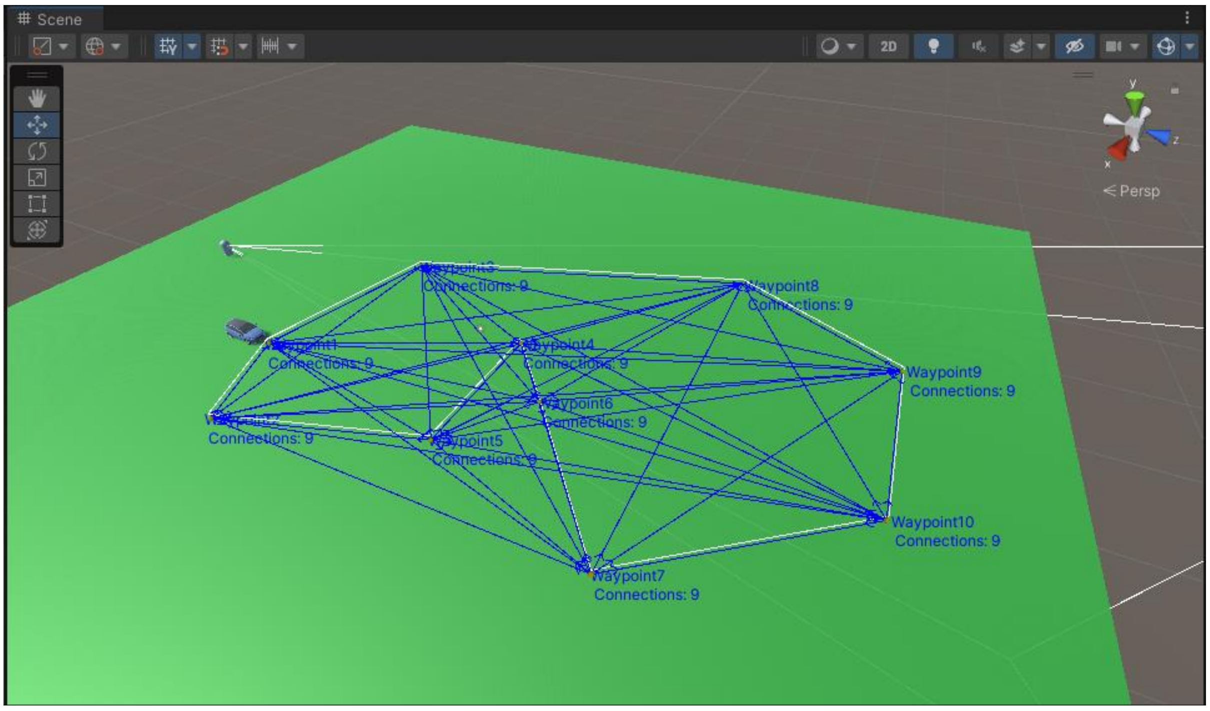


Image description: *The scene window. A route has been generated by the ACO Rank-based Ant System (ASrank) algorithm.*

17. Exploring the Ant Colony Optimisation Algorithm Further

The ACO algorithm has several parameters that affect the quality of the route produced by the algorithm. A poor outcome from the algorithm can result in no clear transition from a node or transitions that result in a route that loops around a subset of nodes (e.g., the route does not visit all required nodes).

Here are the variables that you could explore values:

Variable and Location	**Description**
EvaporationFactor *ACOCON class variable.*	The evaporation factor of the pheromone. This should be between 0 and 1. In general, I set this to 0.5.
Q *ACOCON class variable.*	Q is a constant, it should be ≤ 1. A low value of 0.0006 works well for this.
Alpha *ACOCON class variable.*	This variable sets the weight / contribution of the pheromone. In general, I set this to 1.0.
Beta *ACOCON class variable.*	This variable sets the weight / contribution of the distance between nodes. Long distances between nodes could result in distance overwhelming the routing algorithm, therefore a low value for this will reduce the impact of distance from the calculation. In general, when there are long distances between nodes, I set this to 0.0001.
DefaultPheromone *ACOCON class variable.*	The default pheromone value. In general, I set this value to 1.0.
IterationThreshold *Variable in the ACO function of the ACOCON class.*	Max number of iterations.
TotalNumAnts *Variable in the ACO function of the ACOCON class.*	The number of ants in each iteration.

Below are some questions that require you to edit the program above. **Please work through these questions:**

1. Experiment with the number and location of goal nodes. See what effect changing waypoint locations has on the routes produced.

2. Experiment with the **IterationThreshold** and **TotalNumAnts** variables. For example, you could try having more ants and few iterations or more iterations and fewer ants.

3. Explore the value of the **Beta** variable. This variable sets the weight / contribution of the distance between nodes. Long distances between nodes could result in distance overwhelming the routing algorithm, therefore a low value for this will reduce the impact of distance from the calculation. In general, when there are long distances between nodes, this value should be low to avoid distance overwhelming the pheromone value; however, you could increase the pheromone weighting or default value instead.

4. Add code to the Update method to move the vehicle along the ACO algorithm route. The route is stored in the **MyACORoute** list.

> END OF STUDENT WORKBOOK █