the unity® experiments

Developed by
a Unity Certified Instructor
and Unity Certified Developer:

GAME TRAINING

Created by Jeff Ayling and Jamie Ayling

Introduction

The Unity® experiments are a series of challenging coding and game development activities which utilise Unity®, the world's most popular game engine. Rather than following a tutorial designed to recreate a specific game, the Unity® experiments will teach your students specific coding and game design concepts which they can combine to create their own unique projects.

Unity® will change the way your students learn to write code by engaging them in a way that no other coding environment can offer. Combined with the Unity® experiments, you have a tool which will motivate, inspire and fast-track your students learning process and bring their code to life.

- Students learn how to create commercial quality virtual worlds and explore their own environments with a character, car or plane.
- Write code in C# and experiment with variables, functions, physics, conditional statements, loops, keyboard controls, detect collisions, create spawn points and more.
- Create algorithms, debug code and compile games for multiple operating systems.

The Unity® experiments will take students through the absolute basics, designing virtual worlds, through to essential coding concepts and artificial intelligence.

Each experiment focuses on a specific concept in as much detail as required to fast-track the learning process and help students master each technique.

Your teacher account allows you to login, complete the experiments yourself, view student participation and individual grades.

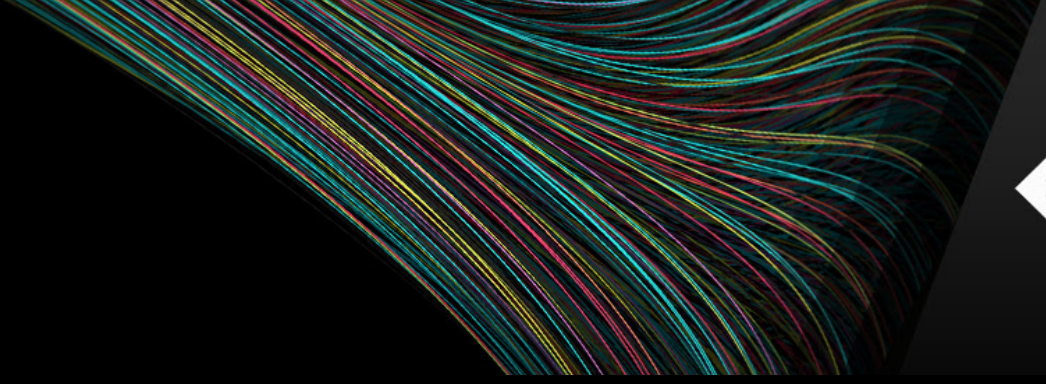
the Unity® experiments has been designed to motivate and inspire the next generation of coders.

amazon web services™

moodle

Login at http://unity.gametraining.com.au

For support, call 0402 268 066
or email experiments@gametraining.com.au



the unity® experiments

Getting Started

The Unity® Experiments: How it Works

Each student completes the design brief to ensure that they understand the objective. Students then complete an analysis of the design brief by answering questions related to the brief.

Once the design brief analysis has been completed, the experiments are unlocked one at a time.

Each experiment has an overview, an activity and an exercise.

Students can work at the their own pace or a teacher may choose to discuss the Overview in a group followed by students working through the Activity and Exercise on their own.

The Overview - explains a concept in as much detail as required so that a student is prepared to work through the activity in Unity®.

The Activity - students launch Unity® and follow the Step By Step Guide which contains a walkthrough, images and videos to help master the concept.

The Exercise - is unique for each experiment and tests a student using a series of activities such as dragging and dropping words to complete a paragraph, dragging words onto the correct positions in images, multiple choice questions, choose the correct statement games, complete the code and more.

Experiment 1 Exercise: Choose the correct answer

Learn to create a new Unity project, about the Unity Project folder structure, moving a project between computers, advice on backing up and importing files.

QUESTION: If you want to open your Unity project on a different computer, which file should you copy onto your USB?

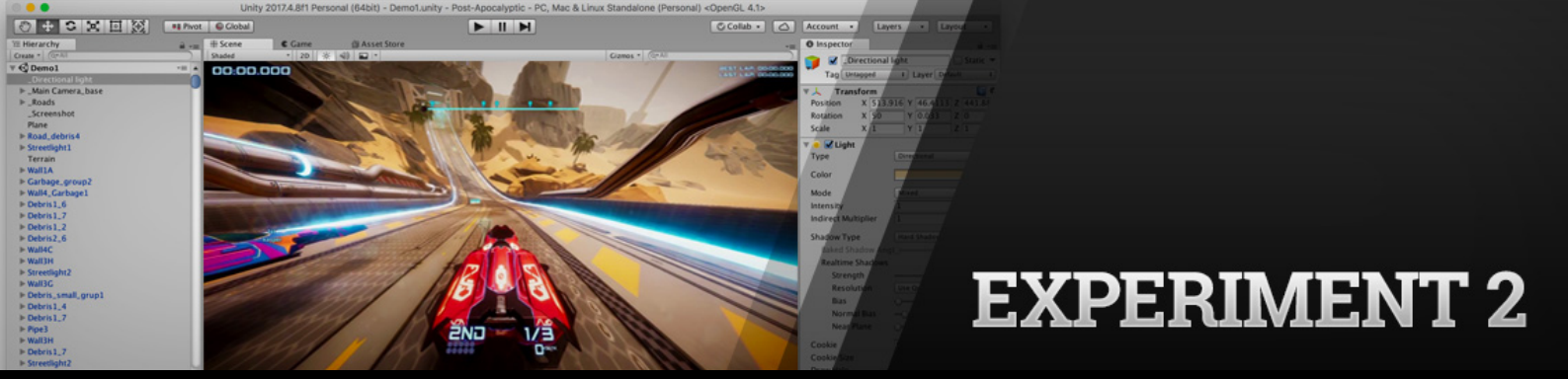
ANSWER: Entire Unity project folder

QUESTION: Is a Unity project a single file like a Photoshop .psd file or a folder full of sub-folders and files?

ANSWER: A Unity project is a folder full of sub-folders and files

QUESTION: If you create a project in Unity using version 2018 you can open the project successfully in version 2017 as well as any other older version.

ANSWER: False



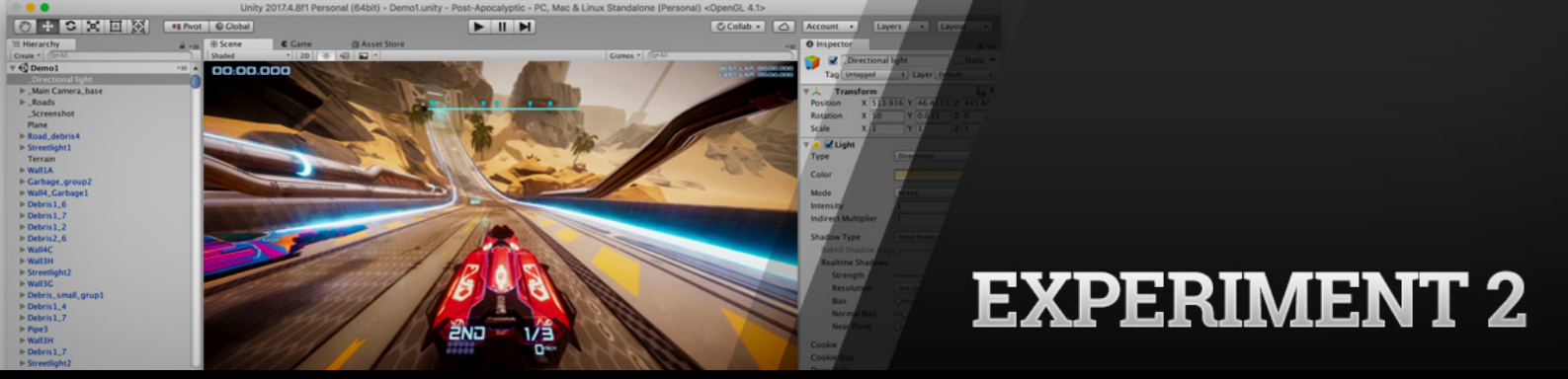
EXPERIMENT 2

Identify Elements of the Unity® Interface

Experiment 2 Exercise 1: Drag and drop the window names onto the correct part of the Unity® interface

Understanding the various windows in Unity, how to rearrange the layout and return to the default window layout.

PLEASE NOTE: This experiment contains 2 exercises.



Identify Elements of the Unity® Interface continued…

Experiment 2 Exercise 2: Fill in the missing words

The Project Window:

The Project Window displays your library of **assets / files / prefabs / items / gameobjects** that are available to use in your project. When you **import / drag / bring** assets into your project, they appear here.

The Scene View:

The Scene View allows you to visually navigate and **edit / design** your scene. The scene view can show a 3D or 2D perspective, depending on the type of **project / game / app / animation** you are working on.

The Hierarchy Window:

The Hierarchy Window is a hierarchical text representation of every **item / object /gameobject / game object** in the scene. Each item in the scene has an entry in the hierarchy, so the two windows are inherently linked. The hierarchy reveals the structure of how objects are attached to one another.

The Inspector Window:

The Inspector Window allows you to view and **edit / adjust / modify / change** all the properties of the currently selected **item / asset / object / gameobject**.
Because different types of objects have different sets of properties, the layout and contents of the inspector window will vary.

The Toolbar:

The Toolbar provides access to the most essential working features. On the left it contains the basic **tools / buttons** for manipulating the scene view and the objects within it. In the centre are the play, **pause** and step controls. The buttons to the right give you access to your **Unity** account, followed by a layer visibility menu, and finally the editor layout menu (which provides some alternate layouts for the editor windows, and allows you to save your own custom layouts).

The toolbar is not a window, and is the only part of the Unity **interface / window / app / application /software** that you can't rearrange.

Experiment 3 Exercise: Drag the words into the correct boxes

This experiment explains the Transform controls and how to move, rotate and scale GameObjects.

Games created in Unity are made up of **GameObjects** as well as code and other types of files and components.

Every item in the Hierarchy is a GameObject. When you create a new project there are already 2 GameObjects in the game - the **Main Camera** and **Directional Light**.

You can create simple 3D objects in Unity such as a Cube, Sphere, Capsule, **Cylinder** and a Plane (which is a flat square surface) or you can import files into Unity which become Game Objects once they are used in the game.

Every GameObject can be moved, rotated and **scaled** bigger or smaller. Most GameObjects can also have a **material** so that its colour can be changed or a **texture** applied to its surface.

Experiment 4 Exercise: Match the keyboard shortcuts to their function

Rotating, zooming and focusing on GameObjects, positioning, snapping and surface snapping as well as creating, colouring and applying materials to GameObjects are all covered in this experiment.

Match the keyboard short-cuts to their function

Focus on any game object in the Hierarchy by	Double-clicking
Move your view of the scene left and right using	The Hand Tool
Rotate your view of the scene using	Alt + click and drag
Snap two game objects together using	The V key + click and drag

EXPERIMENT 5

Creating Virtual Worlds

Experiment 5 Exercise: Choose the correct statement

Create a spectacular 3D World with hills, trees, grass and ocean.

ANSWERS:

1. The Raise/Lower Terrain tool can create hills and valleys but can not create caves

2. The default Terrain size is 500 metres X 500 metres

3. The default Terrain height is 600 metres

4. If you walk or drive too far, you can fall off the edge of your Terrain

5. Too many trees will affect the performance of your game

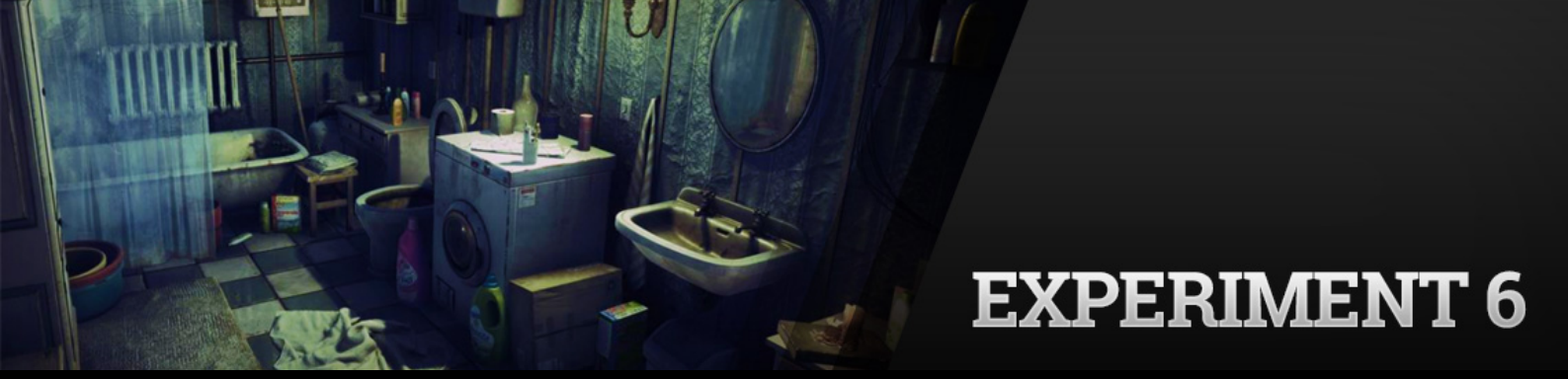
6. Search for "Seamless Texture" in Google to find images which you can use on your Terrain

7. Use the Mass Place Tree tool to randomly place trees around your Terrain

8. You don't need to create a new Unity project every time you want to create a new Terrain. Just choose File > New Scene and create a new Terrain

9. Create a Wind Zone using "Game Object > 3D Object > Wind Zone" if you would like your trees to blow in the wind

10. For best results, adjust the brush size, opacity and target strength settings



Experiment 6 Exercise: True or false

Use the FPSController to allow the user to walk, run, jump and explore a virtual World.

QUESTION: It is called a First Person Controller because it is usually the first player in the game.

ANSWER: False

QUESTION: You must always place the FPSController above the terrain otherwise it will fall into the abyss.

ANSWER: True

QUESTION: You can change the FPSController's walk speed and run speed but you can not change the jump speed.

ANSWER: False

QUESTION: You can drag as many FPSControllers into your game as you like.

ANSWER: False

EXPERIMENT 7

Cars and Aircraft

Experiment 7 Exercise: Choose the correct statement

Import cars and aircraft to explore a virtual world.

ANSWERS:

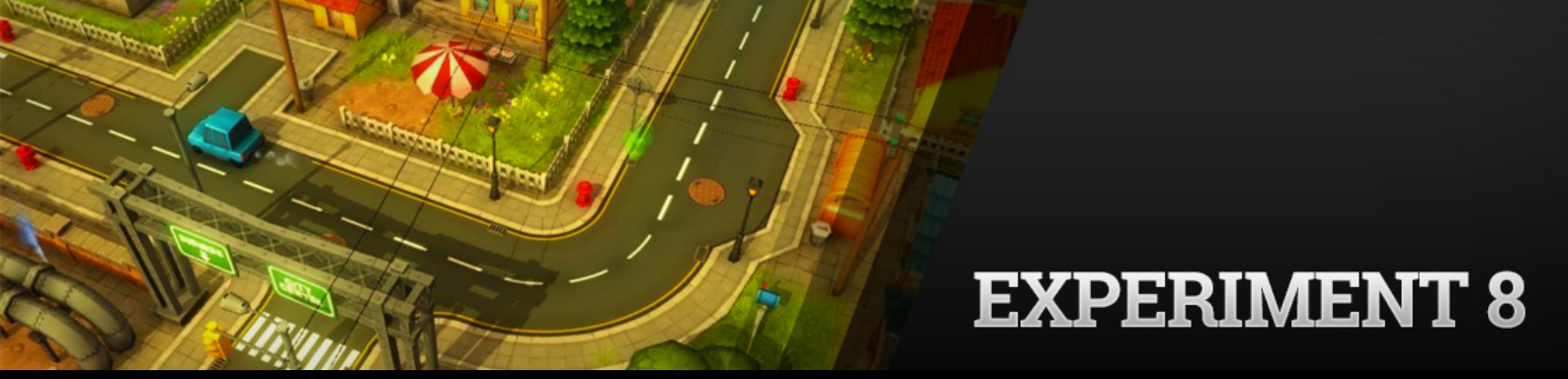
1. You need a Camera to follow your vehicle

2. Drag the SmoothFollow script onto your Camera

3. Click on the Main Camera then drag your vehicle onto the Target field on the SmoothFollow script in the Inspector

4. Set the Rotation Damping to 2 and Height Damping to 10 on the SmoothFollow script

5. You can change the colour of your vehicle by dropping a material onto the vehicle 3d model



Move, Rotate and Scale with C# Code

Experiment 8 Exercise: Choose the correct answer

Create a C# script to move, rotate and scale GameObjects.

QUESTION: Which of the following lines of code will move an object to the right?

ANSWER: transform.position += transform.right;

QUESTION: Select the names of the 2 functions which are created by default when you create a new C# script in Unity

ANSWER: Start and Update

QUESTION: Thinking about the x, y, z positions on the Transform, which of the following positions would centre an object in your scene, raised 5 metres in the air?

ANSWER: X=0 Y=5 Z=0

QUESTION: If your script is named MoveObject, the correct class name, which must appear at the top of the script, is?

ANSWER: MoveObject

EXPERIMENT 9

Keyboard Controls via the Input Manager

Experiment 9 Exercise: Choose the correct code

Write C# code to communicate with the Input Manager. This will allow the end user to use the arrow keys to control game play.

QUESTION: Select the line of code which allows the user to use the keyboard controls to move an object left and right:

ANSWER: transform.position += transform.right * Input.GetAxis("Horizontal");

QUESTION: Select the line of code which allows the user to use the keyboard controls to move an object forward and back:

ANSWER: transform.position+= transform.forward * Input.GetAxis("Vertical");

QUESTION: Select the line of code which allows the user to use the keyboard controls to rotate an object to the right and left:

ANSWER: transform.position+= transform.forward * Input.GetAxis("Vertical");

QUESTION: Select the line of code which allows the user to use the keyboard controls to scale an object bigger and smaller:

ANSWER: transform.localScale += transform.localScale * Input.GetAxis("Vertical");

EXPERIMENT 10

Variables

Experiment 10 Exercise: Choose the correct code

Variables are information containers. There are different types of containers for different types of information.

QUESTION: Three of the five lines of code below will return an error and prevent your game from running. Identify these three lines by ticking the boxes to the left:

ANSWER: float speed="0.01";
float speed=0.1f
speed float=0.01;

QUESTION: Which of the following lines of code will make our variable 'thePlayer' show up in the Inspector:

ANSWER: public GameObject thePlayer;

Experiment 11 Exercise: Complete the function and fill in the blanks

Functions are a way of grouping code into manageable reusable portions but they can also act as mini applications which can be set to run a task and even return information when the function has finished processing.

QUESTION: Complete the following function so that the GameObject will be positioned at (0,0,0) when the game starts.

ANSWER:

```
void Start () {

    transform.position=new Vector3(0,0,0);

}
```

void Start () is a built-in Unity **function** which will occur once when the game is launched. This is most commonly used to **set** some values such as giving the player 3 lives, starting a spawnpoint, setting the high score etc.

void **Update** () is a built-in Unity function which will loop over and over as fast as your **computer** can go. Think about the frame **rate** of a game which may run at 60 frames per **second**, that means the void Update() is **looping** 60 times per second.

EXPERIMENT 12

Physics, Gravity and Rigidbodies

Experiment 12 Exercise: Drag the words into the correct boxes

Discover how to add physics to an object by adding a Rigidbody and move objects by applying forces.

To have convincing physical behaviour, an object in a game must **accelerate** correctly and be affected by collisions, **gravity** and other forces.

Unity's built-in **physics** engines provide components that handle the physical simulation for you.

With just a few parameter **settings**, you can create objects that behave in a **realistic** way.

By controlling the **physics** from scripts, you can give an object the dynamics of a **vehicle**, a machine, or even a piece of fabric.

Experiment 13 Exercise: Complete the following functions

Learn how to detect collisions and trigger code based on the object you collided with.

```
 void OnCollisionEnter(Collision info)
  {
     if (info.gameObject.name == "Enemy")
     {
        Destroy(gameObject);
     }
     if (info.gameObject.name == "Powerup")
     {
        playerLives++;
     }
  }
```

```
void OnCollisionEnter(Collision info)
  {
     if (info.gameObject.name == "Plane")
     {
        Debug.Log("Collision Detected");
     }
  }
  void OnTriggerEnter(Collider info)
  {
     if (info.gameObject.name == "Plane")
     {
        Debug.Log("Trigger Detected");
     }
  }
```

Experiment 14 Exercise: Choose the correct answer

Learn about the powerful Instantiate function and how to create Spawnpoints.

QUESTION: The Instantiate function requires three important pieces of information. What are they?

ANSWER: The rotation of the object
Which GameObject to spawn
Where to spawn the object in the 3D environment

QUESTION: You should never instantiate within void Update() because Unity will most likely crash.

ANSWER: True

QUESTION: When writing your code, the word Instantiate/instantiate should have a lowercase 'i'

ANSWER: False

InvokeRepeating

Experiment 15 Exercise: Fill in the missing code

Learn an advanced technique to call a function at a recurring predetermined interval.

The Spawner script below uses InvokeRepeating to call a function. Enter the function name and the name of the GameObject to be spawned below:

```
public class Spawner : MonoBehaviour {

    public GameObject theProjectile;

    void Start () {
        InvokeRepeating("SpawnAProjectile",1.0f, 5.0f);
    }


    void SpawnAProjectile () {

        Instantiate(theProjectile,transform.position,transform.rotation);

    }
}
```

Experiment 16 Exercise: Fill in the missing code

Discover an advanced technique for spawning objects using a coroutine.

```
public class Spawner : MonoBehaviour {

    public GameObject theProjectile;

    void Start () {

        StartCoroutine(SpawnACube());

    }

    IEnumerator SpawnACube () {

        while(true){

            Instantiate(theProjectile,transform.position,transform.rotation);
            yield return new WaitForSeconds(0.5f);

        }
    }
}
```

Communicating with Game Objects

Experiment 17 Exercise: Fill in the missing words

A simple technique for Communicating between objects in a game.

```
public class Spawner : MonoBehaviour {

public GameObject theCube;

GameObject newObject;
Color rndColor;

void Start() {

    StartCoroutine(Spawn());

}
```

```
IEnumerator Spawn() {

    while (true) {

        newObject = Instantiate(theCube, transform.position, transform.rotation);

        rndColor = new Color(Random.Range(0F, 1F), Random.Range(0, 1F), Random.Range(0, 1F));

        newObject.GetComponent<Renderer>().material.color = rndColor;

        Destroy(newObject, 20f); yield return new WaitForSeconds(0.5f); }

    }

}
```

Experiment 18 Exercise: Fill in the missing words

Getting started with Artificial Intelligence

```
void Update()
  {
     range = Vector3.Distance (thePlayer.transform.position, transform.position);
     if (range < 40)
     {
        transform.LookAt(thePlayer.transform.position);
     }
     if (range < 30 && range > 15)
     {
        transform.position += transform.forward;
     }
     if (range< 12)
     {
        transform.position += transform.back;
     }
     if (range < 6)
     {
        Instantiate(explosion, transform.position, transform.rotation);
        Destroy(gameObject);
     }
  }
```