INTRODUCTION TO UNITY

CS4HS 2016

The University of Queensland

Table of Contents

INTRODUCTION **1**
EXAMPLES 1
INSTALLING UNITY 2
Unity Download Assistant *2*
Individual Installers *2*
2D OR 3D PROJECTS 3
Full 3D *3*
Orthographic 3D *3*
Full 2D *4*
2D gameplay with 3D graphics *4*
2D gameplay and graphics, with a perspective camera *4*
STARTING UNITY FOR THE FIRST TIME 6
LEARNING THE INTERFACE 8
The Project Window *9*
The Scene View Window *9*
The Hierarchy Window *10*
The Inspector Window *11*
The Toolbar *11*

LET'S MAKE A GAME **12**
CREATE A NEW PROJECT 12
THE FIRST SCENE – MAIN MENU 12
Step 1. The first thing we want to do is save the scene. *12*
Step 2. Next we will import some assets that we will use on the Main Menu. *12*
Step 3. Now it is time to create the UI. *14*
Step 4. Now on to some basic programming. *17*
THE SECOND SCENE – THE GAME PROPER 19
Step 1: Create a new scene and save it. *19*
Step 2: Some more assets to import. *19*
Step 3: Time to create the vehicle and track. *19*
Step 4: Coding some movement for the vehicle. *21*
Step 5: Time for scrolling backgrounds. *22*
Step 6: Creating obstacles and other collectables. *23*
Step 7: Colliding with the obstacles and collectables. *24*
CONNECTING THE TWO SCENES 26
Step 1: Save the current scene. *26*
Step 2: Setting the Build Settings. *26*
Step 3: Return to the Start Button. *26*
Step 4: Test it. Build It. Play It. *26*

INTRODUCTION

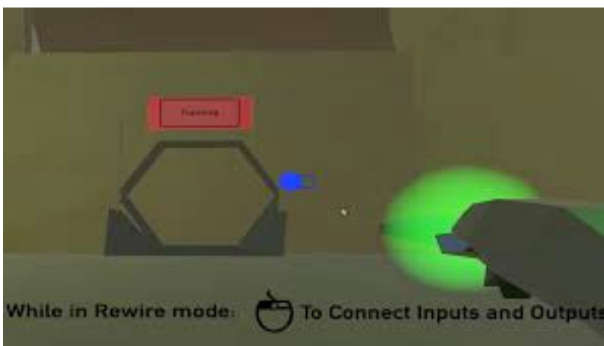
> Unity is a feature rich, fully integrated development engine that provides out-of-the-box functionality for the creation of interactive 3D content.
>
> You use Unity to assemble your art and assets into scenes and environments; add physics, light, video, audio and post-processing special effects; play test, edit and optimize your game, and when ready, publish to your chosen platforms, such as desktop computers, web browsers, iPhone, iPad, Android, Kinect and CAVE's™.
>
> Unity Technologies - https://unity3d.com/unity/industries/sim

Unity is a cross-platform development platform initially created for developing games but is now used for a wide range of things such as: architecture, art, children's apps, information management, education, entertainment, marketing, medical, military, physical installations, simulations, training, and many more.

Unity takes a lot of the complexities of developing games and similar interactive experiences and looks after them behind the scenes so people can get on with designing and developing their games. These complexities include graphics rendering, world physics and compiling. More advanced users can interact and adapt them as needed but for beginners they need not worry about it.

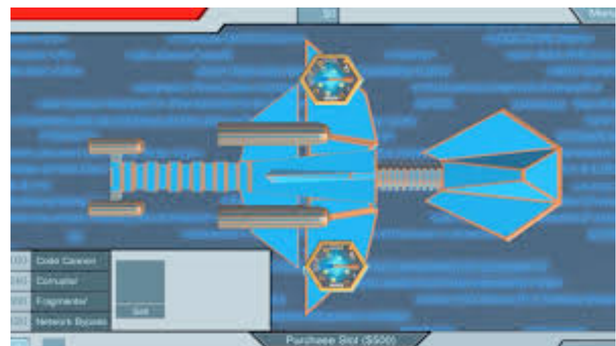
Games in Unity are developed in two halves; the first half - within the Unity editor, and the second half - using code, specifically C#. Unity is bundled with MonoDevelop or Visual Studio 2015 Community for writing C#.

EXAMPLES



Rewire
Kwergan Gregory
Bunbury Senior High School, WA

Kwergan is student at Bunbury Senior High School in WA. ***Rewire*** is a three-dimensional problem solving/puzzle game. Armed with an experimental piece of technology called the 'Rewire Tool', players must navigate through rooms using the tool to connect various combinations of output and input nodes in order to achieve objectives. The Australian STEM Video Game Challenge judges were impressed by the game's excellent implementation and gameplay elements that make it fun to play.



Malware Meltdown
Invisible Wombats
Trinity Christian School, ACT

Members of the winning group, Invisible Wombats, are students at Trinity Christian School in Canberra. ***Malware Meltdown*** is a creative two-dimensional game in which players take control of antivirus software within a corrupt computer system, using an upgradable collection of items to destroy pieces of 'malware' that threaten to compromise the system. The Australian STEM Video Game Challenge judges described Malware Meltdown as a fun and engaging way to learn about cyber security.

INSTALLING UNITY

You can download and install the Unity Editor from the Unity Website; unity3d.com/download.

Unity Download Assistant

From Unity version 5.0 onwards, the Unity Download Assistant, a small executable program (approximately 1 MB in size), lets you select which components of the Unity editor you want to download and install. If you are not sure which components you want to install, leave the default selections and click on Continue, following the installer's instructions.

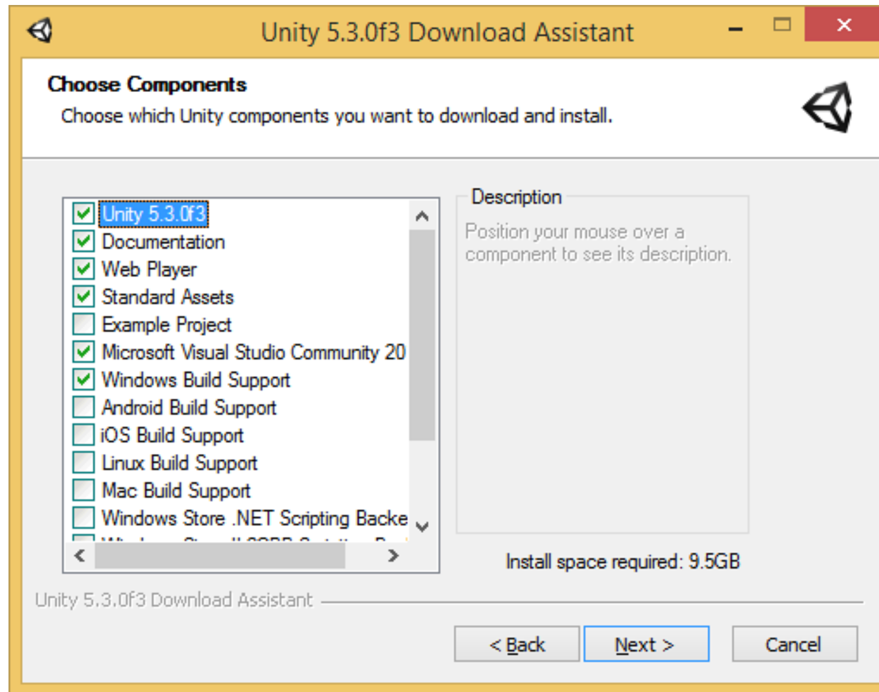


Figure 1 - Unity Download Assistant

Individual Installers

Alternatively individual installers for each of the components can be downloaded and used on multiple machines to save downloading repeatedly.

For Windows there are **six** required files:

- UnitySetup64.exe
- UnityDocumentationSetup.exe
- UnityStandardAssetsSetup.exe
- UnitySetup-Windows-Support-for-Editor-5.3.1f1.exe
- UnityWebPlayerDevelopment.exe
- vstu.msi

Each of the files should be installed in the above order.

2D OR 3D PROJECTS

Unity is equally suited to creating both 2D and 3D games. But what's the difference? When you create a new project in Unity, you have the choice to start in 2D or 3D mode. You may already know what you want to build, but there are a few subtle points that may affect which mode you choose.

The choice between starting in 2D or 3D mode determines some settings for the Unity Editor - such as whether images are imported as textures or sprites. Don't worry about making the wrong choice though, you can swap between 2D or 3D mode at any time regardless of the mode you set when you created your project. Here are some guidelines which should help you choose.

Full 3D

Figure 2 - Some 3D scenes from Unity's sample projects on the Asset Store

3D games usually make use of three-dimensional geometry, with materials and textures rendered on the surface of these objects to make them appear as solid environments, characters and objects that make up your game world. The camera can move in and around the scene freely, with light and shadows cast around the world in a realistic way. 3D games usually render the scene using perspective, so objects appear larger on screen as they get closer to the camera. For all games that fit this description, start in **3D** mode.

Orthographic 3D

Figure 3 - Some 3D games using Orthographic view

Sometimes games use 3D geometry, but use an orthographic camera instead of perspective. This is a common technique used in games which give you a bird's-eye view of the action, and is sometimes called "2.5D". If you're making a game like this, you should also use the editor in **3D** mode, because even though there is no *perspective*, you will still be working with 3D models and assets. You'll need to switch your camera and scene view to **Orthographic** though. *(scenes above from Synty Studios and BITGEM)*

Full 2D

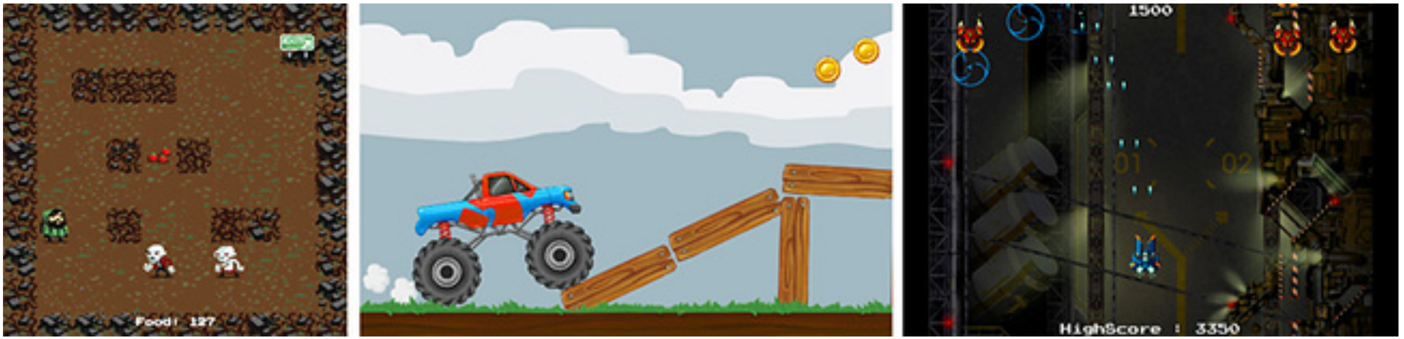


Figure 4 - Some examples of typical 2D game types

Many 2D games use flat graphics, sometimes called sprites, which have no three-dimensional geometry at all. They are drawn to the screen as flat images, and the game's camera has no perspective. For this type of game, you should start the editor in **2D** mode.

2D gameplay with 3D graphics

Figure 5 - A side scrolling game with 2D gameplay, but 3D graphics.

Some 2D games use 3D geometry for the environment and characters, but restrict the *gameplay* to two dimensions. For example, the camera may show a "side scrolling view" and the player can only move in two dimensions, but game still uses 3D models for the obstacles and a 3D perspective for the camera. For these games, the 3D effect may serve a stylistic rather than functional purpose. This type of game is *also* sometimes referred to as "2.5D". Although the gameplay is 2D, you will mostly be manipulating 3D models to build the game so you should start the editor in **3D** mode.

2D gameplay and graphics, with a perspective camera

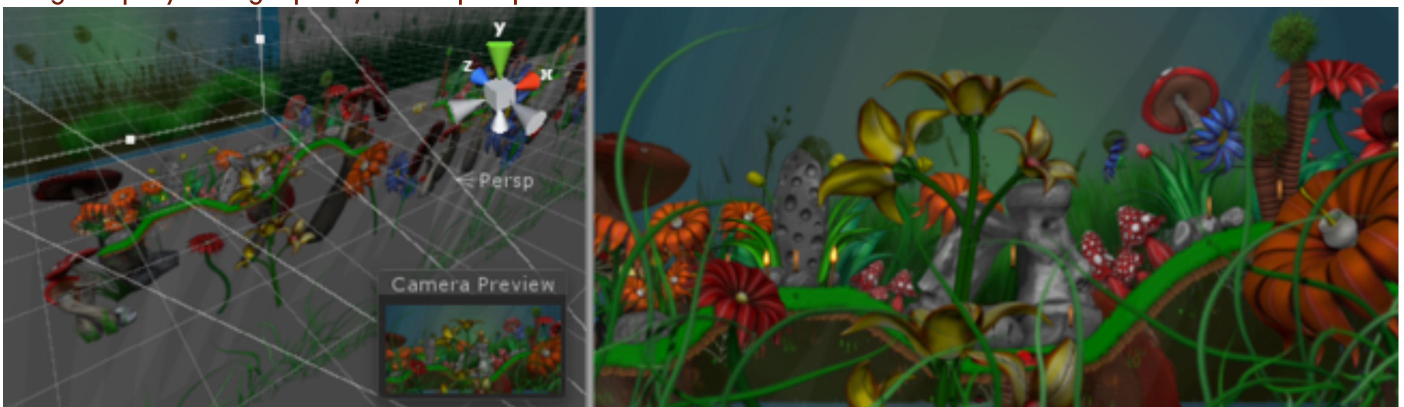


Figure 6 - A 2D "cardboard theatre" style game, giving a parallax movement effect

This is another popular style of 2D game, using 2D graphics but with a perspective camera to get a parallax scrolling effect. This is a "cardboard theater" style scene, where all graphics are flat, but arranged at different distances from the camera. It's most likely that **2D** mode will suit your development in this case. However, you will want to change your game camera's

projection mode to **Perspective** and the scene view mode to **3D**. (scene above from One Point Six Studio)

STARTING UNITY FOR THE FIRST TIME

Whenever you launch the Unity editor, the Home Screen displays. If you have no existing Unity projects on your computer, or Unity doesn't know where they are, it asks you to create a project. To get started, you can click on New project which will take you to the Home Screen's Create Project view.

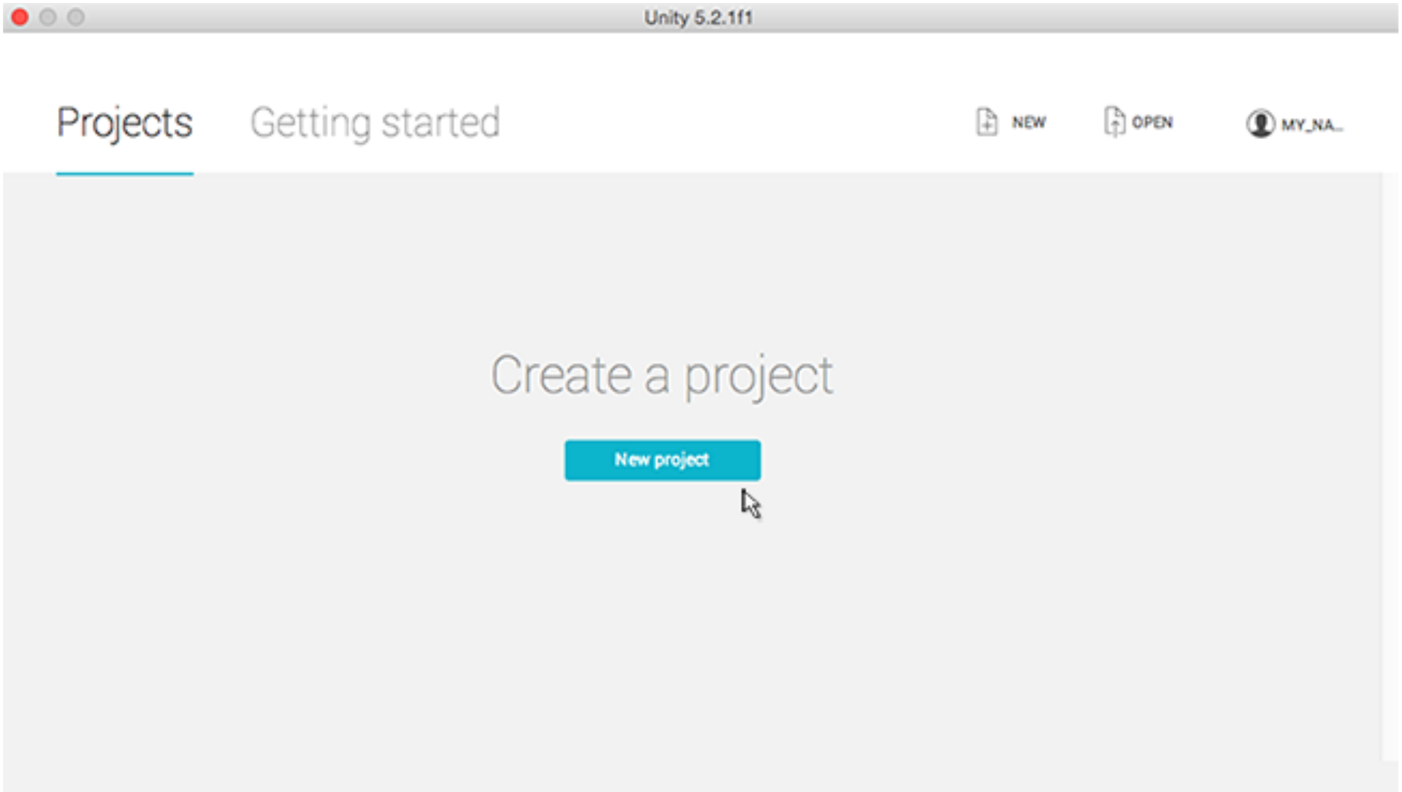


Figure 7 - The Home Screen displays on launch, click on "New Project" to get started OR in the top right corner of the Home Screen, select 'New' to see the Create Project view

From the Home Screen's Create Project view, you can name, set options, and specify the location of your new project.

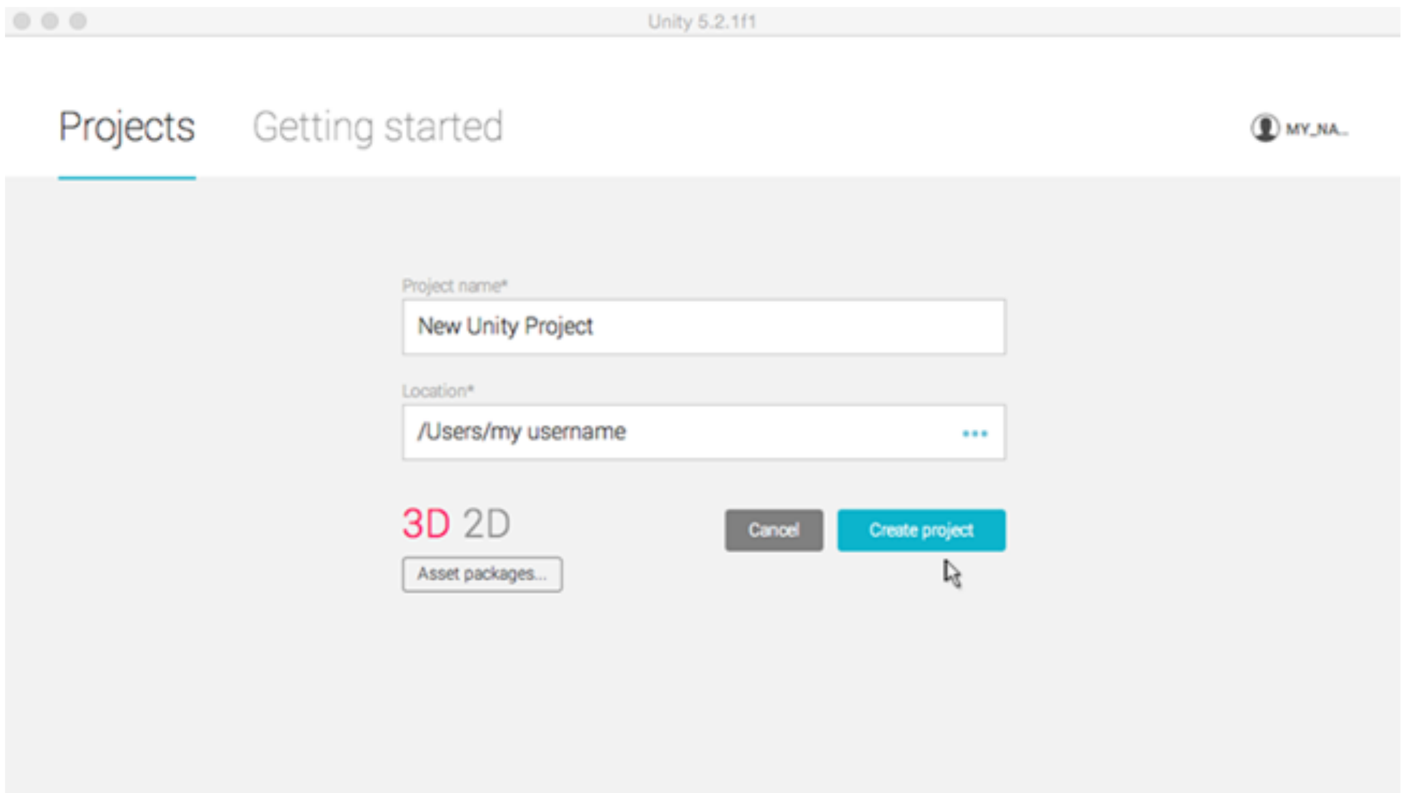


Figure 8 - The Home Screen's Capture Project view

To create a new project:

1. The name defaults to New Unity Project but you can change it to whatever you want. Type the name you want to call your project into the Project name field.
2. The location defaults to your home folder on your computer but you can change it. **EITHER** (a) Type where you want to store your project on your computer into the Location field. **OR** (b) Click on the three blue dots **'…'**. This brings up your computer's Finder (Mac OS X) or File Explorer (Windows OS).
3. Then, in Finder or File Explorer, select the project folder that you want to store your new project in, and select "Choose".
4. Select 3D or 2D for your project type. The default is 3D, coloured red to show it is selected. (The 2D option sets the Unity editor to display its 2D features, and the 3D option displays 3D features. If you aren't sure which to choose, leave it as 3D; you can change this setting later.)
5. There is an option to select Asset packages… to include in your project. Asset packages are pre-made content such as images, styles, lighting effects, and in-game character controls, among many other useful game creating tools and content. The asset packages offered here are free, bundled with Unity, which you can use to get started on your project. **EITHER:** If you don't want to import these bundled assets now, or aren't sure, just ignore this option; you can add these assets and many others later via the Unity editor. **OR:** If you do want to import these bundled assets now, select Asset packages… to display the list of assets available, check the ones you want, and then click on Done.
6. Now select Create project and you're all set!

LEARNING THE INTERFACE

Take your time to look over the editor interface and familiarize yourself with it. The main editor window is made up of tabbed windows which can be rearranged, grouped, detached and docked.

This means the look of the editor can be different from one project to the next, and one developer to the next, depending on personal preference and what type of work you are doing. The default arrangement of windows gives you practical access to the the most common windows. If you're not yet familiar with the different windows in Unity, you can identify the, by the name in the tab. The most common and useful windows are shown in their default positions, below:

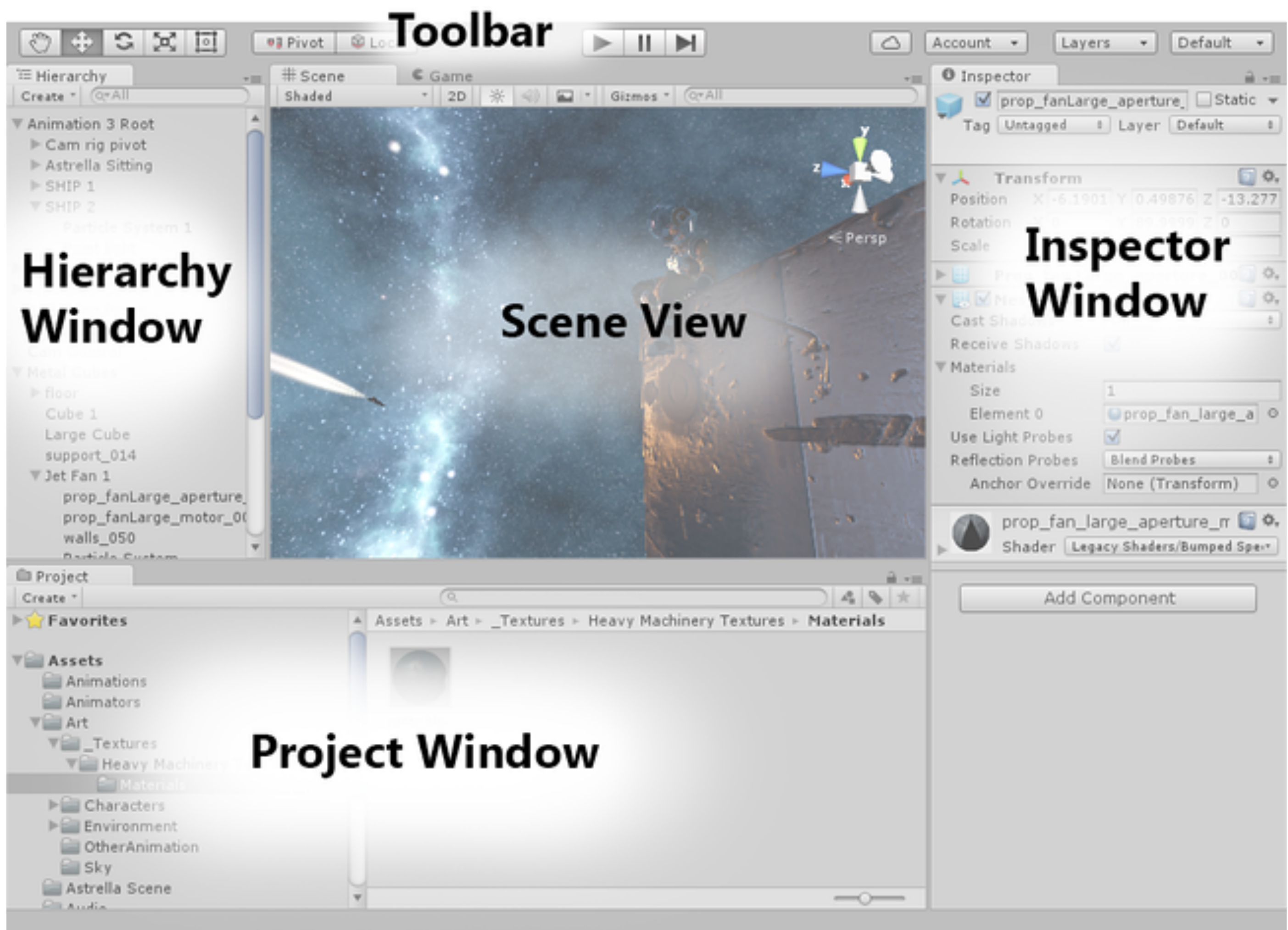


Figure 9 - Overview of Unity interface

The Project Window

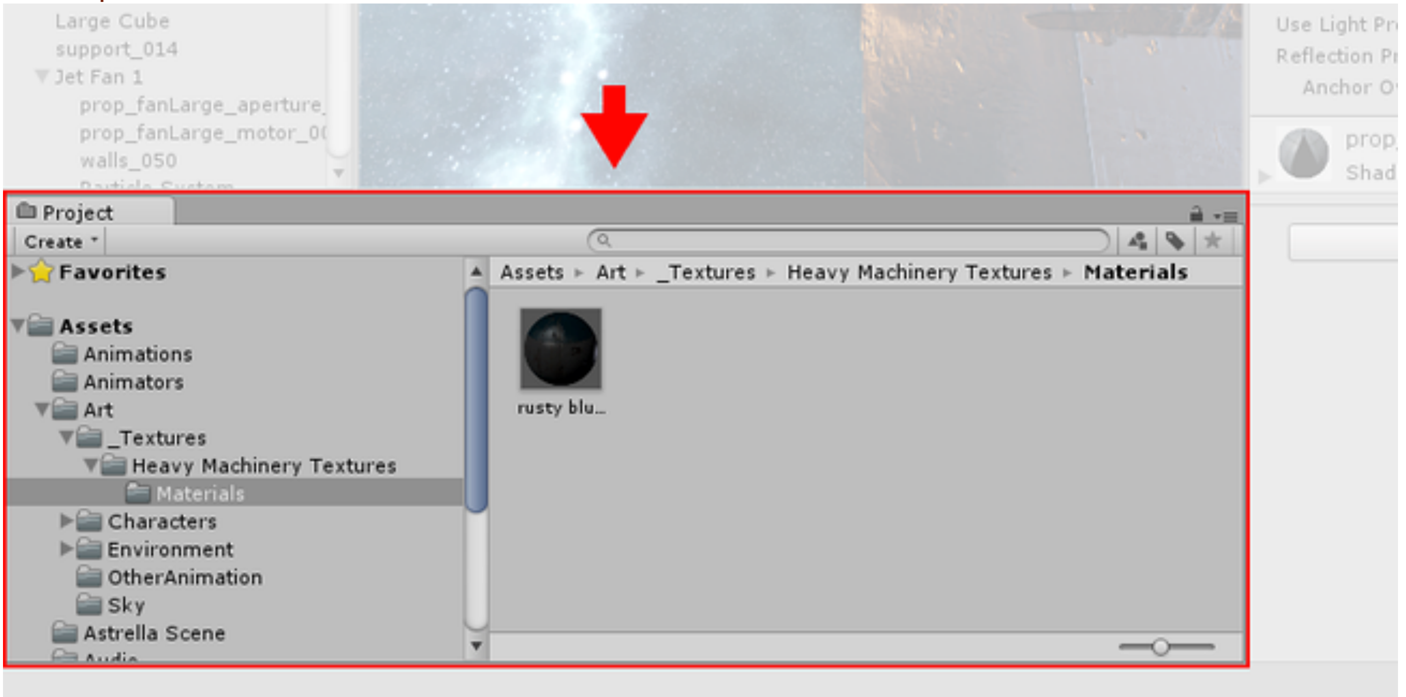


Figure 10 - The Project Window

The Project Window displays your library of assets that are available to use in your project. When you import assets into your project, they appear here.

The Scene View Window

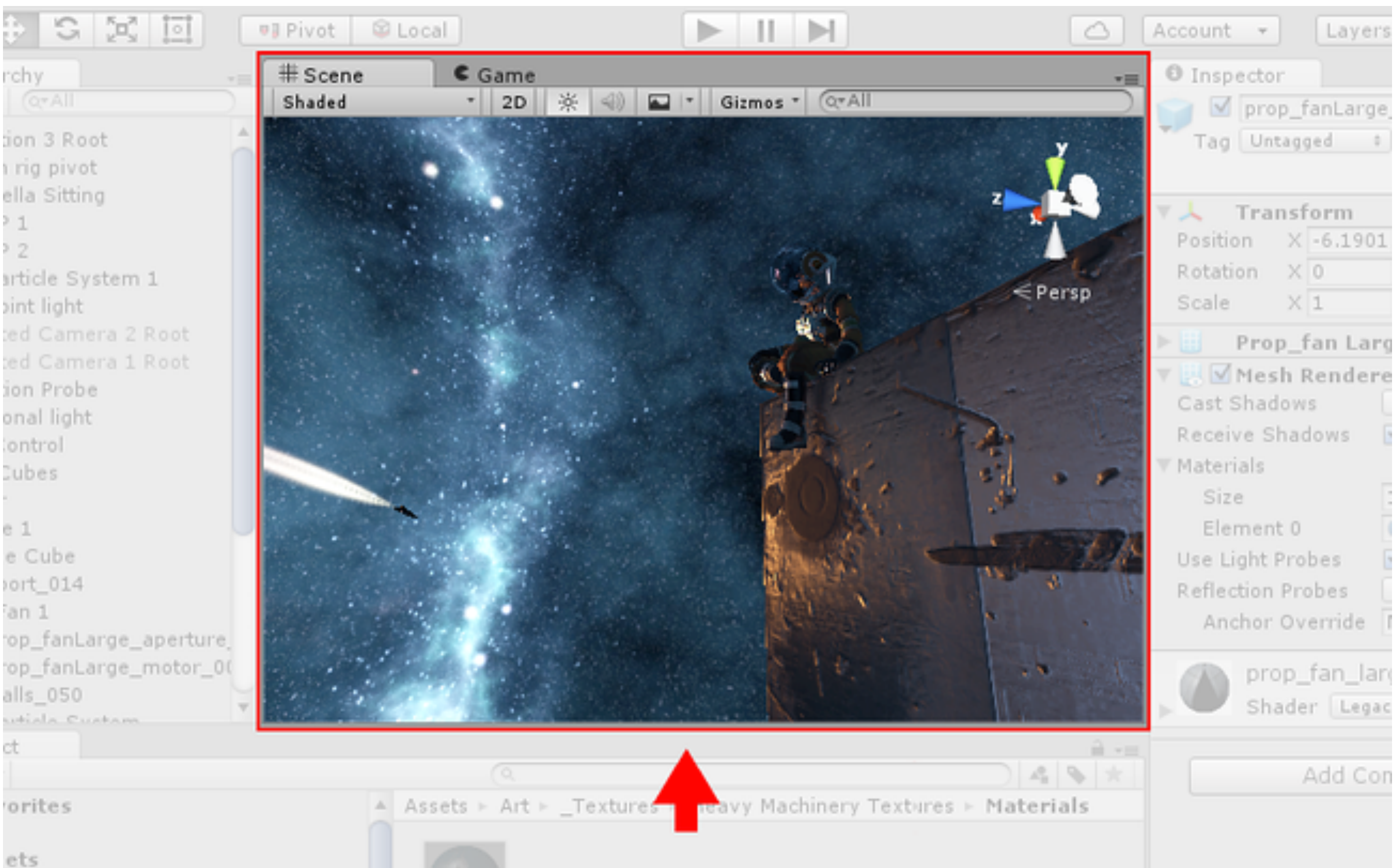


Figure 11 - The Scene View Window

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

The Hierarchy Window

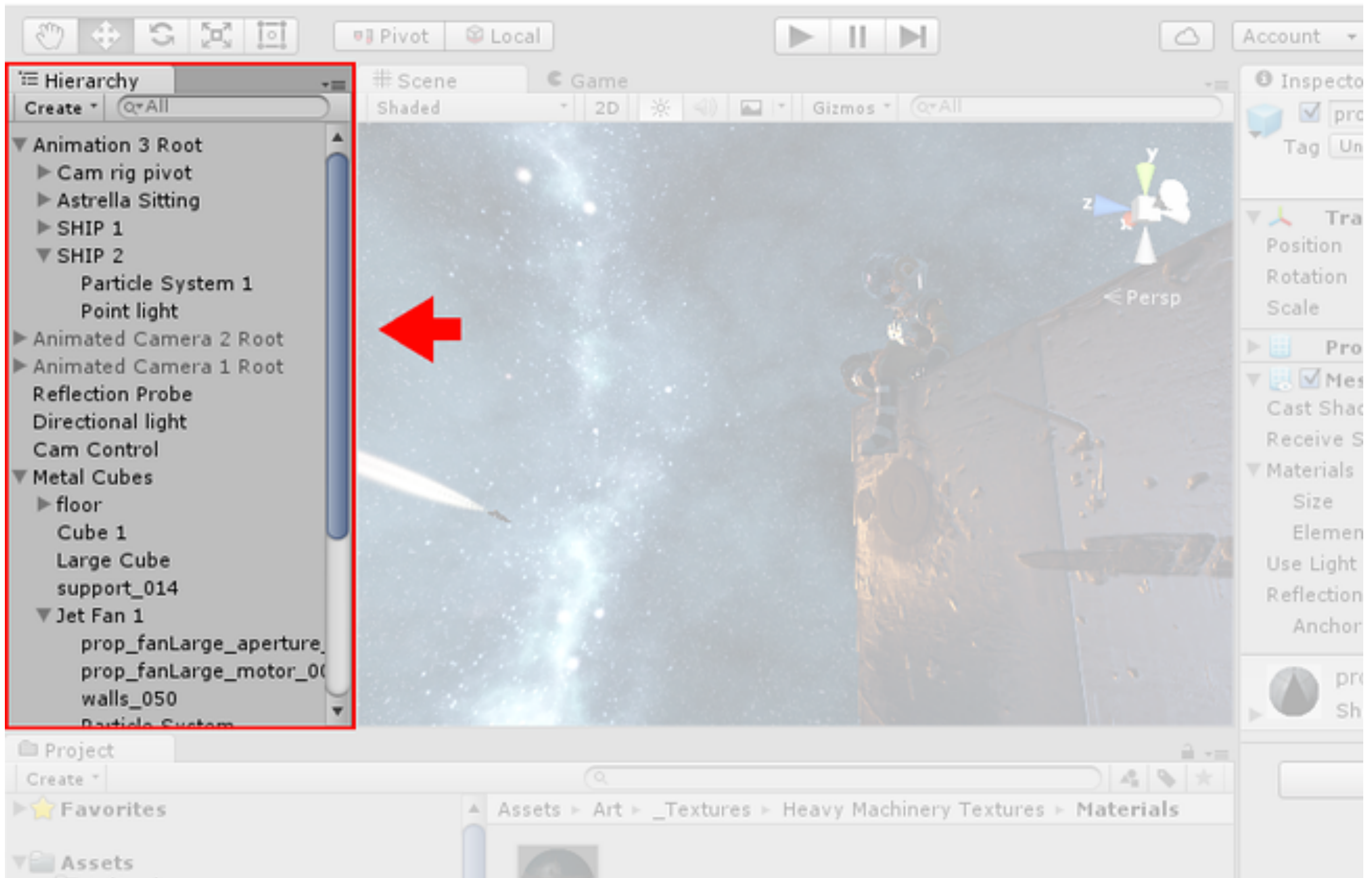


Figure 12 - The Hierarchy Window

The Hierarchy Window is a hierarchical text representation of every 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

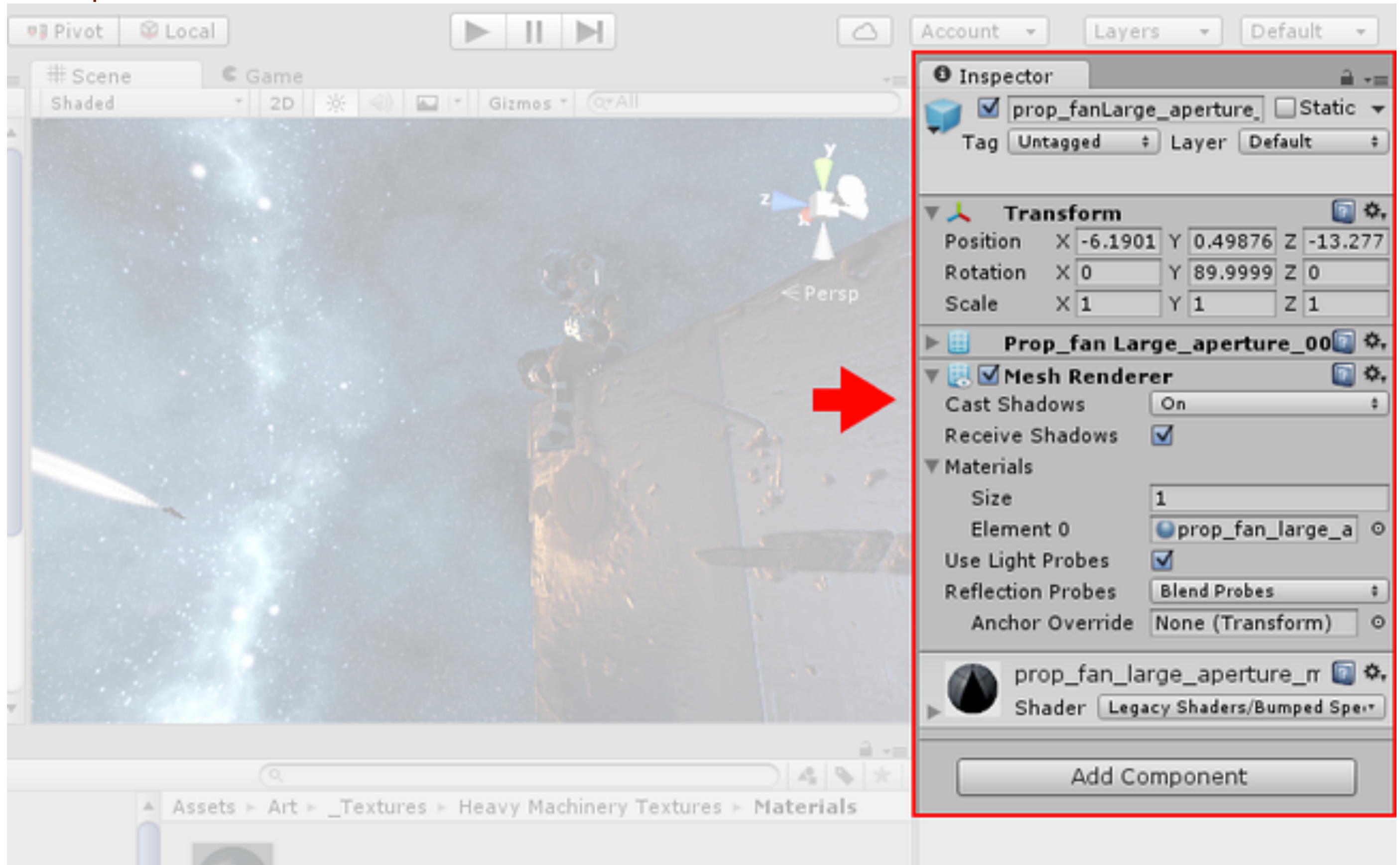


Figure 13 - The Inspector Window

The Inspector Window allows you to view and edit all the properties of the currently selected object. Because different types of objects have different sets of properties, the layout and contents of the inspector window will vary.

The Toolbar

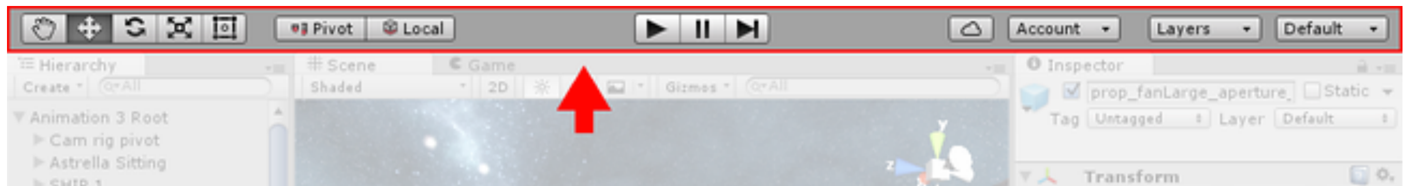


Figure 14 - The Toolbar

The Toolbar provides access to the most essential working features. On the left it contains the basic tools 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 Cloud Services and 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 that you can't rearrange.

LET'S MAKE A GAME

We are going to make a 2D side-scrolling racing-like game. The player will need to collect items and ammunition to continue in their progress. The background will be a small, simple looping background.

You can find the assets we will be using for this game in the folder UNITYGAMEASSETS.

Create a New Project

Project Name: CS4HS Endless Racer
Location:
3D or 2D: 2D

This will generate a new project for you.

The First Scene – Main Menu

Unity allows games to have multiple scenes to separate different logical sections into different hierarchy's. This makes it so only the needed resources are loaded in rather than having everything running in the background.

Step 1. The first thing we want to do is save the scene.

Go to **File -> Save Scene as…**. Enter '***MainMenuScene***' as the name.

You will notice that in the Project View you now have an object called ***MainMenuScene***.

Step 2. Next we will import some assets that we will use on the Main Menu.

Right click in the Project View and go to **Create -> Folder**. Enter '***Fonts***' as a name. Repeat and name the new folder '***Sprites***'

In UNITYGAMEASSETS, drag the font file ***NovaSlim.ttf*** into the ***Fonts*** folder in the Project View. Drag the image ***menuCharacter.png*** into the ***Sprites*** folder in Project View. Drag the image ***button.png*** into the ***Sprites*** folder in Project View.

Click on **NovaSlim** in the Project View. Make sure the following settings are the same. Click Apply.

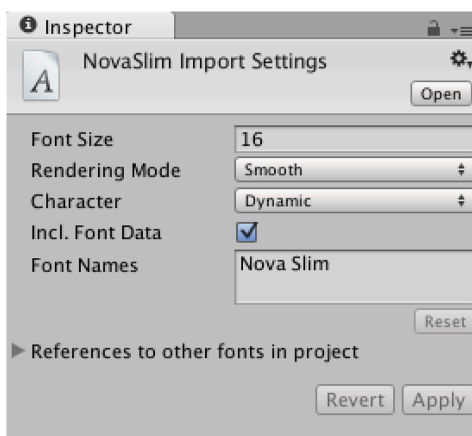


Figure 15 - NovaSlim Font Settings

Click on ***menuCharacter*** in the Project View. Make changes so the following settings match. Click Apply.

Pixels Per Unit: 400

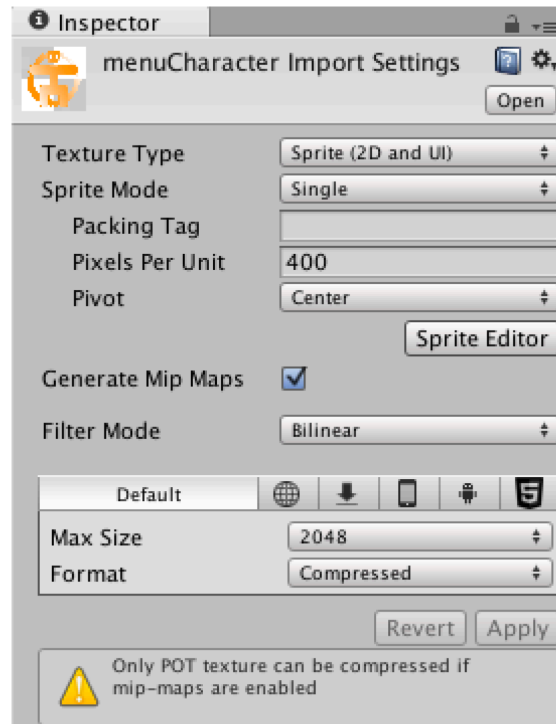


Figure 16 - menuCharacter Sprite Settings

Click on ***button*** in the Project View. Change the Sprite Mode from ***Simple*** to ***Multiple***. Click on Sprite Editor. Make changes so the follow settings match. Click the Sprite Editor and click Apply.

Border L: 8 **Border T**: 8
Border R: 8 **Border B**: 8

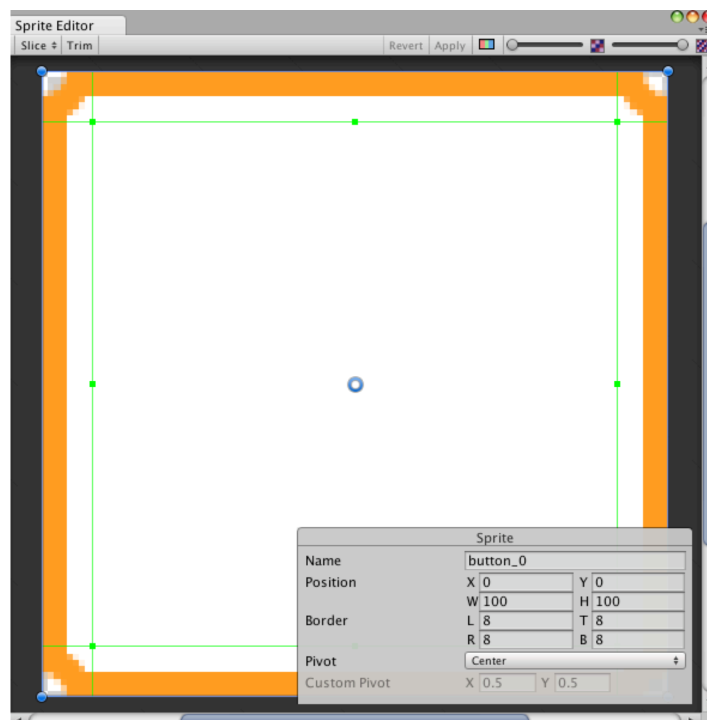


Figure 17 - button Sprite Editor Settings

Step 3. Now it is time to create the UI.

First click on **Game** at the top of the Scene View to switch to the Game View. At the top of the Game View change the aspect ratio from ***Free Aspect*** to ***16:9***.

This will make it so the game's aspect ratio is set to 16:9 so it is easier for us to design a UI.

Switch back to Scene View.

Click on Main Camera. Click on the Background and set the background to the following colour.

R:	26	**G**:	26	**B**:	26	**A**: 255

Right click in the Hierarchy View and go to ***UI -> Canvas***. This will create a Canvas and an EventSystem. The Canvas will be were we arrange the UI. The EventSystem is what Unity uses to handle Mouse and Keyboard events to do with the UI.

Click on Canvas in the Hierarchy View so it is selected. Right click on the Canvas and go to ***UI -> Panel***. This will create a grey panel in your Scene View and there should now be a Panel indented under the Canvas. This means that the Panel is a child of the Canvas.

Click on the Panel in the Hierarchy View. Change the name from ***Panel*** to ***Image Panel***. This can be done at the top of the Inspector or by right clicking on the object in the Hierarchy View and selecting ***Rename***.

In the Inspector View you can see that a Panel has an Image (Script). This is means that the Panel is actually just an Image object with a fancy name. The current default Source Image however isn't what we want it to be. Click on the small circle next to the default Source Image. This will open a dialog where you can see all the available sprites. Select ***menuCharacter***.

Now you will see a faded out version of the ***menuCharacter***. We want this to actually be a solid opaque image on the right hand side of the menu. To do this we need to change settings for both the ***Rect Transform*** and ***Image*** of the script.

Rect Transform
Anchors

Min	**X**:	0.5	**Y**:	0
Max	**X**:	1	**Y**:	1

Image (Script)
Background

R:	255	**G**:	255
B:	255	**A**:	255

Preserve Aspect
Ticked

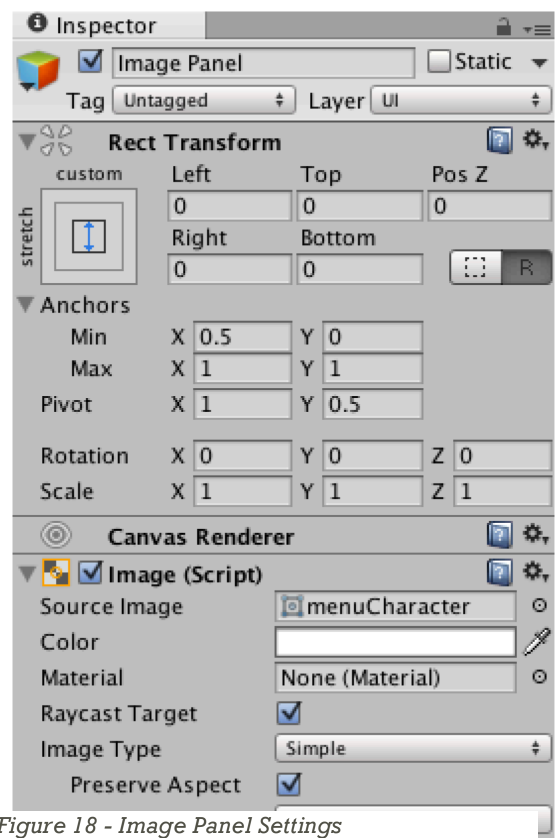


Figure 18 - Image Panel Settings

Now on to over half of the UI.

Select the Canvas again and create a new Panel. Call this Panel ***Info Panel***.

Set apply the following settings for the new Panel.

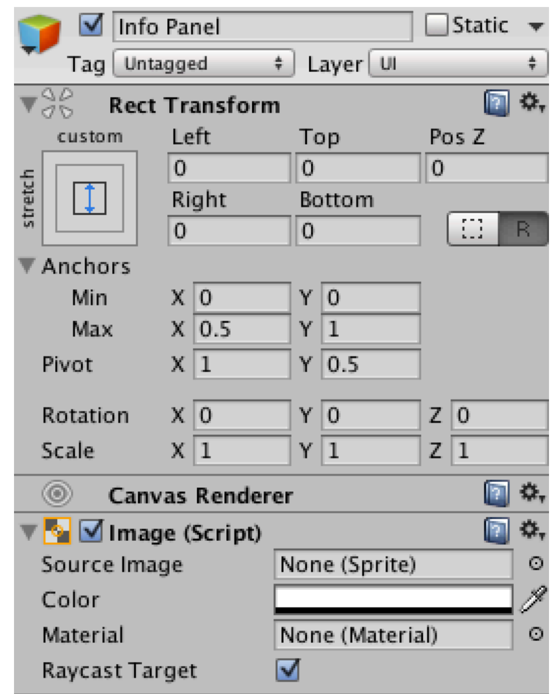


Figure 19 - Info Panel Settings

Rect Transform
Anchors

Min	**X**:	0	**Y**:	0
Max	**X**:	0.5	**Y**:	1

Image (Script)
Source Image

None

Background

R:	0	**G**:	0
B:	0	**A**:	0

With the ***Info Panel*** selected, right click on it and go to ***UI -> Text***. This will create a Text object as a child of the Info Panel. Change the name of this to ***Title***. We want to fix this to the top of the screen in the Info Panel so will will alter the **Rect Transform**. First change the Anchors then change the Positions.

Rect Transform
Anchors

Min	**X**:	0	**Y**:	0.75
Max	**X**:	1	**Y**:	1

Position

Left:	0	**Top**:	0
Right:	0	**Bottom**:	0

Now we want to edit the ***Text (Script)***. Click on the small circle selector for the Font and change it to ***NovaSlim***. **Centre** the alignment both horizontally and vertically. Set the Horizontal Overflow to ***Wrap*** and the Vertical Overflow to ***Truncate***. Set the colour to the following:

R:	255	**G**:	255	**B**:	255	**A**:	255

Set the Text to ***CS4HS Endless Runner***. Select ***Best Fit*** and set the size to ***10*** to ***100***.

Create another Text object as a child of ***Info Panel***. Call it ***Info Text***. Set the following settings for the new Text object.

Rect Transform
Anchors

Min	**X**:	0.1	**Y**:	0.3
Max	**X**:	1.9	**Y**:	0.7

Position

Left:	0	**Top**:	0
Right:	0	**Bottom**:	0

Change the font to ***NovaSlim***. **Centre** the alignment both horizontally and vertically. Set the Horizontal Overflow to ***Wrap*** and the Vertical Overflow to ***Truncate***. Set the colour to the following:

R:	255	**G**:	255	**B**:	255	**A**:	255

The text here will be a simple description of the game. You can set this at the end. For now, just put together some template text. Select ***Best Fit***.

Lastly a button to start the game.

Click on ***Info Panel***, right click and go to ***UI -> Button***. Rename the button to ***Start Button***. We want this button to be in the bottom left. It will expand when we hover over it and click it. Set the following settings for the button.

Rect Transform
Anchors

Min	**X**:	0.25	**Y**:	0.1
Max	**X**:	0.75	**Y**:	0.2

Position

Left:	0	**Top**:	0
Right:	0	**Bottom**:	0

Change the ***Image (Script)*** to the button sprite we created earlier – ***button_0***. Change the Image Type to ***Sliced*** and make it ***Fill Center***.

In the ***Button (Script)*** change the Transition to ***Animation***. Click ***Auto Generate Animation***. Call the animation ***StartButtonAnimation***. Change the Navigation to ***None***.

Click on ***Window -> Animation*** from the top menu to open the Animation View. Make sure you have the ***Start Button*** object selected in the Hierarchy and click on the Animation View. Click the dropdown in the top left to change from ***Normal*** to ***Highlighted***. Click on the red **Record** button. In the Inspector View change the following settings.

Rect Transform
Anchors

Min	**X**:	0.1	**Y**:	0.05
Max	**X**:	0.9	**Y**:	0.25

Click the red **Record** button to stop recording the animation. Change from ***Highlighted*** to ***Pressed*** and record the following animation.

Rect Transform
Anchors

Min	**X**:	0	**Y**:	0
Max	**X**:	1	**Y**:	0.3

Stop recording and change back to the Scene View.

Click on the Play Button in the ***Toolbar*** and look and see when happens when you interact with the button.
Click Play again when you are done testing it out.

Step 4. Now on to some basic programming.

First we want to set the basic code editor. Click on ***Edit -> Preferences***. Under ***External Tools*** make sure the External Script Editor is ***Visual Studio 2015***.

Close the preferences window.

We will now add our first custom script to the project. Click on the ***Canvas*** and in the Inspector View, click ***Add Component***. With this we can search for and add any component/script from Unity or that we have created and is in the project. Type ***StartMenu***. Click ***New Script***, make sure the Language is set to ***C Sharp***. Click ***Create and Add***.

You will now see a new component in Inspector View and Project View. In the Project View, double click on ***StartMenu*** and Visual Studio 2015 Community for Unity will open. This may take a couple of seconds to open up so give it a little bit.

You will see the following when it opens.

```
using UnityEngine;
using System.Collections;

public class StartMenu : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }
}
```

Code Sample 1 - StartMenu

The ***Start*** function will run once when the GameObject with the script is created. The ***Update*** function will run on every new frame of the game. Other functions exist such as ***FixedUpdate*** which will be run every 20 milliseconds; this is normally used for games with physics to ensure the physics are calculated correctly. There are many other default functions that you can use along with writing your own functions.

We will now make some simple changes to test out this functionality. Make the following changes to ***Start***.

```
    void Start () {
        Debug.Log("Hello, world!");
    }
```

Code Sample 2 - StartMenu

And the following changes to ***Update***.

```
    void Update () {
        Debug.Log("I'm Updating");
    }
```

Code Sample 3 - StartMenu

Save the file and switch back to Unity. You will see that Unity will take a second or two and it will compile the code you have just written.

Click ***Play*** and check out all the spam in the Console. If your Console isn't visible turn it on from the Window menu.

Click ***Play*** again to stop the spam.

Change back to the code editor and undo the changes to ***Update*** but leave ***Start*** as it is.

Add the following two ***using*** lines at the top of the file.

```
using UnityEngine.UI;
using UnityEngine.SceneManagement;
```

Code Sample 4 - StartMenu

Now will will add our first function. This function will be used as the on click function for the start button. For now, however it will just post a message to the console until we have the next scene done. Add function below ***Update*** but before the final **} brace**.

```
    public void StartGame ()
    {
        Debug.Log ("Start Game Clicked - Start New Scene");
        // TODO: Remove comment below once GameScene is made.
        // SceneManager.LoadScene ("GameScene");
    }
```

Code Sample 5 - StartMenu

Now we have the function for the button but we still need to link it up to the button click. Switch back to Unity. In the Inspector View with the ***Start Button*** selected you will see an ***On Click ()*** section. Click the **+**. From the Hierarchy, drag the ***Canvas*** to the ***None*** object field in the ***On Click ()*** section. Make sure not to click the ***Canvas*** as this will swap to it, just drag. Now if you click on ***No Function*** you can see all the scripts that are attached to Canvas and their public functions. Select the ***StartMenu -> StartGame()*** function.

Click ***Play*** and see your button in action.

Click ***Play*** again when your done checking out the button.

Save the Scene.

The Second Scene – The Game Proper

Now we have the first scene done for the most part. Time to move on to the second scene; one with some gameplay.

Step 1: Create a new scene and save it.

Go to ***File -> New Scene*** to create a new scene. You will now notice a very empty Hierarchy and Scene View. Save the Scene with the name ***GameScene***.

Step 2: Some more assets to import.

Create a new folder in Project View called ***Textures***.

From the UNITYGAMEASSETS folder, drag ***bottomBarrier.png***, ***topBarrier.png*** and ***track.png*** into the Textures folder.

In the Project View select the three newly imported images. Change the settings to the following.

Texture Type: Texture
Wrap Mode: Repeat
Filter Mode: Trilinear

Click **Apply**.

From the UNITYGAMEASSETS folder, drag ***knowledge.png***, ***trackBarrier.png***, ***obstacles.png***, ***vehicle.png***, and ***ammo.png*** into the Sprites folder.

For ***knowledge***. Set Pixels Per Unit to ***400***.
For ***trackBarrier***. Set Pixels Per Unit to ***100***.
For ***vehicle***. Set Pixels Per Unit to ***400***.
For ***ammo***. Set Pixels Per Unit to ***400***.

For ***obstacles***. Change Sprite Mode to ***Multiple*** and click ***Sprite Editor***.

In the Spite Editor draw boxes around each of the obstacles. You can have some extra around the obstacles but make sure it is entirely surrounded. Once all have been drawn, click the ***Trim*** button. Rename each of the obstacles in the Sprite window in the Sprite Editor. Name them ***obstacle 1x1***, ***obstacle 1x2***, ***obstacle 1x3***, ***obstacle 1x4***, ***obstacle 2x1***, ***obstacle 2x2*** and ***obstacle 3x1*** in line with their sizes. Click ***Apply*** and close the Sprite Editor. Click ***Apply*** in the Inspector View.

Step 3: Time to create the vehicle and track.

First we will create an empty game object to store all of the components of the vehicle and the track. While you wouldn't normally have a setup where the background is a part of the moving vehicle, for the scrolling background technique that we will be using this is a valid method.

Right click in the Hierarchy View and go ***Create Empty***. This is now a game object that is … well … empty. Rename this to ***Vehicle***. Make sure the position of this is the Origin (x = 0, y = 0, z = 0).

Select ***Vehicle*** and right click it. Go to ***2D Object -> Sprite***. Change the name of this to ***VehicleGFX***. This has a ***Sprite Renderer***. We want to set the Sprite of this to the ***vehicle*** sprite.

Click ***Sorting Layers -> Add Sorting Layer***. Create the following layers under ***Default*** in the given order.

Tutorial
Obstacles
Collectables
Player

Sorting Layers determine how sprites will be ordered. The lower the sorting layer, the closer the sprite is to the camera. Set the sorting layer to ***Player***. Set the Order in Layer to ***1***,

Set the Position of the ***VehicleGFX*** to (X = -5, Y = 0, Z = 0). We will be moving the parent object and the graphics will stay relative it it.

Click ***Add Component*** and search for and add a ***Trail Renderer***. Set it to not cast shadows. Have a time of ***2.5***, a start width of ***1***, and an end width of ***0***. Set the colours of the trail however you please.

Now for the track. Select ***Vehicle*** and right click it. Go to ***Create Empty***. Change the name of this to ***BackgroundGFX***. This will contain the three parts of the background, ***topBarrier***, ***track***, and ***bottomBarrier***. Make sure the position of this is the origin.

Select the ***BackgroundGFX*** and right click it, go to ***3D Object -> Quad***. Do this twice more. Rename them ***Top Barrier***, ***Bottom Barrier***, and ***Track***.

From the Project View drag the appropriate texture from the Texture folder to the appropriate Quad in the Hierarchy View.

The the positions and scale for the three as follows.

Top						
Position	**X**:	0	**Y**:	4.5	**Z**:	0
Scale	**X**:	20	**Y**:	1	**Z**:	0
Bottom						
Position	**X**:	0	**Y**:	-4.5	**Z**:	0
Scale	**X**:	20	**Y**:	1	**Z**:	0
Track						
Position	**X**:	0	**Y**:	0	**Z**:	0
Scale	**X**:	20	**Y**:	8	**Z**:	0

Now we need to parent the camera to the vehicle object. The camera will follow the vehicle around. In the Hierarchy View drag the ***Main Camera*** onto the ***Vehicle*** game object.

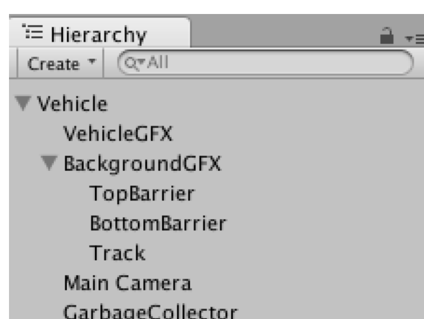


Figure 20 - Vehicle Hierarchy

Step 4: Coding some movement for the vehicle.

Select the ***Vehicle*** game object and add a new script to it. Call the script ***VehicleMovement***.

Double click on the new script to open it in the script editor.

The script will start off looking the same as the one for the ***StartMenu***.

First will need to set up some variables to use in the script. Put these above the ***Start*** function.

```
private const float vehicleGFXOffset = -5f;

public float speed = 0f;
public float acceleration = 2.5f;

public GameObject vehicleGFX;

public float startingDistance = 0f;
public float distanceTravelled = 0f;
```

Code Sample 6 - VehicleMovement

The ***f*** indicates that the number is a floating point number.

vehicleGFXOffset is the X offset we set previously. This will be used for updating the location of the graphics to indicate speed.
speed is how fast we are travelling.
acceleration is how fast we can accelerate.
vehicleGFX is a reference to the graphics of the vehicle so we can control it.
startingDistance is where we start each round.
distanceTravelled is how far we have travelled from the starting position.

For this we do not need a ***Start*** function. We could have one where we set the values of the variables but pre-initalising them works fine. Since the variables are public we will be able to see them in the Inspector View of the Unity Editor. If the variables are set in the editor, they will override the ones you have set in the script for the game object they are attached it.

We will modify the ***Update*** function. In it we will get the input from the user, increase or decrease the speed of the player, change the position of the player, change the position of the vehicle graphics and update the distance travelled.

```
void Update ()
{
    speed += acceleration * Input.GetAxis ("Horizontal") *
                  Time.deltaTime;
    speed = Mathf.Clamp (speed, 0f, 10f);
    transform.Translate (new Vector3 (
        speed,
        0,
        0f
    ) * Time.deltaTime);
```

```
        Vector3 position = vehicleGFX.transform.localPosition;

        position.y += Input.GetAxis ("Vertical") * Time.deltaTime *
                                  speed;
        position.y = Mathf.Clamp (position.y, -3, 3);
        position.x = vehicleGFXOffset + (speed / 2);

        vehicleGFX.transform.localPosition = position;

        distanceTravelled = transform.position.x - startingDistance;
    }
```

Code Sample 7 - VehicleMovement

Time.deltaTime is a variable from Unity that stores how long it has been since the last frame was renderer.
Input.GetAxis() is a function that gets the fractional value of how far the input axis has been pushed.
Mathf.Clamp() is a function that takes a number, and an upper and lower bounds. It returns the number if it is between the bounds or the bound that it exceeds.

We will add another function here as well for when we are resetting the vehicle if we stop at a track barrier.

```
    public void PassedBarrier ()
    {
        startingDistance = transform.position.x;
        distanceTravelled = 0f;
        speed = 0f;
    }
```

Code Sample 8 - VehicleMovement

Save the code and return to Unity. Before hitting the play button and testing the code, have a look at the Inspector View for the new script attached to the ***Vehicle***. You will see the variables from the code everything is set except for the ***Vehicle GFX***. Drag the ***VehicleGFX*** from the Hierarchy view into the empty variable.

Click ***Play*** and test your game. You will note now that this scene loads. This is because it is the one we are currently editing. You will also note as we control the vehicle not much happens. The background isn't scrolling yet. On to that next.

Step 5: Time for scrolling backgrounds.

Select ***Top Barrier***, ***Bottom Barrier*** and ***Track***. Click ***Add Component*** and create a new script called ***ScrollingBackground***. Open the script in the code editor.

Here we will put in the ***Update*** function a modification to the texture to make it loop. The textures you have been provided with are able to be looped.

```
void Update ()
{
    MeshRenderer mr = GetComponent<MeshRenderer> ();
    Material mat = mr.material;
    mat.mainTextureOffset = new Vector2 (
        transform.position.x, 0
    ) / transform.localScale.x;
}
```

Code Sample 9 - VehicleMovement

Return to Unity. Click ***Play***.

Step 6: Creating obstacles and other collectables.

First we will create some scripts that we will need. To create a script without adding to a game object. Right click in the Project View and go to ***Create -> C# Script***.

Create an ***Obstacle*** script. We don't need this script to do anything. It is just to indicate that the object is an obstacle.

Create a ***Collectable*** script. This one doesn't need to do anything also but it needs to store some more information. Open up the script. You can delete the ***Start*** and ***Update*** function and replace it with the following.

```
public enum PickupType
{
    AMMO,
    KNOWLEDGE
}

public PickupType type;
```

Code Sample 10 - Collectables

This will allow us to use the same ***Collectable*** script on both the ammo and knowledge game objects.

Now to setup the obstacles and collectables.

From the Project View expand ***obstacle***, drag one of each of the obstacle types into the Scene View. This will create a new Sprite in the Scene for each of the obstacles. Set the position of each of these to the origin. Add the ***Obstacle*** script to each of these. Also add a ***Box Collider 2D*** and a ***Rigidbody 2D***. Note the ***2D*** in the name. Since we are working in 2D space we need to make sure we use the 2D colliders and rigidbodies to make sure the physics engine knows what's happening.

Set the ***Box Collider 2D*** to ***Is Trigger***. Set the ***Rigidbody 2D*** to ***Is Kinematic***.

Create a new Folder in the Project View called ***Prefabs***. Prefabs are game objects that are prefabricated, they are already set up and you can create them easily by copying them.

Separately drag each of the obstacles from the Hierarchy View to the ***Prefabs*** folder. You will note the colour of the text changes. This is indicating that it is linked to a Prefab that exists in the project.

After you have created all the prefabs, you can delete them from the ***Hierarchy View***.

Drag both the ***ammo*** and ***knowledge*** sprites into Scene View and set them to the Origin. Add the ***Collectable*** script, ***Box Collider 2D*** and ***Rigidbody 2D*** to them.

Set the ***Box Collider 2D*** to ***Is Trigger***. Set the ***Rigidbody 2D*** to ***Is Kinematic***. Set the appropriate ***PickupType*** for both of the components.

Make prefabs out of them and delete them from the Hierarchy.

Step 7: Colliding with the obstacles and collectables.

Now we need to set up collisions with the obstacles and collectables. Select the ***VehicleGFX*** and add a ***Circle Collider 2D*** and a ***Rigidbody 2D***.

Set the ***Circle Collider 2D*** to ***Is Trigger***. Set the ***Rigidbody 2D*** to ***Is Kinematic***.
Also on the ***Circle Collider 2D*** set the Radius of the collider to be ***0.85***.

Add a script to the ***VehicleGFX*** called ***VehicleGFX***. Since it is the graphics that actually runs into the obstacles and collectables it needs to handle them.

For this we do not need an ***Update*** function as it is handled by it's parent ***Vehicle***. We do however need a ***OnTriggerEnter2D*** function. This is called when ever the object enters a RigidBody2D and Collider2D combination with the trigger set. Here we will also fix an issue with the TrailRenderer.

In the ***Start*** function we get the TrailRenderer that is attached to the game object this script is running on. We set the sorting layer so that it is on the same as the vehicle graphics and the order in the layer so that it is behind the vehicle.

In the ***OnTriggerEnter2D*** function we look at what the thing we collided with is, log a message to the console and then delete the object in 0.1 second.

ClearTrail will be used later when we stop at a track barrier. It calls a Coroutine. A Coroutine is a function in Unity that can run over multiple frames but it must return control by yielding when it has finished some work. If it doesn't yield the game will lock up until it returns. When the next frame starts processing control will be returned to the Coroutine where it left yielded.

```
    public VehicleMovement vehicleMovement;
    private TrailRenderer trailRenderer;

    // Use this for initialization
    void Start ()
    {
        trailRenderer = GetComponent<TrailRenderer> ();
        trailRenderer.sortingLayerName = "Player";
        trailRenderer.sortingOrder = 0;
    }

    void OnTriggerEnter2D (Collider2D other)
    {
        if (other.gameObject.GetComponent<Obstacle> () != null) {
            vehicleMovement.speed = 0f;
            Debug.Log("Hit Obstacle");
        } else if (other.gameObject.GetComponent<Collectable> () != null) {
            if (other.gameObject.GetComponent<Collectable> ().type ==
                        Collectable.PickupType.KNOWLEDGE) {
                Debug.Log("Collected Knowledge");
            } else if (other.gameObject.GetComponent<Collectable> ().type ==
                        Collectable.PickupType.AMMO) {
                Debug.Log("Collected Ammo");
            }
        } else {
            Debug.Log("Hit Track Barrier");
            return;
        }

        Destroy (other.gameObject, 0.1f);
    }

    public void ClearTrail ()
    {
        StartCoroutine (ResetTrail ());
    }

    private IEnumerator ResetTrail ()
    {
        var trailTime = trailRenderer.time;
        trailRenderer.time = 0;
        yield return new WaitForEndOfFrame ();
        trailRenderer.time = trailTime;
    }
```

Code Sample 11 – VehicleGFX

Save your code and return to Unity. The ***Vehicle Movement*** variable needs to be set up. To do this drag the ***Vehicle*** from the Hierarchy View into the variable. Unity will know that you are referring to the ***VehicleMovement*** script that is attached to ***Vehicle***.

Drag a couple of the prefabs of obstacles and collectables into the scene and see what happens when you hit ***Play***.

Creating every single obstacle and collectable by hand could get a bit tedious so we should do something about that later.

Connecting the Two Scenes

To connect the two scenes together we need to tell Unity that we want to package the two scenes together.

Step 1: Save the current scene.

Make sure you save the current scene.

Step 2: Setting the Build Settings.

Click ***File -> Build Settings***. In this window you will be able to see the scenes your game will have in it when you publish your game. Currently that is none unfortunately. To add scenes, drag the scenes from your Project View into the Build Settings window.

Make it so that the ***MainMenuScene*** scene is at the top and ***GameScene*** is at the bottom. Thus the menu is 0 and game is 1. The scene that is 0 will be the one that launches when the game starts up after it is published.

While in the Build Settings window click on the ***PC, Mac and Linux Standalone***. Make sure the Target Platform is the one you are currently using – ***Windows***. Click on ***Player Settings***. In the Inspector View you will now see settings for the PC, Mac and Linux Standalone Player. Set the Company Name and Product Name as you want. Set the Default Icon to be the ***menuCharacter***.

Set the resolution to not default to full screen or native resolution.
Make it so the window is not resizeable.
Make it so the only supported aspect ratio is ***16:9***.

Step 3: Return to the Start Button.

Edit your ***StartMenu*** script to remove the TODO and the comment (//) from the line to that will load in the new scene.

```
public void StartGame ()
{
    Debug.Log ("Start Game Clicked - Start New Scene");
    SceneManager.LoadScene ("GameScene");
}
```

Code Sample 12 – StartMenu

Step 4: Test it. Build It. Play It.

Switch to the ***MainMenuScene*** by double clicking on it. Test the button to make sure it works.

Once it is working. Now it is time to build it.

Go to ***File -> Build Settings***. Click ***Build & Run***. You will be asked where you want to save the it. Select the current location it is suggesting. Wait a few seconds to a minute and your game should launch in as its own program.

ACKNOWLEDGEMENT

The ***Installing Unity***, ***2D or 2D Projects***, ***Starting Unity for the First Time*** and ***Learning the Interface*** sections were taken from the http://docs.unity3d.com website. Used under Fair Use for an Education Purpose.