Tutorial – Scripting: Ocean Rescue part 1

For this tutorial we will start creating the Ocean Rescue game that forms your first piece of assessment for this course.

The project we make in this session will start of simple, and we'll add more functionality to it over the next several lessons until we have a finished Ocean Rescue game.

There will probably be a few times where you want to expand on what is presented in these tutorials and do your own thing. We encourage you to explore Unity and customise the game you make, but we also recommend keeping a version of your project that strictly follows the tutorial. That way if your own game suddenly breaks you'll always have a version that matches the tutorial and can be used to pass the assignment.

With that said, let's start making our Ocean Rescue game…

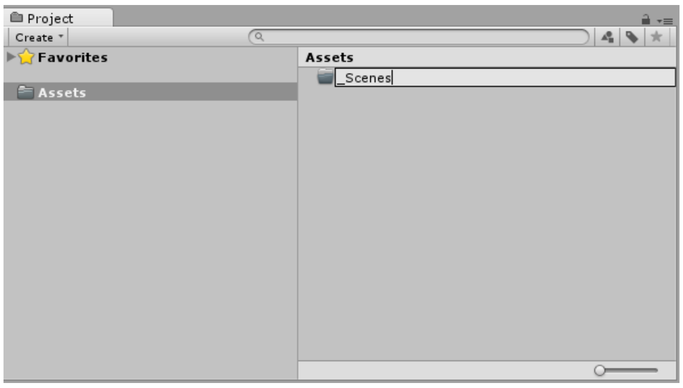
Creating a Simple Scene:

The first thing we need is a new project and scene.

Create a new project and give it a name. (I suggest "Ocean Rescue"). You should now see an empty project.

The first thing we will so is save the scene. In the Project Hierarchy window, right-click on the *Assets* folder and select **Create** -> **Folder**. This will create a new folder in the Assets folder. Call the new folder "*_Scenes*".

Tip: By calling your Scenes folder "_Scenes" it will always appear at the top of the hierarchy, making your Unity scenes easy to find.



Now select **File** -> **Save Scene**. A dialog box will appear asking for the location and name for your new scene. Make sure to save it under the *_Scenes* folder, and call it something like *Game*.

Your scene should now appear in the hierarchy window.

Importing Assets:

For this game we are using some free (Creative Commons) 3D assets available from http://www.kenney.nl/assets/. To build our game, we'll use models from four asset packs available from this site. To save you searching for them, we've made them available from the AIE website.

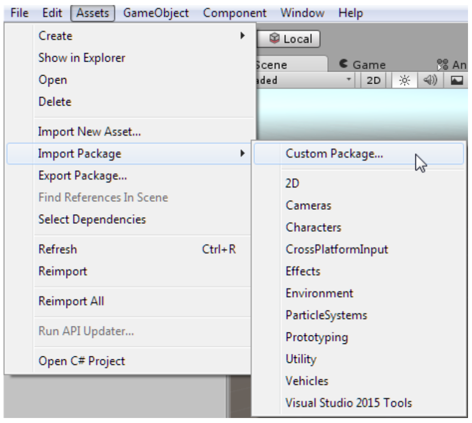
Download these four files from the AIE course page:

- blockyCharacters.unitypackage
- Castle Kit.unitypackage
- naturePack_unity.unitypackage
- watercraftPack.unitypackage

If you find some 3D assets on the *Kenny* website that you want to use, you can download and import them also. You'll find the *.unitypackage* file in the *Unity* folder within the compressed (.zip) file.

These files contains the models we will use to make our game.

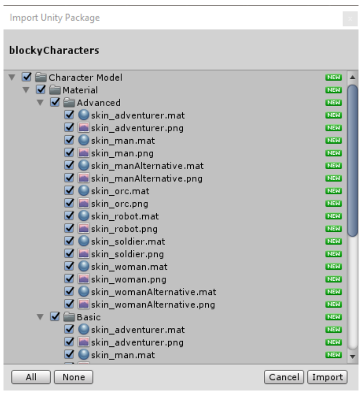
Select **Assets** -> **Import Package** -> **Custom Package**



Locate the file *blockyCharacters.unitypackage* and click **Open**.

After a few seconds you will see a pop-up dialog open asking you which assets you would like to import into your project.

Make sure all the assets are selected and press the **Import** button.


Import Unity Package

blockyCharacters

```
Character Model                          NEW
  Material                               NEW
    Advanced                             NEW
      skin_adventurer.mat                NEW
      skin_adventurer.png                NEW
      skin_man.mat                       NEW
      skin_man.png                       NEW
      skin_manAlternative.mat            NEW
      skin_manAlternative.png            NEW
      skin_orc.mat                       NEW
      skin_orc.png                       NEW
      skin_robot.mat                     NEW
      skin_robot.png                     NEW
      skin_soldier.mat                   NEW
      skin_soldier.png                   NEW
      skin_woman.mat                     NEW
      skin_woman.png                     NEW
      skin_womanAlternative.mat          NEW
      skin_womanAlternative.png          NEW
    Basic                                NEW
      skin_adventurer.mat                NEW
      skin_adventurer.png                NEW
      skin_man.mat                       NEW
```

All None Cancel Import

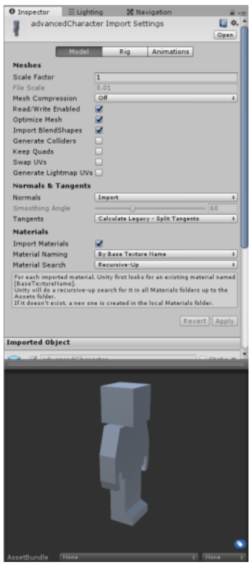
Once the import process has finished, you will see the *Character Model* folder in the *Project* window, containing the *Material* and the *Model* subfolders.

Materials are definitions of how a surface should be rendered, including references to textures used, tiling information, colour tints and more. Each file in this folder contains information about how a model will be drawn (for example, what colour to use when drawing a specific part of a model). Without the files in this folder, all the models in our game would be a dull grey colour.

The second folder is the *Model* folder. This folder contains the models we need to draw our game (like the people we will rescue, for example).

Take a moment to explore the different models in this folder. As you click on each model file you should see a preview of that asset appear in the Inspector window.

Once you get a bit more experienced using Unity you may want to find your own models and put them into the scene. (The Unity Asset Store is a good place to start looking. From the **Window** menu, select **Asset Store**.



advancedCharacter Import Settings

Model | Rig | Animations

Meshes

Scale Factor	1
File Scale	0.01
Mesh Compression	Off
Read/Write Enabled	✓
Optimize Mesh	✓
Import BlendShapes	✓
Generate Colliders	☐
Keep Quads	☐
Swap UVs	☐
Generate Lightmap UVs	☐

Normals & Tangents

Normals	Import
Smoothing Angle	60
Tangents	Calculate Legacy - Split Tangents

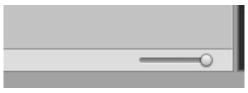
Materials

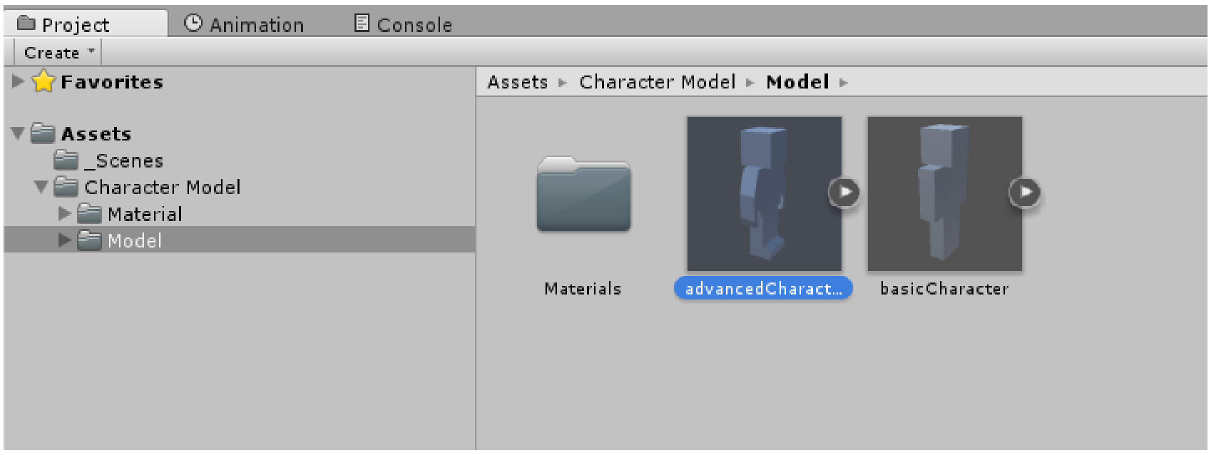
Import Materials	✓
Material Naming	By Base Texture Name
Material Search	Recursive-Up

For each imported material, Unity first looks for an existing material named [BaseTextureName].
Unity will do a recursive-up search for it in all Materials folders up to the Assets folder.
If it doesn't exist, a new one is created in the local Materials folder.

You will also see a thumbnail image of each model in the *Project* window.

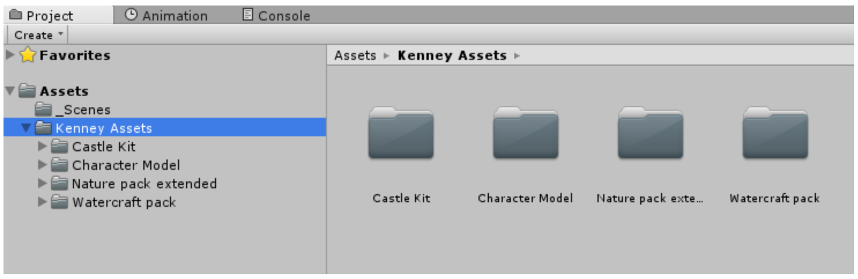
If you are only seeing small file icons in this window, then adjust the zoom. The slider in the bottom right of this window will allow you to increase the size of the icons until a thumbnail image of the model appears.





Now that you have imported the character models, import the remaining 3 *.unitypackage* files.

You may want to drag the *Character Model*, *Nature pack extended* and *Watercraft pack* folders into the *Kenny Assets* folder so that everything is nice and ordered.



Create a Basic Scene:

For this tutorial we will only need two things – a boat model and a 3D plane. Once you've completed the tutorial you can go back add more models to the scene, but for now let's just focus on getting some basic boat movement.

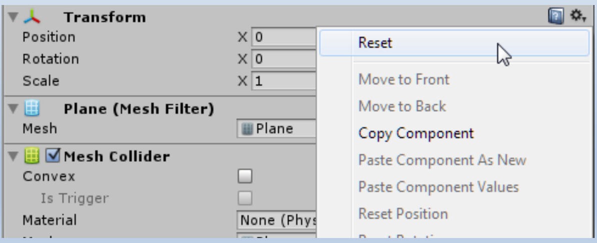
Select **Game Object** -> **3D Object** -> **Plane**. Reset the transform (set the x,y,z position to 0,0,0; do the same for the rotation, and ensure the scale is set to 1,1,1).

Set the scale of the plane to (10, 1, 10).

This plane will form the water of our level.

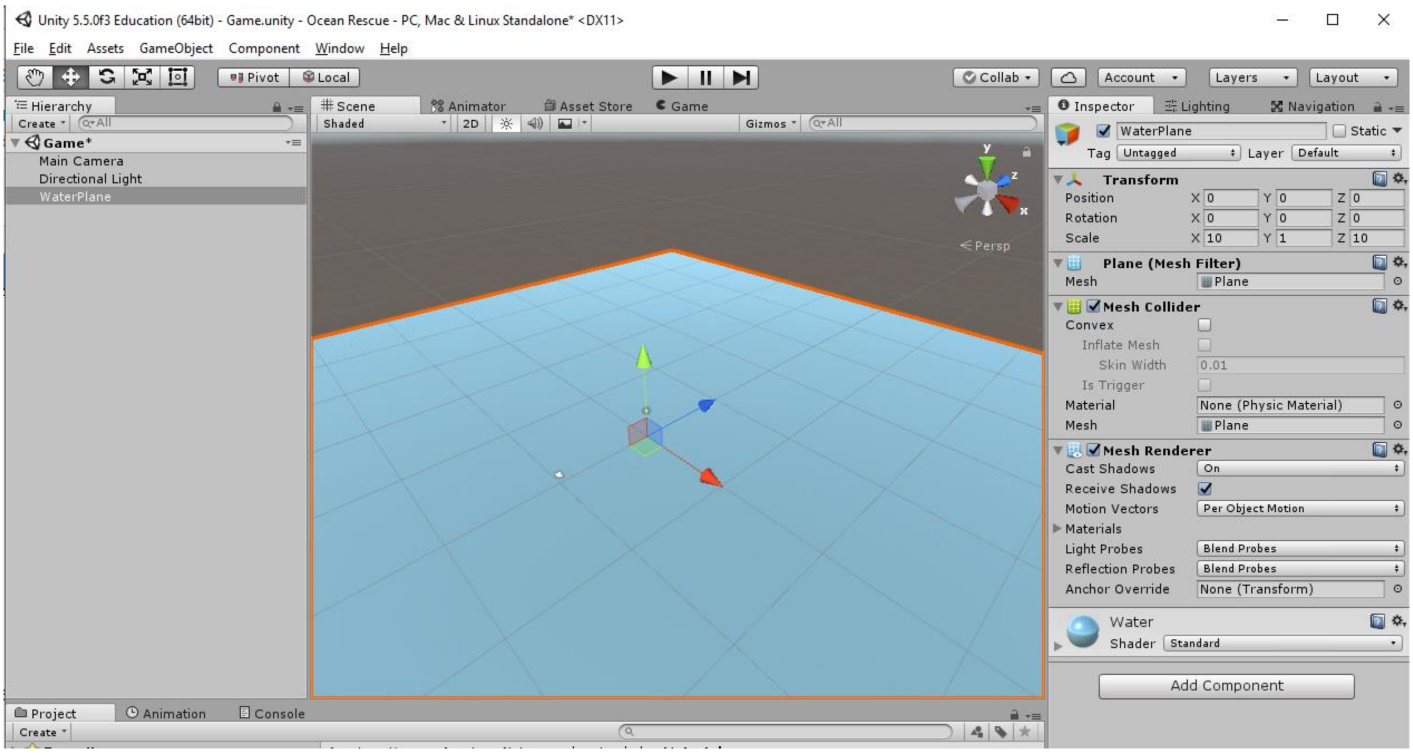
You may want to rename your plane and give it a more descriptive name. Rename it to *WaterPlane*.

A quick way to reset the transform (or any component) is to click the options icon on the upper-right of the component and select **Reset**.


Transform
Position X 0
Rotation X 0
Scale X 1
Plane (Mesh Filter)
Mesh Plane
Mesh Collider
Convex
Is Trigger
Material None (Phys

Lastly, let's make the water look like the water by giving it a colour. In the *Project* window find and open the *Nature pack extended/Materials* folder. Drag the *Water* material onto the plane. (You can either drag it onto the plane in the scene, or drag it onto the plane's properties in the *Inspector* window. Both ways will work).

Your scene should look something like this:


Unity 5.5.0f3 Education (64bit) - Game.unity - Ocean Rescue - PC, Mac & Linux Standalone* <DX11>

Hierarchy: Game*, Main Camera, Directional Light, WaterPlane

Inspector: WaterPlane — Tag Untagged, Layer Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 10 Y 1 Z 10

Plane (Mesh Filter)
Mesh Plane

Mesh Collider
Convex
Inflate Mesh
Skin Width 0.01
Is Trigger
Material None (Physic Material)
Mesh Plane

Mesh Renderer
Cast Shadows On
Receive Shadows ✓
Motion Vectors Per Object Motion
Materials
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)

Water
Shader Standard

This is all we need for now. Later, you can add more assets to make a complete level.

Add a Boat:

In the *Watercraft pack* folder, find the *watercraftPack_020* model and drag it into the scene. (You can use a different boat if you like, but we'll be adding some characters inside the boat later on, so I suggest choosing one with an open roof). Reset its transform.

Before we begin scripting the boat, we need to add a few components to the model. These components will influence how the boat behaves in the game.

With the boad selected in the *Hierarchy* press the **Add Component** button in the *Inspector* window, and select **Physics** -> **Rigidbody**. Set the *Mass*, *Drag* and *Angular Drag* to 1.

The *Mass* value here is relative to other objects in the scene, and is not very important for our purposes. Setting a higher mass value will mean that our boat reacts less when it is hit by (or hits) another object (since nothing in our game will hit our boat, we don't need to worry about mass too much).

The *Drag* and *Angular Drag* values are important. These values will make our boat gradually stop moving after we stop applying a force to it (i.e., drag will slow the boat down after we 'take our foot off the accelerator'). So if you don't want your boat sliding all over the level we'll need to set these values to 1. (A higher value will make the boat stop faster, but it also means the boat will move slower.)

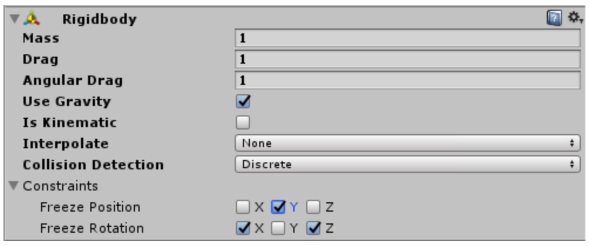
We also want to use gravity, so ensure *Use Gravity* is checked.

Finally we want to add a freeze constraint. We want to limit the boat's movement so that it never moves along the *Y* axis (i.e., the boat cannot jump or fly). Under *Constraints*, for the *Freeze Position* ensure *Y* is checked.

When we add collision objects to the level you may see why this is necessary. Basically, without this freeze constraint when the tank hits something in the level it can bounce a little. It's possible for the boat to bounce so much that it starts flying into the air. We want our boat to react a little more believably, so we make sure the boat can never fly into the air after a collision.

Also freeze rotation on the *X* and *Z* axis. This will make sure our boat doesn't flip due to collisions with random objects in our scene

The properties for the *Rigidbody* should look like this:



Rigidbody

Mass	1
Drag	1
Angular Drag	1
Use Gravity	✓
Is Kinematic	☐
Interpolate	None
Collision Detection	Discrete
Constraints	
Freeze Position	☐ X ✓ Y ☐ Z
Freeze Rotation	✓ X ☐ Y ✓ Z

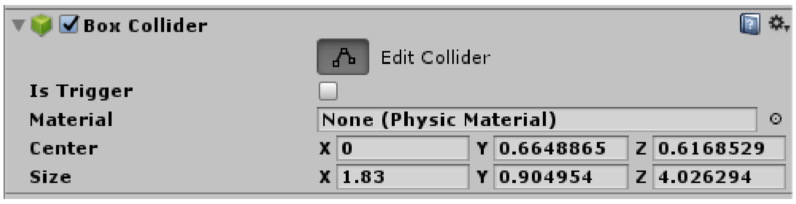
Next add a **Box Collider** component. Click **Add Component** -> **Physics** -> **Box Collider**.

This will add a collision box around our boat. When the box collider is added, it won't be big enough to cover the whole boat, so we need to enlarge it.

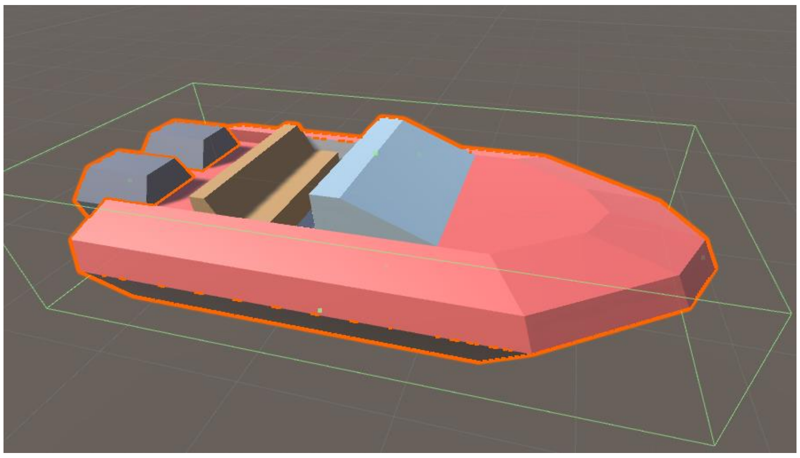
You can try to do this yourself by modifying the values in the *Box Collide* component, or you can enter the following values:

Center	x: 0	y: 0.6648	z: 0.6168
Size	x: 1.83	y: 0.9049	z: 4.0262

The component values should look like this:



Box Collider
Edit Collider
Is Trigger
Material: None (Physic Material)
Center: X 0 Y 0.6648865 Z 0.6168529
Size: X 1.83 Y 0.904954 Z 4.026294



We now have everything set up for the boat. It's time to do some scripting.

Scripting the Boat Movement:

In the *Project* window, under the *Assets* folder make a new folder called "*Scripts*". Under this folder, make a new folder called "*Boat*".

We'll put all the scripts that control the tank under this *Assets/Scripts/Boat* folder. (We will be writing a lot of scripts, so it's a good idea to keep everything ordered).

In the *Assets/Scripts/Boat* folder, make a new *C# Script* and call it "*BoatMovement*"

When you create a new script, make sure there are no spaces in the name. Use **"BoatMovement"** instead of "Boat Movement"

Double-click the *BoatMovement* script. Either *MonoDevelop* or *Visual Studio* will open for you to start editing your script. (Your teacher will discuss with you the differences between these two programs, and how to set up your preferred editor).

Enter the following code into this script (we'll discuss its meaning next):

```
using UnityEngine;
using System.Collections;

public class BoatMovement : MonoBehaviour
{

    public float m_Speed = 12f;        // How fast the boat moves forward and back
    public float m_TurnSpeed = 180f;   // How fast the boat turns in degrees/second

    private Rigidbody m_Rigidbody;
    private float m_MovementInputValue; // The current value of the movement input
    private float m_TurnInputValue;     // the current value of the turn input

    private void Awake()
    {
        m_Rigidbody = GetComponent<Rigidbody>();
    }

    private void OnEnable()
    {
        // when the boat is turned on, make sure it is not kinematic
        m_Rigidbody.isKinematic = false;

        // also reset the input values
        m_MovementInputValue = 0f;
        m_TurnInputValue = 0f;
    }

    private void OnDisable()
    {
        // when the boat is turned off, set it to kinematic so it stops moving
        m_Rigidbody.isKinematic = true;
    }

    // Update is called once per frame
    private void Update()
    {
        m_MovementInputValue = Input.GetAxis("Vertical");
        m_TurnInputValue = Input.GetAxis("Horizontal");
    }

    private void FixedUpdate()
    {
        Move();
        Turn();
    }

    private void Move()
    {
        // create a vector in the direction the boat is facing with a magnitude
        // based on the input, speed and time between frames
        Vector3 movement = transform.forward * m_MovementInputValue * m_Speed *
                           Time.deltaTime;

        // Apply this movement to the rigidbody's position
        m_Rigidbody.MovePosition(m_Rigidbody.position + movement);
    }
```

(continued on next page)

```
    private void Turn()
    {
        // determine the number of degrees to be turned based on the input,
        // speed and time between frames
        float turn = m_TurnInputValue * m_TurnSpeed * Time.deltaTime;

        // make this into a rotation in the y axis
        Quaternion turnRotation = Quaternion.Euler(0f, turn, 0f);

        // apply this rotation to the rigidbody's rotation
        m_Rigidbody.MoveRotation(m_Rigidbody.rotation * turnRotation);
    }
}
```

Let’s discuss the code you just wrote.

The variables *m_Speed* and *m_TurnSpeed* are public variables, meaning you will be able to modify their initial value in the Editor. These variables control how fast your boat will move and turn. Think of these values as ’12 units’ forward per second, and 180 degrees of rotation per second.

The private variable *m_Rigidbody* will allow us to modify the movement of the boat via the rigid body component that we placed on the boat earlier. (The rigid body component will control anything to do with physics – so that means all the movement of the boat and any collisions are handled by this component).

The next private variables *m_MovementInputValue* and *m_TurnInputValue* allow us to store the movement and turn information we get from keyboard presses and use it later in our script to move the tank.

Awake:
The awake function runs when the object this script is attached to is added to the scene. Here we can initialize any variables in our script. Here we get the rigid body component attached to our boat and set the m_Rigidbody variable to point to this component.

OnEnable:
This function is executed when we ‘turn on’ the boat. Later on, when we add the ‘Game Over’ part of the game, we’ll turn the boat off (so that it will freeze in its current positions).

When we start a new game we’ll need to turn the boat on again. This function handles what happens when we do that.

Here, we make sure that the ‘isKinematic’ option is turned off. (Kinematic objects can’t move using physics – so by turning this off we allow our boat to move again). Finally we reset the input values.

OnDisable:
This is the opposite of the *OnEnable* function described above. When the game ends we ‘freeze’ the boat by making it Kinematic. (This effectively turns its physics processing off, turning the moving boat into a static object).

Update:
This function runs once every frame. It is the function responsible for updating the movement of our boat in the game.

Here we get the Axis input values. Imagine you are playing with a gamepad and one of the joysticks is controlling the movement of the boat. When you move the joystick on the horizontal axis the boat turns, and when you move it on the vertical axis the boat moves forward.

We won't be using a gamepad to control our boat. Instead Unity maps keyboard keys to these axes. The up and down arrow keys (or 'W' and 'S') are mapped to the vertical axis, and the left and right keys (or 'A' and 'D') to the horizontal.

This function checks the value of this input every frame and stores the value in the member variables for us to use later.

FixedUpdate:
The *FixedUpdate* function is similar to the *Update* function, but with one important difference. The *Update* function runs once every frame (how often the *Update* function runs per second can vary depending on the speed of your computer, CPU load, etc), but the *FixedUpdate* function runs at a fixed interval (a set number of times per second).

The *FixedUpdate* function is used specifically to update any physics that happens in the game. Processing physics actually takes a bit of time, so it is done less often than anything in the *Update* function.

In this function we take the input values we read in the *Update* function and modify the movement of our boat via the *Rigidbody* component.

You should never do any physics processing in the *Update* function, or else you can get unpredictable results. All physics processing needs to happen in the *FixedUpdate* function, but you should *only* do physics processing in this function – which is why we've put the keyboard event handling in the *Update* function and the boat movement updating in the *FixedUpdate* function.

To find out more about Unity's physics handling and the *FixedUpdate* function, you can refer to the Unity documentation: https://unity3d.com/learn/tutorials/modules/beginner/scripting/update-and-fixedupdate

Move:
Called by the *FixedUpdate* function to move the tank according to the keyboard presses.

Turn:
Called by the *FixedUpdate* function to turn the tank according to the keyboard presses.

Testing the Boat Movement:

Return to the Unity editor. Select the Boat in the *Hierarchy* and in the *Inspector* window press **Add Component** -> **Scripts** -> **Boat Movement**.

The script is now attached to the boat object, meaning that the script will execute while the boat is in the level.

Press the *Play* button now to test the movement of your boat.