Projects

Non-player characters

Create 3D non-player characters to interact with, avoid, and collect power-ups from

Step 1 Introduction

Create 3D non-player characters to interact with, avoid, and collect from.

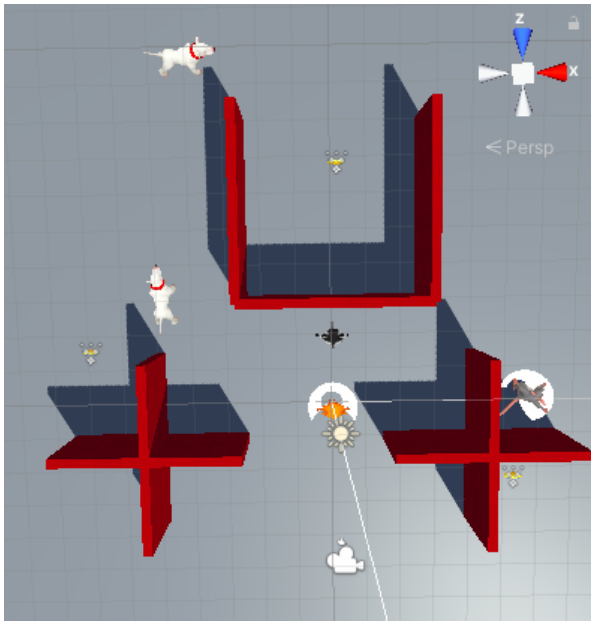
This project is brought to you with generous support from **Unity Technologies** (https://unity.com/). These **projects** (https://projects.raspberrypi.org/en/pathways/unity-intro) offer young people the opportunity to take their first steps in creating virtual worlds using Real-Time 3D.

This project follows on from **Star collector** (https://projects.raspberrypi.org/en/projects/star-collector). You can use the Unity scene that you created in that project as the basis for this project. We've also provided a starter project that you can use.

> A **non-player character (NPC)** is a character that is not controlled by the person playing the game. NPCs can be set up for the player to interact with, avoid, compete with, or are simply used to populate the game world.

You will:

- Add non-player characters (NPCs) with animation and movement
- Interact with NPCs using simple dialog and buttons
- Use child GameObjects to allow players to hold other GameObjects


Z
X
Persp

Step 2 Gamemaster NPC

The player will talk to a Gamemaster NPC to set the scene and click a button when they are ready to start.

One role an NPC can be programmed to carry out is that of gamemaster. Gamemasters are storytelling NPCs that provide instructions and direct the game. Your gamemaster NPC will give details to introduce the minigame and start the game once the player presses the ‘Ready’ button.

Launch the Unity Hub and open the project you created for **Star collector** (https://projects.raspberrypi.org/en/projects/star-collector/0).

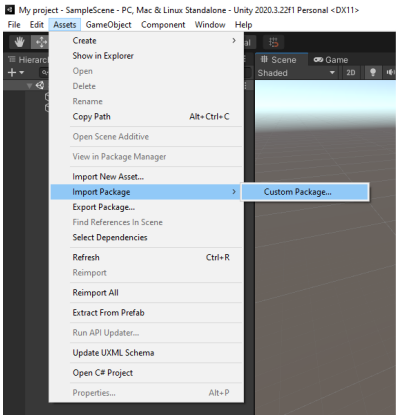
I haven't got my Star collector project

If you are not able to open your project, you can download, unzip, and import this non-player character assets pack.

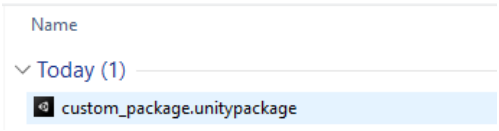
rpf.io/p/en/non-player-characters-go (https://rpf.io/p/en/non-player-characters-go)

Import a package

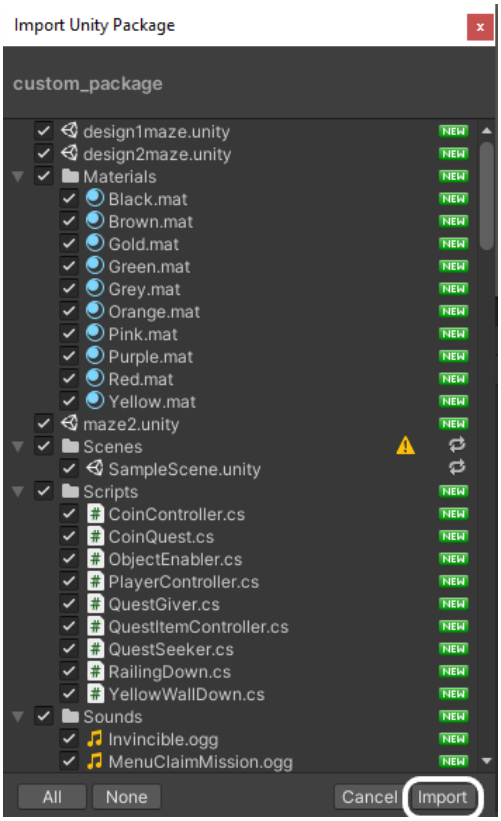
Import your downloaded package.



Select the location of your downloaded package.

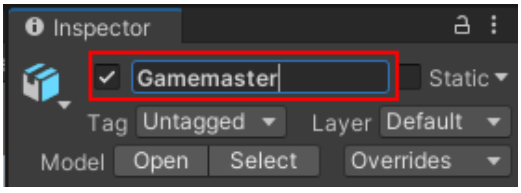


Click on the **Import** button to import all of the package.



In the Project window, go to the **Models** folder and drag a **Cat** or **Raccoon** character into the Scene view.

With your new character GameObject selected, go to the Inspector window and rename it `Gamemaster`:

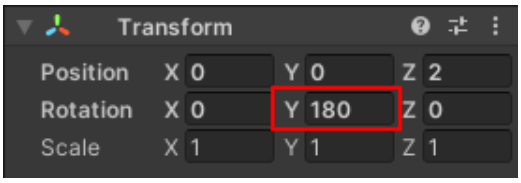


Position your gamemaster NPC using either:

- The arrows from the Transform and Rotate tools and the Scene view
- The coordinates from the Transform component in the Inspector window

Your gamemaster NPC character should be close to the Player's starting point and visible at the start of the game.

To make your Gamemaster face toward the Player, change the y rotation to `180`:



Moving in the Scene view

To use flythrough mode, hold down the **right** mouse button **and use**:

- WASD to move around
- QE for up and down
- Shift to go faster
- The mouse to rotate the camera

Use the scroll wheel on your mouse to zoom in and out. Your trackpad may also have a scroll motion.

To move the scene around, hold the `ALT` key and the middle mouse button and then drag to move. You can also use the arrow keys to move around.

To focus on an object, click on the GameObject in the Scene view and then click `F`.

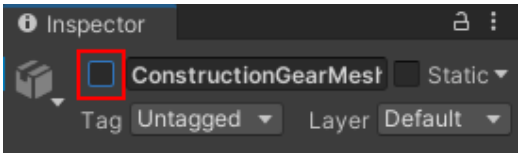
You can also click on an object in the Hierarchy window and then click `Shift`+`F` to focus on that GameObject in the Scene view.

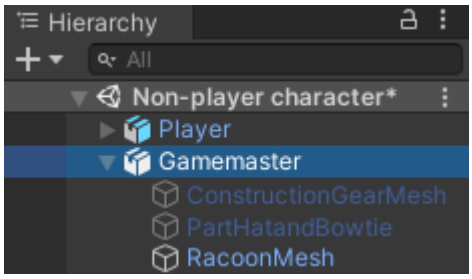
Tip: If you get lost, click on your Player in the Hierarchy window and then `Shift`+`F` to focus on the Player. Then you can use the scroll wheel to zoom out.

In Unity, a **parent GameObject** can have **child GameObjects** that move, rotate, and scale with it. This is really useful for positioning the child objects in relation to their parent. A parent can have many child GameObjects, but a child can have only one parent.

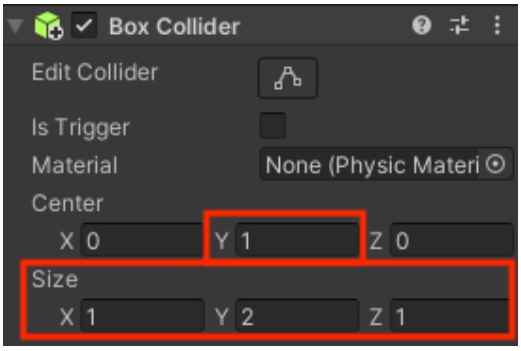
The Gamemaster GameObject has several child GameObjects enabled that represent costumes for the character.

Choose which costumes to keep enabled and which to disable by unchecking the box in the Inspector window for any you want to remove:





Select the **Gamemaster GameObject** and click on **Add Component**. Add a **Box Collider** so that the Player cannot walk through, or climb on top of, the Gamemaster. Change the y Center and Size:



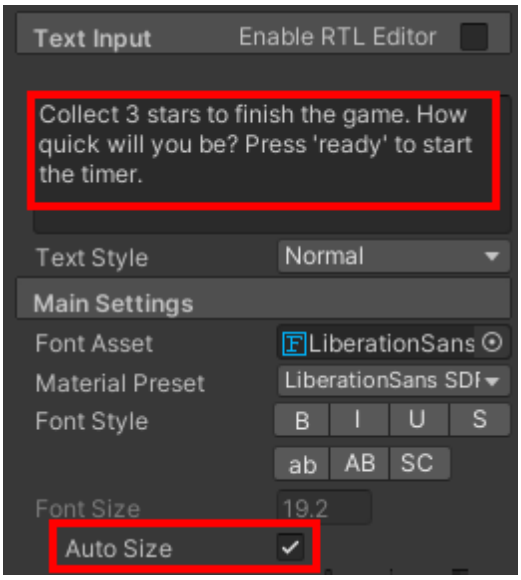
The Gamemaster will use UI child GameObjects to display the game instructions and a button to press to start the timer. These child GameObjects will only be displayed when the Player is close enough to talk to the Gamemaster and the game is not already in progress.

Right-click on the **Gamemaster** in the Hierarchy window and from UI select **Text - TextMeshPro** to create text that is a child GameObject of the Gamemaster. This will also automatically create a canvas for the text to sit on:

Tip: If you accidentally create the object at the top level, or as a child of the wrong GameObject, you can drag it to the Gamemaster GameObject in the Hierarchy window.

From the Hierarchy window, select the **Text (TMP) GameObject** and rename it to `Message`. In the Text Input component, add a message to explain your minigame. Include the message `Press 'Ready' to start the timer`.

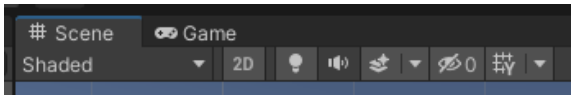
Put a checkmark in the Auto Size property so that the text resizes to fit the message to the screen of the player:



Use the Rect Transform component in the Inspector window to anchor the text to the bottom left, then change the Pos x and Pos y coordinates, and the width and height:

Positioning text with your mouse

You can position and size your text box using your mouse. Select your **Text (TMP)** object in the Hierarchy window. Then, while in Scene view, click on **2D** in the toolbar.



Press `Shift+F` to focus on the text, and then zoom out a little to see the position it will sit on the screen.

You can now use the mouse to position and resize the text.

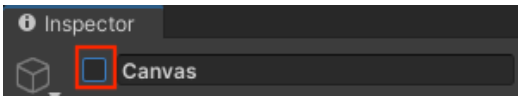
From the Hierarchy window, right-click on the Gamemaster's **Canvas** child GameObject and from **UI** select **Button - TextMeshPro**. This creates a second UI GameObject for the Gamemaster.

Click on the drop-down arrow next to the Button GameObject and select the **Text (TMP)** GameObject. This controls the text message shown on the button. Go to the Inspector window and change the **Text Input** property to `Ready`:

Test: Experiment with the Transform properties of your message and button until you are happy with how they look in the Game view:

Exit Play mode.

Select the Gamemaster's **Canvas** and disable it by unchecking the box in the Inspector.



This means that you will still be able to focus on the Gamemaster in the Scene view using `F` (or `Shift+F` from the Hierarchy window). Your code will enable the canvas when it is needed.

Save your project

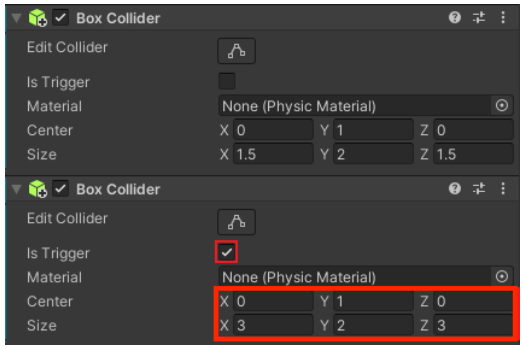
Step 3 Controlling the game

It's not fair if the time starts before the player is ready! The `Ready` button will allow the player to start the time AND activate the stars.

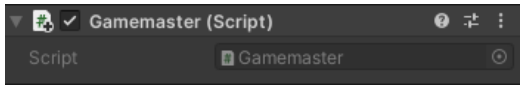
At the moment, the canvas is always visible. It should only be enabled when the Player is interacting with the Gamemaster.

Select your **Gamemaster GameObject** and click on **Add Component** in the Inspector window then add a second **Box Collider**.

This Box Collider will trigger the canvas with the message and the button to be shown, so it needs to be bigger than the Box Collider that stops the Player walking into the Gamemaster:



With the Gamemaster GameObject selected, add a new Script component and name it `GamemasterController`.



Double-click on the **GamemasterController** script to open it in your script editor. Add code to use TMPro:

GamemasterController.cs

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

Create a public canvas variable called `canvas` and add code to make sure the canvas is disabled at the start:

GamemasterController.cs

```
6  public class GamemasterController : MonoBehaviour
7  {
8      public GameObject canvas;
9      // Start is called before the first frame update
10     void Start()
11     {
12         canvas.SetActive(false);
13     }
```

Add two new methods. The first to enable the canvas when the Player is in the collider. The second to disable the canvas when the Player has moved away:

GamemasterController.cs

```
16 void Update()
17 {
18
19 }
20
21 void OnTriggerEnter(Collider other)
22 {
23     if (other.CompareTag("Player"))
24     {
25         canvas.SetActive(true);
26     }
27 }
28
29 void OnTriggerExit(Collider other)
30 {
31     if (other.CompareTag("Player"))
32     {
33         canvas.SetActive(false);
34     }
35 }
```

Save your script and return to the Unity Editor.

Find the **Gamemaster Canvas child GameObject** in the Hierarchy window. Drag the Canvas GameObject to the Canvas variable field in the GamemasterController script component in the Inspector.

Test: Play your minigame, walk up to the Gamemaster and move away again. The canvas appears when the Player triggers the Gamemaster collider and disappears when the Player moves away.

Exit Play mode.

The button looks great, but needs to trigger an event when it is pressed.

Open the **GamemasterController** script and create two new public variables called `gameStarted` and `startTime`:

GamesmasterController.cs

```
6  public class GamemasterController : MonoBehaviour
7  {
8      public GameObject canvas;
9      public bool gameStarted = false;
10     public float startTime = 0.0f;
```

Create a public method called `PlayerReady` to set the game conditions when the Player has clicked the 'Ready' button.

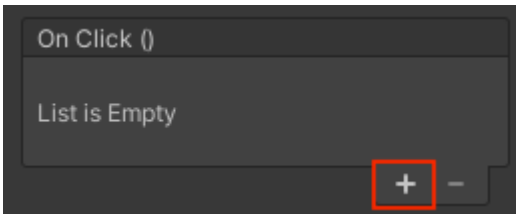
The time at which the button was pressed needs to be stored so you can work out how long the game has been in play:

GamemasterController - PlayerReady()

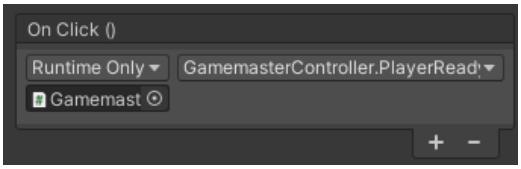
```
8  public GameObject canvas;
9  public bool gameStarted = false;
10 public float startTime = 0.0f;
11
12 public void PlayerReady()
13 {
14     gameStarted = true;
15     startTime = Time.time; // Time when the button is pressed
16     canvas.SetActive(false);
17 }
```

Save your script and return to the Unity Editor.

From the Hierarchy window, select the **Button GameObject**, then go to the Inspector window **On Click ()** property and click on the **+**.



Drag the **Gamemaster GameObject** from the Hierarchy window to the field underneath 'Runtime Only'. In the Function drop-down menu select **GamemasterController.PlayerReady** to join your new method to the button's click event:



Test: Play your minigame. The button disables the canvas, but the time still counts up from the second the game begins.

Fix any errors that appear.

Exit Play mode.

Open your **StarPlayer** script to see the code that controls the time displayed.

Create a new public variable for your Gamemaster script:

StarPlayer.cs

```
6  public class StarPlayer : MonoBehaviour
7  {
8      public int stars = 0; // An integer whole number
9      public TMP_Text starText;
10     public TMP_Text timeText;
11     public GamemasterController gamemaster;
```

Change the code in your `Update` method to only update the time if the button has been pressed and stars are less than three.

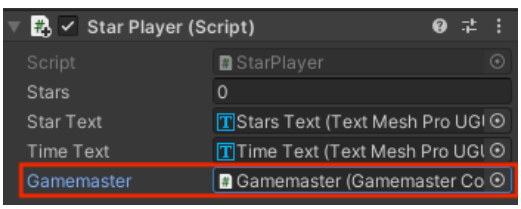
`Time.time` starts when the game begins. Minus the `startTime` from `Time.time` to display the elapsed time since the button was pressed:

StarPlayer.cs - Update()

```
20 void Update()
21 {
22     starText.SetText("Stars: " + stars);
23     if (stars < 3 && gamemaster.gameStarted == true)
24     {
25         timeText.SetText("Time: " + Mathf.Round(Time.time - gamemaster.startTime));
26     }
27 }
```

Save your script and return to the Unity Editor.

Select the **Player** and go to the **Star Player (script)** component. Click on the circle next to Gamemaster and choose the **Gamemaster GameObject**:



Test: Play your minigame. Check that the time doesn't start until the button has been pressed. What happens if you go back to the Gamemaster a second time?

Exit Play mode.

Open your **GamemasterController** script and amend the condition in **OnTriggerEnter** to only run if the Player collides and the button hasn't been pressed:

GamemasterController.cs - OnTriggerEnter(Collider other)

```
31 void OnTriggerEnter(Collider other)
32 {
33     if (other.CompareTag("Player") && gameStarted == false)
34     {
35         canvas.SetActive(true);
36     }
37 }
```

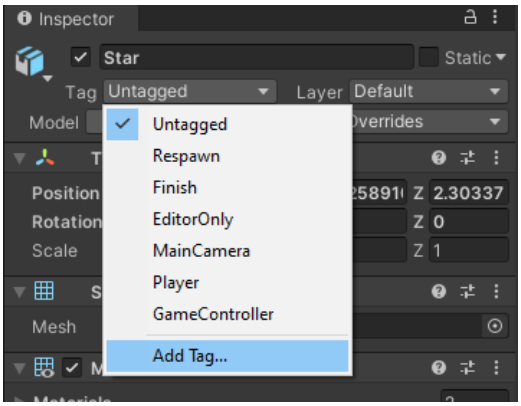
Test: Play your minigame again. Are there any other ways a player could cheat?

At the moment the stars are active when the game begins, so the player could collect the stars before going to the Gamemaster — this would mean a very quick time taken to complete the game!

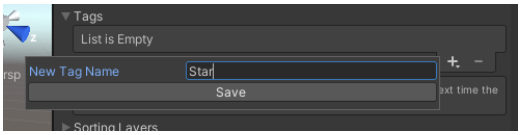
Exit Play mode.

You can use Tags to identify objects that you want to treat in the same way.

Select one of your **Star GameObjects** and click **Add Tag** in the Inspector.



Create a new tag called `Star` by clicking on the **+** icon.



Save your tag and then select all of the **Star GameObjects** in the Hierarchy window by holding down `Ctrl` (or `Cmd`) and then clicking on each of them.

Set the tag to 'Star' in the Inspector; this sets the tag for all of the Stars.

In C#, you can store multiple objects of the same type in an **Array** variable. An array variable has left and right square brackets `[]` after the type, so `GameObject[] stars;` stores multiple Star GameObjects.

Open your **GamemasterController** script and add a new variable to store your Star GameObjects:

GamemasterController.cs

```
6  public class GamemasterController : MonoBehaviour
7  {
8      public GameObject canvas;
9      public bool gameStarted = false;
10     public float startTime = 0.0f;
11     GameObject[] stars;
```

You can use a `for` loop to perform the same action on each item in an array.

Find the Star GameObjects and set them to inactive when the game starts:

GamemasterController.cs - Start()

```
21  void Start()
22  {
23      canvas.SetActive(false);
24      stars = GameObject.FindGameObjectsWithTag("Star");
25      foreach (var star in stars)
26      {
27          star.SetActive(false);
28      }
29  }
```

Set the stars to active once the player has clicked the Ready button:

GamemasterController.cs - PlayerReady()

```
13  public void PlayerReady()
14  {
15      gameStarted = true;
16      startTime = Time.time; // Time when the button is pressed
17      canvas.SetActive(false);
18      foreach (var star in stars)
19      {
20          star.SetActive(true);
21      }
22  }
```

Test: Play your minigame again. Notice that the stars do not appear until the player has clicked on the Ready button.

Debug: Make sure every star has the ‘Star’ tag.

Exit Play mode.

Save your project

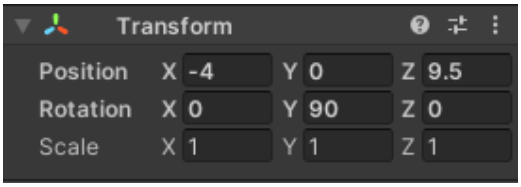
Step 4 NPC patrol

Patrolling NPCs can be used to slow players down.
Changing their path, size, position, and speed can alter the game difficulty.

Open the **Models** folder in the Project window and add a **Dog** to your scene.

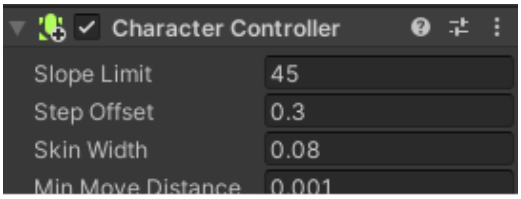
Use the Transform and Rotation tools or the Transform component to position the dog in a good position for patrolling — and to obstruct the Player reaching a star!

Tip: To see your map in a top-down view, right-click where it says **Persp** in the top right of the Scene view and choose **Top**. To return to the normal view, right-click on **Top** and choose **Free**.



With the Dog selected, go to the Inspector window and **Add Component**. Choose the **Character Controller**. Position and size the controller.

Tip: Select the Dog GameObject in the Hierarchy window and press `Shift+F` to focus on the Dog in the Scene view.



Click on **Add Component** and add a **Box Collider** to the **Dog** so that the Player cannot walk through, or climb on top of, the Dog. Change the y Center and Size:

As both the Dog and the Player will be moving, you will need to add a Box Collider to the **Player** so that the Dog cannot climb on top of the Player.

Select the **Player GameObject** from the Hierarchy window, then click **Add Component** and add a **Box Collider**. Change the y Center and Size:

With the Dog GameObject selected, add a new Script component and name it `PatrolController`.

Open the **PatrolController** script and create a `patrolSpeed` variable. Create another two public variables for the `minPosition` and `maxPosition` of the patrol space.

DogController.cs

```
5  public class DogController : MonoBehaviour
6  {
7      public float patrolSpeed = 3.0f;
8      public float minPosition = -4.0f;
9      public float maxPosition = 4.0f;
```

Add code to the `Update` method so the Dog moves foward until the `maxPosition` is reached, then turns `180` degrees and moves forward again until the minimum positon is reached.

DogController.cs - Update()

```
17 void Update()
18     {
19         CharacterController controller = GetComponent<CharacterController>();
20         Vector3 forward = transform.TransformDirection(Vector3.forward);
21         controller.SimpleMove(forward * patrolSpeed);
22 
23         if (transform.position.x > maxPosition)
24         {
25             transform.Rotate(0, 180, 0);
26             transform.position = new Vector3(maxPosition, transform.position.y, transform.position.z);
27         }
28         else if (transform.position.x < minPosition)
29         {
30             transform.Rotate(0, 180, 0);
31             transform.position = new Vector3(minPosition, transform.position.y, transform.position.z);
32         }
33     }
```

Setting the `transform.position` makes sure the Dog isn't past the limit when they turn around. If you don't do this, you might find that the Dog 'glitches' back and forward.

Save your script and return to the Unity Editor.

Test: Play your game and check that the Dog makes it harder to reach a star quickly.

Track the movement of the Dog. If the patrol length is not right for your scene, you can adjust the Min Position and Max Position in the Inspector whilst the game is playing.

Tip: Remember that variables edited in Play mode are not saved after exiting Play mode so make a note of the positions you like best then exit Play mode and go back to your script to update the values in your `minPosition` and `maxPosition` variables. Save your script then return to the Unity Editor.

Now that the position and path of the patrolling dog is decided, it's time to make things more realistic with animation.

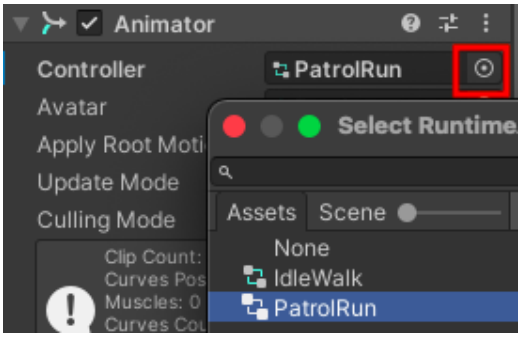
In the Project window, navigate to the **Animation** folder. Right-click and go to **Create** then select **Animation Controller** and name your new animation controller `PatrolRun`.

Double-click on the **PatrolRun** animation controller to open it in the Animator window.

The patrol dog will have just one animation that will run repeatedly. From the Animation folder in the Project window, drag the **Dog_Run** animation up to the Animator window.

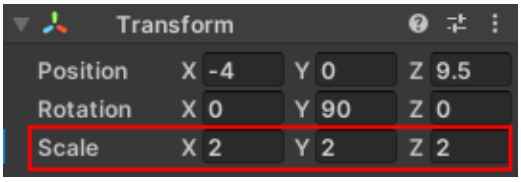
Tip: If you can't see all of the boxes in the Animator window, you can click on the black grid then press the **a** key to refocus the window. Then pan left and right using `Alt`+left mouse button or zoom in and out using `Alt`+right mouse button.

From the Hierarchy window, select the **Dog GameObject**, then go to the Inspector window **Animator** component. Click on the circle next to Controller and select **PatrolRun** to link your animation controller.



Test: Play your game to see the patrol dog run across the patrol path.

Test: Tweak your patrol dog until you are happy with the path and animation. To change the difficulty level, you can alter the Scale to make a bigger or smaller dog.



Debug: If your animation isn't working, in the Inspector, check that **Apply Root Motion** is not selected for your non-player character.

Exit Play mode.

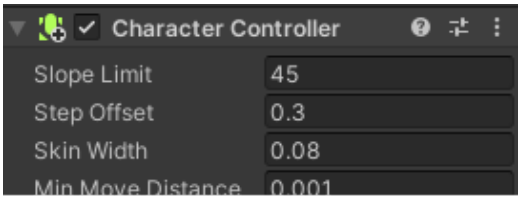
Save your project

Step 5 Follower NPC

An NPC that follows the Player can be an obstacle — and very annoying!

Drag another Dog into the Scene view and into a position that would be hard to navigate around.

With the Dog selected, go to the Inspector window and **Add Component**. Choose the **Character Controller**. Position and size the controller so it covers the whole of your Dog.



Click on **Add Component** and add a **Box Collider** to the Dog so that the Player cannot walk through, or climb on top of, the Dog. Change the y Center and Size:

Go to the **Add Component** button again and add a second **Box Collide** to the Dog.

This Box Collider will use `IsTrigger` to make the Dog follow the Player if the Player gets close enough to draw the Dog's attention. This Box Collider needs to be big enough that the Player can't easily sneak past:

With the new Dog GameObject selected, add a new Script component and name it `FollowController`.

Double-click on the **FollowController** script and create a public GameObject variable. Add code so that the script can access the Player attributes:

FollowController.cs

```
5  public class FollowController : MonoBehaviour
6  {
7      public GameObject Player;
```

Add a line in the Update method so that the dog will always look at the Player:

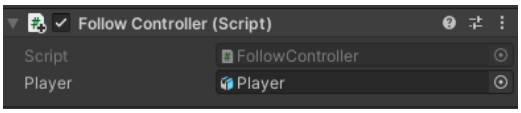
FollowController.cs - Update()

```
16 void Update()
17 {
18     transform.LookAt(Player.transform);
19 }
```

Save your script and return to the Unity Editor.

Click on your second dog in the Hierarchy and scroll down in the Inspector to see the **FollowController** script in the window.

Click on the circle next to Player and select the **Player GameObject** from the menu:



Test: Play your minigame. Make sure you can't walk through the Dog. Check that the Dog continuously rotates to face the Player.

Exit Play mode.

Open the **FollowController** script and create an `IsFollowing` variable set to `false`.

FollowController.cs

```
5  public class FollowController : MonoBehaviour
6  {
7      public GameObject Player;
8      public bool isFollowing = false;
```

Add a method that triggers when the Player collides with the Dog. This method will set `IsFollowing` to `true`:

FollowController.cs - OnTriggerEnter(Collider other)

```
5  public class FollowController : MonoBehaviour
6  {
7      public GameObject Player;
8      public bool isFollowing = false;
9
10     void OnTriggerEnter(Collider other)
11     {
12         if (other.CompareTag("Player"))
13         {
14             isFollowing = true;
15         }
16     }
```

Create three new variables to set the mechanics of the follow action:

FollowController.cs

```
5  public class FollowController : MonoBehaviour
6  {
7      public GameObject Player;
8      public bool isFollowing = false;
9      public float followSpeed = 3f;
10     public float followDistance = 2f;
11     Vector3 moveDirection = Vector3.zero; // No movement
```

Add code to the `Update` method to move the Dog towards the Player using `SimpleMove`.

Subtracting the Follower's position vector from the Player's position vector with `Player.transform.position - transform.position` gives the direction and distance between them. The `Vector3.Normalize` method turns this into a single unit vector, which can be used with `SimpleMove`.

The Dog should only move at a distance from the Player so that the Dog doesn't try to move into the same space as the Player.

FollowController.cs - Update()

```
void Update()
{
    transform.LookAt(Player.transform);
    if (isFollowing == true)
    {
        if (Vector3.Distance(Player.transform.position, transform.position) > followDistance)
        {
            CharacterController controller = GetComponent<CharacterController>();
            var moveDirection = Vector3.Normalize(Player.transform.position - transform.position);
            controller.SimpleMove(moveDirection * followSpeed);
        }
    }
}
```

Save your script and return to the Unity Editor.

Test: Play your Scene and walk up to the Dog so that you are close enough to trigger the event, then walk away. Check that the Dog follows you.

Exit Play mode.

Animation Controllers can have more than one animation. The Follower Dog will need animations for when idle and when moving.

In the Project window, select the **Animation** folder and right-click then create a new **Animation Controller** called `FollowerMove`.

Click on the **Dog** and go to the Inspector window. Drag the **FollowerMove** controller to the **Controller** property in the Animator component:

Double-click on the **FollowerMove** controller to open it in the Animation window. Drag the **Dog_Idle** animation into the grid and place it near the green box marked 'Entry':

Test: Play your minigame and check that the Dog animates when idle.

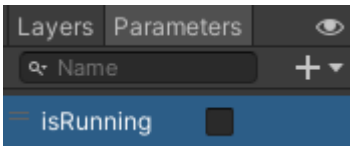
Exit Play mode.

The Dog needs a different animation for when it is moving.

Drag the **Dog_Run** animation into the Animator window for the **FollowerMove** controller.

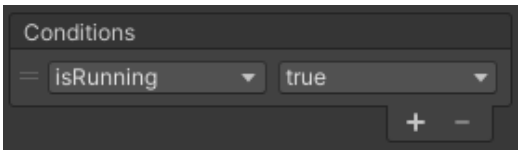
Right-click on **Dog_Idle** and select **Make Transition** and connect the transition to **Dog_Run**. Right-click on **Dog_Run** and select **Make Transition** and connect the transition to **Dog_Idle** so you have transitions in both directions.

Go to the **Parameters** tab and click on the drop-down arrow next to the '+'. Choose **bool** and name your new variable `isRunning`.



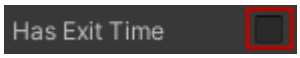
Go to the Animator window and click on the transition arrow from Dog_Idle to Dog_Run:

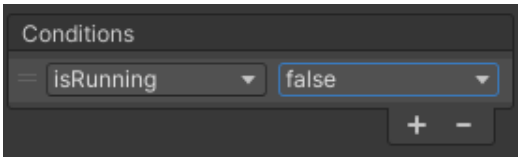
In the Inspector window for that transition, go to the Conditions component and click on the **+**. The condition should read `isRunning` is `true`:



Uncheck the 'Has Exit Time' box so that the animation transitions straight away:

Select the transition arrow from Dog_Run to Dog_Idle and follow the same steps. Uncheck the 'Has Exit Time' box, but this time add the condition `isRunning` is `false`:





Open the **FollowController** script and create an **Animator variable**. Add code to the `Start` method to set `isRunning` to `false`:

FollowerController.cs - Start()

```
Animator anim;
// Start is called before the first frame update
void Start()
{
    anim = gameObject.GetComponent<Animator>();
    anim.SetBool("isRunning", false);
}
```

Update the `if (isFollowing)` code to control the animation:

FollowerController.cs - Update()

```
void Update()
{
    transform.LookAt(Player.transform);
    if (isFollowing == true)
    {
        if (Vector3.Distance(Player.transform.position, transform.position) > followDistance)
        {
            anim.SetBool("isRunning", true);
            CharacterController controller = GetComponent<CharacterController>();
            var moveDirection = Vector3.Normalize(Player.transform.position - transform.position);
            controller.SimpleMove(moveDirection * followSpeed);
        }
        else
        {
            anim.SetBool("isRunning", false);
        }
    }
}
```

Save your script and return to the Unity Editor.

Test: Play your minigame and watch what happens in the animator as you collide with and run from the Dog.

Tip: To see the animation effect better whilst testing in Play mode:

- Click on the Dog in the Hierarchy window and then go to the Follow Controller script in the Inspector window. Slow the Dog's Follow Speed to `0.1`.
- Click on the Player in the Hierarchy window and then go to Main Camera child GameObject. In the Inspector window change the z position of the camera to `-10`.

Tip: Make sure there are no points in your game where the follower dog can completely trap the player.

Exit Play mode.

Save your project

Step 6 NPC allies

Allies are characters that help the Player by giving them clues or items; or by giving them abilities such as turbo speed.

So far the minigame has several enemies but no allies. It would be great to have an ally that gives the Player a turbo charge to make the player move and turn faster to complete the game quicker.

Drag a Rat into the Scene view and into a position that can't be seen by the Player when the game starts:

With the Rat selected, go to the Inspector window and **Add Component**. Choose the **Character Controller**. Position and size the controller so it covers the centre of the Rat:

Click on **Add Component** and add a **Box Collider** to the Rat so that the Player cannot walk through, or climb on top of, the Rat. Change the y Center and Size:

Using animation makes an NPC come to life.

In the Project window, navigate to the **Animation** folder. Right-click and go to **Create** then select **Animation Controller** and name your new animation controller `AllyIdle`.

Double-click on the **AllyIdle** animation controller to open it in the Animator window.

From the Animation folder in the Project window, drag the **Cat_IdleHappy** animation up to the Animator window:

Tip: You can use the Cat animations on the Rat and Raccoon because they are designed as humanoids.

From the Hierarchy window, select the **Rat** then go to the Inspector window **Animator** component. Click on the circle next to Controller and select **AllyIdle** to link your Animation Controller:

Tip: You can also drag the Animation Controller from the Projects window to the Controller property of the Animator in the Inspector.

Test: Play your game to see the Rat animate:

Exit Play mode.

A character with the Shield model as a child GameObject would look like they have a special effect or power. In your minigame the shield will represent a turbo speed powerup.

When the Player has the shield, they will move and turn twice as fast — but with the Ally hidden will they manage to find the shield early enough to make a difference?!

In the Project window, go to the **Models** folder and find the **Shield**. Drag the shield up to the Hierarchy window and position it as a child GameObject of the Player:

This will automaticaly add the Shield in the same position as the Player:

You will use code to hide the shield until the player picks up the turbo power boost from the Ally NPC.

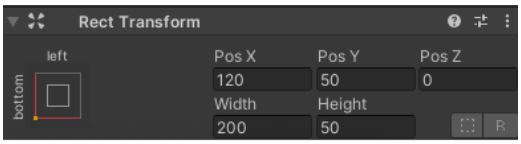
Also add a Shield as a child GameObject of the Rat:

This will automaticaly add the Shield in the same position as the Rat:

Right-click on the **Rat** in the Hierarchy window and from UI select **Text - TextMeshPro**:

In the Inspector window for the new Text (TMP) GameObject, add **Text Input** and tick the **Auto Size** box:

Use the Rect Transform component in the Inspector window to anchor the text to the bottom left then change the Pos x and Pos y coordinates:

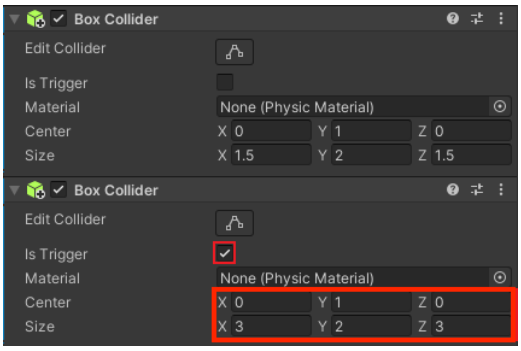


Tip: Click on the **Game** tab to see what the text looks like in Game view.

The Rat will have the shield visible until the Player collides with it. The shield will then transfer to the Player and the Rat will disappear.

Go to the **Add Component** button again and add a second **Box Collider** to the Rat.

Check `IsTrigger` and change the size so that it is bigger than the first Box Collider:



With the ally Rat GameObject selected, add a new Script component and name it `AllyController`.

Double-click on the **AllyController** script to open it in your script editor. Add code to use the TMPro namespace:

AllyController.cs

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

Create public GameObject and Canvas variables and add code to activate the turbo speed boost on the Ally and not the Player, and disable the canvas at the start:

AllyController.cs

```
public class AllyController : MonoBehaviour
{
    public GameObject turbo; // Turbo shield on NPC
    public GameObject playerTurbo; // Turbo shield on Player
    public PlayerController player;
    public GameObject canvas;

    // Start is called before the first frame update
    void Start()
    {
        turbo.SetActive(true);
        playerTurbo.SetActive(false);
        canvas.SetActive(false);
    }
```

Add code to enable the canvas and switch the turbo from the Ally to the Player and give the Player the turbo speed-up:

AllyController.cs - OnTriggerEnter(Collider other)

```
public class AllyController : MonoBehaviour
{
    public GameObject turbo; // Turbo shield on NPC
    public GameObject playerTurbo; // Turbo shield on Player
    public PlayerController player;
    public GameObject canvas;

    void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            turbo.SetActive(false);
            playerTurbo.SetActive(true);
            player.moveSpeed *= 2;
            player.rotateSpeed *= 2;
            canvas.SetActive(true);
        }
    }
```

Add an `OnTriggerExit` method to remove the Rat once the Player moves away to continue the game:

AllyController - OnTriggerExit(Collider other)

```
25  void OnTriggerExit(Collider other)
26  {
27      if (other.CompareTag("Player"))
28      {
29          gameObject.SetActive(false);
30      }
31  }
32
33  // Start is called before the first frame update
34  void Start()
35  {
```

Save your script and return to the Unity Editor.

Click on the **Rat** in the Hierarchy window and find the **AllyController** script in the Inspector window.

The component should now have four new properties.

Debug: The properties won't appear if your script has errors. Check the Console and fix any errors.

From the Hierarchy window drag:

- The Shield child GameObject of the Rat to the Turbo property
- The Shield child GameObject of the Player to the Player Turbo property
- The Player GameObject to the Player property
- The Canvas child GameObject of the Rat to the Canvas property

Test: Run your minigame and make sure the Player speeds up when the turbo has been applied.

Experiment with the values of Move Speed and Rotate Speed whilst in Play mode until you have the turbo effect you want — remember any changes you make here will not be saved when you exit Play mode, so jot down the values then edit them in the script afterward.

Tip: If you can't see the difference in speed from the Game view, you can watch the variables for the Player in the Inspector view. They will change from 3 to 6 when the turbo has transferred to the Player:

Tip: If the shield appears on the wrong character then check the `turbo` property has the Rat's Shield and the `playerTurbo` property has the Player's Shield.

Exit Play mode.

Save your project

Upgrade your project

Well done, you have finished your minigame! Now think about ways you could upgrade your game to change the difficulty, add other collectables and NPCs, or develop the world with extra scenery. You could ask people to play your game and watch them play or ask them for feedback.

You could:

- Use the Enemy tag to write code that ends the game or adds time penalties if the enemies touch the Player
- Add sound effects to the characters when their events are triggered
- Add more allies with child GameObjects that help in other ways such as: destroying enemies, protecting the player, or removing walls

Completed project

You can download the **completed project here** (https://rpf.io/p/en/non-player-characters-get).

Save your project

What next?

If you are following the **Introduction to Unity** (https://projects.raspberrypi.org/en/raspberrypi/unity-intro) pathway, you can move on to the **World builder** (https://projects.raspberrypi.org/en/projects/world-builder) project. In this project, you will design and build you own 3D world for a player to experience.