CODE NINJAS®

Silver Belt Ninja Guide
Activity 02: Find the Exit

Activity 2

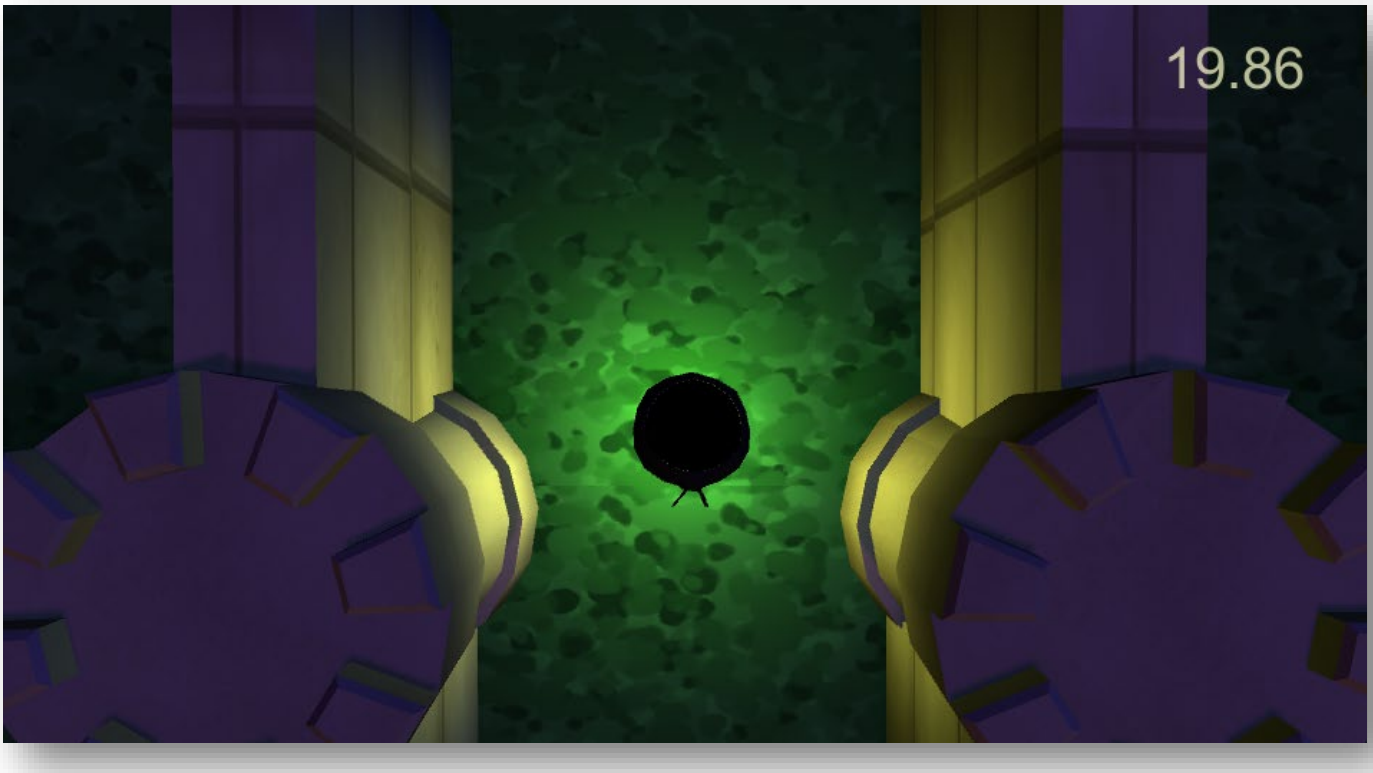
Find the Exit

In this game, you will be able to add movement to a character by *translating* (or moving) the object at a fixed rate of time.

Your mission: Escape the maze in the least amount of time possible. If you press **Play**, you'll discover you are trapped! The player character, Codey, can't move yet because there is no movement code.

Code your player character's movement. This is a key step in making your game because it lets the player character, Codey, move around the game, advance through challenges and reach their goal!

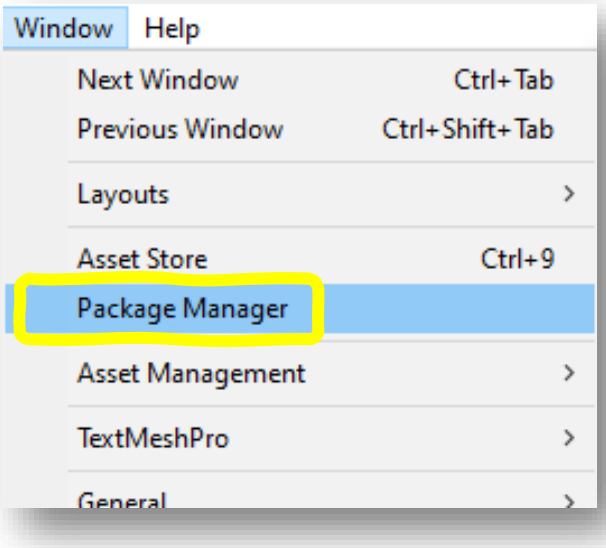
Now that you know your mission, let's get moving!



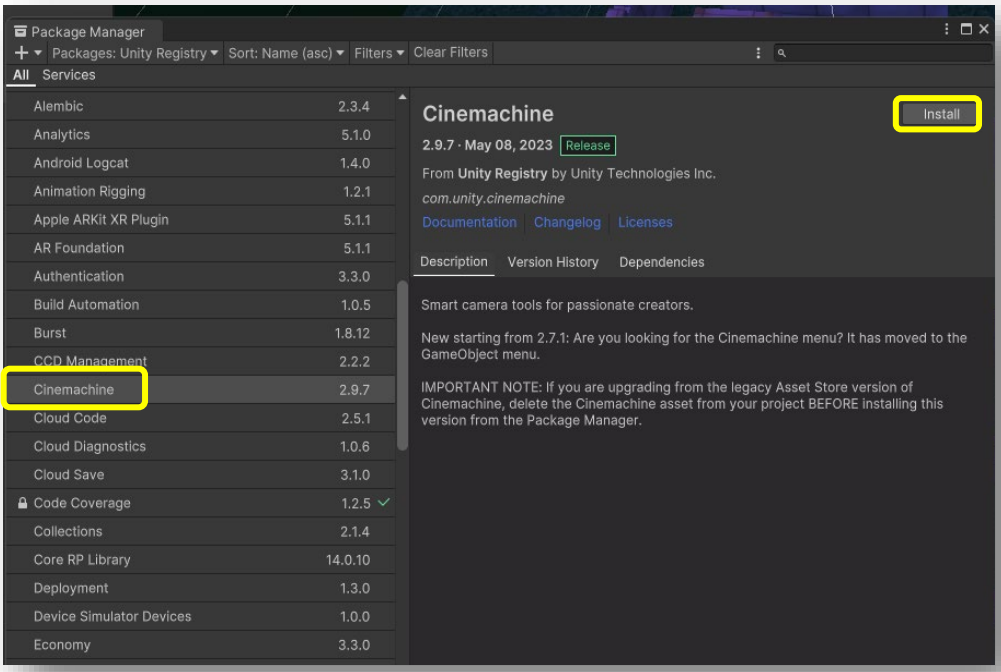
1 Start a new Unity Project and name it *YOUR INITIALS – Find the Exit*. Select **3D core**.

2 We will be using Cinemachine to control our camera. This will help us create awesome camera shots and angles for our game. Go ahead and open **Window > Package Manager**.

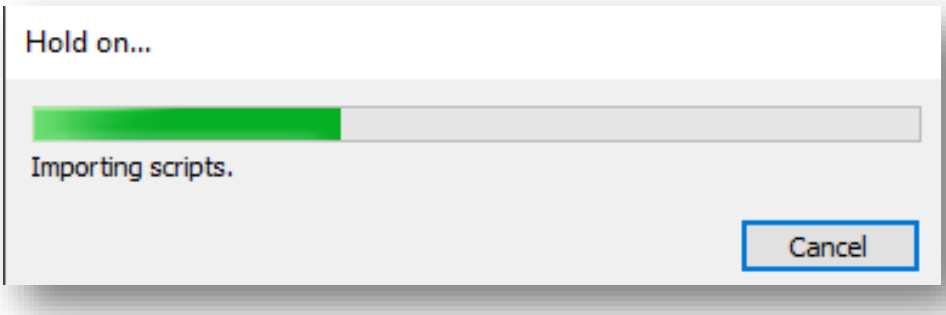
Be patient if it doesn't open right away. It might take a minute to load.



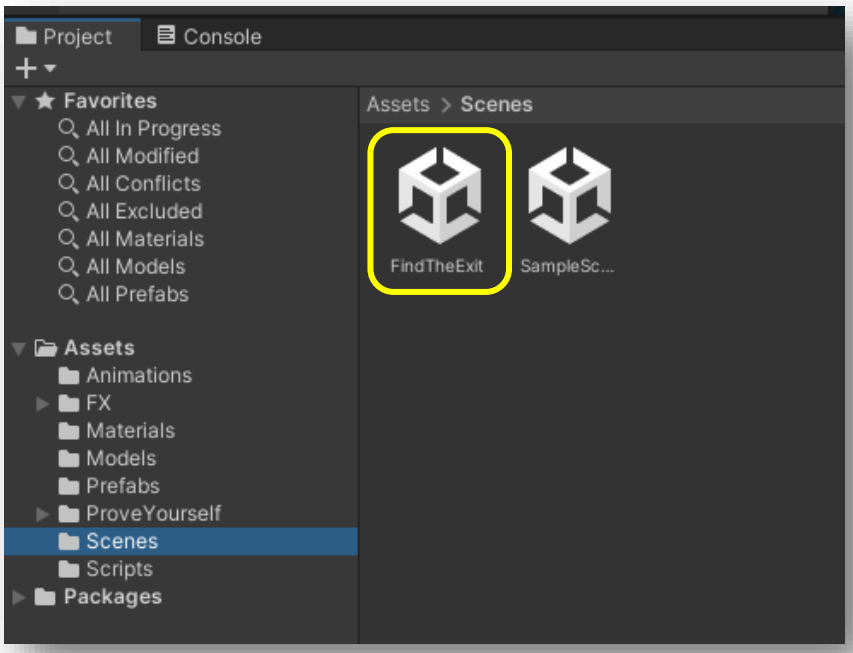
3 Switch to the Unity Registry in the top left corner, then find **Cinemachine** and click **Install** in the top right of the window.



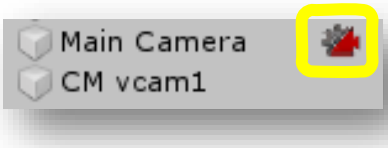
4 We've created a starter pack to give you a head start! To use it, import the **Activity 02 - Find the Exit.unitypackage** by going to **Assets > Import Package > Custom Package> All > Import**.



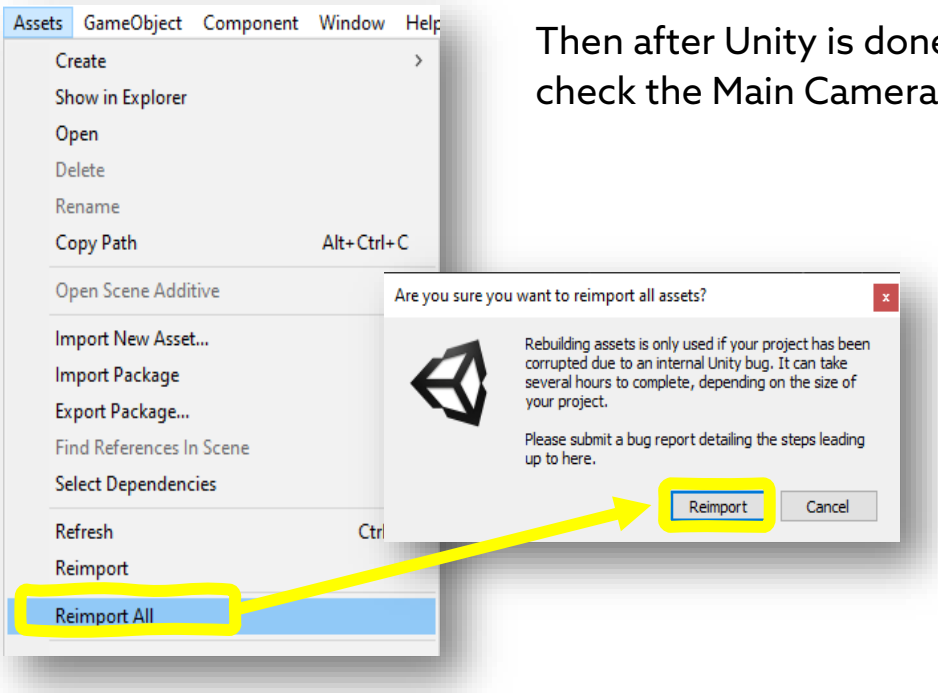
5 To open the starter package, double-click on the **FindTheExit** scene. You can find this in the **Project** tab under **Assets > Scenes > FindTheExit**.



6 We should check that Cinemachine is working like it is meant to. In the **Hierarchy** tab, you should see a list of all the game objects in the scene. If you see a little grey and red icon attached to the Main Camera (shown in the picture), Cinemachine is working!



If you do not see it, go into the **Assets** tab and press **Reimport All**.



Then after Unity is done importing, check the Main Camera again.

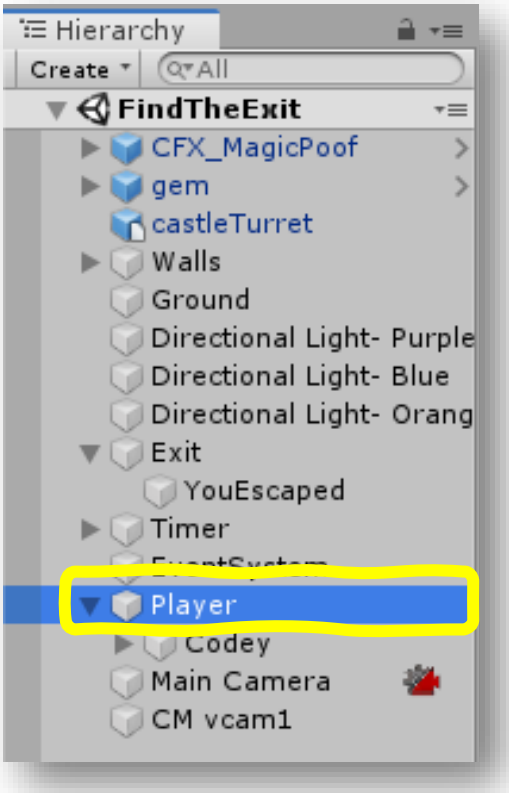
Codey

Codey will be the main character in several Silver Belt games! You will learn how to make him move, jump, and interact with the world around him.

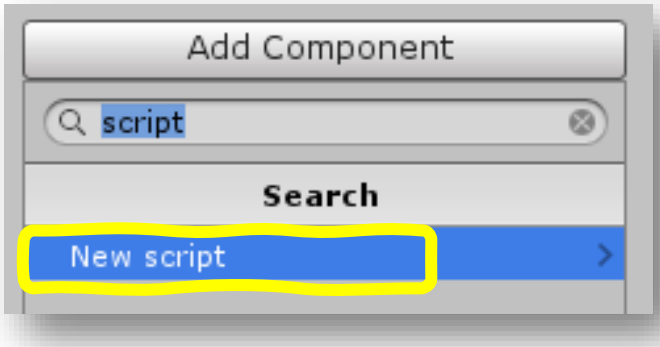
7 Now you can start creating the script to give Codey the ability to move in the game!

To get started, go to the **Hierarchy** and click the **Player** game object to open it in the **Inspector** tab.

The **Inspector** is where you can see all of the components and properties of a single game object in your scene.



8 In the Inspector, click **Add Component** and search for "*script*". Click on **New Script**.



9 Name the script "*PlayerMovement*" and press **Enter** or click **Create and Add**.

In the **Project** tab, move the new **PlayerMovement** script to the **Scripts** folder to keep things organized.

Staying organized is important so you can find what you need when you need it.

10 After you have saved and moved the script to the correct folder, double-click on the **PlayerMovement** script in the **Inspector** to open the file in Visual Studio. This is where you'll add code to get Codey in motion!

11 First, we need to set a speed for Codey. On a new line before `void Start()`, type `public float speed = 5;` to initialize Codey's speed. A `public` variable can be changed in the Inspector. We want the value of speed to be a `float` because that means we can have decimal values and not just whole numbers.

```
public class PlayerMovement : MonoBehaviour
{
    public float speed = 5;

    void Start()
```

12 In the `Update` function, we want to add one line of code to move our ninja, Codey.

To do this, let's try using Unity's `Translate` function on Codey's `transform`. Unity's `Translate` function will let us change an object's position. In the `void Update()` function, type `transform.Translate(Vector3.forward * speed * Time.deltaTime);` to tell Codey to move forward at our set speed of 5 at a constant rate of time.

```
void Update()
{
    transform.Translate(Vector3.forward * speed * Time.deltaTime);
}
```

All we do when we *Translate* an object in Unity or in the real world is move it from one location to another.

Vector 3

A Vector3 is a special object that contains three numbers, one for each of the directions in 3D space. We are using Vector3.forward to ask Unity to give us a vector that points forward.

Time.deltaTime

Time.deltaTime is a fancy way of asking Unity to give the value for how much time has passed since the last time the game loop updated. We use it to make sure any movements and other calculations are not done too slowly or too quickly. We want them to be done exactly right each time!

13 **Save** your code and press **Play**. Codey should translate forward at a speed of 5. But what about at higher speeds? Try changing your speed variable and see what happens at high speeds.

Something seems wrong! Codey can move through walls at high speeds! This is because `transform.Translate` doesn't take into account any walls if the distance it is moving is on the other side of the wall. Instead, let's try using Rigidbody and Velocity!

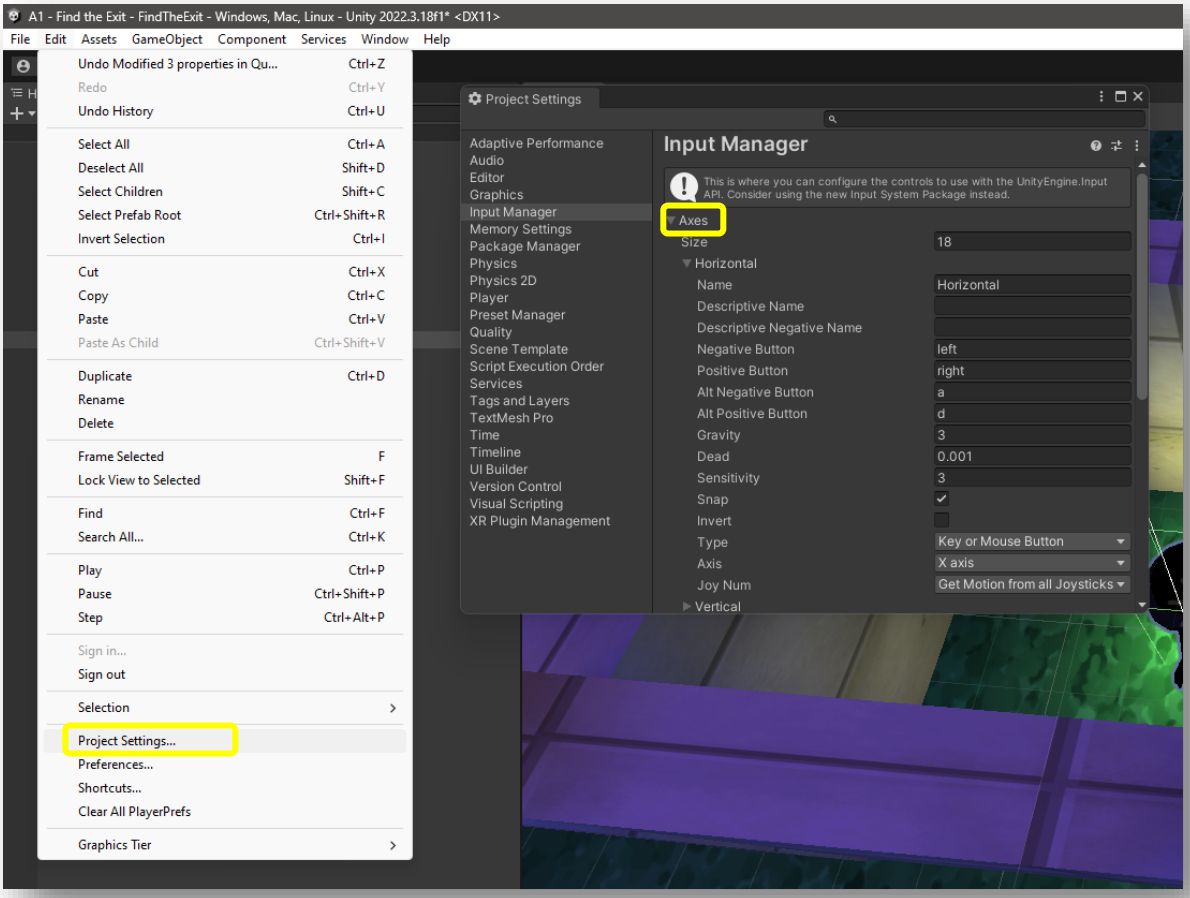
14 Comment out your previous line so that it will not run.

```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    //transform.Translate(Vector3.forward * speed * Time.deltaTime);
}
```

15 Get the Rigidbody component from Codey, and then we will set the velocity to the forward direction at the speed we set. Try out different speeds now and see the difference! Next, let's get Codey moving with arrow keys!

```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    //transform.Translate(Vector3.forward * speed * Time.deltaTime);
    GetComponent<Rigidbody>().velocity = Vector3.forward * speed;
}
```

16 Let's first see how Unity interprets user input. Go to **Edit > Project Settings > Input Manager**. Expand the **Axes** settings by clicking on the drop-down arrow next to it. This **Project Settings** panel will show a lot of behind-the-scenes Unity settings. For Horizontal, notice the `right` arrow and `d` are listed as *Positive Button*, meaning it will output a `1`. While the `left` arrow and `a` are listed as *Negative Button*, meaning it will output a `-1`. Close the **Project Settings** window.



17 Go back to your **PlayerMovement** script. After the `GetComponent<Rigidbody>()` line, log the value of the horizontal axis by typing `Debug.Log(Input.GetAxisRaw("Horizontal"));` in the `Update` function.

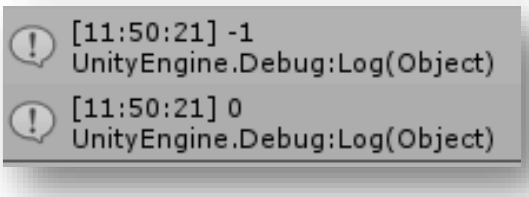
```
void Update()
{
    //transform.Translate(Vector3.forward * speed * Time.deltaTime);
    GetComponent<Rigidbody>().velocity = Vector3.forward * speed;
    Debug.Log(Input.GetAxisRaw("Horizontal"));
}
```

On every frame, Unity will receive the user input, and get the value for the horizontal axis. With Debug, we are then asking Unity to log that value into the console.

18 Before you play the game, what number do you think will be in the log when you press the `a` key? The `d` key?

What about the left and right arrows? What will happen if you don't press any keys? Write your answers down somewhere so you can check to see if you were right!

19 Play your game and look at the console. It will display a `0` if there is no player input. Now press the left arrow or `a`; the console should have a new log saying `-1`. Next, test out the right arrow and `d`. Did you get it right?



Before moving on to the next step, think about what would happen if you wrote *Vertical* in your script instead of Horizontal? Test out your guess! Once you are done, you can delete the `Debug.Log` line from the `Update` function.

20 We now understand how to read player input to get `-1` or `1` depending on which key the user is pressing, but how can we connect this to the movement of Codey?

Remember, we are using `Vector3.Forward` to move our character. This will make Codey always move forward and ignore the user's input. Before we change the `Rigidbody's velocity`, let's make a new `Vector3` based on user inputs!

21 Above the `GetComponent<Rigidbody>()` line, declare a new `float` variable named `horizontal` to store the value of the user input's horizontal axis by typing:
`float horizontal = Input.GetAxisRaw("Horizontal");`

```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    //transform.Translate(Vector3.forward * speed * Time.deltaTime);
    float horizontal = Input.GetAxisRaw("Horizontal");
    GetComponent<Rigidbody>().velocity = Vector3.forward * speed;
}
```

22 On the next line after `float horizontal`, write a similar piece of code but for the `Vertical` axis. Don't forget to check your work!

```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    //transform.Translate(Vector3.forward * speed * Time.deltaTime);
    float horizontal = Input.GetAxisRaw("Horizontal");
    float vertical = Input.GetAxisRaw("Vertical");
    GetComponent<Rigidbody>().velocity = Vector3.forward * speed;
}
```

23 **Play** your game and see what happens.
Codey still always moves in a straight line! This is because while we are getting the user input, we aren't using it to move Codey!

24 On the next line, create a new `Vector3` destination with three parameters of `horizontal`, `0`, and `vertical` by typing `Vector3 destination = new Vector3(horizontal, 0, vertical);`.

This will create a new vector that will change based on the user input! Our `horizontal` variable is in the `x` value spot to move the character left and right, and the `vertical` variable is in the `z` value spot to move the character forward and backward in the game world. The y-axis would be used for moving the character up and down above the game world, like for jumping or falling. If you want to see what this vector looks like, you can `Debug.Log` it to the console.

```
void Update()
{
    //transform.Translate(Vector3.forward * speed * Time.deltaTime);
    float horizontal = Input.GetAxisRaw("Horizontal");
    float vertical = Input.GetAxisRaw("Vertical");
    Vector3 destination = new Vector3(horizontal, 0, vertical);
    GetComponent<Rigidbody>().velocity = Vector3.forward * speed;
}
```

Parameter

A parameter, or argument, is a value or variable that we give to a function by placing it inside parentheses.

25 Codey will still ignore user input because we haven't told Codey to move based on the destination vector we just made. In the final line of `Update`, change `Vector3.forward` to `destination`. Save and test out your game!

```
void Update()
{
    //transform.Translate(Vector3.forward * speed * Time.deltaTime);
    float horizontal = Input.GetAxisRaw("Horizontal");
    float vertical = Input.GetAxisRaw("Vertical");
    Vector3 destination = new Vector3(horizontal, 0, vertical);
    GetComponent<Rigidbody>().velocity = destination * speed;
}
```

26 In the `Update` function, change `Input.GetAxisRaw` to `Input.GetAxis`. Now play the game.

Did you notice a change in how Codey moves? If you didn't, that's okay! `Input.GetAxisRaw` is like an on/off switch. Either the user is pressing a key (value `1`) or the user is not pressing a key (value `0`). `Input.GetAxis` will behave slightly differently.

If a user is not pressing a key, the value will always be `0`. As soon as a user presses a key, the value will start slowly increasing from `0` to `1`.

This will mean that Codey will slowly speed up until the `Input.GetAxis` value reaches `1`.

While it might not make a big difference for all games and player movement code, sometimes this very small difference can make a big impact on the enjoyment of someone playing your game!

This is your game, you decide which way you like better:

`Input.GetAxisRaw` or `Input.GetAxis`!

```
void Update()
{
    //transform.Translate(Vector3.forward * speed * Time.deltaTime);
    float horizontal = Input.GetAxis("Horizontal");
    float vertical = Input.GetAxis("Vertical");
    Vector3 destination = new Vector3(horizontal, 0, vertical);
    GetComponent<Rigidbody>().velocity = destination * speed;
}
```

27 Now that you have Codey responding to the user's input, you can change the `destination` vector!

You can try switching the places of `horizontal` and `vertical`, making one variable negative, changing the `y` value to be `5` instead of `0`, or anything else you want to try.

By changing these values and playing the game to see what happens, you can gain a better understanding of what `Input` and Rigidbody's velocity are doing to your ninja.