Moving Sprites

[Topics covered](#_tgqx56az4g3j)

[Starting a new Project](#_ugf7y2ocebwg)

[Getting Started with Scripts](#_3qz93rvsm9vn)

[Moving a Game Object](#_umr39k93rui)

[Public Variables](#_hhgghxs765mk)

[Keyboard Inputs](#_36ikren0rxc2)

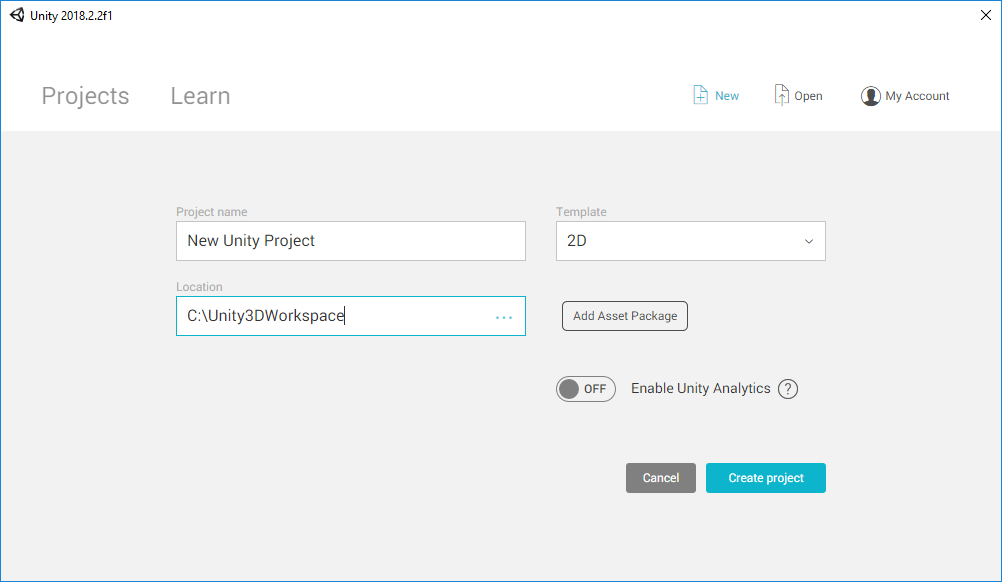
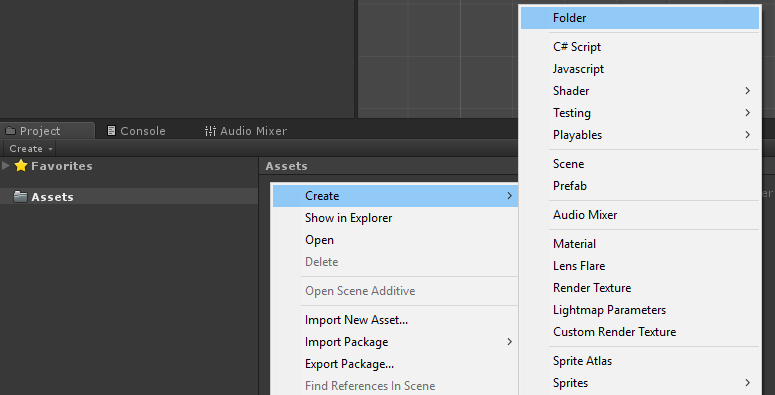
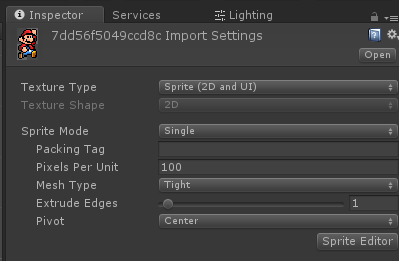
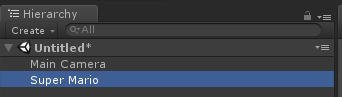
[Moving Continuously](#_if1sncj4ejui)

[Moving Left and Right](#_6cl93qgofgcg)

# Topics covered

* Introduction to Unity
* Naming conventions
* Creating sprites
* Creating scripts
* Start / Update, Frame Cycle
* Console logging and debugging
* Moving an object via its transform
* Input by keyboard
* Public variables
* Frame independent speed

# Starting a new Project

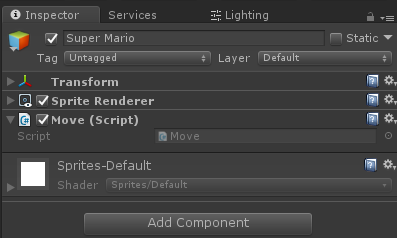
1. Start Unity
2. From the main screen, select **NEW** to create a new project.  
   Select 2D. You can change that later, but for this lecture, we will only focus on 2D.  
   You can give the name you want to your project.  
   
3. Create a new folder called **Sprites**, which will contain the sprites used in the game.  
   Select the **Project** tab. Right click the **Assets** panel > **Create** > **Folder**.Name it **Sprites**.  
   
4. Find or create an image that will be used in this tutorial.  
   Drag it into the **Sprites** folder. You can do this from Unity or from **File Explorer** in Windows.  
   Click on the image: make sure that it is imported as **Sprite (2D and UI)** in the **Inspector**.  
     
   Drag the sprite from the **Assets** panel to the **Scene** window.  
   This will create a new **game object** for the sprite.  
   Move it around until it is visible in the **Game** window as well.  
   

# Getting Started with Scripts

If you start the game, nothing happens. To change the standard behaviour of game objects we can create and attach custom scripts.

1. Create a new folder in the **Project** window, called **Scripts**.
2. Inside it, create a new **C# Script** and call it **Mover**.  
   **⚠** The name of the script is important.  
   Start with a Capital Letter and do not use spaces.  
   In Unity you can rename all assets freely; but not scripts.  
   Renaming them will break their connection with the objects they are attached to.
3. Open the script. It should look like this (below).  
   Make sure the name of the script file (Mover) matches the class name.

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class **Mover** : MonoBehaviour {  // Use this for initialization  void Start () {    }    // Update is called once per frame  void Update () {    }  } |

1. Drag the script on the sprite in the scene.  
   This will attach the script to the sprite.  
   If you have done everything correctly, you will see the script in the **Inspector**.  
     
   At the moment it does nothing, so you will not see any change if you run the game.

# Moving a Game Object

Each **Game Object** in the scene has a component called **Transform**.  
It is used to place to move, scale and rotate an object.  
We can change the transform using the Mover script to move the character.

1. Edit the Start function of the Mover script so that it moves the character to the right when the game start.  
   To do this, we change the value of the property **position** in the **transform**.  
   The position is a **Vector3**, which means that is a wrapper that contains 3 values.  
   To add a value to the position, we need to create another a new **Vector3** and add it.  
   **Vector3**s are created with the keyword **new**.

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class Mover : MonoBehaviour {  // Use this for initialization  void Start () {  **transform.position = transform.position + new Vector3(2, 0, 0);**  }    // Update is called once per frame  void Update () {    }  } |

1. If you run the game, the character should move to the left when it starts.

## 

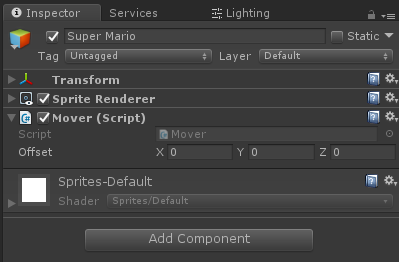
## 

# Public Variables

At the moment the script moves the character by 2 units. We can make that number adjustable from the inspector using **public variables**.

1. Create a public variable called **Offset**, and use its value to replace the previous **Vector3**.

|  |
| --- |
| public class Mover : MonoBehaviour {  **public Vector3 Offset;**  // Use this for initialization  void Start () {  **transform.position = transform.position + Offset;**  }  } |

1. If you go back to the Inspector, you will see that Offset can now be changed by the user.  
   

|  |  |
| --- | --- |
| A Vector3 is made out of 3 elements. If you want to input only one of them, you can use the type **float** which is used for numbers.   |  | | --- | | public class Mover : MonoBehaviour {  **public float X;**  // Use this for initialization  void Start () {  **transform.position = transform.position + new Vector3(X, 0, 0);**  }  } | |

|  |  |
| --- | --- |
| If you want to force the input between two specific values, you can use the attribute **Range**, as a **decorator** of the variable.   |  | | --- | | **[Range(-10,+10)]**  public float X; | |

# Keyboard Inputs

We can use the keyboard to control the movement of the character.  
To do this, we will need to use the **Update** function, which is run one time per frame.  
If the player is pressing a specific key during that frame, we will move the character.

1. Add a conditional statement that requires the space bar to pressed.  
   **Input.GetKeyDown** is the method that is used to check if a key has been pressed.  
   **KeyCode.Space** indicates the spacebar.

|  |
| --- |
| public class Mover : MonoBehaviour {  public Vector3 Offset;  void Update () {  **if (Input.GetKeyDown(KeyCode.Space))**  {  transform.position = transform.position + Offset;  }  }  } |

1. Run and game. Now the character should move only when you press space.

# Moving Continuously

The method Input.GetKeyDown triggers the movement only once per pressing. If you keep pressing the spacebar, the character will not move. If this is not the intended behaviour, you can use **Input.GetKey** instead.

|  |
| --- |
| if (Input.**GetKey**(KeyCode.Space))  {  transform.position = transform.position + Offset;  } |

The character is now moving by **Offset** units per Update call.  
This means that if the game runs at 60 FPS, it is moving at 60 metres per second. But if the game runs at 30 FPS, it will be moving at half the speed.

This behaviour is **frame dependent** (metres per frame). To make it **frame independent** (metres per second) we can use **Time.deltaTime**.

|  |
| --- |
| if (Input.GetKey(KeyCode.Space))  {  transform.position = transform.position + Offset **\* Time.deltaTime**;  } |

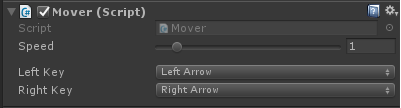
# Moving Left and Right

To move left and right, we can add two **if statements**. One for the left arrow and one for the right arrow.

|  |
| --- |
| **[Range(0,10)]**  **public float Speed;**  void Update () {  if (Input.GetKey(**KeyCode.LeftArrow**))  {  transform.position = transform.position **- new Vector3(Speed, 0, 0)** \* Time.deltaTime;  }  if (Input.GetKey(KeyCode.RightArrow))  {  transform.position = transform.position **+ new Vector3(Speed, 0, 0)** \* Time.deltaTime;  }  } |

As an additional improvement, we can make the keys into public variables.  
If we use the type **KeyCode**, Unity will create a drop-down menu in the inspector.

|  |
| --- |
| **public KeyCode LeftKey;**  **public KeyCode RightKey;**  void Update () {  if (Input.GetKey(**LeftKey**))  {  transform.position = transform.position - new Vector3(Speed, 0, 0) \* Time.deltaTime;  }  if (Input.GetKey(**RightKey**))  {  transform.position = transform.position + new Vector3(Speed, 0, 0) \* Time.deltaTime;  }  } |



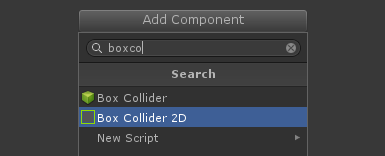
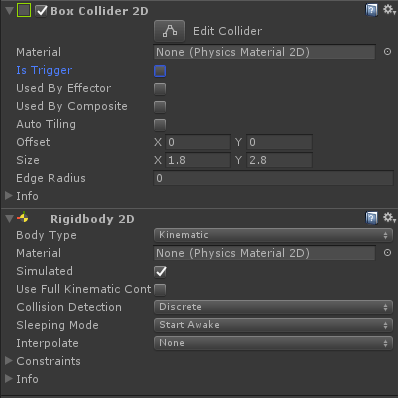
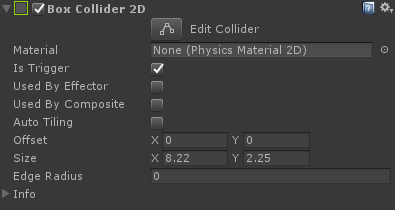
Unity provides a method that allows us to do this automatically.

The method **Input.GetAxis** returns a number that ranges from -1 (left) to +1 (right).  
It is used to control inputs across different devices and controllers.

|  |
| --- |
| void Update () {  **float inputX = Input.GetAxis("Horizontal");**  transform.position = transform.position + new Vector3(**Speed \* inputX**, 0, 0) \* Time.deltaTime;  } |

# Detecting Collisions

Unity comes with an integrated system to handle, detect and resolve collisions between objects. We can use this system to check if the character is in a specific region.

1. Select the character. Go in the Inspector. Add a new **BoxCollider2D**.  
   
2. Collisions are deeply connected with physics.  
   If you want to make full use of the collision system, at least one of the objects involved should have a **Rigidbody2D**.  
   Add a **Rigidbody2D** as well.  
   Since we do not want physics to affect our character, mark it as **Kinematics**.  
   
3. Create a new sprite for the collision area, and attach a **BoxCollider2D** to it.  
   Mark that colliders as **Trigger**, since it will be a walkable area, not a solid object.  
   
4. The collision systems is event based. When a collision occurs, Unity calls a special method inside the class. There are three that we can use:

|  |
| --- |
| void OnTrigger**Enter**2D(Collider2D collision)  {  Debug.Log("ENTER!");  }  void OnTrigger**Stay**2D(Collider2D collision)  {  Debug.Log("STAY!");  }  void OnTrigger**Exit**2D(Collider2D collision)  {  Debug.Log("EXIT!");  } |