



# Camera Follow + Zoom



# Exercise #18 - Camera Follow

- To make the camera follow your player throughout the level, you must create a script and attach it to your Main Camera
- **Create New Component > New Script > Name it CameraFollow** or any suitable name
- Attach CameraController to Main Camera
- Open script in Visual Studio

# Exercise #18 - Camera Follow (Cont.)

- Create a public variable of type Transform. This will be the object the camera must follow
  - Reminder: any GameObject in your game has a Transform component. It contains the rotation, position and scaling data of that game object
- Create a public float and name it CameraSpeed. This value controls how quickly your camera follows the player game object. If your camera's movement is jittery or too slow, you may need to tweak the value
- Control the borders/limits of your camera by creating 4 variables, minX and maxX, minY and maxY

# Exercise #18 - Camera Follow (Cont.)

```
public Transform Target;
public float Cameraspeed;

//Determine the minimum and maximum values in both directions where the camera can move. This is to stop it
//from showing game edges or things you don't want to appear to the player
public float minX, maxX;
public float minY, maxY;

void Start()
{
```

# Exercise #18 - Camera Follow (Cont.)

- Create a FixedUpdate() function.
  - As long as there's a target to follow (the player character), the camera will always be updating its position to have the same position as that character. Since we're in a 2D game, we will use a variable of type Vector 2. The new camera position is calculated as follows:

```
{  
    if(Target!=null)  
    {  
        Vector2 newCamPosition = Vector2.Lerp(transform.position, Target.position, Time.deltaTime * Cameraspeed);  
    }  
}
```

- The Lerp function stands for *Linear Interpolation*. It will help us calculate how fast the camera will move across time, and how it will follow the player game object
- The camera's new position is determined by taking its previous position, target's position, and speed multiplied by time that passed since the last frame

# Note on Lerp Function

- One way to explain Linear Interpolation is that it is a mathematical formula designed to take **3 values**: the FROM/BEGINNING point, the TO/ENDING point, and the smoothness by which you arrive to the ending point
- Example: If you wish to move from point 1 to point 3 on the x-axis, you provide that the FROM value is 1, the TO value is 3, and the smoothness could be 0.1. As you move between points 1 and 3, you'll automatically pass through points 1.1, 1.2, 1.3...2.2,2.3...etc. until you arrive at 3

## Exercise #18 - Camera Follow (Cont.)

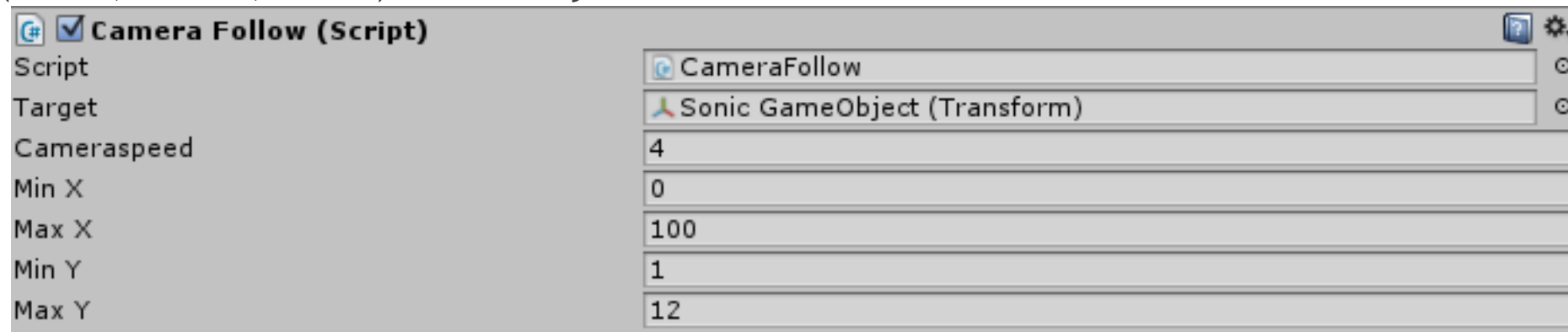
- Immediately beneath the Lerp function and before the If-condition is over, we add 2 private variables, ClampX and ClampY. We use them to control the limits of the camera's movement
- Use the Mathf.Clamp function, which needs 3 variables: current Camera position, and the minimum and maximum values on the x and y axis you don't want to exceed

```
float ClampX = Mathf.Clamp(newCamPosition.x, minX, maxX);  
float ClampY = Mathf.Clamp(newCamPosition.y, minY, maxY);  
  
transform.position = new Vector3(ClampX, ClampY, -10f);  
}
```

- Finally, update the newest camera position by assigning a Vector3 of values ClampX as x, ClampY as y, and -10f as z

# Exercise #18 - Camera Follow (Cont.)

- Go back to Unity
- Give your target variable the player game object by dragging it from the Hierarchy window, and experiment with any speed value. Play around with your camera limits (minX, maxX, etc.) and run your scene

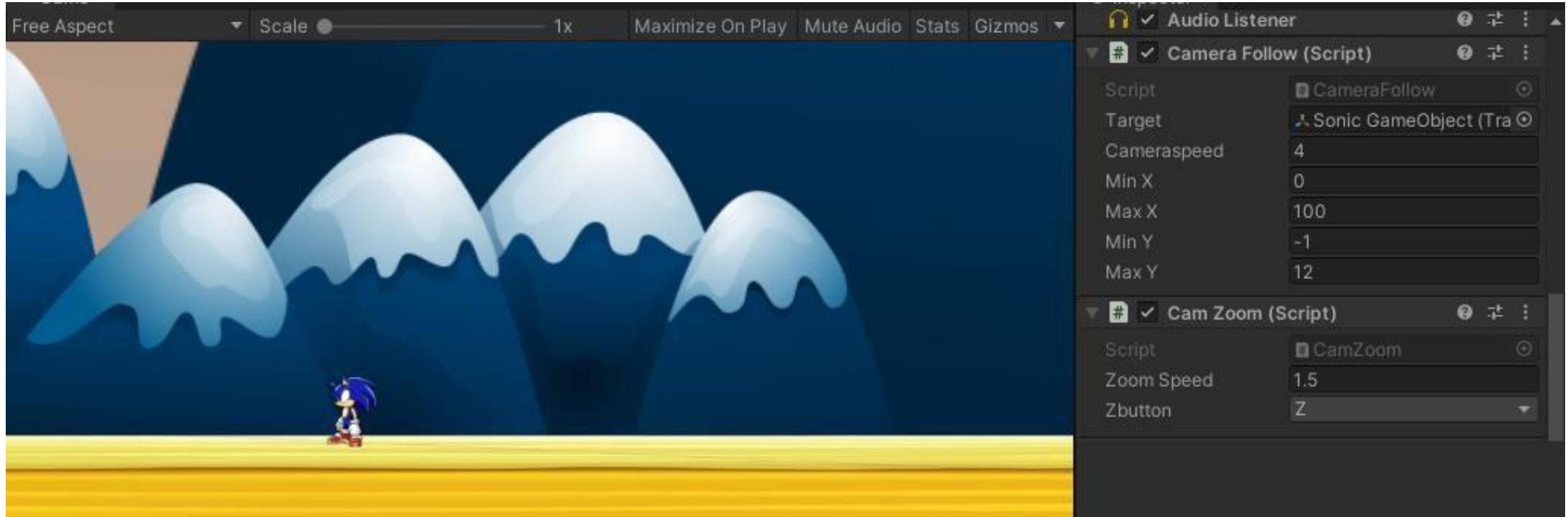


- If your camera moves smoothly, but your player game object doesn't, go to its Rigidbody2D component and turn on Interpolation.
- (Your camera movement may be a bit laggy anyway if the computer's graphics card is limited)



# Camera Zoom

- To make your Main Camera zoom in and out, it can be done via Button press, or Mouse Scroll
- To create Zoom via button press, create a new script and name it Cam Zoom



## Exercise #19 – Camera Zoom (Button Press)

```
public class CamZoom : MonoBehaviour
{
    private Camera Cam; //Will use this to GetComponent

    public float ZoomSpeed; //Speed of zooming in or out
    public KeyCode Zbutton; //Letter Z on the keyboard

    // Start is called before the first frame update
    void Start()
    {
        Cam = GetComponent<Camera>(); //GetComponent<>()
    }
}
```

# Exercise #19 – Camera Zoom (Button Press)

## (Cont.)

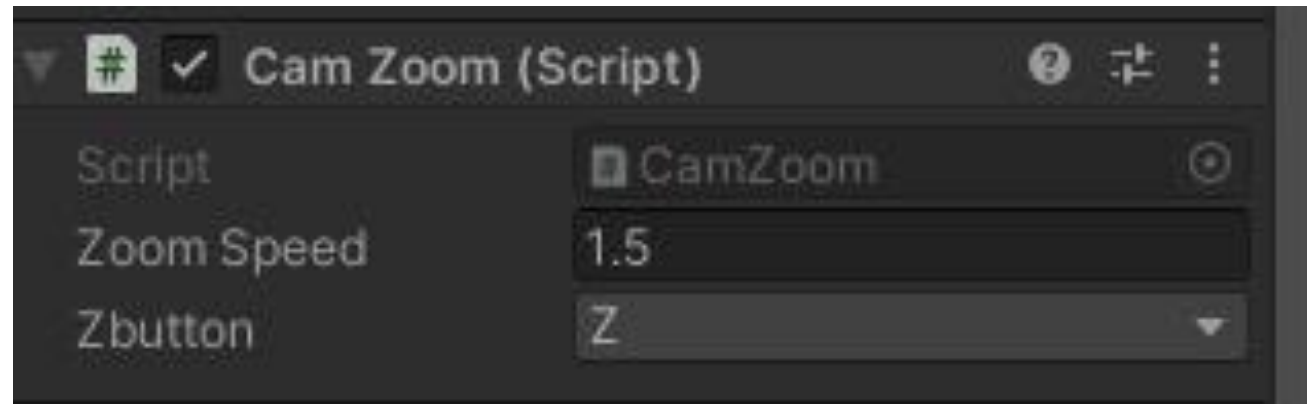
- Create a FixedUpdate() function and determine that *if* the player presses Z, the camera's size changes from the original orthographic size, to the smaller value (also set by YOU)
- Use the Lerp function again to smooth the effect from value A to value B
- Else if the player removes their finger, the size returns to normal, also using
- Lerp to smooth out the movement from small to big

```
if (Input.GetKey(Zbutton))
{
    Cam.orthographicSize = Mathf.Lerp(Cam.orthographicSize, 4, Time.deltaTime * ZoomSpeed);
}

else
{
    Cam.orthographicSize = Mathf.Lerp(Cam.orthographicSize, 6, Time.deltaTime * ZoomSpeed);
}
```

# Exercise #19 – Camera Zoom (Button Press) (Cont.)

- Make sure to update the values of your public variables in Unity and test your game



# Exercise #19 – Camera Zoom (Mouse Scroll)

- Now create another Script and call it something relevant, like Mouse Zoom



# Exercise #19 – Camera Zoom (Mouse Scroll) (Cont.)

- Create the following variables and assign them

```
private Camera Cam;

public float TargetZoom = 3; //The value of Zoom I want by manipulating the Camera Size

private float ScrollData; //Float collected upon Mouse Scrolling

public float ZoomSpeed = 3; //Speed of zooming process in or out

void Start()
{
    Cam = GetComponent<Camera>();

    TargetZoom = Cam.orthographicSize; //The game will begin with TargetZoom being assigned the default given Camera size
}
```

# Exercise #19 – Camera Zoom (Mouse Scroll) (Cont.)

- Now in the Update() function, use Input.GetAxis(“Mouse ScrollWheel”) to gather float value input and store it in the ScrollData variable

```
// Update is called once per frame
void Update()
{
    //when the Mouse is still, this function returns 0. Forward scrolling returns a +ve value, Bakward scrolling return -ve value.
    ScrollData = Input.GetAxis("Mouse ScrollWheel");
}
```

- Update the TargetZoom value by subtracting from the ScrollData value

```
//Update the Target Zoom to change from the default to however much I'm scrolling up or down.
TargetZoom = TargetZoom - ScrollData;
```

# Exercise #19 – Camera Zoom (Mouse Scroll)

- To make sure your zooming in and out is not too extreme, Clamp your camera to not exceed certain minimum and maximum values

```
//Clamp the Camera and set limits to avoid zooming in or out too much  
TargetZoom = Mathf.Clamp(TargetZoom, 3, 6);
```

- Finally, to make the Zoom function smooth, use the Lerp function to move up or down from size A to size B

```
//Lerp function to smooth the transition from size 1 to size 2, AKA from current Orthographic Size to Target Orthographic Size  
Cam.orthographicSize = Mathf.Lerp(Cam.orthographicSize, TargetZoom, Time.deltaTime * ZoomSpeed);
```

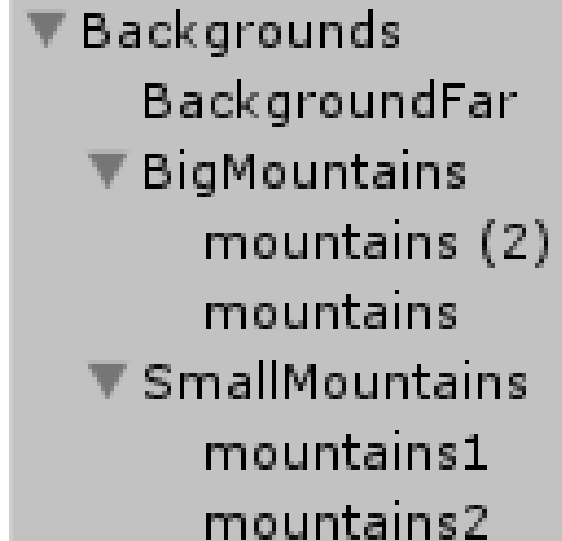


# Parallax Effect

- To give your game more visual depth, you can create what is called a Parallax Effect
- This effect makes your game's background feel like a car ride: Elements in the far horizon seem to move very slowly, while closer elements such as trees move by much faster
- Parallax effect can be written in many different ways. The following exercise #19 is one variation. Create a new script, name it BackgroundParallax and attach it to the Main Camera
- Note: make sure you don't leave spaces in a script file's name

# Extra Exercise – Parallax Effect

- Design your game's background however you want, and make sure all the elements are on the Background Layer
- Create a GameObject and call it Background. This will be the parent that contains all the background elements
- Inside Background, make more divisions. For example, you can have brown mountains at the very back, green mountains in the middle,
  - and trees at the front



```
▼ Backgrounds
    BackgroundFar
        ▼ BigMountains
            mountains (2)
            mountains
        ▼ SmallMountains
            mountains1
            mountains2
```

# Extra Exercise – Parallax Effect (Cont.)

- After you've designed your background, go to your Parallax script and open in Visual Studio. Declare the following variables

```
public Transform[] Backgrounds;//an array of backgrounds
public float ParallaxScale; //proportion of camera's movement to move backgrounds by
public float ParallaxReductionFactor;//as we move along in the array, we want the farthest backgrounds to move a little
//slower. This value allows us to make the furthest background move slower than the closer backgrounds. The closest
//moves fastest, and speed becomes slower and slower as we move backwards through the backgrounds
public float smoothing;//the smoothness of the parallax effect

private Vector3 lastPosition;//the position of the camera in the previous frame
```

# Extra Exercise – Parallax Effect (Cont.)

- In the Start() function, add:

```
lastPosition = transform.position; //assign the camera's previous position to be the camera's position right now
```

- This initializes your camera's position. Unity does not know where exactly your camera is when this script starts executing, so the camera's position from the last frame is assigned to be the camera's position in the current frame

# Extra Exercise – Parallax Effect (Cont.)

- In the Update() function, add:

```
var parallax = (lastPosition.x - transform.position.x) * ParallaxScale; //calculate the difference between the camera's  
//position from this frame and its position from last frame
```

- Create a var named parallax. This variable will contain the difference between the camera's position from this frame and its position from last frame along x-axis, and multiply by the ParallaxScale. This gives you the overall speed the camera is supposed to move.

# Extra Exercise – Parallax Effect (Cont.)

- Create a loop that iterates through the backgrounds array

```
for(int i = 0; i < Backgrounds.Length; i++)
{
    var backgroundTargetPosition = Backgrounds[i].position.x + parallax * (i * ParallaxReductionFactor + 1);
    Backgrounds[i].position = Vector3.Lerp(
        Backgrounds[i].position,
        new Vector3(backgroundTargetPosition, Backgrounds[i].position.y, Backgrounds[i].position.z),
        smoothing * Time.deltaTime);
}
```

- Create a var and name it backgroundTargetPosition. This variable contains the location the background is supposed to be at the next frame (along x-axis).
- Assign it the value of the current x-position of the background layer you are currently at, and add the parallax factor (the camera position at that particular moment). Multiply the current iteration (background layer) by the Parallax Scale Factor
- Inside the loop, we need a statement that makes every layer of background slower than the layer before it.

# Extra Exercise – Parallax Effect (Cont.)

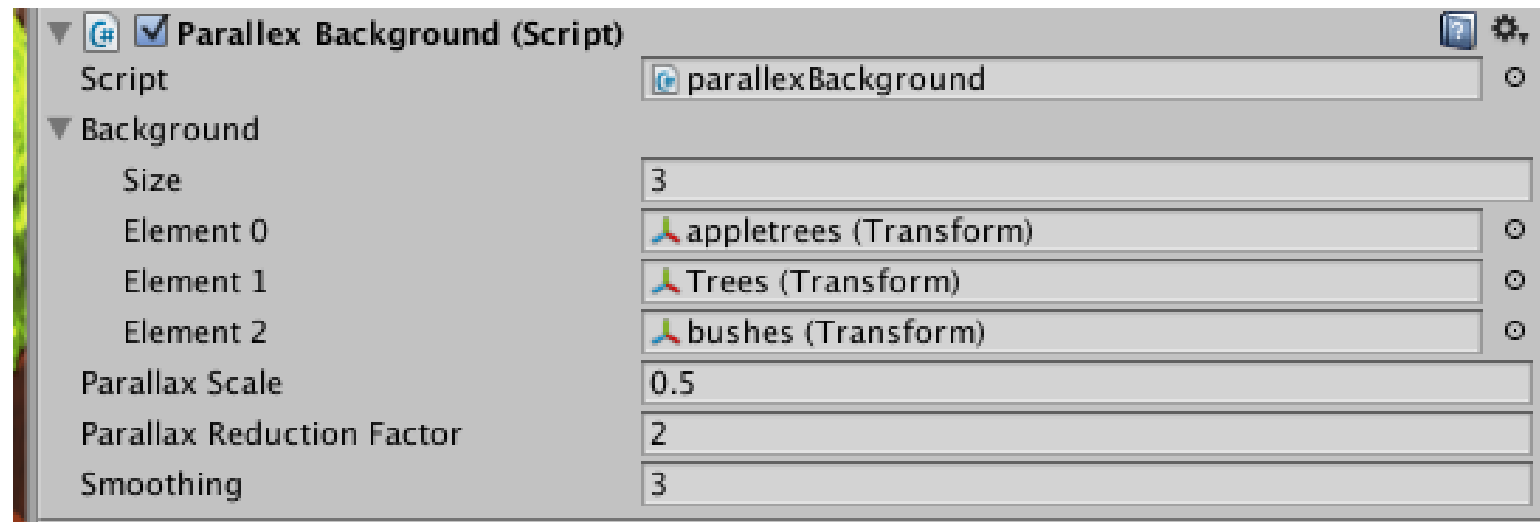
```
for(int i = 0; i < Backgrounds.Length; i++)
{
    var backgroundTargetPosition = Backgrounds[i].position.x + parallax * (i * ParallaxReductionFactor + 1);
    Backgrounds[i].position = Vector3.Lerp(
        Backgrounds[i].position,
        new Vector3(backgroundTargetPosition, Backgrounds[i].position.y, Backgrounds[i].position.z),
        smoothing * Time.deltaTime);
}
```

- Use the Lerp function to determine the FROM value as the current background layer's position, The TO value which is where the background is supposed to be (note that it takes 3 values because it's a vector3 type), and the smoothing factor multiplied by the time that passes between each frame and the next when the game runs

# Extra Exercise – Parallax Effect (Cont.)

- Go back into Unity and see the public variables appear as the script's properties
- Drag the Background super parent from the Hierarchy and drop it onto the Background property in the Inspector as seen below
- Drag each of the 3 background children and drop them into the text boxes as seen below
- Experiment with the values of the Parallax Scale,

Reduction Factor, and Smoothing





# Extra Exercise – Parallax Effect (Cont.)

- After the loop ends, do not forget to update the camera's position by adding
  - once more the statement:

```
lastPosition = transform.position;
```

- So when the Update function is called against for the next frame, the camera is where it should be
- Run your scene

# References

- Unity Parallax Tutorial - Infinite Scrolling Background. URL Retrieved from: <https://www.youtube.com/watch?v=zit45k6CUMk>
- Shooting Projectiles & Camera Control - Unity 2D Platformer Tutorial - Part 8. URL retrieved from: <https://www.youtube.com/watch?v=8aVZuL9ocrk>
- Creating 2D Games in Unity 4.5 #15 - Camera Controller. URL retrieved from: <https://www.youtube.com/watch?v=u67fbxe8xxY>
- 2D RPG Smooth Camera - Unity3D. URL retrieved from: <https://www.youtube.com/watch?v=KMhPYf9zzlA>
- How to make a 2D Platformer - Parallax Scrolling - Unity Tutorial. URL Retrieved from: [https://www.youtube.com/watch?v=5E5\\_Fquw7BM](https://www.youtube.com/watch?v=5E5_Fquw7BM)
- Unity 5 2D Platformer Tutorial - Part 8 - Camera Bounds. URL Retrieved from: <https://www.youtube.com/watch?v=l6xmOMsRWeo>