



# Tutorial four: Spawning an item

Game Programming Project  
By Mariana Neiva Santos Silva

# What you'll learn

In this Tutorial you will learn how to spawn an item in a 2D game in Unity.



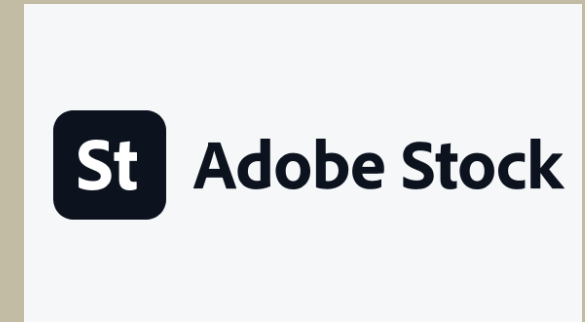
# Programs used



UNITY  
Game Engine



VISUAL STUDIO  
Code Editor



ADOBE STOCK  
Stock images

# What you should already know:

1 A basic understanding of **Unity**;

---

2 Basic understanding of **C#**

---

2 Have followed tutorial 1, 2 and 3

---

The background features a light gray base with large, organic, overlapping shapes in muted olive green and a dusty rose color. In the top left corner, there is a stylized, light gray illustration of a pine branch with needle-like leaves. A thin, white, wavy line curves across the bottom right portion of the image.

Let's beginning!



# Steps

Step 1: Setting up the project  
9

---

Step 2: Creating The Items  
10

---

Step 3: Creating the Buttons.  
12

---

Step 4: The Script  
16

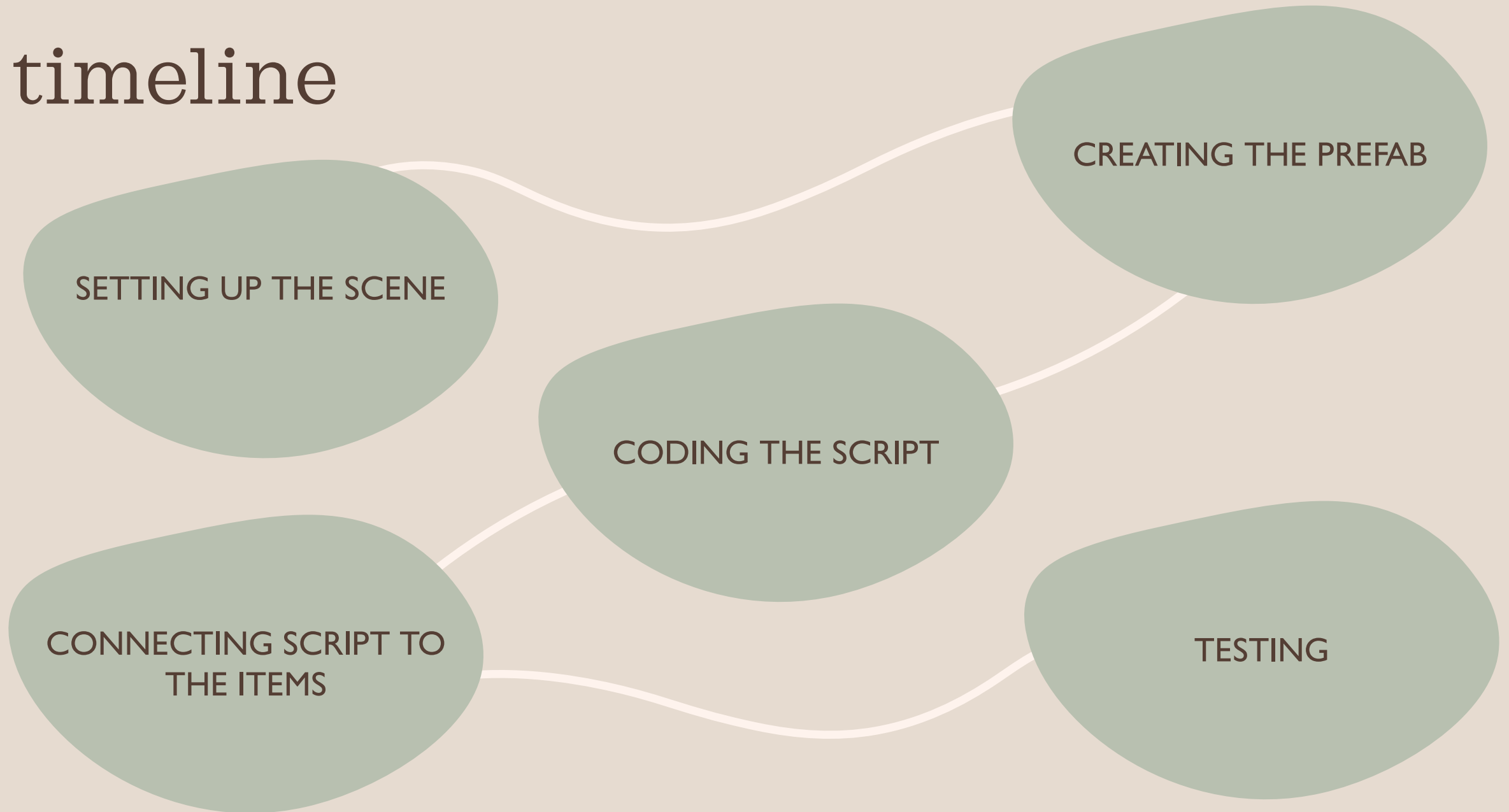
---

Step 5: Connecting the buttons  
22

---

Step 6: Testing.  
28

# timeline



# Step 1: Setting the scene

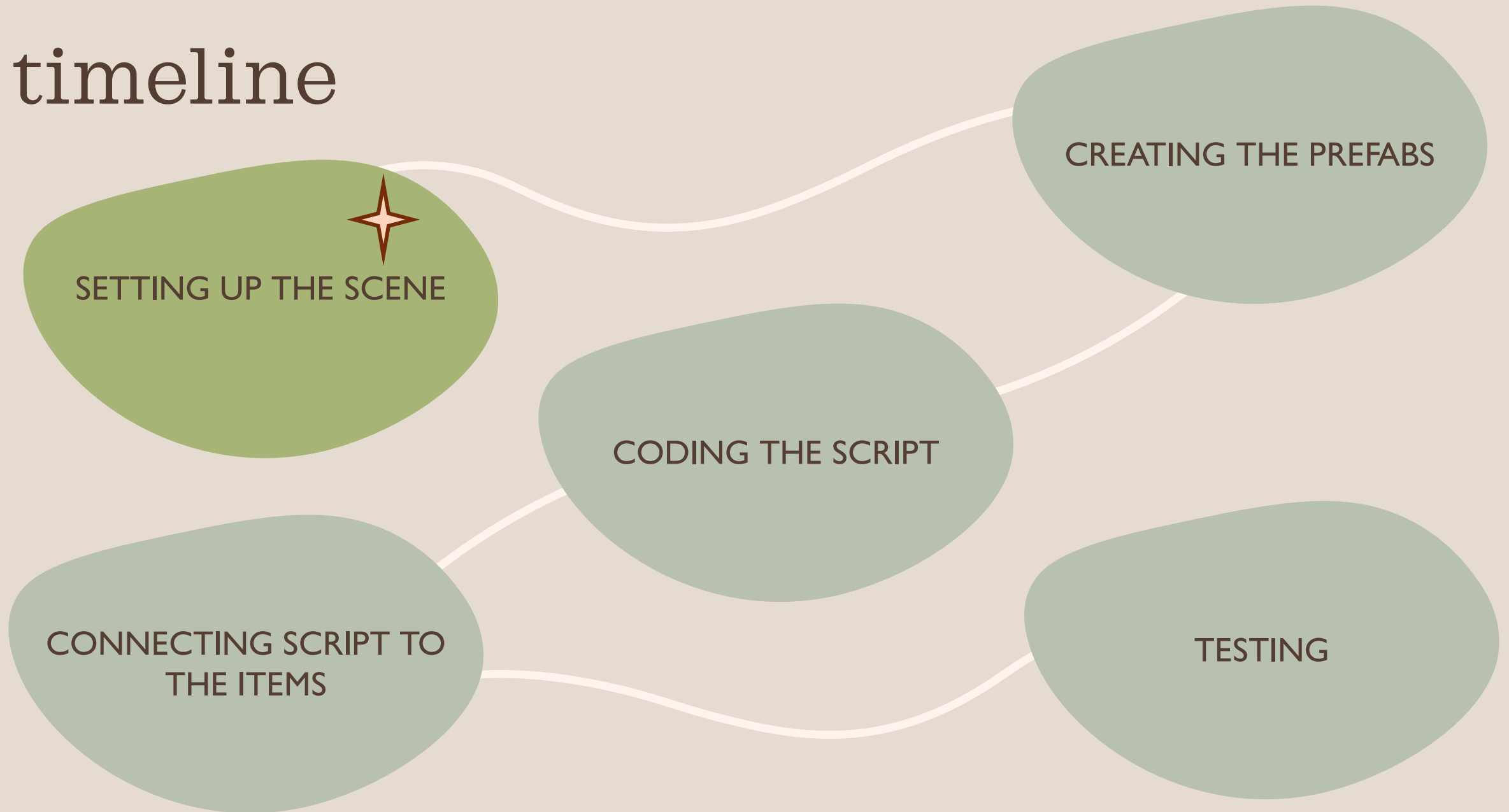
## THE SCENE:

- For this tutorial we will be using the same scene and script from the previous tutorials.
- If you want to add anything extra to your scene now is the time!
- For example, I added a Quest board, with the skills I showed in the previous tutorials





# timeline



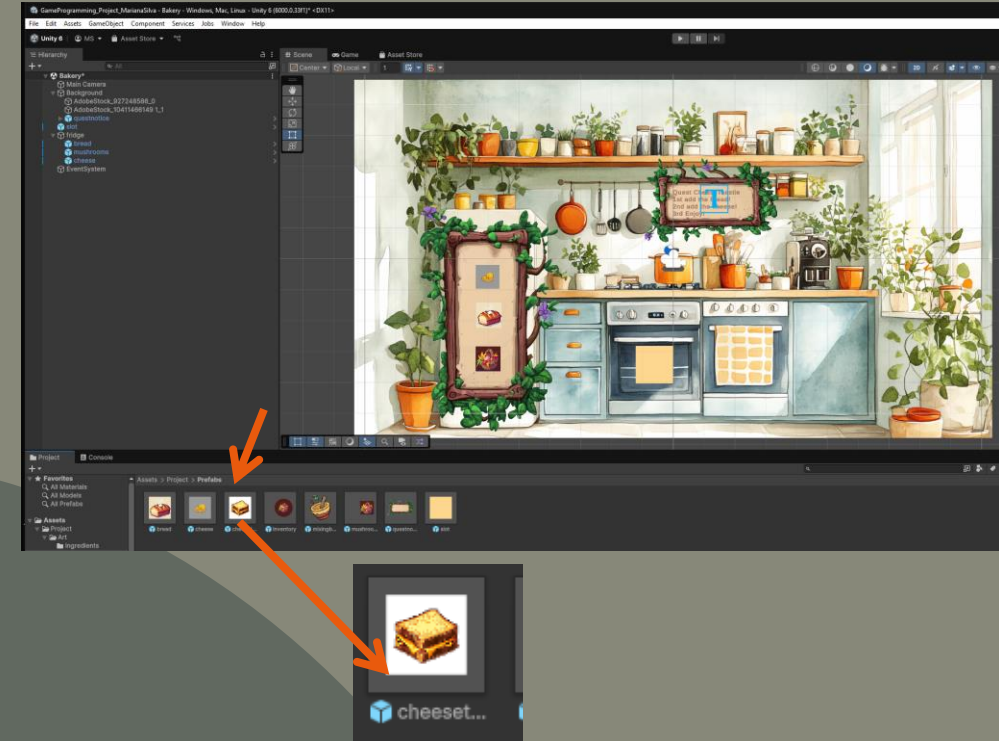
# Step 2: Creating the Prefabs:

## WHAT IS A PREFAB?

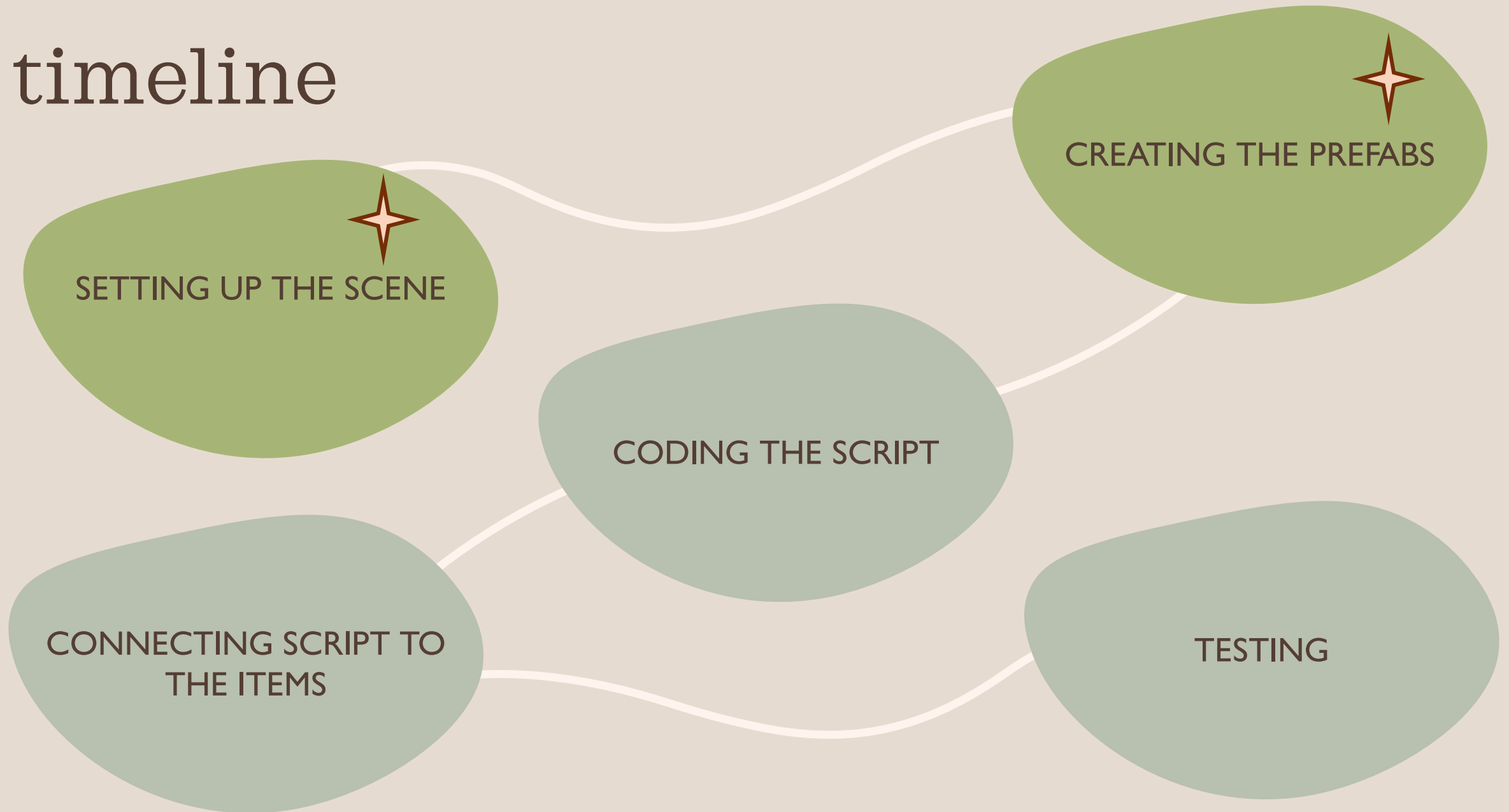
- As I said before a prefab is an asset you can re-use.
- The reason as to why we will be using one now is because we don't want the item to be in the scene when it starts but on when we combine 2 items.

## CREATING A PREFAB.

- Now create a prefab of whatever asset you want to spawn and drag it on to the project.



# timeline



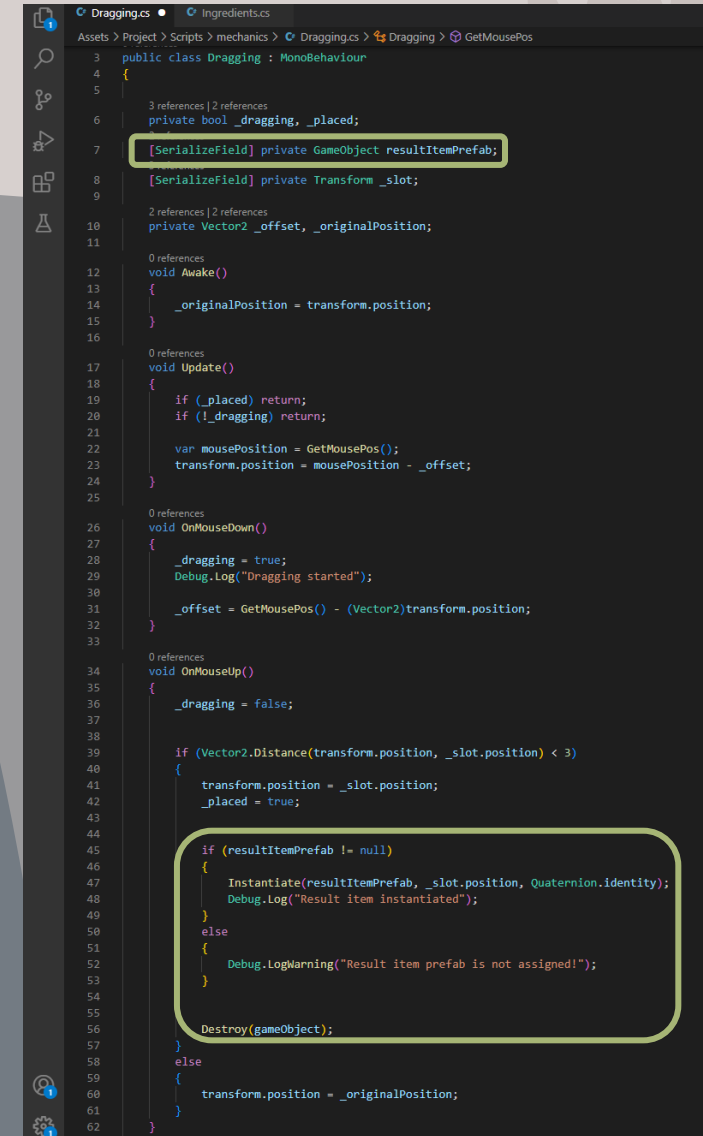
# Step 3: The Script

## ADDING TO THE SCRIPT

- We will be adding onto the script from the last tutorials as highlighted.
- If you change the name of the script in Unity, the script will not update itself.
- My script is called “Dragging” if I want to change it to DraggingAndDropping I would need to change it in Unity and in the script in the line “**public class Dragging : MonoBehaviour**”.
- It would now be “**public class DraggingAndDropping : MonoBehaviour**”.

## THE SCRIPT

- This script will allow you to drag and drop an object around in Unity and of course spawn an item if the player adds the item you chose.

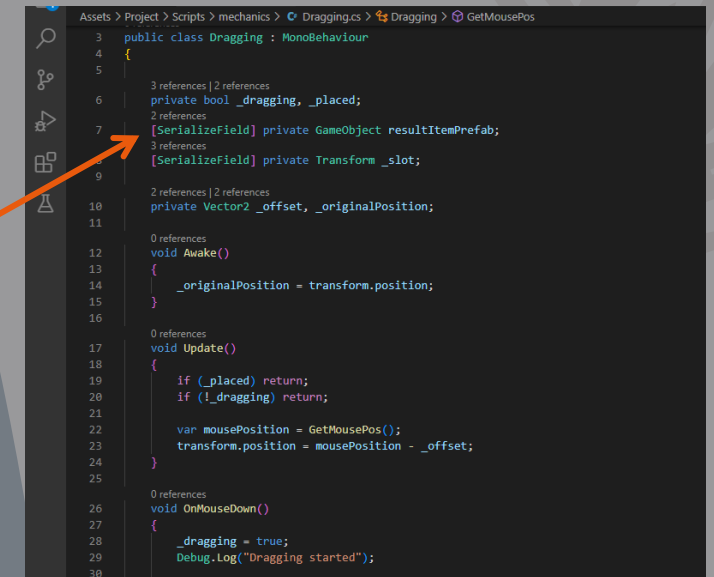


```
Assets > Project > Scripts > mechanics > Dragging.cs > Dragging > GetMousePos
3 public class Dragging : MonoBehaviour
4 {
5
6     3 references | 2 references
     private bool _dragging, _placed;
7
8     [SerializeField] private GameObject resultItemPrefab;
9
10    [SerializeField] private Transform _slot;
11
12    2 references | 2 references
     private Vector2 _offset, _originalPosition;
13
14    0 references
     void Awake()
15    {
16        _originalPosition = transform.position;
17    }
18
19    0 references
     void Update()
20    {
21        if (!_placed) return;
22        if (!_dragging) return;
23
24        var mousePosition = GetMousePos();
25        transform.position = mousePosition - _offset;
26    }
27
28    0 references
     void OnMouseDown()
29    {
30        _dragging = true;
31        Debug.Log("Dragging started");
32
33        _offset = GetMousePos() - (Vector2)transform.position;
34    }
35
36    0 references
     void OnMouseUp()
37    {
38        _dragging = false;
39
40        if (Vector2.Distance(transform.position, _slot.position) < 3)
41        {
42            transform.position = _slot.position;
43            _placed = true;
44
45            if (resultItemPrefab != null)
46            {
47                Instantiate(resultItemPrefab, _slot.position, Quaternion.identity);
48                Debug.Log("Result item instantiated");
49            }
50            else
51            {
52                Debug.LogWarning("Result item prefab is not assigned!");
53            }
54
55            Destroy(gameObject);
56        }
57        else
58        {
59            transform.position = _originalPosition;
60        }
61    }
62 }
```

# Step 3: Understanding the script

## ADDING VARIABLES

- This line sets the second variable:
- “[SerializeField] private GameObject *resultItemPrefab*;
- Let’s break it down:
  - While this variable is private “[SerializeField] ” forces unity to “serialize” or in other words show us the private variable. However, it is important to be aware this doesn’t work for all variables.
  - “*GameObject*” makes this variable an asset from our game, in this case we will make it our prefab.
  - Therefore, by declaring the variable “*resultItemPrefab*” that we want to be able to add a GameObject in the inspector of whatever asset we add to the script.




The screenshot shows a C# script named `Dragging` in the `Assets > Project > Scripts > mechanics > Dragging.cs` file. The script is a `MonoBehaviour` class. An orange arrow points to line 7, which is the declaration of the `resultItemPrefab` variable. The code is as follows:

```
3 public class Dragging : MonoBehaviour
4 {
5
6     3 references | 2 references
     private bool _dragging, _placed;
7     2 references
     [SerializeField] private GameObject resultItemPrefab;
8     3 references
     [SerializeField] private Transform _slot;
9
10    2 references | 2 references
     private Vector2 _offset, _originalPosition;
11
12    0 references
     void Awake()
13    {
14        _originalPosition = transform.position;
15    }
16
17    0 references
     void Update()
18    {
19        if (!_placed) return;
20        if (!_dragging) return;
21
22        var mousePosition = GetMousePos();
23        transform.position = mousePosition - _offset;
24    }
25
26    0 references
     void OnMouseDown()
27    {
28        _dragging = true;
29        Debug.Log("Dragging started");
30    }
```

# Step 3: Understanding the script

## AN IF STATEMENT

- We are using an If statement to either (if) place the item or (else) return it to its original position.
- In the If statement we created in the last tutorial in “**void OnMouseUp()**” we will be more code.
- This time we will add an if statement inside another if statement!
- In essence this will work as follows:
  - Since we added it inside the If part of the If statement and not the else, if the player meets the requirements our new if statement will run after the 2 first lines before it.
  - This new If statement will spawn a prefab if we assign one.




```
34 void OnMouseUp()
35 {
36     _dragging = false;
37
38     if (Vector2.Distance(transform.position, _slot.position) < 3)
39     {
40         transform.position = _slot.position;
41         _placed = true;
42
43         if (resultItemPrefab != null)
44         {
45             Instantiate(resultItemPrefab, _slot.position, Quaternion.identity);
46             Debug.Log("Result item instantiated");
47         }
48         else
49         {
50             Debug.LogWarning("Result item prefab is not assigned!");
51         }
52
53         Destroy(gameObject);
54     }
55     else
56     {
57         transform.position = _originalPosition;
58     }
59 }
60
61
```

# Step 3: Understanding the script

## AN IF STATEMENT

- The requirements of this If statement:
- “***if (resultItemPrefab != null)***”
  - This is checking if the variable we created is empty or not. The symbol “***!=***” means not equal to and “***null***” means that it has no value (or it zero), in this case that it has zero prefabs assigned.
- “***Instantiate(resultItemPrefab, \_slot.position, Quaternion.identity);***”
- This spawns an item in unity.
- How does unity know where to spawn what?
- When we declare “***Instantiate()***” there needs to be 3 things inside like this:
  - The prefab you want to copy – “***resultItemPrefab***”
  - The position you want to add it to – “***\_slot.position***”
  - The rotation, in this case this makes it have no rotation since we are making a 2D game. – “***Quaternion.identity***”
- “***Debug.Log("Result item instantiated");***”
- This creates a Debug log that reads “***Result item instantiated***”.

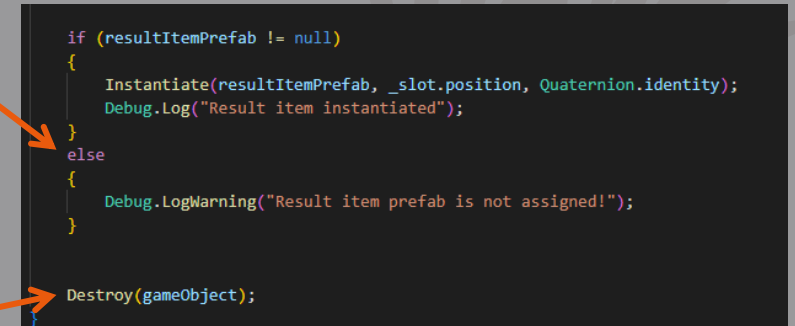


```
if (resultItemPrefab != null)
{
    Instantiate(resultItemPrefab, _slot.position, Quaternion.identity);
    Debug.Log("Result item instantiated");
}
else
{
    Debug.LogWarning("Result item prefab is not assigned!");
}
```

# Step 3: Understanding the script

## AN IF STATEMENT

- “*else*”
- “*Debug.LogWarning("Result item prefab is not assigned!");*”
- This is not for the player but for us when we are testing!
- If we forget to add the prefab this creates a debug message in the console that reads “*Result item prefab is not assigned!*”.
- Lastly outside of this If statement inside the original one we made in the last tutorial we will add the line:
- “*Destroy(gameObject);*”
- This will make any item dropped in the slot disappear!



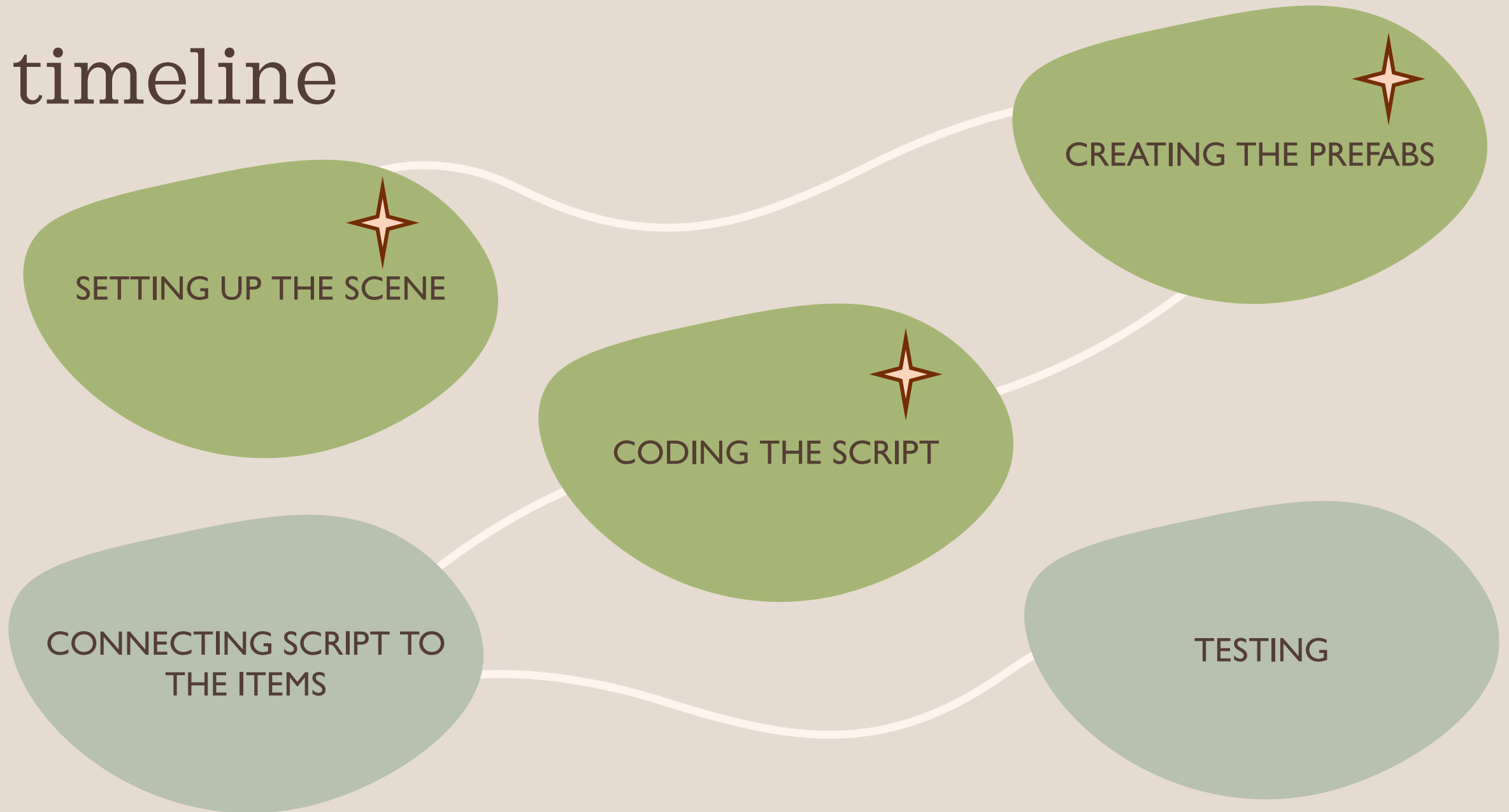
```
if (resultItemPrefab != null)
{
    Instantiate(resultItemPrefab, _slot.position, Quaternion.identity);
    Debug.Log("Result item instantiated");
}
else
{
    Debug.LogWarning("Result item prefab is not assigned!");
}

Destroy(gameObject);
```

The image shows a C# code snippet for an if-else statement. Two orange arrows point from the text in the list to the code: one points to the `Debug.LogWarning` line in the `else` block, and the other points to the `Destroy(gameObject);` line at the bottom of the snippet.



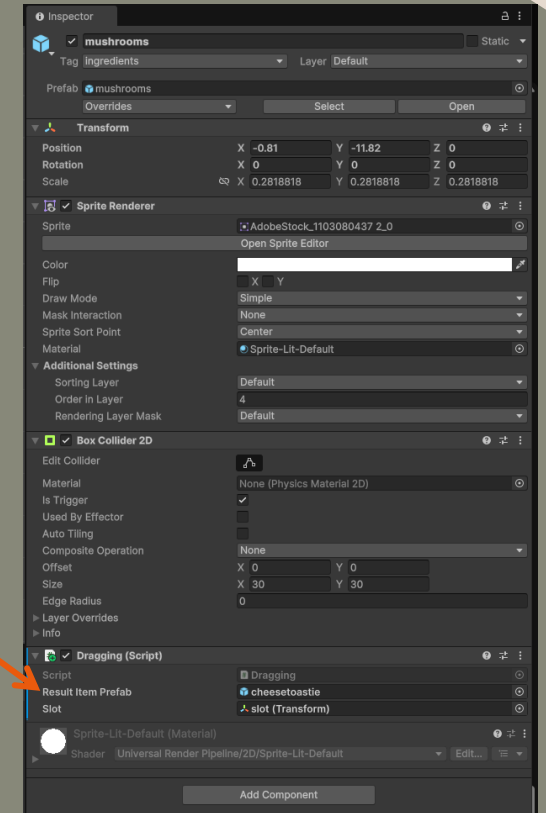
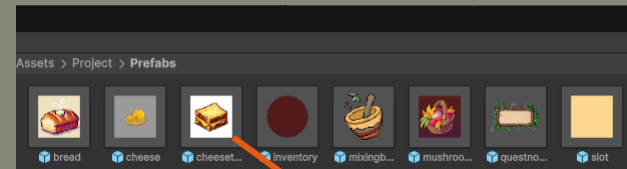
# timeline



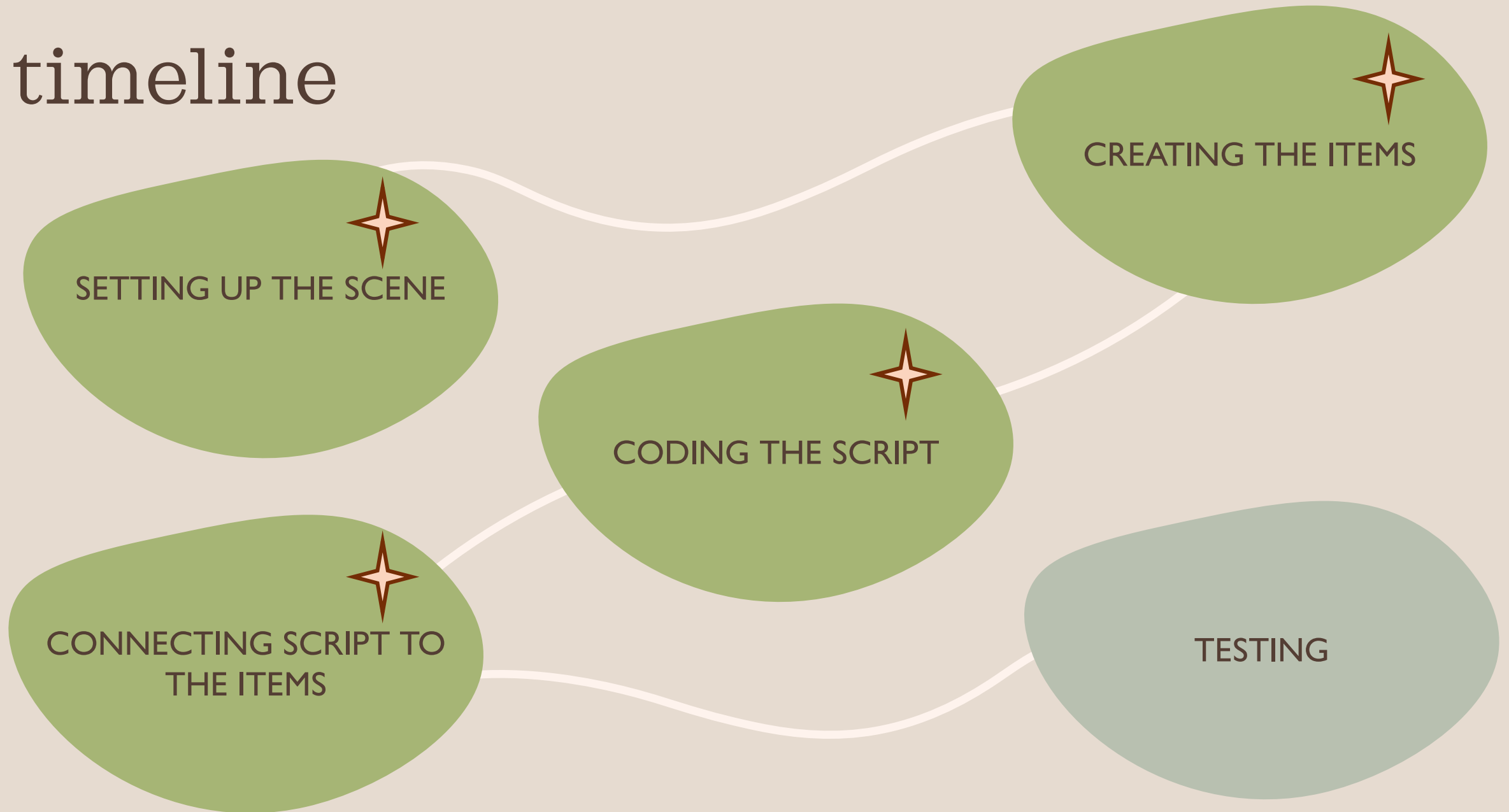
# Step 4: Connecting the script.

## SLOT

- Finally, after saving your script.
- In the items that have the script:
  - Go to the project and drag the prefab you want to spawn to the Result Item Prefab variable that we created that will be seen in the inspector.



# timeline



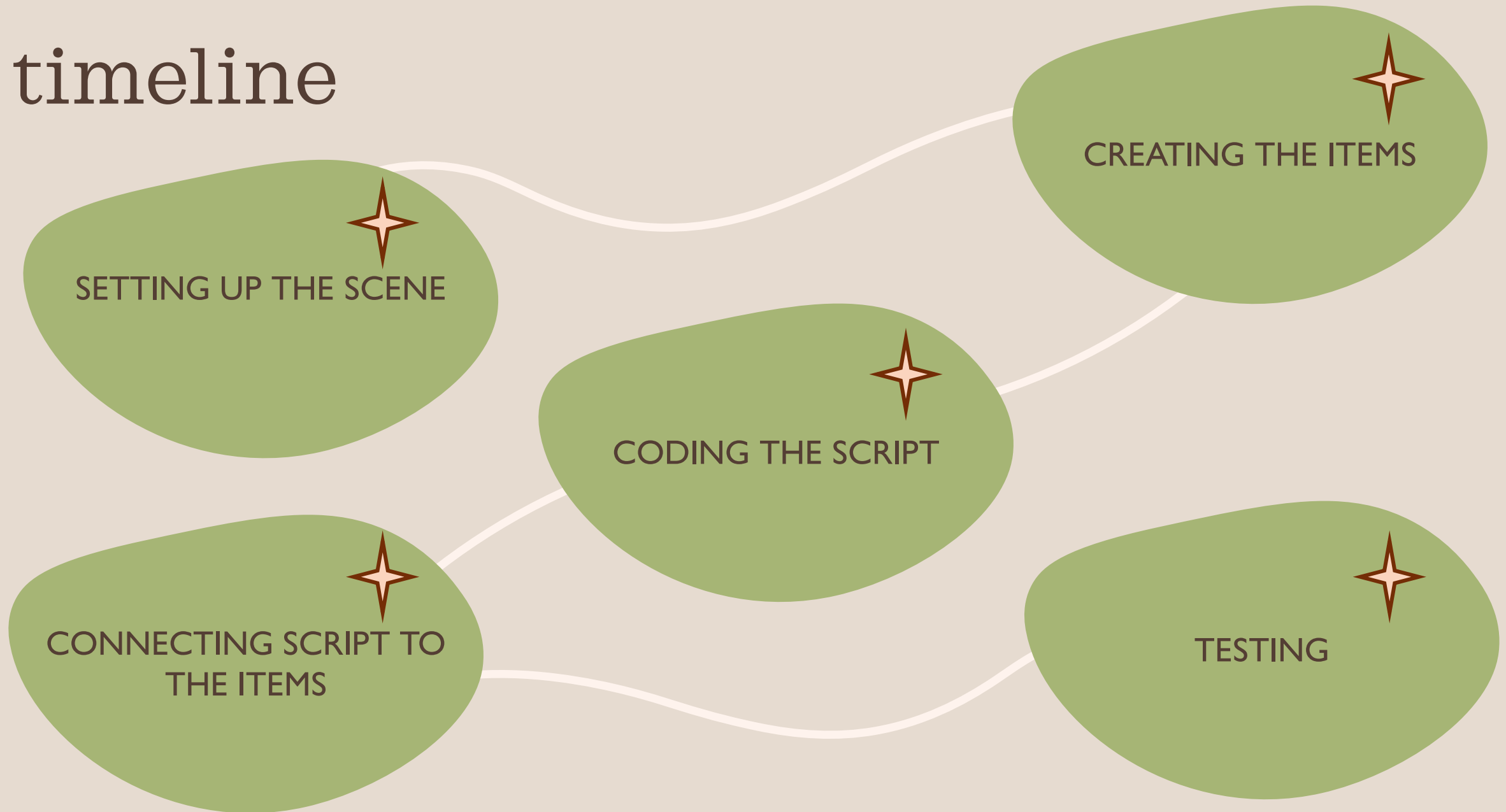
# Step 6: Testing.

## PLAYING OUR GAME:

- Now let's test our work!
- If you click on an item with the script, it will follow your mouse, and you will now be able to drop it in the slot!
- Remember that logs will appear in the console! Don't forget to check if you added the prefab!

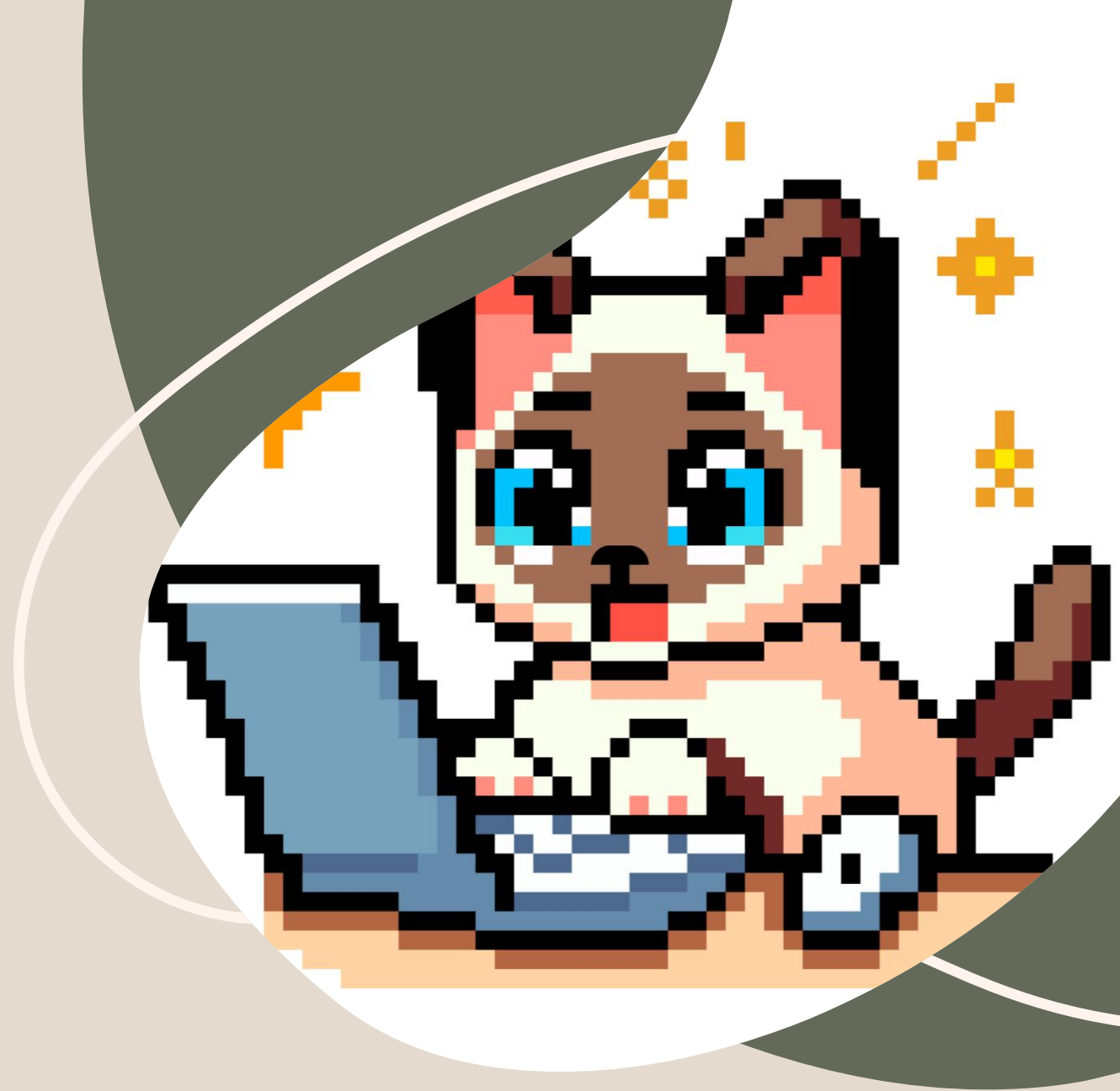


# timeline



# Congratulations!

You now can drag and drop and spawn an item in unity!



The background features a light gray base with large, soft-edged organic shapes in muted red and olive green. A thin white line outlines a shape on the right. In the top left, there is a faint, light gray sketch of a leafy branch.

Thank you