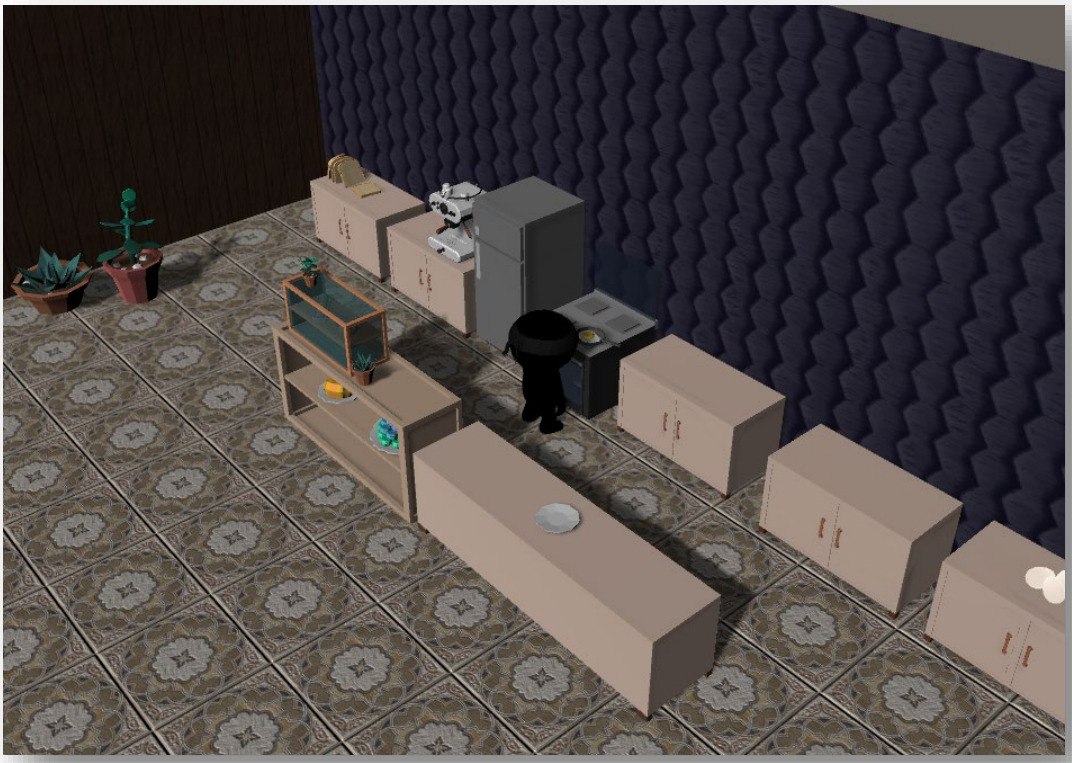
CODE NINJAS®

Platinum Belt Ninja Guide
Activity 04: Chef Codey

Chef Codey

Your goal is to plan, program, and playtest a game where the player interacts with objects in the environment! You will use objects from the Unity Asset Store to create your own themed environment for Codey.



The game we will create together finds Codey managing a café. As you plan and program, feel free to create your own theme – farming, mining, or anything else you can think of!

Plan and Design

In our café, Codey must prepare three dishes that are built out of smaller parts. For example, coffee requires bringing a mug to the coffee station, waiting for the coffee to brew, and then placing the filled mug on the counter. Codey can only hold one object at a time, so the player must plan how they prep their dishes to get the fastest time.

Overcooked 2 by Team17
Built in Unity

Untitled Goose Game by House House
Built in Unity

Your diagram must contain a starting location for Codey, six stations where Codey can interact with items, and a location to place completed dishes.

Ninja Planning Document

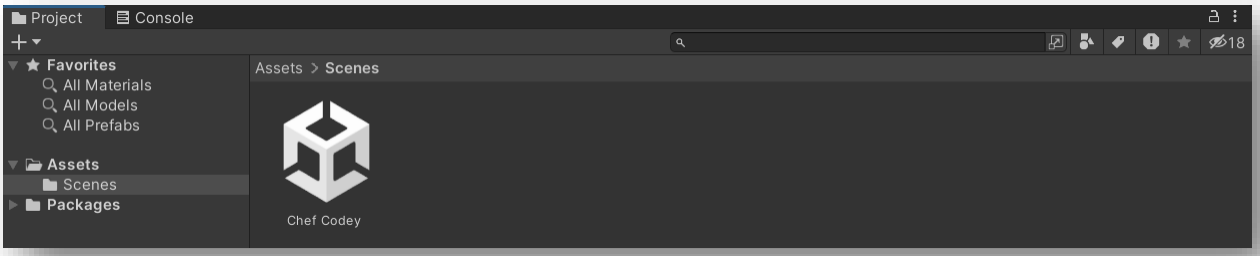
Take at least 5 but no more than 10 minutes and complete your Ninja Planning Document – Designing a Scene

Take a look at the sample projects below for some inspiration!

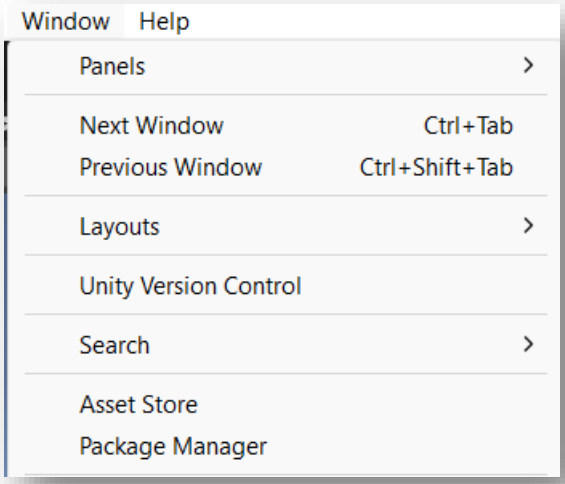
Project Setup

1 Start a new Unity Project and name it *YOUR INITIALS – Chef Codey*. Select **3D template**. Be sure to save your project to your own flash drive If you have one!

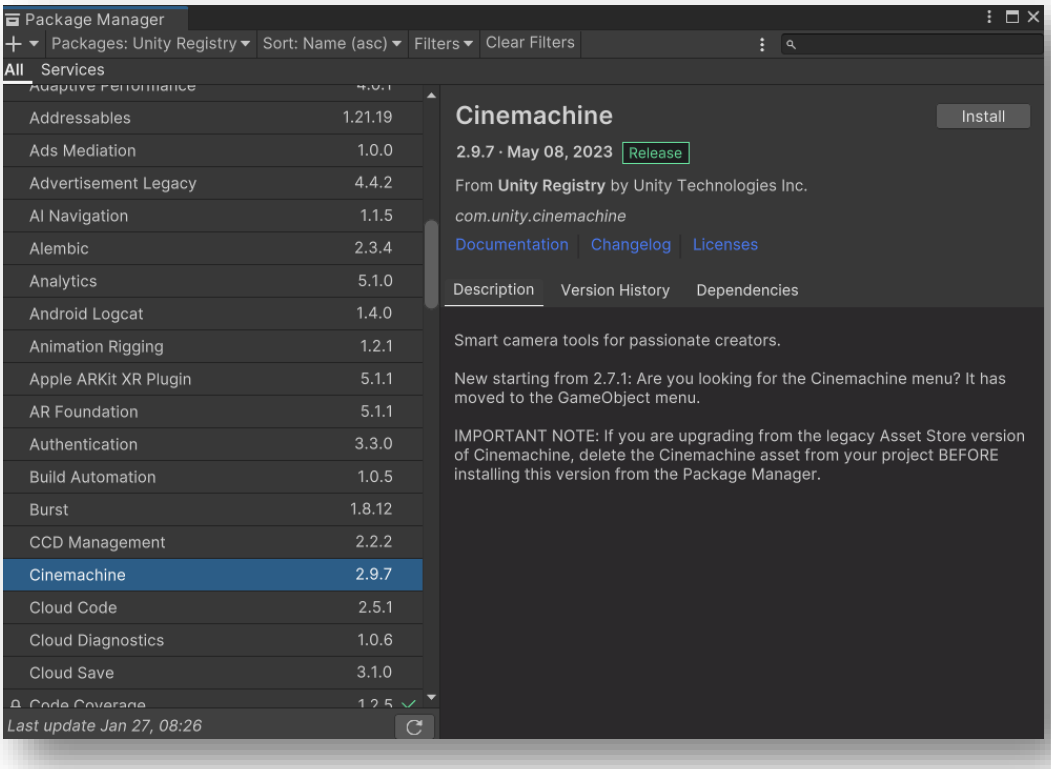
2 After it loads, rename the Sample Scene to Chef Codey.



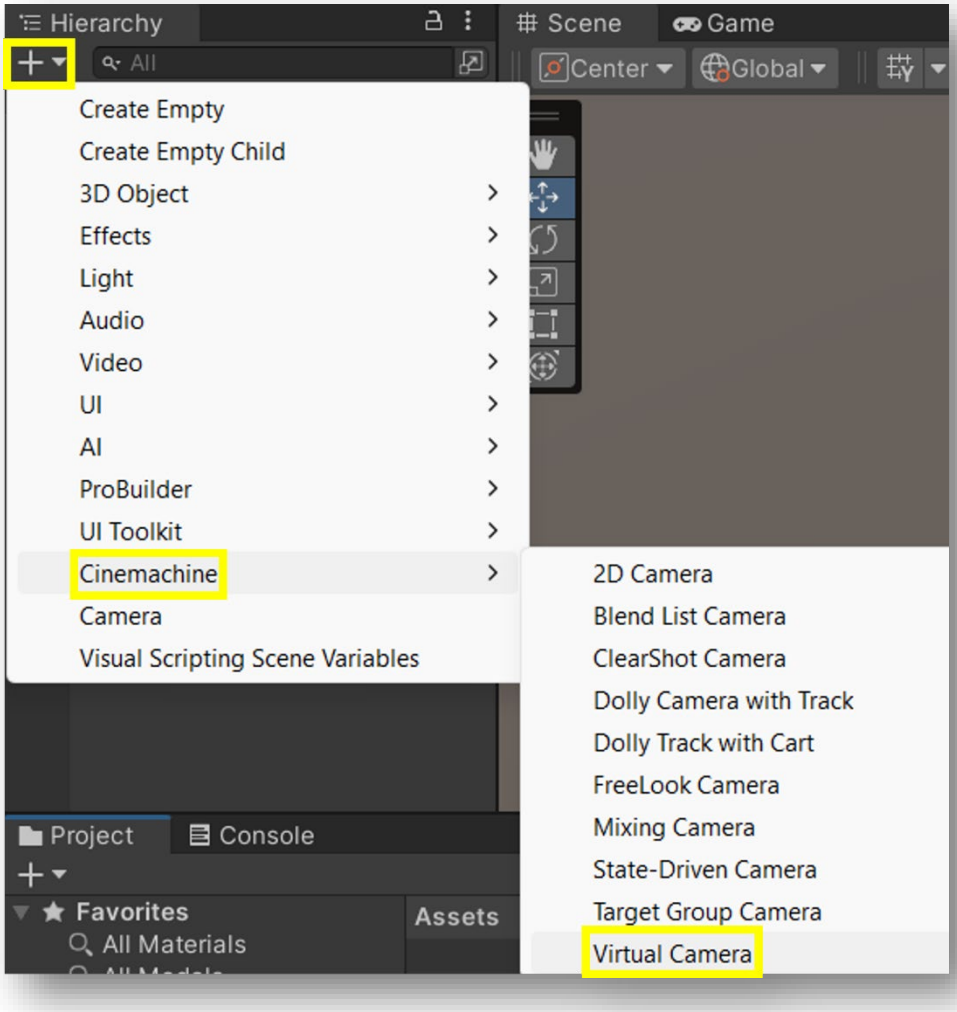
3 Open the **Unity Package Manager** from the **Window Menu**.

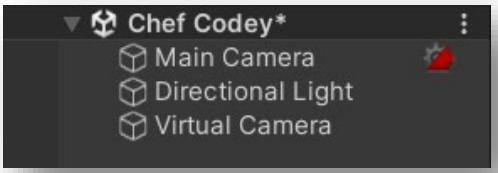


Find **Cinemachine** inside of **Packages: Unity Registry** and click **Install** in the top right of the window.



4 You should now see a **Virtual Camera** in your object **Hierarchy**. If not, go to **Cinemachine -> Create Virtual Camera**.

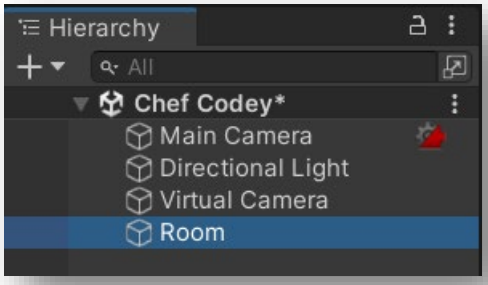




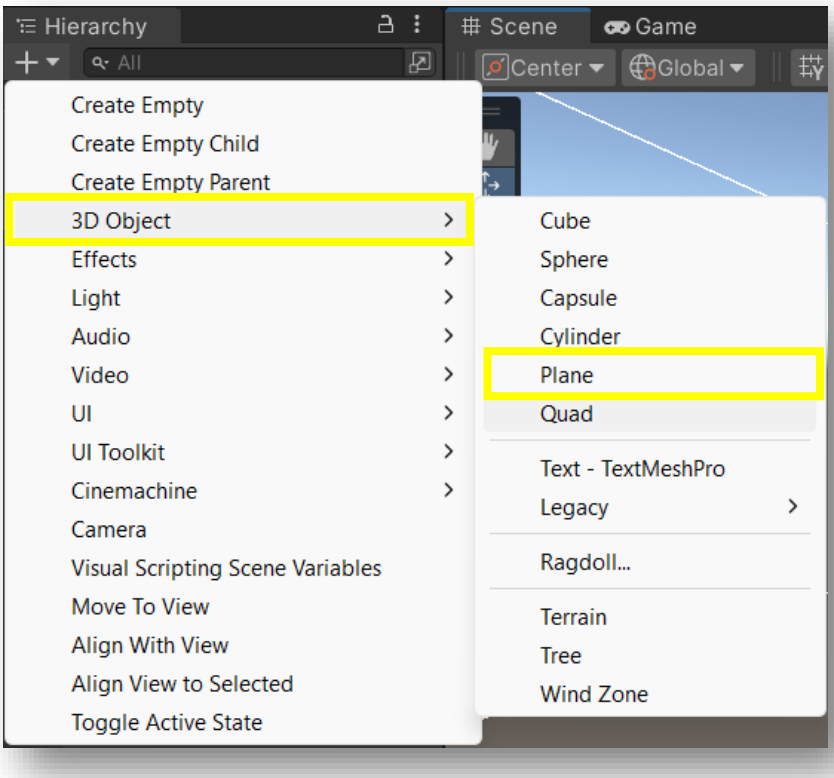
A Room with a Ninja

5 Use Code Ninjas **assets**, **assets** from the **Unity Asset Store**, or your own creations to build a basic room. If you need help getting started, you can follow the next few steps.

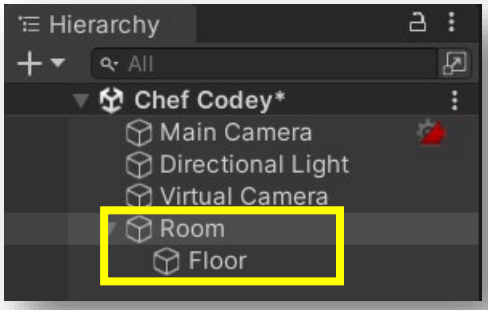
First, create an empty **Game Object** and rename it "room". We can create our room out of six **Plane objects**.



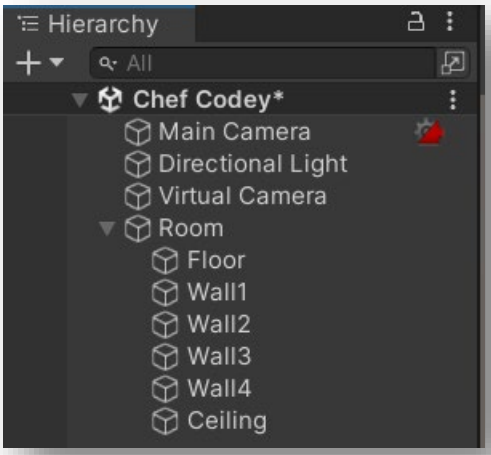
6 Create a new 3D Plane **game object**.



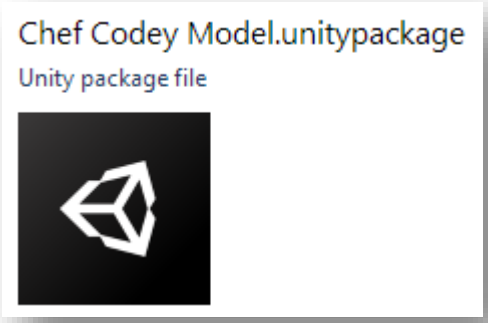
7 Rename the new **Plane object** "Floor" and place it inside of the Room **Game Object**.

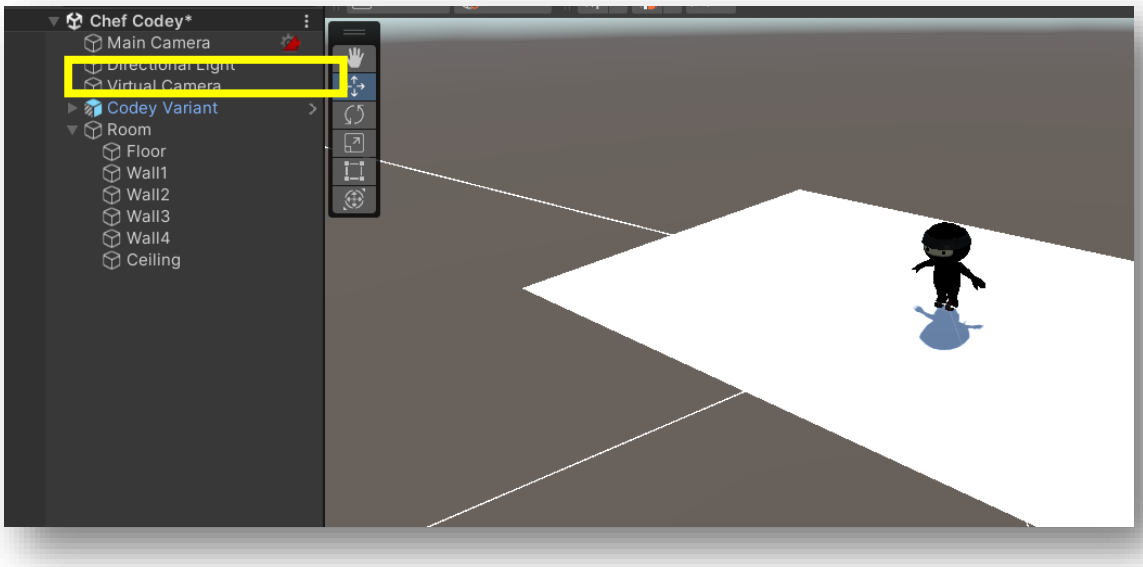


8 Repeat this for the four walls and ceiling. You can create a new plane each time or duplicate your first plane five times.



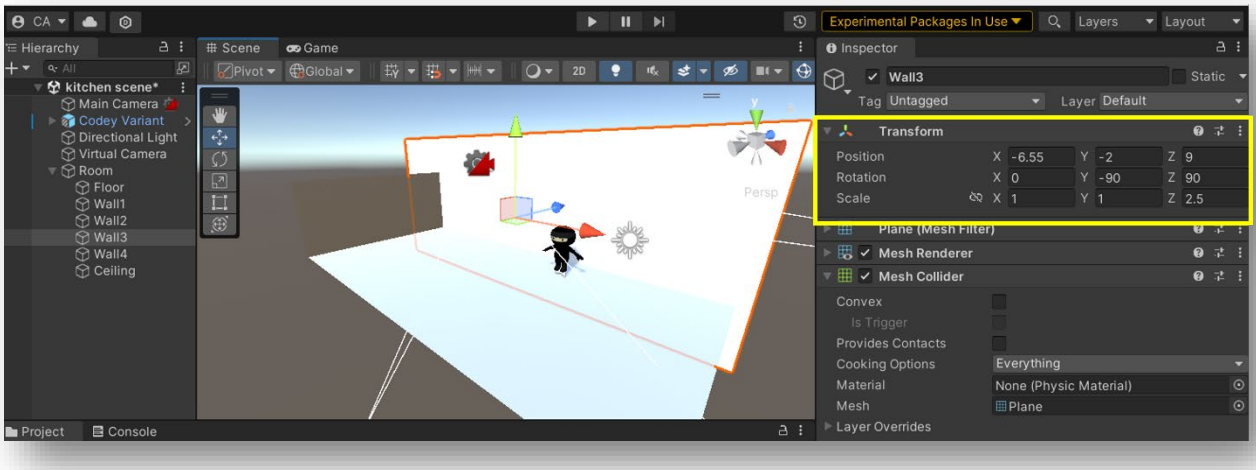
9 Import the **Activity 04 - Chef Codey Model.unitypackage**. Place the asset in the scene to make sure your room is properly **scaled** to your character.



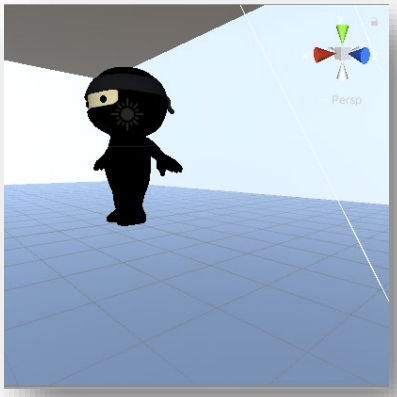


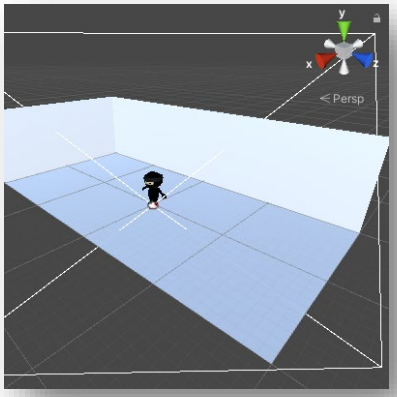
10 Each of the **planes** is positioned at the same **coordinate**. **Rotate**, **move**, and **scale** each of the planes into their proper positions. Use your **Ninja Planning Document** to help you build out your **scene**.

11 Did you notice that a **plane** is see-through on one side? Make sure you **rotate** the ceiling by 180 **degrees** in the **x** direction.



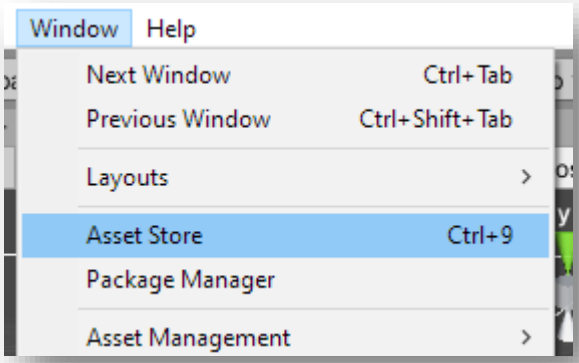
From Codey's perspective, it's an enclosed room. From the camera's perspective, it's a box with see-through walls!

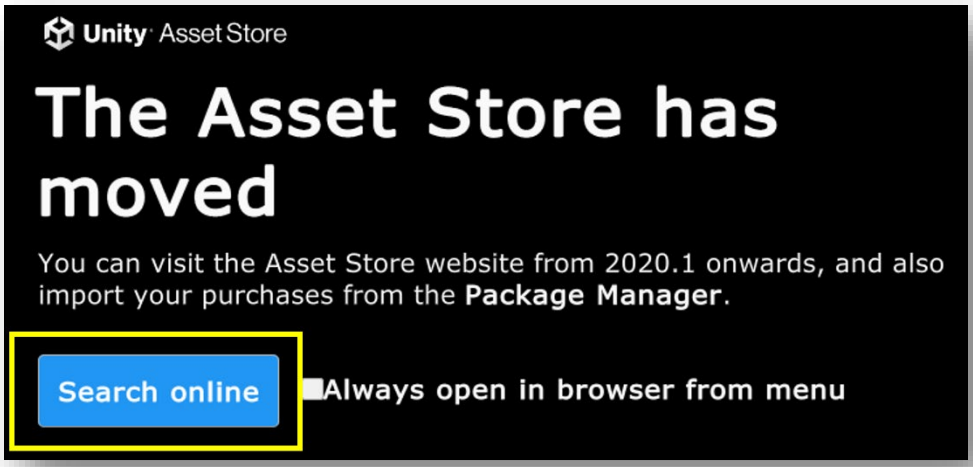




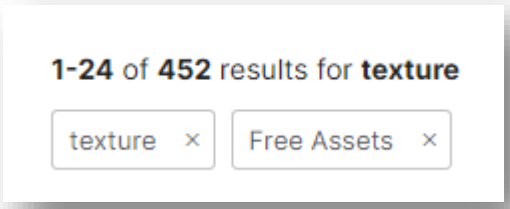
12 We can use the **Unity Asset Store** to find some **textures** to place on these **planes** to liven up the environment.

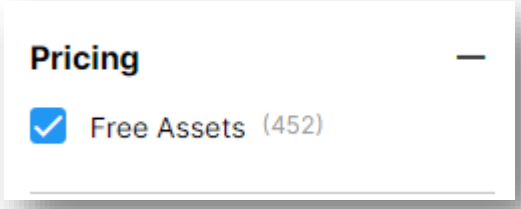
Open the **Unity Asset Store** by clicking on **Asset Store** in the Window menu. Then click the **Search Online** button.





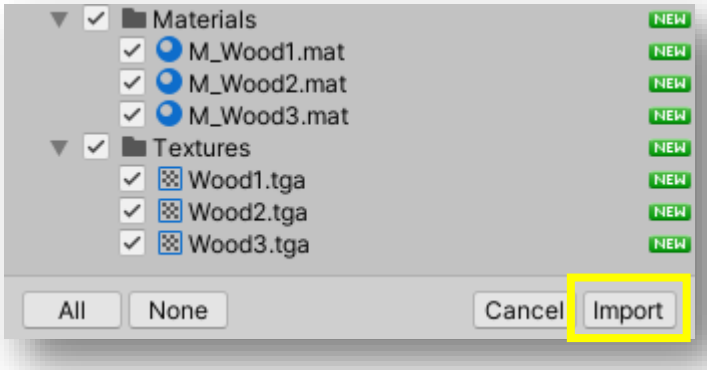
13 Search for "texture" and make sure you select "Free Assets" under **Pricing**.



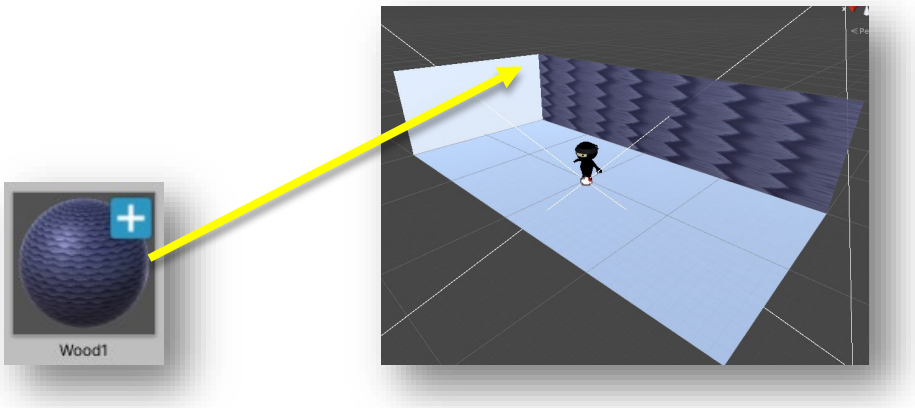


14 After you find an **asset** that you like, click **download**. Once it downloads, it will be in your **Unity** account. Click **Import** to put the **asset** from the **store** to your current **project**.

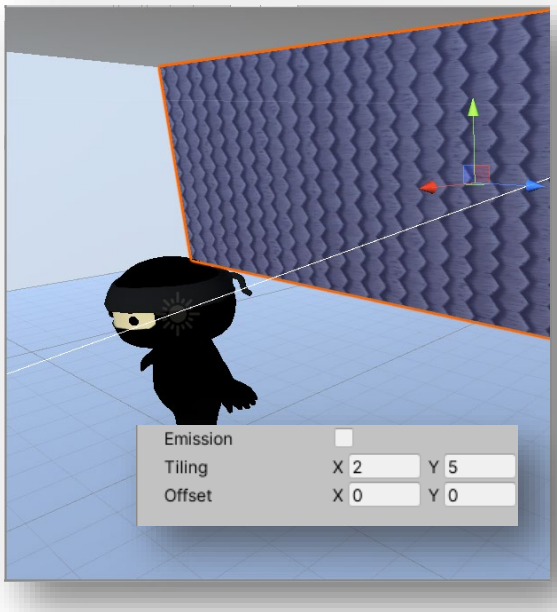
15 In the **Import Unity Package** window, click **Import**.

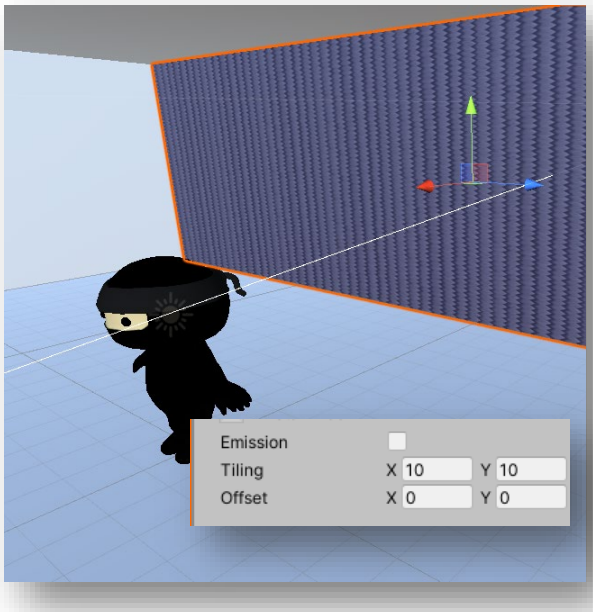


16 Look in your **Assets** folder in the **Project** tab for where the new **assets** are located. Drag your **texture asset** from the **Project** tab onto one of the **plane objects** in the **scene**.

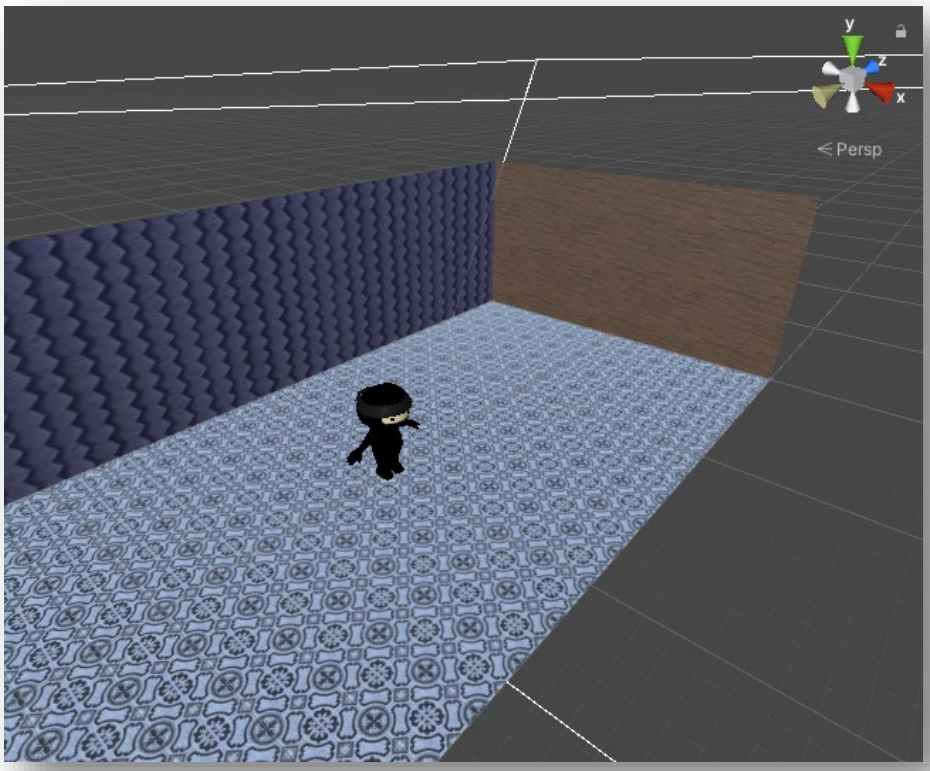


17 Select the **plane** and look at its **texture component**. Adjust the **Tiling X** and **Y parameters** to adjust how the **texture** is applied to the **object**.

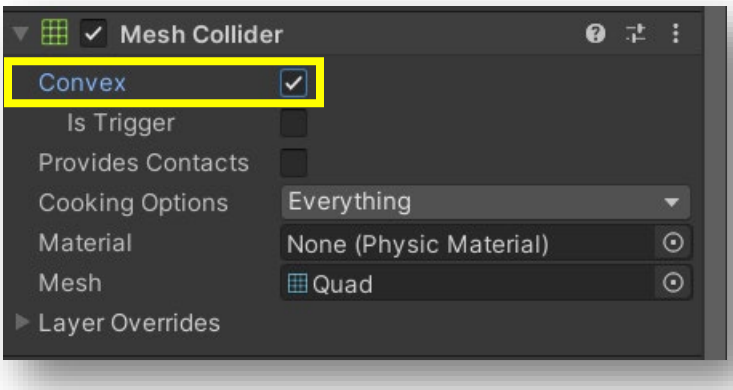




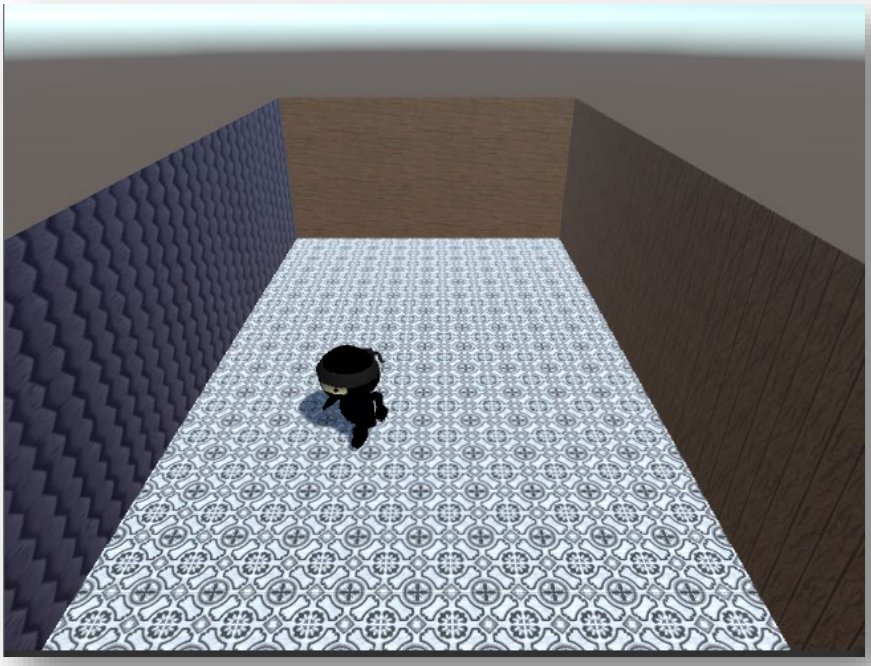
18 Repeat this process to design the other **planes** that make up your room.

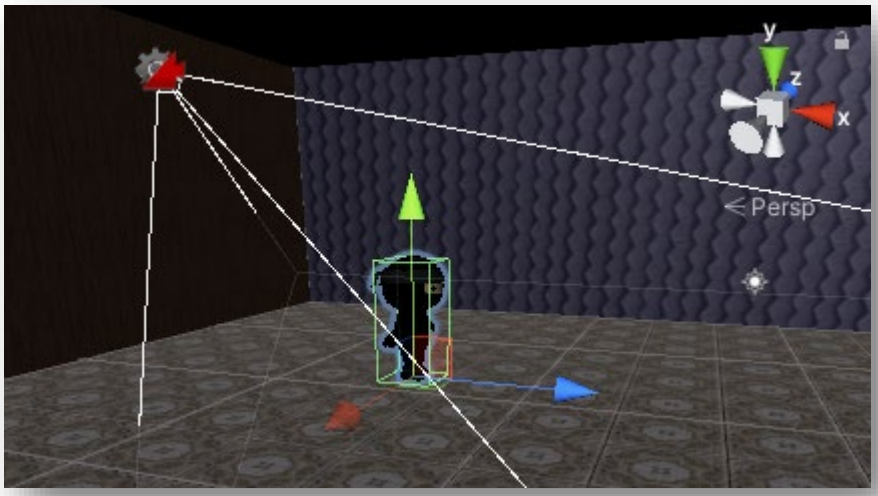


19 With a basic room complete, **playtest** your game. You might need to adjust the **scene** based on how you initially set it up. If Codey can run through your walls, make sure that your planes have **colliders** with the **Convex** property enabled.

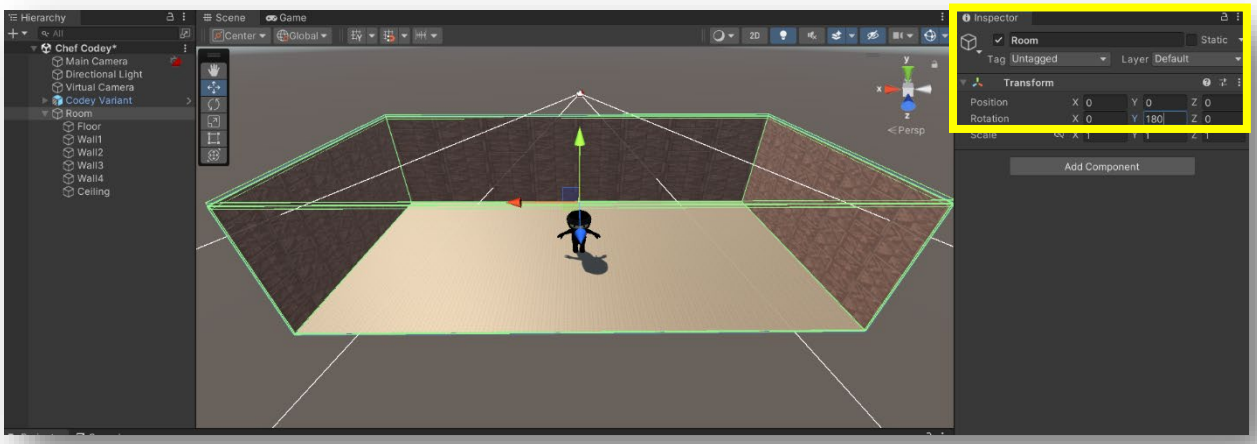


20 If Codey does not move up when you press up (or w), then you need to **rotate** and move the **camera**. To match Codey's movement **script**, the camera must be pointing in the **positive Z** direction. Since we are using **Cinemachine**, change the **transform properties** on the **Virtual Camera Game Object**. Make sure that the **camera** is pointing to the back wall of your room.





21 Make sure to adjust the **rotation** of your room **game object** to align it if necessary. Instead of adjusting each **plane** individually, you can change the **rotation** of the room **object**.



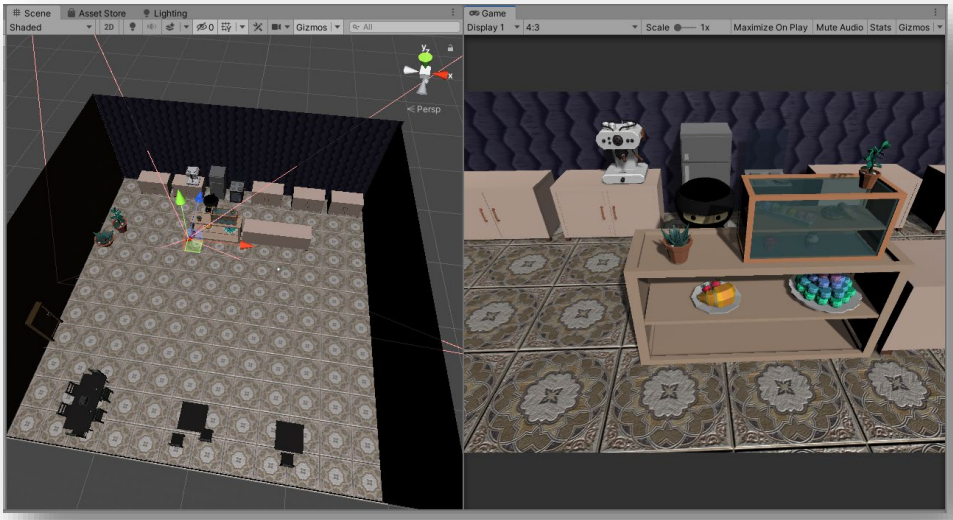
Sensei Stop

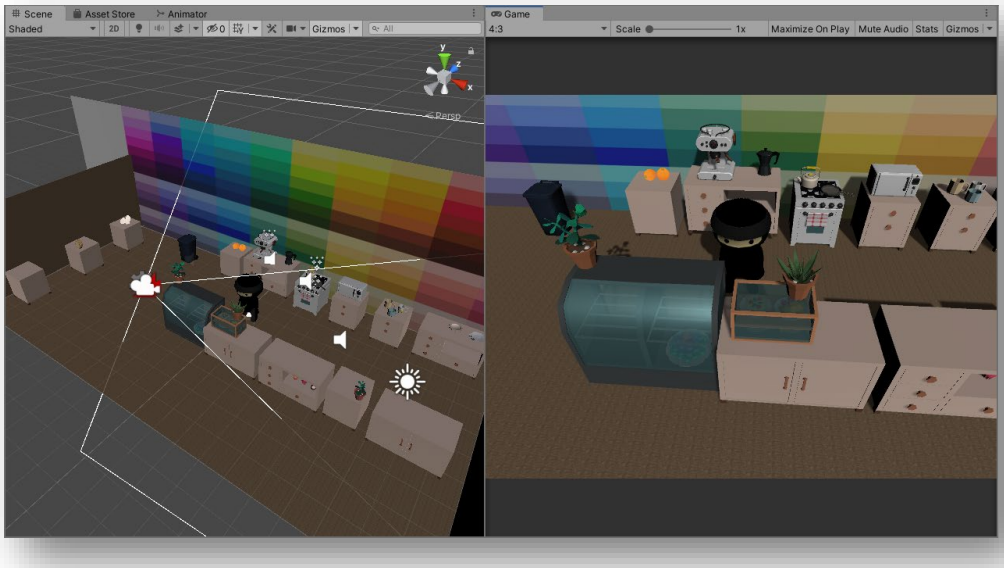
Discuss your room design with your sensei. Does this align with your planning document? Play test to make sure that Codey cannot run through walls. What type of textures did you use? How does this fit your theme?

Extreme Café Makeover

22 Using your Ninja Planning Document, use the **Unity Asset Store** to find **assets** to place in your room. Use keywords like "coffee", "park", "shop", and "classroom" to help you find cool **assets** to use!

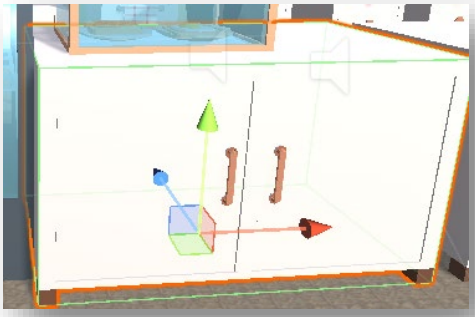
As you place your items, be sure to **playtest** frequently. It's a good idea to place your **objects** in an empty **game object**. This helps keep your **hierarchy** clean.

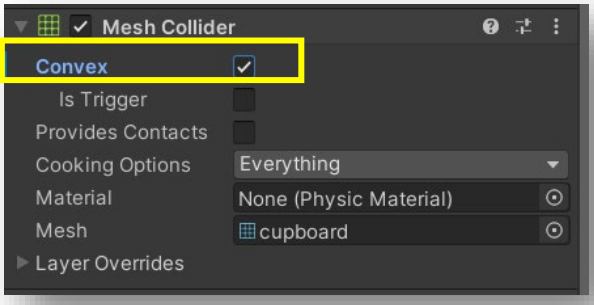




23 See if Codey can clip into the **objects** in your **scene**.

Each **asset** you download will have different **properties** and **components**. **Objects** in your **scene** might need to be given **box**, **capsule**, **sphere**, or **mesh colliders**. If you use a **mesh collider**, make sure the **Convex** checkbox is enabled.

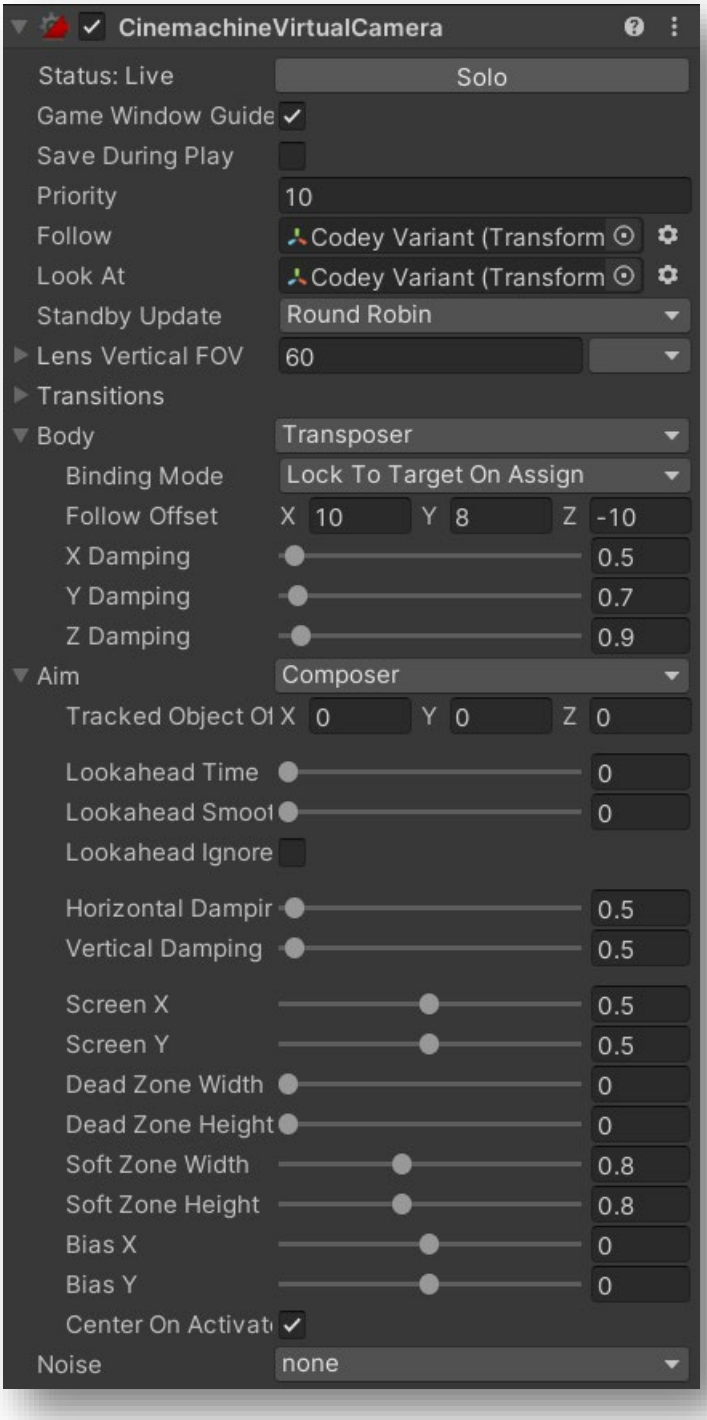




Mesh Colliders

A Mesh Collider is special because it matches the shape of the object. Because the mesh colliders are more complex and require many computations, Unity prevents interaction between mesh and colliders by default. To have mesh colliders interact properly with the other colliders (like the one on Codey), you must enable its Convex property.

24 You can program the camera to track Codey around the scene by changing a few properties in **Virtual Camera Game Object**.



Don't be afraid to change the properties until you like the camera.

The Toast with the Most

25 With our **environment** set up, we can start on the game **logic**.

The player can pick up **items**, transform the **items**, and build new **items**. The player's goal is to cook all the **items** available as quickly as possible.

We can tackle the **logic** piece by piece. First, we can start with food "sources" that give an **item** to Codey to hold.

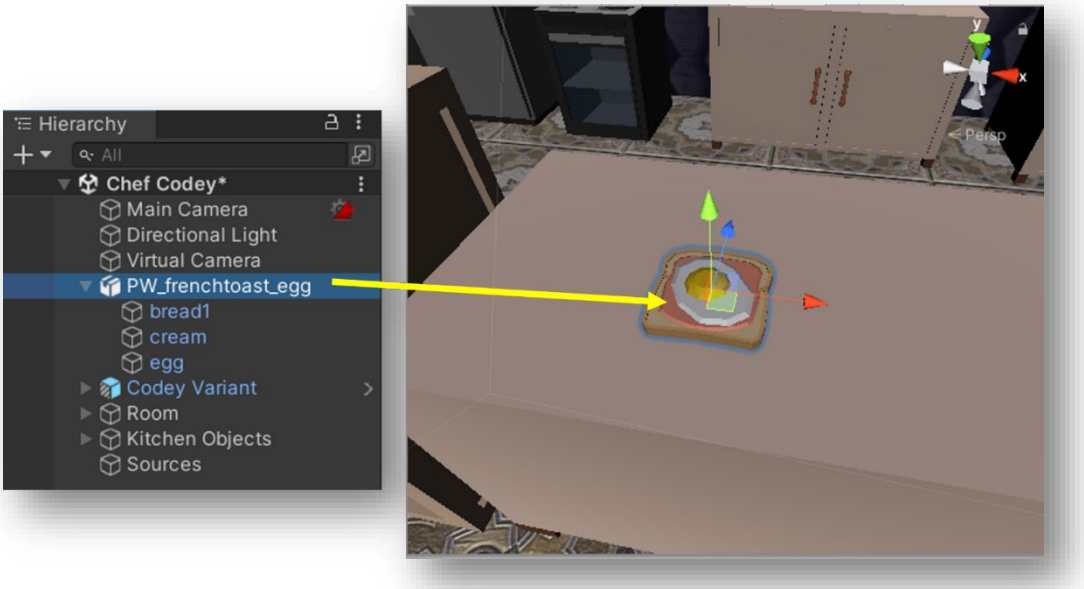
Start by creating an empty **object** to hold all our sources.

26 Revisit the **Unity Asset Store** to find some objects for Codey to make. For example, you can use a French toast asset that is built out of three smaller parts.

Asset Parts

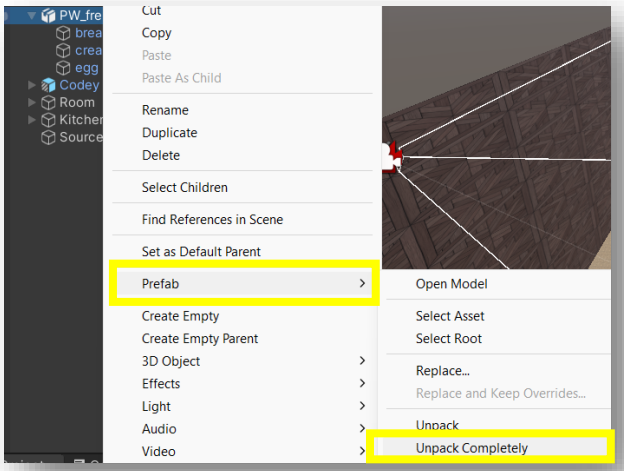
Every asset will be different. You might find one that is built out of many different smaller parts, or you might have to take several assets and combine them together to make your own!

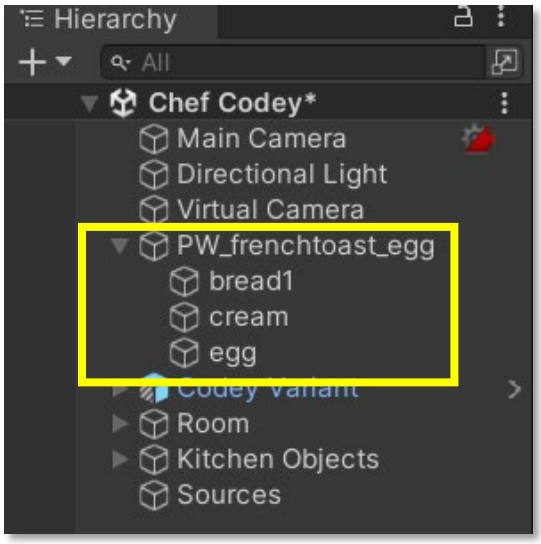
27 Once you have your **model**, you might have to separate the pieces to use in your **scene**. If your **object** is not made of different parts, then the next steps might not be necessary.



28

Right click the **Blue Prefab** in the **hierarchy** and select “Unpack Prefab Completely”. Remember to resize your **object** so it looks like it belongs in your **scene**.



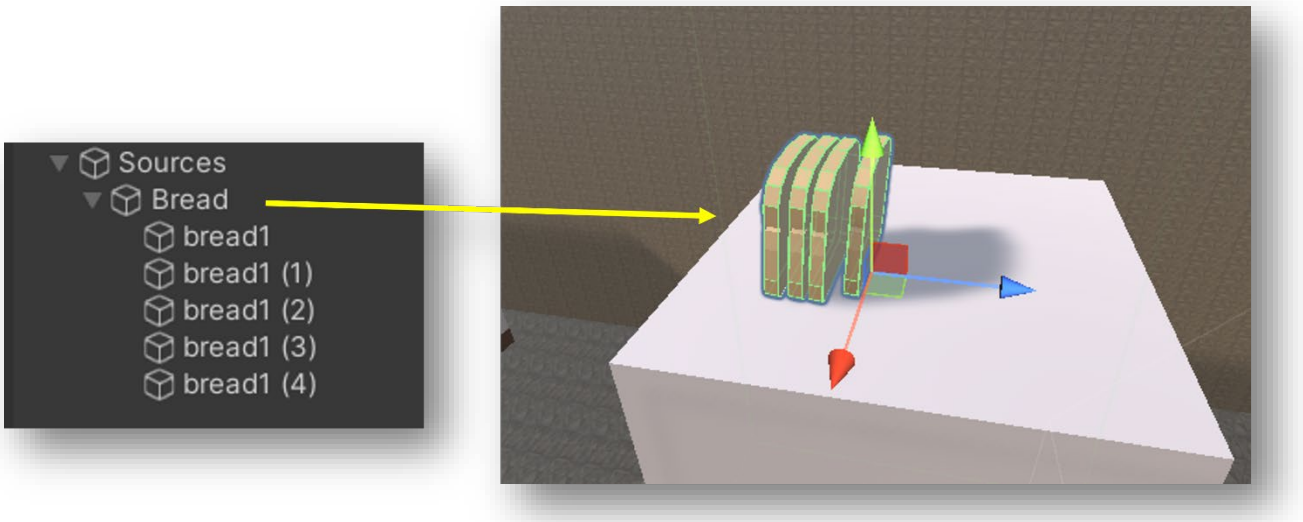


29

Now the **object** names are black meaning we can edit them and save them as our own **prefabs**!

Let’s start with the bread. Create a new empty **object** inside of the Sources **object** and call it Bread.

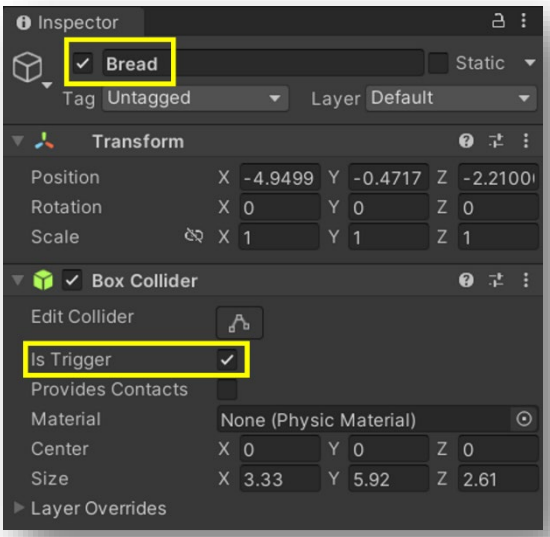
30 Place the bread **object** into our new empty **object** and duplicate it several times. We want to place these slices on a counter to tell the player where to find bread.

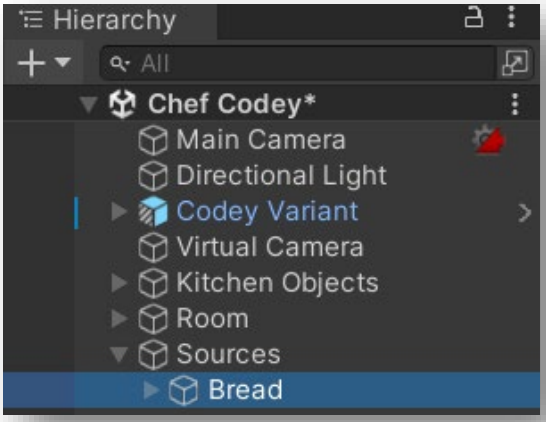


31 We need a way to tell if Codey is close enough to pick up a piece of bread. We don't want to give Codey the ability to teleport **objects** from around the **scene**!

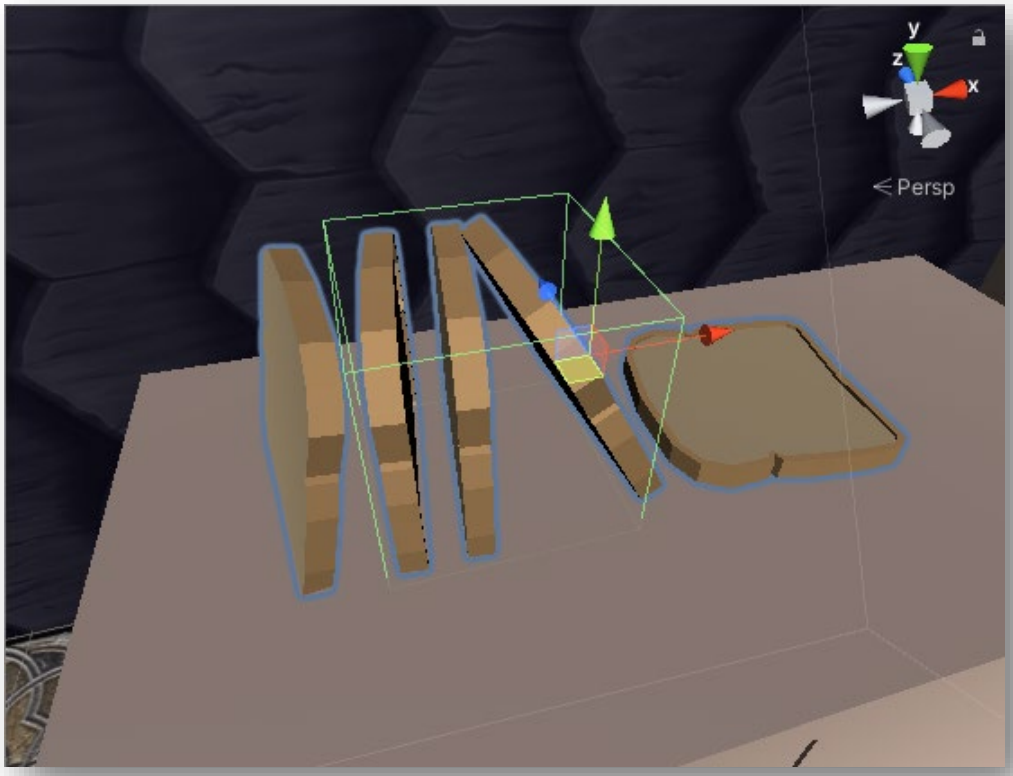
Add a **box collider** to the Bread game **object**. Make sure that you don't accidentally add a box collider to any of the individual game **objects** – we want our **collider** on the **parent** container **object**.

Enable **Is Trigger** on your **Collider**.



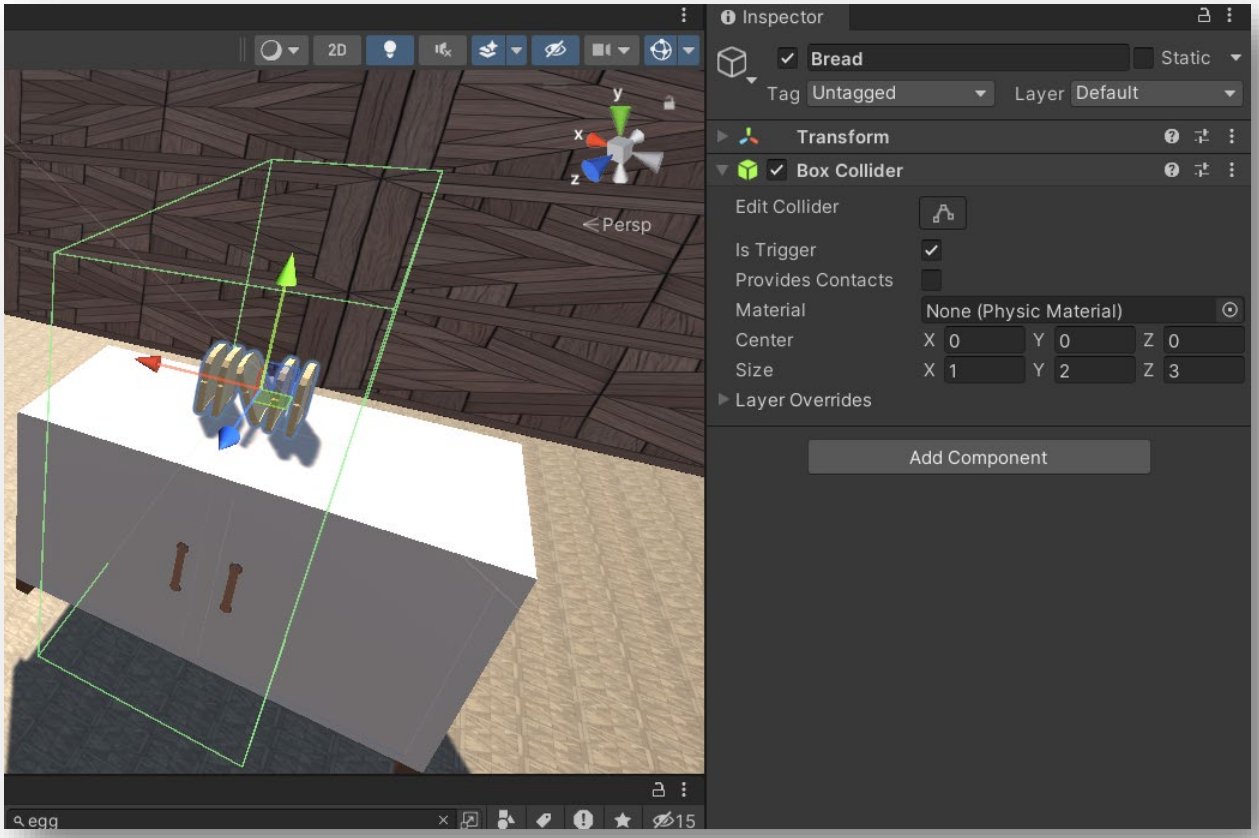


32 If you notice that your **objects** are not aligned with your **box collider**, don't be afraid to reset each of the **model**'s **coordinates** to zero and reposition them. Part of game making includes constant adjustments!



33

Use the **collider**'s **Center** and **size** properties to make a box that extends in front of the **models**.

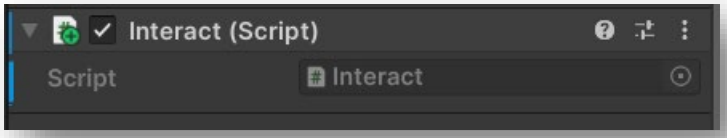


34

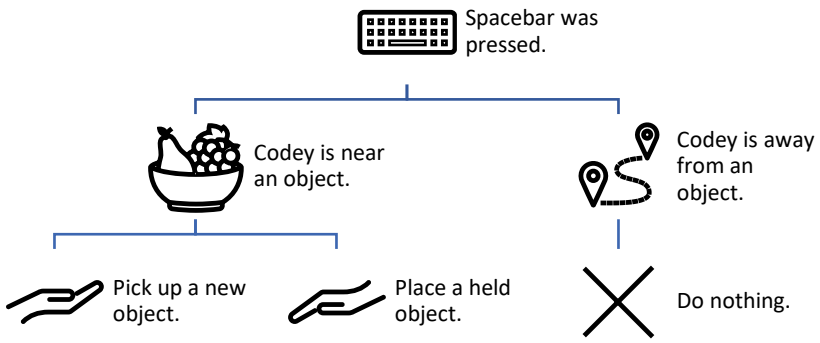
We are going to use this **collider** to know when Codey is close enough to the bread to pick it up. We need to create a **script** and **program** Codey to pick up an item.

Create a new **script** in your Script folder in the **Assets** tab and name it Interact.

Attach the **script** to the Codey **game object** and open it in Visual Studio.



35 Whenever the player presses the **spacebar**, we need to ask two things –first, is Codey standing near an object? Second, is Codey picking up or setting down an object? We can sketch out our logic in a chart.



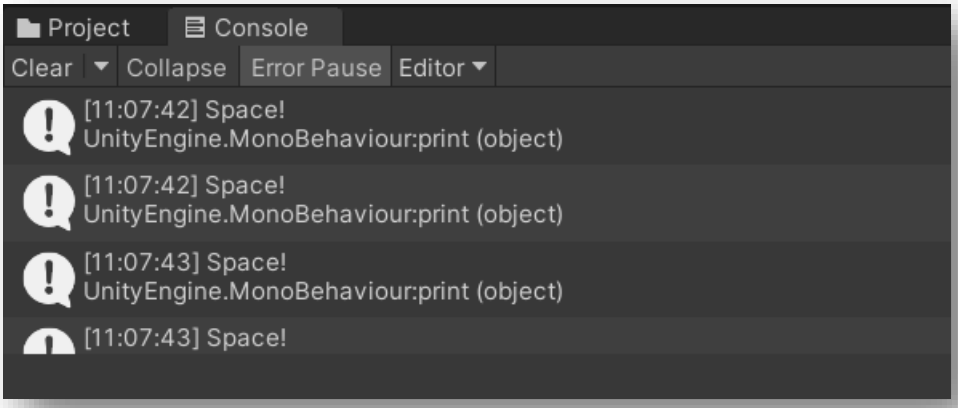
We can think of this logic as several branching **if** and **if else statements**. Since we are responding to when the **spacebar** is pressed, we will write our logic in the Update **function**.

36 Knowing when the spacebar is pressed is very easy in Unity. Try to think about previous games you created and how you listened for player input.

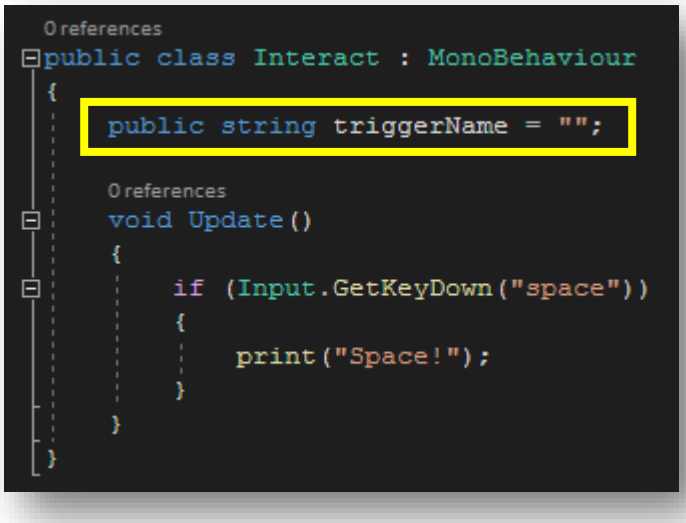
In the Update **function**, create an **if statement** that checks to see **if** the space **key** was pressed by using the **Input object**'s **GetKeyDown function**. Inside the **if statement**, **print** a message to the **console**. You can remove the Start **function**.

```
0 references
public class Interact : MonoBehaviour
{
    0 references
    void Update()
    {
        if (Input.GetKeyDown("space"))
        {
            print("Space!");
        }
    }
}
```

37 Save your **script** and **playtest** your game. What happens when you press the spacebar?



38 Next, we need to know if Codey is near an **object**. This requires tracking when Codey enters and exits the food object's **collider trigger**.
Back inside the Interact **script**, create a **variable** to store the name of the trigger that Codey collides with.



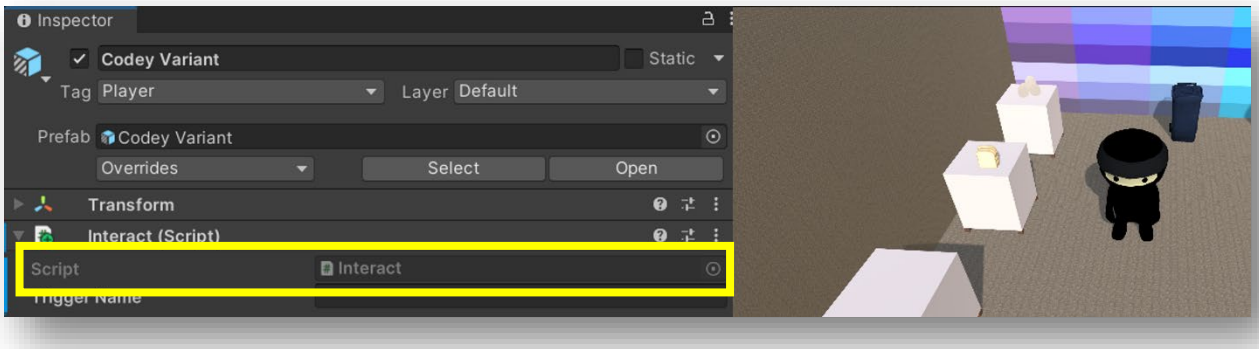
39 Add the two Unity **functions** that run when a **game object** enters or exits a **trigger**.

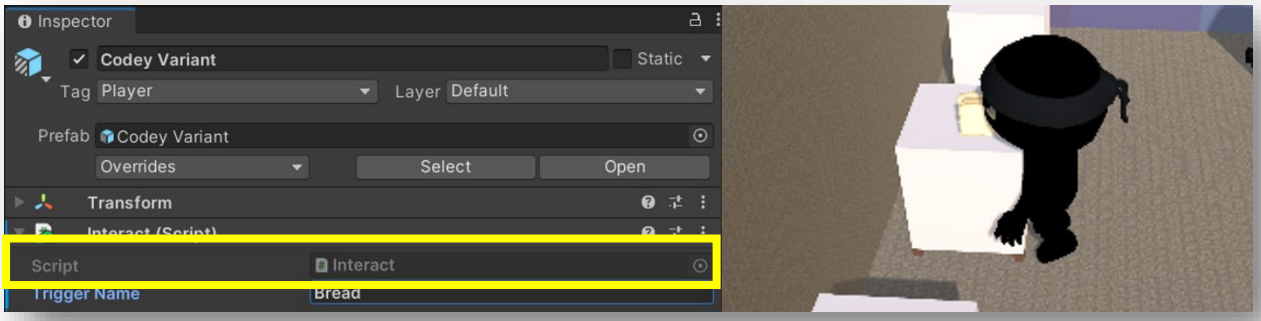
```
0 references
public class Interact : MonoBehaviour
{
    public string triggerName = "";

    0 references
    void Update()
    {
        if (Input.GetKeyDown("space"))
        {
            print("Space!");
        }
    }

    0 references
    private void OnTriggerEnter(Collider other)
    {

    }

    0 references
    private void OnTriggerExit(Collider other)
    {

    }
}
```

40 When Codey enters a **trigger**, we want to set the **triggerName variable** to the **name** of the **collider**. When Codey leaves a **trigger**, we want to reset the **triggerName variable** to an empty **string**.

```
0 references
private void OnTriggerEnter(Collider other)
{
    triggerName = other.name;
}

0 references
private void OnTriggerExit(Collider other)
{
    triggerName = "";
}
```

41 **Playtest** your game. What happens to the **triggerName variable** in Codey's **inspector** when you get close to the bread **objects**? What happens when you walk away from the bread **objects**?



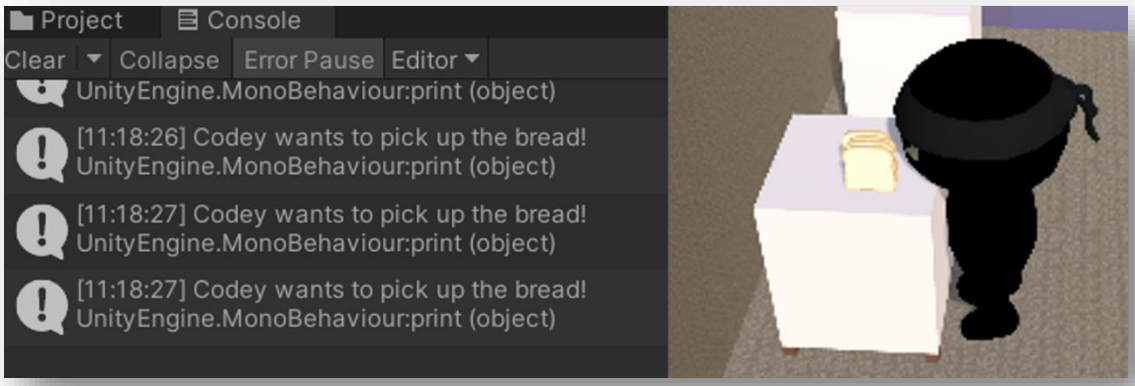


42 We can now combine our **input code** with our new **trigger code**. Inside our **if statement** inside the update **function**, write a new **if statement** that checks to see **if** the **triggerName variable** is equal to "Bread". **If** it is, then print a message in the **console**.

```
0 references
void Update()
{
    if (Input.GetKeyDown("space"))
    {
        if (triggerName == "Bread")
        {
            print("Codey wants to pick up the bread!");
        }
    }
}
```

If you used an **object** other than bread, make sure that you use the correct **name** of your **object**!

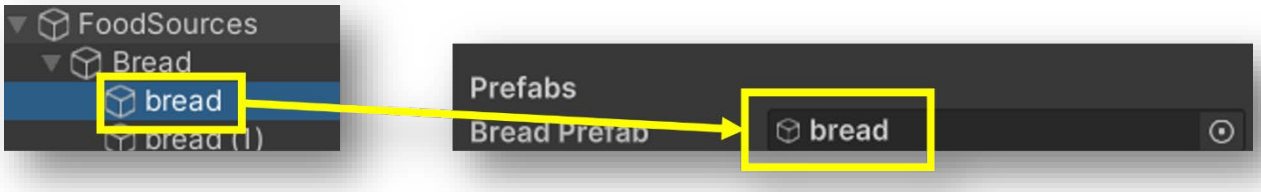
43 **Playtest** your game. Now pressing the spacebar should only send a message if Codey is standing inside the bread **object**'s **trigger**.



44 First, Codey needs to know what **object** to pick up. In the Interact **script**, create a **public game object** and name it **breadPrefab**.

```
public class Interact : MonoBehaviour
{
    public string triggerName = "";

    public GameObject breadPrefab;

    0 references
    void Update()
    {
```

45 Save the **script** and go back to the Unity Engine. Drag a bread **object** from the **hierarchy** and drop it into the Bread Prefab **variable** in Codey's **inspector**.



46 Codey also needs a way to hold an **item**. In the Interact **script**. Create a **public game object** named **heldItem**.

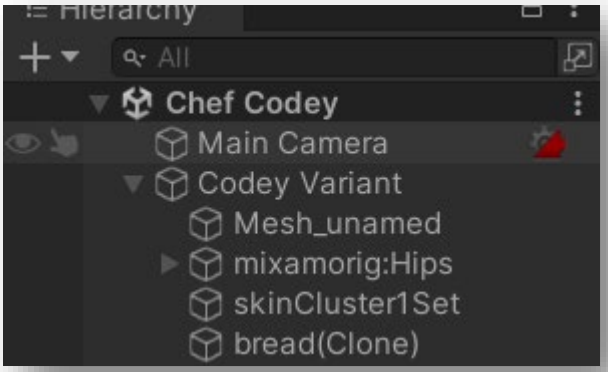
```
public class Interact : MonoBehaviour
{
    public string triggerName = "";

    public GameObject breadPrefab;

    public GameObject heldItem;

    0 references
    void Update()
    {
```

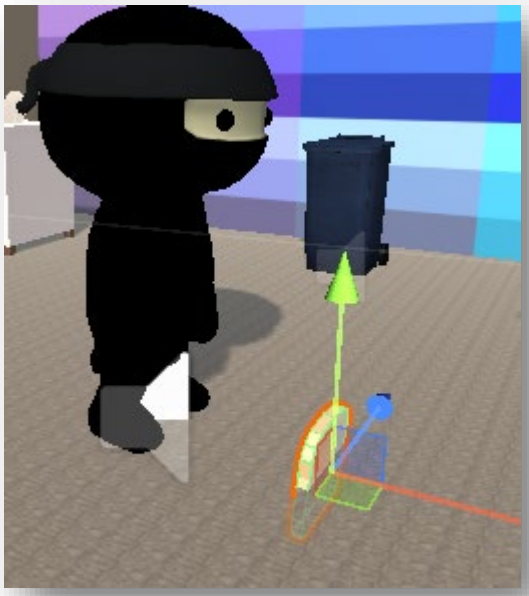
47 We need to set **heldItem** to a **new game object** that we create with **code**. Inside the **if** the **trigger** name is Bread **statement**, we will use a special version of the Instantiate **function**.

The first **argument** is the **model** or **prefab** of the **object** we want to create. The second **argument** is what **game object** we want as our new **object**'s **parent**. The final **argument** is **false** to tell Unity how to position the new object in relation to the world.

```
0 references
void Update()
{
    if (Input.GetKeyDown("space"))
    {
        if (triggerName == "Bread")
        {
            heldItem = Instantiate(breadPrefab, transform, false);
        }
    }
}
```

48 Playtest your game and try to pick up a slice of bread.





You can check to see whether Codey has picked up the bread by clicking on Codey in the **hierarchy**.

49 Codey now has a piece of bread, but it's not in the best **position**. Stop the game and open your Interact **script** again. After the Instantiate line, we can set the heldItem's **position**.

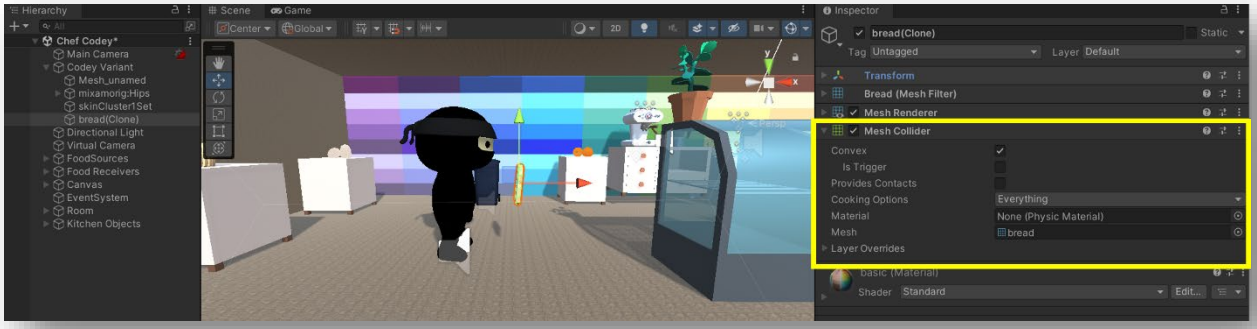
```
0 references
void Update()
{
    if (Input.GetKeyDown("space"))
    {
        if (triggerName == "Bread")
        {
            heldItem = Instantiate(breadPrefab, transform, false);
            heldItem.transform.localPosition = new Vector3(0, 2, 2);
        }
    }
}
```

We need to use the **transform**'s **local position** to place the piece of bread based on Codey's **position**, not the world's **position**.

Using the **localPosition** property means that the **object** will be placed zero **x** units, 2 **y** units, and 2 **z** units away from Codey's **position**.

If we used the **position** instead, the **object** would be placed 0 **x** units, 2 **y** units, and 2 **z** units away from the world's center.

50 Save and close the Interact **script**, and playtest your game again. You should now be able to see the piece of bread Codey has picked up.



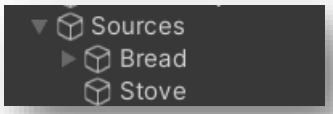
Sensei Stop

While the slice of bread might look good at (0, 2, 2), it might not be quite right for other objects. Change the values of your Vector3 in the code to position your object. Tell your Sensei how you tested your different values.

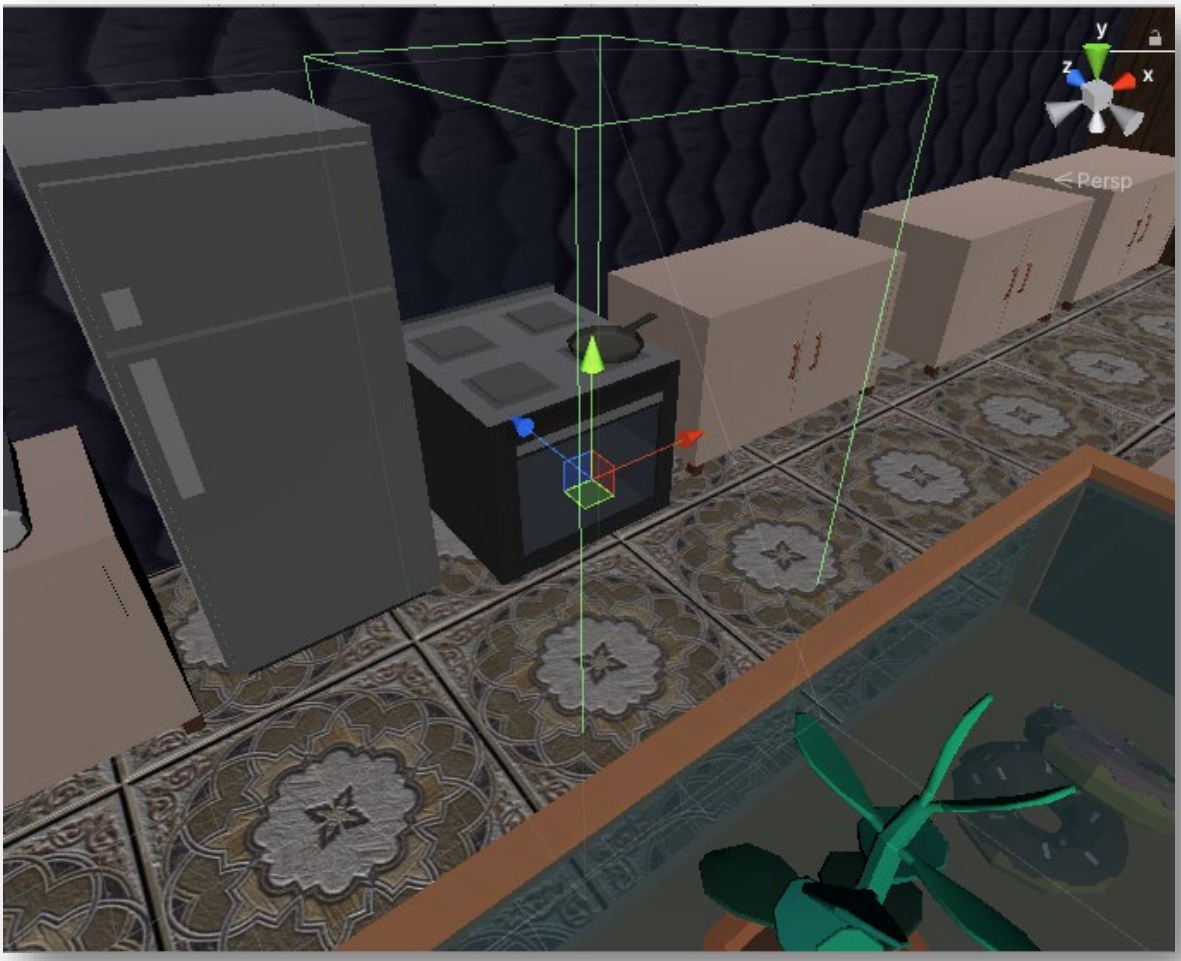
Now We're Cooking

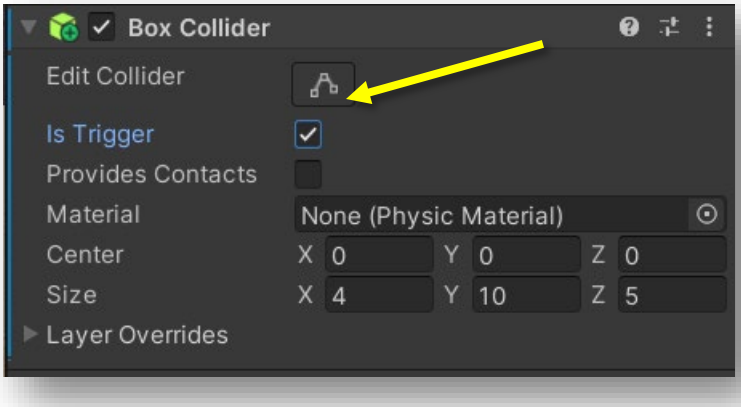
51 In many resource gathering games, you need to process **items** before you can use them. You might have to chop wood, smelt stone, or even toast bread. In this game, we will need to process our bread **object**.

Next, we can set up a stove to cook our food. Find a stove or another **object** in the Unity Asset Store and place it in your **scene**. In your Source **game object**, create a new empty **object** named Stove or the name of your **object**.



52 Add a **box collider** to this new stove **object**. Move and resize it so it overlaps with the **model**. Make sure you check the **Is Trigger** box.



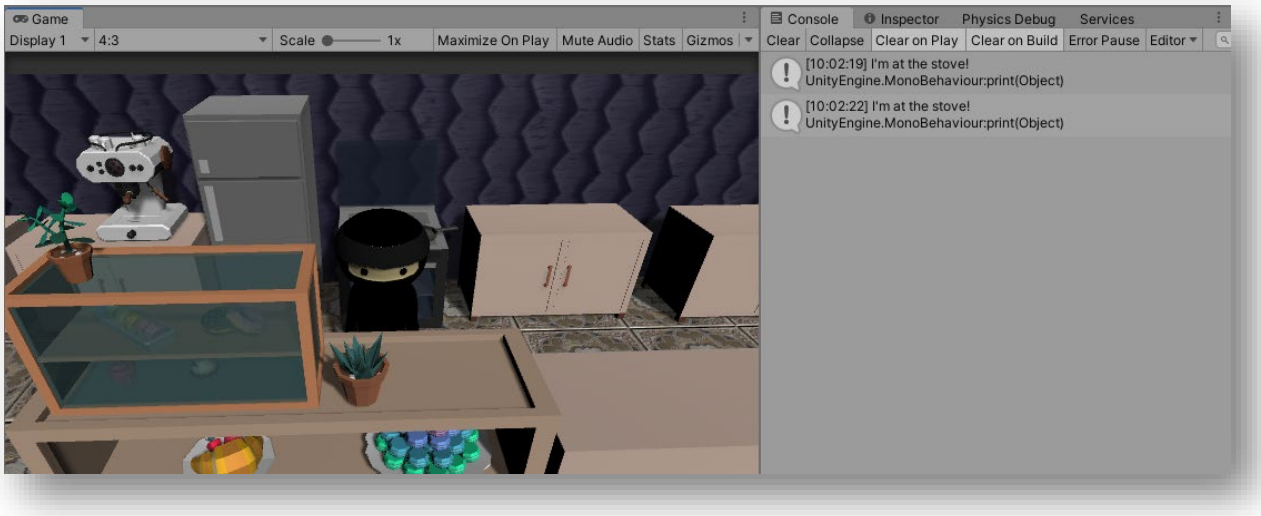


53 In the Interact **script**, we can add another **if statement** to print a message if the space key is pressed while Codey is inside the stove's **box collider trigger**.

Sensei Stop

Using the same conditional that we used for the bread trigger, can you code a message to appear if the space bar is pressed when Codey is inside the stove's box collider trigger?

54 Play the game and make sure you can see the message in the **console** when Codey is near the stove and you press the spacebar.



55 Now that Codey knows how to interact with the stove, we can toast the bread. We need an easy way to keep track of what item Codey is holding. In the Interact **script**, create a new **public string variable** named **heldItemName**.

```
public class Interact : MonoBehaviour
{
    public string triggerName = "";

    public GameObject breadPrefab;

    public GameObject heldItem;
    public string heldItemName;

    0 references
    void Update()
```

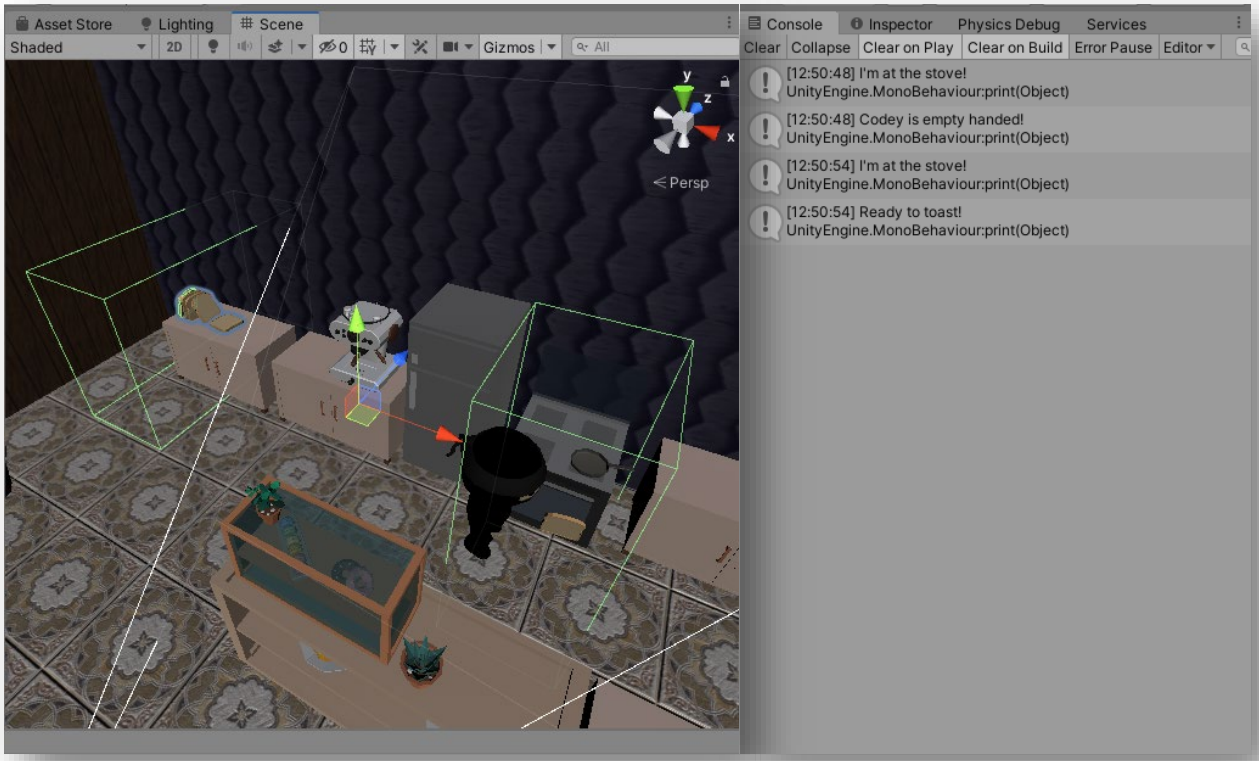
56 We need to set the **value** of this **variable** when Codey picks up a piece of bread. Inside the **if statement** that checks the bread **trigger**, set the **variable**'s **value** to "breadSlice".

```
0 references
void Update()
{
    if (Input.GetKeyDown("space"))
    {
        if (triggerName == "Bread")
        {
            heldItem = Instantiate(breadPrefab, transform, false);
            heldItem.transform.localPosition = new Vector3(0, 2, 2);
            heldItemName = "breadSlice";
        }

        if (triggerName == "Stove")
        {
            print("I'm at the stove!");
        }
    }
}
```

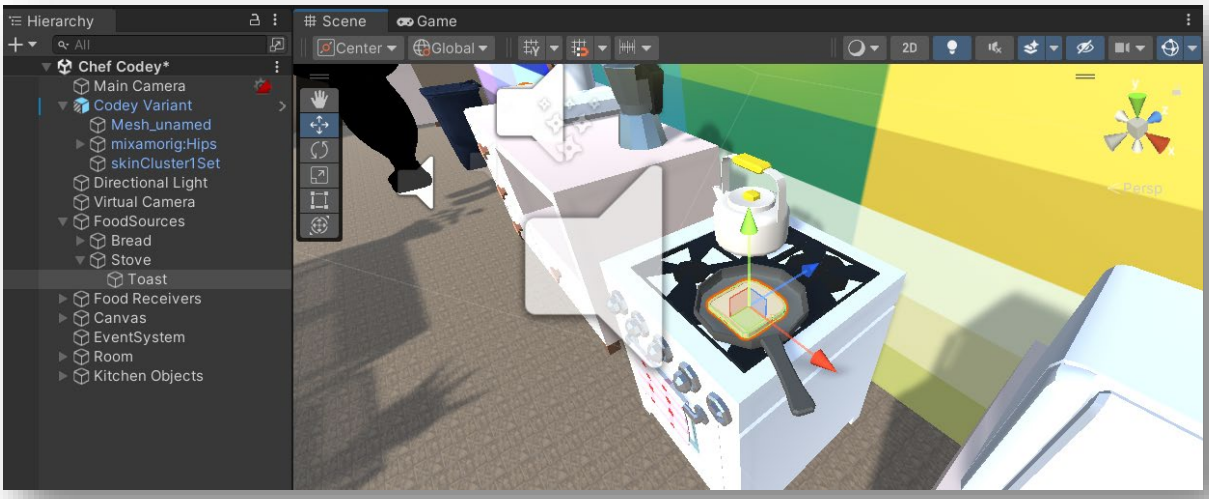
57 Now that we know what item Codey is holding, we can check to see **if** Codey is holding it when near the stove. Create **if else statements** for when Codey is holding the "breadSlice" and when Codey is not holding any items.

```
0 references
void Update()
{
    if (Input.GetKeyDown("space"))
    {
        if (triggerName == "Bread")
        {
            heldItem = Instantiate(breadPrefab, transform, false);
            heldItem.transform.localPosition = new Vector3(0, 2, 2);
            heldItemName = "breadSlice";
        }

        if (triggerName == "Stove")
        {
            print("I'm at the stove!");
            if (heldItemName == "breadSlice")
            {
                print("Ready to toast!");
            }
            else
            {
                print("Codey is empty handed!");
            }
        }
    }
}
```

58 Playtest your game. Press the spacebar near the stove with and without holding bread. Make sure that you can see all the different **console** messages.



59 Next, we need to place Codey's held item onto the stove. First, duplicate a bread object and add it to the empty Stove object. Position it wherever you want – on top or inside the stove. Rename it to "Toast" so it's easier to differentiate from our bread source.



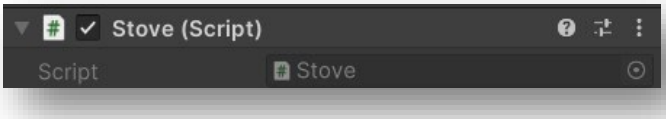
Like Magic

In game development, there are a lot of tricks that make the player think something complicated is happening when something simple is actually taking place. Instead of using the exact piece of bread that Codey is holding, we can give the stove its own piece of bread and destroy the one Codey is holding.

60 Codey's job is to pick up and place the items on the Stove, and the Stove's job is to cook the object. Since the Stove has its own job, we need to create a new **script** to manage the cooking logic.

Create a new **script** in your Script folder in the **Assets** tab and name it Stove.

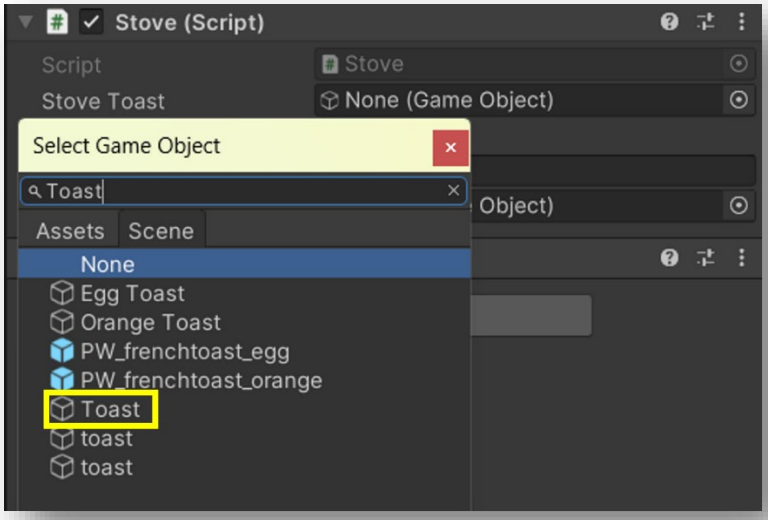
Attach the **script** to the Stove **game object** and open it in Visual Studio.



61 The Stove needs to know about the Toast **object**, so create a **public game object variable** named toast. You can delete the Update **function**.

```
public class Stove : MonoBehaviour
{
    public GameObject toast;

    0 references
    void Start()
    {

    }

}
```

62 Save and close Visual Studio. In the Unity **inspector**, set the **value** to the toast **object**.



63 Open the Stove **script** again and create a new **public void function** named ToastBread.

```
public class Stove : MonoBehaviour
{
    public GameObject toast;

    0 references
    void Start()
    {

    }
    0 references
    public void ToastBread()
    {

    }
}
```

Sensei Stop

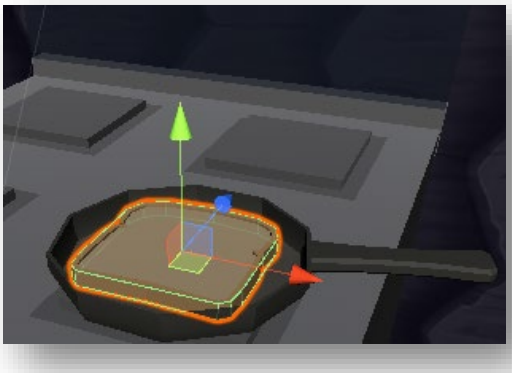
Write a few sentences or design a flowchart to describe the ToastBread function's job. What's the first thing that it needs to do when Codey interacts with the stove? What does it need to do before Codey can interact with the stove again?

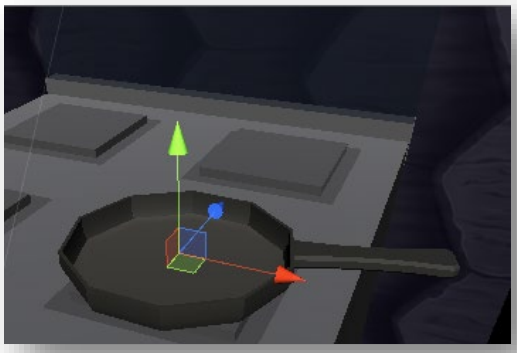
64 This **function** needs to take Codey's bread, place it on the stove, cook it, and let Codey pick it back up. Let's first have Codey put down and pick up the bread. We need to make sure there isn't a piece of toast on the stove when the game starts.

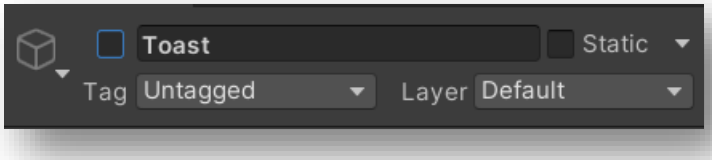
In the Start **function**, **deactivate** the toast **game object** when the game begins using the **game object**'s SetActive **function**.

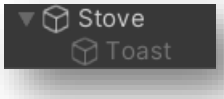
```
public class Stove : MonoBehaviour
{
    public GameObject toast;

    0 references
    void Start()
    {
        toast.SetActive(false);
    }
    0 references
    public void ToastBread()
    {

    }
}
```

65 Playtest your game to see the toast disappear when the game starts running.









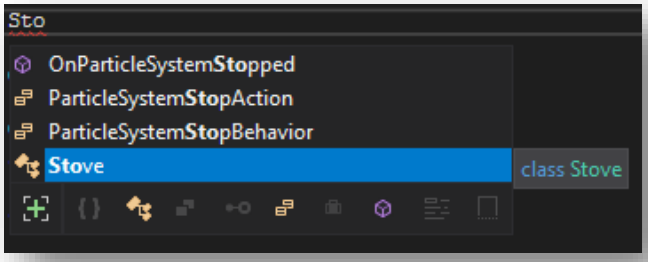
66 In our ToastBread **function**, we need to **enable** our toast object and make it available for Codey to pick up. In the **function**, we need to do the opposite of what we did in the Start **function**.

We can use the same SetActive **function**, but with **true** as the **parameter**.

```
public class Stove : MonoBehaviour
{
    public GameObject toast;

    0 references
    void Start()
    {
        toast.SetActive(false);
    }
    0 references
    public void ToastBread()
    {
        toast.SetActive(true);
    }
}
```

67

This **function** needs to run when the player presses the spacebar when Codey is standing next to the stove with a piece of bread. Open the Interact **script**. Add a **public variable** of the Stove **script**. This will let us reference any **variables** and **functions** from the Stove **script** inside our Interact **script**.

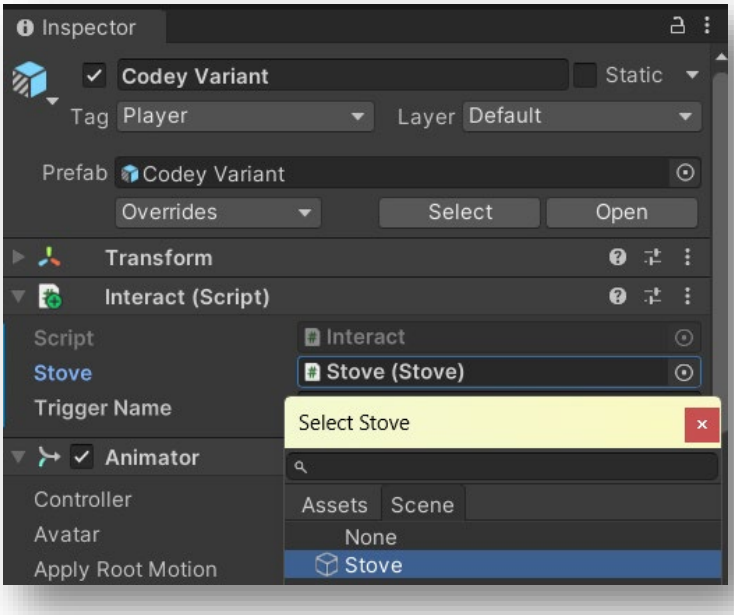


Visual Studio knows that we have a Stove **script** and will suggest it for us!

```
public class Interact : MonoBehaviour
{
    public Stove stove;

    public string triggerName = "";
```

68

Save the script and go back to Unity. Click on Codey in the hierarchy to open the components in the inspector. Set the Stove variable to the Stove object.



69 With the stove connected to our Interact **script**, we can have Codey run the stove's ToastBread **function** when the **conditions** are correct.

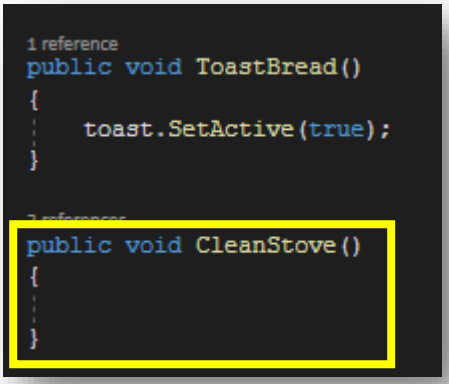
Sensei Stop

Find the correct if statement and call the stove's ToastBread function, destroy the object that Codey is holding, and reset the value of heldItemName to an empty string.

70 Playtest your game. Test what happens when Codey interacts with the stove with and without a piece of bread.

71 We can now have Codey pick up the toast from the stove. This requires the stove giving an object to Codey, so we need to build a little more **logic** inside of the Stove **script**.

Open the Stove **script** and create a new **public void function** named **CleanStove**.



72 We need to **disable** the toast object and tell Codey what **object** was picked up. Create a **public string variable** named cookedFood and **initialize** it to an empty **string**.

```
public class Stove : MonoBehaviour
{
    public GameObject toast;

    public string cookedFood = "";
```

73 Add a line to the ToastBread function that sets the cookedFood string to "toast".

```
1 reference
public void ToastBread()
{
    toast.SetActive(true);
    cookedFood = "toast";
}
```

74 Inside the CleanStove **function**, **disable** the toast **object** and reset the **value** of the cookedFood **variable** to an empty **string**.

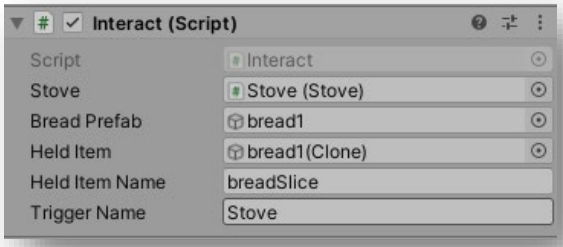
```
public void CleanStove()
{
    toast.SetActive(false);
    cookedFood = "";
}
```

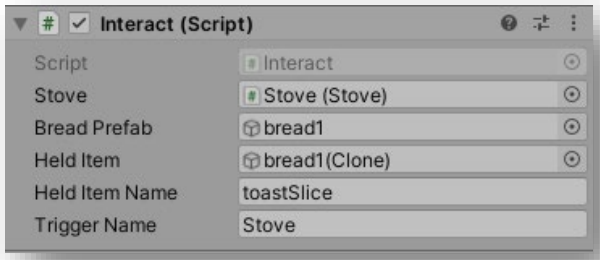
75 Back in the Interact **script**, we need to have Codey check to see if the cookedFood is ready.

If the stove has cooked toast, **then** give Codey the bread **prefab** and position it. Change the **value** of heldItemName to "toastSlice" and run the CeanStove **function**.

```
if (triggerName == "Stove")
{
    if (heldItemName == "breadSlice")
    {
        stove.ToastBread();
        Destroy(heldItem);
        heldItemName = "";
    }
    else
    {
        if (stove.cookedFood == "toast")
        {
            heldItem = Instantiate(breadPrefab, transform, false);
            heldItem.transform.localPosition = new Vector3(0, 2, 2);
            heldItemName = "toastSlice";
            stove.CleanStove();
        }
    }
}
```

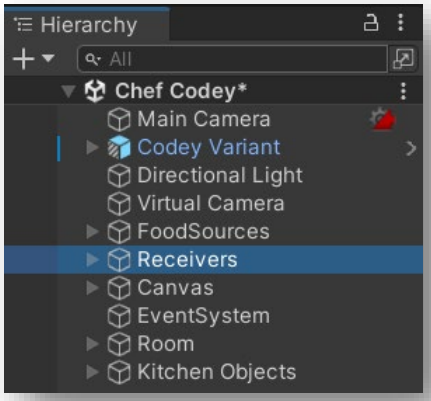
76 Playtest your game. Place a piece of bread on the stove and try to pick it up again. It should look like Codey cooked the bread!

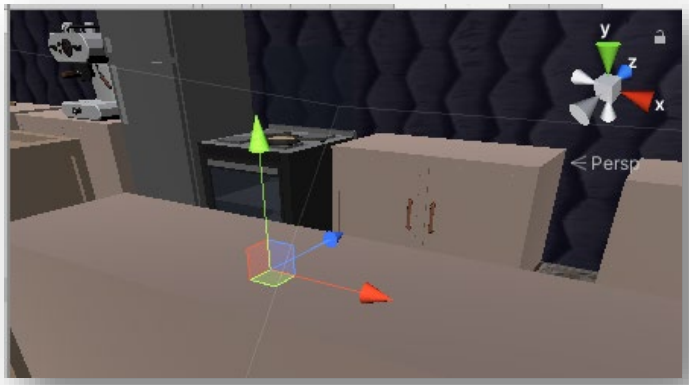




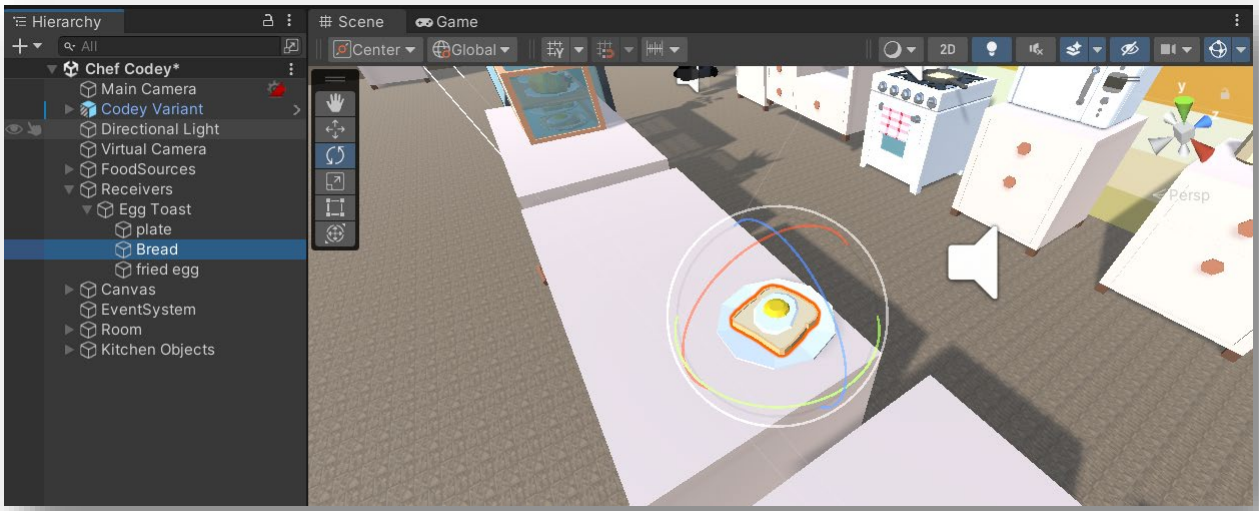
Even though they use the same **model**, our held item name tells us that Codey is holding a slice of toast.

77 Now we need a place to put our toast. Create a new empty **object** named Receivers and place it somewhere on a counter in your **scene**. This object will hold all our finished food items and dishes.

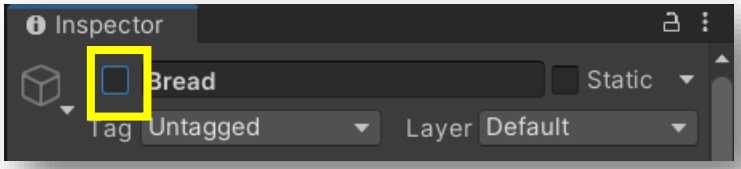


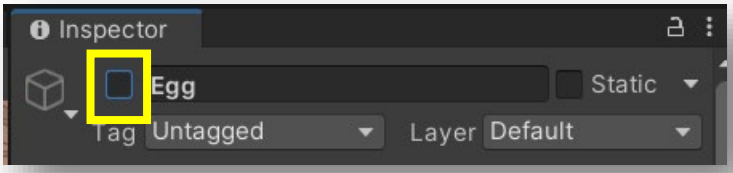


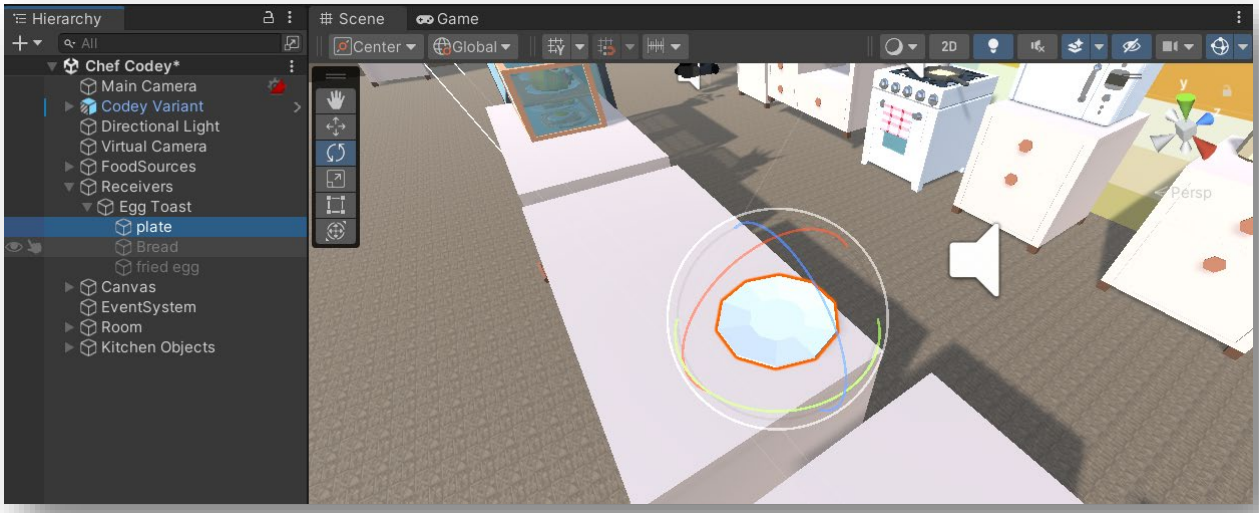
78 Find or build a finished **object** using **models** from the Unity Asset Store. Remember that we are making French Toast with an egg. Unpack your **prefab** and **rename** your objects in you need to. It is important for the object **names** to match up with the values we use for heldItemName.



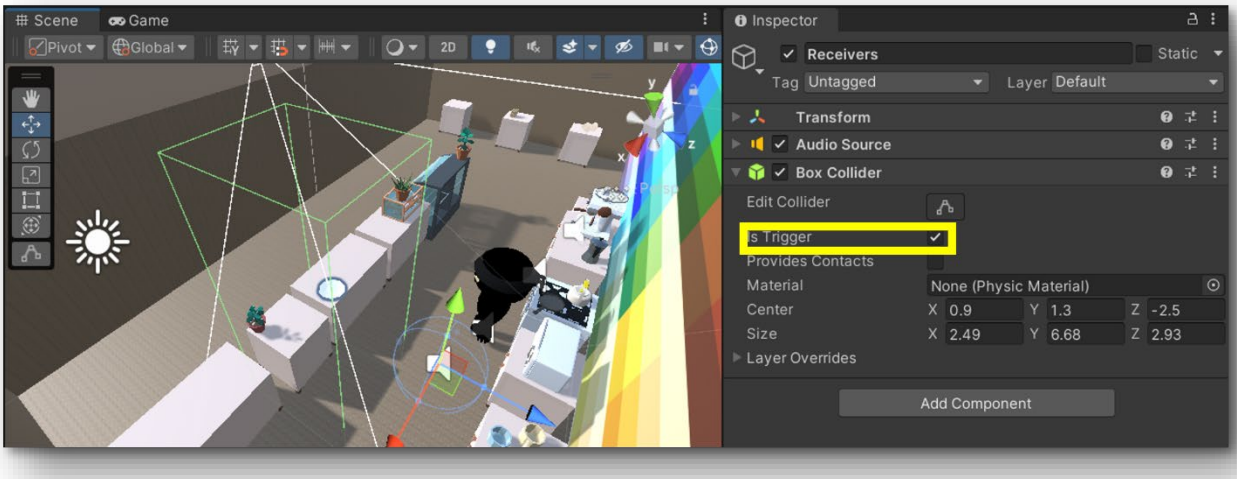
79 Once you have your items placed, **disable** the **objects** that you want Codey to cook by clicking the checkbox next to the name of the **object** in the **inspector**.



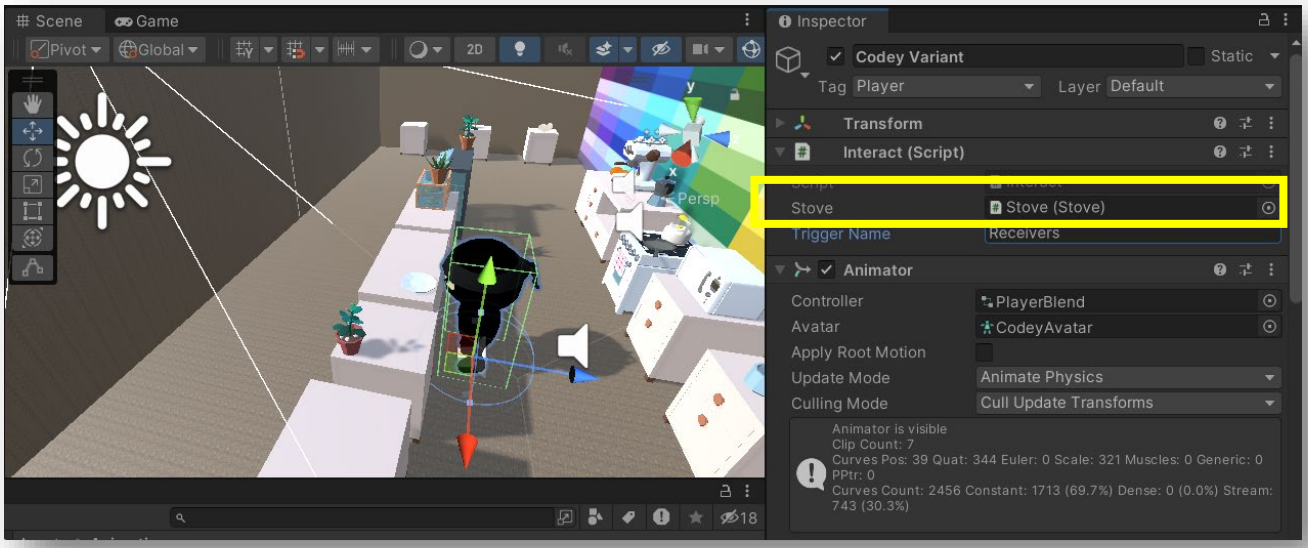




80 Add a **box collider** to the Receivers **object**. Make sure you set its **position** and **size**. Make sure that **Is Trigger** is enabled.



81 Playtest your game and move Codey inside of the new **collider**. Make sure the **name** is properly displaying for the **value** of Trigger Name in the **inspector**.



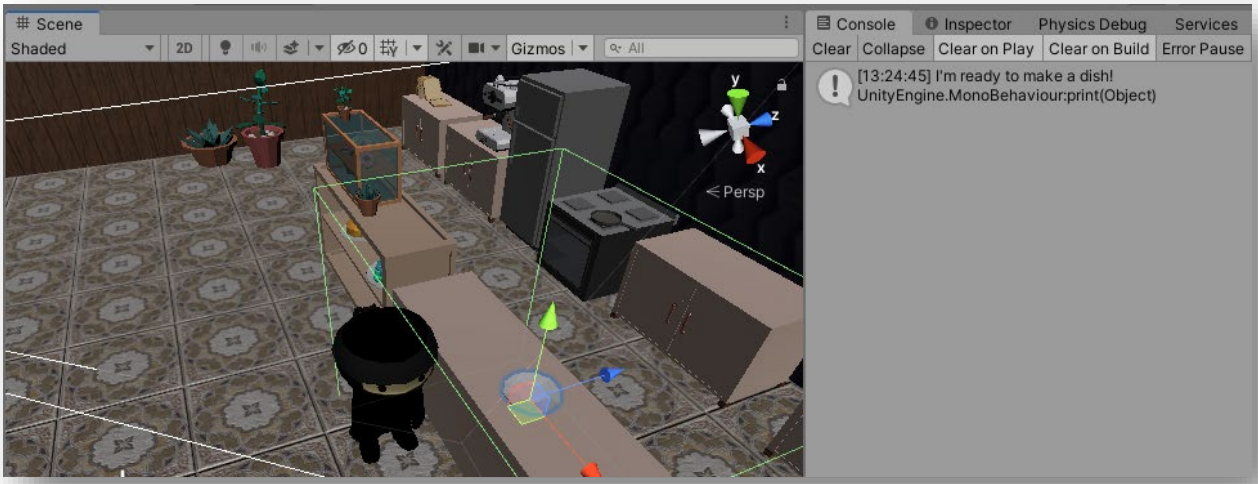
82 Open the Interact **script**. Add a new **if statement** in Codey's Update **function** inside the **if** space key pressed **conditional** that checks to see **if** Codey is inside the Receivers **trigger**.

```
void Update()
{
    if (Input.GetKeyDown("space"))
    {
        if (triggerName == "Bread")...

        if (triggerName == "Stove")...

        if (triggerName == "Receivers")
        {
            print("I'm ready to make a dish!");
        }
    }
}
```

Pro Tip

You can use the [-] buttons to the right of the line numbers to collapse or highlight groups of code to help you focus on coding in the right place.

83 Save and return to the Unity Engine. Playtest your game and make sure you see the message in the **console** when you press the space bar while Codey is standing close to the Receivers game **object**.



Notice how our **code** just works with our new **collider**? It's because we set up **OnTriggerEnter** and **OnTriggerExit** to handle all the work for us.

84 We can now write **code** to know when Codey is holding toast and when Codey's hands are empty.

Inside this new **if statement**, create an **if statement** that checks to see **if** Codey is holding the "toastSlice" **item**. In this **statement**, we want to clear out Codey's inventory by **destroying** the heldItem and resetting the value of heldItemName to an empty **string**. Use the following **pseudocode** to help you write your **statements**.

- If Codey is touching the trigger with the Receivers name, and
    - If Codey is holding an item with the toastSlice name,
        - Then destroy the item that Codey is holding, and
        - Change the value of heldItemName to nothing.

```
void Update()
{
    if (Input.GetKeyDown("space"))
    {
        if (triggerName == "Bread")...

        if (triggerName == "Stove")...

        if (triggerName == "Receivers")
        {
            if (heldItemName == "toastSlice")
            {
                Destroy(heldItem);
                heldItemName = "";
            }
        }
    }
}
```

85 Hmmm... this **code** should seem very, very familiar! We used these exact two **lines** when we wanted to toast our bread slice. Since we are repeating code, we can create a new **function** and use it any time we want to perform these actions.

Outside of the Update **function**, create a **private void function** named PlaceHeldItem. The **body** of this **function** should **destroy** the held item and reset the held item's name to an empty string.

```
private void PlaceHeldItem()
{
    Destroy(heldItem);
    heldItemName = "";
}
```

86 We can now **refactor** our **code**! This just means that we are reorganizing our **code** to make it more efficient. Wherever you used those two lines of **code** in your new **function**'s **body**, we can instead **call** the **function**!

```
void Update()
{
    if (Input.GetKeyDown("space"))
    {
        if (triggerName == "Bread")...
        if (triggerName == "Stove")
        {
            if (heldItemName == "breadSlice")
            {
                stove.ToastBread();
                PlaceHeldItem();
            }
            else
            {
                if (stove.cookedFood == "toast")...
            }
        }
        if (triggerName == "Receivers")
        {
            if (heldItemName == "toastSlice")
            {
                PlaceHeldItem();
            }
        }
    }
}
2 references
private void PlaceHeldItem()
{
    Destroy(heldItem);
    heldItemName = "";
}
```

87 **Playtest** your game to make sure we didn't break anything. Pick up bread and try to place it at the Receivers – nothing should happen because we are looking for toast!

Sensei Stop

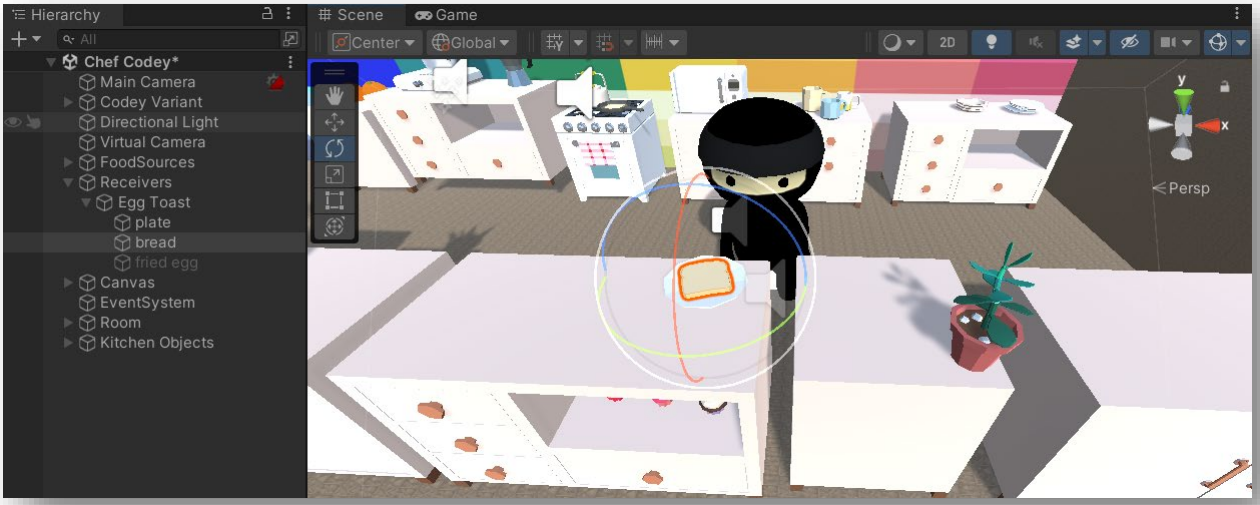
There's one other place that we have repeated code – we will create a function to help us soon! Talk with your Sensei about what code can be placed into a function.

88 We now need to make sure that Codey's toast doesn't disappear, and instead appears on the plate. We need to use the **GameObject class**'s Find **function** to search our **hierarchy** for the toastSlice **object** and set it **active**. It sounds like a lot, but it can be done in one line.

```
if (triggerName == "Receivers")
{
    if (heldItemName == "toastSlice")
    {
        PlaceHeldItem();
        GameObject.Find("Receivers/French Toast/toastSlice").SetActive(true);
    }
}
```

The slashes show that toastSlice belongs to French Toast which belongs to Receivers. Save the **script**.

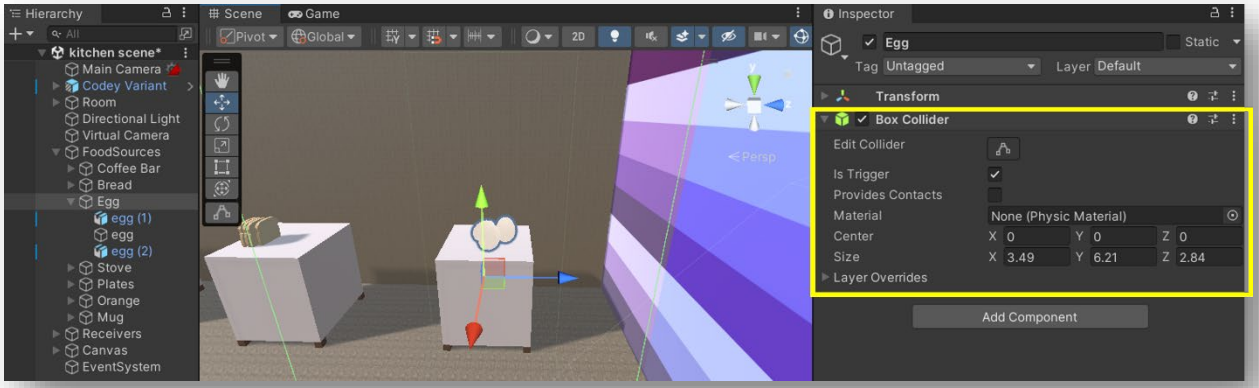
89 Playtest your game. Pressing the spacebar while Codey is holding the toastSlice at the Receivers will now put a piece of toast on the plate.



How to Program an Egg

90 We want to make sure we have a finished dish. We can repeat a lot of the same steps with eggs! Use the following checklist to help you complete all of the tasks to get a second interactable object working.

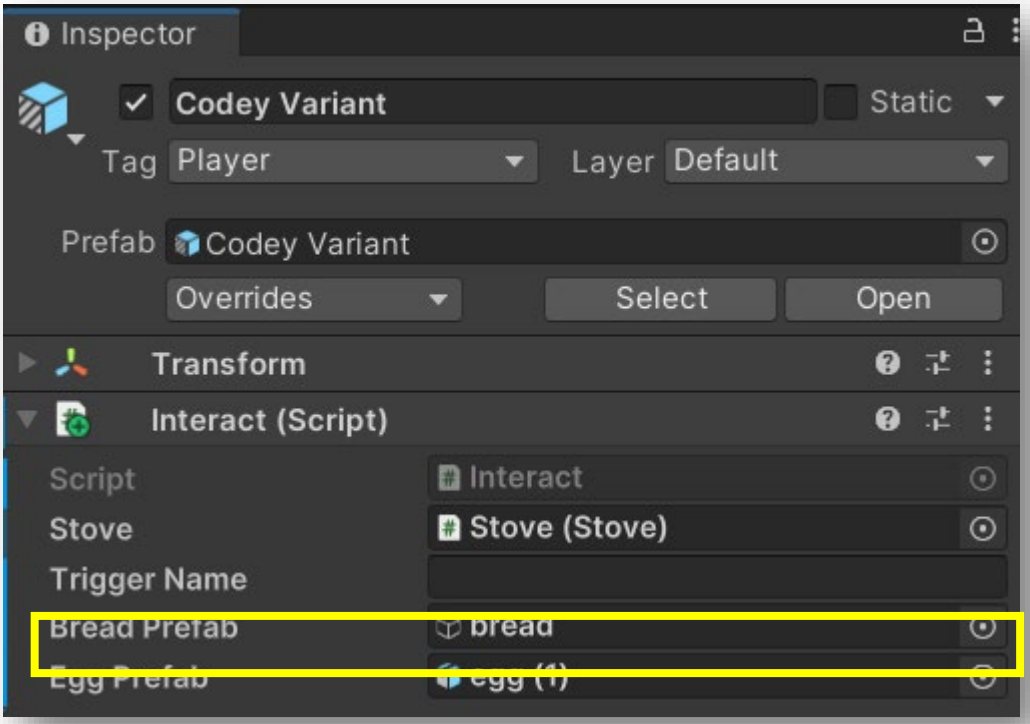
- Design an egg source location in your scene.
- Add an empty game object to the Sources object and name it.
- Search the asset store to find an egg object or make your own out of a capsule or sphere.
- Place your objects inside your new source object.
- Add a box collider to the new source object.
- Enable "Is Trigger" on the box collider.
- Change the position and size so Codey can reach it.
- Playtest your game and make sure Codey's "Trigger Name" property updates when inside the collider.



91 Just like the bread, we need to tell Codey what **prefab** to use when we want to pick up an egg. In the Interact **script**, add a **public game object** named eggPrefab.

```
public GameObject breadPrefab;
public GameObject eggPrefab;
```

92 Save your **script**. In the Unity **inspector**, assign the **value** of eggPrefab to one of the egg **game objects**.



93 We can now use the same code that we used to pick up the bread and the toast to pick up our egg.

Create a new **private void function** named PickUpItem. Copy and paste the three lines that **instantiate** the breadPrefab, set the heldItem's **position**, and changes the **value** of heldItemName into the **body** of the new **function**.

```
0 references
private void PickUpItem()
{
    heldItem = Instantiate(breadPrefab, transform, false);
    heldItem.transform.localPosition = new Vector3(0, 2, 2);
    heldItemName = "breadSlice";
}
```

94 We need to modify these three lines of Code to work for any potential item we want Codey to pick up.

Sensei Stop

Discuss with your Sensei what two elements of the code determine the held item's model and name. What special feature can we add to a function that lets us change its inputs?

95 We need to change breadPrefab and "breadSlice" into generic **parameters**. Change the declaration of the **function** to require one **GameObject** and one **string** as **parameters** and use these two **parameters** in the **body** of the **function**.

```
0 references
private void PickUpIte(GameObject itemPrefab, string itemName)
{
    heldItem = Instantiate(itemPrefab, transform, false);
    heldItem.transform.localPosition = new Vector3(0, 2, 2);
    heldItemName = itemName;
}
```

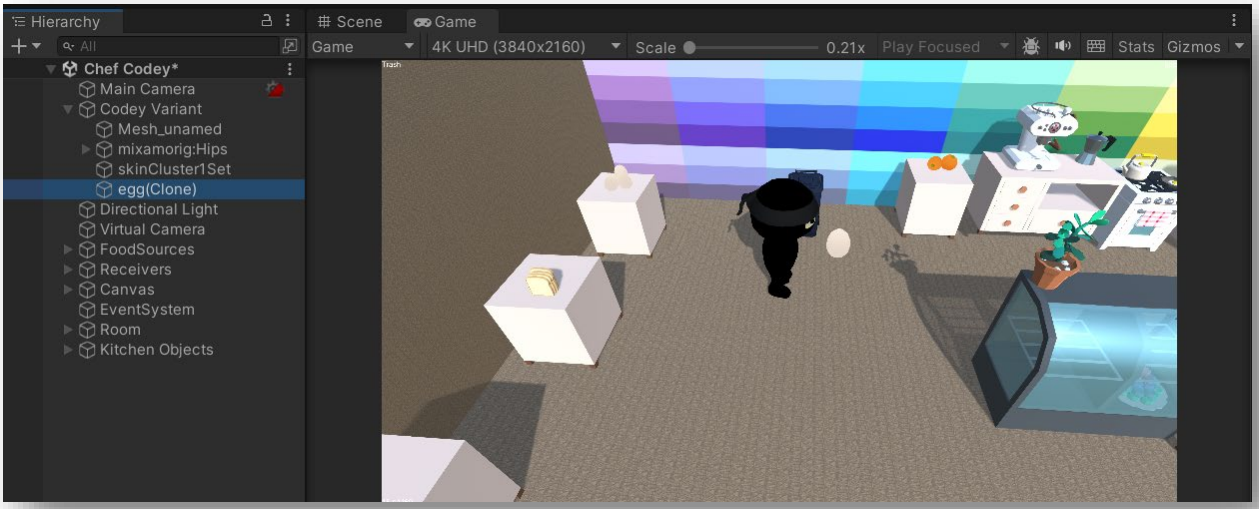
96 We can now use this **function** in our Update **conditional statements**. This **function** also means that Codey can now pick up any new item that you've created as a **source**. You just call this **function** with a **prefab** and an item name, and Codey will pick up the **object**.

```
0 references
void Update()
{
    if (Input.GetKeyDown("space"))
    {
        if (triggerName == "Bread")
        {
            PickUpItem(breadPrefab, "breadSlice");
        }

        if (triggerName == "Stove")
        {
            if (heldItemName == "breadSlice")
            {
                stove.ToastBread();
                PlaceHeldItem();
            }
            else
            {
                if (stove.cookedFood == "toast")
                {
                    PickUpItem(breadPrefab, "toastSlice");
                    stove.CleanStove();
                }
            }
        }

        if (triggerName == "Receivers")
        {
            if (heldItemName == "toastSlice")
            {
                PlaceHeldItem();
                GameObject.Find("Receivers/French Toast/toastSlice").SetActive(true);
            }
        }
    }
}

2 references
private void PickUpItem(GameObject itemPrefab, string itemName)
{
    heldItem = Instantiate(itemPrefab, transform, false);
    heldItem.transform.localPosition = new Vector3(0, 2, 2);
    heldItemName = itemName;
}
```

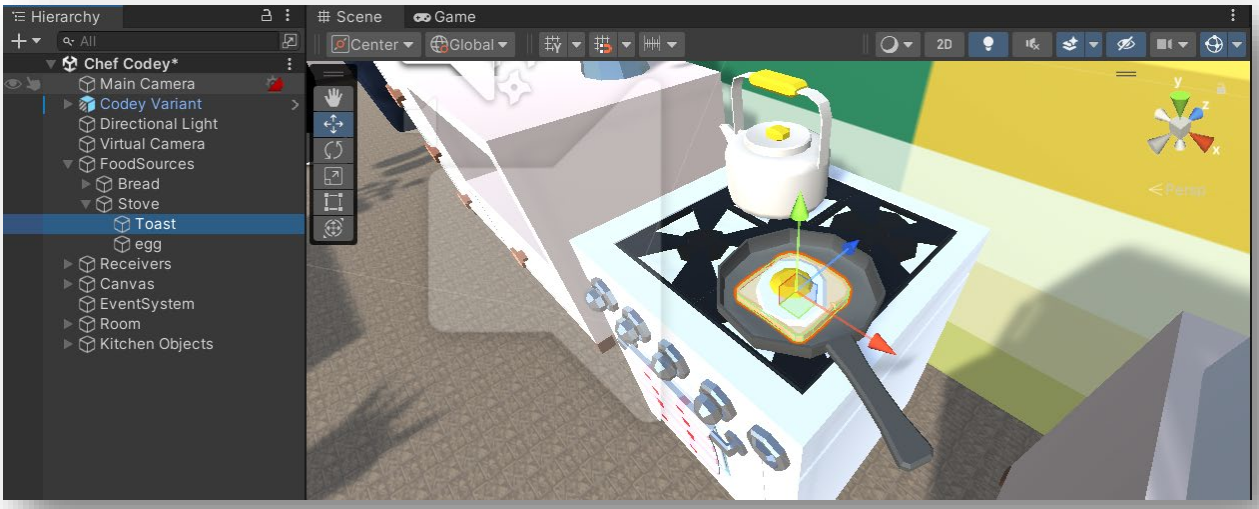
97 Use this new function in a new if statement that checks to see if Codey is standing in the Egg trigger.

```
0 references
void Update()
{
    if (Input.GetKeyDown("space"))
    {
        if (triggerName == "Bread")
        {
            PickUpItem(breadPrefab, "breadSlice");
        }

        if (triggerName == "Egg")
        {
            PickUpItem(eggPrefab, "egg");
        }

        if (triggerName == "Stove")
        {
            if (heldItemName == "breadSlice")
            {
                stove.ToastBread();
                PlaceHeldItem();
            }
            else
            {
                if (stove.cookedFood == "toast")
                {
                    PickUpItem(breadPrefab, "toastSlice");
                    stove.CleanStove();
                }
            }
        }

        if (triggerName == "Receivers")
        {
            if (heldItemName == "toastSlice")
            {
                PlaceHeldItem();
                GameObject.Find("Receivers/French Toast/toastSlice").SetActive(true);
            }
        }
    }
}
```

98 Save your **script** and playtest your game. What happens when you press the spacebar when Codey is in front of the eggs?



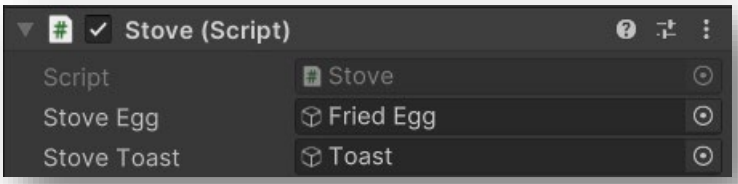
99 Codey is holding an egg! We now need to program the logic so the stove can fry it for us. Place a model for the fried egg inside our Stove Source object. Position it so it is inside the frying pan with the toast.



100 In the Stove **script**, create a **public GameObject** named friedEgg.

```
public class Stove : MonoBehaviour
{
    public GameObject toast;
    public GameObject friedEgg;

    public string cookedFood = "";
```

101 Save the **script** and attach the **variable** to the **game object** in the **inspector**. No matter what type of **object** you used, make sure that you drag the **object** that is on our **source** to the **script**!



102 Open the Stove **script**. Just like the toast, we need to **disable** the egg **object** on **Start** and create a **public function** that fries the egg so Codey can cook.

Sensei Stop

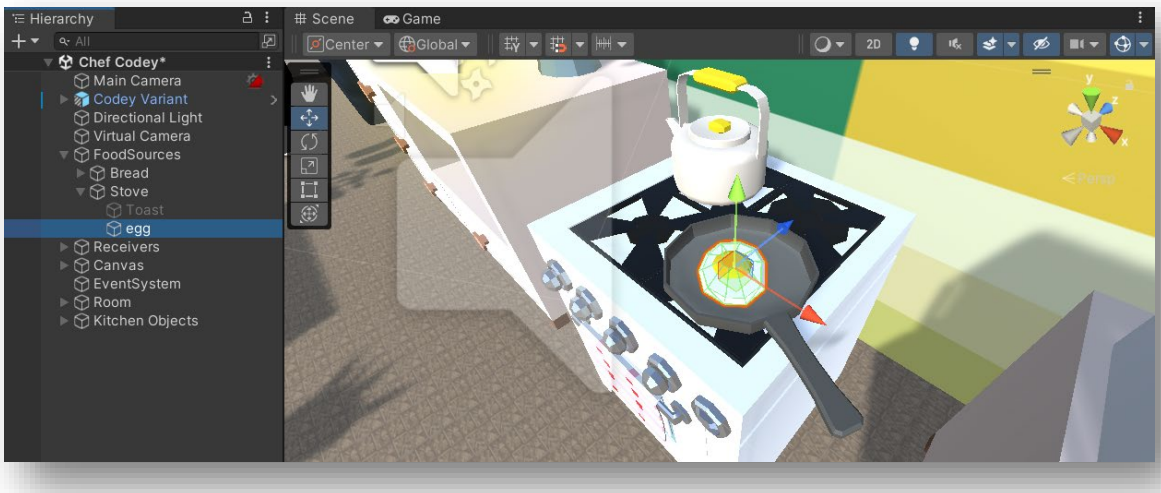
Look back at how we first disabled the toast and then wrote a public function for Codey to use. Can you do the same thing for the fried egg? First, disable the fried egg object when the script starts. Create a public void function to fry the egg that sets the object to active and updates the value of cookedFood to "friedEgg".

103 Back in the Interact **script**, we need to write the **logic** that cooks the egg if Codey is near the stove and holding an egg. Find the section of the code where Codey is **triggering** the stove. Add an else if statement that checks to see if the value of heldItemName is “egg” and then asks the stove to fry the egg and places the item being held.

```
if (triggerName == "Stove")
{
    if (heldItemName == "breadSlice")
    {
        stove.ToastBread();
        PlaceHeldItem();
    }
    else if (heldItemName == "egg")
    {
        stove.FryEgg();
        PlaceHeldItem();
    }
    else
    {
        if (stove.cookedFood == "toast")
        {
            PickUpItem(breadPrefab, "toastSlice");
            stove.CleanStove();
        }
    }
}
```

Pay close attention to where your if, else if, and else statements are! Use brackets and indentations to help you put code in the right place.

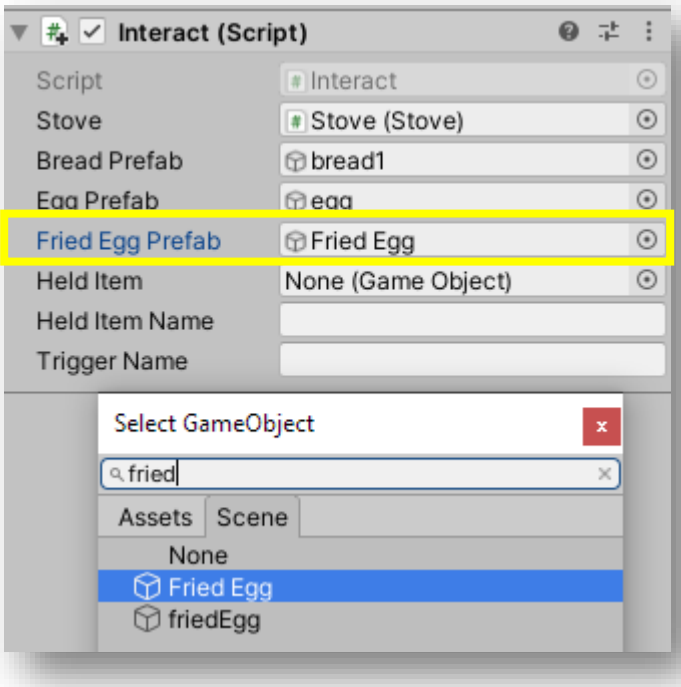
104 Playtest your game and make sure that Codey can take an egg and crack it onto the stove.



105 Now we can **program** Codey to pick up the fried egg from the stove. We can again use the **functions** we created to make this step very easy!

In the Interact **script**, add a new **public GameObject variable** named FriedEgg and attach a fried egg **model** to the **variable** in the **inspector**.

```
public class Interact : MonoBehaviour
{
    public Stove stove;

    public string triggerName = "";

    public GameObject breadPrefab;
    public GameObject eggPrefab;
    public GameObject friedEggPrefab;

    public GameObject heldItem;
    public string heldItemName;
```



106 Back in the Interact **script**, add the **logic** that asks the stove to pick up food and runs the PickUpItem **function** with the fried egg **prefab** and “friedEgg” as the item name.

```
if (triggerName == "Stove")
{
    if (heldItemName == "breadSlice")
    {
        stove.ToastBread();
        PlaceHeldItem();
    }
    else if (heldItemName == "egg")
    {
        stove.FryEgg();
        PlaceHeldItem();
    }
    else
    {
        if (stove.cookedFood == "toast")
        {
            PickUpItem(breadPrefab, "toastSlice");
            stove.CleanStove();
        }
        if (stove.cookedFood == "friedEgg")
        {
            PickUpItem(friedEggPrefab, "friedEgg");
            stove.CleanStove();
        }
    }
}
```

107 Playtest your game to make sure Codey can pick up the fried egg.

Sensei Stop

Oh no! There's still egg in the stove. Something must be wrong with our CleanStove function. Investigate and tell your Sensei what one line of code needs to be added to fix this bug.

108 Playtest your game to make sure both the toast and the fried egg disappear from the stove.

109 We can finally finish our dish! Open the Interact **script**.

Look back at how you **coded** the toast slice. Add the **code** that **finds** and **enables** the fried egg **object** on our French Toast **receiver**. If Codey is standing inside the Receivers **trigger** and **if** Codey is holding the friedEgg **object**, **then** place Codey's held item, **find** the French Toast's fried egg **object**, and **activate** it.

```
if (triggerName == "Receivers")
{
    if (heldItemName == "toastSlice")
    {
        PlaceHeldItem();
        GameObject.Find("Receivers/French Toast/toastSlice").SetActive(true);
    }
    if (heldItemName == "friedEgg")
    {
        PlaceHeldItem();
        GameObject.Find("Receivers/French Toast/friedEgg").SetActive(true);
    }
}
```

110 Playtest your game and cook some French Toast! Make sure that none of your **trigger boxes** overlap. They should be far enough apart so Codey cannot stand in two at the same time.

Time to Sizzle

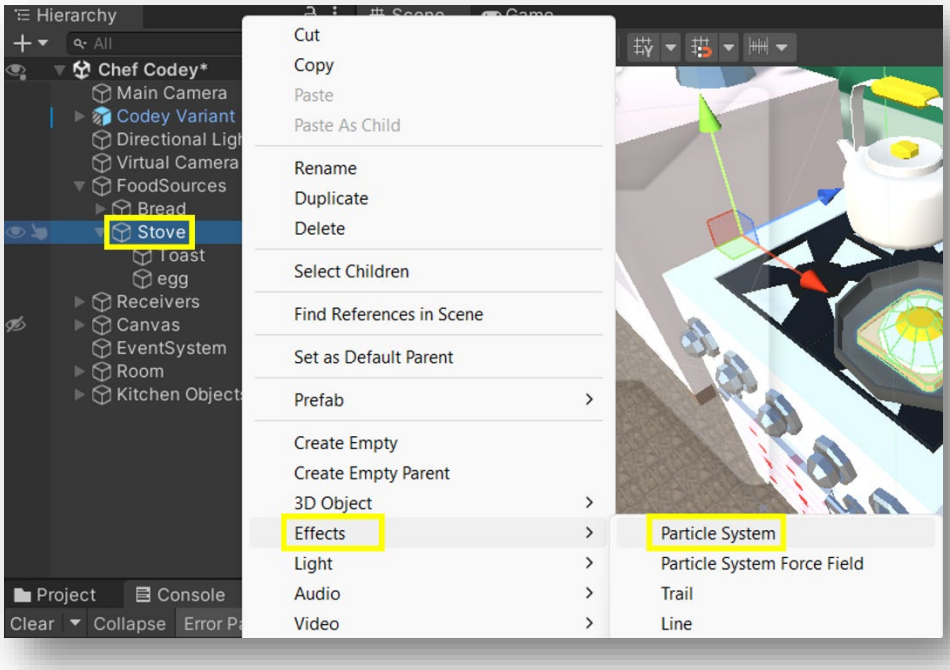
111 The core of our game is complete! We will now cover some visual touches that help communicate mechanics to the player. In the Prove Yourself you will add additional objects for Codey to make.

Right now, the stove toasts the bread and fries the egg instantly. We can add some effects to make it seem like it's actually cooking our items.

Ninja Planning Document

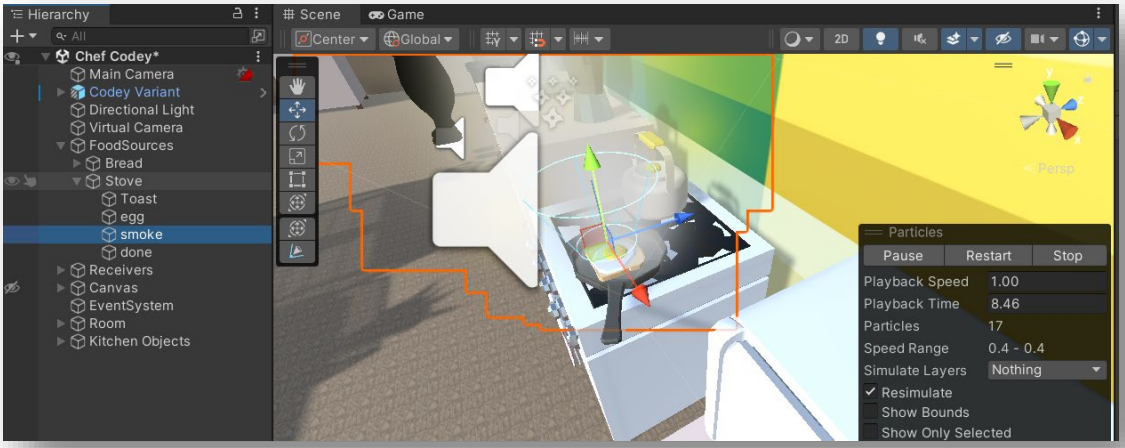
Take at least 5 but no more than 10 minutes and complete your Ninja Planning Document – Communicating to the Player

112 We can use the Unity particle system to make the stove cook our food. Add a **particle system object** to the Stove Source **object**.



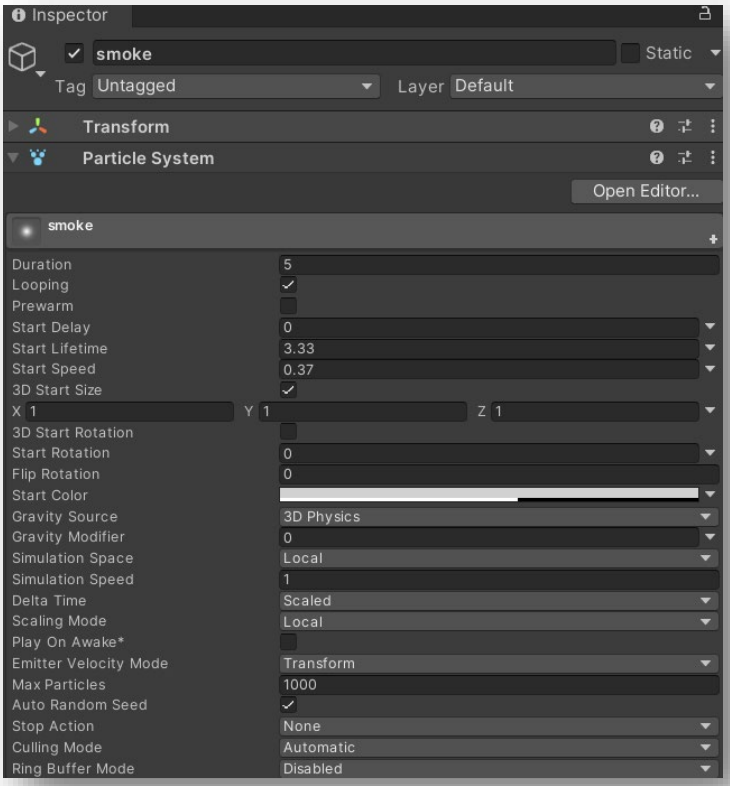
113

Rename your particle system to “smoke” and position it on the frying pan.

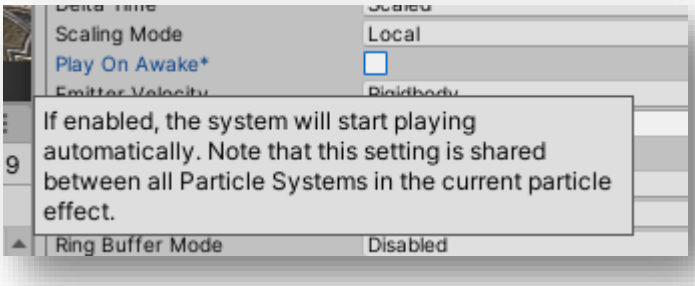


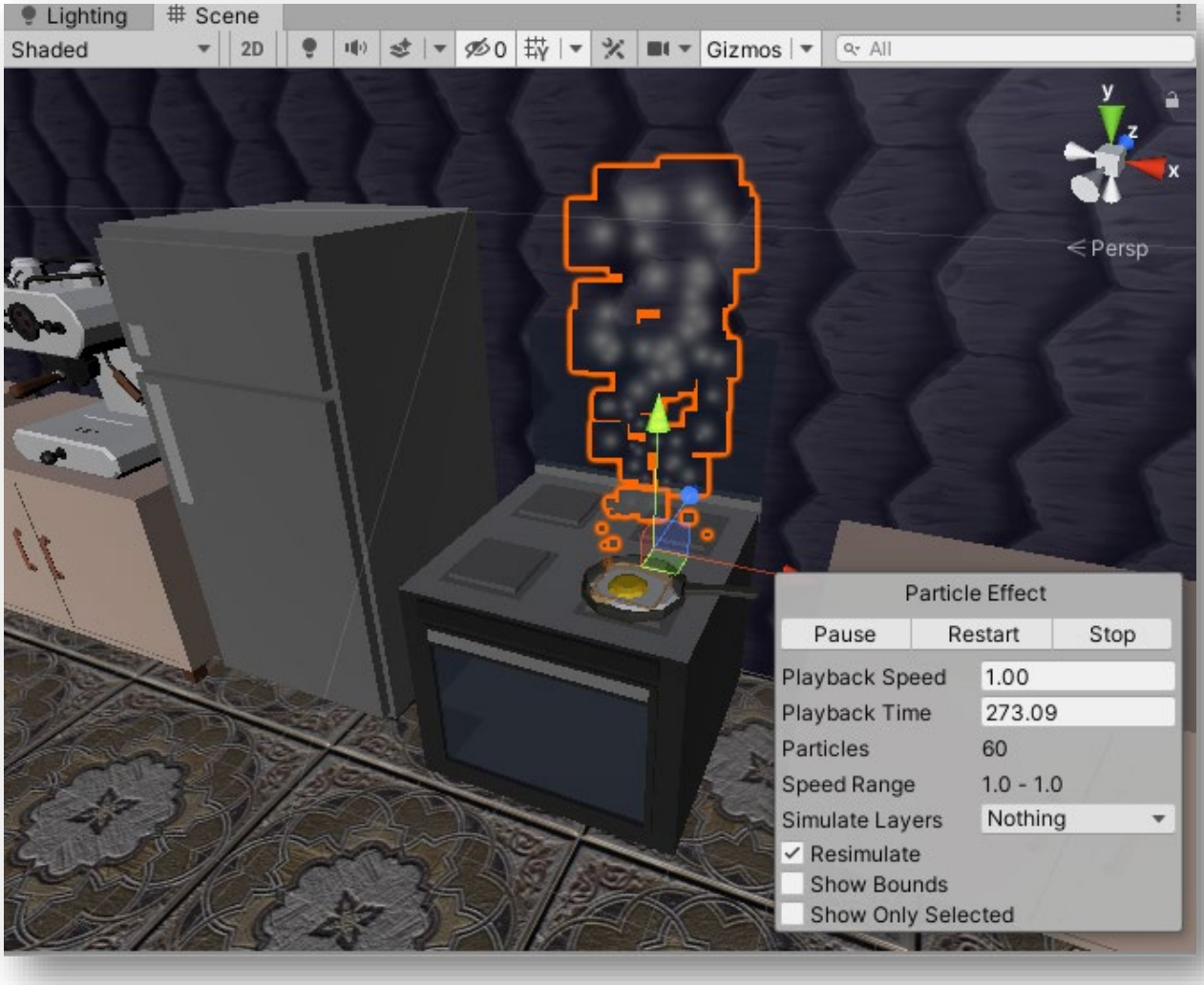
114

Experiment with all the options in the **Particle System component** to create the effect you designed in your Ninja Planning Document.



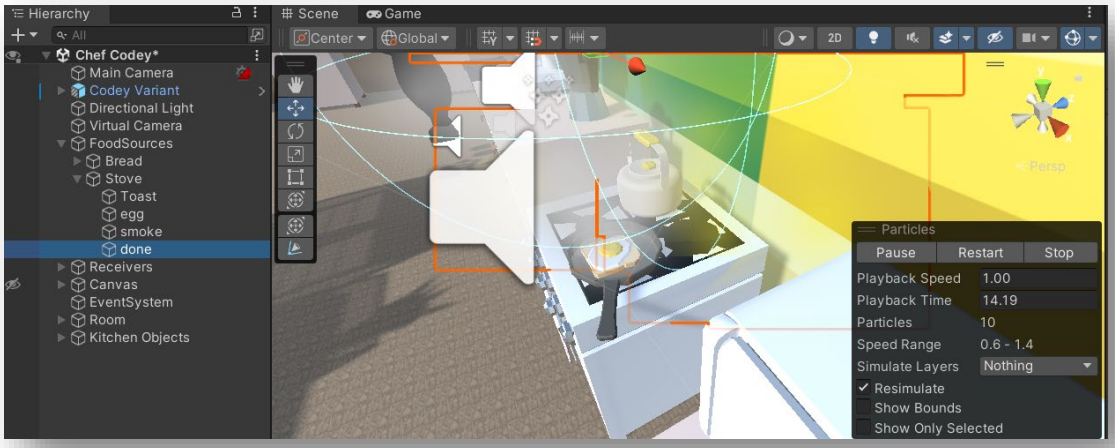
115 Don't be afraid to edit any of these **values**! The only one that you need to be sure is to **disable** is **Play On Awake**. If enabled, **Play On Awake** tells the **particle system** to play as soon as the game starts. Since we want Codey to control when the stove starts cooking, we need to **disable** the option.





116 Once you are satisfied with your first **particle system**, create a second **particle effect system** named “complete” that will indicate that cooking has finished.

Experiment with changing the **texture** located in the Shape **properties**.



117 We can now move to the Stove **script** and **program** the **particle effects** to play and stop based on some simple cooking **logic**.

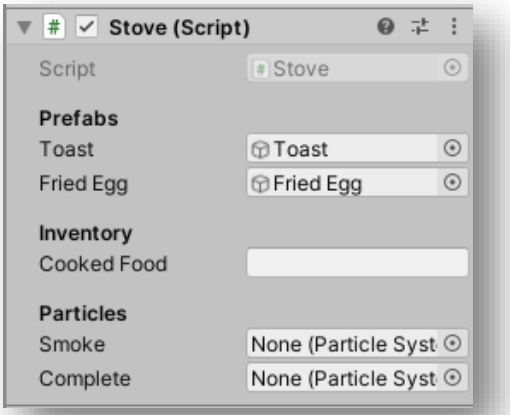
Create two **public ParticleSystem variables** named smoke and complete.

```
public class Stove : MonoBehaviour
{
    public GameObject toast;
    public GameObject friedEgg;

    public string cookedFood = "";

    public ParticleSystem smoke;
    public ParticleSystem complete;
```

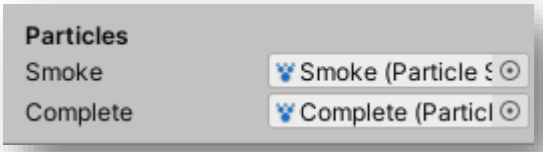
118 We now have a few **scripts** with several **public variables**. We can add a special line of code that helps us organize these **variables**. Add [Header("Prefabs")], [Header("Inventory ")], and [Header("Particles")] above the appropriate **variables**.

You can see what these headers did in the Unity inspector.

```
public class Stove : MonoBehaviour
{
    [Header("Prefabs")]
    public GameObject toast;
    public GameObject friedEgg;

    [Header("Inventory")]
    public string cookedFood = "";

    [Header("Particles")]
    public ParticleSystem smoke;
    public ParticleSystem complete;
```



119 Assign the smoke and complete **particle systems** to the Stove **script**.



120 Inside our two cooking **functions**, tell the smoke **particle system** to **play**.

```
1 reference
public void FryEgg()
{
    smoke.Play();
    friedEgg.SetActive(true);
    cookedFood = "friedEgg";
}
1 reference
public void ToastBread()
{
    smoke.Play();
    toast.SetActive(true);
    cookedFood = "toast";
}
```

121 Playtest your game and see what happens when you try to make toast or fry an egg.

122 We now need to **program** our stove to take time to cook. We can accomplish this by **Invoking** a **function** after a number of seconds.

In the Stove **script**, create a **private void function** named CompleteCooking. This **function** should **stop** the **smoke particle effect** and **play** the **complete particle effect**.

```
0 references
private void CompleteCooking()
{
    smoke.Stop();
    complete.Play();
}
```

123 In our two cooking **functions**, use Unity's **Invoke function** to call CompleteCooking after a few seconds.

```
1 reference
public void FryEgg()
{
    smoke.Play();
    friedEgg.SetActive(true);
    cookedFood = "friedEgg";
    Invoke("CompleteCooking", 8f);
}
1 reference
public void ToastBread()
{
    smoke.Play();
    toast.SetActive(true);
    cookedFood = "toast";
    Invoke("CompleteCooking", 6f);
}
```

Invoke

The Invoke function takes two parameters. The first parameter is a string of the function name that you want to run. The second parameter is the time you want to wait until you run the function. The second parameter has an "f" after the number because it needs to be a float, a special type of number that can use decimals.

124 Now playtest your game and wait to see if your **particles** start and stop properly.

125 We now need to stop the complete **particle effect** when Codey picks up the food in the CleanStove **function**.

```
2 references
public void CleanStove()
{
    toast.SetActive(false);
    friedEgg.SetActive(false);
    cookedFood = "";
    complete.Stop();
}
```

126 Playtest your game and make sure the **particles** stop after Codey picks up the food.

> **Sensei Stop**
>
> What happens when you try to pick up food before it's complete? Does anything stop Codey from picking up food immediately? Discuss with your Sensei how you could fix this bug.

Think about the following questions to help you think about a solution.

- Do you need to create a new **variable**?
- Should this **variable** be **public** or **private**? Why?
- What type of **variable** can be used to answer the question "is the stove cooking, yes or no?"
- What **functions** need to update the **value** of this **variable**?

127 Right now, Codey can always interact with the stove. We need Codey to know that the stove is in use so you can't pick up food too early or cook more than one item at a time.

Add a **public bool variable** named isCooking to the Stove **script**. Initialize it to **false**.

```
public class Stove : MonoBehaviour
{
    [Header("Prefabs")]
    public GameObject toast;
    public GameObject friedEgg;

    [Header("Inventory")]
    public string cookedFood = "";
    public bool isCooking = false;

    [Header("Particles")]
    public ParticleSystem smoke;
    public ParticleSystem complete;
```

128 We need to toggle this **variable** to **true** whenever the stove is cooking and toggle it to **false** whenever it is done.

```
1 reference
public void FryEgg()
{
    isCooking = true;
    smoke.Play();
    friedEgg.SetActive(true);
    cookedFood = "friedEgg";
    Invoke("CompleteCooking", 8f);
}
1 reference
public void ToastBread()
{
    isCooking = true;
    smoke.Play();
    toast.SetActive(true);
    cookedFood = "toast";
    Invoke("CompleteCooking", 6f);
}
```

```
0 references
private void CompleteCooking()
{
    isCooking = false;
    smoke.Stop();
    complete.Play();
}
```

129 Now that the stove can tell Codey when it is cooking, we need to program the logic to prevent Codey from interacting with a cooking stove.

Sensei Stop

Talk with your Sensei about how you can accomplish this. Before you add any code to your scripts, write a few sentences of pseudocode that describe when Codey can and cannot interact with the stove. Implement your code and playtest your game.

Project Submission and Reflection

Now that you have a working game, take time to add your own personal touch to the project. What aspects from other games could you add?

Once you feel like you have a good product that represents your vision of the game, have a Code Sensei and at least one other Ninja playtest it. Use the Playtest Survey Planning Document for questions to ask them when they finish. Record their answers in your Ninja Planning Document.

Based on the results of the playtest and survey, make changes to your game. Once you are complete, share the updates with your Code Sensei and fill out the reflection section of your Ninja Planning Document.

Before you submit your game for grading, use the Chef Codey Project Requirements Checklist to make sure your game has all the required features.

Ninja Planning Document

Use your Ninja Planning Document to record feedback from Senseis and other Ninjas in your Center.