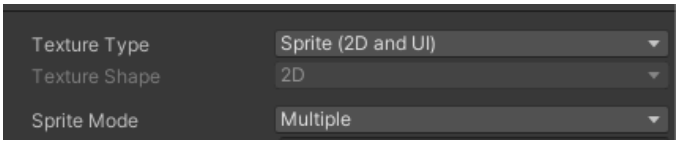
Microgame #7 Action Game Progressing Report
Name: Alan L. Perez
id: 004862867
Unity Play: https://play.unity.com/mg/other/microgame-7-action-game-cse-4410

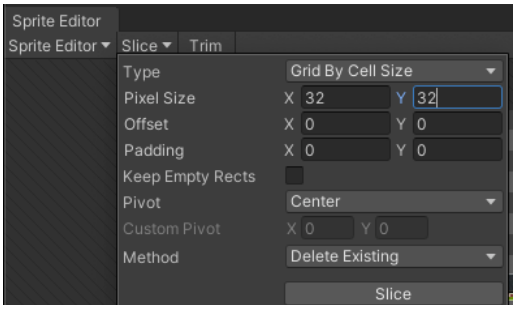
1.) Creating new project “Action”
2.) Create folders under Assets: Animations, Prefabs, Scenes, Scripts, Sprites, Tiles
3.) Download Action.zip from Canvas. Unzip, and drag sprites into Sprite folder. Put the Pride font under the Assets folder.
4.) Open the Player, Slim, and Tiles folder, select all the sprites and change “Pixel Per Unit” to 32. Also change “Filter Mode” to “Point (no filter)”.

5.) Then we go into GameObject -> 2D -> Tilemap. Now we have a “Grid”, and “Tilemap”

6.) In “Sprite Mode” select “Multiple”.

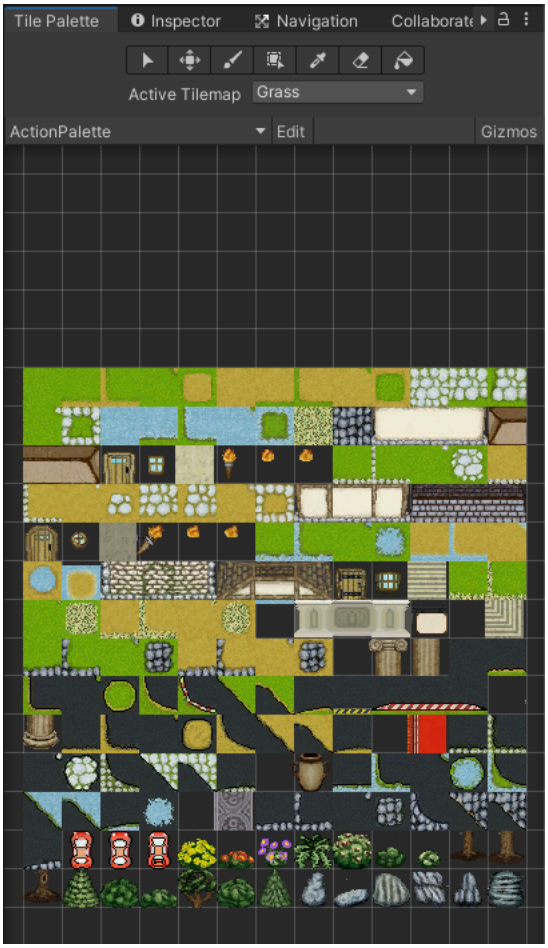


7.) Then go into the Sprite Editor and click “Slice”, then click “Grid By Cell Size”. Then make “Pixel Size” X: 32 Y: 32. Then click “Slice” and “Apply”.

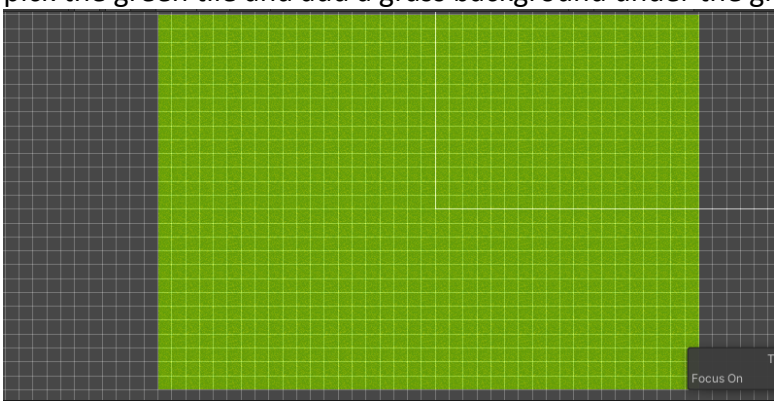


8.) Then we go into “Tile Palette” and “Create New Palette”. We will name this “ActionPalette”. Then we save this under the “Tiles” folder.

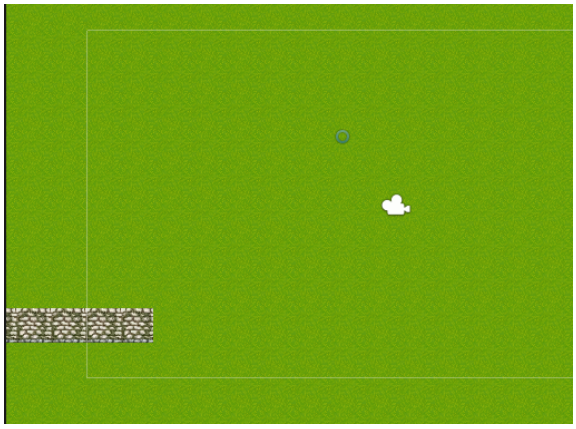
9.) Then go into the “Tiles” folder and select sprites 1-181, and drag them into the “Tile Palette”. We save this under “Tiles” folder.



10.) Now we have a “Grid”, and “Tilemap”. Rename “Tilemap” to “Grass”. Then we pick the green tile and add a grass background under the grass game object.

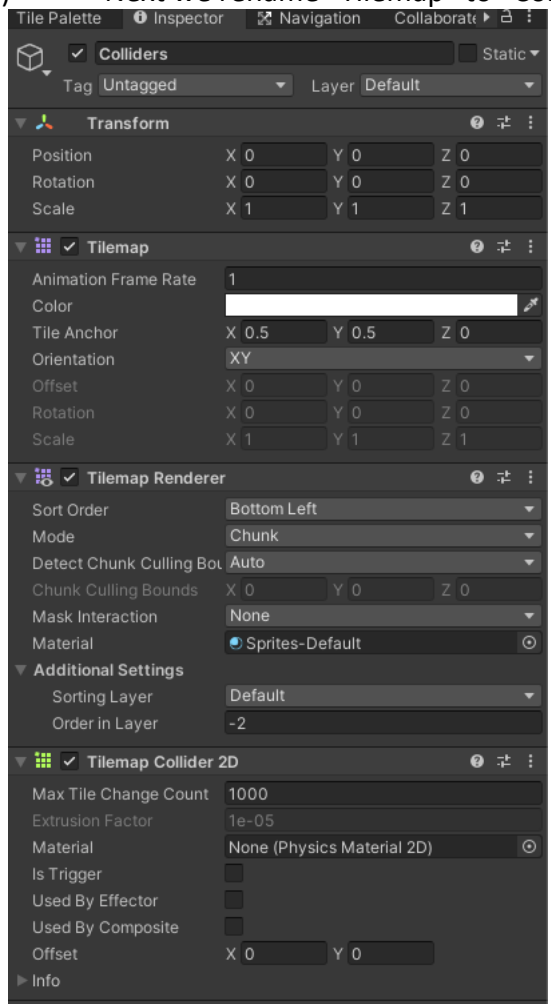


11.) Then we add another tile map. We will add wall tiles to it.

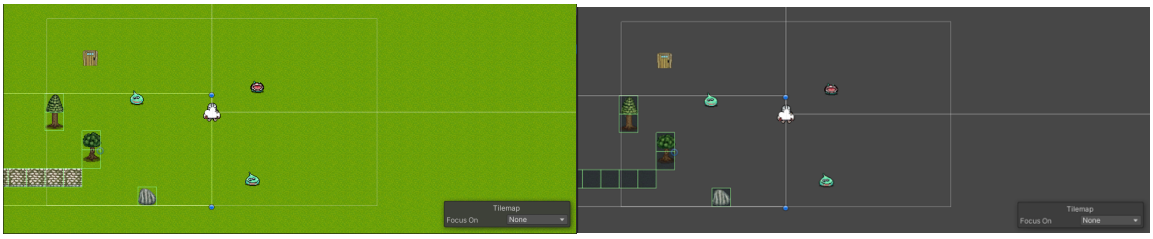


12.) Next we create an empty game object named “Trees”. Then we drag “Tile_170” , “Tiles_168”, “Tiles_171”, “Tiles_169”, and “Tiles_178” and position the tiles, and make them children of “Tree”.

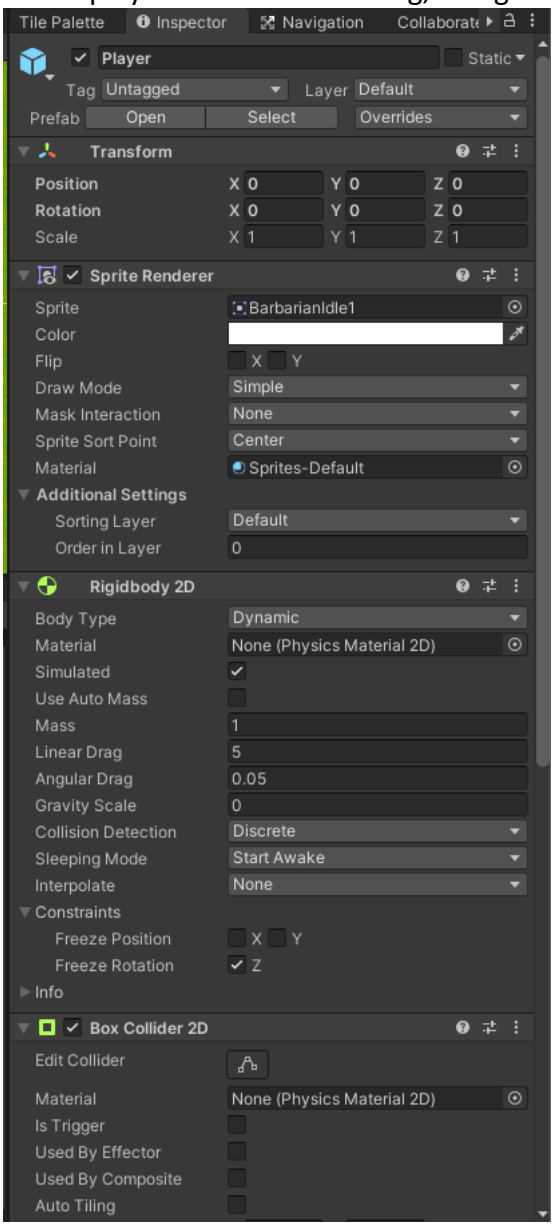
13.) Next we rename “Tilemap” to “Collider” and we add a “Tilemap Collider 2D”



14.) Then under the tree, stone, and wall tiles we put black tiles under them so we have colliders, essentially creating a collider map under the tiles.



15.) Next we will create animation for the player. We will drag “Barbarian Idle 1 (Texture 2D)” to the game object to the Hierarchy. We change the name to “Player”, we then create the script file “PlayerController.cs”, “Rigid Body 2D”, and “Box Collider 2D” to the player. We set linear drag, and gravity.



16.) Then we select sprites “Barbarian Idle 1/2/3 (Texture 2D)” and we drag it to the “Player” game object. We save it under the “Animations” folder as “Idle”.

17.) When then edit the “Player” collider to the size we want.

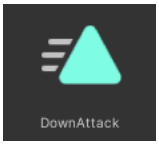
18.) Next, in a similar fashion as part 16 we create an animation of the player walking to the side by selecting all the sprites that pertain to that action and drag it to the player in the Hierarchy. We name this side as “SideWalk”.

19.) Next, in a similar fashion as part 16 and 18 we create an animation of the player walking down by selecting all the sprites that pertain to that action and drag it to the player in the Hierarchy. We name this side as “DownWalk”.

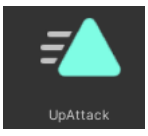
20.) Next, in a similar fashion as part 16, 18, 19 we create an animation of the player walking up by selecting all the sprites that pertain to that action and drag it to the player in the Hierarchy. We name this side as “UpWalk”.

21.) Next, we create an animation of the player attacking to the side by selecting all the sprites that pertain to that action and drag it to the player in the Hierarchy. We name this side as “SideAttack”.

22.) Next, we create an animation of the player attacking down by selecting all the sprites that pertain to that action and drag it to the player in the Hierarchy. We name this side as “DownAttack”.

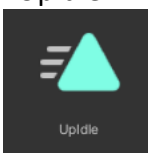


23.) Next, we create an animation of the player attacking up by selecting all the sprites that pertain to that action and drag it to the player in the Hierarchy. We name this side as “UpAttack”.



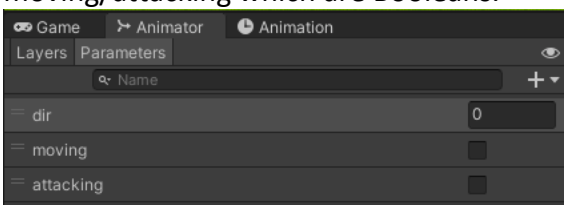
24.) Next, we create an animation of the player attacking up by selecting all the sprites that pertain to that action and drag it to the player in the Hierarchy. We name this side as “UpAttack”.

25.) Then we select pertaining to when the character in facing up and Idle and we drag it to the “Player” game object. We save it under the “Animations” folder as “UpIdle”.

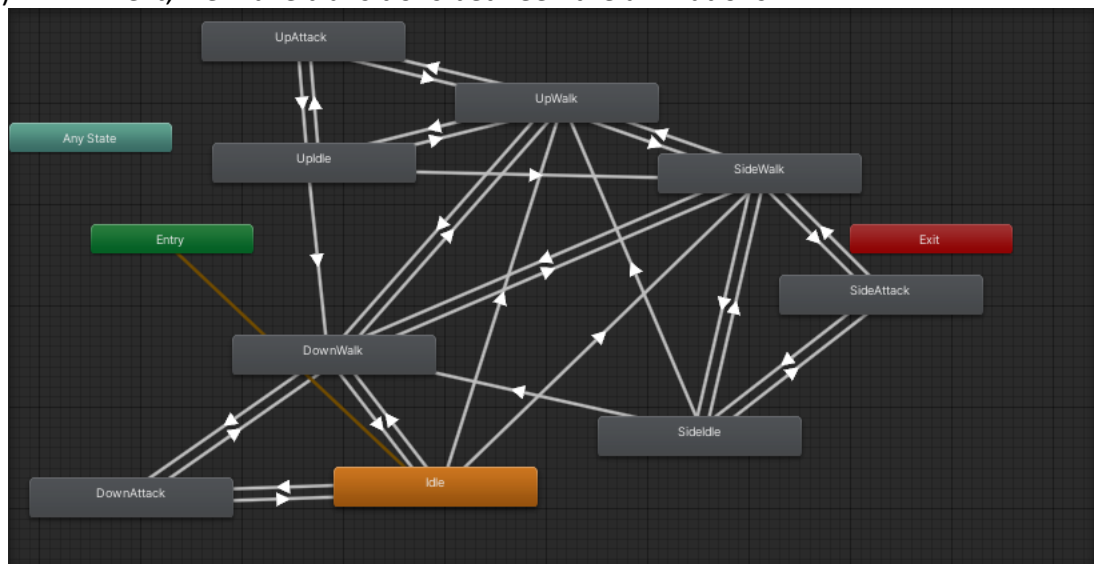


26.) Then we select pertaining to when the character in facing to the side and Idle and we drag it to the “Player” game object. We save it under the “Animations” folder as “SideIdle”.

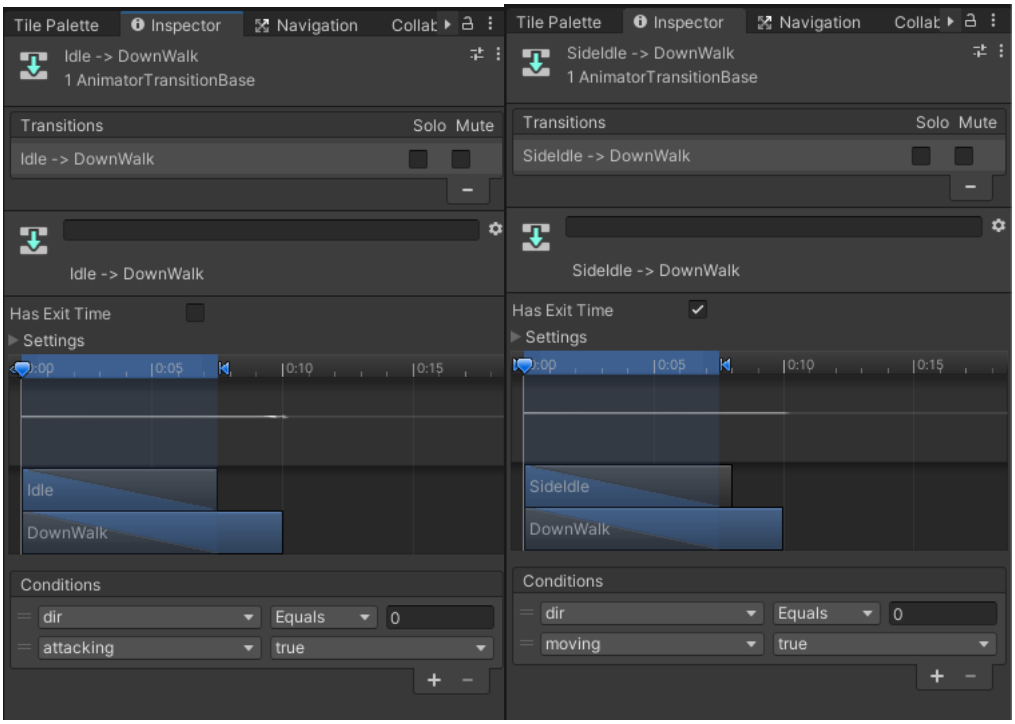
27.) Next we go into the Animator tab and add parameters such as “dir” which is an integer. This integer represents the direction the player is moving. and moving/attacking which are Booleans.



28.) Next, we make transitions between the animations.



29.) Next we click on a transition and some with have “HasExitTime”, and others will exit the animation because of the parameters we set along with “HasExitTime”. Two examples of this can be seen below.



30.) Next we go into “PlayerController.cs” and we create variables. Next, we get a reference to the “RigidBody2D”, “Animator”, and “SpriteRenderer” in the “Awake()” function. In the “Update()” function we read the user input. Then we call the “playerRigidBody” and we add force to it to move the player. Then we initialize “moving” to see if its true or false. Next we add a conditional statement to see what direction the player is moving. Then we add another conditional statement to flip the sprite left or right. We then add another conditional statement to see if the player pressed the space button to attack.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerController : MonoBehaviour
{
    Rigidbody2D playerRigidbody;
    Vector2 input;
    public float speed;

    Animator anim;
    SpriteRenderer rend;
    int lookDir = 0;//0- down 1-left/right 2-up
    bool moving = false;
     private void Awake()
     {
        playerRigidbody = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
        rend = GetComponent<SpriteRenderer>();
```

```
}
// Update is called once per frame
void Update()
{
    input = new Vector2(Input.GetAxis("Horizontal"), Input.GetAxis("Vertical"));

    playerRigidbody.AddForce(input * speed * Time.deltaTime);

    moving = (input.x != 0 || input.y != 0);

    if (input.y > 0)
        lookDir = 2;
    else if (input.y < 0)
        lookDir = 0;

    if (input.x > 0)
    {
        lookDir = 1;
        rend.flipX = false;
    }
    else if (input.x < 0)
    {
        lookDir = 1;
        rend.flipX = true;
    }

    anim.SetInteger("dir", lookDir);
    anim.SetBool("moving", moving);

    if (Input.GetKeyDown(KeyCode.Space))
        SwingAttack();
```

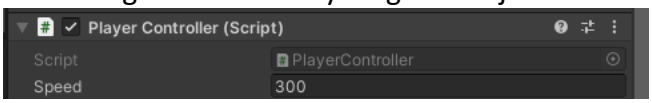
31.) We then create another function “SwingAttack()” to handle the player’s attacks.

```
void SwingAttack()
{
    anim.SetBool("attacking", true);
    Invoke("ResetAttack", 0.1f);

}
```

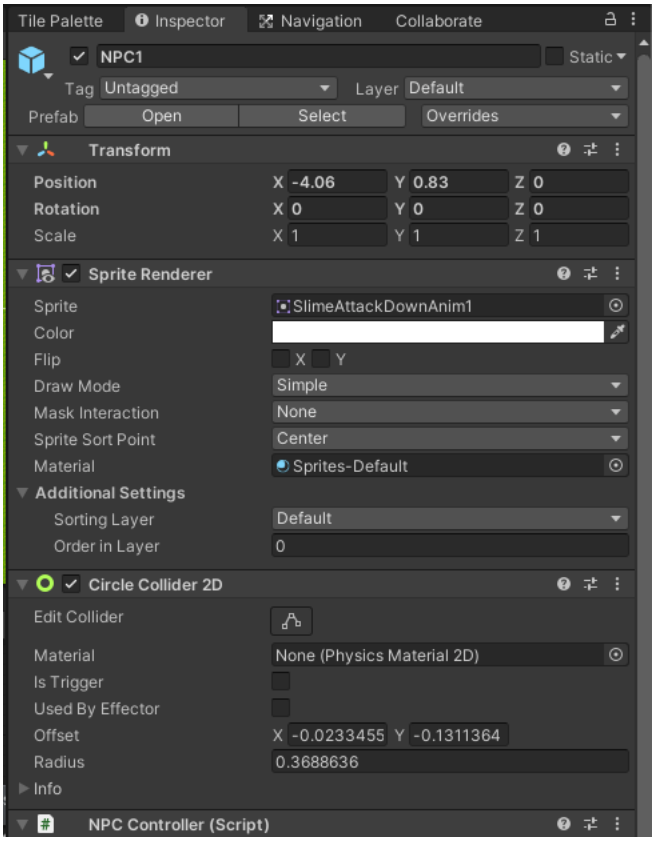
32.) We create another function “ResetAttack()” to reset the attacking boolean.

```
void ResetAttack()
{
    anim.SetBool("attacking", false);
}
```

33.) We then go into the “Player” game object and set the speed.



34.) From the “Sprites” folder we drag “SlimeAttackDownAnim1” to the Hierarchy. We then rename it “NPC1”. Then we attach a circle collider to the game object and adjust the size. Then we create a script file “NPCController” and attach it to “NPC1”.



35.) Then we create a script files “Dialogue.cs” and “DialogueController.cs”. We will attach the “DialogueController.cs” to the canvas, but not the “Dialogue.cs”.

36.) The “Dialogue.cs” only has two attributes, a string, and a array of strings.

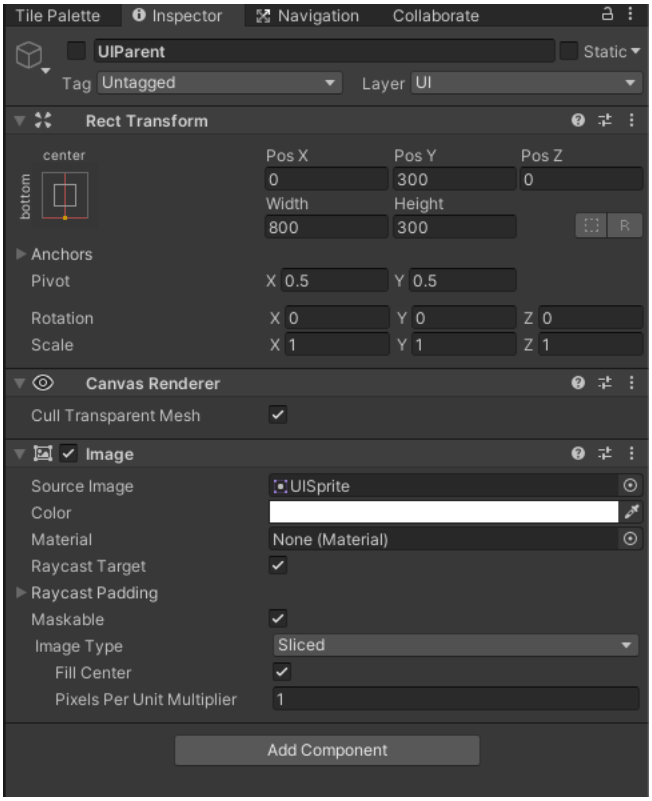
37.) Then we go into “DialogueController.cs” and we define variables. Then we create a function “StartDialogue()” that initiates the dialogue. Next, we create a function to go to the next line of the dialogue in the array of “Dialogues”. We name this function “NextLine()”. We create another function named “ExitDialogue()”, so that if there is no dialogue left in the array we exit out of the UI text.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class DialogueController : MonoBehaviour
{
    Dialogue currentDialogue;
    public GameObject UIParent;
    public Text nameUI;
    public Text dialogueUI;
    int currentIndex;

    public void StartDialogue(Dialogue d)
    {
        currentDialogue = d;
        UIParent.SetActive(true);
        currentIndex = 0;
        nameUI.text = currentDialogue.npcName;
        dialogueUI.text = currentDialogue.dialogue[currentIndex];
    }

    public void NextLine()
    {
        currentIndex++;
        if (currentIndex < currentDialogue.dialogue.Length)
        {
            nameUI.text = currentDialogue.npcName;
            dialogueUI.text = currentDialogue.dialogue[currentIndex];
        }
        else
            ExitDialogue();
    }

    public void ExitDialogue()
    {
        nameUI.text = "";
        dialogueUI.text = "";
        UIParent.SetActive(false);
        currentIndex = 0;
    }
}
```

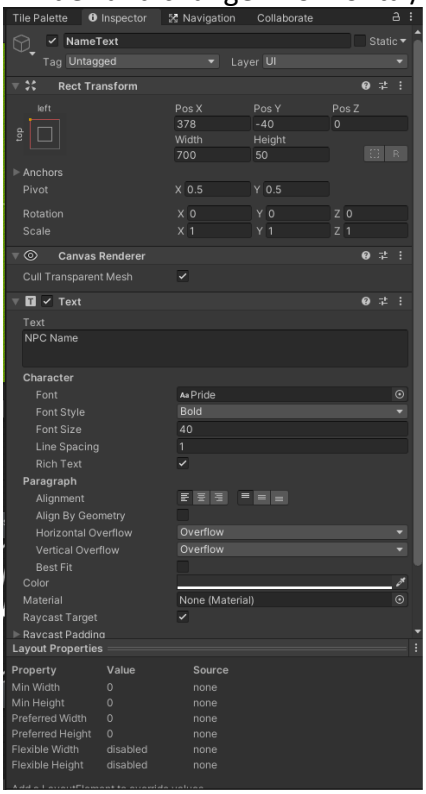
38.) Then we go into “NPCController.cs” we create variables. Then we create two functions. The first is the “Awake()” function and we get a reference to the “DialogueController” and then “OnMouseDown()” if the player clicks on the NPC we start the dialogue.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NPCController : MonoBehaviour
{
    DialogueController dialogueController;
    public Dialogue[] dialogues;
    int currentDia = 0;

    private void Awake()
    {
        dialogueController = FindObjectOfType<DialogueController>();
    }
    public void OnMouseDown()
    {
        dialogueController.StartDialogue(dialogues[currentDia]);
        currentDia = (currentDia + 1) % dialogues.Length;
    }
}
```

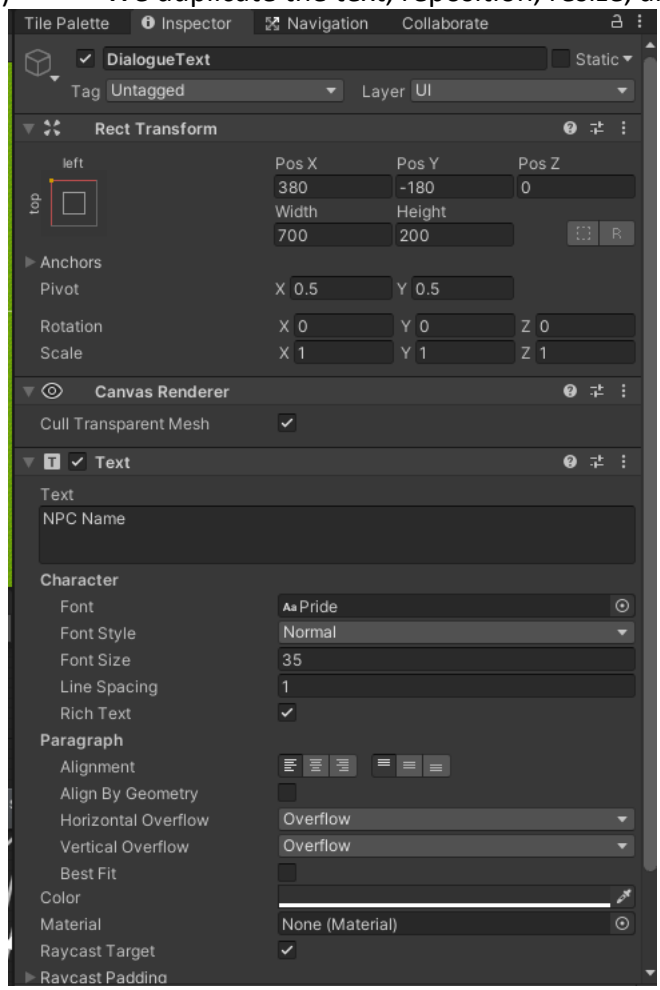
39.) Next, we create the canvas with a child image in the Hierarchy and make the source image the white image from assets. We rename the image “UIParent” and change its position, width, height.



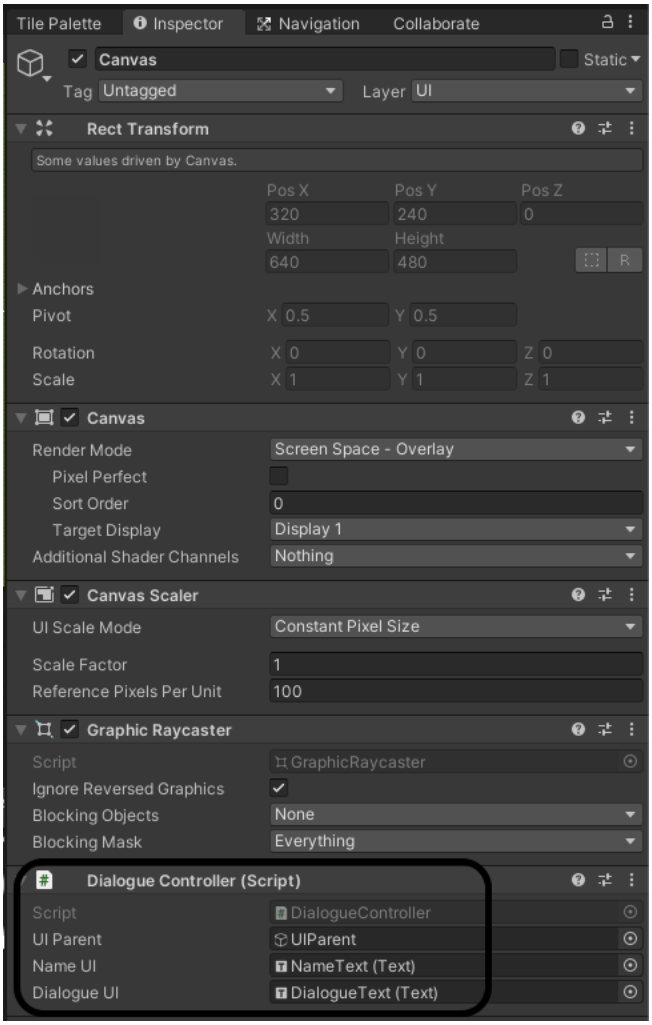
40.) Then we make a UI text under the “UIParent”. We change its position, width, and height. Then we change its font size. This is to show the NPC name. We make the font “Pride” and change “Horizontal/Vertical Overflow” to “Overflow”.



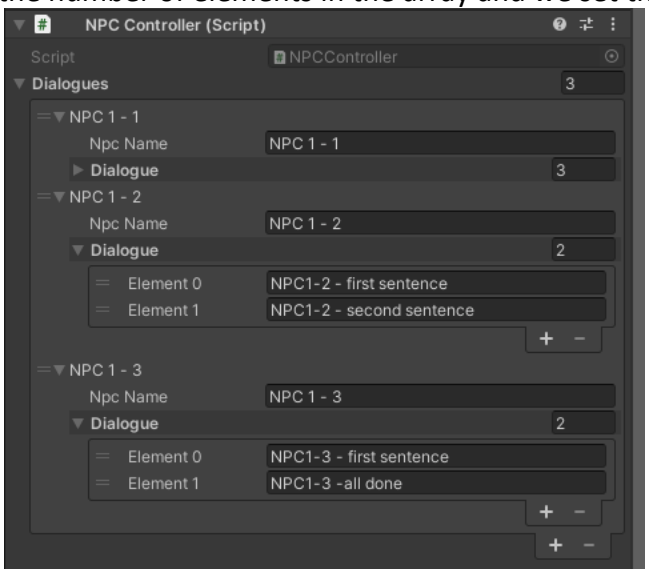
41.) We duplicate the text, reposition, resize, and rename it “DialogueText”.



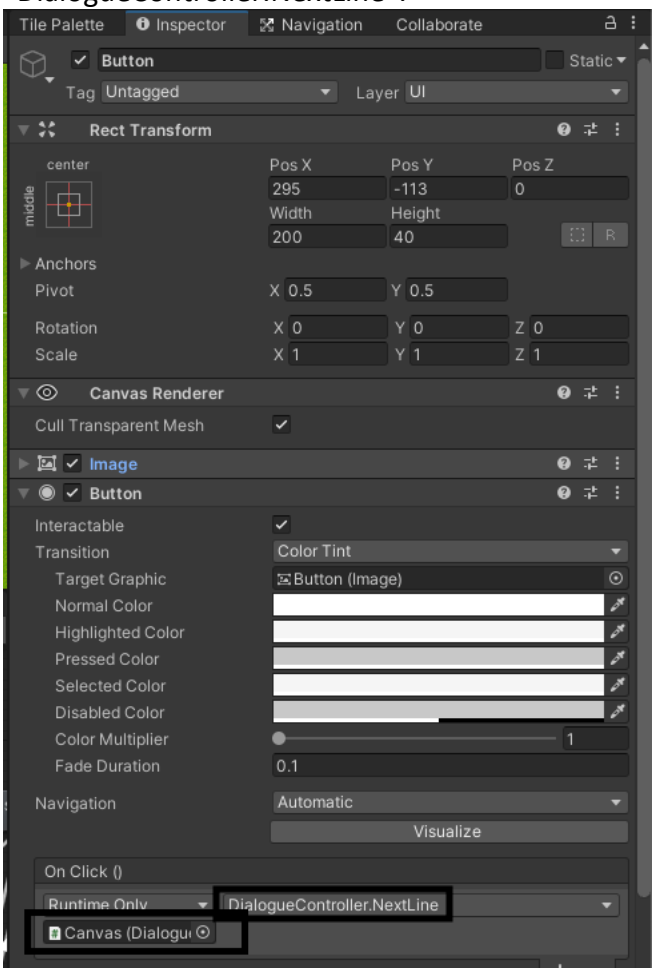
42.) Now under “Canvas” we drag the “UIParent”, “NameText”, and “DialogueText” under the “Dialogue Controller Script”.



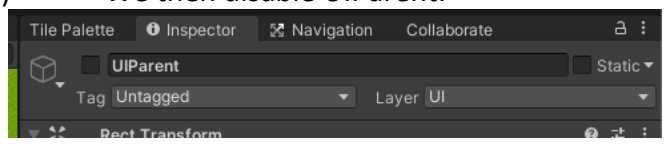
43.) Then we go under the “NPC1” game object and under “NPC Controller” wee set the number of elements in the array and we set the dialogue for the “NPC1”.



44.) Next, we go to GameObject->UI->Button. Then we drag the button under UIParent. Then we set the position and size, and for the text we put “Continue”. Then under “On Click ()” we press the plus sign. Afterwards we lock the inspector and drag canvas under “On Click ()”. Next to “Runtime Only” we select “DialogueController.NextLine”.

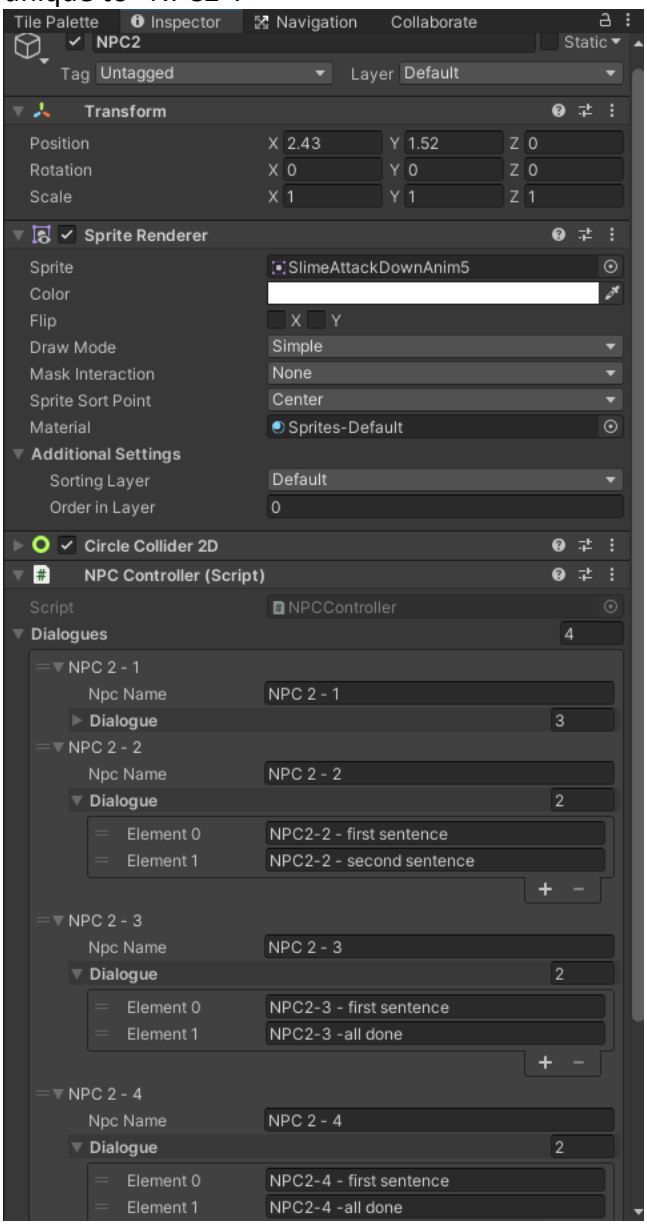


45.) We then disable UIParent.



46.) Next, we duplicate “NPC1” and rename it “NPC2”. We change the sprite to “SlimeAttackDownAnim5”. We reposition it, and we initialize the dialogue so it can be

unique to “NPC2”.



47.) Then we create a folder in “Scripts” and name it “Items”. Under that folder we create the scripts “Consumable.cs”, “Item.cs”, “MeleeWeapon.cs”, “RangedWeapon.cs”, and “Inventory”.

48.) We then go into “Item.cs” and we create variables. Then we create the “Awake()” function that creates a reference to “Inventory”, and “SpriteRenderer”. After, we initialize “itemRend.sprite” to “itemSprite”. Then it creates functions such as “Use()” and “Remove()” which will be used by different items.

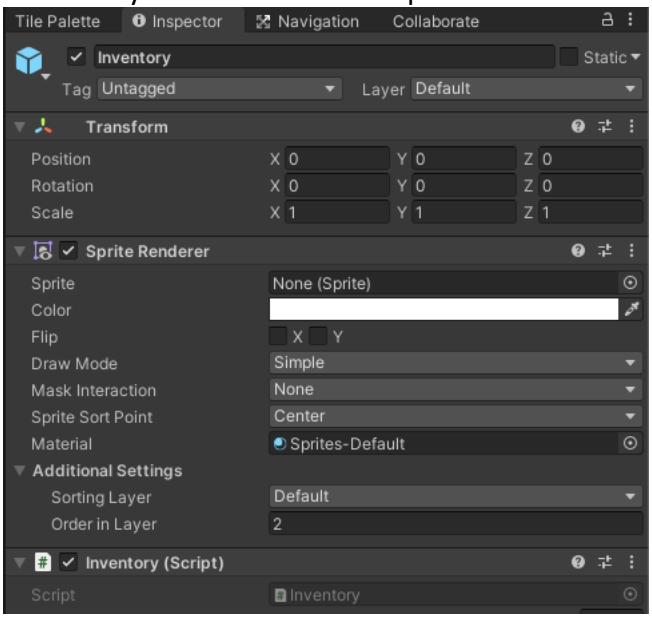
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Item : MonoBehaviour
{
    public string itemName;
    public string itemDescription;
    public int itemCost;
    public bool itemRotate;
    public float amount;//consumable add health, weapon damage amount

    public enum rarity {legendary, rare, uncommon, common };
    public rarity itemRarity;

    public Sprite itemSprite;
    protected Inventory inventory;
    SpriteRenderer itemRend;

    private void Awake()
    {
        inventory = FindObjectOfType<Inventory>();
        itemRend = GetComponent<SpriteRenderer>();
        itemRend.sprite = itemSprite;
    }

    public virtual void Use()
    {
        Debug.Log("base use");
    }

    public virtual void Remove()
    {
        Debug.Log("base remove");
    }

}
```

49.) Then we go into “Consumable.cs” which is a child class of Item. It has an extra attribute “uses”. Then we use the “Use()” function from “Item.cs” and create a conditional statement that if “uses” is greater than zero it subtract the amount of uses. Then it get a reference to the “PlayerController” and uses “heal(amount)”, if anything else happens it removes the item. Then we create the remove function that removes items.

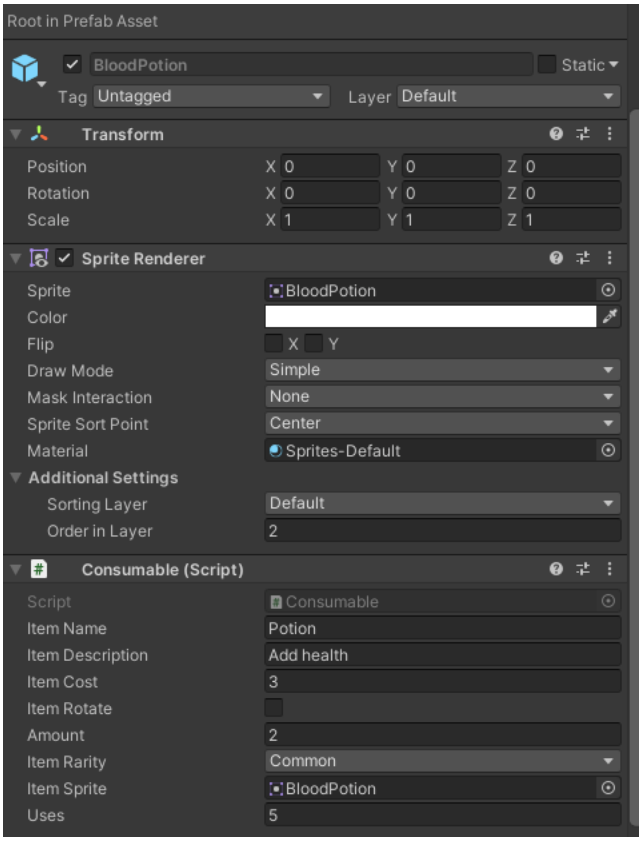
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Consumable : Item
{

    public int uses;

    public override void Use()
    {
        base.Use();
        //Debug.Log("Use Consumable");
        if (uses > 0)
        {
            uses--;
            FindObjectOfType<PlayerController>().Heal(amount);

        }
        else
            Remove();
    }

    public override void Remove()
    {
        base.Remove();
        Debug.Log("Remove Consumable");

    }
}
```

50.) We then go into “MeleeWeapon.cs”, “RangedWeapon” and since both are children of “Item.cs” we implement the “Use()” and “Remove()” functions. The difference is the “RangedWeapon.cs” will have a game object “Projectile”.

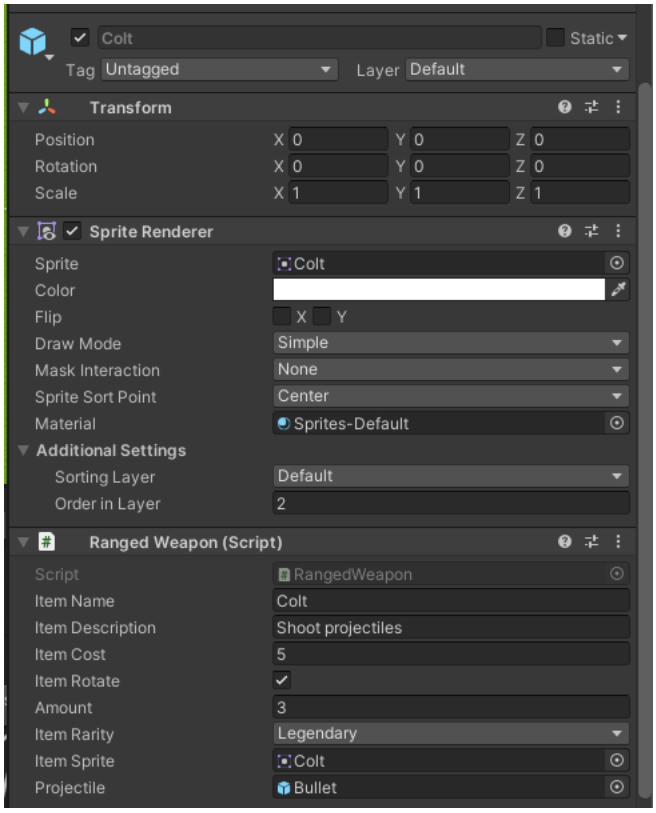
51.) Then we create a child empty game object under “Player” and we name it “Inventory”. Then we attach “SpriteRenderer” and “Inventory.cs” to it.



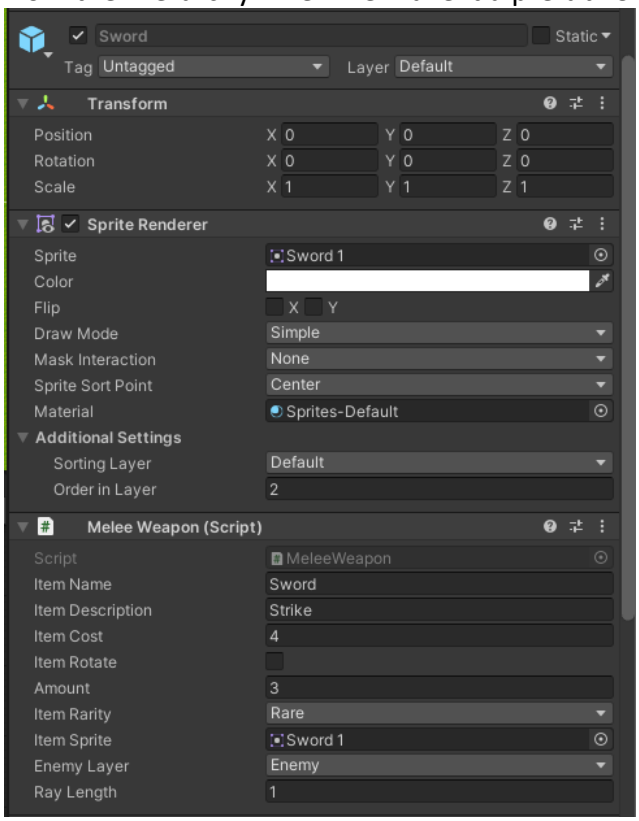
52.) We then drag “BloodPotion” to the Hierarchy and attach the script file “Consumable” to it. Then we initialize all the variables under “Consumable (script)”. Then we make it a prefab. We delete it from the Hierarchy. Change ordering layer to 2.



54.) Next, we drag “Colt” to the Hierarchy and attach the script file “RangeWeapon” to it. Then we initialize all the variables under “Ranged Weapon (script)”. Then we make it a prefab. We delete it from the Hierarchy. Change ordering layer to 2.



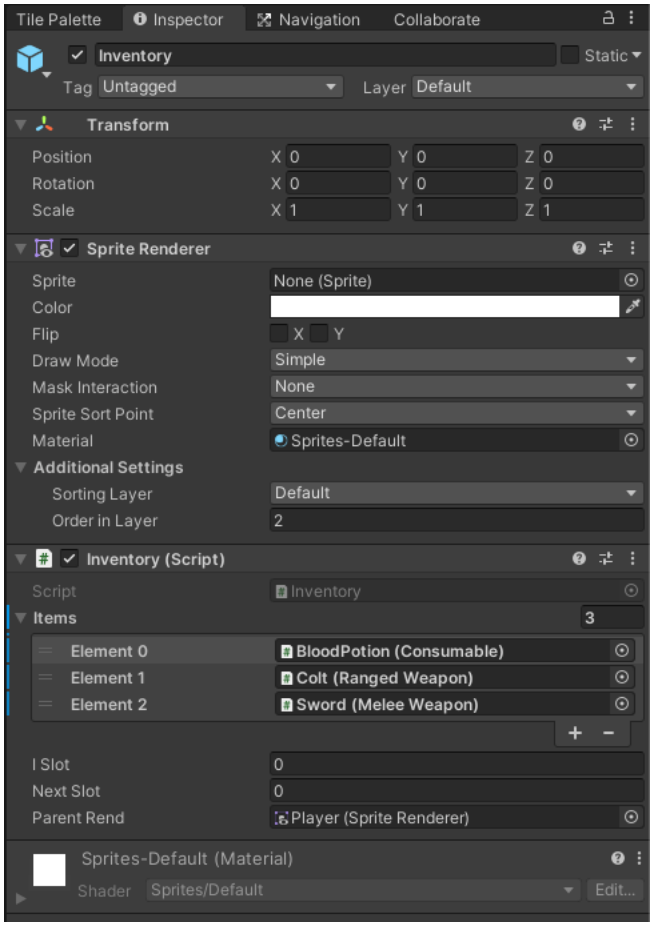
55.) Next, we drag “Sword” to the Hierarchy and attach the script file “MeleeWeapon” to it. Then we initialize all the variables under “Melee Weapon (script)”. We delete it from the Hierarchy. Then we make it a prefab. Change ordering layer to 2.



56.) Next, we go into the “Inventory.cs” script file and create a list of item objects. We also create multiple variable. Then we create the “Awake()” function and get a reference to the “SpriteRenderer”. In the “Update()” funct we check the mouse position to rotate the colt. Then we create the function “AddItem()”, “EquipItem()”, and RemoveItem().

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Inventory : MonoBehaviour
{

    public List<Item> items = new List<Item>();

    [SerializeField]
    int iSlot = 0;// the index of currently equipped item
    [SerializeField]

    int nextSlot = 0; // the index of next equippable item
    SpriteRenderer rend;
    bool rotate = false;

    public SpriteRenderer parentRend;

    private void Awake()
    {
        nextSlot = iSlot;
    }

    // Update is called once per frame
    void Update()
    {
        if (rotate)
        {
            Vector3 dir = Input.mousePosition -
Camera.main.WorldToScreenPoint(transform.position);
            float angle = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg;
            transform.rotation = Quaternion.Lerp(transform.rotation,
Quaternion.AngleAxis(angle, Vector3.forward), Time.deltaTime * 10f);
        }
    public void AddItem(Item item)
    {
    }

    public void EquipItem(int slot)
    {
    }

    public void RemoveItem(Item item)
```

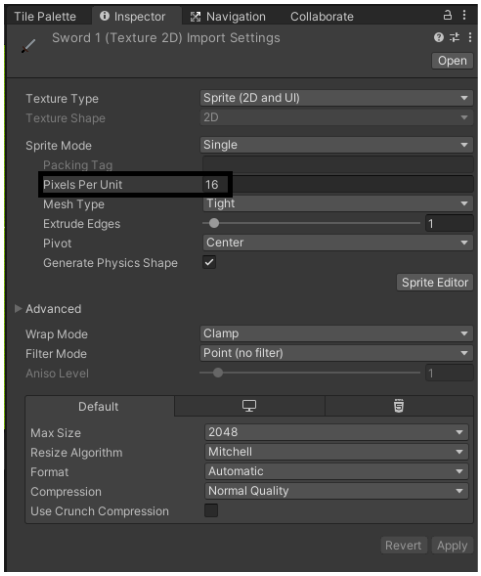
57.)Then in the Inventory game object we put the three prefabs potion, colt, and sword in the Inventory script variables.



58.) Next we check if the player presses z and if so we call “EquipItem()”.

```
if (Input.GetKeyDown(KeyCode.Z))// press z to equip the next item
{
  if (items.Count != 0)
    EquipItem(nextSlot);
  else
    Debug.Log("No item i in Inventory");
}
```

59.) Sword it too small so we set the “Pixels Per Unit” for the sword sprite to 16.



60.) Next, we go into “Item.cs” and add more variables. Then we create the “Awake()” function and create a reference to the “Inventory”. We also get a reference to the “SpriteRenderer”. In “Use()” and “Remove()” we use debug lines to output messages in the console.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Item : MonoBehaviour
{
    public string itemName;
    public string itemDescription;
    public int itemCost;
    public bool itemRotate;
    public float amount;//consumable add health, weapon damage amount

    public enum rarity {legendary, rare, uncommon, common };
    public rarity itemRarity;

    public Sprite itemSprite;
    protected Inventory inventory;
    SpriteRenderer itemRend;

    private void Awake()
    {
        inventory = FindObjectOfType<Inventory>();
        itemRend = GetComponent<SpriteRenderer>();
        itemRend.sprite = itemSprite;
    }

    public virtual void Use()
    {
        Debug.Log("base use");
    }

    public virtual void Remove()
    {
        Debug.Log("base remove");
    }

}
```

61.) Afterwards, we go into “Inventory.cs” and we create more variables. In the “Awake()” function we call “AddItem()” and for the parameter we use the variables that were just created. Then, we set “nextSlot” equal to “iSlot”. Then in the “Update()” function we add a conditional statement that if there are items left equip the next item, else print “No item in Inventory”. Then we add if the player presses “x” it the player will equip the item, and if they press “c” the player will remove equipped item.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Inventory : MonoBehaviour
{

    public List<Item> items = new List<Item>();

    [SerializeField]
    int iSlot = 0;// the index of currently equipped item
    [SerializeField]

    int nextSlot = 0; // the index of next equippable item
    SpriteRenderer rend;
    bool rotate = false;

    public SpriteRenderer parentRend;

    private void Awake()
    {
        nextSlot = iSlot;
    }

    // Update is called once per frame
    void Update()
    {
        if (rotate)
        {
            Vector3 dir = Input.mousePosition - Camera.main.WorldToScreenPoint(transform.position);
            float angle = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg;
            transform.rotation = Quaternion.Lerp(transform.rotation, Quaternion.AngleAxis(angle, Vector3.forward), Time.deltaTime * 10f);
        }

        transform.localScale = new Vector3(parentRend.flipX? -1:1, 1, 1);

        if (Input.GetKeyDown(KeyCode.Z))// press z to equip the next item
        {
            if (items.Count != 0)
                EquipItem(nextSlot);
            else
                Debug.Log("No item i in Inventory");
        }

        if (Input.GetKeyDown(KeyCode.X))// press x to equipped the next item items[iSlot]
        {
            if(items.Count != 0 && items[iSlot] != null)
                items[iSlot].Use();
        }
        if (items.Count != 0 && Input.GetKeyDown(KeyCode.C))// press c to remove the equipped item
        {
            if(items[iSlot] != null)
                RemoveItem(items[iSlot]);
        }
```

62.) Then in “AddItem(Item item)” we add items into the inventory. We set the position and location of this item as well.

```
    public void AddItem(Item item)
    {
        Item newItem = Instantiate(item);
        newItem.transform.SetParent(transform);
        newItem.transform.localPosition = Vector3.zero;
        newItem.transform.localRotation = Quaternion.identity;
        items.Add(newItem);
        newItem.gameObject.SetActive(false);
    }
```

63.) In “EquipItem()” before we equip the next item we set the current item as invisible. Then we set the rotation if the item is able to rotate. Thenn we update the next slot.

```
    public void EquipItem(int slot)
    {
        if (items.Count != 0)
        {

            items[iSlot % items.Count].gameObject.SetActive(false);
            iSlot = slot % items.Count;
            items[iSlot].gameObject.SetActive(true);
            transform.rotation = Quaternion.Euler(0, 0, 0);
            rotate = items[iSlot].itemRotate;
            nextSlot = (iSlot + 1) % items.Count;
        }
    }
```

64.) Next, we go into “Consumable.cs” and implement the use function. We add a conditional statement that says if uses are greater than zero, we decrement uses. We then get a reference to “PlayerController” and use the heal function from player controller. Else, we call remove.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Consumable : Item
{

  public int uses;

  public override void Use()
  {
    base.Use();
    //Debug.Log("Use Consumable");
    if (uses > 0)
    {
      uses--;
      FindObjectOfType<PlayerController>().Heal(amount);

    }
    else
      Remove();
  }
```

65.) We then implement “Heal()” and “Damage()” in the “PlayerController.cs”.

```
  public void Heal(float amt)
  {
    health += amt;
    if (health > maxHealth) health = maxHealth;
  }

  public void Damage(float amt)
  {
    health -= amt;
    if (health <= 0) Die();
  }
```

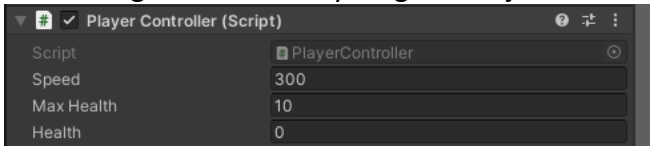
66.) We also implement the “Die()” function where we set the player invisible and pause the game.

```
  void Die()
  {
    gameObject.SetActive(false);
    Time.timeScale = 0f;
  }
```

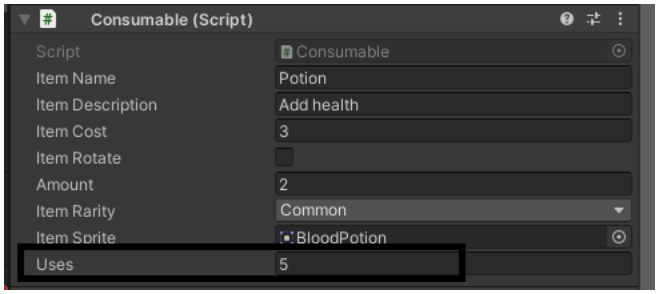
67.) In the “Awake()” function we set “health” equal to “maxHealth”.

```
  private void Awake()
  {
    playerRigidbody = GetComponent<Rigidbody2D>();
    anim = GetComponent<Animator>();
    rend = GetComponent<SpriteRenderer>();
    health = maxHealth;
```

68.) We then go into the “Player” game object and set max health.



69.) Next, we go into the “BloodPotion” prefab and set the uses amount.

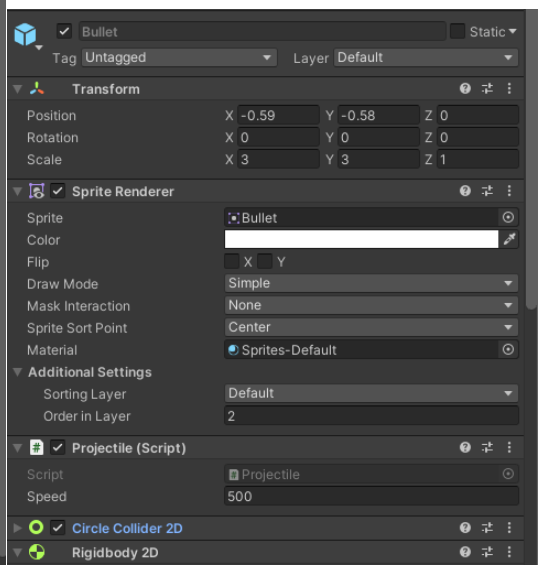


70.) Next, in “RangedWeapon()” we create a projectile and instantiate it.

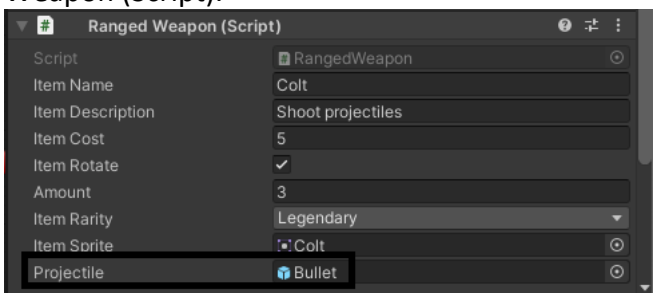
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RangedWeapon : Item
{

   public GameObject Projectile;

      public override void Use()
   {
      base.Use();
      //Debug.Log("Use RangedWeapon");
      Instantiate(Projectile, transform.position, transform.rotation * Quaternion.Euler(0,
0, -90));
   }

   public override void Remove()
   {
      base.Remove();
      Debug.Log("Remove RangedWeapon");

   }
}
```

71.) Then we go into our tower defense game and take the bullet sprite and put it into our current game. Next, we take the “BulletController.cs” and paste it over to our “Projectile.cs”. We then put the bullet sprite into the hierarchy, and attach “Projectile.cs”, “Circle Collider 2D”, and “RigidBody2D” to it. We then make the Bullet into a prefab. We set ordering layer as 2.

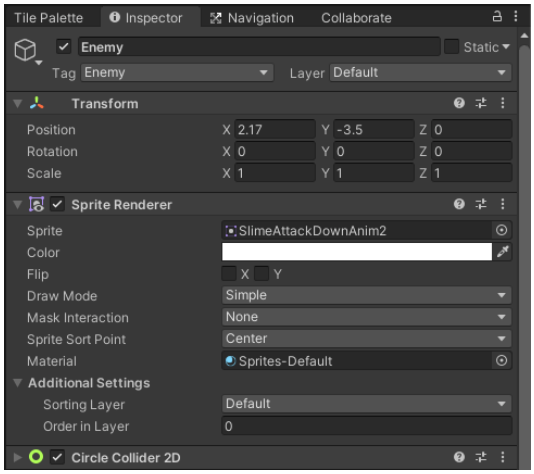
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Projectile : MonoBehaviour
{
    public float speed;
    Rigidbody2D bulletRigidbody;

    RangedWeapon colt;

    void Awake()
    {
        bulletRigidbody = GetComponent<Rigidbody2D>();
        colt = FindObjectOfType<RangedWeapon>();
    }

    private void OnEnable()
    {
        bulletRigidbody.AddForce(transform.up * speed);
        Invoke("Disable", 4f);
    }

    private void Disable()
    {
        gameObject.SetActive(false);
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Enemy"))
        {
            //Debug.Log("hit an enemy");
            collision.GetComponent<EnemyController>().Damage(colt.amount);
            Invoke("Disable", 0.01f);
        }
    }

    private void OnDisable()
    {
        CancelInvoke();
    }
}
```



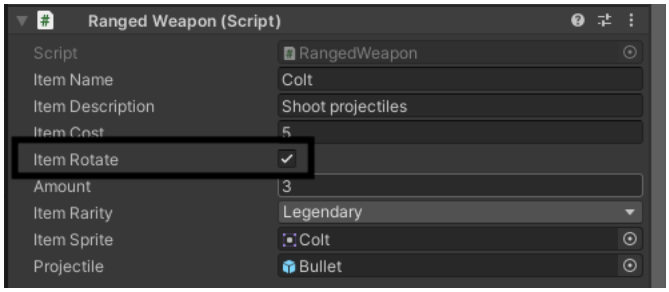
72.) Then we go into the “Colt” prefab and drag our bullet to “Projectile” under “Ranged Weapon (Script).



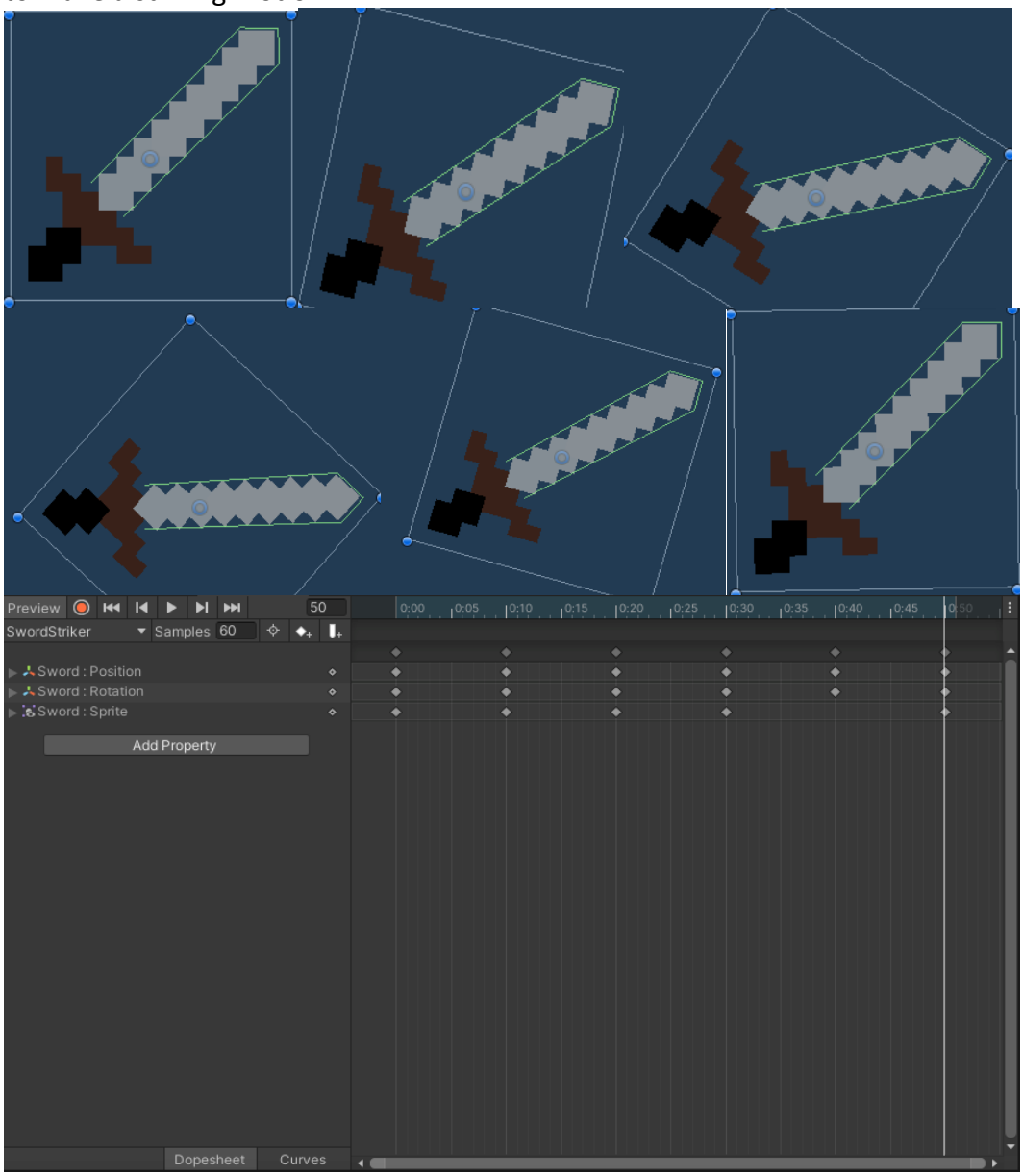
73.) Next, we add “SlimeAttackAnim2” and rename it to “Enemy”. We then create and add the “Enemy” tag to it. We add a circle collider to it as well.



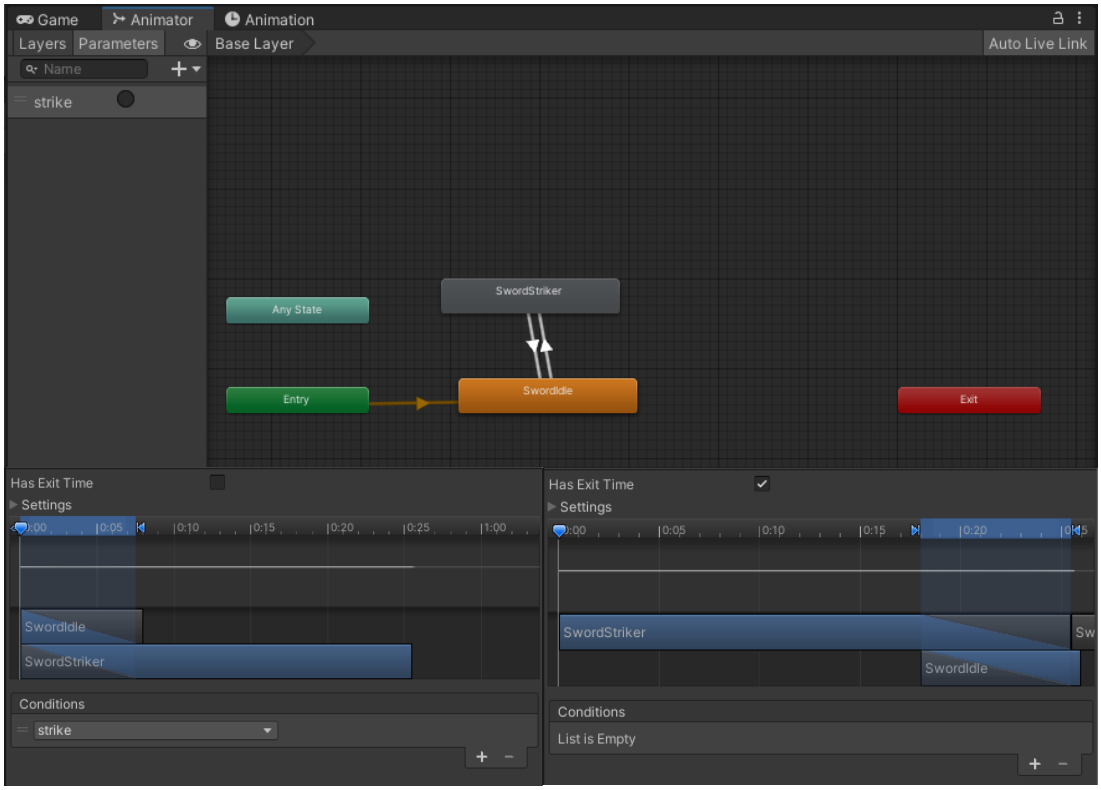
74.) Then we go into the prefab “Colt” and under “Ranged Weapon (Script)” we check “Item Rotate”.



75.) Next, we press on the sword prefab and in the animation tab, and we press “Create”. We save this int the “Animations” folder as “SwordStriker”. Then we press the record button and add key frames. At each key frame we reposition the sword to make a striking motion.



76.) Next, we go into “Animator and make transitions between “SwordStriker” and “SwordIdle”. “SwordIdle” will be the default state. “SwordStriker” -> “SwordIdle” will have “Has Exit Time” and “SwordIdle” -> “SwordStriker” will not have “Has Exit Time”.



77.) We go into “MeleeWeapon.cs” and declare an “Animator” variable. Then in the function we get the reference to the “Animator”, and use the function “SetTrigger” with the parameter “strike”.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MeleeWeapon : Item
{
    Animator anim;
    //public LayerMask enemyLayer;
    //public float rayLength;

    public override void Use()
    {
        base.Use();
        Debug.Log("Use MeleeWeapon");
        FindObjectOfType<Animator>().SetTrigger("strike");
```

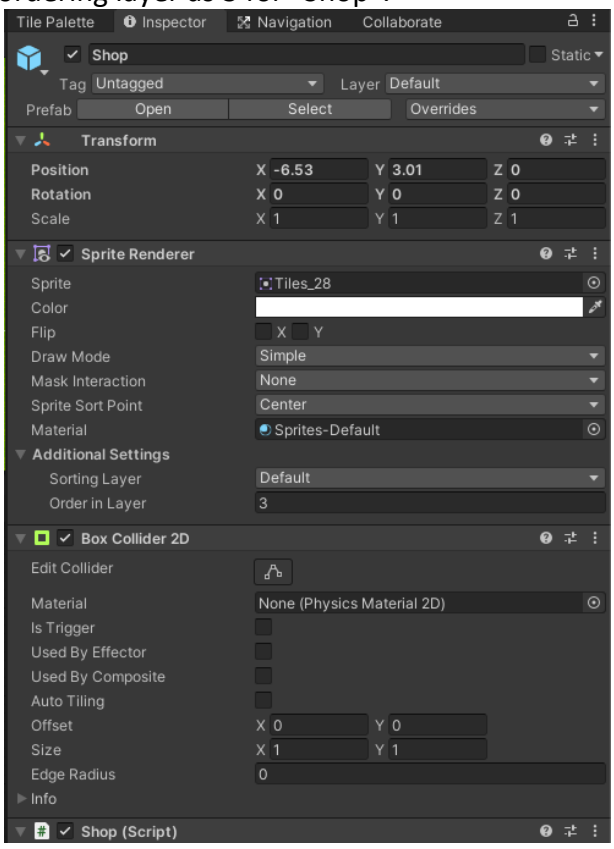
78.) Next, we go into “Inventory.cs”, add variables and in the “Update()” function we add functionality for the sword sprite to flip in the game.

```
public SpriteRenderer parentRend;

// Update is called once per frame
void Update()
{
    if (rotate)
    {
        Vector3 dir = Input.mousePosition -
Camera.main.WorldToScreenPoint(transform.position);
        float angle = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg;
        transform.rotation = Quaternion.Lerp(transform.rotation,
Quaternion.AngleAxis(angle, Vector3.forward), Time.deltaTime * 10f);
    }

    transform.localScale = new Vector3(parentRend.flipX? -1:1, 1, 1);
```

79.) We go into “PlayerController.cs” and add variables related to money. We go to the “Awake()” function and we set “money” equal to “maxMoney”. Then we create the “AddMoney(int amt)” function.

```
public int maxMoney;
[SerializeField]
public int money;

private void Awake()
{
    playerRigidbody = GetComponent<Rigidbody2D>();
    anim = GetComponent<Animator>();
    rend = GetComponent<SpriteRenderer>();
    health = maxHealth;
    money = maxMoney;

public void AddMoney(int amt)
{
    money += amt;
}
```

80.) Next, we go back to Unity and grab “Tile_28” from the “Sprites” folder and drag it to the Hierarchy. We rename it “Shop”. We create a script file “Shop.cs”. We attach a “Box Circle Collider” and “Shop.cs” to the “Shop” game object. We set the ordering layer as 3 for “Shop”.

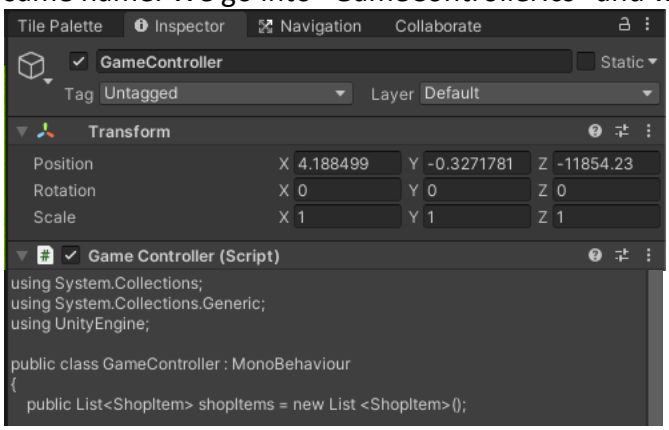


81.) Under “Shop” we attach child game object and name it “ShopParent”.

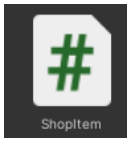
82.) We go into “Shop.cs” and add variables.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Shop : MonoBehaviour
{

  public float interactDistance;
  float distance;
  public GameObject shopParent;
  PlayerController player;
```

83.) We go into Unity and add “GameController” game object into the Hierarchy. Then we create script file “GameController.cs” and attach it to the game object with the same name. We go into “GameController.cs” and we create a list of “ShopItems”.



84.) We then create “ShopItem.cs”



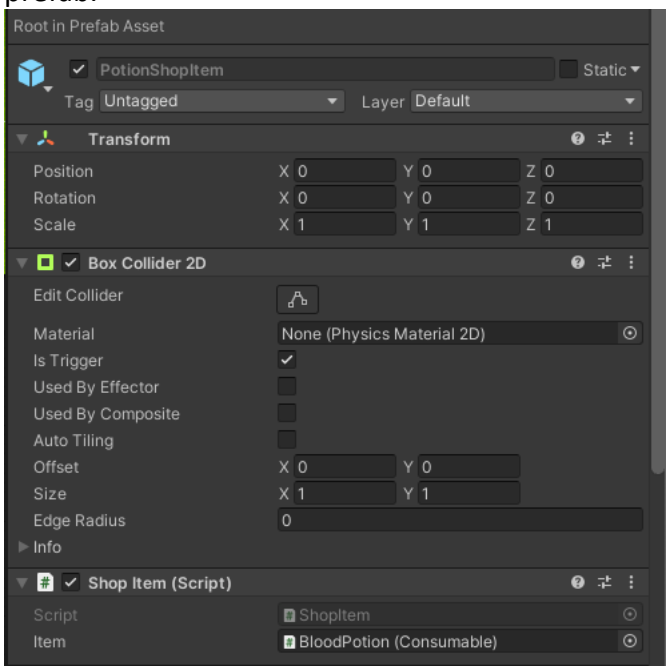
85.) We go back to “Shop.cs” and we add a “GameController” variable, and in “Awake()” we get a reference to “PlayerController” and “GameController”.We also call “PopulateShop(). We go to update and initialize distance, then we add a conditional statement that controls the interaction distance for a player. Then we create the function “PopulateShop()” that will fill the shop with items.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Shop : MonoBehaviour
{

    public float interactDistance;
    float distance;
    public GameObject shopParent;
    PlayerController player;
    GameController cont;


    void Awake()
    {
        player = FindObjectOfType<PlayerController>();
        cont = FindObjectOfType<GameController>();
        PopulateShop();
    }

    // Update is called once per frame
    void Update()
    {
        distance = Vector2.Distance(transform.position, player.transform.position);

        if (distance <= interactDistance)
            shopParent.SetActive(true);
        else
            shopParent.SetActive(false);
    }

    public void PopulateShop()
    {
        ShopItem shopItem;
        for (int i = 0; i < 3; i++)
        {
            shopItem = Instantiate(cont.GetRandomItem(cont.shopItems));
            shopItem.transform.SetParent(shopParent.transform);
            shopItem.transform.localPosition = new Vector3((i * 1.5f) - 1.5f, 0, 0);
        }
    }
}
```

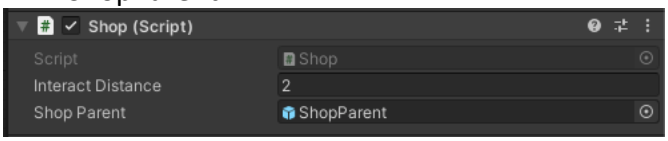
86.) Then we go into “ShopItem.cs” and add variables. We then go into the “Awake()” function and get a reference to the “PlayerController”, “Inventory”, and “SpriteRenderer”. We set “rend.sprite” equal to “item.itemSprite”. Then we create the function “BuyItem” to allow the player to buy the items in the shop. Then we create the function “OnMouseDown()” that when a player click on an item in the shop they will buy the item.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ShopItem : MonoBehaviour
{

    public Item item;

    PlayerController player;
    Inventory inventory;
    SpriteRenderer rend;
    Text itemText;

    void Awake()
    {
        player = FindObjectOfType<PlayerController>();
        inventory = FindObjectOfType<Inventory>();
        rend = GetComponent<SpriteRenderer>();
        itemText = GetComponentInChildren<Text>();
        rend.sprite = item.itemSprite;
    }


    public void BuyItem()
    {
        if (player.money >= item.itemCost)
        {
            player.AddMoney(-item.itemCost);
            inventory.AddItem(item);
            Destroy(gameObject);
        }
    }

    private void OnMouseDown()
    {
        BuyItem();
    }
```

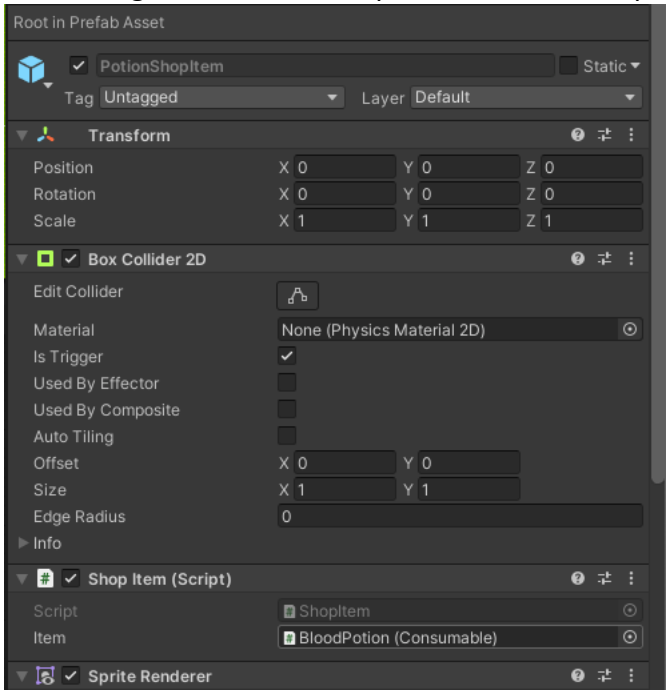
87.) We go into the Hierarchy and create the game object “PotionShopItem” and attach the “Box Collider 2D”, and the “ShopItem.cs” script file. Then we make it into the prefab.



88.) We then go into the “Shop” game object and set “Interact Distance” and “ShopParent”.

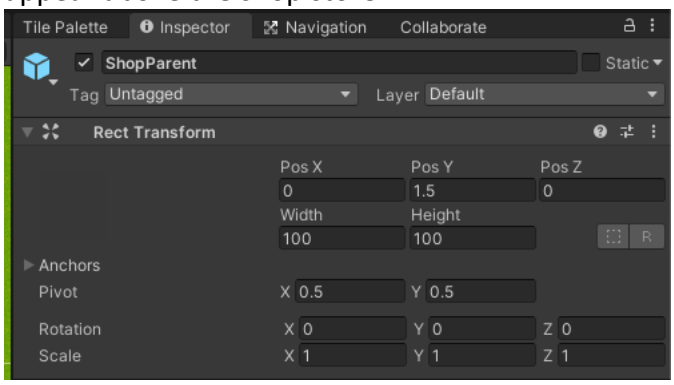


89.) We then go into “PotionShopItem and attach “SpriteRenderer” to it.



90.) We go into the “Player” game object and set the money and maxMoney.

91.) We then go into “ShopParent” and change the position of Y to make the item appear above the shop store.



92.) We then add a game object “Canvas” and under that a game object “Text” to the Hierarchy. We change the “RenderMode” for “Canvas” to “Word Space”. We also set the position. We then put the “Canvas” under the “PotionShopItem”.

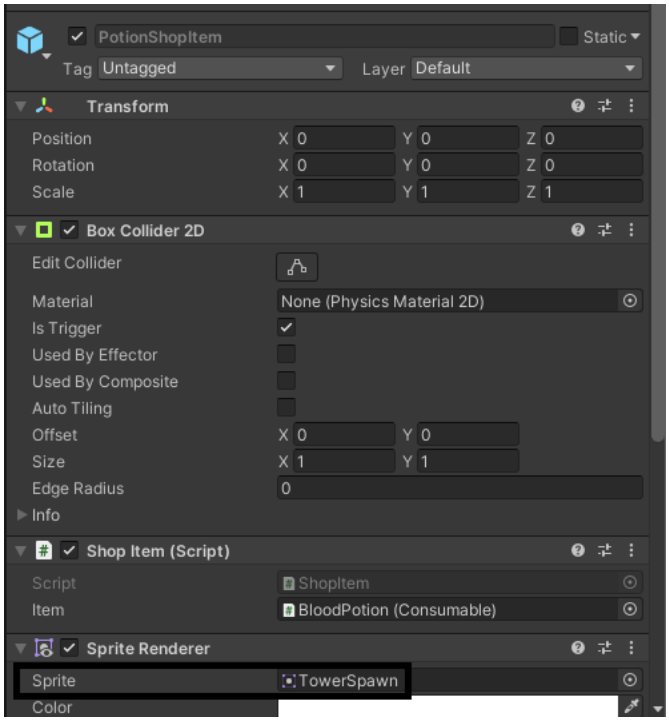
93.) We then go into “ShopItem.cs” and add variables related to text. We then go into “Awake()” and add a reference for the item text.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ShopItem : MonoBehaviour
{

    public Item item;

    PlayerController player;
    Inventory inventory;
    SpriteRenderer rend;
    Text itemText;

    void Awake()
    {
        player = FindObjectOfType<PlayerController>();
        inventory = FindObjectOfType<Inventory>();
        rend = GetComponent<SpriteRenderer>();
        itemText = GetComponentInChildren<Text>();
        rend.sprite = item.itemSprite;
    }
```

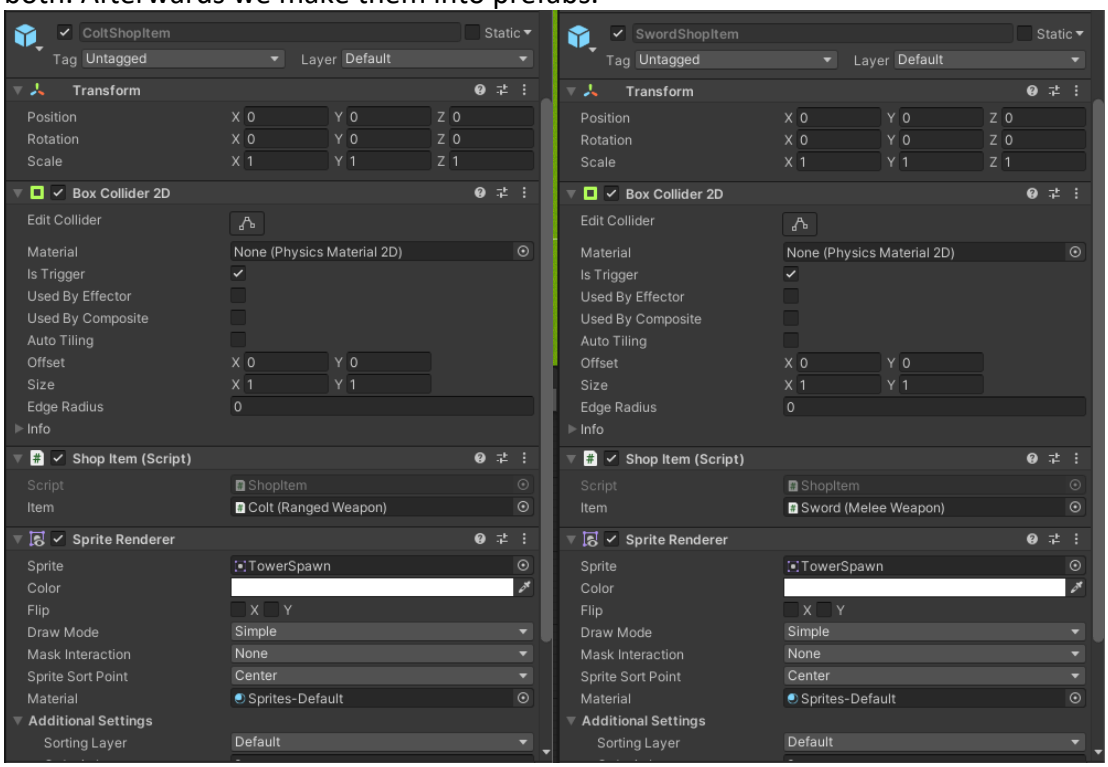
94.) Then in the “Update()” function we set “itemText”.

```
    void Update()
    {
        itemText.text = item.itemName + "\n" + item.itemCost;
        itemText.color = player.money > item.itemCost ? Color.green : Color.red;
    }
```

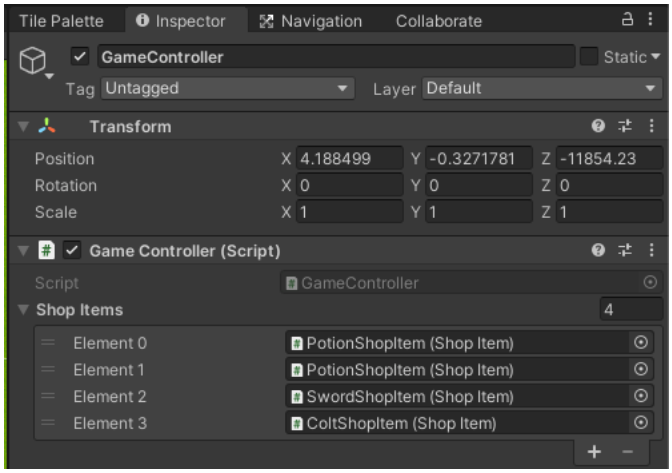
95.) We go into the “PotionShopItem” prefab and set the “TowerSpawn.png” as the “Sprite” in “Sprite Renderer”.



96.) We then unpack the prefab in the Hierarchy and adjust them for the other two items in the game. “Colt” and “Sword”. We do this by changing the script file for both. Afterwards we make them into prefabs.



97.) Then we go into the “GameController” controller game object and insert the “Colt/SwordShopItem” into the “Shop Items” list under “Game Controller (Script)”.



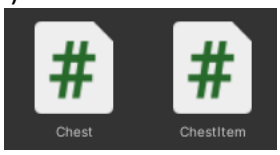
98.) We go into “GameController.cs” and we create a function “GetRandomItem(List<ShopItem> l). This will return a random item from the shop.

```
public ShopItem GetRandomItem(List<ShopItem> l)
{
    int index = Random.Range(0, l.Count);
    ShopItem item = l[index];
    return item;
}
```

99.) We then call this function in “Shop.cs”

```
public void PopulateShop()
{
    ShopItem shopItem;
    for (int i = 0; i < 3; i++)
    {
        shopItem = Instantiate(cont.GetRandomItem(cont.shopItems));
```

100.) We create two Script files “Chest.cs” and “ChestItem.cs”.



101.) Then in “Item.cs” we define functions that are related to the rarity of an item.

```
public enum rarity {legendary, rare, uncommon, common };
public rarity itemRarity;
```

102.) Then in “GameController.cs” we define a list related to a list of items in a chest. In the “Awake()”. function it will check the rarity of each item and then place it in the corresponding sublist.

```
public List<ChestItem> chestItems = new List <ChestItem>();

public List<ChestItem> legendaryItems = new List <ChestItem>();
public List<ChestItem> rareItems = new List <ChestItem>();
public List<ChestItem> uncommonItems = new List <ChestItem>();
public List<ChestItem> commonItems = new List <ChestItem>();
```

103.) In “Chest.cs” we copy and paste what was in “Shop.cs”. However, we delete anything that has to do with distance. We also change “PopulateShop()” to “PopulateChest()”. “PopulateChest()” will generate a random number and that will determine the rarity of item that will pop up in the chest. We also add the function “OnCollisionEnter2D()” we call “PopulateChest()”. We add a Boolean to check if the chest is populated. If not in the “OnCollisionEnter2D()” function we call “PopulateChest()”.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Chest : MonoBehaviour
{
    public GameObject chestParent;
    PlayerController player;
    GameController cont;
    bool populated = false;


    void Awake()
    {
        player = FindObjectOfType<PlayerController>();
        cont = FindObjectOfType<GameController>();
        populated = false;

    }

    // Update is called once per frame
    void Update()
    {

    }

    public void PopulateChest()
    {
        ChestItem chestItem;
        for (int i = 0; i < 3; i++)
        {
            int r = Random.Range(0, 100);
            Debug.Log("random number is " + r);
            if (r < 3 && cont.legendaryItems.Count != 0)//legendary
            {
                chestItem = Instantiate(cont.GetRandomItem(cont.legendaryItems));
                chestItem.transform.SetParent(chestParent.transform);
                chestItem.transform.localPosition = new Vector3((i * 1.5f) - 1.5f, 0, 0);
            }
            else if (r < 10 && cont.rareItems.Count != 0)//rare
            {
                chestItem = Instantiate(cont.GetRandomItem(cont.rareItems));
                chestItem.transform.SetParent(chestParent.transform);
                chestItem.transform.localPosition = new Vector3((i * 1.5f) - 1.5f, 0, 0);
            }
            else if (r < 50 && cont.uncommonItems.Count != 0)//uncommon
            {
                chestItem = Instantiate(cont.GetRandomItem(cont.uncommonItems));
                chestItem.transform.SetParent(chestParent.transform);
                chestItem.transform.localPosition = new Vector3((i * 1.5f) - 1.5f, 0, 0);
            }
            else if (cont.legendaryItems.Count != 0) //common
            {
                chestItem = Instantiate(cont.GetRandomItem(cont.commonItems));
                chestItem.transform.SetParent(chestParent.transform);
                chestItem.transform.localPosition = new Vector3((i * 1.5f) - 1.5f, 0, 0);
            }
            else
            {
                Debug.Log("No item in Chest");
            }
        }

    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (!populated && collision.gameObject.CompareTag("Player"))
        {
            PopulateChest();
            populated = true;
        }

    }

}
```

104.) Then we copy the “ShopItem.cs” script file and paste it into “ChestItem.cs” script file. We add “using UnityEngine.UI”. We change “BuyItem()” to “PickupItem()”. The player does not spend money to get a chest item.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ChestItem : MonoBehaviour
{

    public Item item;

    PlayerController player;
    Inventory inventory;
    SpriteRenderer rend;
    Text itemText;

    void Awake()
    {
        player = FindObjectOfType<PlayerController>();
        inventory = FindObjectOfType<Inventory>();
        rend = GetComponent<SpriteRenderer>();
        itemText = GetComponentInChildren<Text>();
        rend.sprite = item.itemSprite;
    }


    public void PickUpItem()
    {
        inventory.AddItem(item);
        Destroy(gameObject);
    }

    private void OnMouseDown()
    {
        PickUpItem();
    }
    // Update is called once per frame
    void Update()
    {
        itemText.text = item.itemName;

    }
}
```

105.) In the “GameController.cs” script file we create another function “GetRandomItem” but we get it from “List<ChestItem> l”.

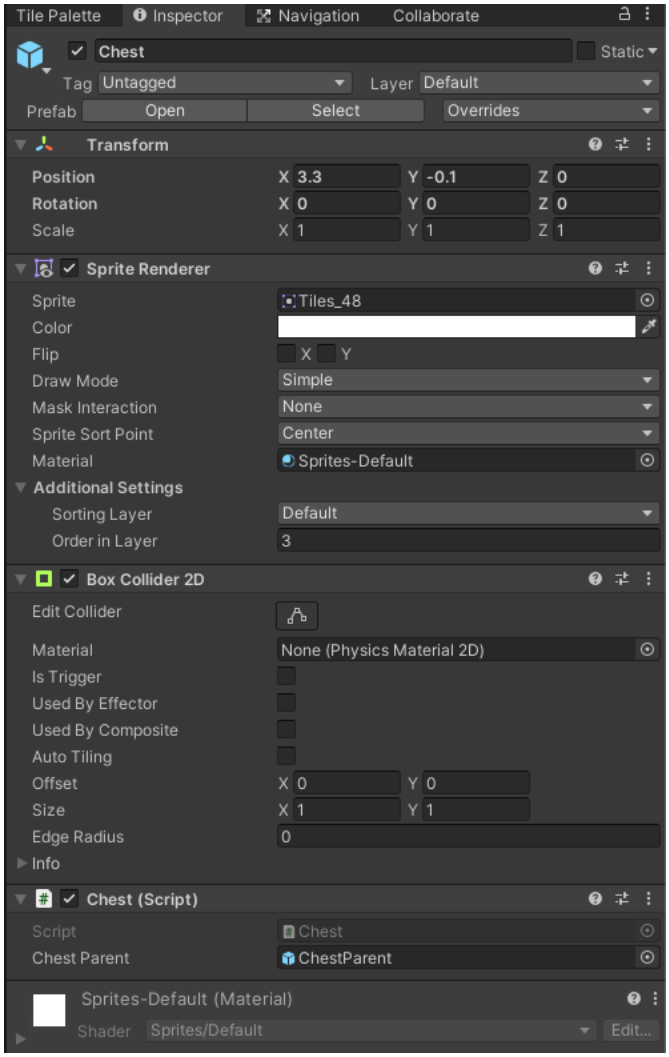
```
public ChestItem GetRandomItem(List<ChestItem> l)
{
    int index = Random.Range(0, l.Count);
    ChestItem item = l[index];
    return item;
}
```

106.) Then we go into “GameController.cs” and in the “Awake()” function we check every item in the chest item list. Then we use a switch statement to add items to the corresponding sublist.

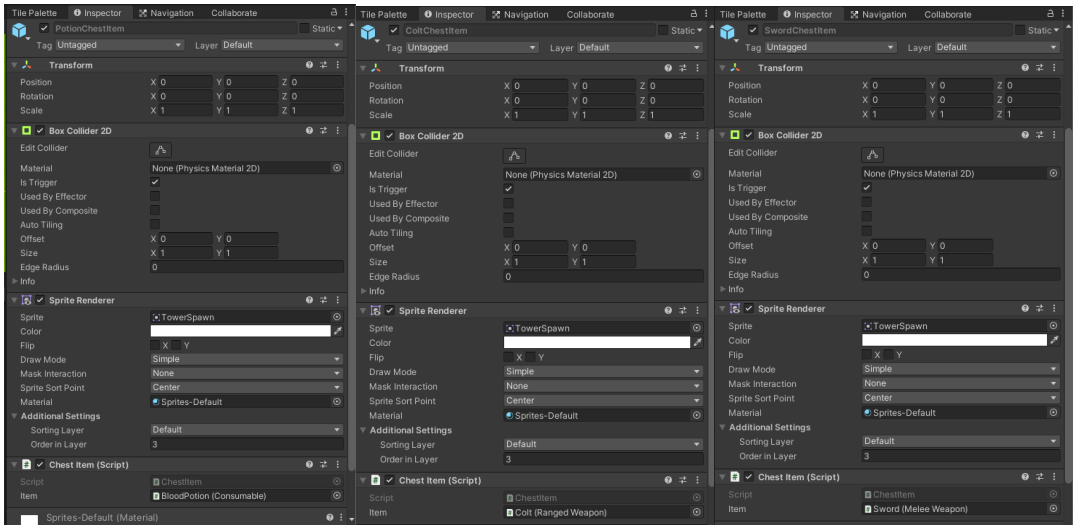
```
private void Awake()
{
    for (int i = 0; i < chestItems.Count; i++)
    {
        switch(chestItems[i].item.itemRarity)
        {
            case Item.rarity.common:
                commonItems.Add(chestItems[i]);
                break;
            case Item.rarity.uncommon:
                uncommonItems.Add(chestItems[i]);
                break;
            case Item.rarity.rare:
                rareItems.Add(chestItems[i]);
                break;
            case Item.rarity.legendary:
                legendaryItems.Add(chestItems[i]);
                break;
        }

    }
}
```

107.) The “Shop” Prefab is moved to the Hierarchy. Then we add another “Shop” to the Hierarchy, unpack it and rename it “Chest”, we change the position and the

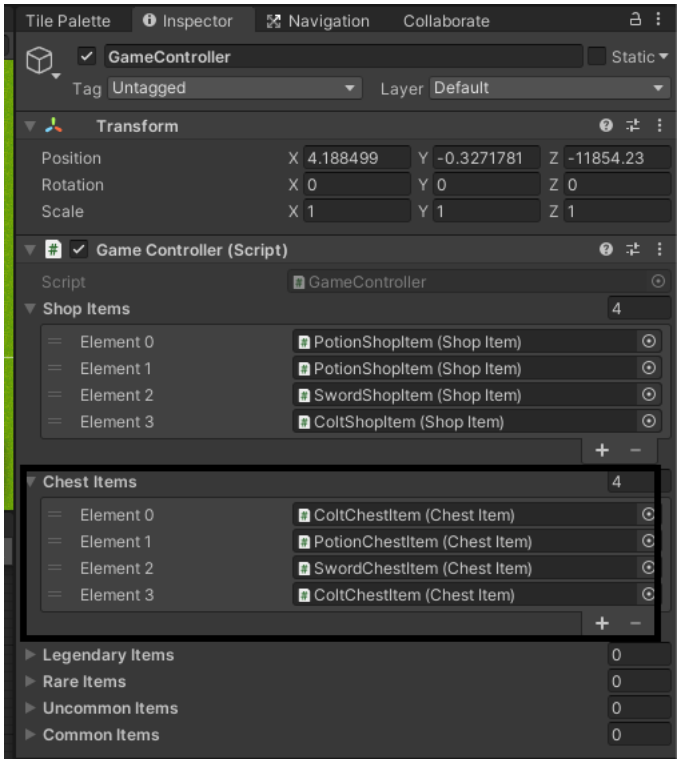
Sprite. We remove the “Shop.cs” and add “Chest.cs”. Then we make it into a prefab. We then set “ChestParent”.



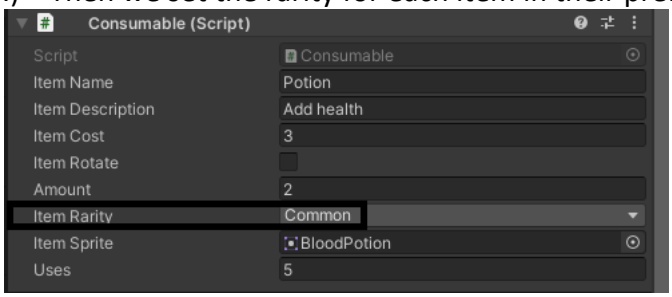
108.) Then we make Prefabs for “Potion/Sword/ColtChestItem” from the “PotionShopItem”. We do this by changing the sprites, and the script file from “ShopItem.cs” to “ChestItem.cs”.

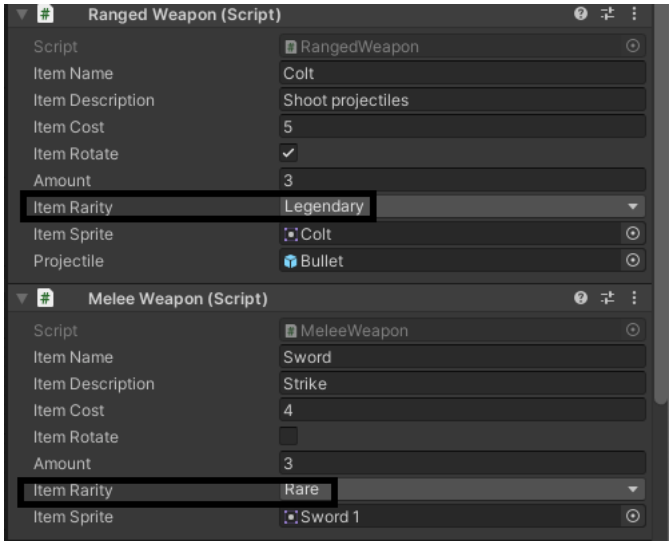


109.) Then we go into the “GameController” game object and add the items in the “Chest Items” list.

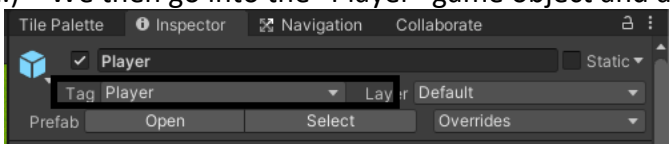


110.) Then we set the rarity for each item in their prefab.

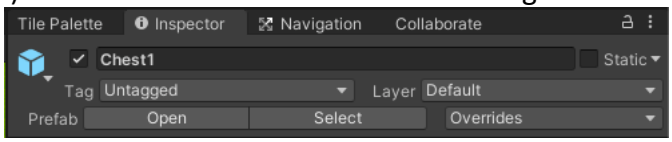




111.) We then go into the “Player” game object and add the tag “Player” to it.



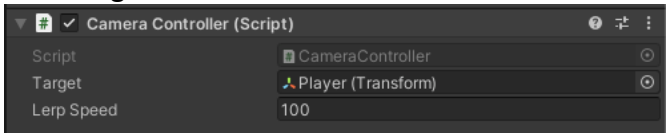
112.) We then add another chest into the game and name it “Chest1”.



113.) We create a script file “CameraController.cs” and its exactly the same as the one from our “Racing” game. So we copy and paste it over to the “CameraController.cs” we just created.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraController : MonoBehaviour
{
    public Transform target;
    public float lerpSpeed;

    // Update is called once per frame
    void FixedUpdate()
    {
        transform.position = Vector3.Lerp(transform.position, new Vector3(target.position.x,
target.position.y, -10), lerpSpeed * Time.fixedDeltaTime);
    }
}
```

114.) We then attach it to the “Main Camera” in the Hierarchy and we set “Lerp Speed” and “ Camera Controller (Script)” to it. We then drag “Player” game object to the “Target”.

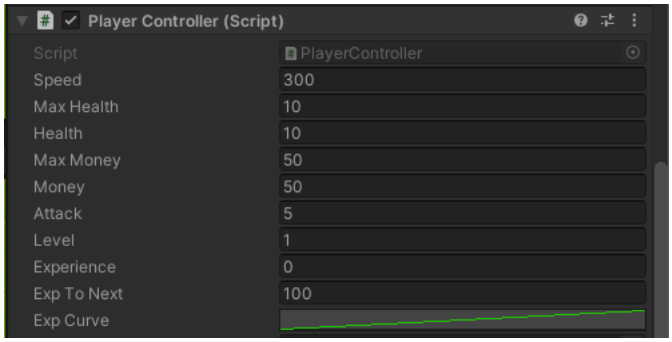


115.) We go into “PlayerController.cs” and add variables for attack, level, and experience. We then add a function “CalculateExp(int level)” which will calculate the

experience needed to move to the next level. Lastly, we call “CalculateExperience()”. Then we add a forloop for the “expCurve” so we can see it in our inspector.

```
private void Awake()
{
    playerRigidbody = GetComponent<Rigidbody2D>();
    anim = GetComponent<Animator>();
    rend = GetComponent<SpriteRenderer>();
    health = maxHealth;
    money = maxMoney;

    expToNext = CalculateExp(level);
    for( int i = 1; i <= 30; i++)
        expCurve.AddKey(i, CalculateExp(i));
}

public float attack;
public int level = 1;
public float experience;
public float expToNext;
public AnimationCurve expCurve = new AnimationCurve();
public float CalculateExp(int level)
{
    float expNeeded = level * 100f;
    return expNeeded;

}
```

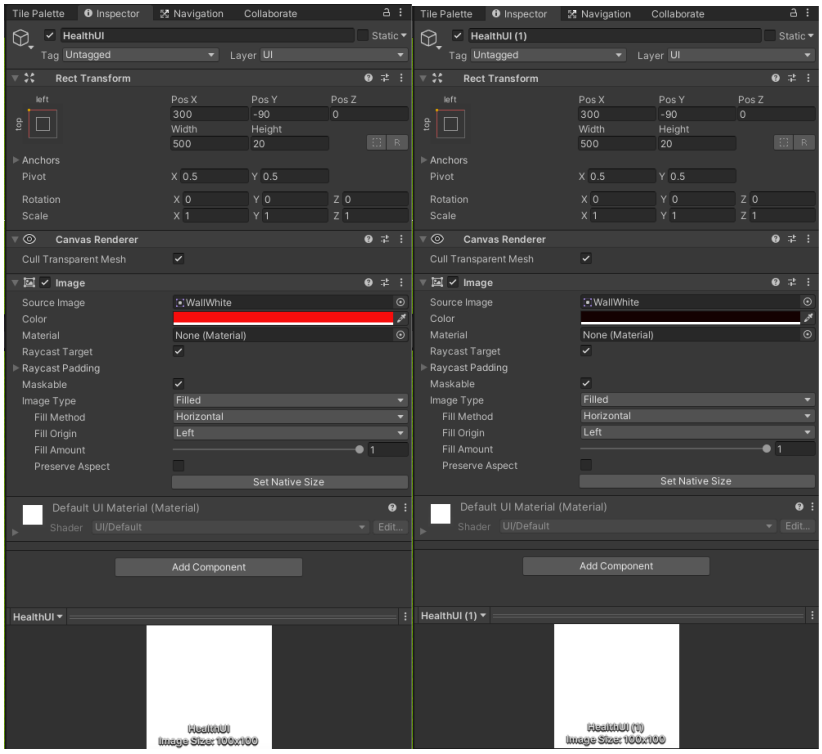
116.) We then go into the “Player” game object and set “Attack”.



117.) We then go into “PlayerController.cs” and add the functions “AddExp()” and “LevelUp()”. “LevelUp()” will handle how the players move set is upgraded.

```
public void AddExp(float amt)
{
    experience += amt;
    if ( experience >= expToNext)
        LevelUp();
}
public void LevelUp()
{
    level++;
    experience -= expToNext;
    attack += 5f;
    speed += 50f;
    Heal(maxHealth);
    expToNext = CalculateExp(level);
}
```

118.) Then we create an Image in the Hierarchy, we attach a white sprite to the source image, and we resize it, change the position. We make the color red. For “Image type” we select “Filled”, “Fill Method” we select “Horizontal”, and “Fill Origin” is “Left”. We then duplicated “HealthUI” named “HealthUI (1)” and we put it above “HealthUI”. We then make the color black.

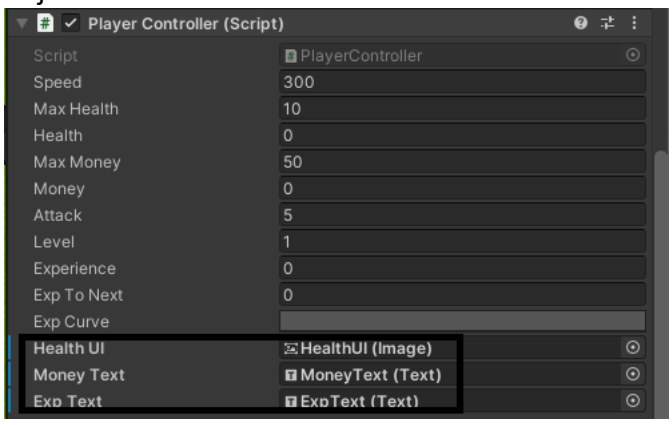


119.) Then we create another “Text” from UI and call it “MoneyText” we set the position, we set the font size, height and widtg. We also set the color as blue, and duplicate it and name it “ExpText” we set the Text for both.

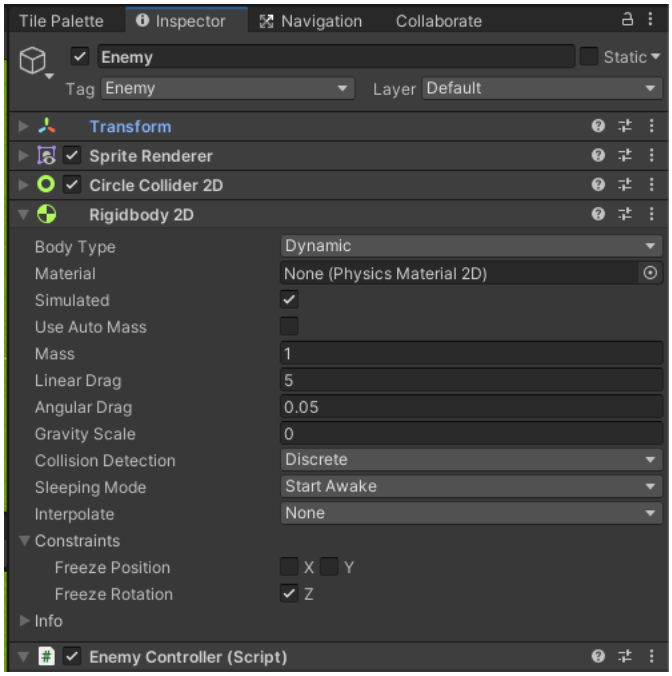
120.) In “PlayerController.cs” we add variables that will hold “MoneyText” and “ExpText”, and we set them in the “Update()” function.

```
public Image healthUI;
public Text moneyText;
public Text expText;
void Update()
{
    input = new Vector2(Input.GetAxis("Horizontal"), Input.GetAxis("Vertical"));

    playerRigidbody.AddForce(input * speed * Time.deltaTime);

    moving = (input.x != 0 || input.y != 0);

    if (input.y > 0)
        lookDir = 2;
    else if (input.y < 0)
        lookDir = 0;

    if (input.x > 0)
    {
        lookDir = 1;
        rend.flipX = false;
    }
    else if (input.x < 0)
    {
        lookDir = 1;
        rend.flipX = true;
    }

    anim.SetInteger("dir", lookDir);
    anim.SetBool("moving", moving);

    if (Input.GetKeyDown(KeyCode.Space))
        SwingAttack();



    if(Application.isEditor)
    {
        if(Input.GetKeyDown(KeyCode.L))
            Damage(5f);
        if(Input.GetKeyDown(KeyCode.J))
            AddExp(20f);

    }

    healthUI.fillAmount = health / maxHealth;
    moneyText.text = "Coins: " + money.ToString();
    expText.text = "Level: " + level.ToString() + " Exp: " + experience.ToString() + "/"
+ expToNext.ToString();
```

121.) Then we set these in the “Player” game object with the corresponding game objects.



122.) We then go into “Enemy” game object and attach the “Rigid Body 2D” to it. We then set “Gravity”, “Linear Drag”, and “Freeze Rotation: Z”. We then attach “EnemyController.cs”



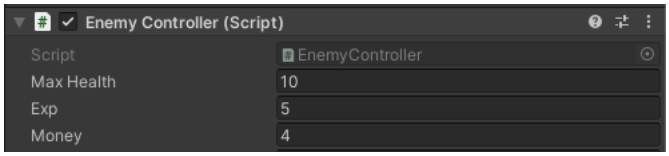
123.) We then go into “EnemyController.cs” and create variables. We then create the “Damage()” and “Die()” Functions. We also get the reference to the “PlayerController” in “Awake ()”.

```
private void Awake()
{
    player = FindObjectOfType<PlayerController>();
// Start is called before the first frame update
void Start()
{

}

// Update is called once per frame
void Update()
{
    if (iframe > 0) iframe -= Time.deltaTime;
}

public void Damage(float amt)
{
        Die();
    }
}

void Die()
{
    gameObject.SetActive(false);

    player.AddExp(exp);
    player.AddMoney(money);
}
}
```

124.) We then go into “Projectile.cs” and get a reference to “Colt” in “Awake()”.

```
RangedWeapon colt;

void Awake()
{
    bulletRigidbody = GetComponent<Rigidbody2D>();
    colt = FindObjectOfType<RangedWeapon>();
}
```

125.) Then we set the variables in the “Enemy Controller (Script)” in the “Enemy” game object.



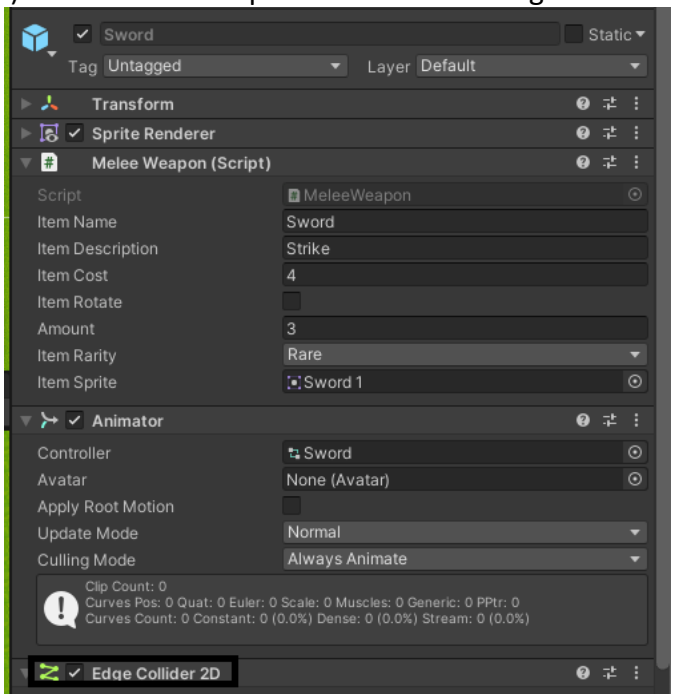
126.) We then go into “EnemyController.cs” and set “health” to “maxHealth” in the “Awake()” function.

```
private void Awake()
{
    player = FindObjectOfType<PlayerController>();
    health = maxHealth;
```

127.) Then we set iframe to know how much time passes between an enemy taking damage.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyController : MonoBehaviour
{

    public float maxHealth;
    float health;
    public float exp;
    public int money;

    PlayerController player;
    public float iframeTime = 0.3f;
    float iframe;

    private void Awake()
    {
        player = FindObjectOfType<PlayerController>();
        health = maxHealth;
        iframe = iframeTime;
    }
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (iframe > 0) iframe -= Time.deltaTime;
    }

    public void Damage(float amt)
    {
        if (iframe <=0)
        {
            health -= amt;
            iframe = iframeTime;
            if (health <= 0)
                Die();
        }
    }

    void Die()
    {
        gameObject.SetActive(false);

        player.AddExp(exp);
        player.AddMoney(money);
    }
}
```

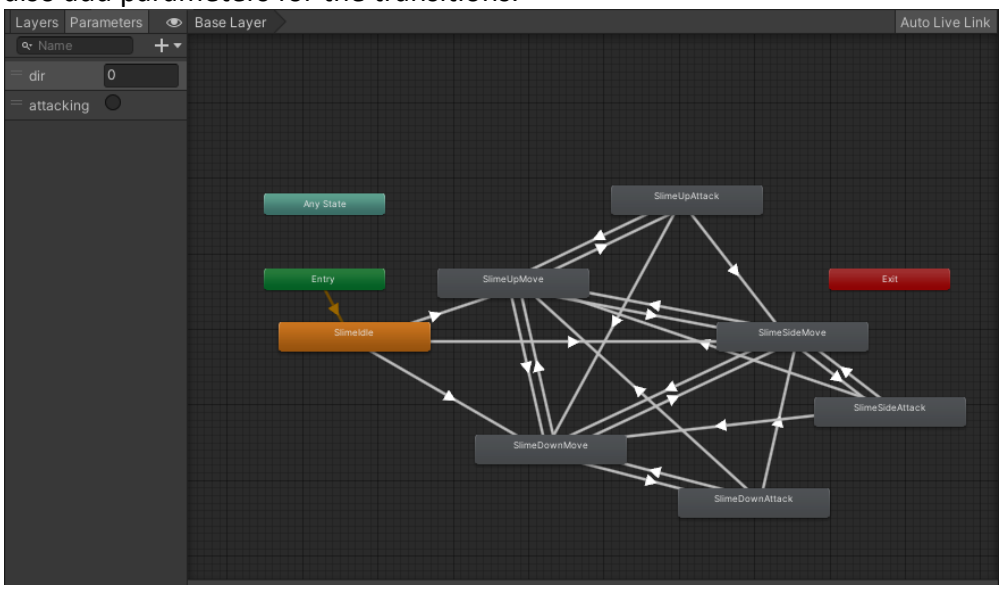
128.) In the “Sword” prefab we add an “Edge Collider 2D”.



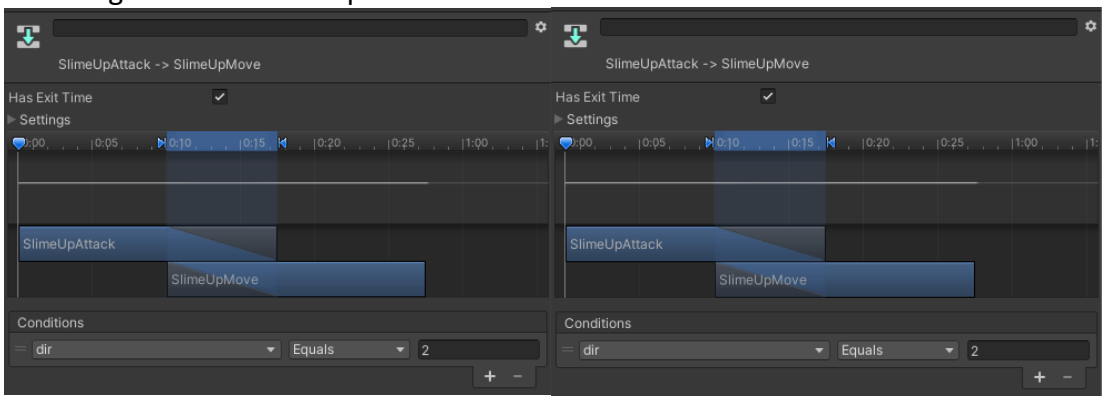
129.) Then in “MeleeWeapon.cs” we add the function “OnTriggerEnter2D()”. This will check if the sword hits the enemy. If it does we will call the function “Damage()” making the enemy take damage.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.CompareTag("Enemy"))
    {
        Debug.Log("Sword hit an enemy");
        collision.GetComponent<EnemyController>().Damage(amount);
    }
```

130.) Then we add animation to the enemy. Some for attacking, moving, idle. We then save these by adding sprites to the “Enemy” game object, and saving them into the “Animations” folder. Afterwards we add transitions between the animations. We also add parameters for the transitions.



131.) Then some transitions will have “Has Exit Time” and some will not. Others will have one parameter or more that dictate the direction and whether or not it is attacking. Below are examples of each case.



132.) There was a bug where a sprite would not face the correct way when getting it from shop. We will set “localScale” in “Inventory.cs”.

```
public void AddItem(Item item)
{
    Item newItem = Instantiate(item);
    newItem.transform.SetParent(transform);
    newItem.transform.localPosition = Vector3.zero;
    newItem.transform.localRotation = Quaternion.identity;
    newItem.transform.localScale = new Vector3(1, 1, 1);
    items.Add(newItem);
    newItem.gameObject.SetActive(false);
}
```

133.) Also, the rotation for the colt is off. We will fix this in the “Update()” function.

```
// Update is called once per frame
void Update()
{
    if (rotate)
    {
        Vector3 dir = Input.mousePosition - Camera.main.WorldToScreenPoint(transform.position);
        float angle = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg;
        if (transform.localScale.x != 1)
            angle += 180;
        transform.rotation = Quaternion.Lerp(transform.rotation, Quaternion.AngleAxis(angle,
Vector3.forward), Time.deltaTime * 10f);
    }
```

134.) Then in the “RangedWeapon.cs” we fix the projectile bug where the projectile shoots in the direction of the mouse rather than where the colt is facing.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RangedWeapon : Item
{

    public GameObject Projectile;

    public override void Use()
    {
        base.Use();
        //Debug.Log("Use RangedWeapon");
        if (inventory.transform.localScale.x == 1)
            Instantiate(Projectile, transform.position, transform.rotation * Quaternion.Euler(0, 0, -90));
        if (inventory.transform.localScale.x == -1)
            Instantiate(Projectile, transform.position, transform.rotation * Quaternion.Euler(0, 0, 90));
    }

    public override void Remove()
    {
        base.Remove();
        Debug.Log("Remove RangedWeapon");

    }
}
```

135.) Next we will go into the “EnemyController.cs” and create the enemy states between chase and attack. We will create a switch statement in the “Update()” function and implement the “Chase()” and “Attack()” functions.

```
public enum enemystates {chase, attack };
public enemystates currentState;
public float timeBetweenAttacks = 1f;
float cools;
public float speed;
int dir;
public float attackRange;
float distance;
// Update is called once per frame
void Update()
{
    if (iframe > 0) iframe -= Time.deltaTime;
    if (cools > 0) cools -= Time.deltaTime;

    switch (currentState)
    {
        case (enemystates.chase):
            Chase();
            break;
        case (enemystates.attack):
            Attack();
            break;
    }
void Chase()
{
    distance = Vector2.Distance(transform.position, player.transform.position);

    if (player.transform.position.y < transform.position.y)
    {
        dir = 0;
        meleeCollider.transform.localPosition = new Vector3 (0, 0, 0);
    }
    else if (player.transform.position.x > transform.position.x)//right
    {
        rend.flipX = false;
        dir = 1;
        meleeCollider.transform.localPosition = new Vector3 (0.3f, 0.2f, 0);

    }
    else if (player.transform.position.x < transform.position.x)//left
    {
        rend.flipX = true;
        dir = 1;
        meleeCollider.transform.localPosition = new Vector3 (-0.3f, 0.2f, 0);

    }
    else if (player.transform.position.y > transform.position.y)
    {
        dir = 2;
        meleeCollider.transform.localPosition = new Vector3 (0f, 0.5f, 0);

    }

    if (distance > attackRange)
    {
        Vector3 direction = player.transform.position - transform.position;
        enemyRigidbody.AddForce(direction * speed * Time.deltaTime);
    }
    else
    {
        if (cools <= 0) currentState = enemystates.attack;
    }
}

void Attack()
{
    anim.SetTrigger("attacking");
    cools = timeBetweenAttacks;
    currentState = enemystates.chase;
}
```

136.) Next, we will get a reference to the “GameController” and the RigidBody2D” in the “Awake()” function.

```
GameController cont;
Rigidbody2D enemyRigidbody;
private void Awake()
{
    player = FindObjectOfType<PlayerController>();
    cont = FindObjectOfType<GameController>();
    anim = GetComponent<Animator>();
    enemyRigidbody = GetComponent<Rigidbody2D>();
```

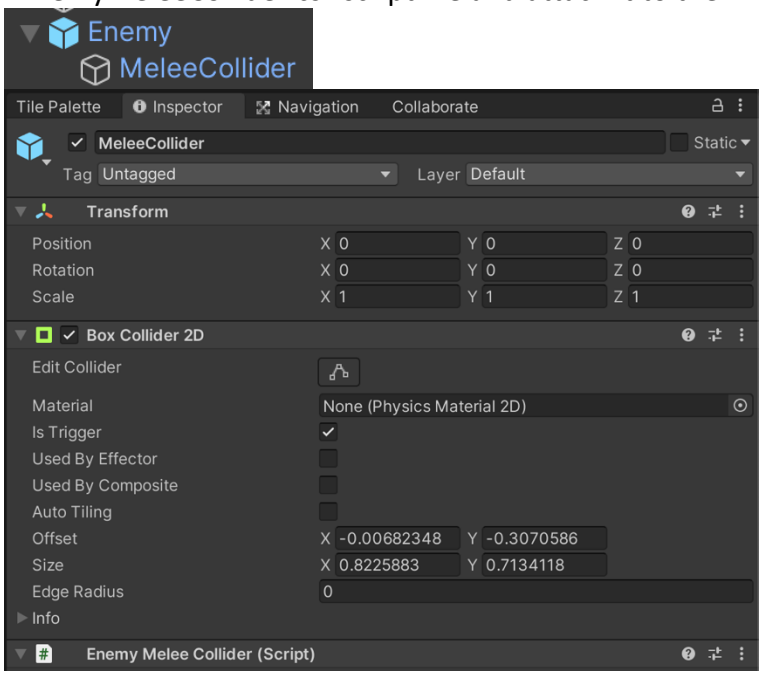
137.) Then in the “Update()” we set the direction.

```
// Update is called once per frame
void Update()
{
    anim.SetInteger("dir", dir);
}
```

138.) In the “Chase()” function we will flip the sprite we add a reference to the “SpriteRenderer” to handle this.

```
SpriteRenderer rend;
private void Awake()
{
    player = FindObjectOfType<PlayerController>();
    cont = FindObjectOfType<GameController>();
    anim = GetComponent<Animator>();
    enemyRigidbody = GetComponent<Rigidbody2D>();
    rend = GetComponent<SpriteRenderer>();
    health = maxHealth;
    iframe = iframeTime;
    cools = timeBetweenAttacks;

}
```

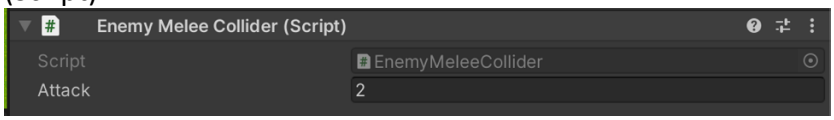
139.) Next, we will add a “MeleeCollider” to the “Enemy” game object. Then we add a “Box Collider 2D” to the “MeleeCollider” and edit the size. Afterwards we create a “EnemyMeleeCollider.cs” script file and attach it to the “MeleeCollider”.



140.) We then go into “EnemyMeleeCollider.cs” and implement it. We add variables and create the “OnTriggerEnter2D()” function for when it comes into contact with the “Player”. We then call the “Damage” function from the “PlayerController.cs” by referencing it. We also create another function “OnTriggerStay2D()” for if the player stays in the “Box Collider 2D” of the enemy.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyMeleeCollider : MonoBehaviour
{
    public float attack;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if(collision.gameObject.CompareTag("Player"))
        {
            collision.gameObject.GetComponent<PlayerController>().Damage(attack);
        }
    }
    private void OnTriggerStay2D(Collider2D collision)
    {
        if(collision.gameObject.CompareTag("Player"))
        {
            collision.gameObject.GetComponent<PlayerController>().Damage(attack);
        }
    }
}
```

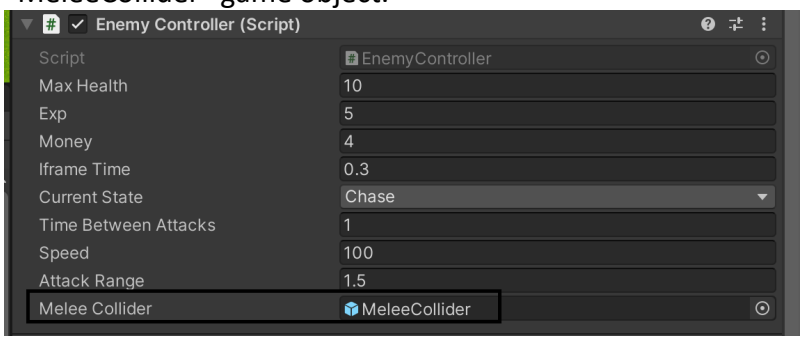
141.) Then we set the “MeleeCollider[‘s]” “Attack” under “Enemy Melee Collider (Script).



142.) When the “Enemy[‘s]” sprite moves we want the “Box Collider 2D” to move with it. We will set this manually in the script file. We go into “EnemyController.cs” and get a reference to “MeleeCollider”. We then take care of this in the “Chase() “ function”.

```
public GameObject meleeCollider;
void Chase()
{
    distance = Vector2.Distance(transform.position, player.transform.position);

    if (player.transform.position.y < transform.position.y)
    {
        dir = 0;
        meleeCollider.transform.localPosition = new Vector3 (0, 0, 0);
    }
    else if (player.transform.position.x > transform.position.x)//right
    {
        rend.flipX = false;
        dir = 1;
        meleeCollider.transform.localPosition = new Vector3 (0.3f, 0.2f, 0);

    }
    else if (player.transform.position.x < transform.position.x)//left
    {
        rend.flipX = true;
        dir = 1;
        meleeCollider.transform.localPosition = new Vector3 (-0.3f, 0.2f, 0);

    }
    else if (player.transform.position.y > transform.position.y)
    {
        dir = 2;
        meleeCollider.transform.localPosition = new Vector3 (0f, 0.5f, 0);

    }

    if (distance > attackRange)
    {
        Vector3 direction = player.transform.position - transform.position;
        enemyRigidbody.AddForce(direction * speed * Time.deltaTime);
    }
    else
    {
        if (cools <= 0) currentState = enemystates.attack;
    }
}
```

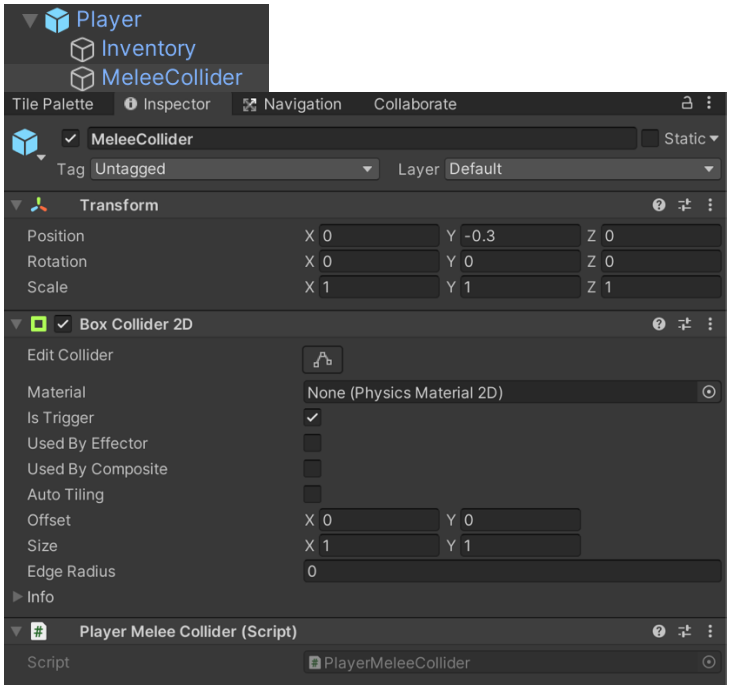
143.) Then in the “Enemy” game object we set the variable “MeleeCollider” with the “MeleeCollider” game object.



144.) Then we set the iframe for the player so that the player doesn’t take so much damage all at once.

```
public float iframeTime = 0.8f;
float iframe;

private void Awake()
{
    playerRigidbody = GetComponent<Rigidbody2D>();
    anim = GetComponent<Animator>();
    rend = GetComponent<SpriteRenderer>();
    health = maxHealth;
    money = maxMoney;
    iframe = iframeTime;

    expToNext = CalculateExp(level);
    for( int i = 1; i <= 30; i++)
        expCurve.AddKey(i, CalculateExp(i));

}
// Update is called once per frame
void Update()
{
    if (iframe > 0) iframe -= Time.deltaTime;
public void Damage(float amt)
{
    if (iframe <= 0)
    {
        health -= amt;
        iframe = iframeTime;
        if (health <= 0) Die();
    }
}
```

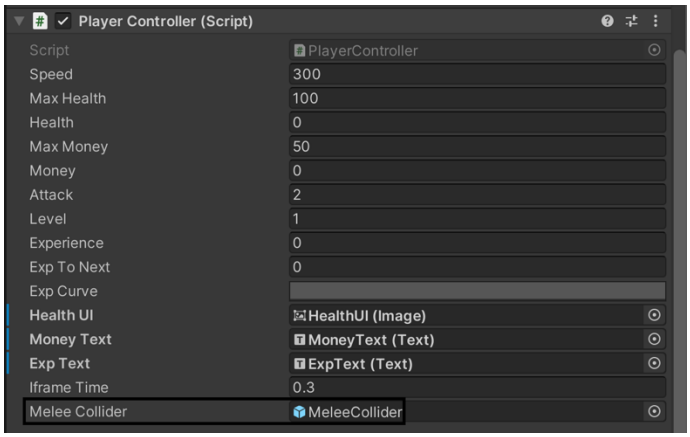
145.) Similar to the “Enemy” we will add a “MeleeCollider” for the “Player”. We then set a “Box Collider 2D” change the size, and create a “PlayerMeleeCollider.cs” and attach it to the “MeleeCollider”.



146.) Like the “Enemy” we will change the “Box Collider 2D” to change with the sprite position.

```
public GameObject meleeCollider;
// Update is called once per frame
void Update()
{
    if (iframe > 0) iframe -= Time.deltaTime;

    input = new Vector2(Input.GetAxis("Horizontal"), Input.GetAxis("Vertical"));

    playerRigidbody.AddForce(input * speed * Time.deltaTime);

    moving = (input.x != 0 || input.y != 0);

    if (input.y < 0)
    {
        lookDir = 0;
        meleeCollider.transform.localPosition = new Vector3 (0f, -0.3f, 0);
    }
    else if (input.x > 0)//right
    {
        rend.flipX = false;
        lookDir = 1;
        meleeCollider.transform.localPosition = new Vector3 (0.3f, 0f, 0);

    }
    else if (input.x < 0)//left
    {
        rend.flipX = true;
        lookDir = 1;
        meleeCollider.transform.localPosition = new Vector3 (-0.3f, 0f, 0);

    }
    else if (input.y > 0)
    {
        lookDir = 2;
        meleeCollider.transform.localPosition = new Vector3 (0f, 0.3f, 0);
    }
```

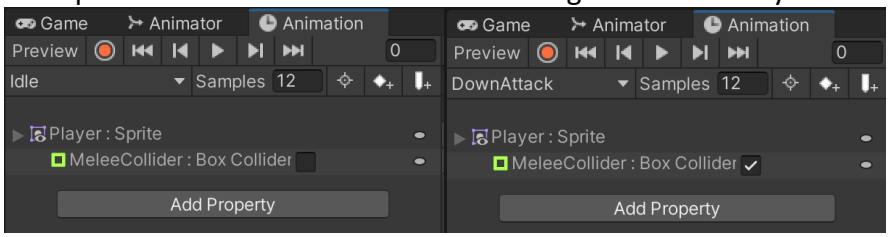
147.) Then under the “Player” game object we place the “MeleeCollider” game object into the variable of the same name.



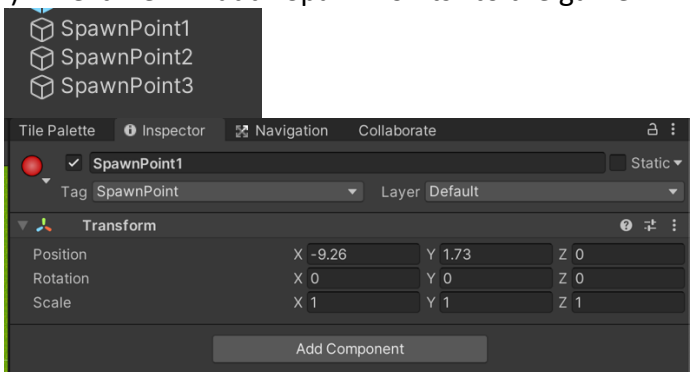
148.) Then we go into “PlayerMeleeCollider.cs” and copy over everything from “MeleeEnemyCollider.cs”. The only difference is change “<PlayerController>” to “<EnemyController>” and change tags to “Enemy”. We also have a reference to the “PlayerController.cs” in the “Awake()” function.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMeleeCollider : MonoBehaviour
{
    PlayerController player;

    private void Awake()
    {
        player = FindObjectOfType<PlayerController>();
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if(collision.gameObject.CompareTag("Enemy"))
        {
            collision.gameObject.GetComponent<EnemyController>().Damage(player.attack);
        }
    }
    private void OnTriggerStay2D(Collider2D collision)
    {
        if(collision.gameObject.CompareTag("Enemy"))
        {
            collision.gameObject.GetComponent<EnemyController>().Damage(player.attack);
        }
    }
}
```

149.) When the “Player” is idle we disable the “Box Collider 2D” and when the “Player” is attacking we enable the Box Collider 2D”. We do this in the “Animation tab. We press record and either check or uncheck the “Box Collider 2D”. Below are two example of this. We do the exact same thing for the “Enemy”.



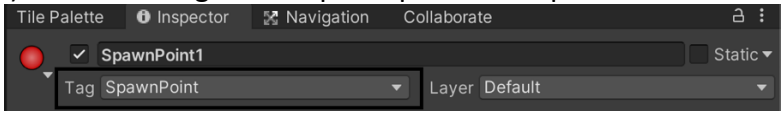
150.) Next we will add “SpawnPoints” to the game.



151.) We then go into the “GameController.cs” and create an array of “SpawnPoints”. Then in the awake function we find game objects with the tag “SpawnPoint”.

```
public GameObject[] spawnPoints;
private void Awake()
{
    spawnPoints = GameObject.FindGameObjectsWithTag("SpawnPoint");
```

152.) We then tag all the Spawn points as “SpawnPoint”.



153.) Next, we will add variables and implement them in the awake function we will implement the “cooldown” for the enemy spawns, then update it in the “Update()” function.

```
public float timeBetweenSpawns = 5f;
float cooldown;
```

```
    cooldown = timeBetweenSpawns;
// Update is called once per frame
void Update()
{

    if (cooldown > 0)
        cooldown -= Time.deltaTime;
    else
        SpawnEnemy();

}
```

154.) Then we implement the “SpawnEnemy()” function.

```
public GameObject enemy;

void SpawnEnemy()
{
    GameObject obj = GetEnemy();
    obj.transform.position = spawnPoints[Random.Range(0,
spawnPoints.Length)].transform.position;
    obj.SetActive(true);
    //Instantiate(enemy, spawnPoints[Random.Range(0, spawnPoints.Length)].transform.position,
Quaternion.identity);
    cooldown = timeBetweenSpawns;
}
```

155.) Then we create a list of enemies in the “GameController.cs” and we implement a “GetEnemy()” function. It will check every enemy in the Hierarchy, it will return an inactive enemy and reuse it in the “SpawnEnemy()”. This will allow us to save resources in the CPU.

```
public List<GameObject> enemies;
GameObject GetEnemy()
{
    for (int i = 0; i < enemies.Count; i++)
    {
        if(!enemies[i].activeInHierarchy)
            return enemies[i];
    }
    GameObject en = Instantiate(enemy, transform.position, Quaternion.identity);
    enemies.Add(en);
    en.SetActive(false);
    return en;
}
```

156.) Then we go into the sword prefab. In the “Animation” tab we enable “Edge Collider 2D” for when it is attacking and disable it when idle.