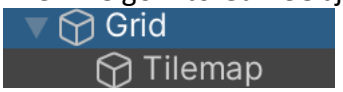
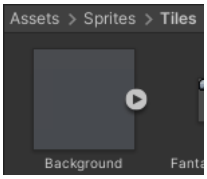
Microgame #5 Platformer Progress Report
Name: Alan L. Perez
id: 004862867
Unity File: https://play.unity.com/mg/other/microgame-5-platformer-cse-4410

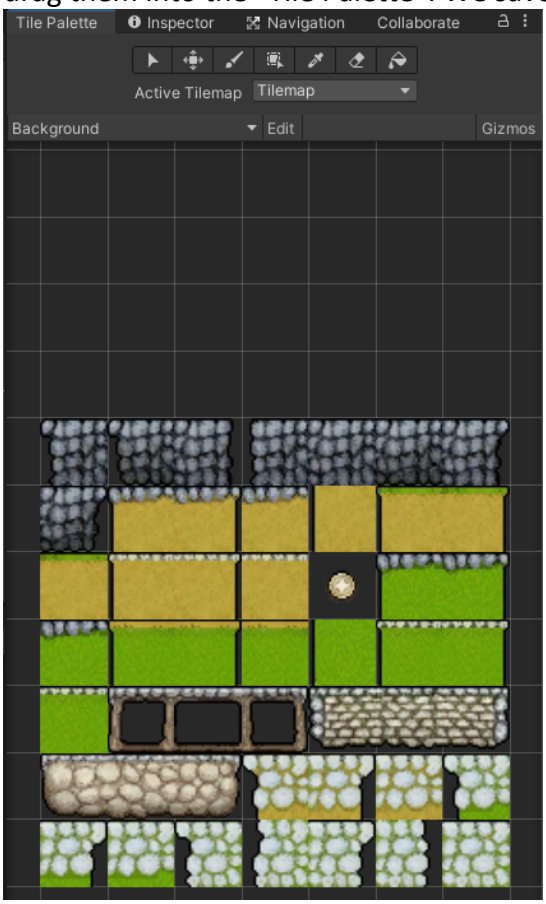
1.) Create a new project Platformer
2.) Create folders under Assets: Animations, Prefabs, Scenes, Scripts, Sprites, Tiles
3.) Download PlatformerSprites.zip from Canvas. Unzip, and drag sprites into Sprite folder. Put the Pride font under the Assets folder.
4.) Open the Enemies, Player, and Tiles folder, select all the sprites and change “Pixel Per Unit” to 32.
5.) Then we go into GameObject -> 2D -> Tilemap. Now we have a “Grid”, and “Tilemap”



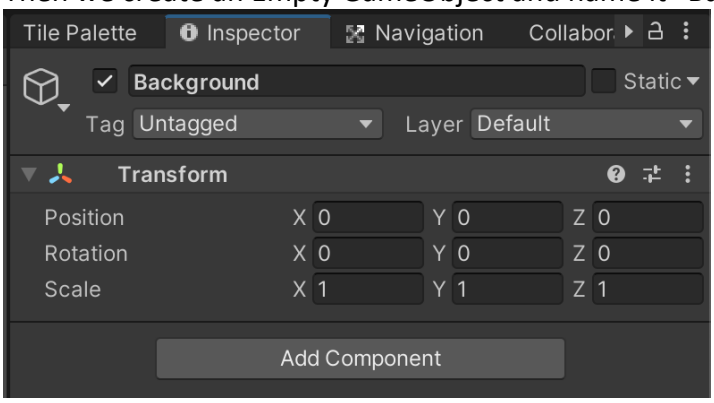
6.) Then we go into “Tile Palette” and “Create New Palette”. We will name this Background. Then we save this under the “Tiles” folder.



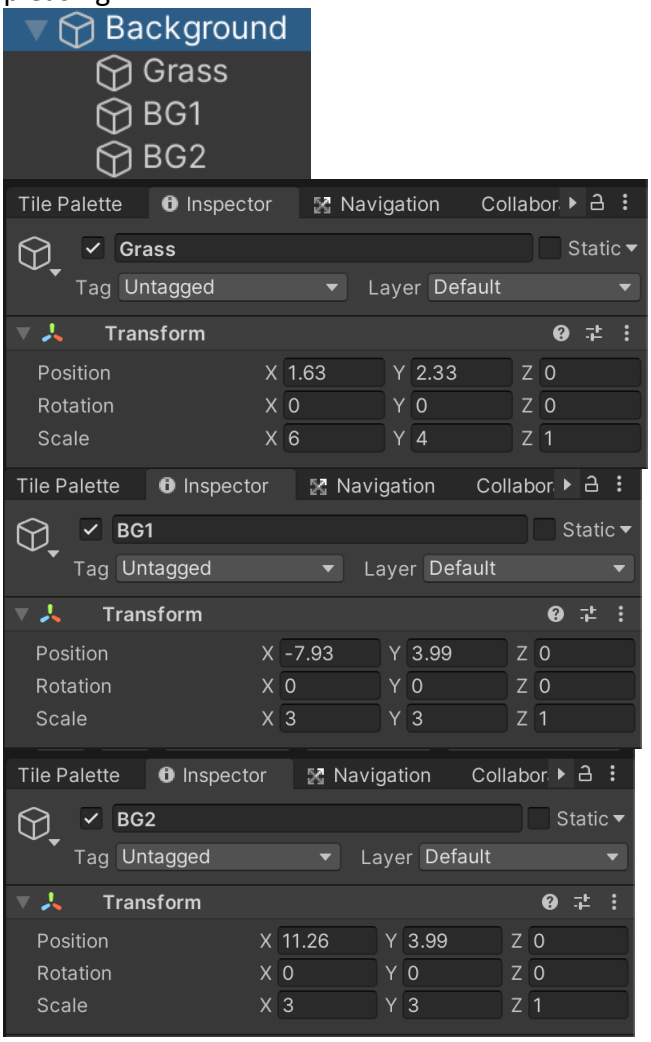
7.) Then go into the “Tiles” folder and select everything except the background sprites, and drag them into the “Tile Palette”. We save this under “Tiles” folder.



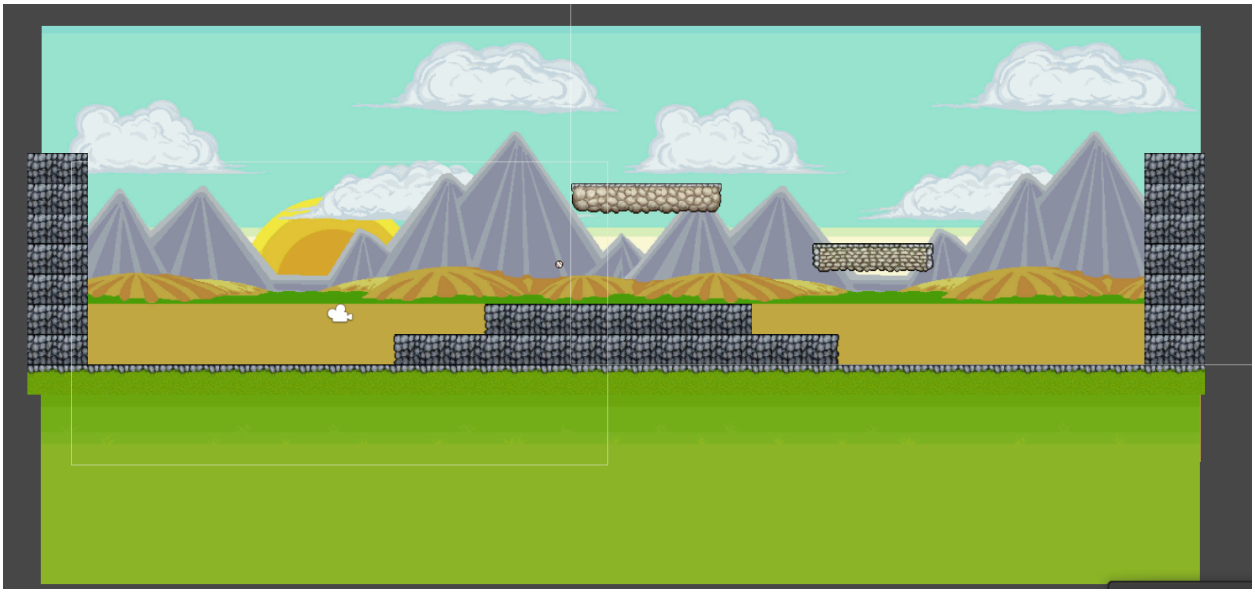
8.) Then we create an Empty GameObject and name it “Background”.



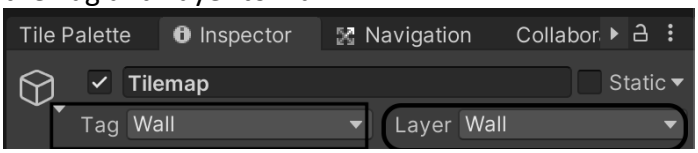
9.) Next we go to the “Tiles folder” and insert the background sprites. We name these “Grass”, “BG1”, and “BG2”. We adjust the size to make the background aesthetically pleasing.



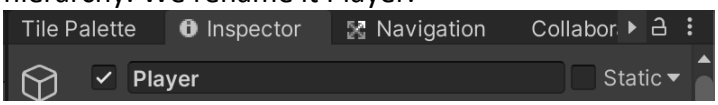
10.) Then we go into “Tile Palette” and we select tiles in order to make the ground and walls. We will also create Platforms



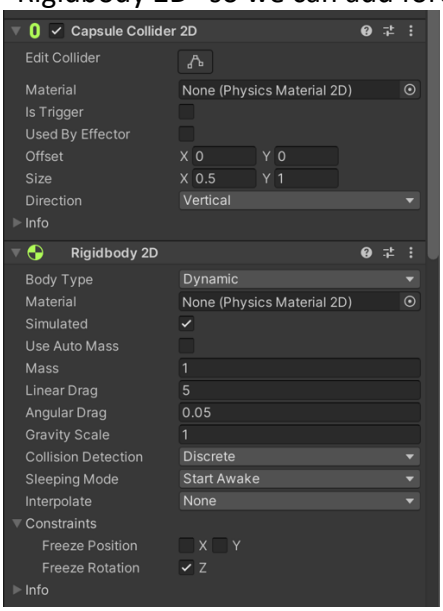
11.) Then we create a new Layer named “Wall”. We then go into “Tilemap” and set the Tag and Layer to wall.



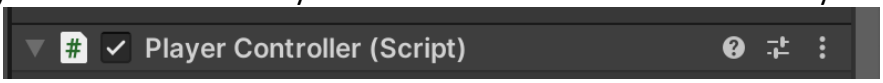
12.) We go to the player folder and grab the sprite “robotidle1” and drag it to the hierarchy. We rename it Player.



13.) We attach the “Capsule Collider 2D” and adjust the size. The attach the “Rigidbody 2D” so we can add force to the player. We then freeze “Z”.



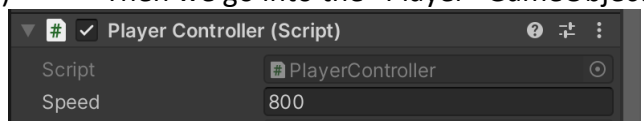
14.) We create “PlayerController.cs” and attach it to the Player GameObject



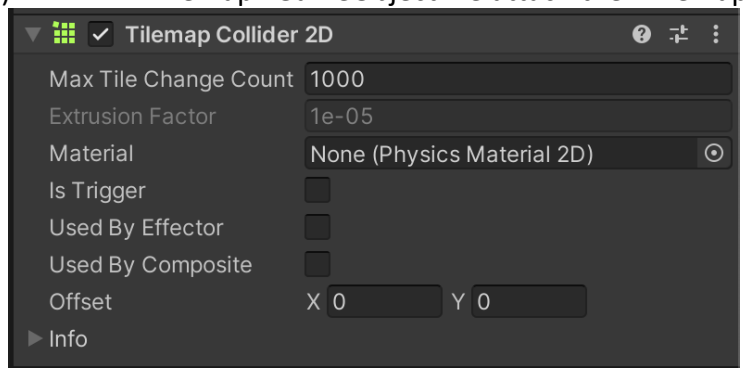
15.) We open “PlayerController.cs” and create variables. Then in the “Awake()” function we “GetComponent” “Rigidbody2D” and assign it to the “playerRigidBody” variable. Then we go into the “Update()” function and “GetAxisRaw” in order to get the keys being pressed. Then we add conditional statements to “AddForce” to the player to move it.

```
public class PlayerController : MonoBehaviour
{

   public float speed;
   Rigidbody2D playerRigidbody;
   float inputX;

    void Awake()
    {
       playerRigidbody = GetComponent<Rigidbody2D>();
   }

   // Update is called once per frame
   void Update()
   {
      inputX = Input.GetAxisRaw("Horizontal");

      if (inputX !=  0)
         playerRigidbody.AddForce(Vector2.right * inputX * speed *
Time.deltaTime);
```

16.) Then we go into the “Player” GameObject and set the speed.



17.) In “Tilemap” GameObject we attach the “Tilemap Collider 2D”.



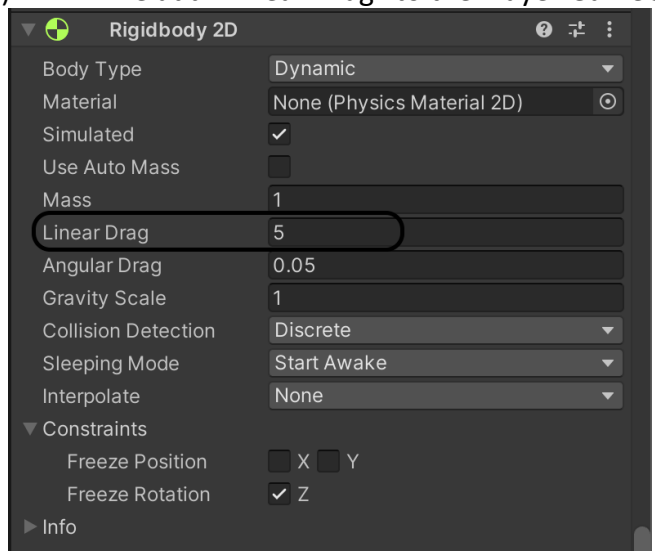
18.) We go back into “PlayerController.cs” and we want to implement player jumping ability. We add variables and in the “Awake()” function we initialize the variables that we created. Then we initialize the “RayCastHit2D Hit” in the “Update()” function. We then add conditional statements to see if the RayCast2D hits the wall layer collider. If it does the player can jump if not then the player cannot jump. It will also take in key input to see if the player wants to jump. We also add “DrawRay” so we can see the ray we created.

```
public LayerMask wallLayer;
public float rayLength;
bool canJump;
public float jumpHeight;
```

```
void Awake()
{
    playerRigidbody = GetComponent<Rigidbody2D>();
    canJump = true;
}

// Update is called once per frame
void Update()
{
    inputX = Input.GetAxisRaw("Horizontal");

    if (inputX !=  0)
        playerRigidbody.AddForce(Vector2.right * inputX * speed *
Time.deltaTime);

    rend.flipX = (inputX < 0);

    RaycastHit2D hit = Physics2D.Raycast(transform.position,
Vector2.down, rayLength, wallLayer);

    if (hit.collider != null)
    {
        canJump = true;

    }

    if (canJump && Input.GetKeyDown(KeyCode.Space))
    {
        playerRigidbody.AddForce(Vector2.up * jumpHeight);
        canJump = false;
    }

    Debug.DrawRay(transform.position, Vector2.down *
rayLength);
```

19.) We then go into “Player” GameObject, under “PlayerController.cs” we set the “Wall Layer” as “Wall”, we set “Ray Length” and we set “Jump Height”

20.) We add “Linear Drag” to the Player GameObject.



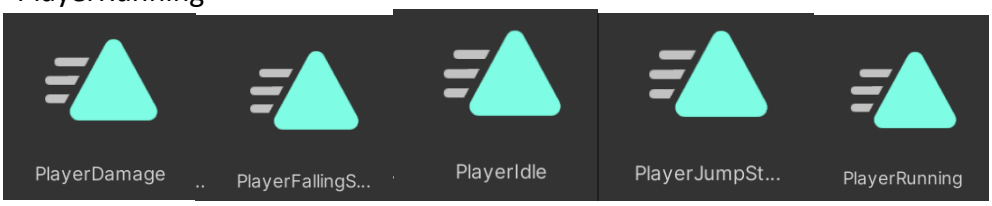
21.) We want the player to be able to take damage. We go into “PlayerController.cs” and we add variable to allow us to do this. We initialize the variables in the “Awake()” function. In the “Update()” function we add conditional statement statements to let the game logic know when a player is supposed to receive damage. We then create a a function “Damage()” that handles the functionality of the player taking damage. We also add a function, “ResetHurt()” for the player will be able to take damage again.

```
bool hurt;
public float maxHealth;
[SerializeField]
float health;
public float timeBetweenHurt;

float iframe;
```

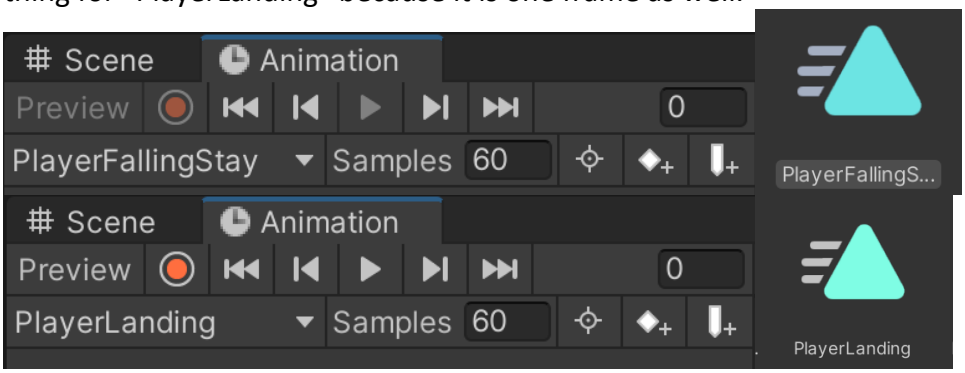
```
void Awake()
{
    playerRigidbody = GetComponent<Rigidbody2D>();
    canJump = true;
    health = maxHealth;
    hurt = false;
    iframe = timeBetweenHurt;
}
// Update is called once per frame
void Update()
{
    inputX = Input.GetAxisRaw("Horizontal");

    if (inputX !=  0)
        playerRigidbody.AddForce(Vector2.right * inputX * speed *
Time.deltaTime);

    rend.flipX = (inputX < 0);

    RaycastHit2D hit = Physics2D.Raycast(transform.position,
Vector2.down, rayLength, wallLayer);

    if (hit.collider != null)
    {
        canJump = true;

    }

    if (canJump && Input.GetKeyDown(KeyCode.Space))
    {
        playerRigidbody.AddForce(Vector2.up * jumpHeight);
        canJump = false;
    }

    Debug.DrawRay(transform.position, Vector2.down *
rayLength);

    if (iframe > 0) iframe -= Time.deltaTime;
```

```
    }
}


public void Damage(float amt)
{
    if(iframe < 0)
    {
        health -= amt;
        hurt = true;
        Invoke("ResetHurt", 0.2f);
        if (health <= 0)
        {
            GameOver();
        }
        iframe = timeBetweenHurt;
    }

}
void ResetHurt()
{
    hurt = false;
}
```

22.) Next we want to add animation to the “Player”. We go into the player sprite folder and select all the player sprites that are associated with the player taking damage, it contains three frames. We then drag those sprites into the Hierarchy. We save them in the “Animation” folder under “PlayerDamage”. We do the same thing with the player falling animation, the player being idle, the player jumping, and the player running. We save it as “PlayerFallingStart”, “PlayerIdle”, “PlayerJumpStart”, and “PlayerRunning”

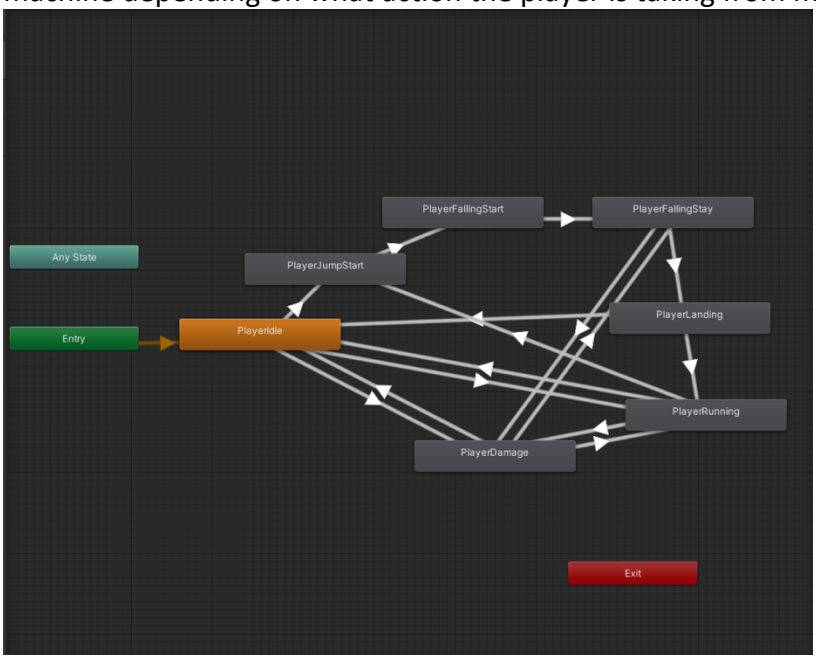


23.) Then we add a one frame animation, but we must do this manually. We go into the animation window and create new clip. We name it “PlayerFallingStay”. We click the recording mode and drag the one frame into the recoding time line. We do the same thing for “PlayerLanding” because it is one frame as well.

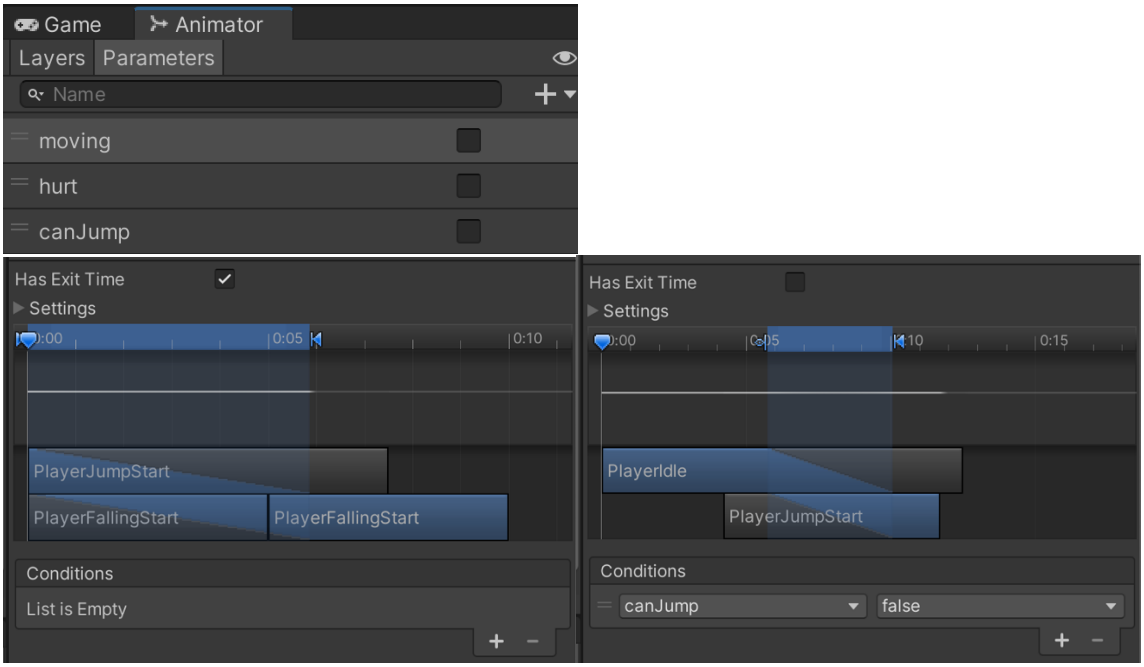


24.) Then we select “Player” and we open the animator window. From there we set “PlayerIdle” as the default state. From here we will create transitions in the state

machine depending on what action the player is taking from moment to moment.



25.) Depending on the animation we will either have it in the mode “HasExitTime” or it will have to meet certain conditions for the animation to end. We create the parameters and we either mark them as true or false in the animation. Below are the parameters and two examples of both cases. A transition can have more than one condition.



26.) Then we go into the “PlayerController.cs” and we add a parameter for the “Animatore” and we reference it in the “Awake()” function. Then in the “Update()” function, we set the animator parameters. We use the module “SetBool” to set the parameters in the script file.

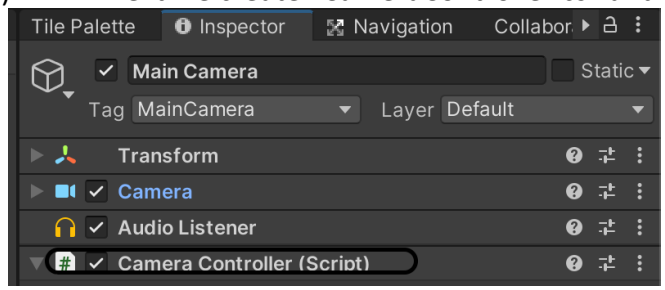
```
Animator anim;

 void Awake()
 {
    playerRigidbody = GetComponent<Rigidbody2D>();
    canJump = true;
    health = maxHealth;
    hurt = false;
    iframe = timeBetweenHurt;
    anim = GetComponent<Animator>();
 }

// Update is called once per frame
void Update()
{
    inputX = Input.GetAxisRaw("Horizontal");

    if (inputX !=  0)
       playerRigidbody.AddForce(Vector2.right * inputX * speed *
Time.deltaTime);

    rend.flipX = (inputX < 0);

    RaycastHit2D hit = Physics2D.Raycast(transform.position,
Vector2.down, rayLength, wallLayer);

    if (hit.collider != null)
    {
       canJump = true;

    }

    if (canJump && Input.GetKeyDown(KeyCode.Space))
    {
       playerRigidbody.AddForce(Vector2.up * jumpHeight);
       canJump = false;
    }

    Debug.DrawRay(transform.position, Vector2.down *
rayLength);


    if (iframe > 0) iframe -= Time.deltaTime;

    // test Damage
    if (!hurt && Input.GetKeyDown(KeyCode.LeftControl))
       Damage(2);

    healthImage.fillAmount = Mathf.Lerp(healthImage.fillAmount,
health/maxHealth, Time.deltaTime * 10f);
    coinsText.text = "X " + coins.ToString();

    anim.SetBool("moving", inputX != 0);
    anim.SetBool("canJump", canJump);
    anim.SetBool("hurt", hurt);
```

27.) We declare a variable for SpriteRenderer and refernce it in the “Awake()” function. Then we go to the “Update()” and we add a “rend.flipX = (input < 0)” so we can flip our sprite. Then in Sprite Renderer in the “Player” GameObject we click “X” to flip our player.

```
SpriteRenderer rend;
void Awake()
{
    playerRigidbody = GetComponent<Rigidbody2D>();
    canJump = true;
    health = maxHealth;
    hurt = false;
    iframe = timeBetweenHurt;
    anim = GetComponent<Animator>();
    rend = GetComponent<SpriteRenderer>();
```

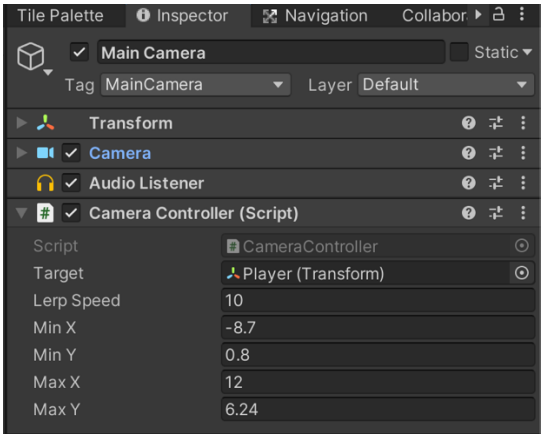
```
// Update is called once per frame
void Update()
{
    inputX = Input.GetAxisRaw("Horizontal");

    if (inputX !=  0)
        playerRigidbody.AddForce(Vector2.right * inputX * speed *
Time.deltaTime);

    rend.flipX = (inputX < 0);
```

28.) Next we create “CameraController.cs” and we attach it to “Main Camera”



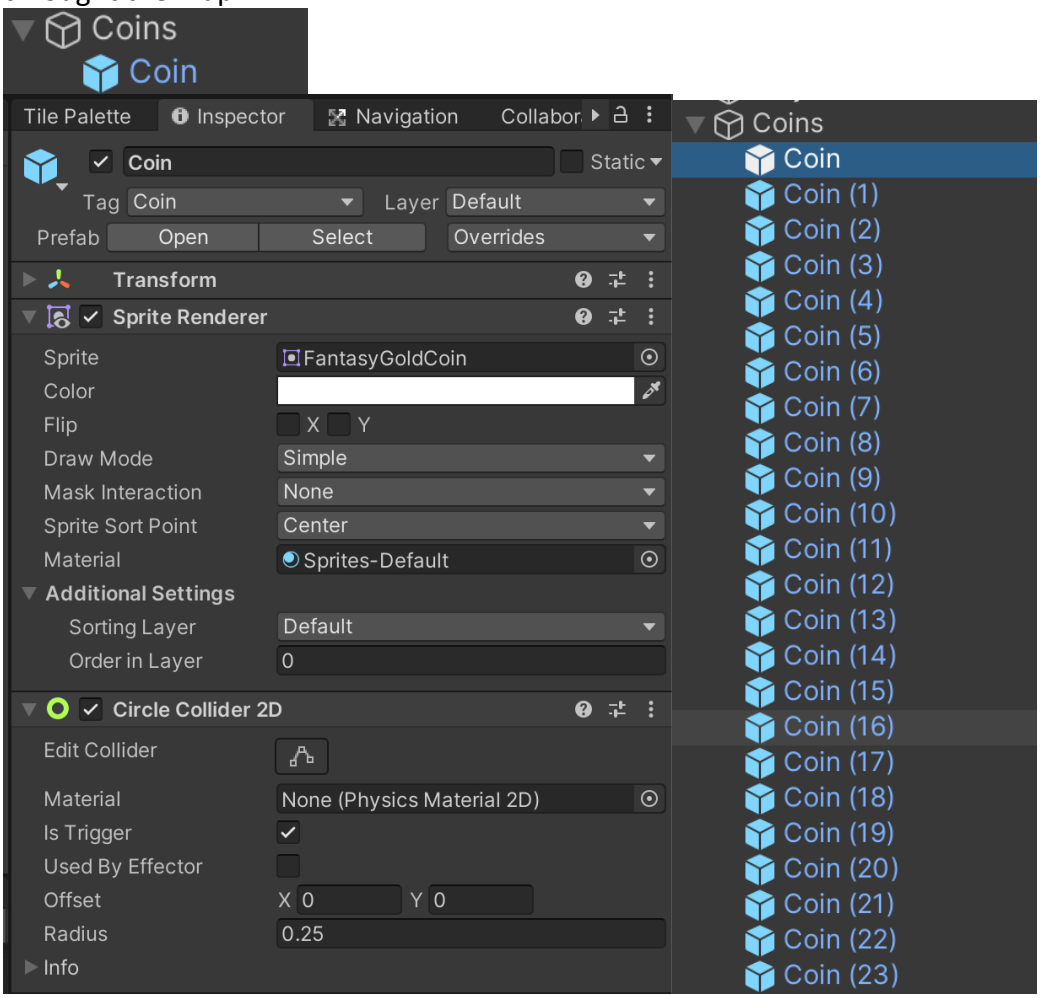
29.) We then go into “CameraController.cs” and add variables. We create a function called “FixedUpdate()”. This function will allow the camera to follow the target “Player”. We set the target as “Player” in the “Main Camera” GameObject. We also set Lerp Speed. We don’t want the camera to move out of bounds of the “Tilemap” we created, we need to specify the “minx”, “minY”, “maxX”, and “maxY” in “CameraController.cs”. We set the variables in the inspector.

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

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

    Vector3 tempPosition;
    [SerializeField]
    float minX, minY, maxX, maxY;

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

        if (target == null) return;
        tempPosition = target.position;
        tempPosition.z = -10;

        if (target.position.x < minX)
            tempPosition.x = minX;
        if (target.position.y < minY)
            tempPosition.y = minY;

        if (target.position.x > maxX)
            tempPosition.x = maxX;
        if (target.position.y > maxY)
            tempPosition.y = maxY;

        transform.position = Vector3.Lerp(transform.position,
tempPosition, lerpSpeed * Time.deltaTime);
    }
}
```



30.) Next, we will add coins to the game. We take the coin sprite and we add it to the Hierarchy and name it “Coin”. Then we create an Empty GameObject named “Coins”

and we drag “Coin” under it. Then we attach a “Circle Collider 2D” to the coin. We change the radius to fit the coin sprite, we click “Is Trigger” and drag the coin to the “Prefab” folder. Then we duplicate multiple coins with “control + d” and we place them throught the map.



31.) We then go into “PlayerController.cs” and declare variables and create the function “OnTriggerEnter2D” so the player can collect coins

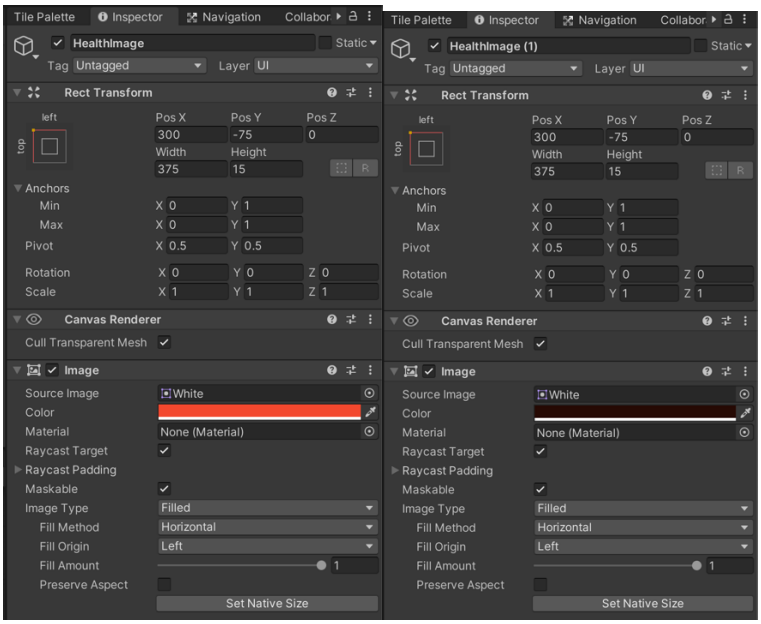
```
int coins = 0;

private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.CompareTag("Coin"))
    {
        coins++;
        collision.gameObject.SetActive(false);
    }
}
```

32.) Then we create the UI so we can see the number of coins the player has collected in game, as well as the player health.

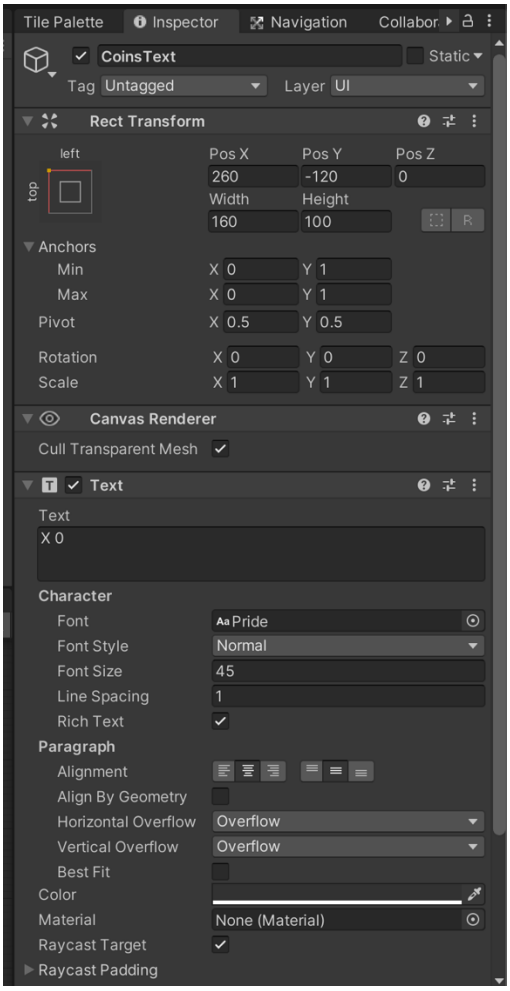
```
int coins = 0;

public Image healthImage;
public Text coinsText;
```

```
    // Update is called once per frame
    void Update()
    {
        inputX = Input.GetAxisRaw("Horizontal");

        if (inputX !=  0)
            playerRigidbody.AddForce(Vector2.right * inputX * speed *
Time.deltaTime);

        rend.flipX = (inputX < 0);

        RaycastHit2D hit = Physics2D.Raycast(transform.position,
Vector2.down, rayLength, wallLayer);

        if (hit.collider != null)
        {
            canJump = true;

        }

        if (canJump && Input.GetKeyDown(KeyCode.Space))
        {
            playerRigidbody.AddForce(Vector2.up * jumpHeight);
            canJump = false;
        }

        Debug.DrawRay(transform.position, Vector2.down *
rayLength);


        if (iframe > 0) iframe -= Time.deltaTime;

        // test Damage
        if (!hurt && Input.GetKeyDown(KeyCode.LeftControl))
            Damage(2);

        healthImage.fillAmount = Mathf.Lerp(healthImage.fillAmount,
health/maxHealth, Time.deltaTime * 10f);
        coinsText.text = "X " + coins.ToString();

        anim.SetBool("moving", inputX != 0);
        anim.SetBool("canJump", canJump);
        anim.SetBool("hurt", hurt);
```

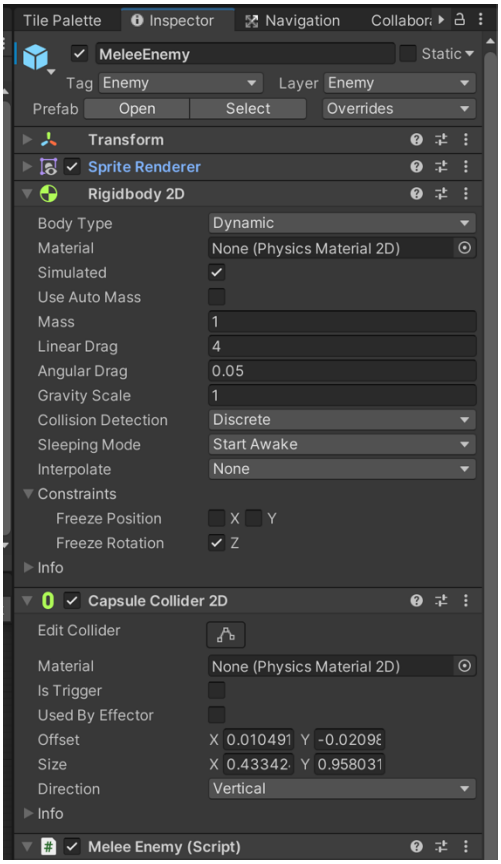
33.) Then we create an Image in the Hierarchy, we attach a white sprite to the source image, and we resize it, change the position. We make the color red. For “Image type” we select “Filled”, “Fill Method” we select “Horizontal”, and “Fill Origin” is “Left”. We then duplicated “Health Image” and we put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



34.) We then add “CoinImage” we resize it, change the position. The source Image is the coin sprite. We add coin text and we resize it, change the position. We set the font with our “Pride” asset, and we write in the text and we set “Horizontal/Vertical Overflow” as “Overflow”.



35.) We go into the Enemy sprite folder and drag “human1AttackAnim6”, and we change the name to “MeleeEnemy”. We attach a “Rigidbody2D” and freeze “Z”. Next, we attach “Capsule Collider 2D”. We change the collider size. We add the Enemy Layer to it and tag it as “Enemy”. We create “EnemyController.cs” parent class and “MeleeEnemy.cs” child class. We attach “MeleeEnemy.cs” to “MeleeEnemy” game object.



36.) In “MeleeEnemy.cs” we change “Monobehavior” to “EnemyController”. We then declare variables in “EnemyController.cs and make some accessible to its child classes. We create “OnEnable()” function and set “health” to “maxHealth”. Then in the “Awake()” we get the reference to our Rigidbody2D”. Then we create the function “public virtual void Move()/Chase()/Attack()/Damage(float amount)/Die()”. The child classes will implement these function differently. In the “Update()” function we make a switch statement to check the current state of the enemy and then we call the function that coincide with the action. Next we set the “healthImage” to keep track of enemy health.

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

public class EnemyController : MonoBehaviour
{
    public float maxHealth;
    protected float health;
    public Image healthImage;
    public float speed;
    public float runSpeed;
    public float chaseRange;
    public float attackRange;
    public enum enemystates { move, chase, attack}
    public enemystates currentState = enemystates.move;
    protected Rigidbody2D enemyRigidbody;
```

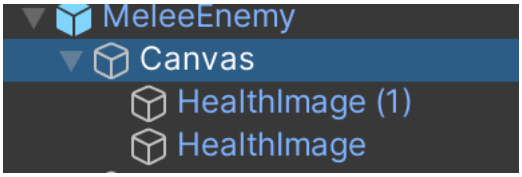
```
    void Awake()
    {
        enemyRigidbody = GetComponent<Rigidbody2D>();
        rend = GetComponent<SpriteRenderer>();
        player = FindObjectOfType<PlayerController>();
        anim = GetComponent<Animator>();
    }

    public virtual void Move() {}
    public virtual void Chase() {}
    public virtual void Attack() {}
    public virtual void Damage(float amount) {}
    public virtual void Die() {}


    // Update is called once per frame
    void Update()
    {
        rend.flipX = (direction == -1);
        switch (currentState)
        {
            case enemystates.move:
            Move();
            break;

            case enemystates.chase:
            Chase();
            break;

            case enemystates.attack:
            Attack();
            break;
        }

        if (attackCools > 0) attackCools -= Time.deltaTime;
        healthImage.fillAmount = health / maxHealth;
    }
}
```

37.) Then we go back into “MeleeEnemy.cs” and create the “Damage()” and “Die()”.

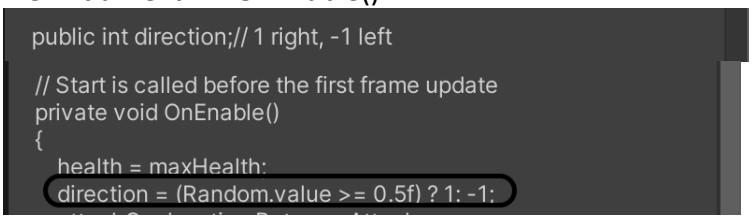
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MeleeEnemy : EnemyController
{
    public override void Damage(float amt)
    {
        health -= amt;
        if (health <= 0) Die();
    }

    public override void Die()
    {
        gameObject.SetActive(false);
    }
```

38.) We add the health bar for the enemy. We go into “Canvas” and change “Render Mode” to “World Space”. Then we set the position and size

39.) Then in the same way we did for the Player Game Object we create a healthImage Game Object, duplicate it, and put the duplicate above the original. For futher detail see #33.

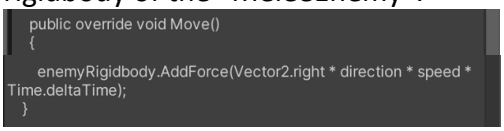


40.) Then we go back to “EnemyController.cs” and make the variable “direction” and we initialize it in “OnEnable()”.



```
public int direction;// 1 right, -1 left

// Start is called before the first frame update
private void OnEnable()
{
    health = maxHealth;
    direction = (Random.value >= 0.5f) ? 1: -1;
```

41.) In “MeleeEnemy.cs” we add the “Move()” function. We add a force to the rigidbody of the “MeleeEnemy”.



```
public override void Move()
{
    enemyRigidbody.AddForce(Vector2.right * direction * speed *
Time.deltaTime);
}
```

42.) Then we go into the “MeleeEnemy” Game Object and under “MeleeEnemy.cs” we set the variables.

43.) Now we want when the enemy hits the wall to turn around. We will implement this using raycast. We will do this in the “EnemyController.cs”.

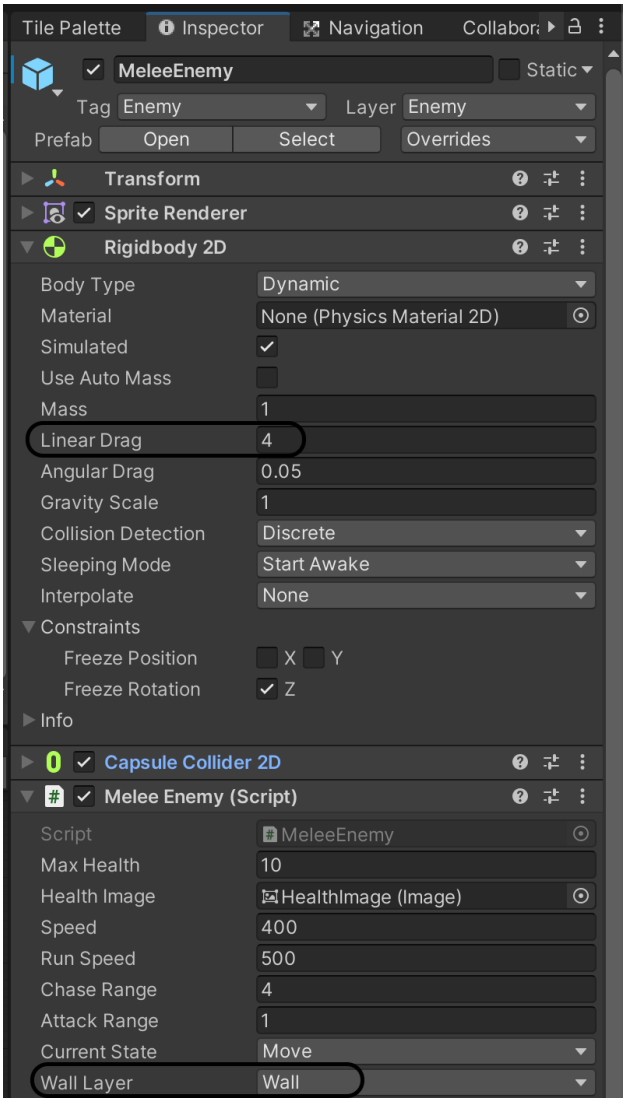
```
public LayerMask wallLayer;
public float rayLength;
```

44.) In the “MeleeEnemy.cs” in the move function we will get the hit collider. We will also add conditional statements for the enemy direction logic in association with colliders.

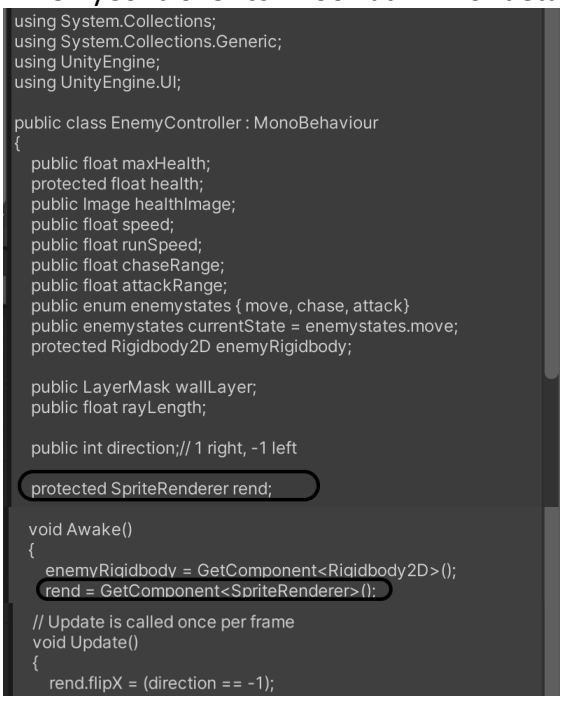
```
RaycastHit2D hit = Physics2D.Raycast(transform.position,
Vector2.right * direction, rayLength, wallLayer);
```

```
if (hit.collider != null) direction *= -1;//-1 - 1  -1 -> 1
```

45.) In the Inspector we set “Wall Layer” as “Wall”. We add linear drag to the enemy as well.

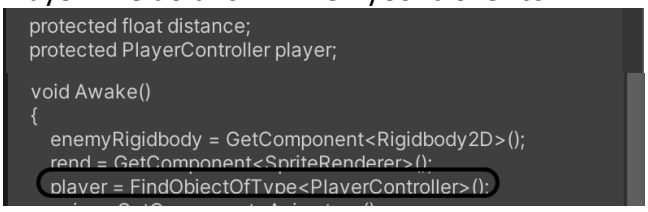


46.) We want to flip x for the enemy in the same way we did for the player in "EnemyController.cs". Look at #27 for details.



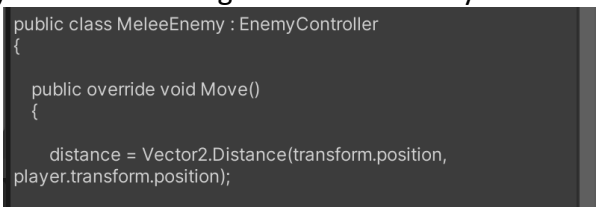
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class EnemyController : MonoBehaviour
{
    public float maxHealth;
    protected float health;
    public Image healthImage;
    public float speed;
    public float runSpeed;
    public float chaseRange;
    public float attackRange;
    public enum enemystates { move, chase, attack}
    public enemystates currentState = enemystates.move;
    protected Rigidbody2D enemyRigidbody;

    public LayerMask wallLayer;
    public float rayLength;

    public int direction;// 1 right, -1 left

    protected SpriteRenderer rend;

    void Awake()
    {
        enemyRigidbody = GetComponent<Rigidbody2D>();
        rend = GetComponent<SpriteRenderer>();
    // Update is called once per frame
    void Update()
    {
        rend.flipX = (direction == -1);
```

47.) When the Player is within chase range of the Enemy, the Enemy will chase the Player. We do this in "EnemyController.cs.



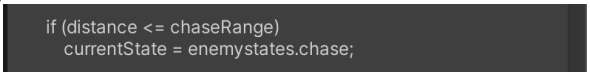
```
    protected float distance;
    protected PlayerController player;

    void Awake()
    {
        enemyRigidbody = GetComponent<Rigidbody2D>();
        rend = GetComponent<SpriteRenderer>();
        player = FindObjectOfType<PlayerController>();
```

48.) Then we go to "MeleeEnemy.cs and set "distance" in the "Move()" function.



```
public class MeleeEnemy : EnemyController
{

    public override void Move()
    {

        distance = Vector2.Distance(transform.position,
player.transform.position);
```

49.) Then we set conditional statement for when the enemy should chase the player.



```
        if (distance <= chaseRange)
            currentState = enemystates.chase;
```

50.) Then we create the "Chase()" function. We set the condition when a enemy should and should not chase.

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

    if (transform.position.x > player.transform.position.x)// player <-
enemy
        direction = -1;//enemy moves left
    else
        direction = 1;

    if (distance >= chaseRange)
        currentState = enemystates.move;
    if (distance <= attackRange)
        currentState = enemystates.attack;

    enemyRigidbody.AddForce(Vector2.right * direction *
runSpeed * Time.deltaTime);

}
```

51.) We go into “EnemyController.cs” and add variable that takes control how a much time needs to pass for a player to take get hurt again after taking damage.

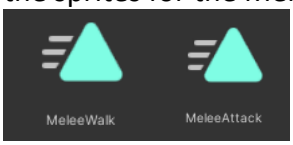
```
public float timeBetweenAttacks;
protected float attackCools;

// Start is called before the first frame update
private void OnEnable()
{
    health = maxHealth;
    direction = (Random.value >= 0.5f) ? 1: -1;
    attackCools = timeBetweenAttacks;
}

// Update is called once per frame
void Update()
{
    rend.flipX = (direction == -1);
    switch (currentState)
    {
        case enemystates.move:
        Move();
        break;

        case enemystates.chase:
        Chase();
        break;

        case enemystates.attack:
        Attack();
        break;
    }

    if (attackCools > 0) attackCools -= Time.deltaTime;
    healthImage.fillAmount = health / maxHealth;
}
```

52.) Then we complete the “Attack()” function in “MeleeEnemy.cs”

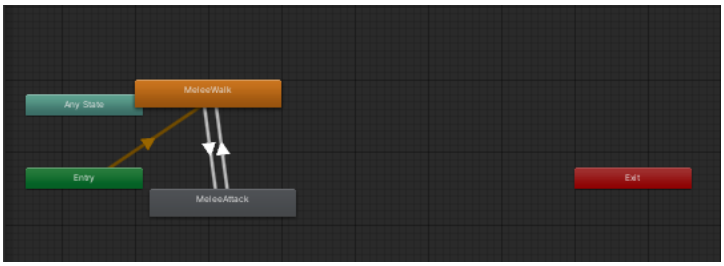
```
public override void Attack()
{
    if (attackCools < 0)
    {
        anim.SetBool("attack", true);
        Invoke("ResetAttack", 0.1f);
        attackCools = timeBetweenAttacks;
    }
    else
        currentState = enemystates.chase;
}
```

53.) Then we set the values for Time Between attacks in the “MeleeEnemy” GameObject.

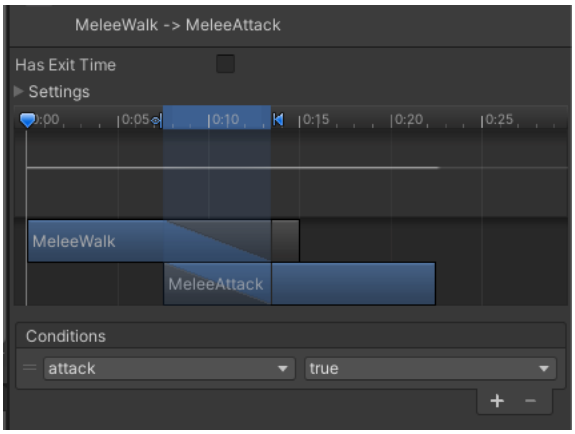
54.) We go into the Enemy folder and select the six sprites that belong to the MeleeEnemy Walking. Drag it to the Hierarchy where the “MeleeEnemy” Game Object is located and save it to the Animation folder. Save it as “MeleeWalk”. Do the same with the sprites for the MeleeEnemy attacking and save it as “MeleeAttack”.



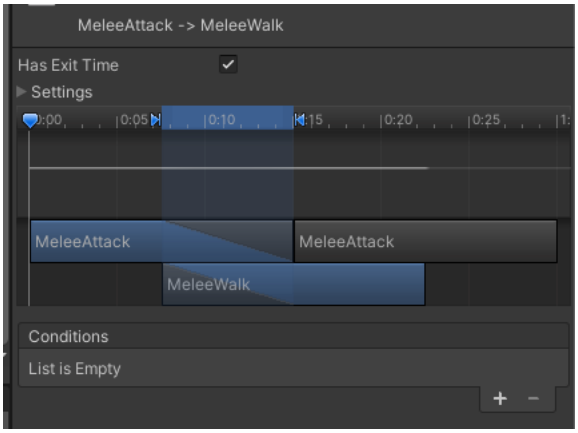
55.) Go into the “Animator” and make transitions between “MeleeWalk” and “MeleeAttack”.



56.) Add the parameter “attack” and make it a condition for “MeleeWalk -> MeleeAttack”. This will not use exit time.



57.) For “MeleeAttack -> MeleeWalk” “HasExitTime” will be used.



58.) Go back to “EnemyController.cs” we add “protected Animator anim;”. In the “Awake()” function we get a reference to the “Animator”.

```
protected Animator anim;

void Awake()
{
    enemyRigidbody = GetComponent<Rigidbody2D>();
    rend = GetComponent<SpriteRenderer>();
    player = FindObjectOfType<PlayerController>();
    anim = GetComponent<Animator>();
}
```

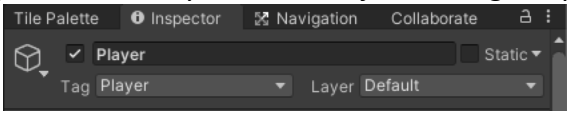
59.) In “MeleeEnemy.cs” in “Attack()” we set the parameter “attack”. Then we will Invoke “ResetAttack()”. In “ResetAttack()” we reset the parameter.

```
public override void Attack()
{
    if (attackCools < 0)
    {
        anim.SetBool("attack", true);
        Invoke("ResetAttack", 0.1f);
    }
    else
        currentState = enemystates.chase;
}

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

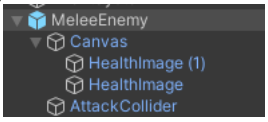
60.) If an enemy is on a platform we want the enemy to switch direction right before it reaches the edge. We will implement this in “MeleeEnemy.cs”

```
public class MeleeEnemy : EnemyController
{

    public override void Move()
    {

        distance = Vector2.Distance(transform.position,
player.transform.position);

        RaycastHit2D hit = Physics2D.Raycast(transform.position,
Vector2.right * direction, rayLength, wallLayer);
        RaycastHit2D hitDown = Physics2D.Raycast(transform.position,
Vector2.right * direction - Vector2.up, rayLength, wallLayer);
        if (hit.collider != null) direction *= -1;//-1 - 1  -1 -> 1
        if (hitDown.collider == null) direction *= -1;

        Debug.DrawRay(transform.position, Vector2.right * direction *
rayLength);
```

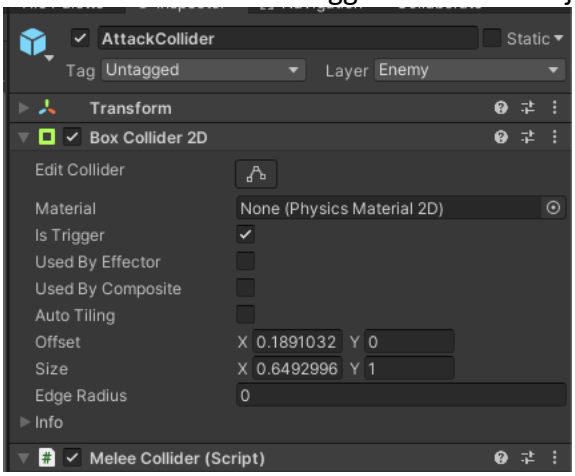
61.) In “Player” GameObject we tag the player as “Player”.



62.) We create “AttackCollider” GameObject under “MeleeEnemy”.



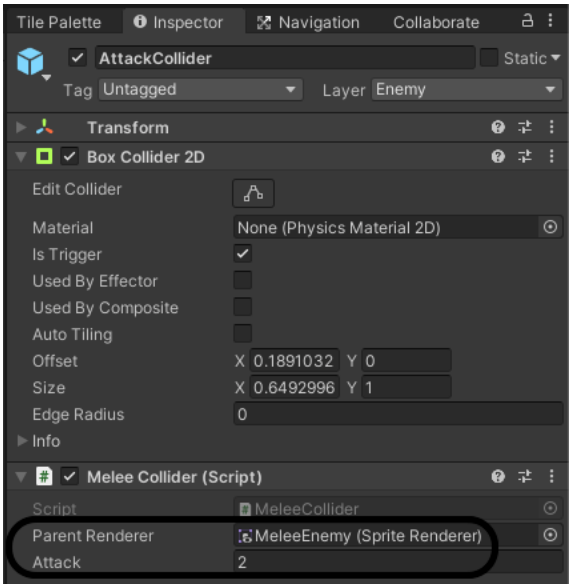
63.) We create “MeleeCollider” and attach it to “AttackCollider”. We attach a “Box Collider 2D” an set “Is Trigger”. We then adjust the box collider.



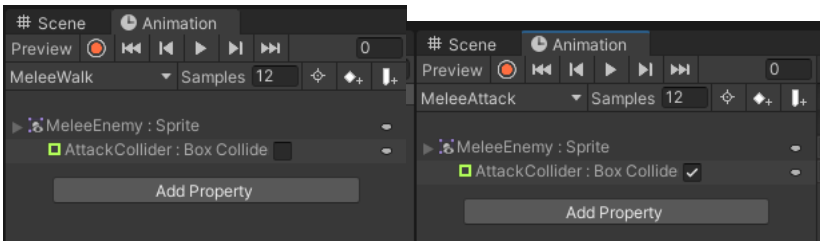
64.) We then go into “MeleeCollider.cs” and create variables as well as implement the functions “Update()” and “OnTriggerEnter/Stay2D()”.

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

public class MeleeCollider : MonoBehaviour
{
    public SpriteRenderer parentRenderer;
    public float attack;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        transform.localScale = new Vector3(parentRenderer.flipX? -1:1, 1, 1);
    }

    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);
    }
}
```

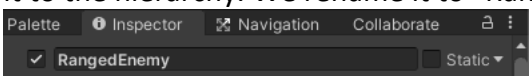
65.) In the “AttackCollider” GameObject we drag “MeleeEnemy” as the “Sprite Renderer”. We also set the Attack values.



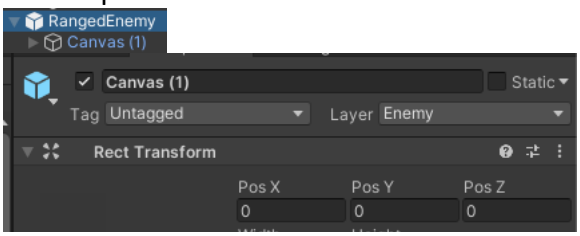
66.) When the enemy is walking it cannot damage player. In “MeleeWalk” animation we disable “Box Collider 2D”. In “MeleeAttack” animation we set the “Box Collider 2D”.



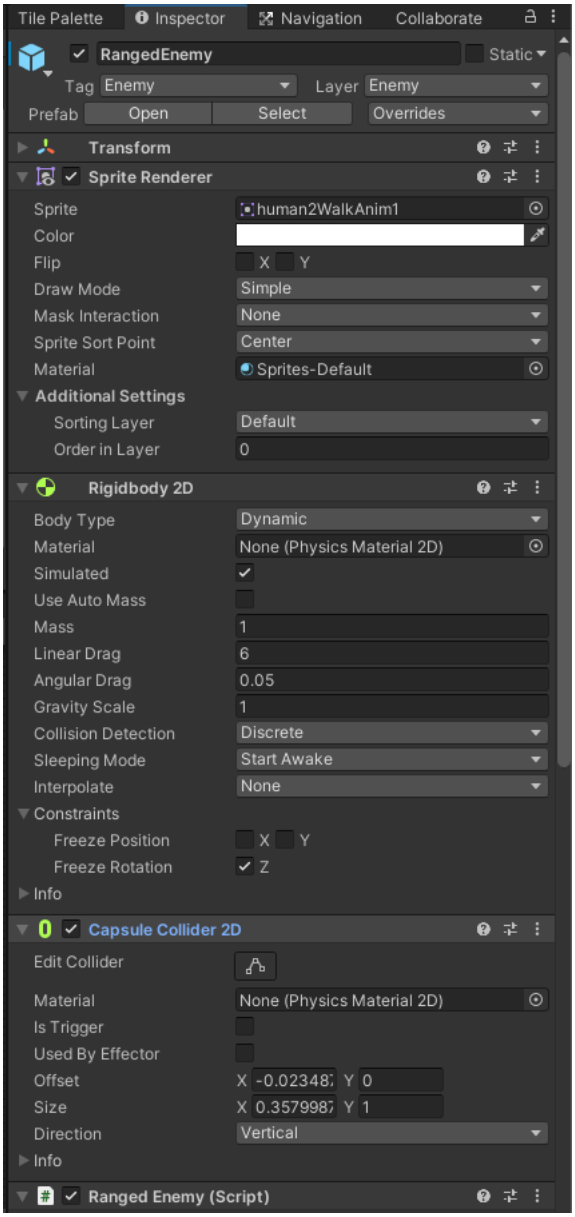
67.) We go into the Enemy sprite folder. We grab “human2walkAnim1” and we drag it to the hierarchy. We rename it to “RangedEnemy”.



68.) We copy “MeleeEnemy[‘s]” “Canvas” and we paste it under “RangedEnemy”. We set the position to 0

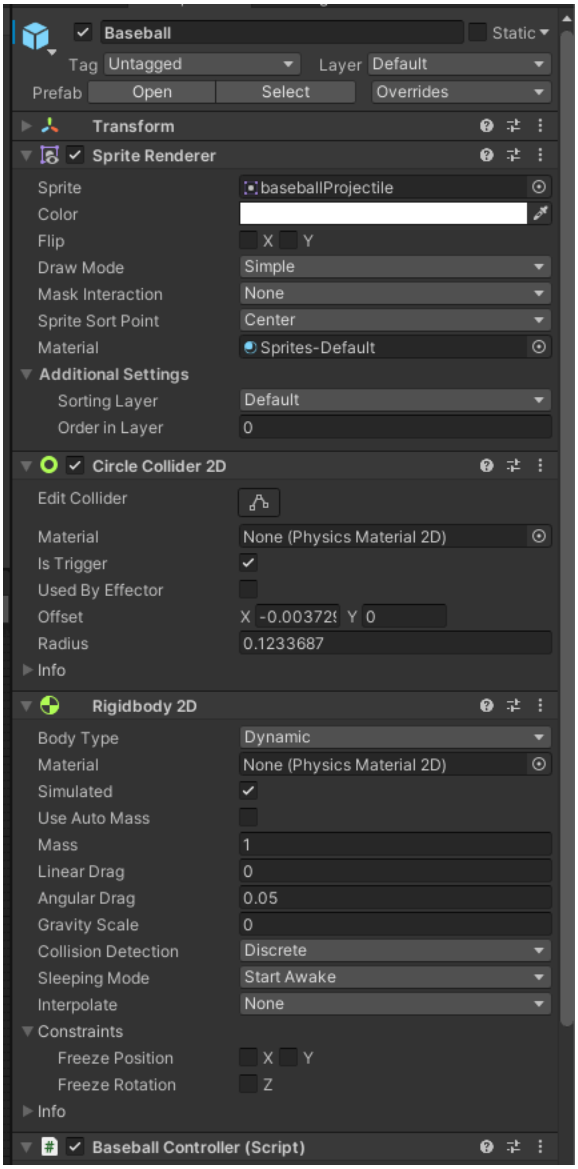


69.) We attach a “Rigidbody 2D” to “RangedEnemy” and we set “Linear Drag” and “Freeze Rotation Z”. Then we attach “Capsule Collider 2D”. Then we change the collider size. Lastly we create the script file “RangedEnemy.cs” and we attach it to “RangedEnemy” game object.



70.) We go into the “MeleeEnemy.cs” and copy everything over to “RangedEnemy.cs”.

71.) Now we get the “Baseball” sprite and drag it to the hierarchy. We attach a “Circle Collider 2D” and adjust the radius. We then attach a “Rigidbody 2D” and make the gravity zero. Then we create a “BaseballController.cs” and attach it to the “Baseball” game object.



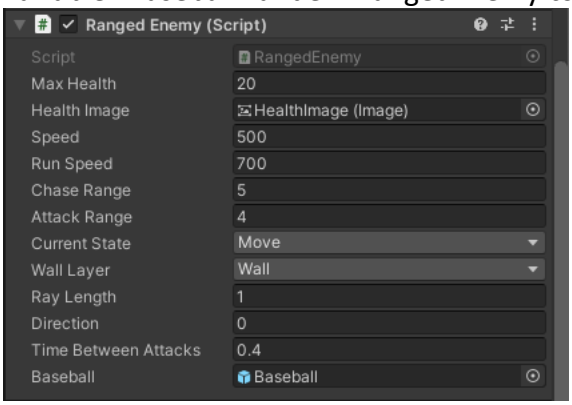
72.) The “BaseballController.cs” is very similar to our “BulletController.cs” from our “Tower Defense” microgame. We copy and paste from “BulletController.cs” to “BaseballController.cs” and make minor adjustments to fit our microgame #5.

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

public class BaseballController : MonoBehaviour
{
    public float speed;
    Rigidbody2D baseballRigidbody;
    public float damage;


    void Awake()
    {
        baseballRigidbody = GetComponent<Rigidbody2D>();
    }

    private void OnEnable()
    {
        baseballRigidbody.AddForce(transform.up * speed);
    }
    private void Disable()
    {
        gameObject.SetActive(false);
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            collision.GetComponent<PlayerController>().Damage(damage);
            Invoke("Disable", 0.001f);
        }

        if (collision.gameObject.CompareTag("Wall"))
            Invoke("Disable", 0.001f);
    }

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

73.) In “RangedEnemy.cs” under “Attack()” the enemy will throw the ball.

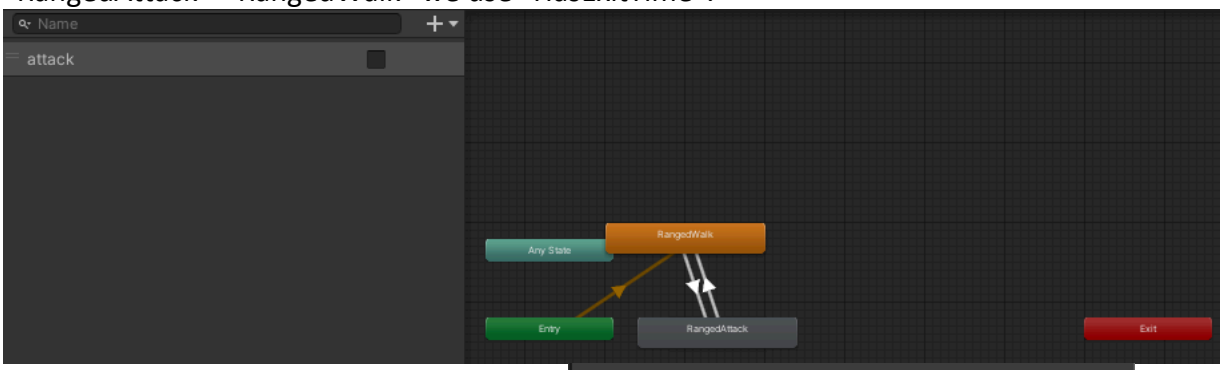
```
    public GameObject baseball;
    public override void Attack()
    {
        if (attackCools < 0)
        {
            anim.SetBool("attack", true);
            Vector3 dir = player.transform.position - transform.position;
            float angle = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg - 90;
            Instantiate(baseball, transform.position,
Quaternion.AngleAxis(angle, Vector3.forward));
            Invoke("ResetAttack", 0.1f);
            attackCools = timeBetweenAttacks;
        }
```

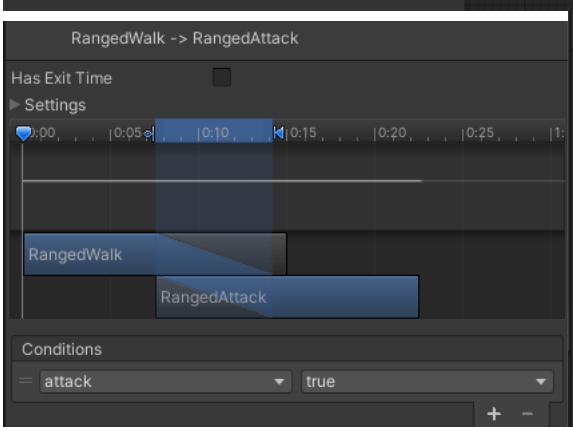
74.) In “RangedEnemy” Game Object we initialize all the variables under “RangedEnemy.cs”. Then you make “Baseball” GameObject a prefab and put it into the variable “Baseball” under “RangedEnemy.cs”.

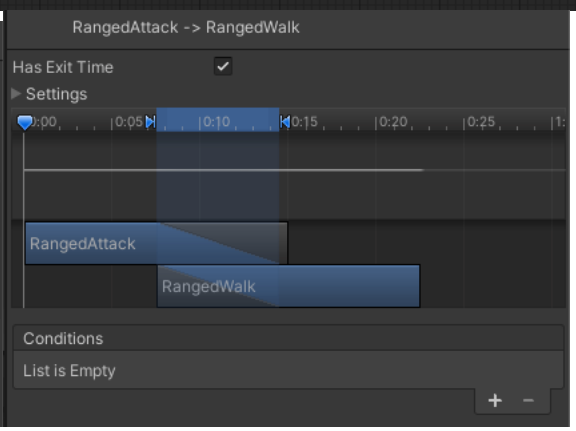


75.) Then we add the “RangedEnemy” walk and attack sprites, like we did for “MeleeEnemy” in #54. We save them to the animation folder under “RangedWalk” and “RangedAttack”.

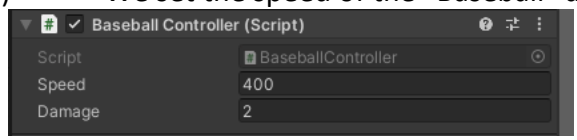
76.) We go into “Animator” and make transitions between “RangedWalk” and “RangedAttack”. We then create the variable “attack”. For “RangedWalk -> RangedAttack” we do not use exit time. We set the condition “attack” to be “true”. For “RangedAttack -> RangedWalk” we use “HasExitTime”.



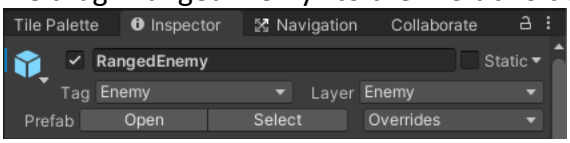




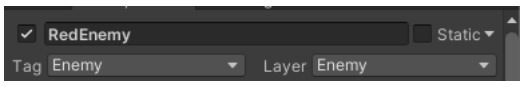
77.) We set the speed of the “Baseball” and its “Damage”



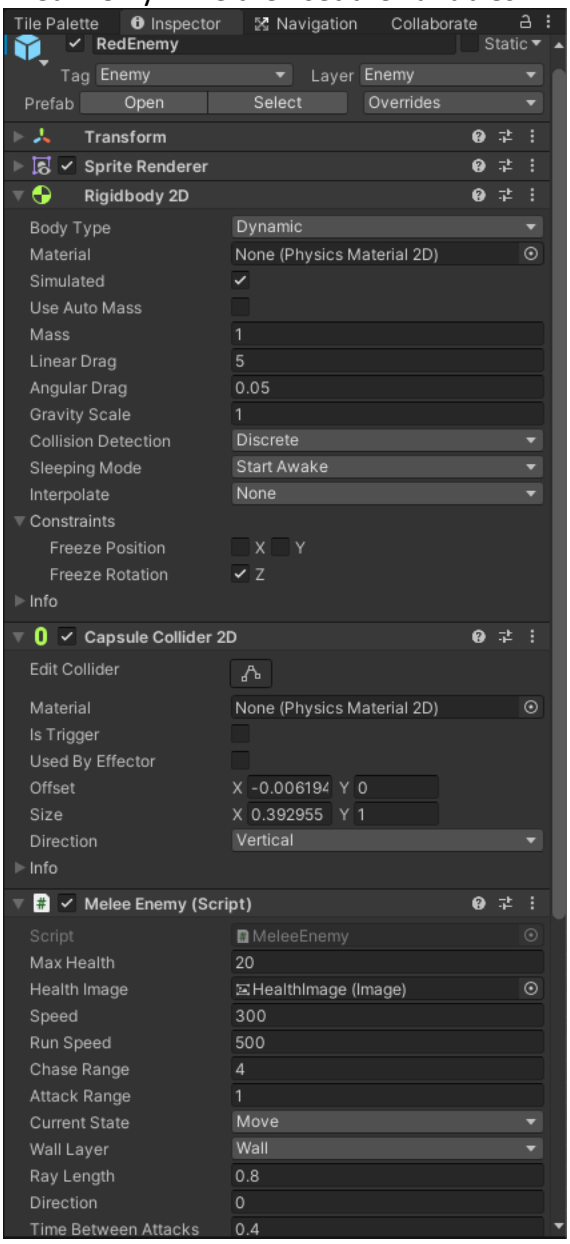
78.) In the “RangedEnemy” game object we set “Tag” and “Layer” as “Enemy”. Then we drag “RangedEnemy” to the Prefab folder.



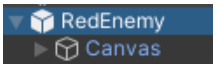
79.) We go into the Enemy sprites folder and we drag “human3AttackAnim6” into the hierarchy. Then we rename it “RedEnemy”. The we add the “Tag” and “Layer” as “Enemy”.



80.) We then attach “Rigidbody 2D” to the “RedEnemy” and add “Linear Drag” and we “Freeze Rotation Z”. Next, we attach a “Capsule Collider 2D” to the game object. Then we adjust the collider. Then we add the component “MeleeEnemy.cs” to “RedEnemy”. We then set the variables.

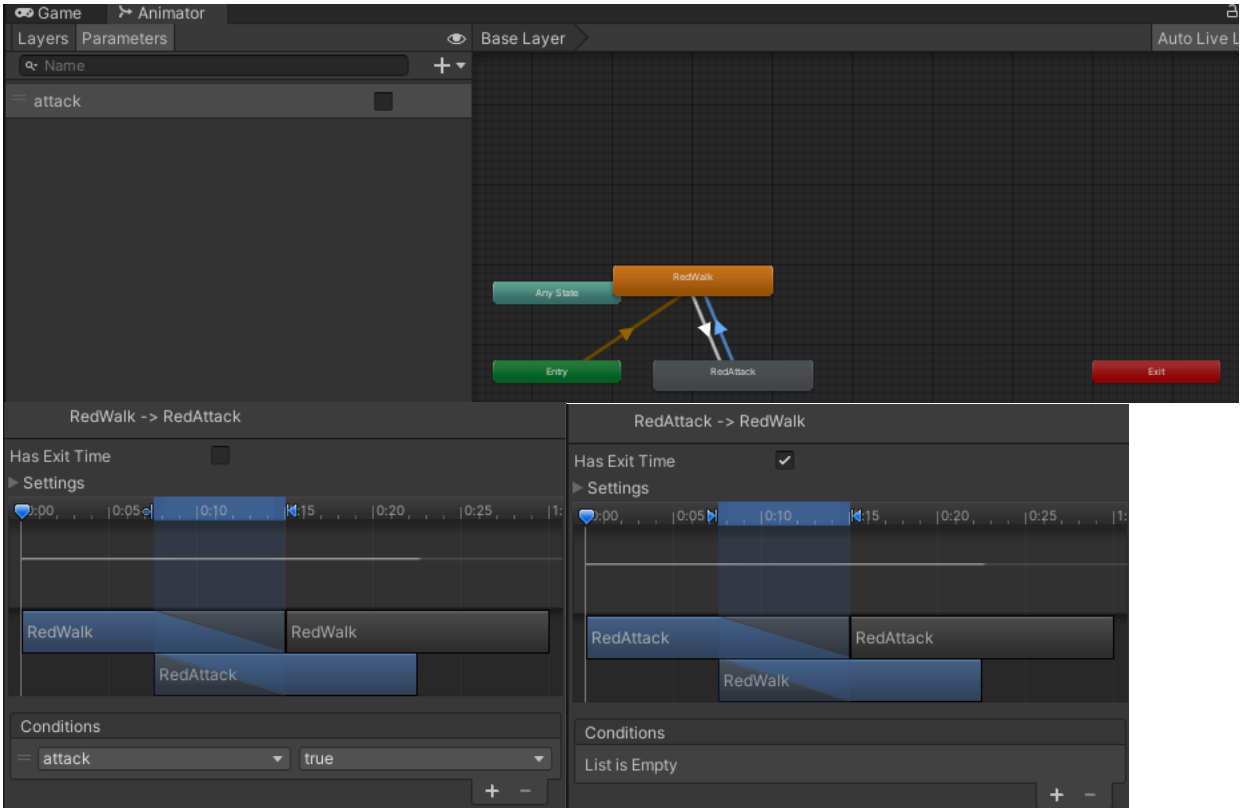


81.) Then we copy and paste the Canvas from “MeleeEnemy” to “RedEnemy” just like we did in #68.

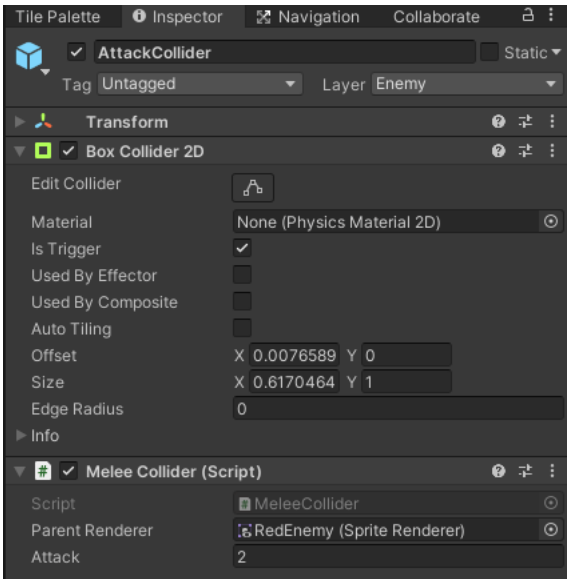


82.) Then we add the “RedEnemy” walk and attack sprites, like we did for “MeleeEnemy” and “RangedEnemy” in #54 & #75 respectively. We save them to the animation folder under “RedWalk” and “RedAttack”.

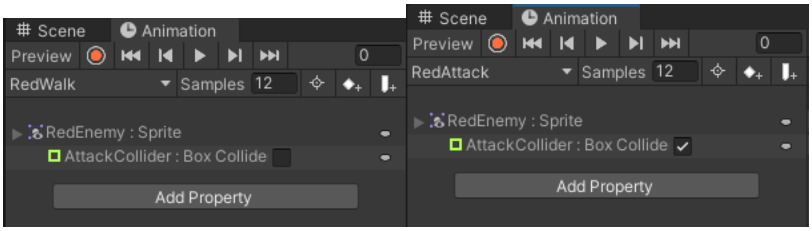
83.) Then we go into “Animator” and create the parameter “attack”. Next we make transitions between “RedWalk” and “RedAttack. “RedWalk -> RedAttack” we do not use “HasExitTime”. We set the condition of “attack” to “true. Then for “RedAttack -> RedWalk” we set use “HasExitTime”.



84.) Attach an empty game object “AttackCollider” to “RedEnemy”. In “AttackCollider” attach “Box Collider 2D” and set “IsTrigger”. Then edit the collider size. The attach “MeleeCollider.cs” to “RedEnemy”. Drag “RedEnemy” to the “Parent Renderer” variable, and set the “Attack” variable.

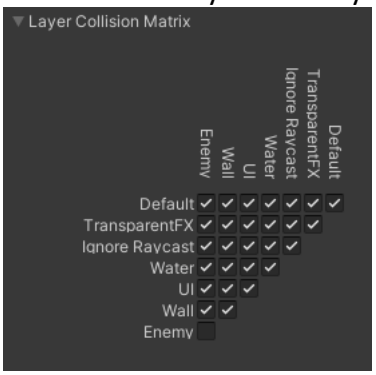


85.) Then in “Animation” for “RedWalk” we disable “Box Collider 2D” then in “RedAttack” we enable “Box Collider 2D”. Just like in #66



86.) We then save “RedEnemy” as a Prefab.

87.) Now go to “Edit -> Project Settings -> Physics 2D”. In the “Layer Collision Matrix” uncheck “Enemy” x “Enemy”. This will allow enemies to pass through one another.



88.) Next we want for the enemy to take damage if the player lands on the enemy’s head. We will implement this in “PlayerController.cs”. We add the function “OnCollision Enter2D()”.

```
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Enemy") &&
playerRigidbody.velocity.y < 0)
        {
            float boundsY =
collision.gameObject.GetComponent<SpriteRenderer>().bounds.size.y /
2;
            if(transform.position.y > collision.gameObject.transform.position.y
+ boundsY)
            {

playerRigidbody.AddForceAtPosition(-playerRigidbody.velocity.normali
zed * jumpHeight/2, playerRigidbody.position);

collision.gameObject.GetComponent<EnemyController>().Damage(5f);
            }

        }
    }
```

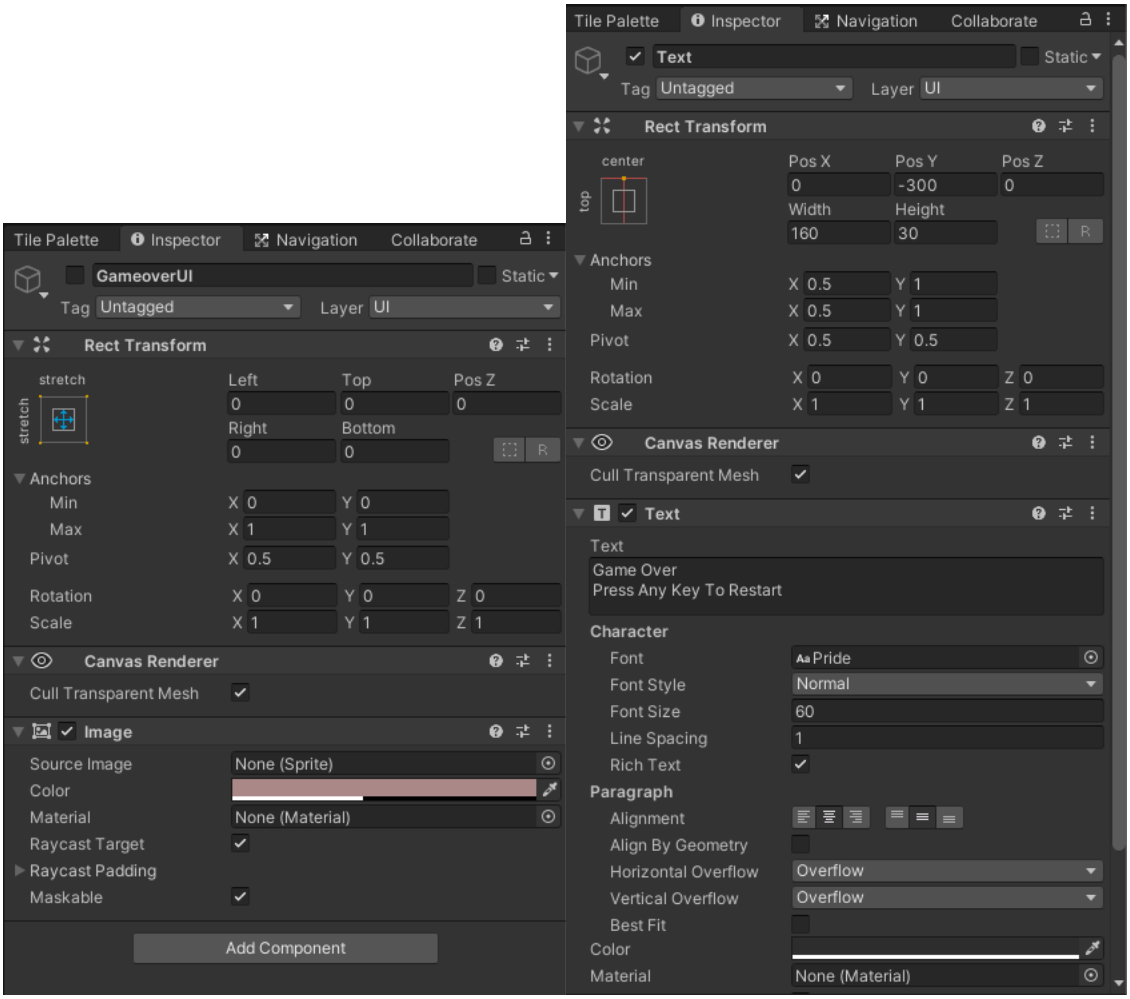
89.) We then go to the “Damage()” in “PlayerController.cs”. “If (health <= 0) {GameOver()}”. We create the “GameOver()” function. Then we add function “public GameObject GameoverUI” as well as “bool gameover”. We initialize “gameover” in “Awake()”. Then in the “Update()” if gameover” is true we pause the game.

```
using UnityEngine.SceneManagement;

    public GameObject gameoverUI;
    bool gameover;
```

```
    void Awake()
    {
        playerRigidbody = GetComponent<Rigidbody2D>();
        canJump = true;
        health = maxHealth;
        hurt = false;
        iframe = timeBetweenHurt;
        anim = GetComponent<Animator>();
        rend = GetComponent<SpriteRenderer>();
        gameover = false;
    }

    // Update is called once per frame
    void Update()
    {
        inputX = Input.GetAxisRaw("Horizontal");

        if (inputX !=  0)
            playerRigidbody.AddForce(Vector2.right * inputX * speed *
Time.deltaTime);

        rend.flipX = (inputX < 0);

        RaycastHit2D hit = Physics2D.Raycast(transform.position,
Vector2.down, rayLength, wallLayer);

        if (hit.collider != null)
        {
            canJump = true;

        }

        if (canJump && Input.GetKeyDown(KeyCode.Space))
        {
            playerRigidbody.AddForce(Vector2.up * jumpHeight);
            canJump = false;
        }

        Debug.DrawRay(transform.position, Vector2.down * rayLength);


        if (iframe > 0) iframe -= Time.deltaTime;

        // test Damage
        if (!hurt && Input.GetKeyDown(KeyCode.LeftControl))
            Damage(2);

        healthImage.fillAmount = Mathf.Lerp(healthImage.fillAmount,
health/maxHealth, Time.deltaTime * 10f);
        coinsText.text = "X " + coins.ToString();

        anim.SetBool("moving", inputX != 0);
        anim.SetBool("canJump", canJump);
        anim.SetBool("hurt", hurt);

        if (gameover && Input.anyKeyDown)
        {
            SceneManager.LoadScene("SampleScene");
            Time.timeScale = 1f;
        }
    }

    public void Damage(float amt)
    {
        if(iframe < 0)
        {
            health -= amt;
            hurt = true;
            Invoke("ResetHurt", 0.2f);
            if (health <= 0)
            {
                GameOver();
            }
            iframe = timeBetweenHurt;
        }

    }
```

```
    public void GameOver()
    {
        gameover = true;
        gameoverUI.SetActive(true);
        Time.timeScale = 0;
    }
```

90.) Under “Canvas” add “GameOverUI” and add “Text” under it. Change the position of “GameOverUI”, change color, set font to the “Pride” font. For “Horizontal/Vertical Overflow” choose “Overflow”. Then set the text. Uncheck “GameOverUI.



91.) Then go to the “Player” game object and set “GameOver UI” variable with the “GameOverUI” gameobject.