Part 1: The MVP

Created by Jesus:

- Section 1: Overview of Features
  The features include: A jumping cube(The player), ground floor, and platforms for the player, and camera control.

- Section 2: Resources
  Needed: Nothing!

- Section 3: Instructions on creating objects.
  Layer: Ground
  Main Camera:
    Size: 8
    Scale: 0.82x
    Position: 0, 0, -10
  Cubey:
    Rigidbody2D: Continuous, Never sleep, Interpolate.
    Box Collider 2D:
    GroundCheck
    HeadCheck
  Floor:
    Scale: X: 28.5, Y: 2
    Box Collider 2D
  BorderWall:
    Box Collider 2D
    Scale: X: 15.99999
    Position: -14.72, -0.048, 0.
    Rotation: 0, 0, -90
  Square:
    Box Collider 2D
    Scale: 8, 1, 1
- Section 4: Instructions on creating code.
  Player Code:

```
private float horizontal;
private bool isFacingRight = true;

[SerializeField]
private float speed;


[SerializeField]
private float jumpingPower;
```

```
        public Rigidbody2D rigidBody;
        public Transform groundCheck;
        public LayerMask groundLayer;

        public Transform headCheck;
        public LayerMask brickLayer;

    void Update()
    {
        horizontal = Input.GetAxis("Horizontal");

        if (Input.GetButtonDown("Jump") && isGrounded())
        {
            rigidBody.velocity = new Vector2(rigidBody.velocity.x, jumpingPower);
        }

        flip();
    }

    private void FixedUpdate()
    {
        rigidBody.velocity = new Vector2(horizontal * speed, rigidBody.velocity.y);
    }

    private bool isGrounded()
    {
        return Physics2D.OverlapCircle(groundCheck.position, 0.2f, groundLayer) ||
Physics2D.OverlapCircle(groundCheck.position, 0.2f, brickLayer);
    }

    private bool isCollidingWithBrick()
    {
        return Physics2D.OverlapCircle(headCheck.position, 0.2f, brickLayer);
    }

    private void flip()
    {
        if (isFacingRight && horizontal < 0f || !isFacingRight && horizontal > 0f)
        {
            isFacingRight = !isFacingRight;
            Vector3 localScale = transform.localScale;
            localScale.x *= -1f;
            transform.localScale = localScale;
        }
    }
```

Camera:

```
public Transform player;

void Update()
{
    if (player.transform.position.x > 0)
```

```
  {
    transform.position = new Vector3(player.transform.position.x, 0, 0) + new Vector3(0, 0, -10);
  }
}
```

Part 2: The State Machine

Created by Zoë:
Branches:
States:
The completed states branch, up-to-date with main.
PrsentationStage2:
Includes 1-MVP and the state machine.

Sections 1: Overview of Features
Features include: Animations for the player character, and (adding the animations for the enemies), The state machine, which allows for the implementation of states on any entity (enemies, player).

Section 2: Resources
Needed: A sprite sheet off a website, or from yourself.
Sample Sprites:
https://drive.google.com/drive/folders/11YfMGCbu_RgWLZ9RcvRERjxF53dKr0qL?usp=sharing
Sprite Creator: http://charas-project.net/charas2/index.php

Section 3: Instructions
Include the order of events that you need to take to add on to the previous build.

STATE MACHINE

Instructions: Adding on to the MVP.

- To start with, add all the assets you'll use in the process of adding the state machine. (to the proper folders)
  Entity.cs, EntityState.cs, EntityDeadState.cs, EntityJumpingState.cs, EntityRunState.cs, EntityIdleState.cs.

- Change the player type from Monobehavior to Entity. (this is to expand the use of states to any entity, including enemies and the player.)
  - `Public class Player : Entity`

- Move some of the variables up to the Entity.cs File:
  - `public float horizontal;`
  - `public Rigidbody2CD rigidBody;`
  - ```
    [SerializeField]
    public float speed;
    ```
  - ```
    [SerializeField]
    public float jumpingPower;
    ```
  - Remove these variables from the Player class.

- Add currentState variable to keep track of states.
    - `public EntityState currentState;`

- Add the states to the Entity.cs in the form of:
    - `public StateFileName stateName = new StateFileName();`
    - Add this function for each state:
        - `EntityIdleState idleState`
        - `EntityDeadState deadState`
        - `EntityJumpingState jumpingState`
        - `EntityRunState runState`

- Add in Entity.cs: Start(),
    - `currentState = idleState;`
      `currentState.EnterState(this);`
    - This is to set the current state to idle upon creating an Entity.

- Move to EntityState.cs, Create the lines inside as abstracts,
    - `public abstract class EntityState{`
      `public abstract void EnterState(Entity entity); }`
    - follow form for UpdateState, FixedUpdateState, OnCollisionEnter.

- Copy these functions in non-abstract form into the Entity{state}State files.

- Move to Player.cs. Add:
    - `currentState.UpdateState(this);` to Update();
      `currentState.FixedUpdateState(this);` to FixedUpdate();
      `currentState.OnCollisionEnter(this);` to OnCollisionEnter();

- Modify Update(); inside of the `if(Input.GetButtonDown);`
    - `currentState = jumpingState;`
      `currentState.EnterState(this);`

- Modifying the state files for functionality:
  Add to the EntityRunState.cs in UpdateState();
    - `If(entity.horizontal == 0f){`
      `entity.currentState = entity.idleState;`
      `entity.currentState.EnterState(entity); }`

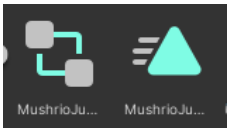
- Add to the EntityRunState.cs in FixedUpdateState();
    - Take this function from Player.cs in FixedUpdate(); (this is the function that causes the character to move right and left. We want to apply it to both the jumping state and the running state. (You don't move when you're idle… or dead)).
    - `entity.rigidBody.Velocity = new Vector2(entity.horiontal *entity.speed, entity.rigidBody.velocity.y);`
    - Remove this function from Player.cs in FixedUpdate();
- Add to the EntityJumpingState.cs in EnterState();

- Take this function from Player.cs in Update();
- `entity.rigidBody.velocity = new Vector2(entity.rigidBody.velocity.x, entity.jumpingPower);`

- Add to the EntityJumpingState.cs in FixedUpdateState();
  - This is the same function from the part just before it.
  - `entity.rigidBody.velocity = new Vector2(entity.horizontal * entity.speed, entity.rigidBody.velocity.y);`

That's it for the structure of the state machine, next up, player animation using it.

ANIMATION



(Left: Animation Controller, Right: Animation Clip)

- A couple ways to do this, You can use the animator, which includes a state machine, (show example here of using the unity animator state machine) or you can use the one we created ourselves, we're going to use our custom one.

- In Entity.cs, add the animator controller fields to pick what animation you'll use:
  - `public RuntimeAnimatorController Mushrio{action};`
  - You'll add this object for actions: Jump, WalkR, Idle, Dead

- Head to Unity and add these animations to the player character inspection menu (which should now have fields attached that you can add an AnimationController to.

- Find your sprite sheet or add it to assets if you haven't already, click on the object and change the "sprite mode" to "multiple".

- Use the "sprite editor" option on the object to enter sprite editor and use it to cut your sprite sheet into individual sprites.

  - There are a couple ways to cut these sprites.
    - Inside of the "sprite editor" button on the top left corner of the window, you can choose to use a custom outline, this can be time consuming depending on the complexity of the sprites you are using.
    - For our demo today we will leave the background of the sprites on. The sprite sheet from the website I gave out has sprites that are x: 24 y: 32, you can plug these numbers into the slice menu at the top after clicking type: "Grid by Cell Size"

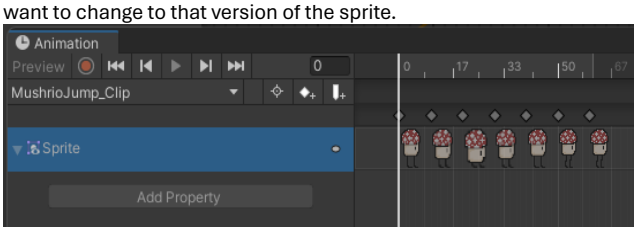
        - Before pressing “apply” your sprite sheet should be separated by red lines between each of the images.

- Add the main sprite you want to use to the “sprite” section of the player inspection window in the “sprite renderer” part.

- To create a new animation clip, go to “window” at the top of the screen, hover over “animation” and choose “animation”, click “create” and save the clip as “SpriteClip”.

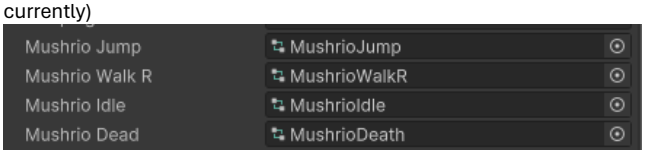
- Open the “SpriteClip” animation clip you just made and add the main sprite property of the player as a “property”.

    - Now click the small red record button in the top left corner and right click in the window on the right side to add keyframes. You can then add three keyframes, one for each frame of the walking animation.
        - (these are sprites 3,4,5 on the given sprite sheet website)
    - To change the sprites in the keyframes, you need to change the sprite on the player in the inspector menu and then double click on the frame you want to change to that version of the sprite.



    - In the inspector menu for the SpriteClip animation clip, set the wrap mode to “once” if you only want the animation to play once.

- To test this animation, after exiting the animation window, right click in the assets and create an animation controller, then add the clip to the properties on the right side.
    - Then add this animation controller to the RuntimeAnimationController properties on the player inspector window. (we can use all actions to test this because we do not have animations for a jump, idle, or death currently)



    - We also need to add the animation call to the run state,
      In the EntityRunState.cs script add:
        - ```
          Animator animator =
          entity.GetComponent<Animator>();
          animator.runtimeAnimatorController =
          entity.MushrioWalkR as RuntimeAnimatorController
          ```
          ```

Part 2.5: Title Card and Menus

Created by: Lacey
Branch: new-menus-lacey
Section 1: Overview of features (sample on screen)
- Main Menu Scene -> End of Game Scene -> Game Pause Button/Pause Menu

Section 2: Downloads for code-along (My sample folder you can use)
- Clickable buttons png(s) (choose one with multiple buttons, clicked and unclicked)
    - Great site for looking
- Background for buttons
    - I just looked on google images...
- Font(s) (I will be using two; title and body)
    - Free font site
- Unzip if necessary and put all these downloads in new folder Assets->Menu_Items

Section 3: Steps to implement:

Section 3A: MAIN MENU AND TITLE
- Open scenes, new scene for start menu, change background to a solid color
- Right click hierarchy-> UI->Canvas
    - Will probably be huge... select canvas and click f to fit to screen
    - Canvas render mode to screen space camera then insert your main camera
    - Say yes to vertex color to remove error
    - Scaler->UI scale mode->scale with screen size
    - Resolution 1920x1080 is HD
- Add an image (call it buttons, will be panel for buttons)
    - Add sprite for background of buttons
    - Show how it moves when you choose free aspect and adjust game menu... we need to anchor it
        - Click box with cross (anchor)
        - Canvas items have rect transform instead of normal transform
        - You can make your anchor the top of the screen so different aspect ratios won't make it move around, demonstrate
        - If you hold shift and click one of them, that sets pivot point (so if you rotate that's where it will pivot) and alt allows you to set the position or stretch
- Back to image
    - Set native size to get original ratio
    - If blurry: Click on sprite in asset menu, advanced, and change filter mode from bilinear to point... fixed!
    - If not working, make sure sprites are all texture type Sprite 2d and ui
    - When I resize, edges stretch too but I don't want that, let's go back to sprite editor
    - Adjust green lines so that all outside won't stretch then apply

    - Change image type to sliced
    - Now I can resize and just wanted parts stretch
- Title text
    - Ui text mesh pro, download if not already have it and add title, resize
    - We can add our font
    - Window->TMP->font asset creator
    - Select font and generate, save into assets folder
    - Now I can drag this font into font asset in text object
    - Now I can add and center text, add outline, underlay (drop shadow)
- Animating title
    - Window -> animation -> animation -> select title text and create
    - Double click animation to make it full screen if you don't see it right away
    - For example, I will rotate. Make first and last keyframes the same (-1.5) and add keyframe in the middle (1.5)
    - Click and drag keyframes to adjust speed, double click to add new keyframe
    - Preview
    - You can animate lots here! (size, spacing, etc)

Section 3B: MAIN MENU BUTTONS

- You can right click on the background image panel and add ui button, but it auto adds text and that might not be what you want… I am going to make a button from a sprite
    - Right click on panel that will hold buttons and insert UI-> image
    - Make sure sprites are all texture type Sprite 2d and ui
    - Hold shift to resize and keep aspect ratio
    - If blurry: Click on sprite in asset menu, advanced, and change filter mode from bilinear to point… fixed!
- A lot of the time, buttons are downloaded with multi buttons on the screen at once, so we need to edit them… change sprite mode to multiple in inspector
    - Open sprite editor and click slice, automatic and then adjust as needed (invert colors can help in top right)
    - Apply and close
    - Now you can click arrow by original sprite to see what it created
    - Add your image and then add button component to the image
    - Play, unity adds some shading already to indicate we are clicking but we will fix this (remove the shading)
- Button clicks
    - I want my button to change from one sprite to another when clicked
    - New script “clickyButton”

Add this:

```
using UnityEngine.EventSystems;
using UnityEngine.UI;
```

And this

```
public class NewBehaviourScript : MonoBehaviour, IPointerDownHandler,
IPointerDownHandler
```

If you are using visual studio, you can alt enter on pointer handlers and implement interface

If not, replace class with:

```
public class NewBehaviourScript : MonoBehaviour, IPointerUpHandler,
IPointerDownHandler
{
    [SerializeField] private Image img;
    [SerializeField] private Sprite up, pressed;
    // [SerializeField] private AudioClip compressClip, uncompressClip;
    // [SerializeField] private AudioSource audioSource;

    public void OnPointerDown(PointerEventData eventData) {
        img.sprite = pressed;
        // audioSource.PlayOneShot(compressClip);
    }

    public void OnPointerUp(PointerEventData eventData){
        img.sprite=up;
        // audioSource.PlayOneShot(uncompressClip);
    }
}
```

Brief run through of what this code does (We comment out audio stuff for later)

Save and exit

- Add script to button
    - Image is current image object, and sprites are the up and down buttons
- Play and test
    - If your pressed button resizes, go back to sprite editor and make sure the size of your pressed button sprite is the same size as the non-pressed sprite (change all pressed buttons to same size as their non-pressed counterparts)
    - You can remove the auto highlight features under the button component if you haven't already, play and test again
- Text on buttons
- You can make your play button have text in a png editor before uploading to Unity but I did not so I can show you how to add text.
    - Add TMP text as child of button and put in font and text
    - Put it where you want… test but it doesn't move when I click
    - To fix this add these to your button script:

```
using TMPro;

    [SerializeField] private TextMeshProUGUI playText;
    [SerializeField] private Vector3 moveOffset;
```

In down:

```
        if (playText != null) {
            playText.rectTransform.localPosition += moveOffset;
        }
```

In up:

```
        if (playText != null)
        {
            playText.rectTransform.localPosition -= moveOffset;
        }
```

Adjust the offset in your Unity to make the button match the text

- If text over your button keeps you from clicking:
    - Select text->tmp->extra settings->deselect raycast target and fixed!
- Now let's make button mean something

Add this to script:

```
using UnityEngine.SceneManagement;

    public void SwitchScene(string sceneName){
        SceneManager.LoadScene(sceneName);
    }
```

Could alternatively use int sceneID with build settings ID but string is easier to understand

Back to button

- Onclick-> + -> drag playbutton and select function switch scene and type scene to switch to
- Play and test… it won't work
    - To fix: file->build settings-> add open scenes so we build all scenes when we play
- Now test, it switches to the game
- Now grab the width and height of your play button! We are about to duplicate it as exit button
- Duplicate so we can make an exit button… rename new one and change the text
- Add grid layout group component to panel that contains buttons
    - Constraint: fixed column count (you can do row if you want)
    - Make cell size same as what you want your buttons to be (see grabbed earlier)
    - Adjust spacing
- Now adjust exit button source image and sprites and add this to script:

```
public void LeaveTheGame(){
        Application.Quit();
    }
```

- Select that function instead of switch scenes to exit button

Section 3C: GAME OVER MENU AND BUTTONS

- Duplicate play screen scene
    - Adjust new game over scene visually
    - Adjust buttons
    - Add third button that takes you back to main menu, change scene so it takes you to main menu
- Menu not opening when you test? Add to build assets
    - File->build settings-> add open scenes so we build all scenes when we play

- Now test and it should work

Section 3D: PAUSE BUTTON/MENU

- In main game scene, make canvas and pause button (follow steps in main menu to make image button that switches sprites but has no functionality yet) with the clicky button script for clicky-ness
- Make panel (pause menu) under canvas and adjust background overlay how you like
    - If the pause menu is behind other elements, decrease canvas-> plane distance (default is 100, you can change to 1)
- Add resume, home, and done buttons (copy from start screen) onto pause menu
    - Test the clicky-ness of buttons
    - Disable pause menu

Make new script for pause button (we do this so we don't have to add pause menu component to all other buttons)

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

public class PauseButton : MonoBehaviour
{
    [SerializeField] private GameObject pauseMenu;
    // [SerializeField] private AudioClip compressClip, uncompressClip;
    // [SerializeField] private AudioSource audioSource;

    public void SwitchScene(string sceneName){
        Time.timeScale=1f;
        SceneManager.LoadScene(sceneName);
    }

    public void PauseGame(){
        pauseMenu.SetActive(true);
        Time.timeScale=0f;
    }
    public void ResumeGame(){
        pauseMenu.SetActive(false);
        Time.timeScale=1f;
    }
}
```

- Add script to pause button
    - Add pause menu to script component
    - Add on click(), drag script in and select pause game function
    - Repeat for resume button but make it resume game

Test and Done!

Part 3: The Background Graphics

Created by: Sam

Branches: ***Parallax***

Section 1: Adding the Background sprites to Unity

- The first thing you need to do is to create your background sprites (or find a sprite online) and paste them in your Unity folder.
- This is a simple process where you paste the sprites that you have created into your assets folder in Unity. (*Always make a sub folder so that it is easy to locate different assets*)
- You can then drag the sprite from your folder to the Unity scene and Unity will automatically create a game object for you that has the component "*Sprite Renderer*"

Section 2: Adding the Parallax effect

2.1 What is the Parallax effect?

In simple terms: Parallax is an effect that makes different parts of the background move in different speeds.

For example:

When you look outside from the window of a moving car, you can see that the things that are near you appear in your sight for a shorter amount of time.

Let's say you see a farm and the moon while driving your car. You probably know that the farm is going to disappear in at most 10 seconds from your sight whereas the moon will still be in your sight. That is obviously not because the farm is moving away faster than the moon, but because the moon is so huge and so far away that its FOV (Field of View) is very large.

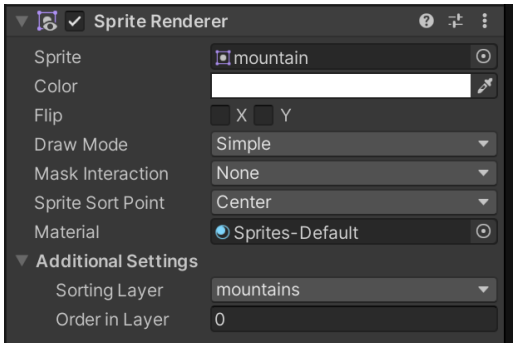
2.2 How do you make sure that the components have different ***Depths*** in a 2D game?

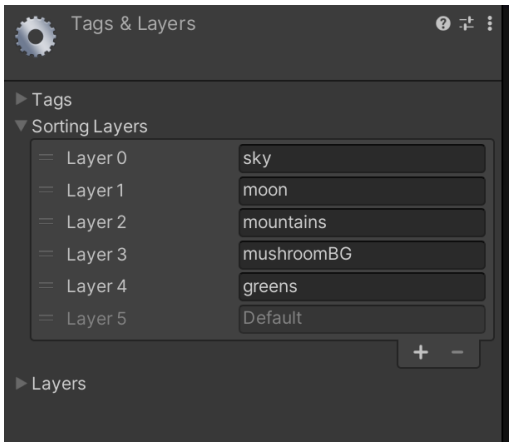
- The first thing you need to make sure is that you have multiple sprites for multiple layers of your background (*Different images/sprites for different things in the background*).
    - For example:

- A png image of mountains.
- A png image of Trees.
- A png image of moon.

– After you add multiple sprites in your Unity Assets folder, drag all of them to your Unity scene. (*This will create multiple game objects for each of your added sprites*)
– Now you need to start assigning them the order they need to be displayed on the screen.
    - For example:

      The trees should be in front of the mountains and the mountains should be in front of the moon.
– You can do this by creating different *sorting layers* and assigning them to your game objects. This is done through the *Sprite Renderer* component of the Game Object.





- Those are the sorting layers for our game, and as you can see the sky is the sorting layer that is the furthest behind and *Default* is the sorting layer that is the closest to the camera of the game.
- You can create or check all your *sorting layers* through *sorting layers --> Add Sorting Layer.*
    - Important:
        - "Sorting layers" and "Layers" are different things. Sorting layers are how objects are seen by the camera whereas layers are for grouping different objects that need the same functionalities like post-processing, same collision effects, etc.
            - Sorting Layers are visual
            - Layers are functional

2.3 Making the background move:

This is where all the coding is involved for the ***Parallax*** effect (*It's just one C# file).*

- First, you must create multiple copies of your added background GameObjects and extend them further by pasting them beside each other.

Note: The background must be long enough for the parallax effect to work. When you add new backgrounds to extend the environment, you must make sure that the edges of the background images match the heigh so it will not look off-putting.

- All you need to do is move the background components with the camera.
- Here's how you do that in C#:

```
public class Parallax : MonoBehaviour
{
    private float length, startpos;
    public GameObject cam;

    public float parallaxEffect;

    void Start()
    {
        startpos = transform.position.x;
        length = GetComponent<SpriteRenderer>().bounds.size.x;
    }

    void FixedUpdate()
    {
        float distance = (cam.transform.position.x * parallaxEffect);
```

```
        transform.position = new Vector3(startpos+distance,
transform.position.y, transform.position.z);
    }
}
```

length - A floating point number that is used as a GetComponent holder that points this script to a SpriteRenderer component. (*This is not mandatory, but it is good practice)*

startpos - A floating point number that stores the value of the starting "x position" of the background image.

cam – Camera

parallaxEffect – A floating point number that is used to change the speed at which the background moves. It is *Public* because we want to be able to change the speed from outside the script as well.

distance – A floating point number that checks how fast the background should move with respect to the camera. For example:

if(parallaxEffect = x)
{
    The background moves x times the speed of the camera..
}
so
{
    The lower the parallaxEffect, the slower the background
}

- After you're done making the script, drag the script into each of your backgrounds (*GameObjects)*
- You can now see two options inside the Parallax script in your GameObject component: Cam and Parallax Effect. Just drag your main camera's GameObject into the "Cam" option and just select a floating-point number for the "Parallax Effect" option which will move your backgrounds accordingly.

Section 3: Some Background sprite and video links

1. Free Background with multiple layers for Parallax:

   https://assetstore.unity.com/packages/2d/environments/free-2d-cartoon-parallax-background-205812

2. More backgrounds (*need to make an account even for free assets):*

   https://pressstart.vip/assets

3. Best Pixel Art Software:

   Paid: Aseprite

   Free: Pixel Studio

4. Best 2D Animation Software:

   Paid: Spine 2D, Adobe Animate

   Free: Krita

Part 4: Collectibles and HUD

Created by Upal:
https://docs.google.com/presentation/d/1Zl0rTbUZKtIY2HFIl0iF9f9rVZYD_MD_wrNsHUBQ0b4/edit?usp=sharing

Part 5: Enemies

Created by K:

Click me!

Commented [A(1]: Don't worry, just a google drive link ;D

Commented [ZA2R1]: So suspicious... I'm gonna call the cyber police on u. B)

Section 1 - 2:

1. Download the "MehrioDead.aseprite", "FinalBoss.prefab", "Enemy.prefab", "Mehrio.aseprite", and "Bar.png" from the Google Folder above
2. In your Unity Editor -> Assets -> create "Enemies" folder
3. Inside Enemies -> right click -> import custom -> select your three downloads

Section 3:

Let's start with the HealthBar which will decrease by some # when hit Enemy:

Border, sliders for green/yellow/red, locked to camera in left-top corner

1. Under "Canvas" (the same one that holds the pause button), Layer = UI
-> Add Canvas "HealthBarCanvas" (right click -> UI -> Canvas)
-> Uncheck Canvas Scaler
-> Rect Transform -> top left
-> Drag your HealthBarCanvas into the main Canvas
-> Shrink it down and move the image to the top left
-> Right click HealthBarCanvas -> UI -> Image
-> Click "Image"
-> add Source Image = Bar.png
-> rename to "Border"
-> On HealthBarCanvas right click to "create Empty"
-> Resize to fit Border
-> Rename to "HealthBar"
-> Drag Border into HealthBar
-> Right click HealthBar -> UI -> Image
-> Rename to "Fill"
-> Move it above Border
-> Rect Transform -> "alt" key -> bottom right button to scale it
(DO THE SAME FOR BORDER WITHOUT HOLDING ALT)
-> Select HealthBar
-> Add Slider component
-> Interactable unchecked
-> Transition = None
-> Navigation = None
-> Fill Rect = Fill
-> Max Value = 30 (or another number)
-> Whole Numbers checked

2. Create the HealthBar.cs script

-> On HealthBar
-> Add component
-> New script

3. Add objects to script
-> Slider = Health Bar
-> Gradient -> Fixed -> Add green/yellow/red colors
-> Fill = Fill
In later steps, we'll attach this HealthBar to the Player through the Health.cs script.

Next, the Player needs to be linked to the Health Bar with a Health script:

Player hits enemy script, health script to decrease health

1. Click on your mushroom MC player to make sure MushrioDead anim and Animator are good to go.
2. Create the two scripts:
-> PlayerHitsEnemy.cs
-> Health.cs
3. Add your game objects to the scripts
-> Nothing to be added for PlayerHitsEnemy.cs
-> For Health.cs
-> Max Health = 150
-> Current Health = 0 (or whatever default is)
-> Health Bar = HealthBar
-> Entity = Cubey (Player)

Alrighty, let's spawn enemies with the Spawner prefab:

Enemy.prefab used to spawn on platforms, random time, lifetime, speed

0. Make sure your Enemy.prefab has all the right components beforehand.
1. Right click anywhere in the SampleScene -> create Empty
-> Rename to "Spawner"
-> Choose any color diamond icon
-> Drag into Enemies folder to make it a prefab
2. Add SpawnEnemy.cs script
-> Change any of the min, max, speed, and lifetime values for more random spawns
3. Under the Spawn Enemy (Script) component
-> Obstacle Prefabs -> click "+"
-> Assets -> add "Enemy" prefab
-> Move your Spawner prefab into each platform that you want enemies to spawn on (far right corner of the squares).

Finally, enter the FinalBoss:

Enormous, jump on top of his head to kill him, death animation

1. Add an Enemy.prefab to the SampleScene.
-> On the Scene manager, make him a chonky boi
-> Scale: X = 60, Y = 57

-> Rename to “FinalBoss”
-> Create a tag called “FinalBoss”
-> FinalBoss Prefab loaded in “Prefab”
-> Sprite Renderer:
    -> MehrioRun_Frame_0 (he’s stationary)
-> Animator:
    -> Controller = MehrioDead.aseprite (not AC_Finalcontroller)
-> Rigidbody2D
    -> Body Type = Kinematic
    -> Collision Detection = Discrete
-> Box Collider 2D
    -> Move this so that the box collider is a horizontal platform on top of Mehrio’s head (for Player to hit and unalive him)
    -> *(optional) Create another empty game object and make a wall in front of Mehrio’s body. Have the Sprite Renderer be a Square, Tag = Brick, Layer = Brick, and add Box Collider 2D. This is to ensure Player cannot walk through Mehrio’s body.*

2. Add only the Player.cs script and modify/explain the changes in Health.cs script in order to make death animation work
-> Remove any empty scripts
-> Remove “Player Hits Enemy.cs” script

3. For the Player.cs script, you’ll need to add:
-> Rigid Body = FinalBoss
-> MushrioJump through Mushrio Idle = Mehrio.aseprite
-> Mushrio Dead = MehrioDead.aseprite
-> Ground Check = GroundCheck
-> Ground = Head
-> Brick Layer = Brick

Test and Run!

Errors?

Challenge: Fix MehrioDead’s off-centered animation.

Commented [A(3]: stash wd ID with recreated items: d754985

Part 6: Sounds

Created by Matthew:
Resources:

- Jump Sound Effect - https://pixabay.com/sound-effects/cartoon-jump-6462/
- Click Sound Effect (Classic Click) - https://mixkit.co/free-sound-effects/click/
- Background Music – YT Vid
    - YT to MP3 Converter - https://ytmp3.nu/CNtD/

TEMPLATE:

Created by: {insert name}
Branches:
Branch1: {description}
Branch2: {description}

Sections 1: Overview of the features you will be implementing, and preview of what you created (Show main for preview)

Section 2: resources
Include all resources you need that would help explain your process. You can find files as example for the other sections by using main

Section 3: instructions
Include the order of events that you need to take to add on to the previous build.