CODING STUFF

- https://drive.google.com/file/d/1oO-zJufUKMUYErlDx3guwEUhAQZxZzX8/view?usp=sharing

Language: C# (vs JavaScript of PlayCanvas)

- Variable - a name to which data can be assigned. The data can be accessed and modified at runtime (e.g enemyHealth = 15; , enemyHealth = enemyHealth - 5;)

```
Int enemyHealth = 15;
enemyHealth = enemyHealth - 5;

Float enemyHealth = 15f;

Bool enemyIsAlive = true;

String enemyType = “Zombie”;
```

certain data types require values to follow specific rules:
- STRING values must be enclosed in quotation marks " "
- FLOAT values must end with the letter 'f'
- a SEMICOLON ; is used to indicate the end of the line

- Methods - also known as functions, are blocks of code that perform specific tasks and can be called whenever needed (e.g play a sound effect, make a flash of light, apply 5 damage, to zombies within a 10ft radius, etc)
- Is a way to isolate code that performs a specific task, allowing it to be reused in multiple places
- It can accept input data (parameters) to work with
- It can either return no value (void) or return a specific piece of data

```
// damage instructions

Void Explode(float dmgAmnt, float dmgScope) {
```

// Normal Zombie Explode (10, 15);	// Boss Zombie Explode (5, 10);

Return data example:

```
String GenerateRandomEnemy(){
String randomEnemyName = ‘’;

// randomly add enemy entity name to randomEnemyName variable

Return randomEnemyName;
```

1. *Complete the method to calculate and display the sum of two integer variables*

```
Int num1 = 5;
Int num2 = 10;

// call the method below
DisplaySUmOfNumbers (num1, num2);

Void DisplaySumofNumbers (int a, int b) {
Int sum = a + b;
Print (“the sum is:” + sum);
}
```

2. *A method that returns a value in a function designed to perform a task and send a result back to the caller. Write a method that returns the message “Hello World” using a string variable*

```
// call the method below
String msg = DisplayTheMessage ();
Print (msg);

String DisplayTheMessage () {
String message = “Hello World!”;
Return message;
}
```

- Class - is a way to organize related methods and variables into a single unit, making it easier to manage and reuse code

```
Class Player {

// methods and variables
// belonging to player class

Void Movement (){
}

Void Shoot (){
}

Void WeaponChange () {
}
}
```

Inheritance
– enemy : // move, attach

– normal zombie : // walk
– boss zombie : // fly, resistances (but fish zombie : // swim)

docs.unity.com
https://docs.unity3d.com/2022.3/Documentation/ScriptReference/index.html

---

RequiredComponent should be written before declaring a class
Both of the given scripts are attached to the player

PLAYER MOVEMENT:

- To create plane

GameObject/Right Click > 3D Object > Plane

- To create a capsule as player

GameObject > 3D Object > Capsule
*if it is below the ground, check the height

(1, -1, 0)
X - Horizontal
Positive - Right
Negative - Left

Z - Vertical
Positive - Forward
Negative - Backward

GameObject > Align with view

- To create material

Right Click > Create > Material (Albedo Color)

- To create script

Right Click > Create > C# Script
*will open Microsoft Visual Studio but any other code editor will do

- The script name is also the class name; you cannot use space in the name
- Add component > Type in the name of component (Player) or Drag and Drop the script to the component in the hierarchy (Player)

*if the script does not appear in the component due to an error, close and open the file again

- To see the assigned keys for control: Edit > Project Settings > Input Manager

- To add RigidBody to Player

Add Component > Type in RigidBody (not the 2D)

*fixedDeltaTime is based on the specs of the computer
*can remove RequireComponent if all the required components are present; it is a back-up/just incase a component is removed

Player.cs

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

[RequireComponent(typeof(PlayerController))]

public class Player : MonoBehaviour
{

    public float moveSpeed = 5f;

    PlayerController controller;

    // Start is called before the first frame update
    void Start()
    {
        controller = GetComponent<PlayerController>();
    }

    // Update is called once per frame
    void Update()
    {
        Vector3 moveInput = new Vector3(Input.GetAxisRaw("Horizontal"), 0,
Input.GetAxisRaw("Vertical"));
        Vector3 moveVelocity = moveInput.normalized * moveSpeed;
        controller.Move(moveVelocity);
    }
}
```

PlayerController.cs

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

[RequireComponent(typeof(Rigidbody))]


public class PlayerController : MonoBehaviour
```

```
{
    Vector3 velocity;
    Rigidbody playerRigid;

    // Start is called before the first frame update
    void Start()
    {
        playerRigid = GetComponent<Rigidbody>();
    }

    public void Move(Vector3 _velocity)
    {
        velocity = _velocity;
    }

    // Update is called once per frame
    void FixedUpdate()
    {
        playerRigid.MovePosition(playerRigid.position + velocity * Time.fixedDeltaTime);
    }
}
```

Pseudocode – a way to describe a program's logic using plain, natural language

Example:
HOW TO CREATE OJ?
Prepare Materials: get a knife, cutting board, juicer, container, spoon, and a glass
Gather Ingredients: take 3-5 fresh oranges depending on desired juice quantity
Wash the oranges: Rinse each orange thoroughly under clean water to remove dirt
Cut the oranges: Place one orange on the cutting board. Use the knife to slice the orange in half. Repeat for all oranges
Juice the oranges: Take one orange half and place it on the juicer. Press and twist the orange half to extract the juice. Repeat for all orange halves
Filter the juice (optional): If you prefer pulp-free juice, use a strainer to filter out the pulp while pouring the juice into the container
Add optional ingredients: If desired, add 1–2 teaspoons of sugar for sweetness. Add a small amount of water if you want to dilute the juice
Mix the juice: Use a spoon to stir the juice until the sugar (if added) dissolves completely
Pour the juice: Carefully pour the juice into a clean glass
Serve: Serve the orange juice immediately for the best taste

Script > Player ; Script > Player Controller

Player Controller

Gun Model – needs to be a prefab! (drag and drop the model into the prefab folder then you can delete the gun in the hierarchy)

Player.cs (movement + camera look)

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

[RequireComponent(typeof(PlayerController))]

public class Player : MonoBehaviour
{
    public float moveSpeed = 5f;

    Camera viewCamera;
    PlayerController playerController;

    // Start is called before the first frame update
    void Start()
    {
        playerController= GetComponent<PlayerController>();
        viewCamera = Camera.main;
    }

    // Update is called once per frame
    void Update()
    {
        Vector3 moveInput = new Vector3(Input.GetAxisRaw("Horizontal"), 0,
Input.GetAxisRaw("Vertical"));
        Vector3 moveVelocity = moveInput.normalized * moveSpeed;
        playerController.Move(moveVelocity);

        Ray ray = viewCamera.ScreenPointToRay(Input.mousePosition);
        Plane groundPlane = new Plane(Vector3.up, Vector3.zero);
        float rayDistance;

        if (groundPlane.Raycast(ray, out rayDistance))
        {
            Vector3 point = ray.GetPoint(rayDistance);
            playerController.LookAt(point);
        }
    }
}
```

PlayerController.cs (movement + camera look)

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

[RequireComponent(typeof(Rigidbody))]

public class PlayerController : MonoBehaviour
{
    Vector3 playerVelocity;
    Rigidbody playerRigid;

    // Start is called before the first frame update
    void Start()
    {
        playerRigid = GetComponent<Rigidbody>();
    }

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

    }

    public void Move(Vector3 velocity)
    {
        playerVelocity = velocity;
    }

    public void FixedUpdate()
    {
        playerRigid.MovePosition(playerRigid.position + playerVelocity * Time.fixedDeltaTime);
    }

    public void LookAt(Vector3 lookPoint)
    {
    Vector3 correctedHeightPoint = new Vector3(lookPoint.x, transform.position.y, lookPoint.z);
    transform.LookAt(correctedHeightPoint);
    }
}
```

GunController.cs (adding gun to the player) should be attached to the player

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

public class GunController : MonoBehaviour
```

```
{
    public Transform weaponPosition;
    public Gun defaultGun;
    Gun equippedGun;

    // Start is called before the first frame update
    void Start()
    {
        if (defaultGun != null)
        {
            EquippedGun(defaultGun);
        }
    }

    public void EquippedGun(Gun gunToEquip)
    {
        if (equippedGun != null)
        {
            Destroy(equippedGun.gameObject);
        }
        equippedGun = Instantiate(gunToEquip, weaponPosition.position,
weaponPosition.rotation) as Gun;
        equippedGun.transform.parent = weaponPosition;
    }

    public void Shoot()
    {
        if (equippedGun != null)
        {
            equippedGun.Shoot();
        }
    }

}
```

Gun.cs

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

public class Gun : MonoBehaviour
{
    public Transform BulletPosition;
    public BulletProjectile bulletProjectile;
    public float bulletSpeed = 35f;
    public float bulletInterval = 100f;

    float nextShotTime;
```

```
    public void Shoot()
    {
        if(Time.time > nextShotTime)
        {
            nextShotTime = Time.time + bulletInterval/1000;
            BulletProjectile newProjectile = Instantiate(bulletProjectile, BulletPosition.position,
BulletPosition.rotation) as BulletProjectile;
            newProjectile.SetSpeed (bulletSpeed);
        }

    }
}
```

BulletProjectile.cs

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

public class BulletProjectile : MonoBehaviour
{
    float speed = 10f;

    public void SetSpeed(float newSpeed)
    {
        speed = newSpeed;
    }
    // Update is called once per frame
    void Update()
    {
        transform.Translate(Vector3.forward * Time.deltaTime * speed);
    }
}
```

- How to install AI Navigation?

Windows > Package Manager > Packages: Unity Registry > Search: AI Navigation

- How to Set a Mesh Surface?

Click on the Root > Game Object > AI > NavMesh Surface >
Agent Type: Humanoid:
Default Area: Walkable
Use Geometry: Render Meshes
> Bake

*The blue on the floor is the walkable area

- How to Rebake a walkable area?

Hide Character (in the actual object, not eye) > Clear > Rebake

- Add to Enemy

Get Component > Add: NavMeshAgent

- Add to Player

Player Object > Tag: Player

BulletProjectile.cs (with shooting target)

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

public class BulletProjectile : MonoBehaviour
{
    public LayerMask collisionMask;
    float speed = 10f;

    public void SetSpeed(float newSpeed)
    {
        speed = newSpeed;
    }
    // Update is called once per frame
    void Update()
    {
        float moveDistance = speed * Time.deltaTime;
        CheckCollisions(moveDistance);
        transform.Translate(Vector3.forward * moveDistance);
    }

    void CheckCollisions (float moveDistance)
    {
        Ray ray = new Ray(transform.position, transform.forward);
            RaycastHit hit;

        if (Physics.Raycast(ray, out hit, moveDistance, collisionMask,
QueryTriggerInteraction.Collide))
        {
            OnHitObject(hit);
        }

        void OnHitObject(RaycastHit hit)
        {
           // print(hit.collider.gameObject.name);
            GameObject.Destroy(gameObject);
```

```
            GameObject.Destroy(hit.collider.gameObject);
        }
    }
}
```

Enemy.cs

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

[RequireComponent (typeof(UnityEngine.AI.NavMeshAgent))]

public class Enemy : MonoBehaviour
{

    UnityEngine.AI.NavMeshAgent enemyAI;
    Transform target;

    // Start is called before the first frame update
    void Start()
    {
        enemyAI = GetComponent<UnityEngine.AI.NavMeshAgent>();
        target = GameObject.FindGameObjectWithTag("Player").transform;

        StartCoroutine(UpdatePath());
    }

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

    }

    IEnumerator UpdatePath()
    {
        float refreshRate = 1;

        while (target != null)
        {
            Vector3 targetPosition = new Vector3(target.position.x, 0, target.position.z);
            enemyAI.SetDestination(targetPosition);
            yield return new WaitForSeconds(refreshRate);
        }
    }
}
```

Website for Animation: Mixamo

- Input Animation on Stage

Create > Animation Controller > Drag the clips (rigged animation) > Right click on the clip > Make transition
*Checked ‘Has Exit Time’ = will finish the animation length and complete everything
*UNchecked ‘Has Exit Time’ = will finish the first animation only then follow conditions

- Animation Panel Thing (I FORGOT WHAT YOU CALL IT LOL)

Parameters > Click on + sign > Bool > Inspector: Conditions > Click on + sign > Condition the chosen Bool to T or F

On Entity: Add Animator Component > Drag Controller Asset to Controller Component

- Use the animated model, not just the regular prefab

switchAnim.cs (add script to enemy entity)

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

public class switchAnim : MonoBehaviour
{
    Animator femaleAnimator;

    // Start is called before the first frame update
    void Start()
    {
        femaleAnimator = GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Q))
        {
            femaleAnimator.SetBool("isWalking", true);
            femaleAnimator.SetBool("isIdle", false);
        }
        if (Input.GetKeyUp(KeyCode.Q))
        {
            femaleAnimator.SetBool("isWalking", false);
        }
```

```
        if (Input.GetKeyDown(KeyCode.E))
        {
            femaleAnimator.SetBool("isDead", true);
            femaleAnimator.SetBool("isIdle", false);
        }
        if (Input.GetKeyUp(KeyCode.E))
        {
            femaleAnimator.SetBool("isDead", false);
        }
        if (Input.GetKeyDown(KeyCode.X))
        {
            femaleAnimator.SetBool("isAttack", true);
            femaleAnimator.SetBool("isIdle", false);
        }
        if (Input.GetKeyUp(KeyCode.X))
        {
            femaleAnimator.SetBool("isAttack", false);
        }
    }
}
```

- 2D Screens
- Save in the same folder as the actual game scene

Create New Scene > Hierarchy: UI > Canvas > Add UI - Image

- Canvas Settings:

Render Mode: Screen Space - Overlay

- Canvas Scaler:

UI Scale Mode: Scale with Screen Size
Reference Resolution: X - 1920 , Y - 1080
Screen Match Mode: Match Width Or Height

- Make Outsourced Image into Sprite

Import Outsourced Image >

- To add an Outsourced Image to the Entity

Image > Texture Type: Sprite (2D and UI) > Image: Source Image - Select the Screen Sprite Asset

- To add Button/s

Hierarchy: UI > Button (TextMeshPro)
*import everything

- MAKE SURE TO DO THIS!

File > Build Settings > Add/Drag and Drop Scenes in ‘Scenes in Build’
*0 = first scene to load, menu should always be first layer

- To ‘Continue to Game’ Button

Add component > Script >

playGame.cs (add to ‘continue to game’ button)

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

public class playGame : MonoBehaviour
{
   public void LoadGame(string sceneName)
    {
        sceneName = "SampleScene";
        SceneManager.LoadScene ("SampleScene");
    }
}
```

- To add in-game gameBar

Canvas > Create Empty Entity: gameBar > Inside Entity, Insert UI: Panel

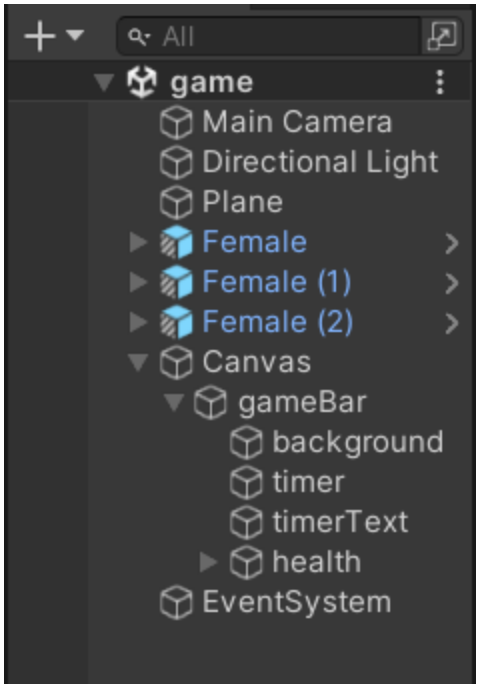
*Shift + Alt to set pivot and pivot position

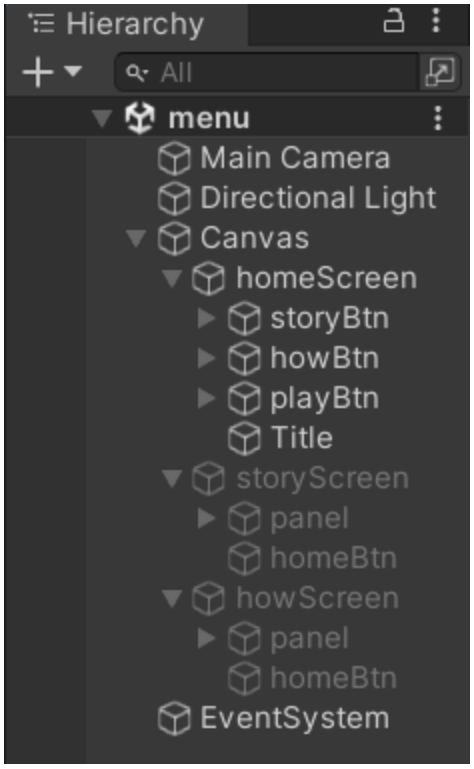
Time code attach to ‘canvas’
Timer display: Drag and Drop Text Asset

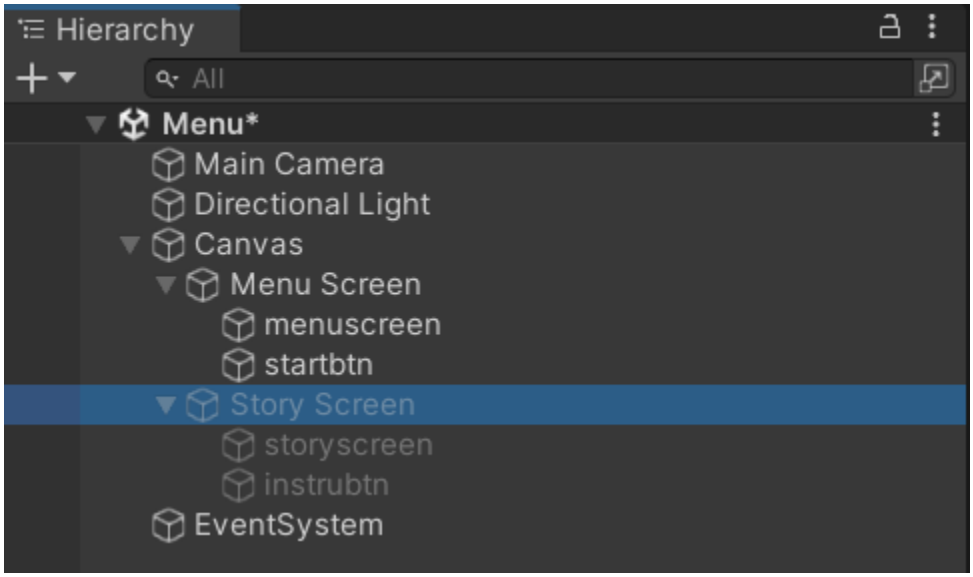
timerCode.cs (attach to ‘canvas’)

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

public class timerCode : MonoBehaviour
{
    [SerializeField] TMP_Text timerDisplay;
    public float TimerLeft;
    public bool TimerOn = false;

    // Start is called before the first frame update
    void Start()
    {
```

```
        TimerOn = true;
        timerDisplay.text = TimerLeft.ToString();
    }

    // Update is called once per frame
    void Update()
    {
        if (TimerOn)
        {
            if(TimerLeft > 0)
            {
                TimerLeft -= Time.deltaTime;
                updateTimer(TimerLeft);
            }
            else
            {
                Debug.Log("TIME IS UP!");
                TimerLeft = 0;
                TimerOn = false;
            }
        }
    }

    void updateTimer(float currentTime)
    {
        currentTime += 1;

        float minutes = Mathf.FloorToInt(currentTime / 60);
        float seconds = Mathf.FloorToInt(currentTime % 60);

        timerDisplay.text = string.Format("{0:00} : {1:00}", minutes, seconds);
    }
}
```


All
game
Main Camera
Directional Light
Plane
Female
Female (1)
Female (2)
Canvas
gameBar
background
timer
timerText
health
EventSystem


Hierarchy
All
menu
Main Camera
Directional Light
Canvas
homeScreen
storyBtn
howBtn
playBtn
Title
storyScreen
panel
homeBtn
howScreen
panel
homeBtn
EventSystem


Hierarchy
All
Menu*
Main Camera
Directional Light
Canvas
Menu Screen
menuscreen
startbtn
Story Screen
storyscreen
instrubtn
EventSystem