Goals:

Create an Agent that utilizes box and circle collider for proximity check for the following reasons:

1- Hiding between narrow objects.
2- Coming in contact with the target enemy, floating objects, item to be collected.
3- To limit the distance between agent and target so that they don't overlap.
4- AutoSpawn Objects onto the scene
5- Player/agent re-spawn if killed by target enemy.

Technologies, Tools and Resources used:

- Visual Studio IDE
- C# used instead of python as I used C# for my Custom Project.
- YouTube Tutorials
- Unity

Tasks Undertaken:

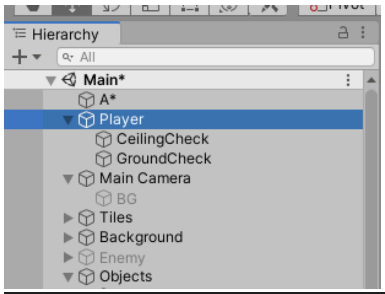
1. Ground Check and Ceiling Check implemented so that it detects if the object can fit or pass between the objects, used collider Method with height of character/agent and gap between objects to determine player height when in crouching state as it has the ceiling check over player and if that collides with the object then no space to pass between or hide between the object.
2. Use Box Collider with Circle Collider to limit the proximity it can detect the player within plus helps to avoid them from overlapping as used bodies to determine that hinderance exist.
3. Used Instantiate Object in Unity while considering the scene width and height the object can auto spawn into.
4. If enemy/eagle attacks by colliding in with the player it damages the player and thus the Player HP decreases once reached zero, an empty object is used as a checkpoint for the Player to respawn from.

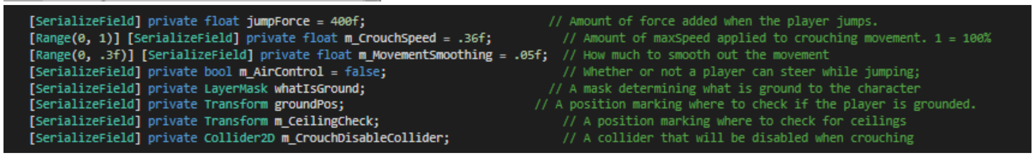
How Each Tasks undertaken above as well as the ones listed in spike extensions on doubtfire relates to ILO

ILO believed to be implemented or used in this Game spike extension report	How they relate
Discuss and implement software development techniques to support the creation of AI behavior in games	Agile backlog, GitHub used for project repository, committing changes, a small map of how the AI behavior should be like and what it could support to extend its intended use.
Understand and utilize a variety of graph and path planning techniques	Used A* algorithm for path finding towards the intended target and can be further used for patrolling behavior
Create realistic movement for agents using steering force models	Used the seeker behavior to seek enemy and close in, further implementation can be done to flee from enemy.

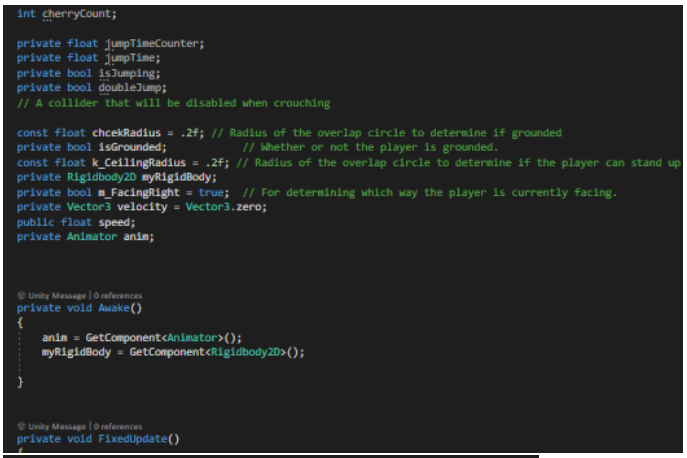
Output Observations and what we found out:

1. Empty Objects created over player to act as Top shield check and bottom shield check, so they can be used to decide whether player can crouch between the objects plus these were also used to decide if player can transition to Jumping state if in crouch state and not between Objects.

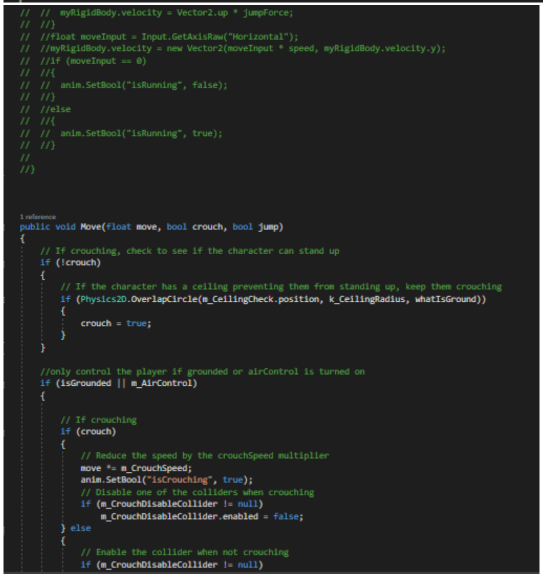




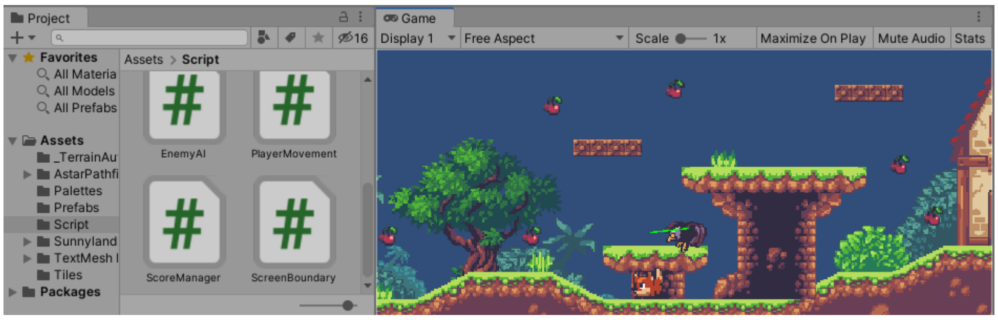
```
[SerializeField] private float jumpForce = 400f;                          // Amount of force added when the player jumps.
[Range(0, 1)] [SerializeField] private float m_CrouchSpeed = .36f;          // Amount of maxSpeed applied to crouching movement. 1 = 100%
[Range(0, .3f)] [SerializeField] private float m_MovementSmoothing = .05f;  // How much to smooth out the movement
[SerializeField] private bool m_AirControl = false;                         // Whether or not a player can steer while jumping;
[SerializeField] private LayerMask whatIsGround;                          // A mask determining what is ground to the character
[SerializeField] private Transform groundPos;                           // A position marking where to check if the player is grounded.
[SerializeField] private Transform m_CeilingCheck;                          // A position marking where to check for ceilings
[SerializeField] private Collider2D m_CrouchDisableCollider;                // A collider that will be disabled when crouching
```



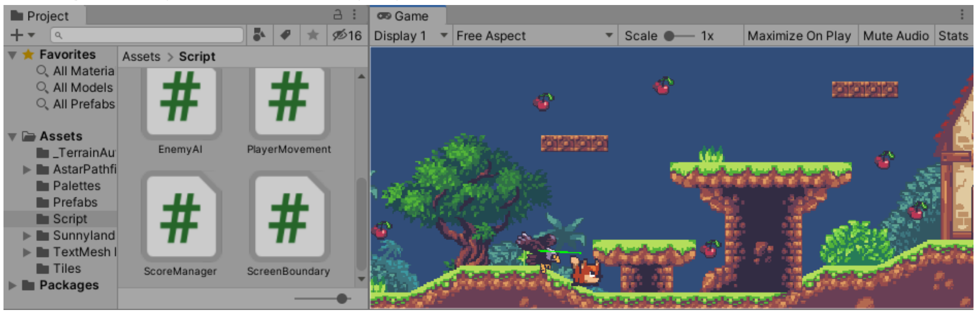
```
int cherryCount;

private float jumpTimeCounter;
private float jumpTime;
private bool isJumping;
private bool doubleJump;
// A collider that will be disabled when crouching

const float chcekRadius = .2f; // Radius of the overlap circle to determine if grounded
private bool isGrounded;            // Whether or not the player is grounded.
const float k_CeilingRadius = .2f; // Radius of the overlap circle to determine if the player can stand up
private Rigidbody2D myRigidBody;
private bool m_FacingRight = true;  // For determining which way the player is currently facing.
private Vector3 velocity = Vector3.zero;
public float speed;
private Animator anim;



Unity Message | 0 references
private void Awake()
{
    anim = GetComponent<Animator>();
    myRigidBody = GetComponent<Rigidbody2D>();

}


Unity Message | 0 references
private void FixedUpdate()
```



```
//  //  myRigidBody.velocity = Vector2.up * jumpForce;
//  //}
//  //float moveInput = Input.GetAxisRaw("Horizontal");
//  //myRigidBody.velocity = new Vector2(moveInput * speed, myRigidBody.velocity.y);
//  //if (moveInput == 0)
//  //{
//  //  anim.SetBool("isRunning", false);
//  //}
//  //else
//  //{
//  //  anim.SetBool("isRunning", true);
//  //}
//
//}



1 reference
public void Move(float move, bool crouch, bool jump)
{
    // If crouching, check to see if the character can stand up
    if (!crouch)
    {
        // If the character has a ceiling preventing them from standing up, keep them crouching
        if (Physics2D.OverlapCircle(m_CeilingCheck.position, k_CeilingRadius, whatIsGround))
        {
            crouch = true;
        }
    }

    //only control the player if grounded or airControl is turned on
    if (isGrounded || m_AirControl)
    {

        // If crouching
        if (crouch)
        {
            // Reduce the speed by the crouchSpeed multiplier
            move *= m_CrouchSpeed;
            anim.SetBool("isCrouching", true);
            // Disable one of the colliders when crouching
            if (m_CrouchDisableCollider != null)
                m_CrouchDisableCollider.enabled = false;
        } else
        {
            // Enable the collider when not crouching
            if (m_CrouchDisableCollider != null)
```

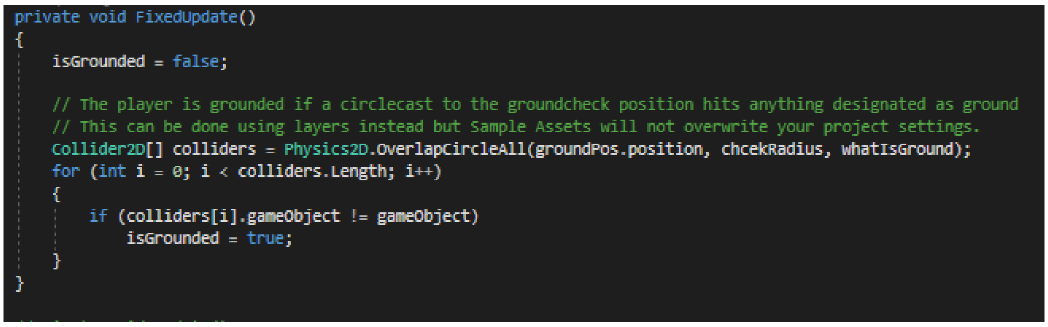


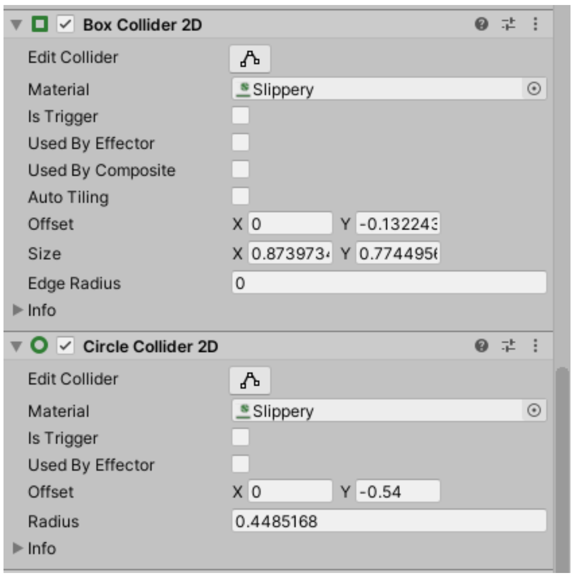
in the above capture it can be seen that the player is able to hide as there is some room with the Ceiling check placed over the player.



in the above capture it can be seen that the player is unable to hide as there is no room with the Ceiling check placed over the player and the player height in crouch state is greater than ceiling height and the object.

2. Box Collider and Circle Collider to determine objects that player can interact with or even act as trigger to certain script events being called in scene to play out, this works by setting the radius around the Player/enemy character as determines what it collides with by comparing the tags of the GameObjects present in the scene to follow out its actions.


```
private void FixedUpdate()
{
    isGrounded = false;

    // The player is grounded if a circlecast to the groundcheck position hits anything designated as ground
    // This can be done using layers instead but Sample Assets will not overwrite your project settings.
    Collider2D[] colliders = Physics2D.OverlapCircleAll(groundPos.position, chcekRadius, whatIsGround);
    for (int i = 0; i < colliders.Length; i++)
    {
        if (colliders[i].gameObject != gameObject)
            isGrounded = true;
    }
}
```



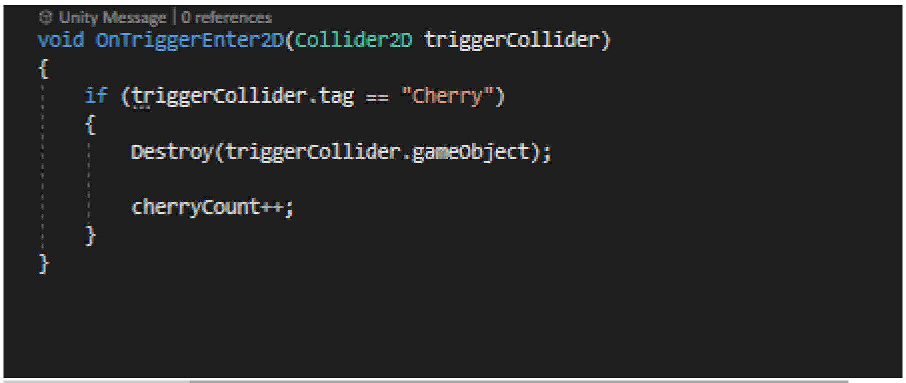
Box Collider 2D

Edit Collider	
Material	Slippery
Is Trigger	☐
Used By Effector	☐
Used By Composite	☐
Auto Tiling	☐
Offset	X 0 Y -0.132243
Size	X 0.873973 Y 0.774495
Edge Radius	0
Info	

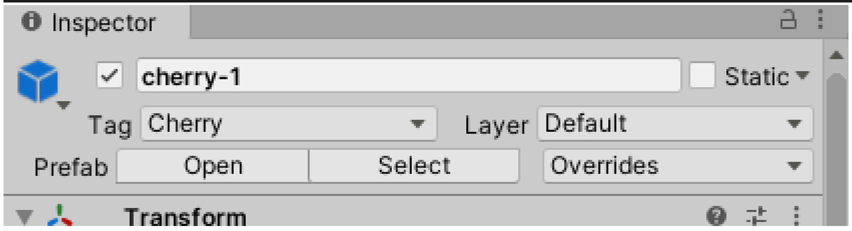
Circle Collider 2D

Edit Collider	
Material	Slippery
Is Trigger	☐
Used By Effector	☐
Offset	X 0 Y -0.54
Radius	0.4485168
Info	

Like in case of collecting cherries we were able to do so by comparing the tags of the objects.



```
Unity Message | 0 references
void OnTriggerEnter2D(Collider2D triggerCollider)
{
    if (triggerCollider.tag == "Cherry")
    {
        Destroy(triggerCollider.gameObject);

        cherryCount++;
    }
}
```



Inspector

cherry-1 Static

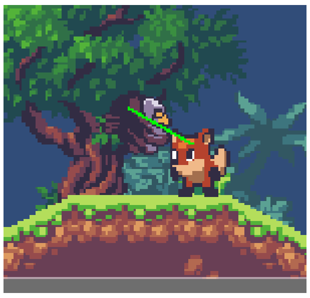
Tag Cherry Layer Default

Prefab Open Select Overrides

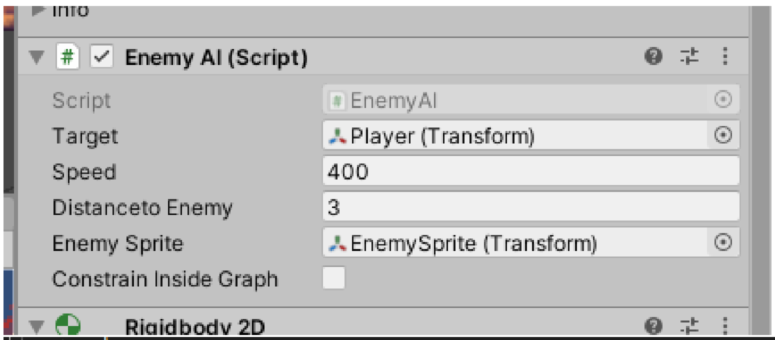
Transform

These can be added and removed in unity tag window which can be accessed in the script.

The Collider methods added also helps to prevent characters from overlapping and they can interact with one another by using default layer which can be changed in the unity inspector window.



they are close to one another based on the proximity radius specified plus using the distance to enemy that can be compared to advance to enemy.

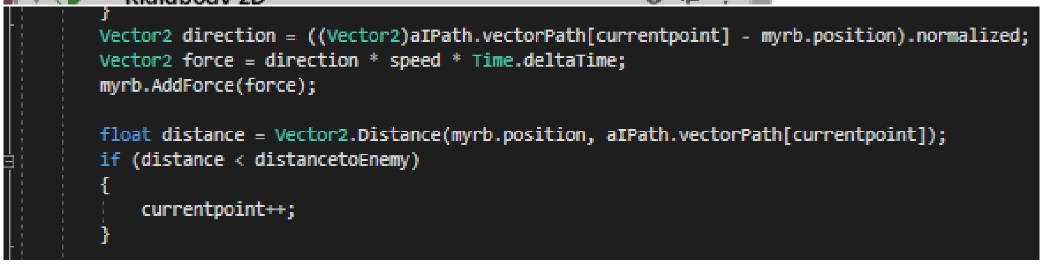


Info

Enemy AI (Script)

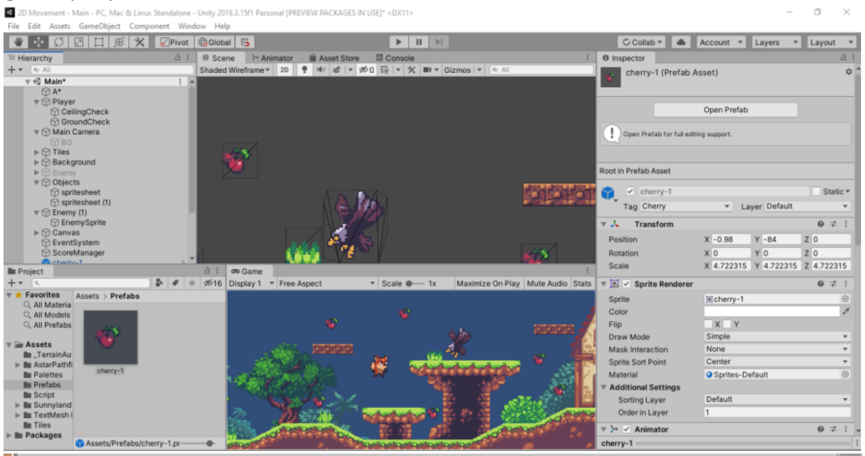
Script	EnemyAI
Target	Player (Transform)
Speed	400
Distanceto Enemy	3
Enemy Sprite	EnemySprite (Transform)
Constrain Inside Graph	

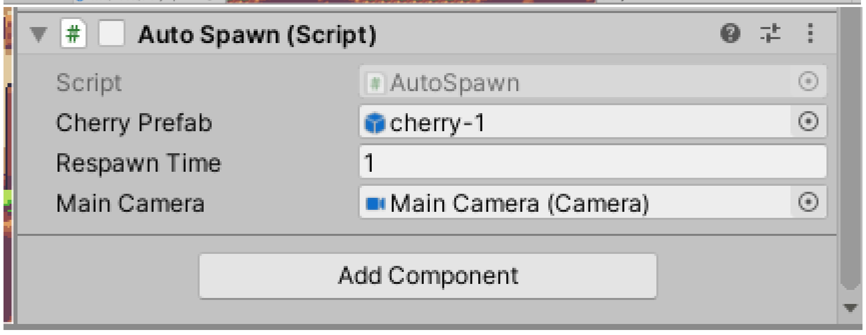
Rigidbody 2D



```
        }
        Vector2 direction = ((Vector2)aIPath.vectorPath[currentpoint] - myrb.position).normalized;
        Vector2 force = direction * speed * Time.deltaTime;
        myrb.AddForce(force);

        float distance = Vector2.Distance(myrb.position, aIPath.vectorPath[currentpoint]);
        if (distance < distancetoEnemy)
        {
            currentpoint++;
        }
```

3. Auto Spawning objects works by creating and object and attaching all the working script of that object like Animator Controller, 2D event Colliders and other stuff then creating a prefab by dragging it from the game scene to assets folder and once that's done used either an Empty game object or Main Camera to assign a new script 'Auto Spawn' which in turn could also utilize the screen boundary to auto spawn objects randomly within the working gameplay screen window.



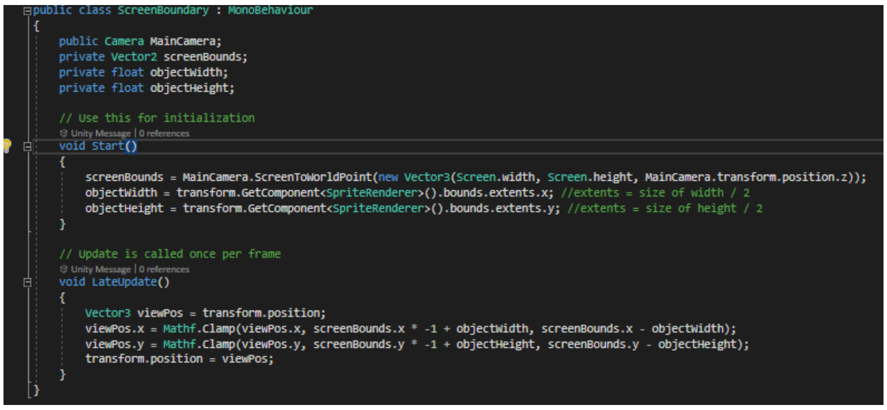


Auto Spawn (Script)

Script	AutoSpawn
Cherry Prefab	cherry-1
Respawn Time	1
Main Camera	Main Camera (Camera)

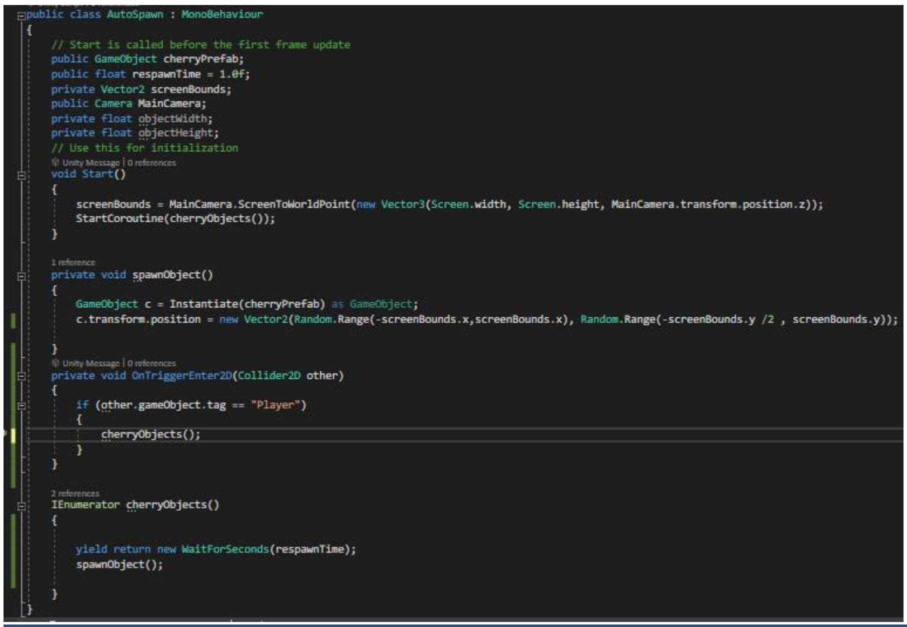
Add Component

To determine Screen boundary

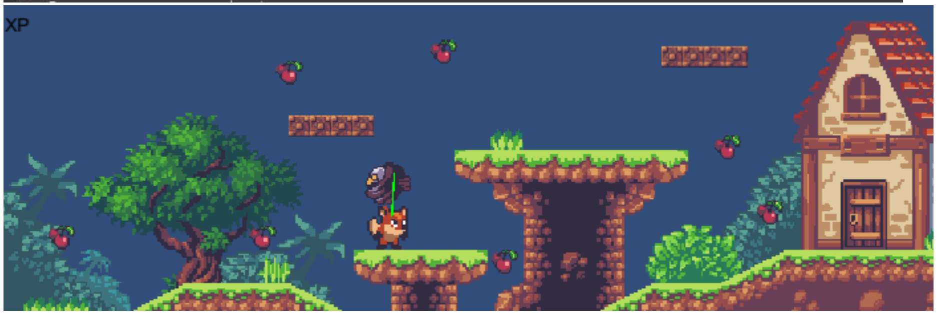


```
public class ScreenBoundary : MonoBehaviour
{
    public Camera MainCamera;
    private Vector2 screenBounds;
    private float objectWidth;
    private float objectHeight;

    // Use this for initialization
    Unity Message | 0 references
    void Start()
    {
        screenBounds = MainCamera.ScreenToWorldPoint(new Vector3(Screen.width, Screen.height, MainCamera.transform.position.z));
        objectWidth = transform.GetComponent<SpriteRenderer>().bounds.extents.x; //extents = size of width / 2
        objectHeight = transform.GetComponent<SpriteRenderer>().bounds.extents.y; //extents = size of height / 2
    }

    // Update is called once per frame
    Unity Message | 0 references
    void LateUpdate()
    {
        Vector3 viewPos = transform.position;
        viewPos.x = Mathf.Clamp(viewPos.x, screenBounds.x * -1 + objectWidth, screenBounds.x - objectWidth);
        viewPos.y = Mathf.Clamp(viewPos.y, screenBounds.y * -1 + objectHeight, screenBounds.y - objectHeight);
        transform.position = viewPos;
    }
}
```

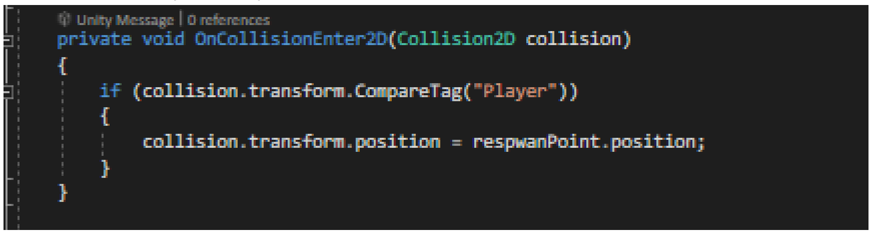
Auto Spawn script which uses a random range generator while considering the screen bounds and then spawns objects using Enumerator and Start Coroutine.



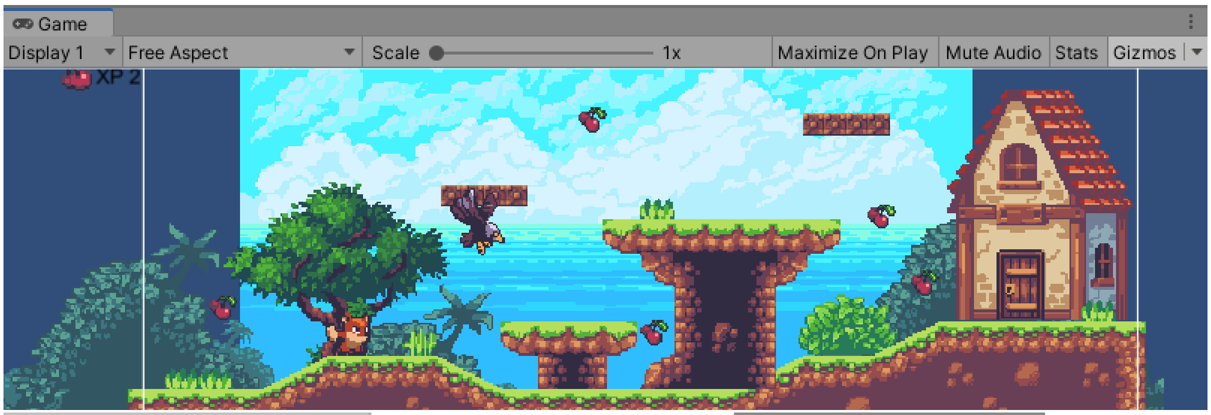
```
public class AutoSpawn : MonoBehaviour
{
    // Start is called before the first frame update
    public GameObject cherryPrefab;
    public float respawnTime = 1.0f;
    private Vector2 screenBounds;
    public Camera MainCamera;
    private float objectWidth;
    private float objectHeight;
    // Use this for initialization
    Unity Message | 0 references
    void Start()
    {
        screenBounds = MainCamera.ScreenToWorldPoint(new Vector3(Screen.width, Screen.height, MainCamera.transform.position.z));
        StartCoroutine(cherryObjects());
    }

    1 reference
    private void spawnObject()
    {
        GameObject c = Instantiate(cherryPrefab) as GameObject;
        c.transform.position = new Vector2(Random.Range(-screenBounds.x,screenBounds.x), Random.Range(-screenBounds.y /2 , screenBounds.y));

    }
    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.gameObject.tag == "Player")
        {
            cherryObjects();
        }
    }

    2 references
    IEnumerator cherryObjects()
    {

        yield return new WaitForSeconds(respawnTime);
        spawnObject();

    }
}
```

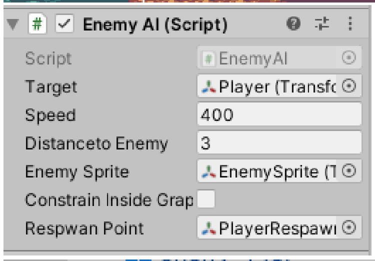


4. Respawn if killed by enemy the way this works is that upon collision of enemy with the player the tags with On collision event are compared and which resets the player position to the respawn check point for the player to start from again.
Done in EnemyAI script:



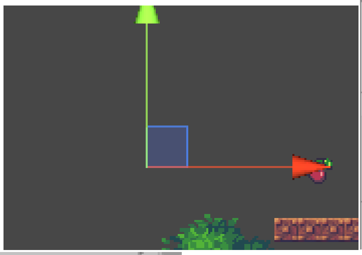
```
    Unity Message | 0 references
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.transform.CompareTag("Player"))
        {
            collision.transform.position = respwanPoint.position;
        }
    }
```

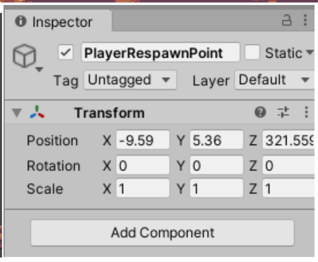




Enemy AI (Script)

Script	EnemyAI
Target	Player (Transfc
Speed	400
Distanceto Enemy	3
Enemy Sprite	EnemySprite (T
Constrain Inside Grap	
Respwan Point	PlayerRespawı





Inspector

PlayerRespawnPoint Static

Tag Untagged Layer Default

Transform

Position	X -9.59	Y 5.36	Z 321.55
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Add Component

PlayerRespawnPoint

Problems faced: although most part of it was working as intended however there are some issues yet to be fixed such as the respawn position shouldn’t really work until an attack is fired, in my case it was working as soon as the enemy interacted with the player.