Game project 1 2020

MAY 4

COMPANY NAME
Authored by: Your Name

Logo Name

Title Heading

Subtitle Text Here

To get started right away, just tap any placeholder text (such as this) and start typing to replace it with your own.

Want to insert a picture from your files or add a shape, text box, or table? You got it! On the Insert tab of the ribbon, just tap the option you need.

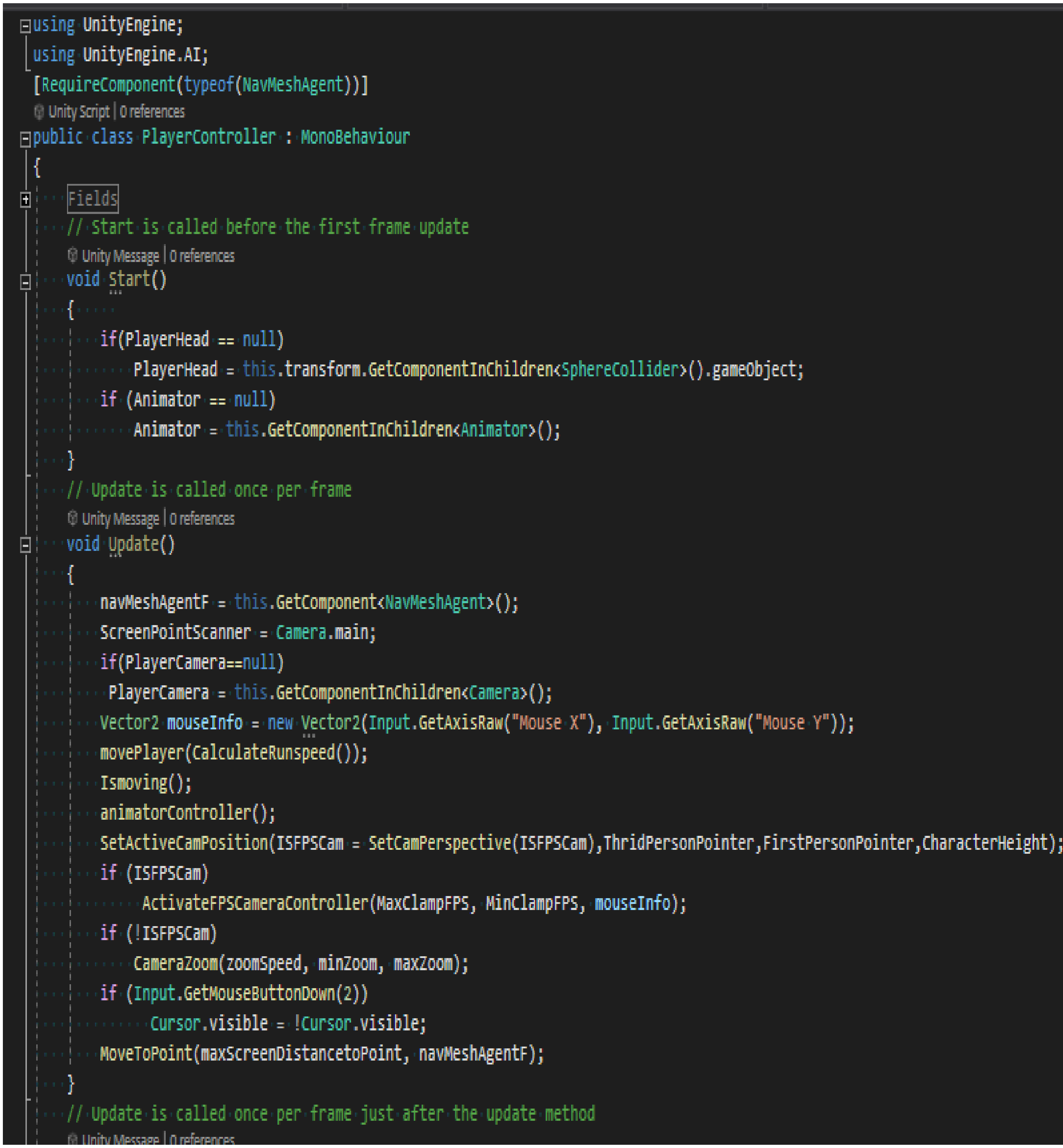
“Find even more easy-to-use tools on the Insert tab, such as to add a hyperlink or insert a comment”

To get started right away, just tap any placeholder text (such as this) and start typing to replace it with your own.

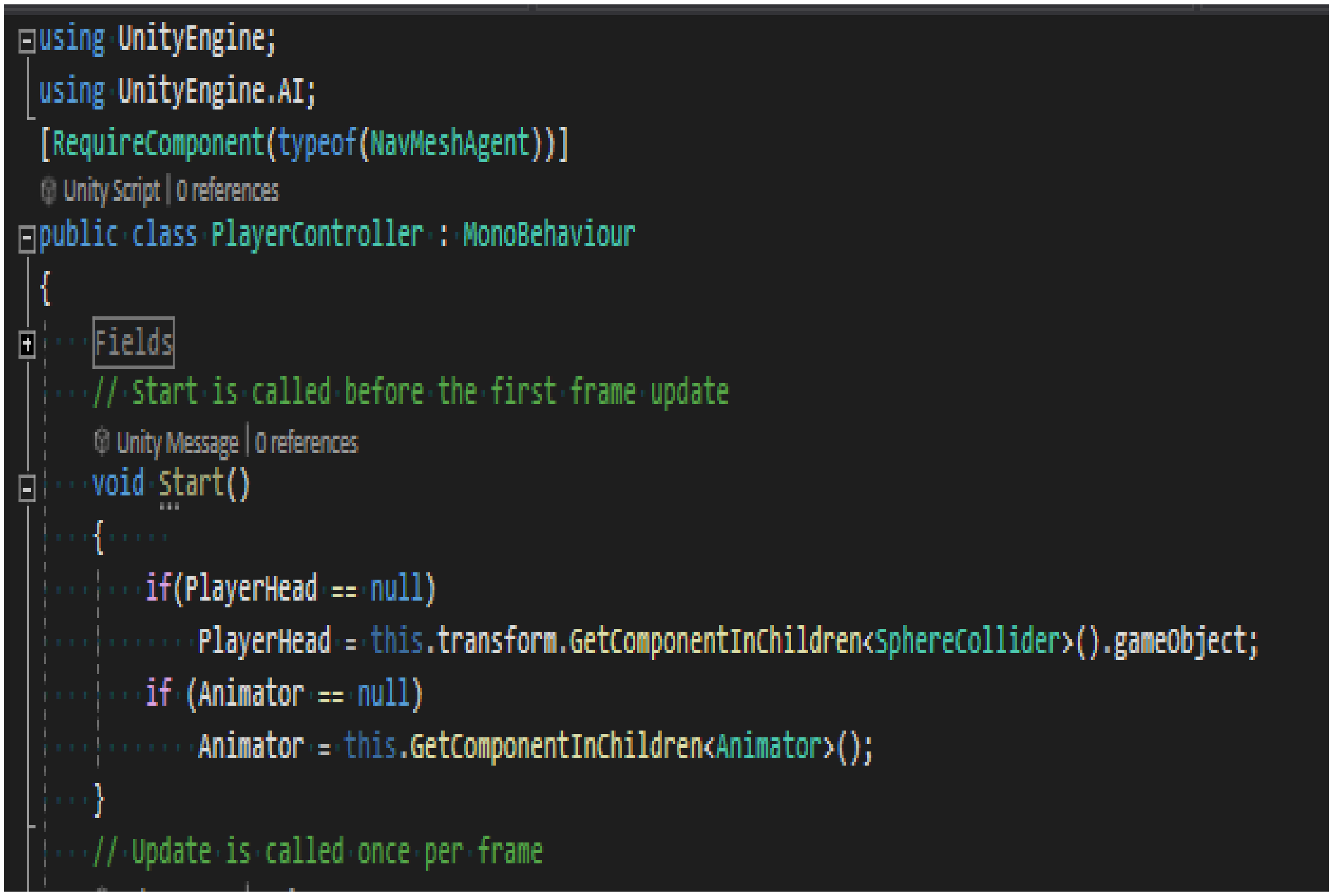
Want to insert a picture from your files or add a shape, text box, or table? You got it! On the Insert tab of the ribbon, just tap the option you need.

Player Controller

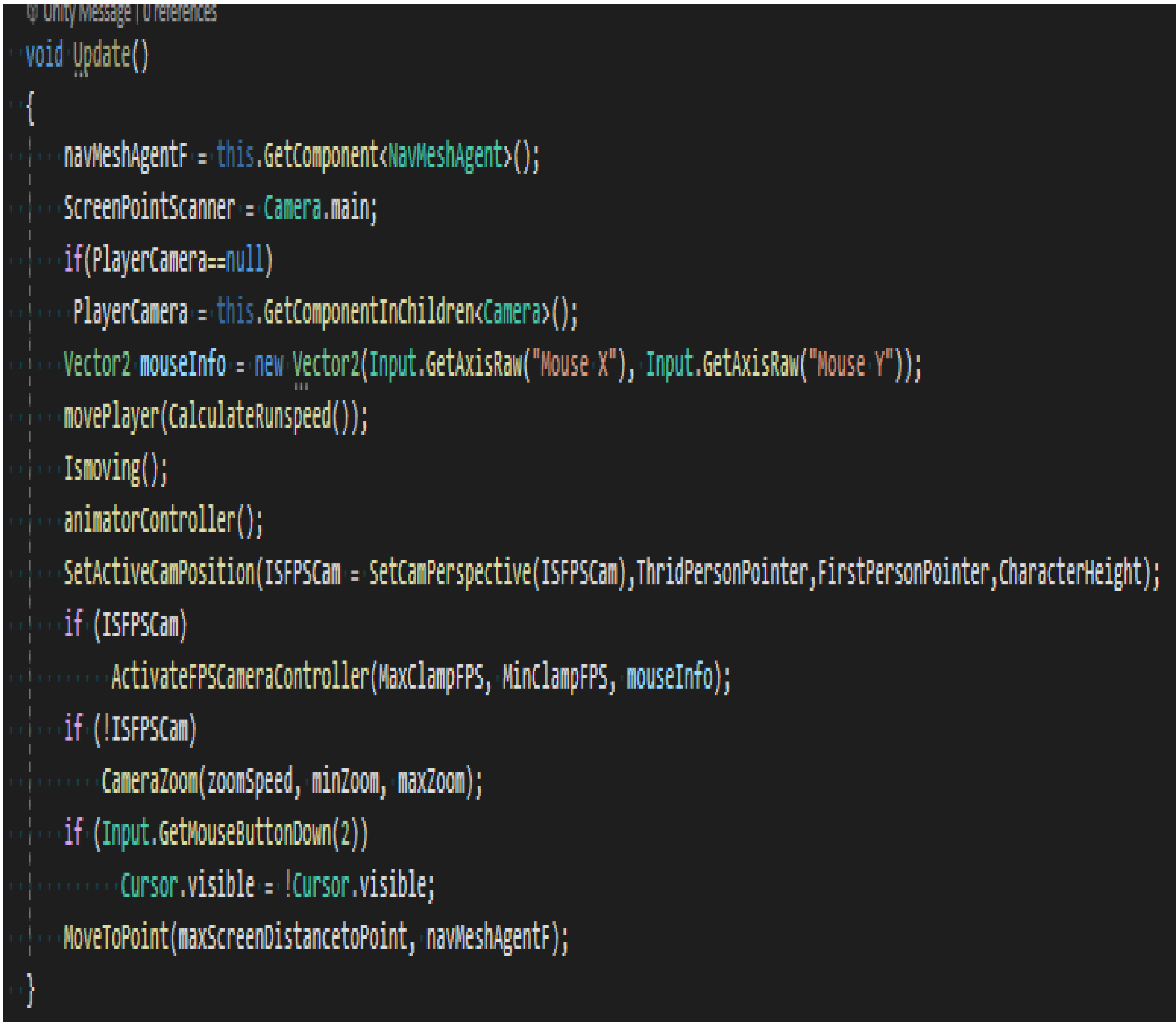
This is the Player Controller Script that controls the character in my game



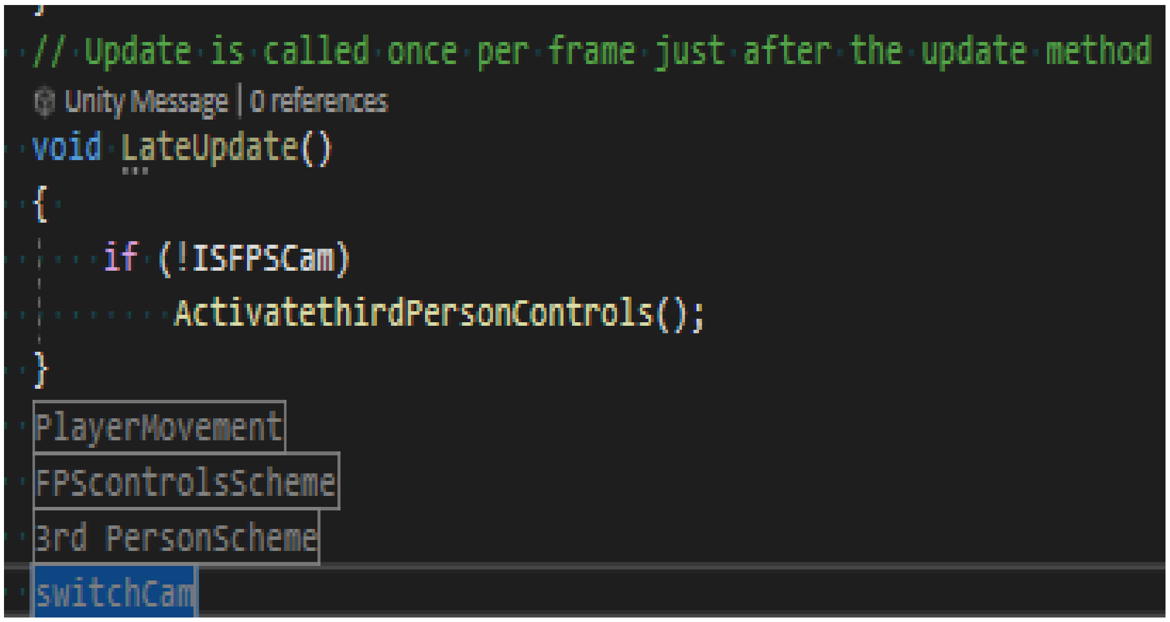
```
using UnityEngine;
using UnityEngine.AI;
[RequireComponent(typeof(NavMeshAgent))]
Unity Script | 0 references
public class PlayerController : MonoBehaviour
{
    Fields
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        if(PlayerHead == null)
            PlayerHead = this.transform.GetComponentInChildren<SphereCollider>().gameObject;
        if (Animator == null)
            Animator = this.GetComponentInChildren<Animator>();
    }
    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        navMeshAgentF = this.GetComponent<NavMeshAgent>();
        ScreenPointScanner = Camera.main;
        if(PlayerCamera==null)
         PlayerCamera = this.GetComponentInChildren<Camera>();
        Vector2 mouseInfo = new Vector2(Input.GetAxisRaw("Mouse X"), Input.GetAxisRaw("Mouse Y"));
        movePlayer(CalculateRunspeed());
        Ismoving();
        animatorController();
        SetActiveCamPosition(ISFPSCam = SetCamPerspective(ISFPSCam),ThridPersonPointer,FirstPersonPointer,CharacterHeight);
        if (ISFPSCam)
             ActivateFPSCameraController(MaxClampFPS, MinClampFPS, mouseInfo);
        if (!ISFPSCam)
            CameraZoom(zoomSpeed, minZoom, maxZoom);
        if (Input.GetMouseButtonDown(2))
              Cursor.visible = !Cursor.visible;
        MoveToPoint(maxScreenDistancetoPoint, navMeshAgentF);
    }
    // Update is called once per frame just after the update method
```



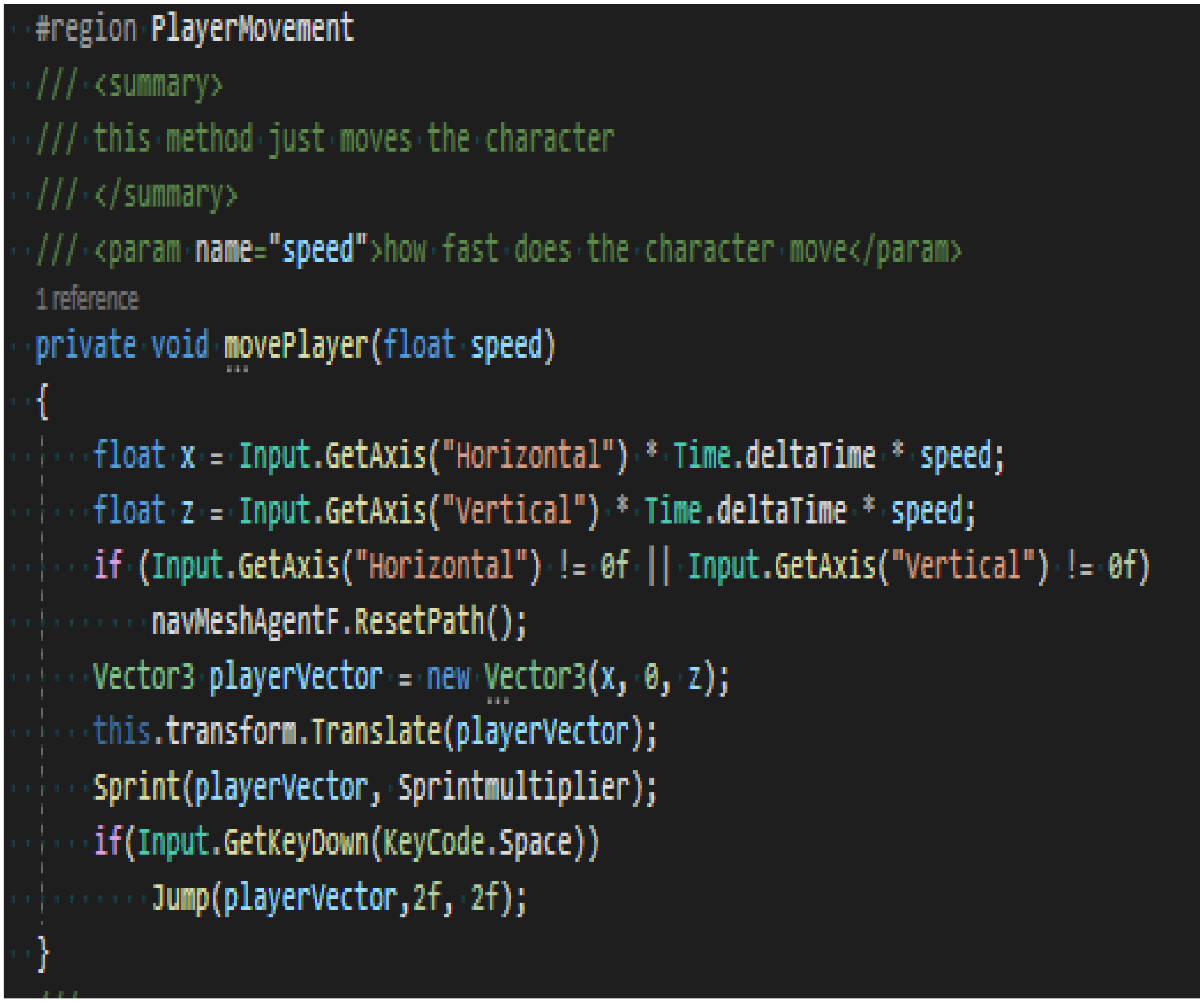
```
using UnityEngine;
using UnityEngine.AI;
[RequireComponent(typeof(NavMeshAgent))]
Unity Script | 0 references
public class PlayerController : MonoBehaviour
{
    Fields
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        if(PlayerHead == null)
            PlayerHead = this.transform.GetComponentInChildren<SphereCollider>().gameObject;
        if (Animator == null)
            Animator = this.GetComponentInChildren<Animator>();
    }
    // Update is called once per frame
```



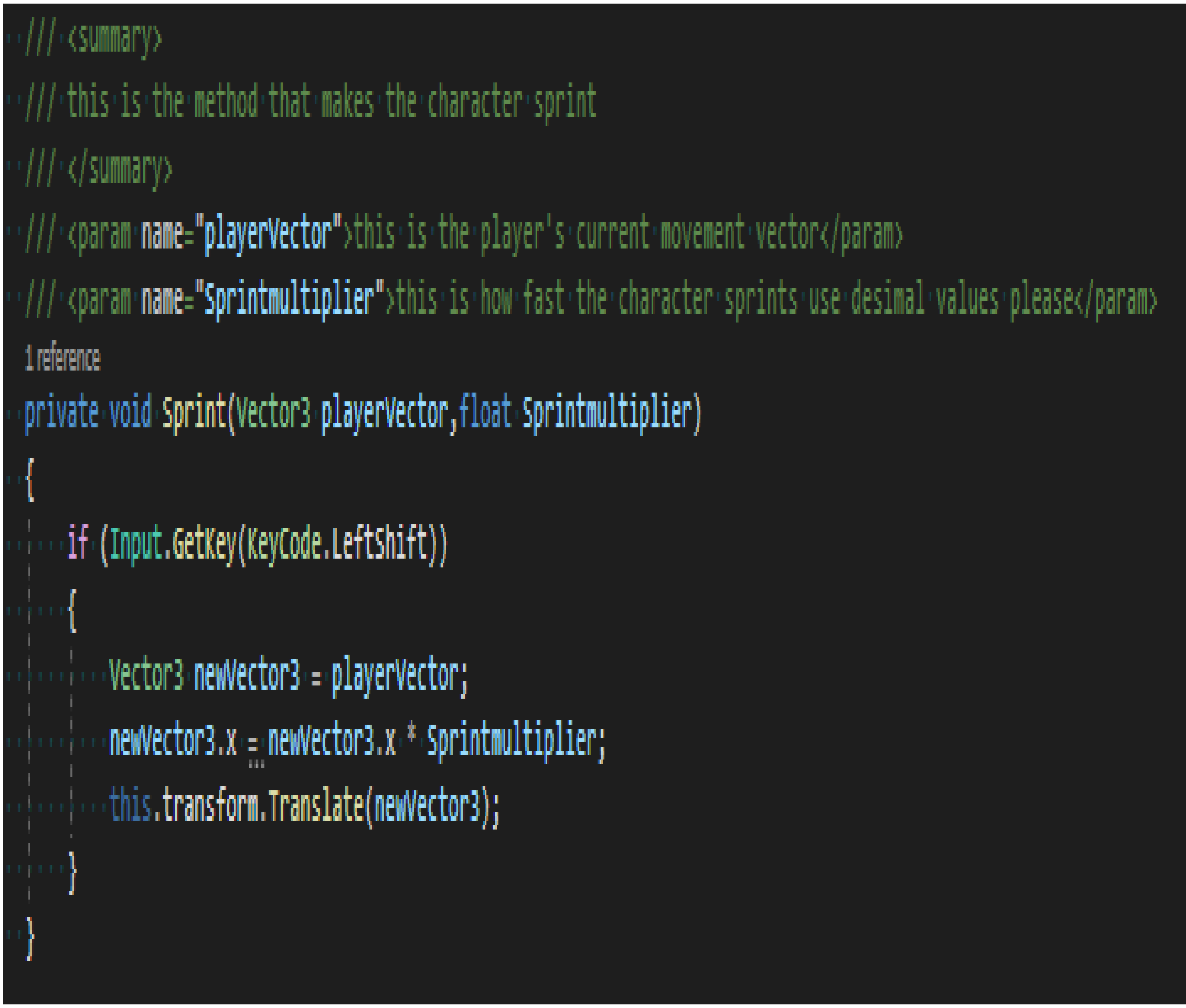
```
⊕ Unity Message | 0 references
void Update()
{
    navMeshAgentF = this.GetComponent<NavMeshAgent>();
    ScreenPointScanner = Camera.main;
    if(PlayerCamera==null)
     PlayerCamera = this.GetComponentInChildren<Camera>();
    Vector2 mouseInfo = new Vector2(Input.GetAxisRaw("Mouse X"), Input.GetAxisRaw("Mouse Y"));
    movePlayer(CalculateRunspeed());
    Ismoving();
    animatorController();
    SetActiveCamPosition(ISFPSCam = SetCamPerspective(ISFPSCam),ThridPersonPointer,FirstPersonPointer,CharacterHeight);
    if (ISFPSCam)
        ActivateFPSCameraController(MaxClampFPS, MinClampFPS, mouseInfo);
    if (!ISFPSCam)
       CameraZoom(zoomSpeed, minZoom, maxZoom);
    if (Input.GetMouseButtonDown(2))
         Cursor.visible = !Cursor.visible;
    MoveToPoint(maxScreenDistancetoPoint, navMeshAgentF);
}
```



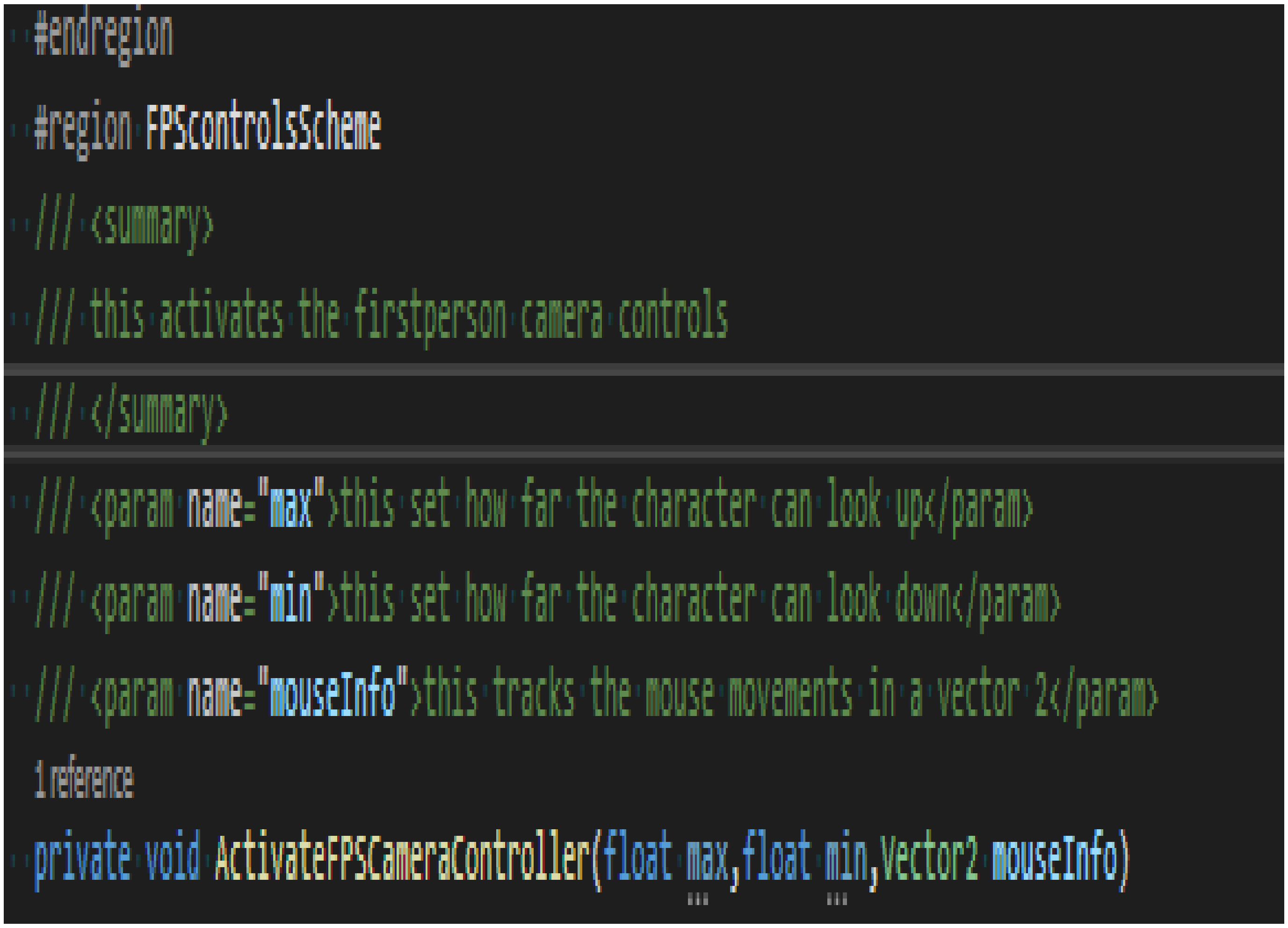
```
// Update is called once per frame just after the update method
Unity Message | 0 references
void LateUpdate()
{
    if (!ISFPSCam)
        ActivatethirdPersonControls();
}
PlayerMovement
FPScontrolsScheme
3rd PersonScheme
switchCam
```



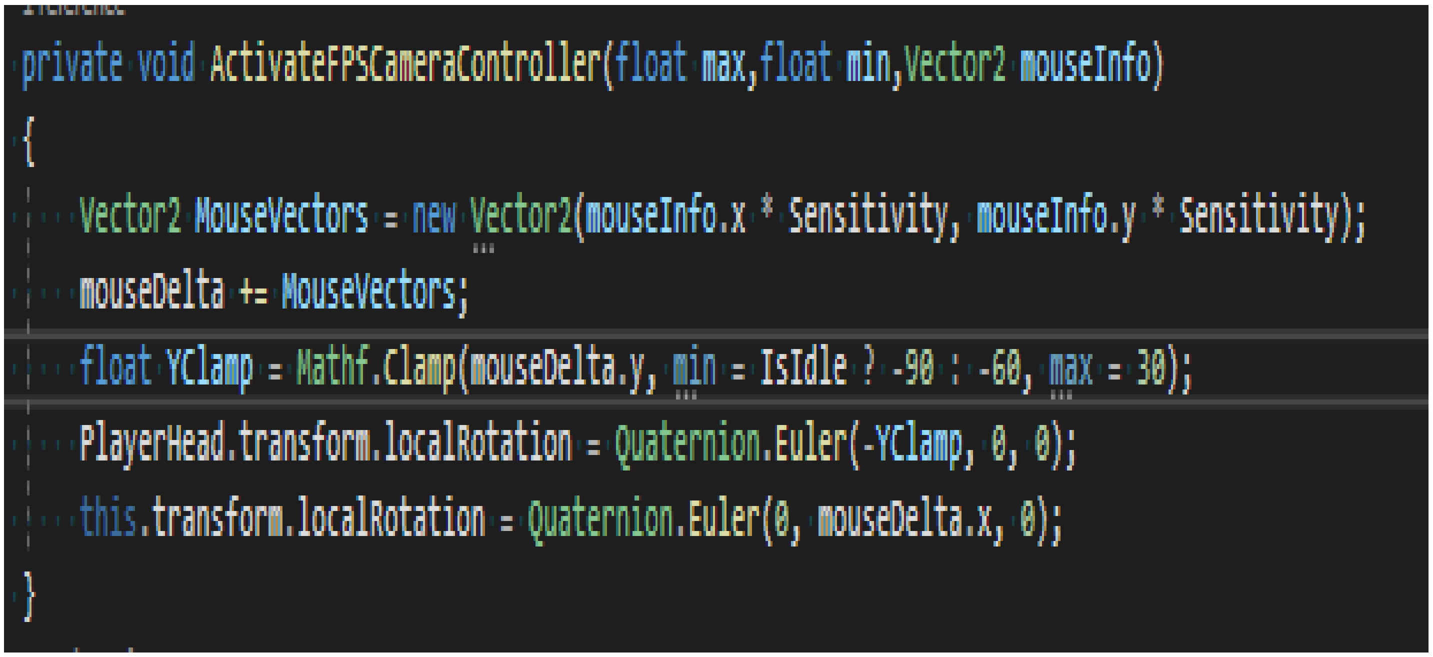
```
#region PlayerMovement
/// <summary>
/// this method just moves the character
/// </summary>
/// <param name="speed">how fast does the character move</param>
1 reference
private void movePlayer(float speed)
{
    float x = Input.GetAxis("Horizontal") * Time.deltaTime * speed;
    float z = Input.GetAxis("Vertical") * Time.deltaTime * speed;
    if (Input.GetAxis("Horizontal") != 0f || Input.GetAxis("Vertical") != 0f)
        navMeshAgentF.ResetPath();
    Vector3 playerVector = new Vector3(x, 0, z);
    this.transform.Translate(playerVector);
    Sprint(playerVector, Sprintmultiplier);
    if(Input.GetKeyDown(KeyCode.Space))
        Jump(playerVector,2f, 2f);
}
```



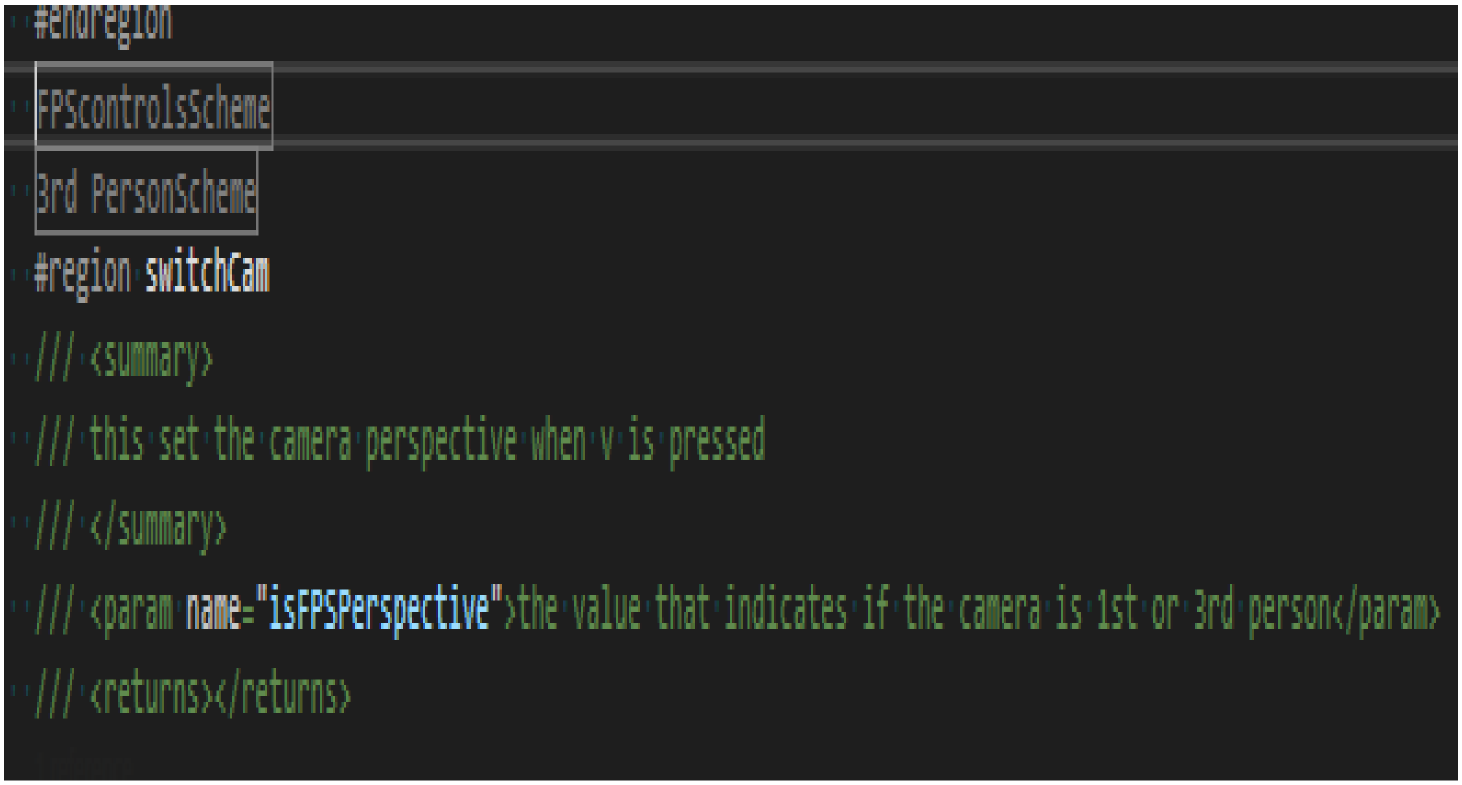
```
/// <summary>
/// this is the method that makes the character sprint
/// </summary>
/// <param name="playerVector">this is the player's current movement vector</param>
/// <param name="Sprintmultiplier">this is how fast the character sprints use desimal values please</param>
1 reference
private void Sprint(Vector3 playerVector,float Sprintmultiplier)
{
    if (Input.GetKey(KeyCode.LeftShift))
    {
        Vector3 newVector3 = playerVector;
        newVector3.x = newVector3.x * Sprintmultiplier;
        this.transform.Translate(newVector3);
    }
}
```



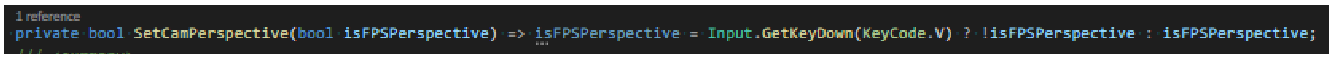
```
#endregion
#region FPScontrolsScheme
/// <summary>
/// this activates the firstperson camera controls
/// </summary>
/// <param name="max">this set how far the character can look up</param>
/// <param name="min">this set how far the character can look down</param>
/// <param name="mouseInfo">this tracks the mouse movements in a vector 2</param>
1 reference
private void ActivateFPSCameraController(float max,float min,Vector2 mouseInfo)
```



```
private void ActivateFPSCameraController(float max,float min,Vector2 mouseInfo)
{
    Vector2 MouseVectors = new Vector2(mouseInfo.x * Sensitivity, mouseInfo.y * Sensitivity);
    mouseDelta += MouseVectors;
    float YClamp = Mathf.Clamp(mouseDelta.y, min = IsIdle ? -90 : -60, max = 30);
    PlayerHead.transform.localRotation = Quaternion.Euler(-YClamp, 0, 0);
    this.transform.localRotation = Quaternion.Euler(0, mouseDelta.x, 0);
}
```

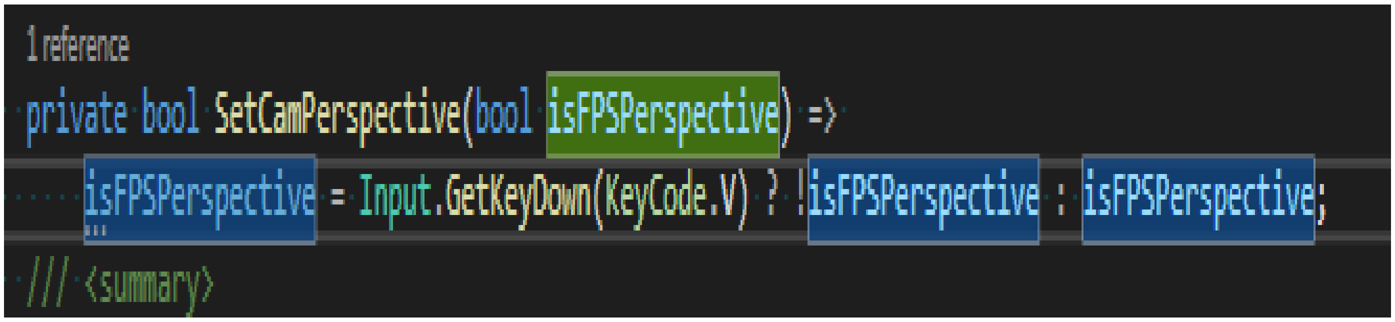


```
#endregion
FPScontrolsScheme
3rd PersonScheme
#region switchCam
/// <summary>
/// this set the camera perspective when v is pressed
/// </summary>
/// <param name="isFPSPerspective">the value that indicates if the camera is 1st or 3rd person</param>
/// <returns></returns>
```

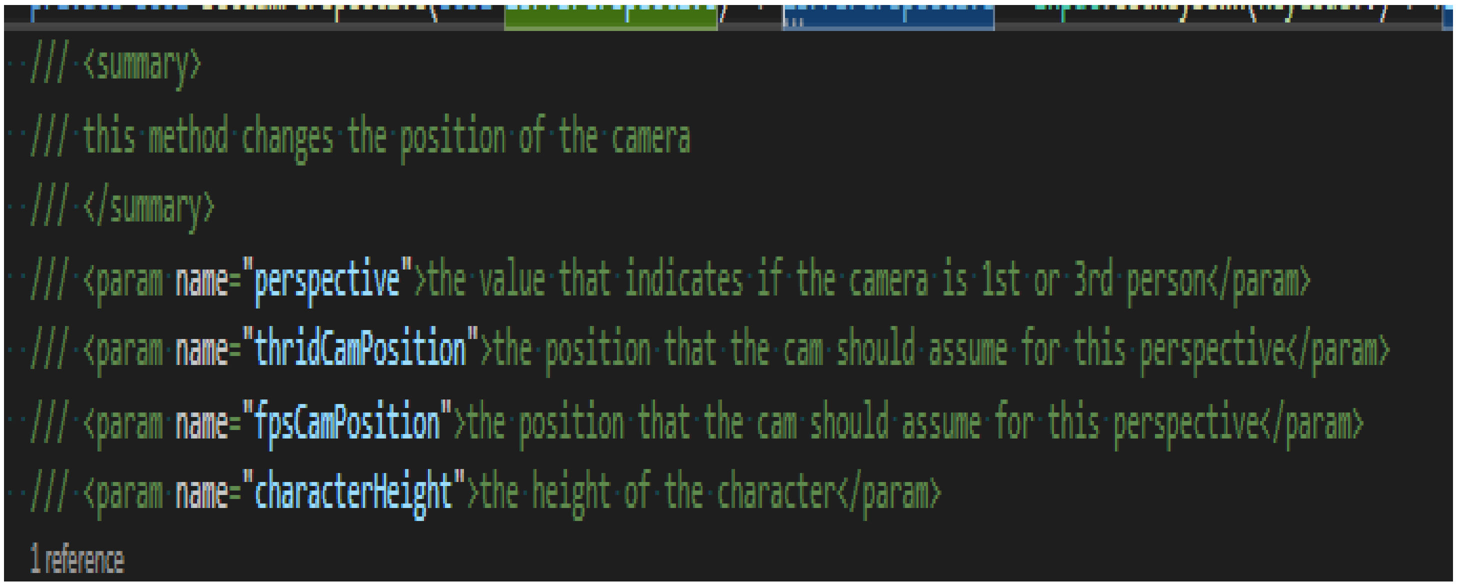


```
1 reference
private bool SetCamPerspective(bool isFPSPerspective) => isFPSPerspective = Input.GetKeyDown(KeyCode.V) ? !isFPSPerspective : isFPSPerspective;
```

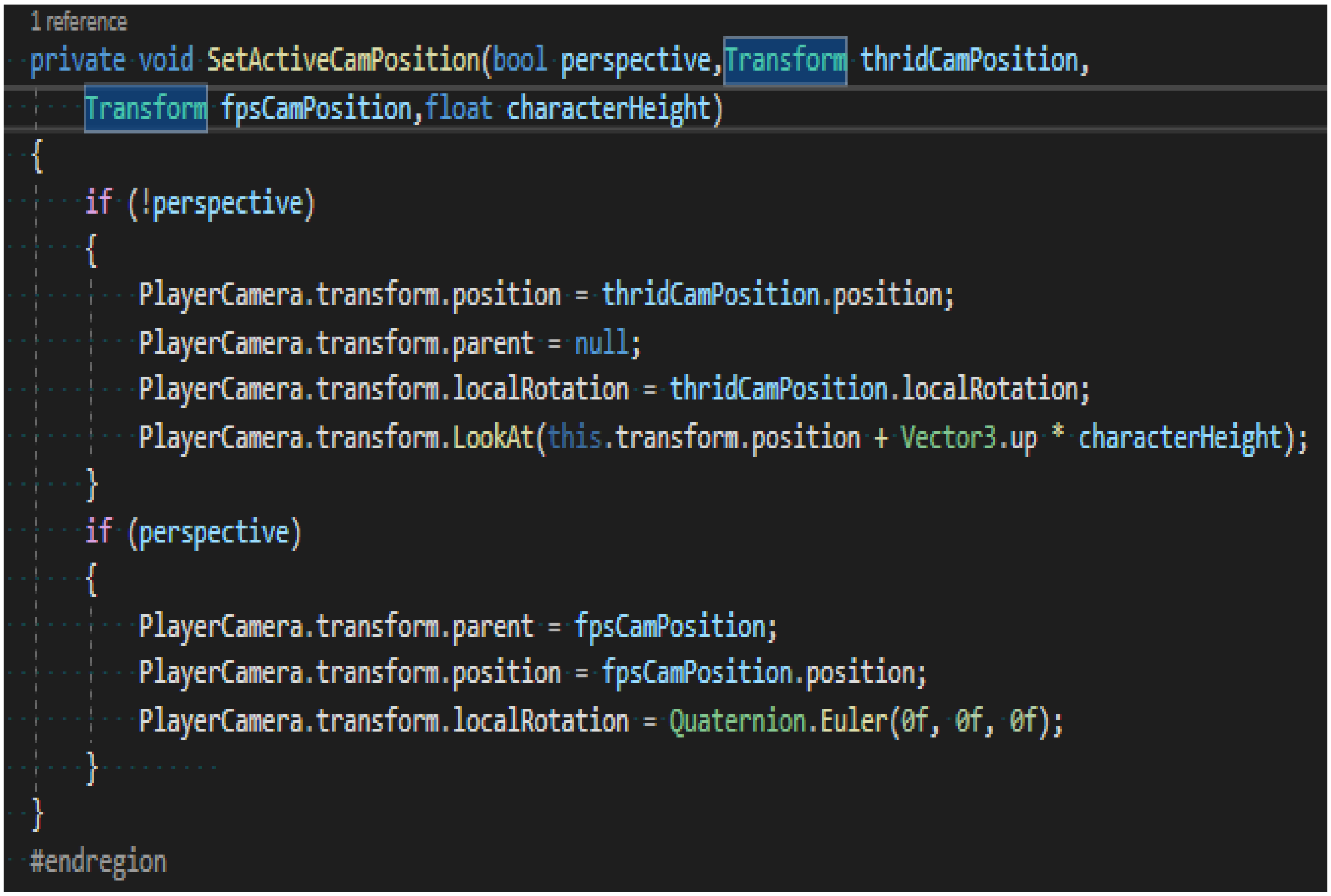
This here below is what is shown above just zoomed in.



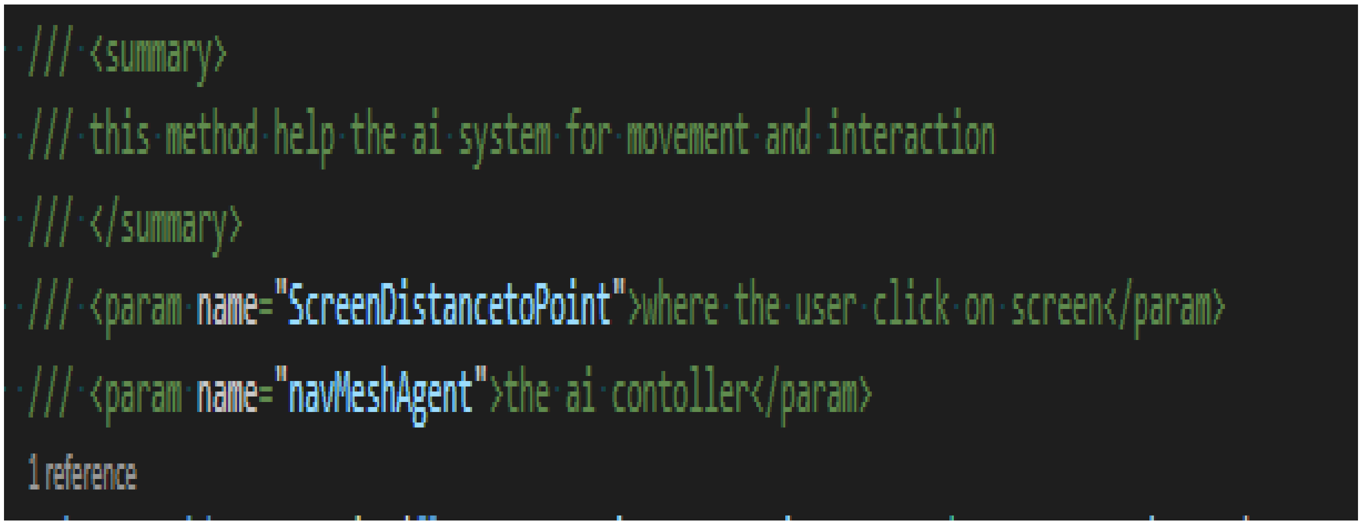
```
1 reference
private bool SetCamPerspective(bool isFPSPerspective) =>
    isFPSPerspective = Input.GetKeyDown(KeyCode.V) ? !isFPSPerspective : isFPSPerspective;
/// <summary>
```



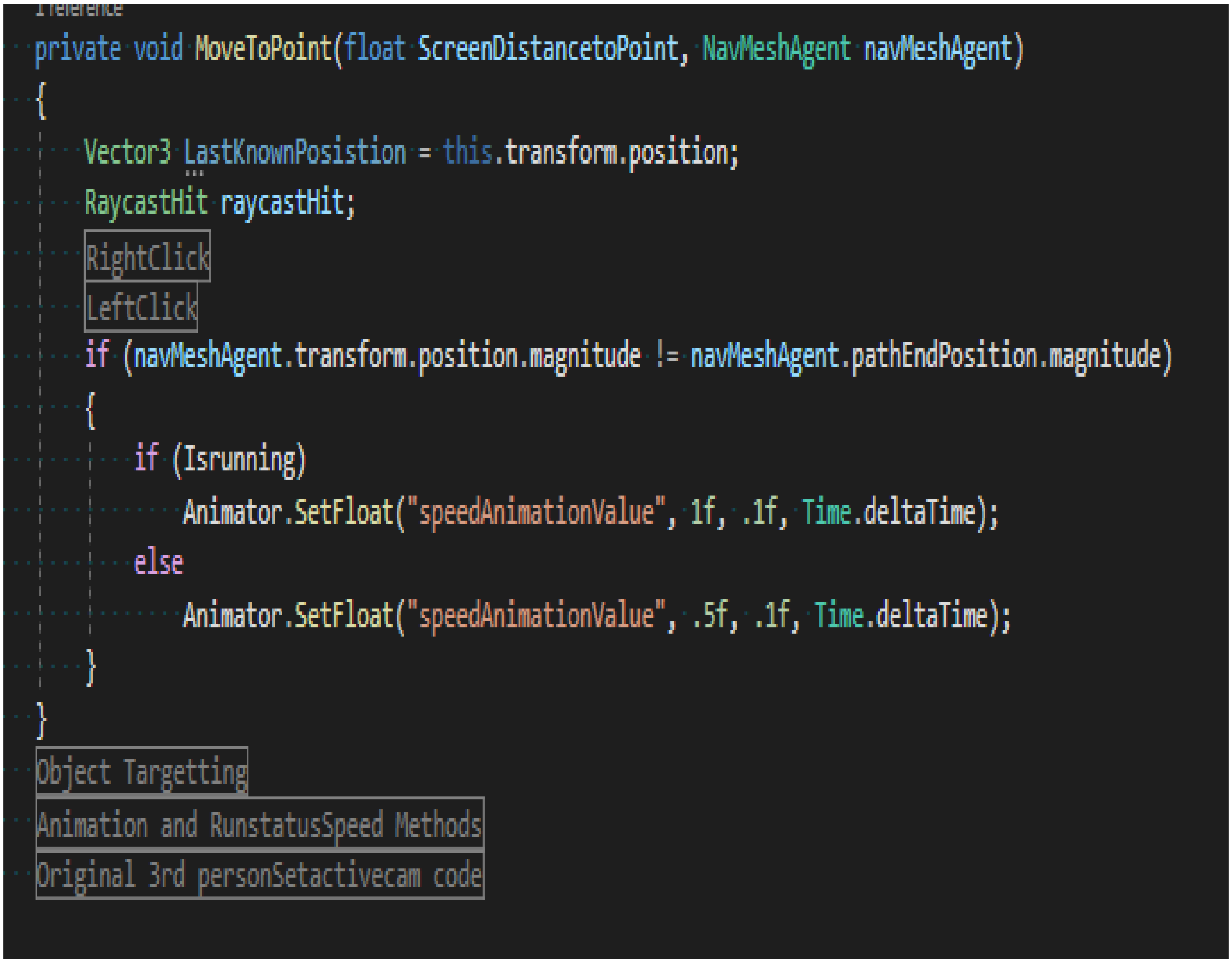
```
/// <summary>
/// this method changes the position of the camera
/// </summary>
/// <param name="perspective">the value that indicates if the camera is 1st or 3rd person</param>
/// <param name="thridCamPosition">the position that the cam should assume for this perspective</param>
/// <param name="fpsCamPosition">the position that the cam should assume for this perspective</param>
/// <param name="characterHeight">the height of the character</param>
1 reference
```



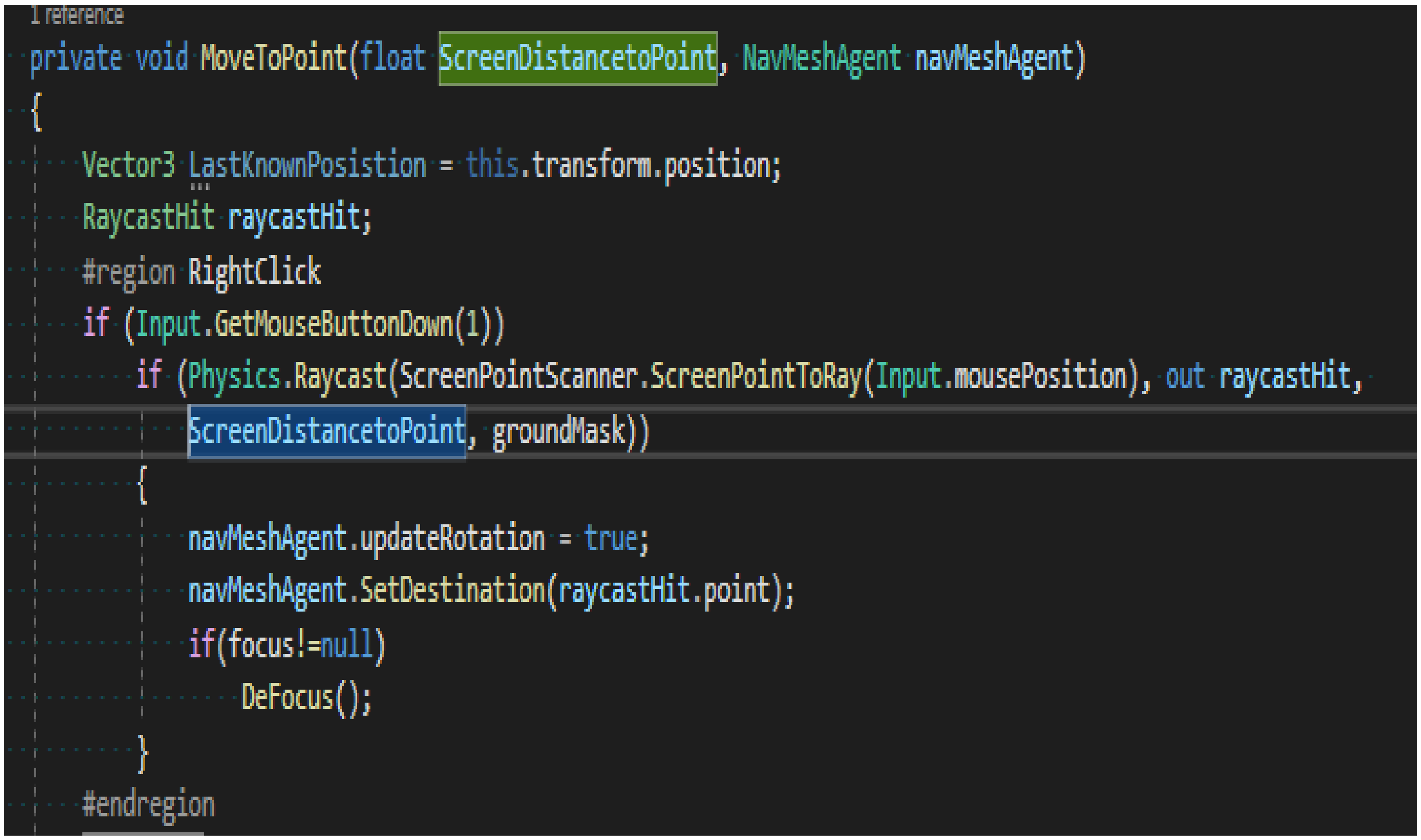
```
1 reference
private void SetActiveCamPosition(bool perspective,Transform thridCamPosition,
    Transform fpsCamPosition,float characterHeight)
{
    if (!perspective)
    {
        PlayerCamera.transform.position = thridCamPosition.position;
        PlayerCamera.transform.parent = null;
        PlayerCamera.transform.localRotation = thridCamPosition.localRotation;
        PlayerCamera.transform.LookAt(this.transform.position + Vector3.up * characterHeight);
    }
    if (perspective)
    {
        PlayerCamera.transform.parent = fpsCamPosition;
        PlayerCamera.transform.position = fpsCamPosition.position;
        PlayerCamera.transform.localRotation = Quaternion.Euler(0f, 0f, 0f);
    }
}
#endregion
```



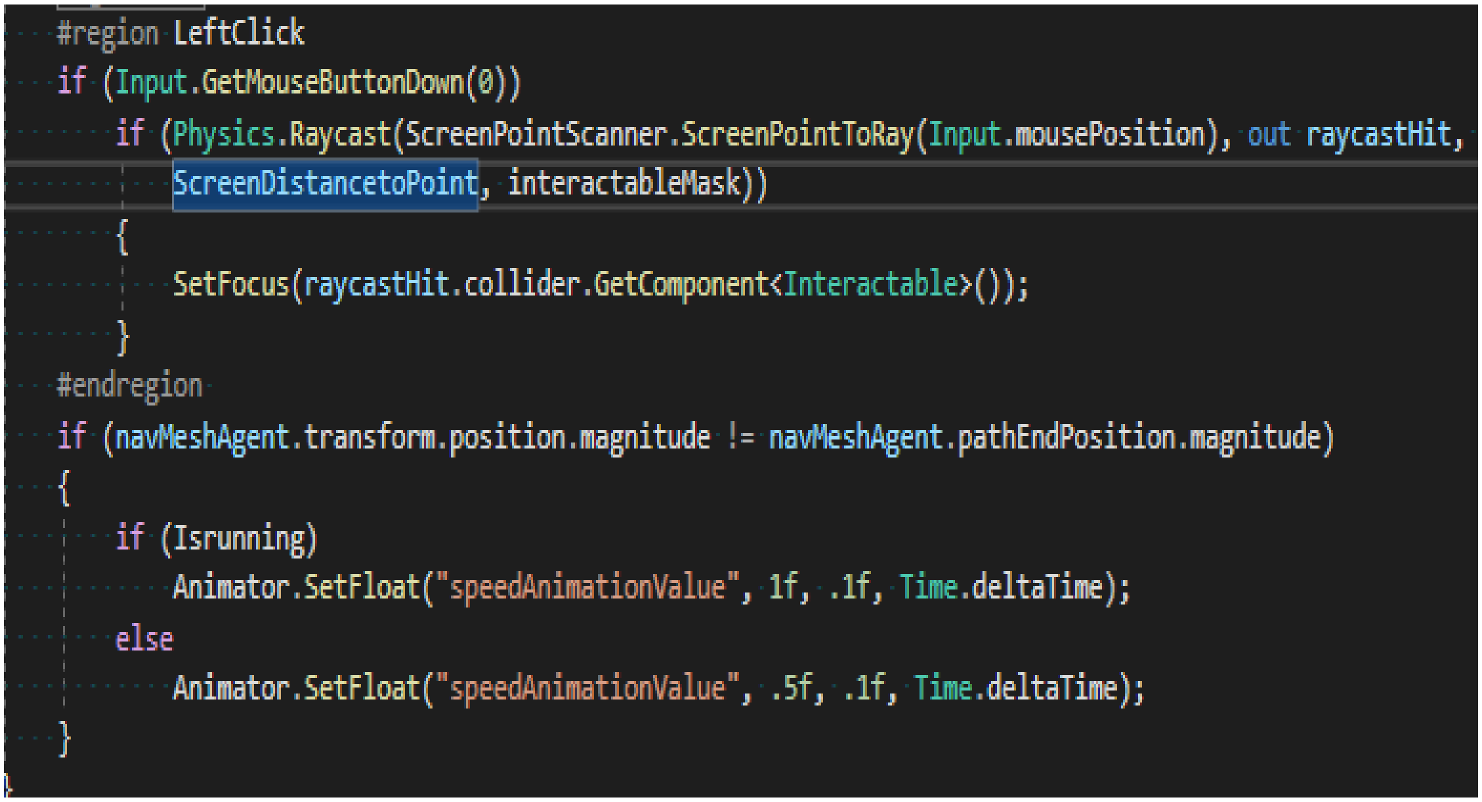
```
/// <summary>
/// this method help the ai system for movement and interaction
/// </summary>
/// <param name="ScreenDistancetoPoint">where the user click on screen</param>
/// <param name="navMeshAgent">the ai contoller</param>
1 reference
```



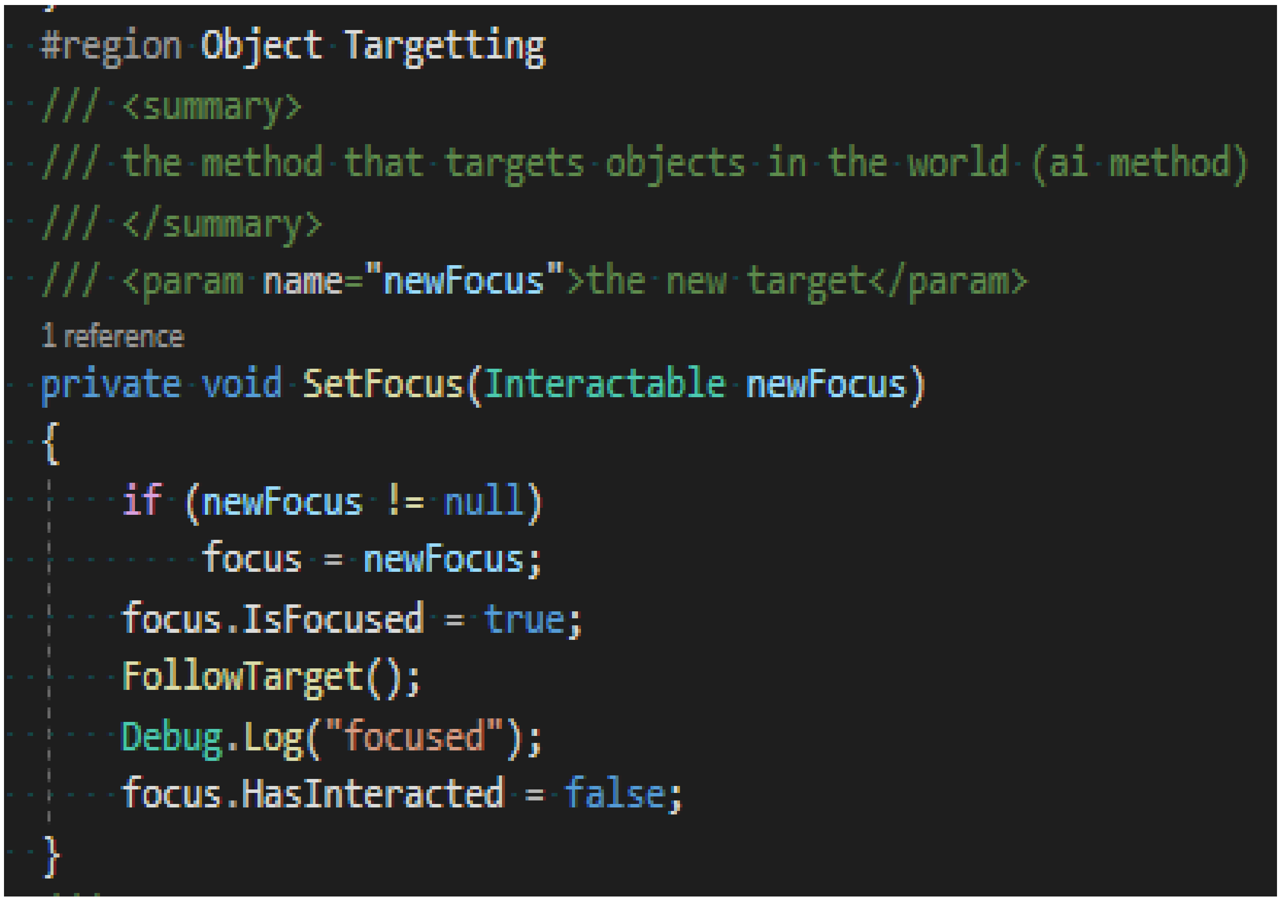
```
1 reference
private void MoveToPoint(float ScreenDistancetoPoint, NavMeshAgent navMeshAgent)
{
    Vector3 LastKnownPosistion = this.transform.position;
    RaycastHit raycastHit;
    RightClick
    LeftClick
    if (navMeshAgent.transform.position.magnitude != navMeshAgent.pathEndPosition.magnitude)
    {
        if (Isrunning)
            Animator.SetFloat("speedAnimationValue", 1f, .1f, Time.deltaTime);
        else
            Animator.SetFloat("speedAnimationValue", .5f, .1f, Time.deltaTime);
    }
}
Object Targetting
Animation and RunstatusSpeed Methods
Original 3rd personSetactivecam code
```



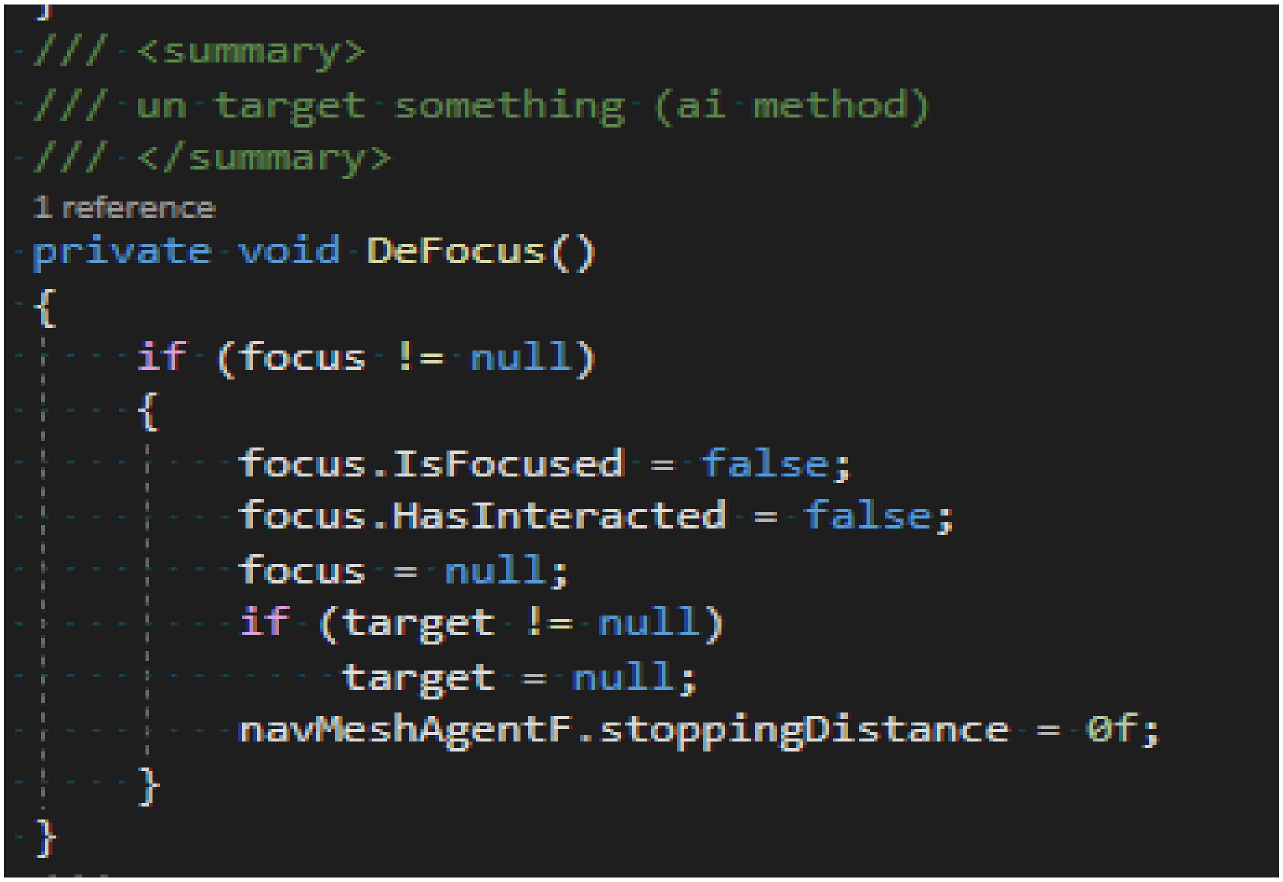
```
1 reference
private void MoveToPoint(float ScreenDistancetoPoint, NavMeshAgent navMeshAgent)
{
    Vector3 LastKnownPosistion = this.transform.position;
    RaycastHit raycastHit;
    #region RightClick
    if (Input.GetMouseButtonDown(1))
        if (Physics.Raycast(ScreenPointScanner.ScreenPointToRay(Input.mousePosition), out raycastHit,
            ScreenDistancetoPoint, groundMask))
        {
            navMeshAgent.updateRotation = true;
            navMeshAgent.SetDestination(raycastHit.point);
            if(focus!=null)
                DeFocus();
        }
    #endregion
```



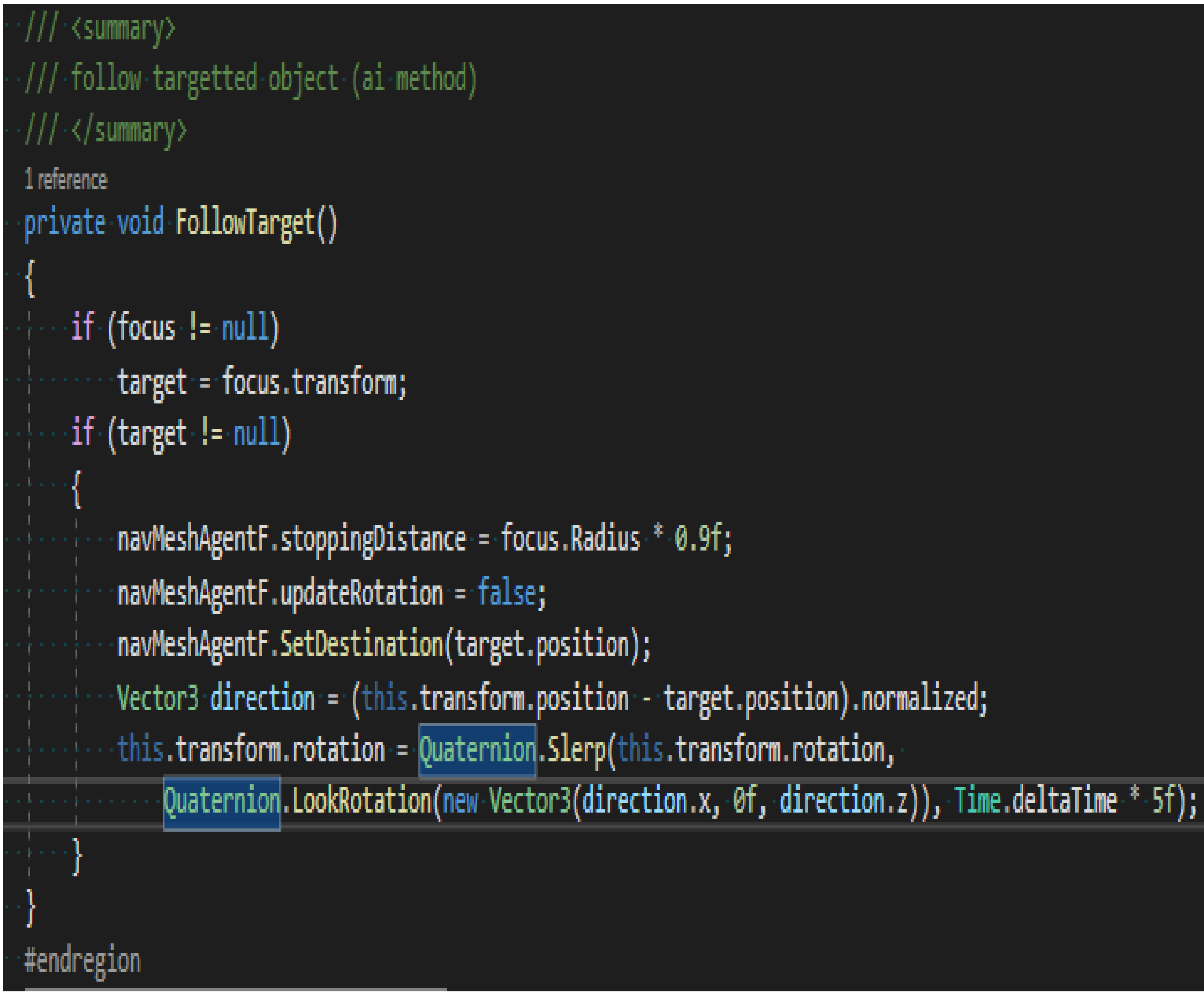
```
    #region LeftClick
    if (Input.GetMouseButtonDown(0))
        if (Physics.Raycast(ScreenPointScanner.ScreenPointToRay(Input.mousePosition), out raycastHit,
            ScreenDistancetoPoint, interactableMask))
        {
            SetFocus(raycastHit.collider.GetComponent<Interactable>());
        }
    #endregion
    if (navMeshAgent.transform.position.magnitude != navMeshAgent.pathEndPosition.magnitude)
    {
        if (Isrunning)
            Animator.SetFloat("speedAnimationValue", 1f, .1f, Time.deltaTime);
        else
            Animator.SetFloat("speedAnimationValue", .5f, .1f, Time.deltaTime);
    }
}
```



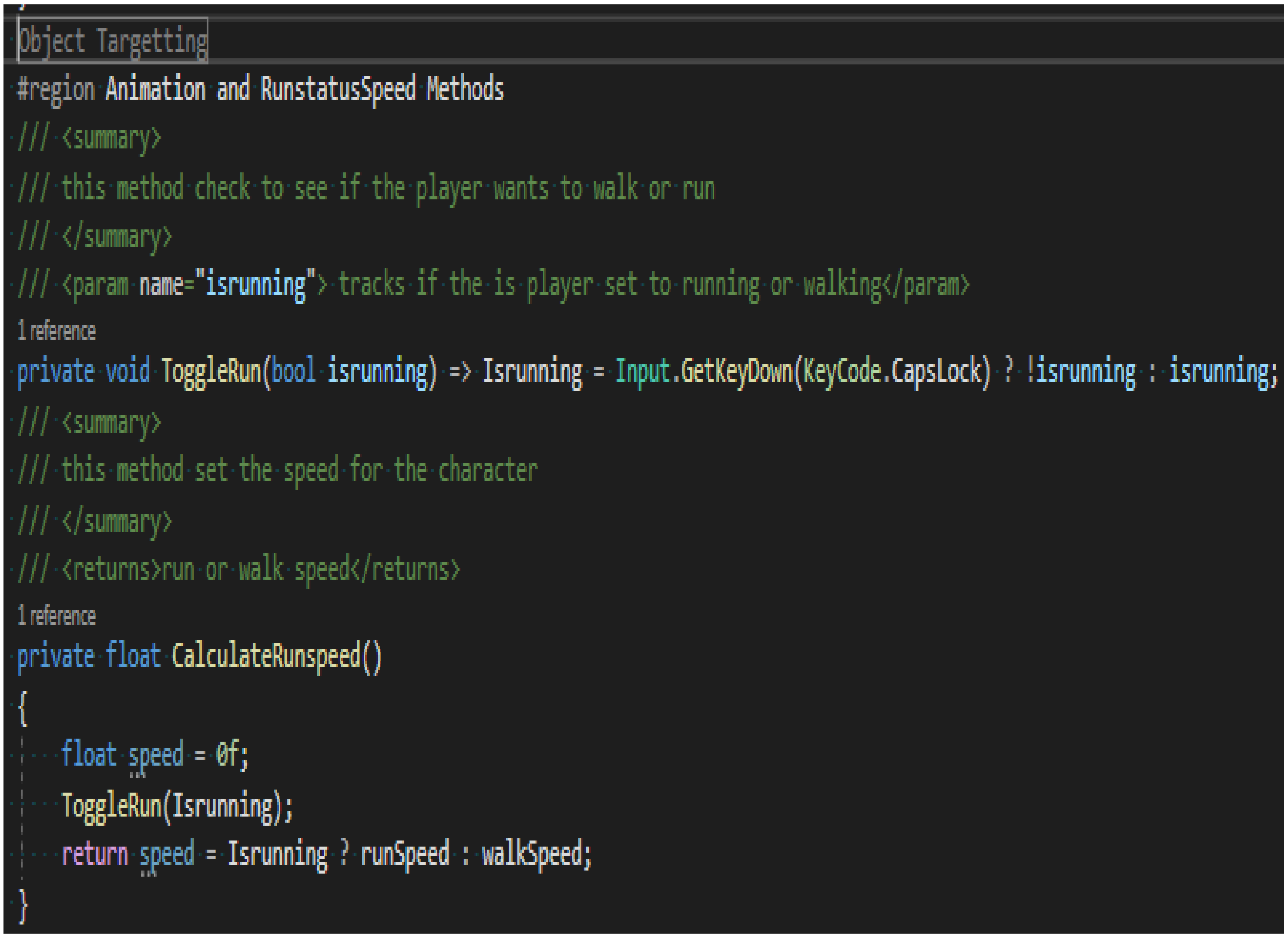
```
#region Object Targetting
/// <summary>
/// the method that targets objects in the world (ai method)
/// </summary>
/// <param name="newFocus">the new target</param>
1 reference
private void SetFocus(Interactable newFocus)
{
    if (newFocus != null)
        focus = newFocus;
    focus.IsFocused = true;
    FollowTarget();
    Debug.Log("focused");
    focus.HasInteracted = false;
}
```



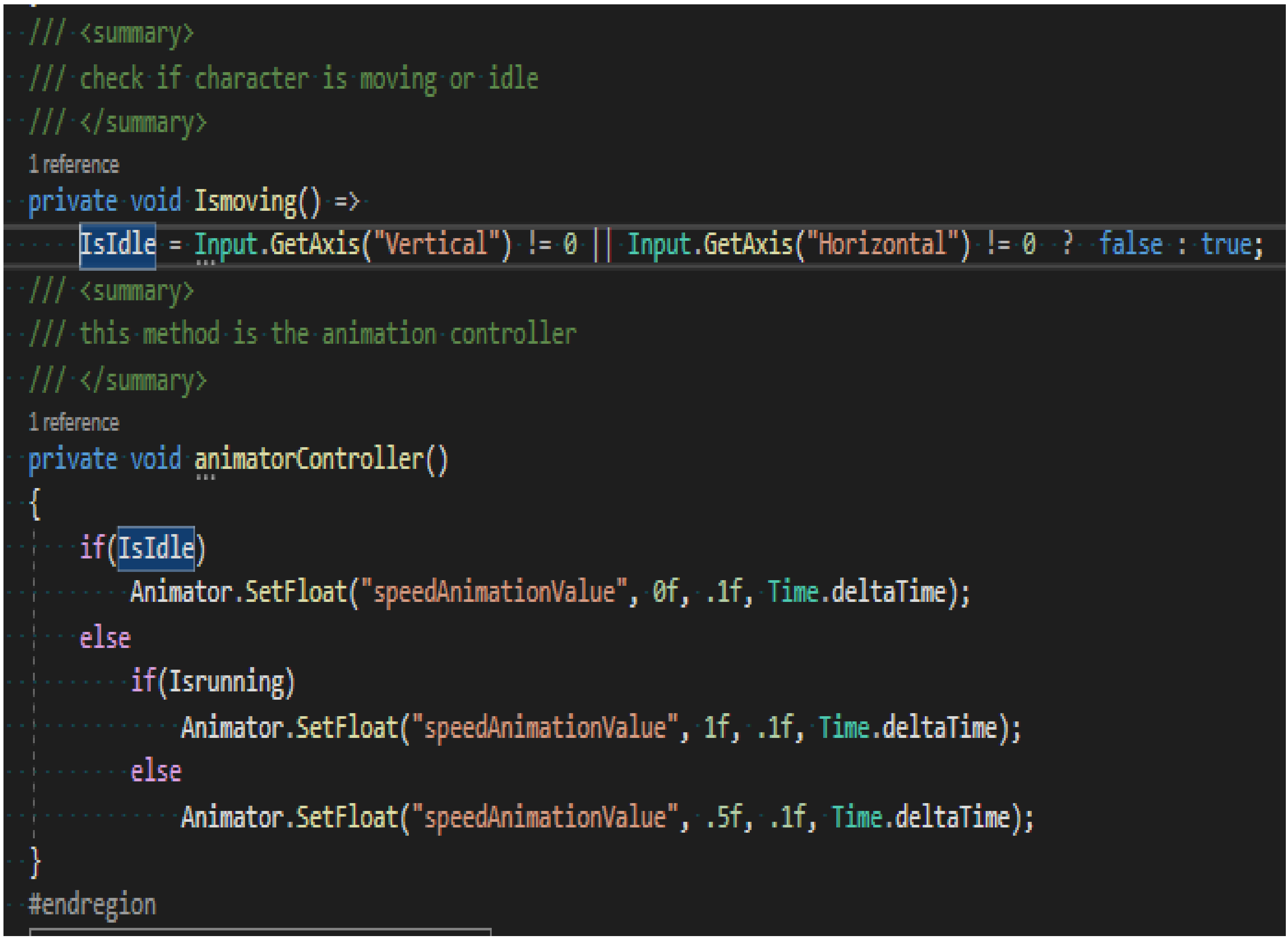
```
/// <summary>
/// un target something (ai method)
/// </summary>
1 reference
private void DeFocus()
{
    if (focus != null)
    {
        focus.IsFocused = false;
        focus.HasInteracted = false;
        focus = null;
        if (target != null)
            target = null;
        navMeshAgentF.stoppingDistance = 0f;
    }
}
```



```
/// <summary>
/// follow targetted object (ai method)
/// </summary>
1 reference
private void FollowTarget()
{
    if (focus != null)
        target = focus.transform;
    if (target != null)
    {
        navMeshAgentF.stoppingDistance = focus.Radius * 0.9f;
        navMeshAgentF.updateRotation = false;
        navMeshAgentF.SetDestination(target.position);
        Vector3 direction = (this.transform.position - target.position).normalized;
        this.transform.rotation = Quaternion.Slerp(this.transform.rotation,
            Quaternion.LookRotation(new Vector3(direction.x, 0f, direction.z)), Time.deltaTime * 5f);
    }
}
#endregion
```

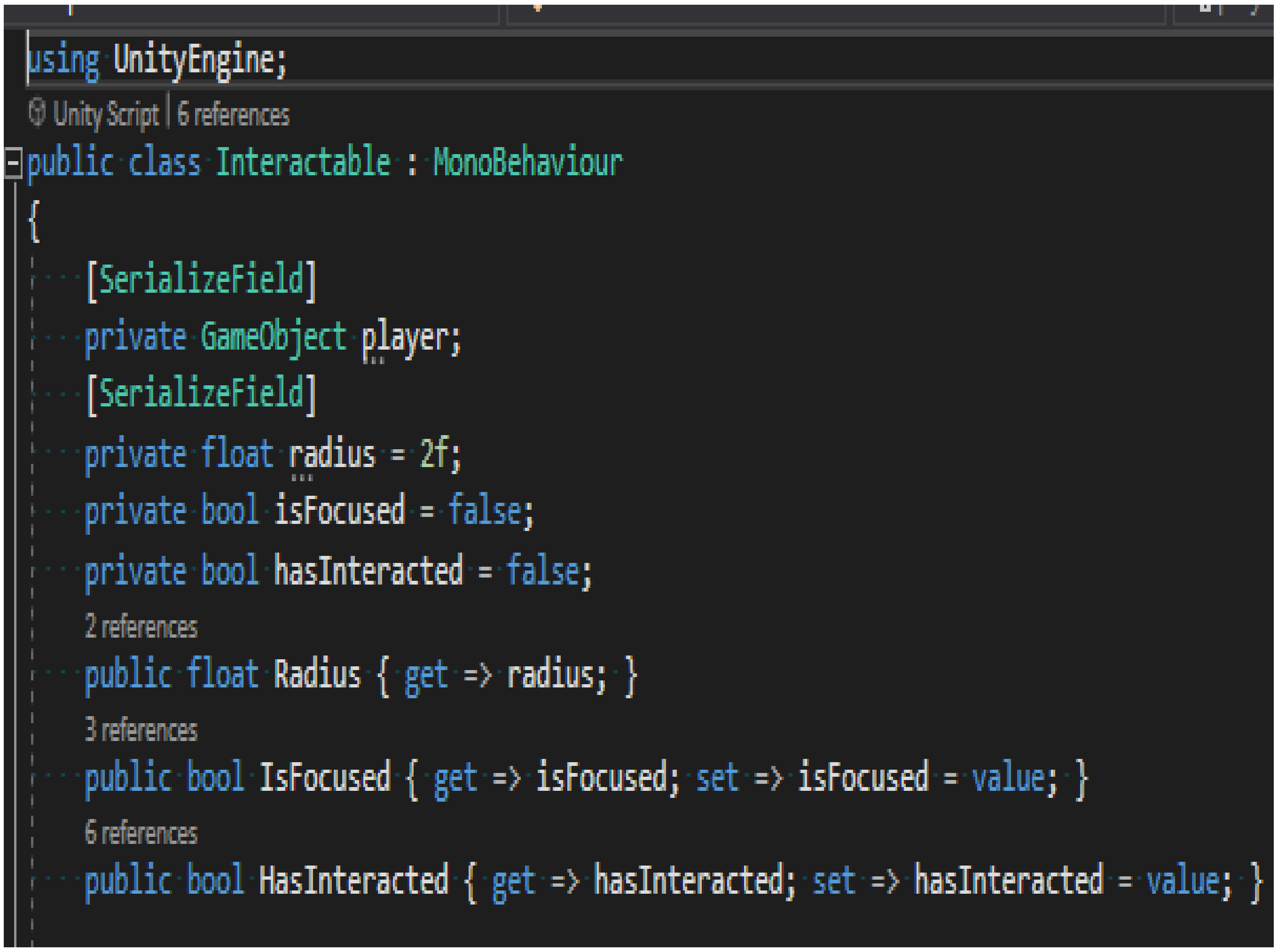


```
Object Targetting
#region Animation and RunstatusSpeed Methods
/// <summary>
/// this method check to see if the player wants to walk or run
/// </summary>
/// <param name="isrunning"> tracks if the is player set to running or walking</param>
1 reference
private void ToggleRun(bool isrunning) => Isrunning = Input.GetKeyDown(KeyCode.CapsLock) ? !isrunning : isrunning;
/// <summary>
/// this method set the speed for the character
/// </summary>
/// <returns>run or walk speed</returns>
1 reference
private float CalculateRunspeed()
{
    float speed = 0f;
    ToggleRun(Isrunning);
    return speed = Isrunning ? runSpeed : walkSpeed;
}
```

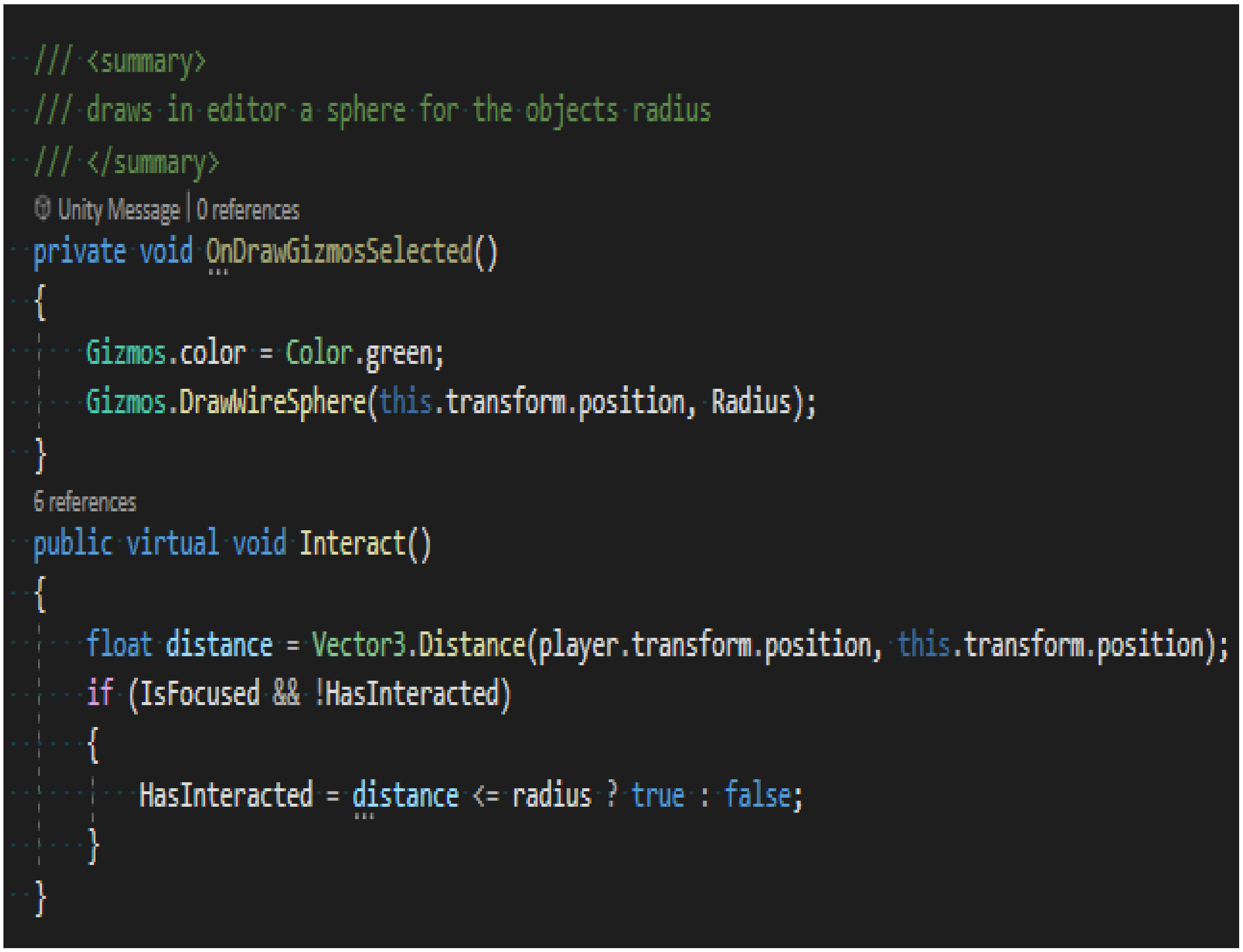


```
/// <summary>
/// check if character is moving or idle
/// </summary>
1 reference
private void Ismoving() =>
    IsIdle = Input.GetAxis("Vertical") != 0 || Input.GetAxis("Horizontal") != 0  ? false : true;
/// <summary>
/// this method is the animation controller
/// </summary>
1 reference
private void animatorController()
{
    if(IsIdle)
        Animator.SetFloat("speedAnimationValue", 0f, .1f, Time.deltaTime);
    else
        if(Isrunning)
            Animator.SetFloat("speedAnimationValue", 1f, .1f, Time.deltaTime);
        else
            Animator.SetFloat("speedAnimationValue", .5f, .1f, Time.deltaTime);
}
#endregion
```

Intractable class

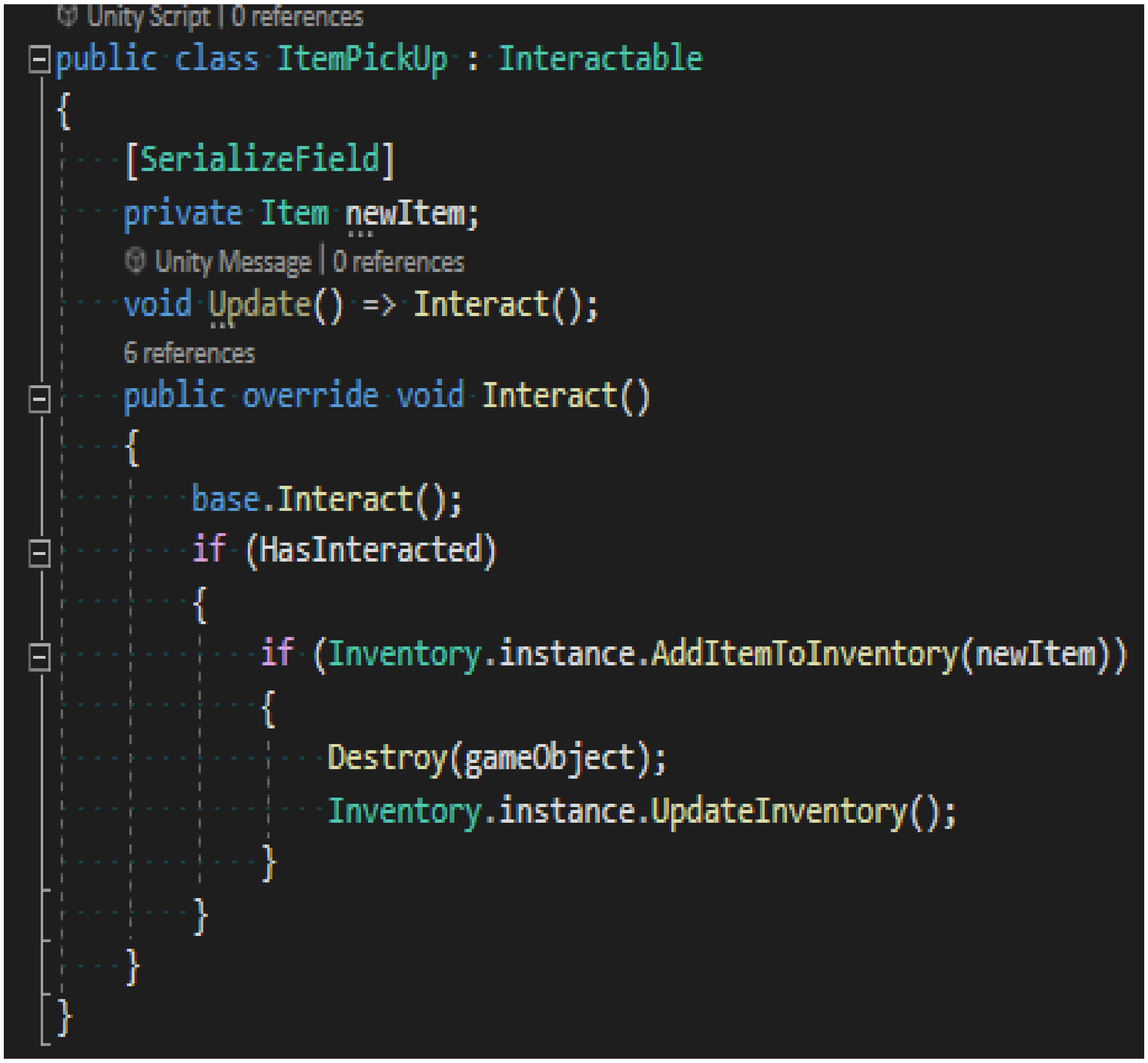


```
using UnityEngine;
Unity Script | 6 references
public class Interactable : MonoBehaviour
{
    [SerializeField]
    private GameObject player;
    [SerializeField]
    private float radius = 2f;
    private bool isFocused = false;
    private bool hasInteracted = false;
    2 references
    public float Radius { get => radius; }
    3 references
    public bool IsFocused { get => isFocused; set => isFocused = value; }
    6 references
    public bool HasInteracted { get => hasInteracted; set => hasInteracted = value; }
```



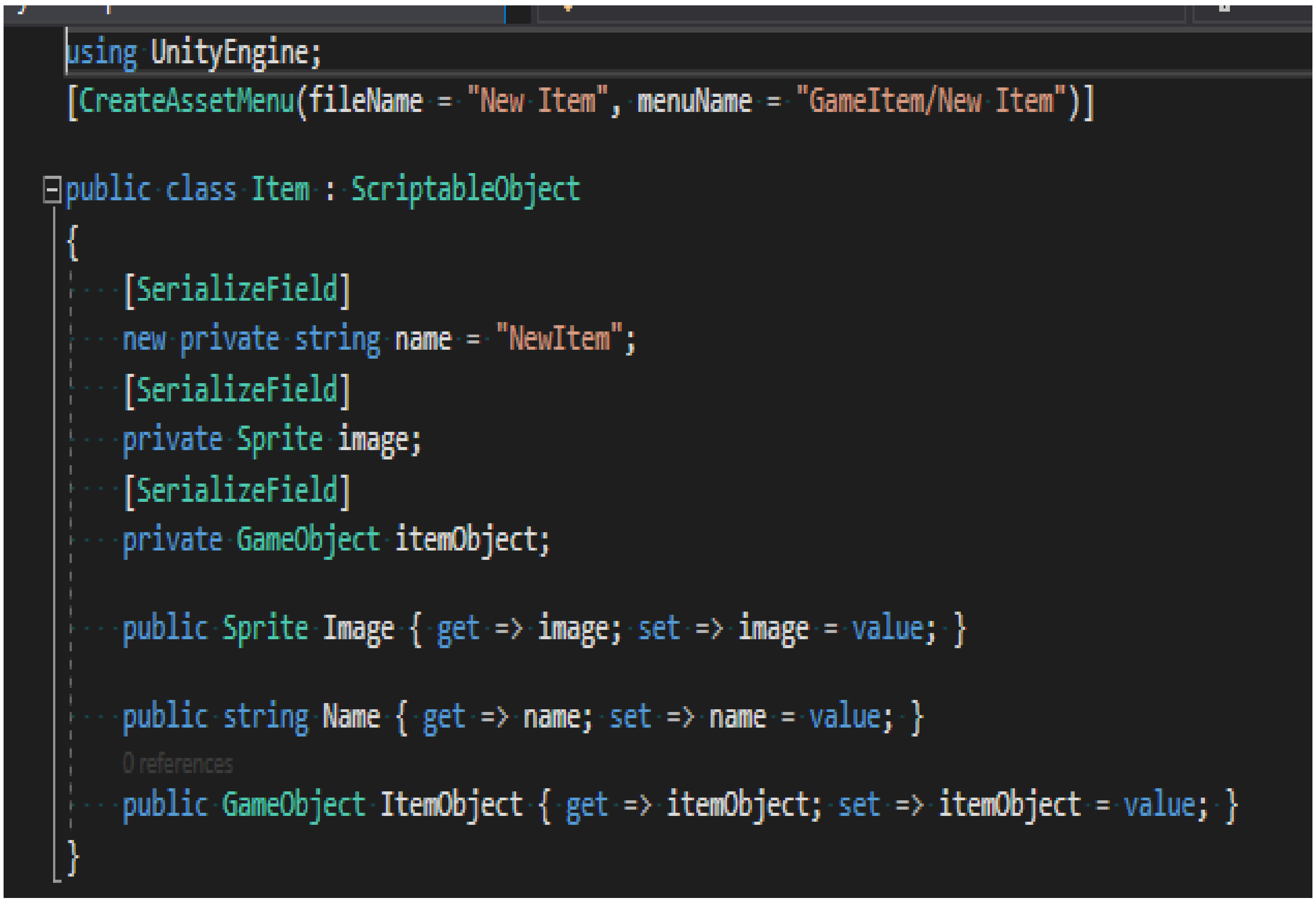
```
/// <summary>
/// draws in editor a sphere for the objects radius
/// </summary>
Unity Message | 0 references
private void OnDrawGizmosSelected()
{
    Gizmos.color = Color.green;
    Gizmos.DrawWireSphere(this.transform.position, Radius);
}
6 references
public virtual void Interact()
{
    float distance = Vector3.Distance(player.transform.position, this.transform.position);
    if (IsFocused && !HasInteracted)
    {
        HasInteracted = distance <= radius ? true : false;
    }
}
```

Item pickup class



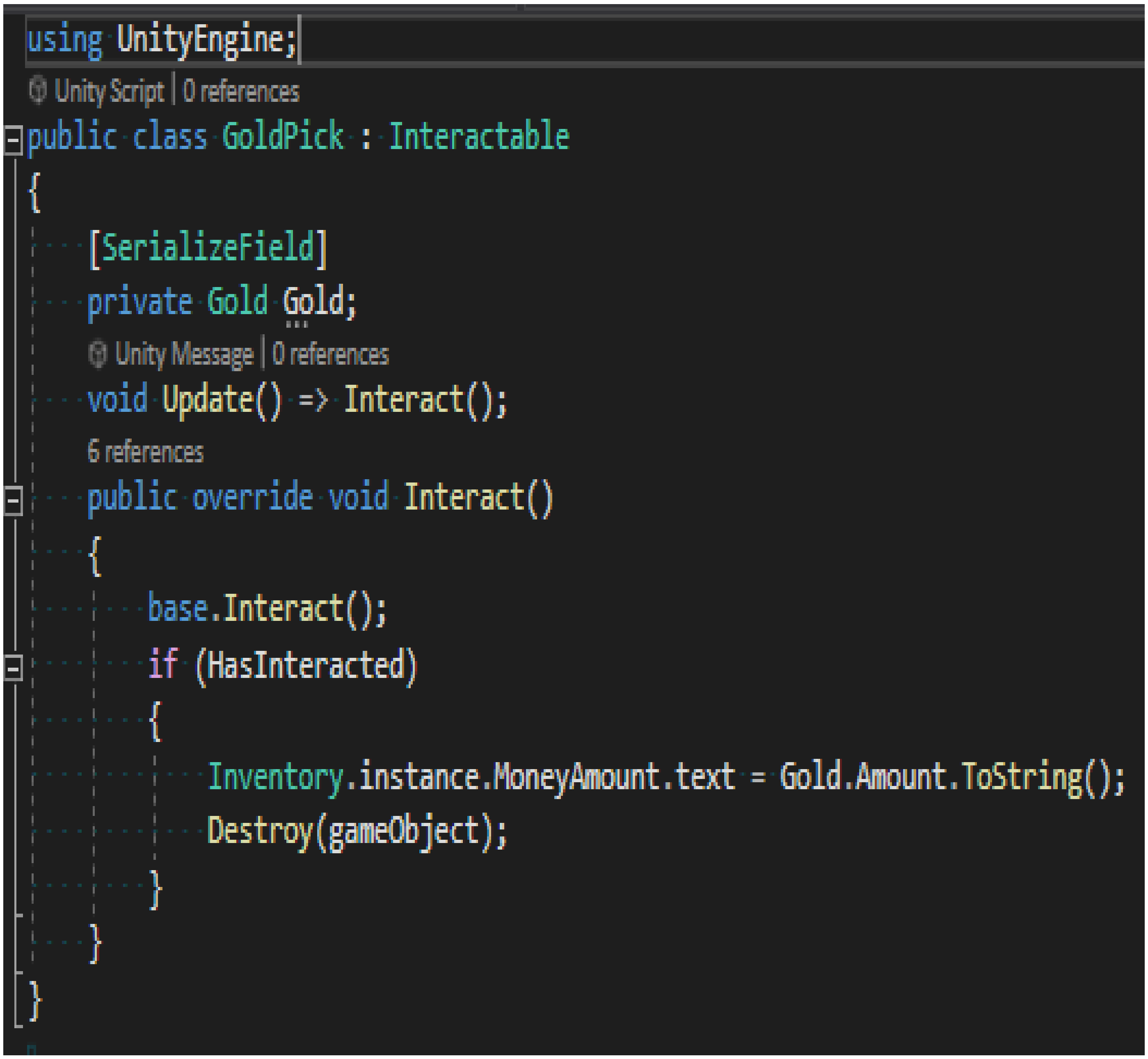
```
Unity Script | 0 references
public class ItemPickUp : Interactable
{
    [SerializeField]
    private Item newItem;
    Unity Message | 0 references
    void Update() => Interact();
    6 references
    public override void Interact()
    {
        base.Interact();
        if (HasInteracted)
        {
            if (Inventory.instance.AddItemToInventory(newItem))
            {
                Destroy(gameObject);
                Inventory.instance.UpdateInventory();
            }
        }
    }
}
```

Item class



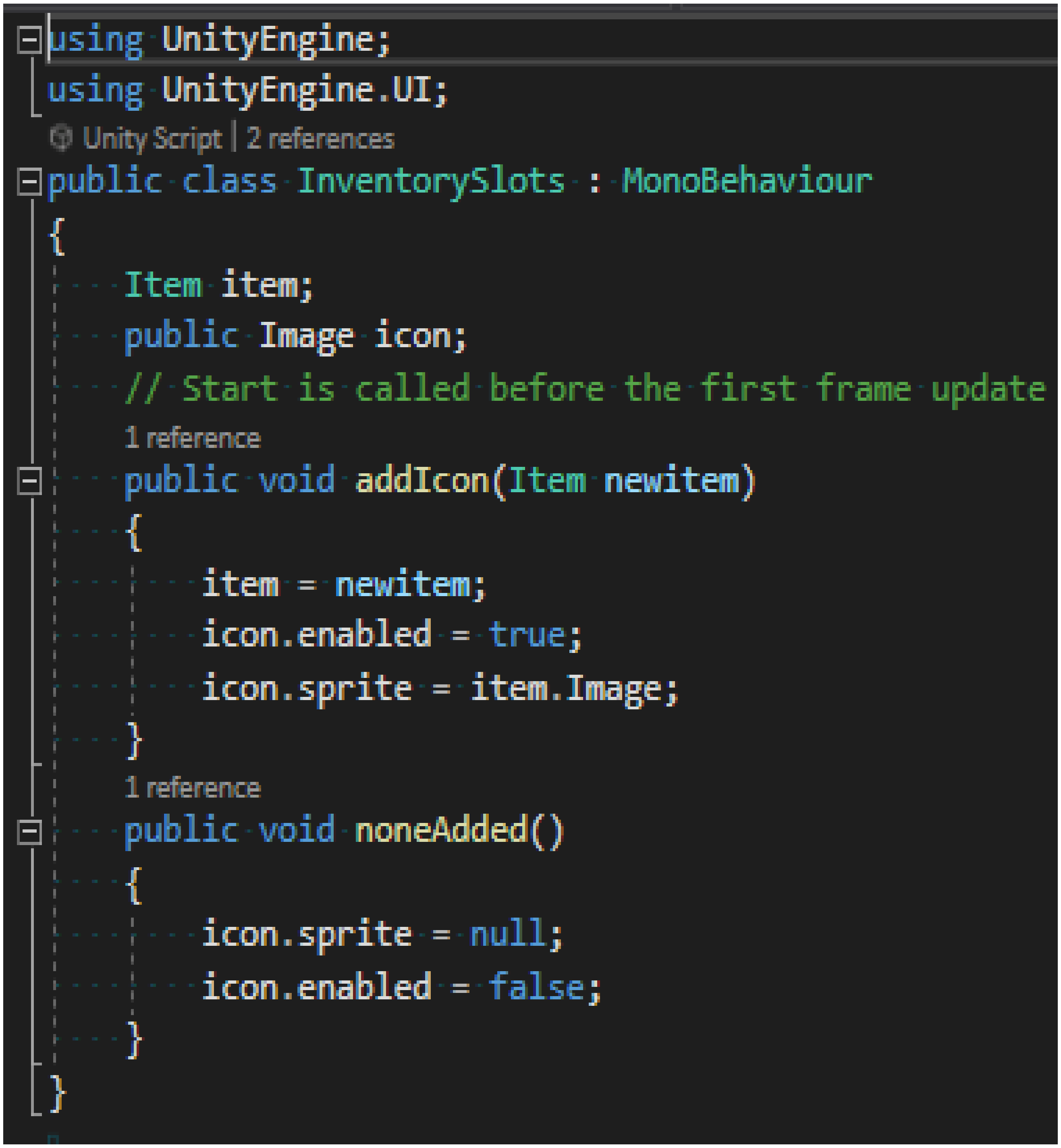
```
using UnityEngine;
[CreateAssetMenu(fileName = "New Item", menuName = "GameItem/New Item")]

public class Item : ScriptableObject
{
    [SerializeField]
    new private string name = "NewItem";
    [SerializeField]
    private Sprite image;
    [SerializeField]
    private GameObject itemObject;

    public Sprite Image { get => image; set => image = value; }

    public string Name { get => name; set => name = value; }
    0 references
    public GameObject ItemObject { get => itemObject; set => itemObject = value; }
}
```

Gold pickup class



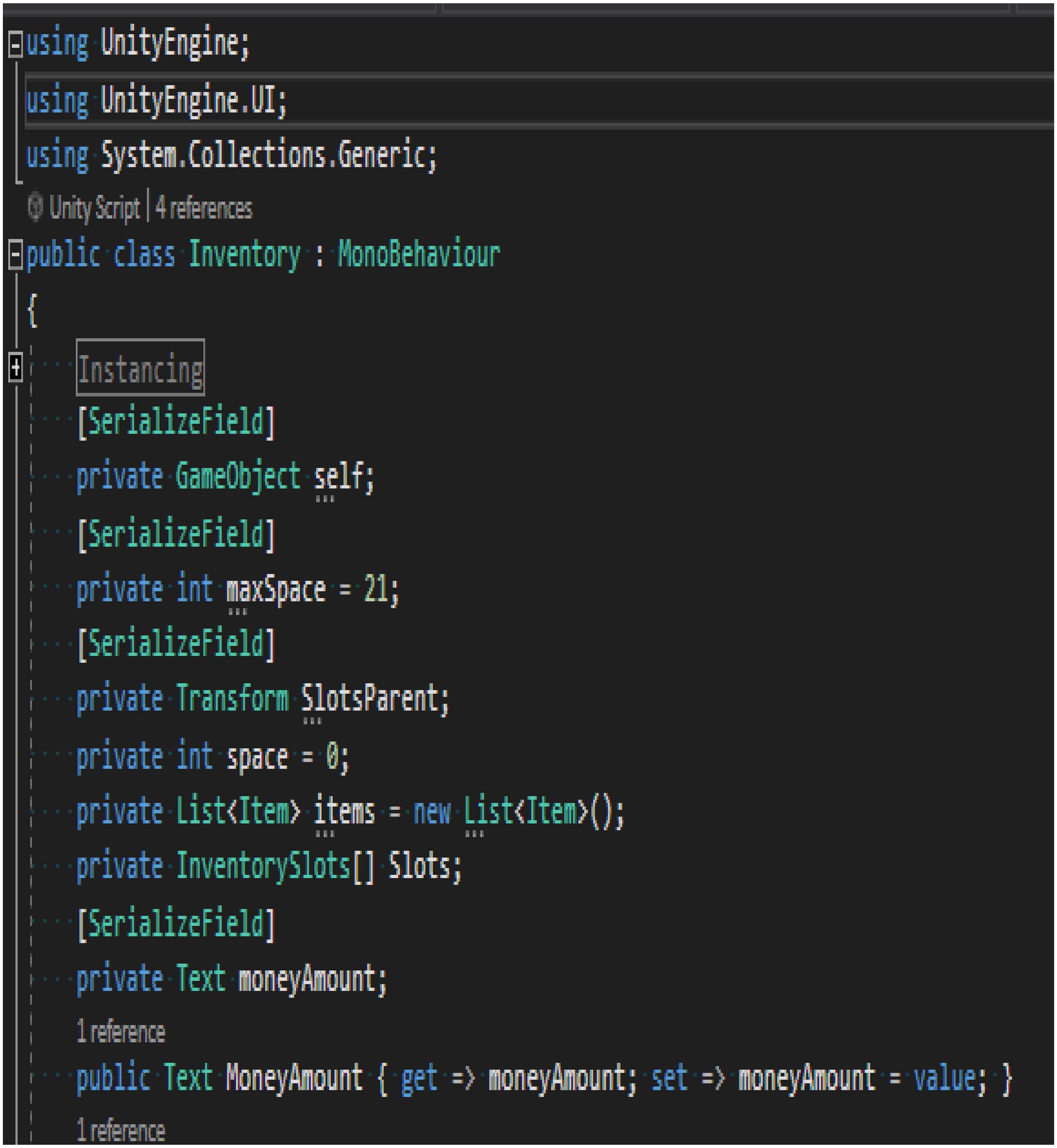
```
using UnityEngine;
Unity Script | 0 references
public class GoldPick : Interactable
{
    [SerializeField]
    private Gold Gold;
    Unity Message | 0 references
    void Update() => Interact();
    6 references
    public override void Interact()
    {
        base.Interact();
        if (HasInteracted)
        {
            Inventory.instance.MoneyAmount.text = Gold.Amount.ToString();
            Destroy(gameObject);
        }
    }
}
```

Inventory Slots

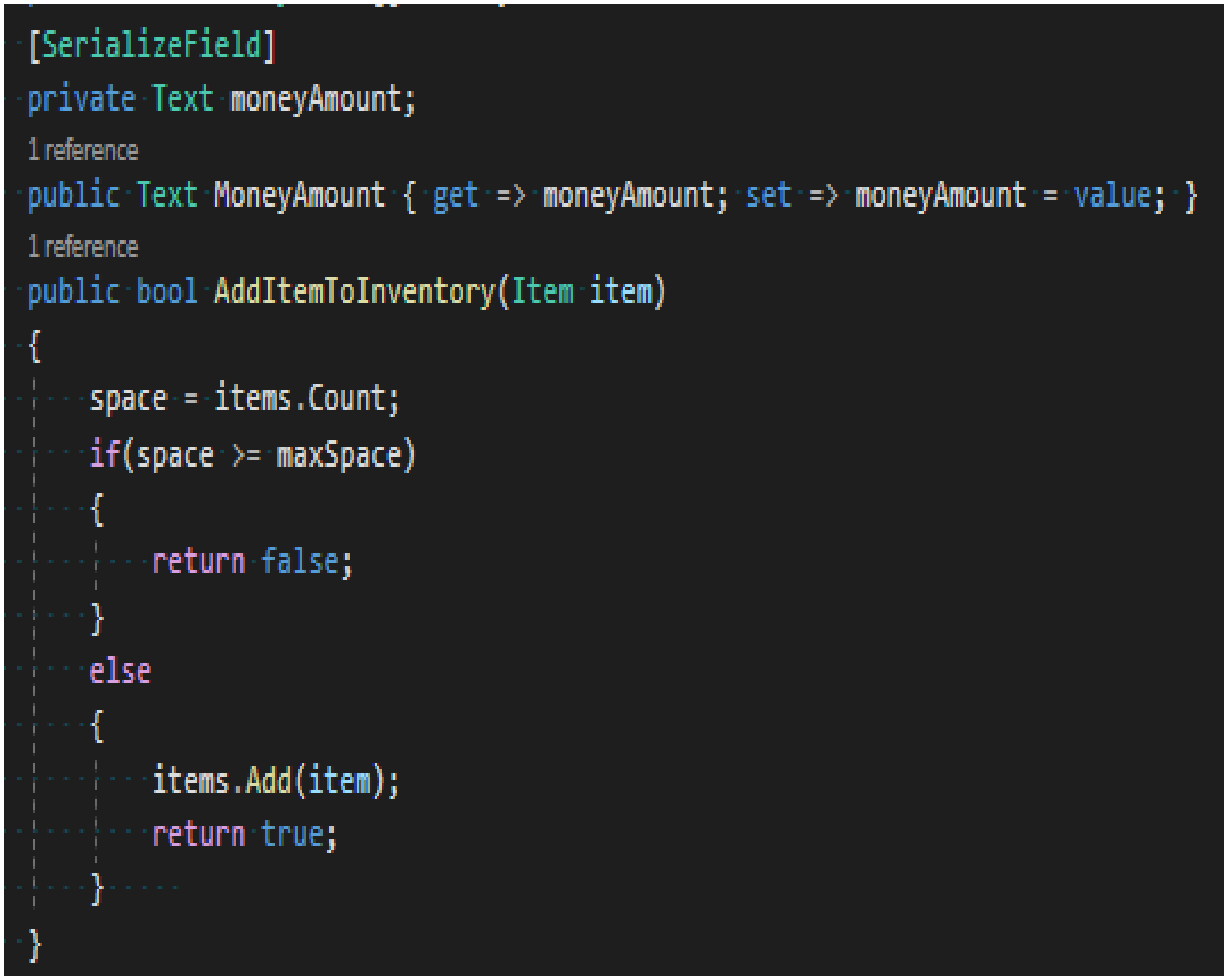
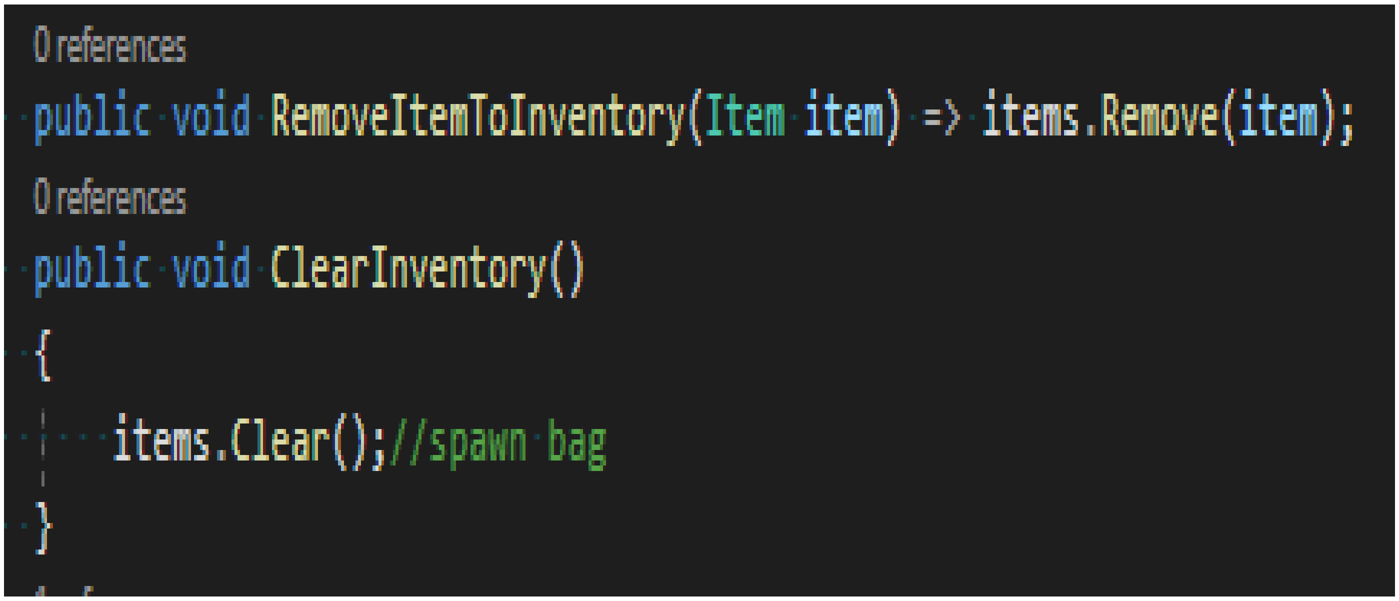


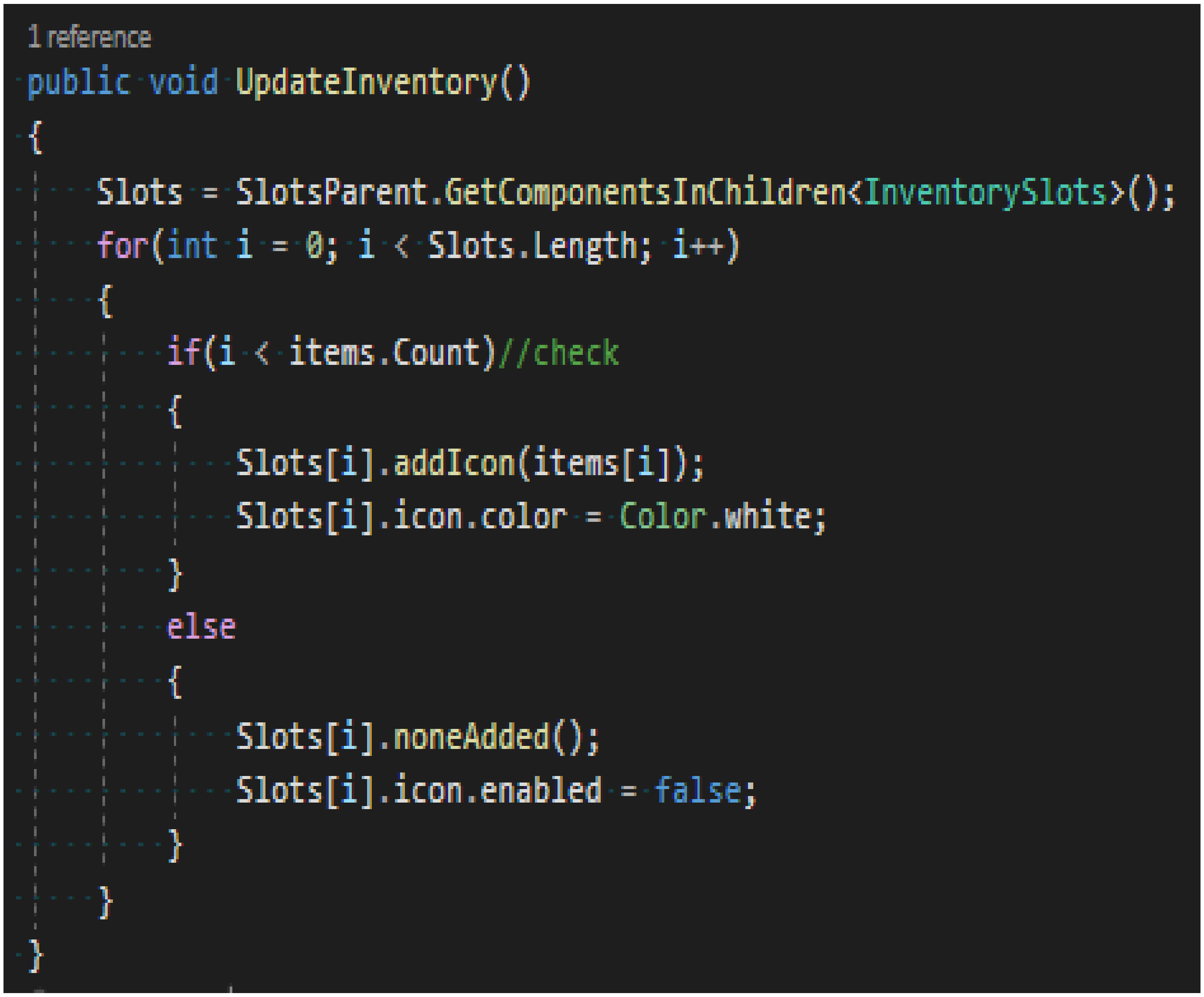
```
using UnityEngine;
using UnityEngine.UI;
Unity Script | 2 references
public class InventorySlots : MonoBehaviour
{
    Item item;
    public Image icon;
    // Start is called before the first frame update
    1 reference
    public void addIcon(Item newitem)
    {
        item = newitem;
        icon.enabled = true;
        icon.sprite = item.Image;
    }
    1 reference
    public void noneAdded()
    {
        icon.sprite = null;
        icon.enabled = false;
    }
}
```

Inventory

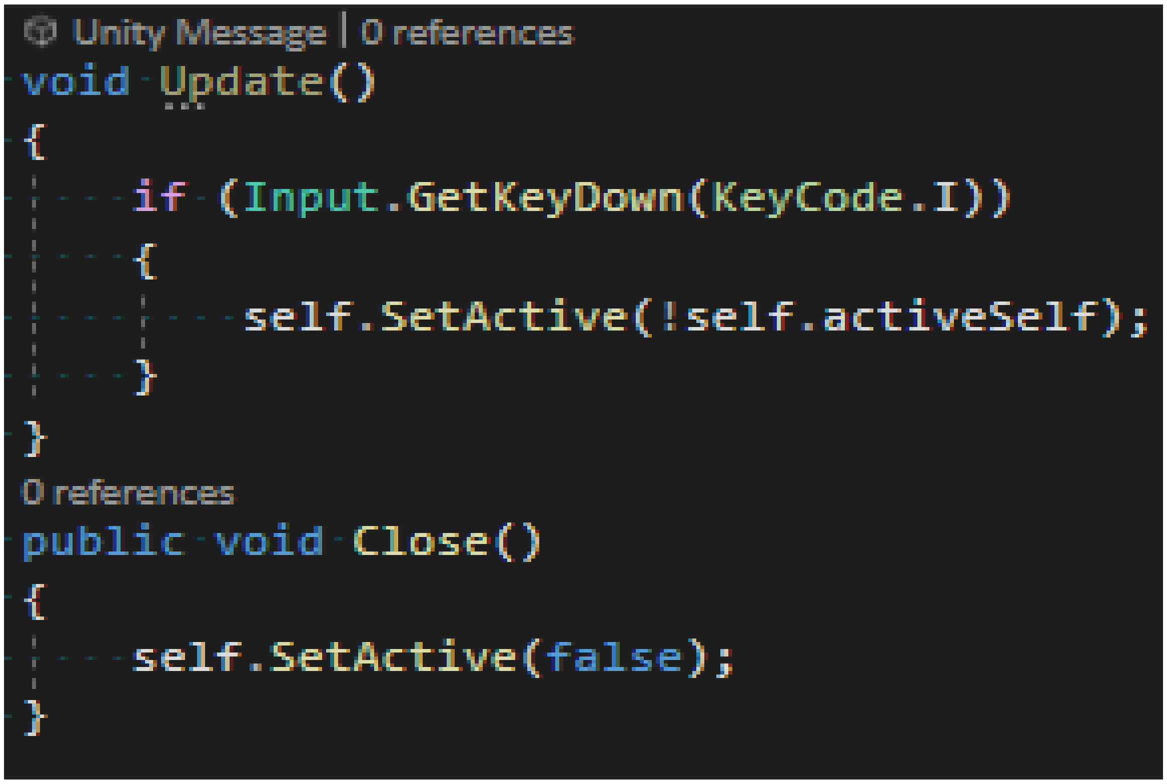


```
using UnityEngine;
using UnityEngine.UI;
using System.Collections.Generic;
Unity Script | 4 references
public class Inventory : MonoBehaviour
{
    Instancing
    [SerializeField]
    private GameObject self;
    [SerializeField]
    private int maxSpace = 21;
    [SerializeField]
    private Transform SlotsParent;
    private int space = 0;
    private List<Item> items = new List<Item>();
    private InventorySlots[] Slots;
    [SerializeField]
    private Text moneyAmount;
    1 reference
    public Text MoneyAmount { get => moneyAmount; set => moneyAmount = value; }
    1 reference
```


```
[SerializeField]
private Text moneyAmount;
1 reference
public Text MoneyAmount { get => moneyAmount; set => moneyAmount = value; }
1 reference
public bool AddItemToInventory(Item item)
{
    space = items.Count;
    if(space >= maxSpace)
    {
        return false;
    }
    else
    {
        items.Add(item);
        return true;
    }
}
```

```
0 references
public void RemoveItemToInventory(Item item) => items.Remove(item);
0 references
public void ClearInventory()
{
    items.Clear();//spawn bag
}
```

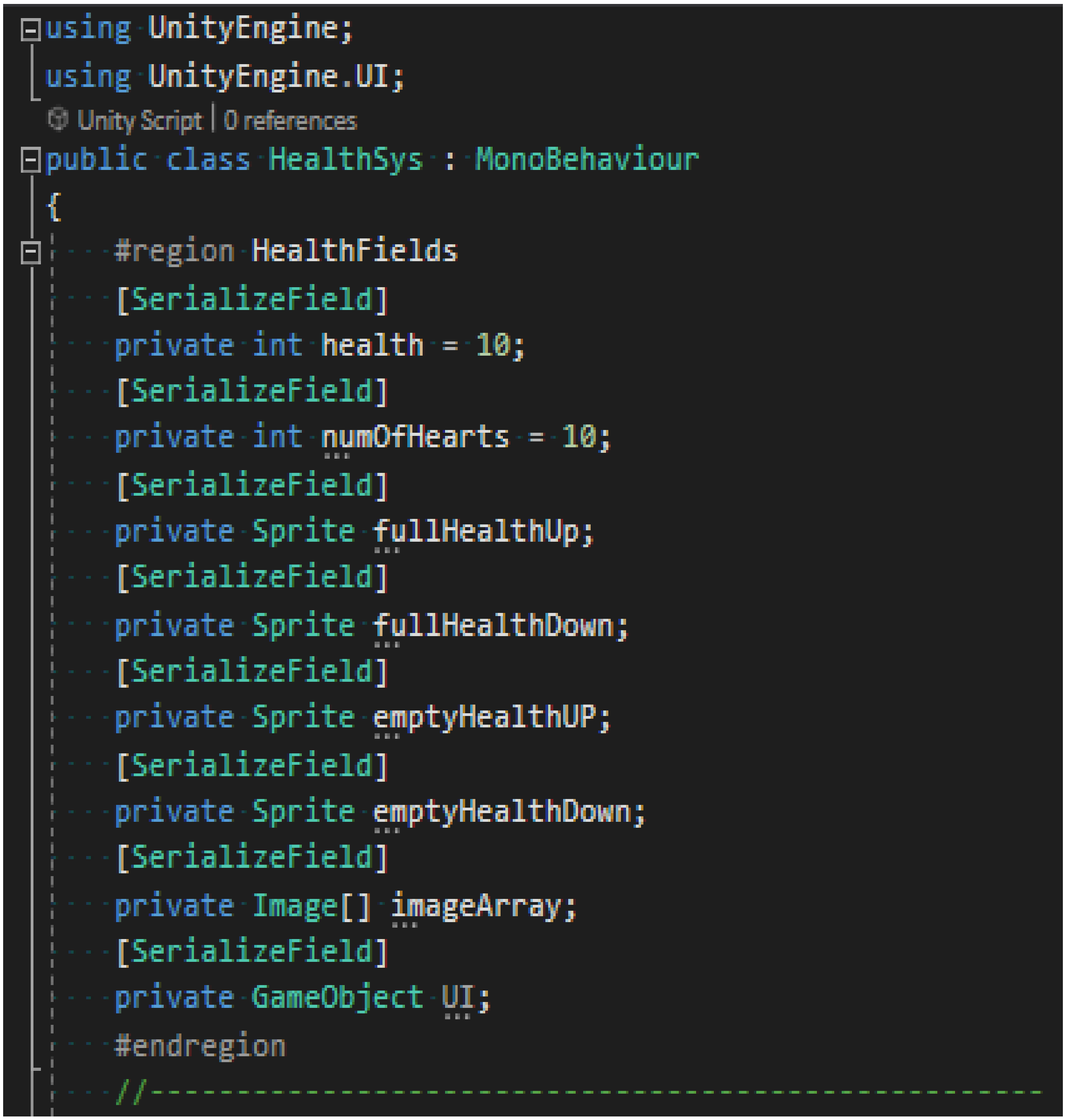


```
1 reference
public void UpdateInventory()
{
    Slots = SlotsParent.GetComponentsInChildren<InventorySlots>();
    for(int i = 0; i < Slots.Length; i++)
    {
        if(i < items.Count)//check
        {
            Slots[i].addIcon(items[i]);
            Slots[i].icon.color = Color.white;
        }
        else
        {
            Slots[i].noneAdded();
            Slots[i].icon.enabled = false;
        }
    }
}
```

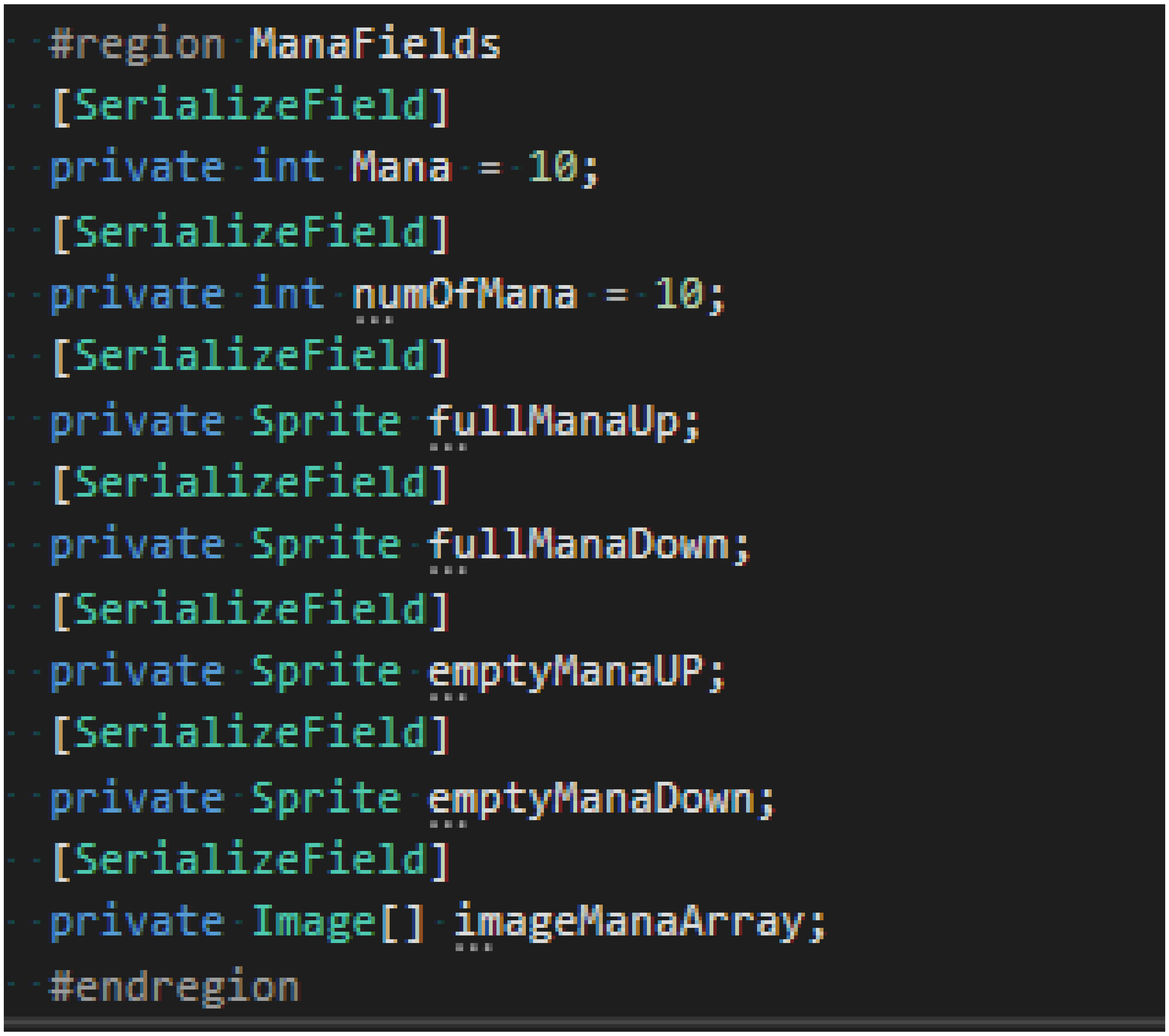


```
Unity Message | 0 references
void Update()
{
    if (Input.GetKeyDown(KeyCode.I))
    {
        self.SetActive(!self.activeSelf);
    }
}
0 references
public void Close()
{
    self.SetActive(false);
}
```

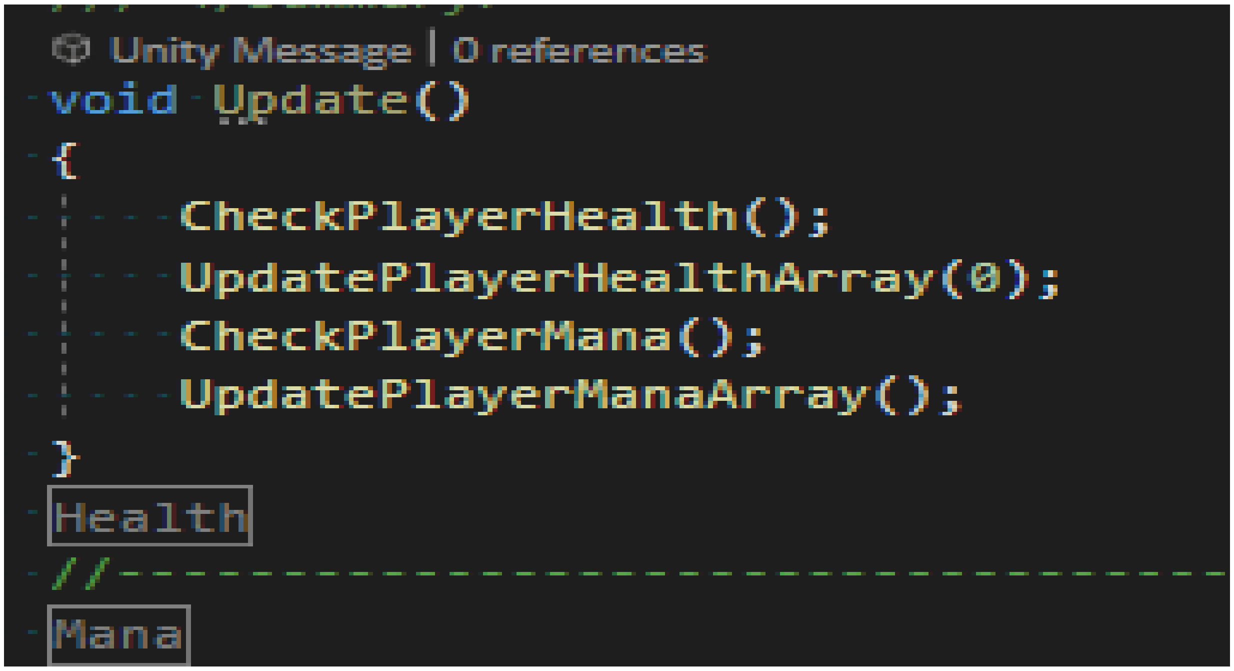
Health and mana system



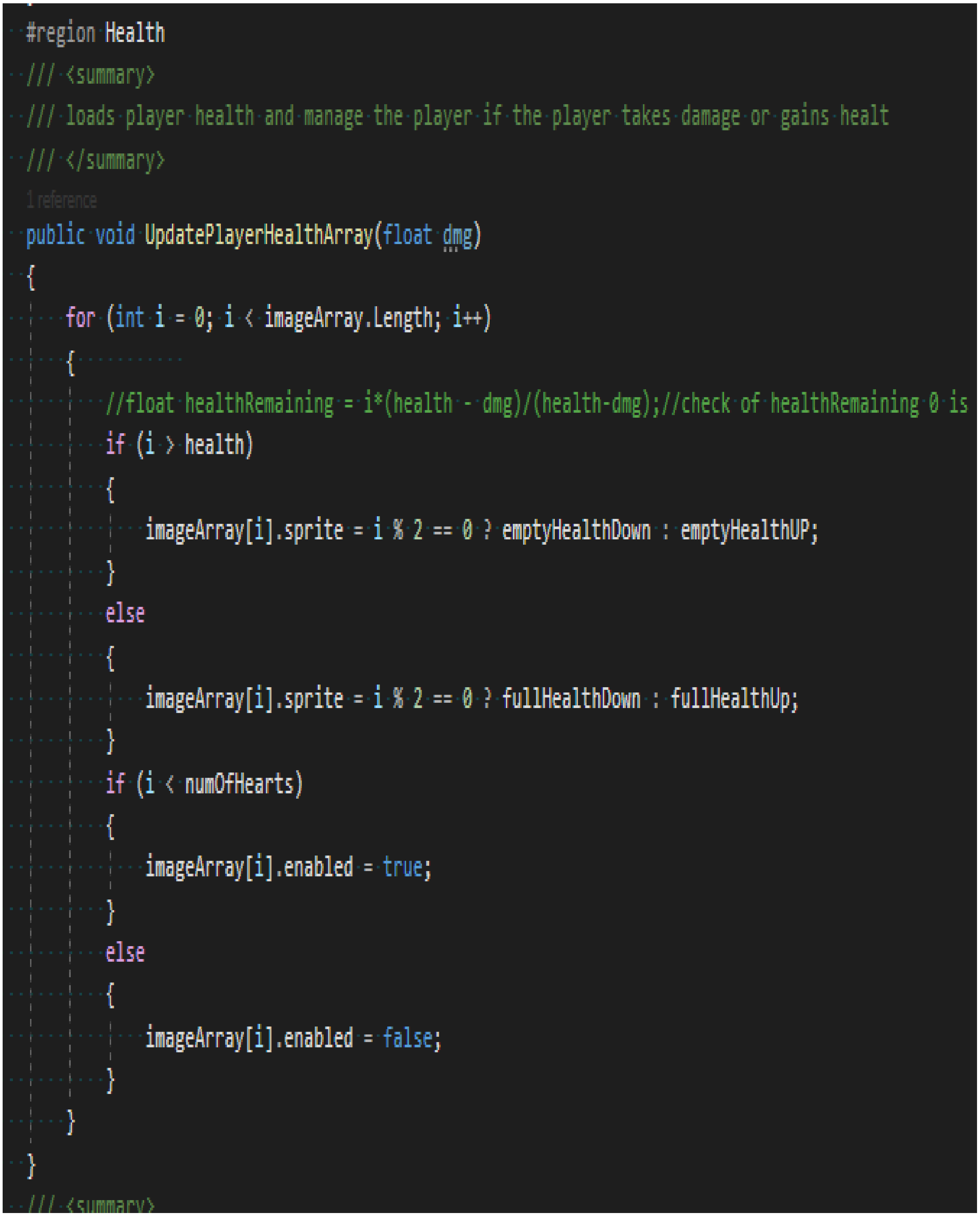
```
using UnityEngine;
using UnityEngine.UI;
Unity Script | 0 references
public class HealthSys : MonoBehaviour
{
    #region HealthFields
    [SerializeField]
    private int health = 10;
    [SerializeField]
    private int numOfHearts = 10;
    [SerializeField]
    private Sprite fullHealthUp;
    [SerializeField]
    private Sprite fullHealthDown;
    [SerializeField]
    private Sprite emptyHealthUP;
    [SerializeField]
    private Sprite emptyHealthDown;
    [SerializeField]
    private Image[] imageArray;
    [SerializeField]
    private GameObject UI;
    #endregion
    //--------------------------------------------------
```



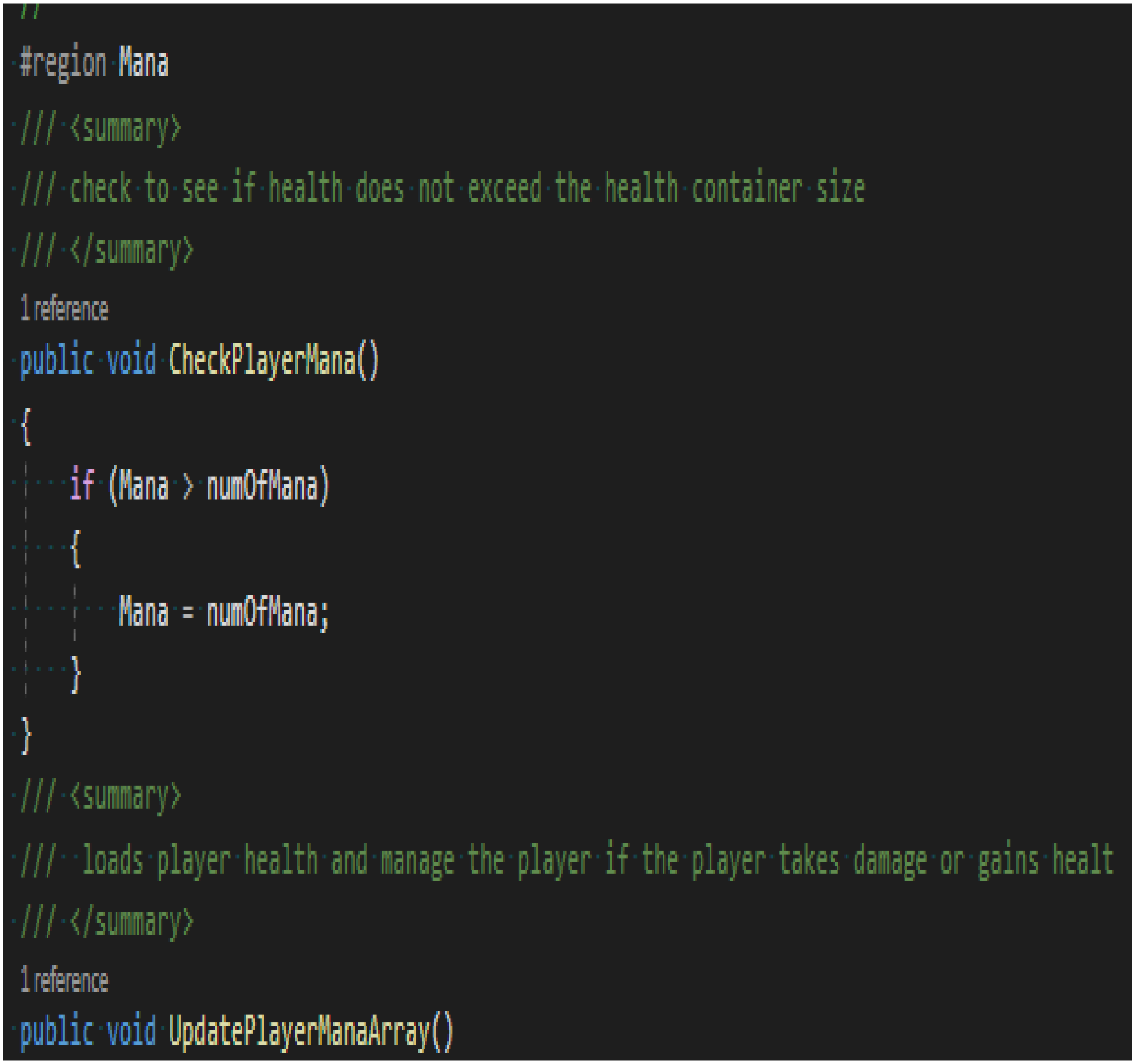
```
#region ManaFields
[SerializeField]
private int Mana = 10;
[SerializeField]
private int numOfMana = 10;
[SerializeField]
private Sprite fullManaUp;
[SerializeField]
private Sprite fullManaDown;
[SerializeField]
private Sprite emptyManaUP;
[SerializeField]
private Sprite emptyManaDown;
[SerializeField]
private Image[] imageManaArray;
#endregion
```



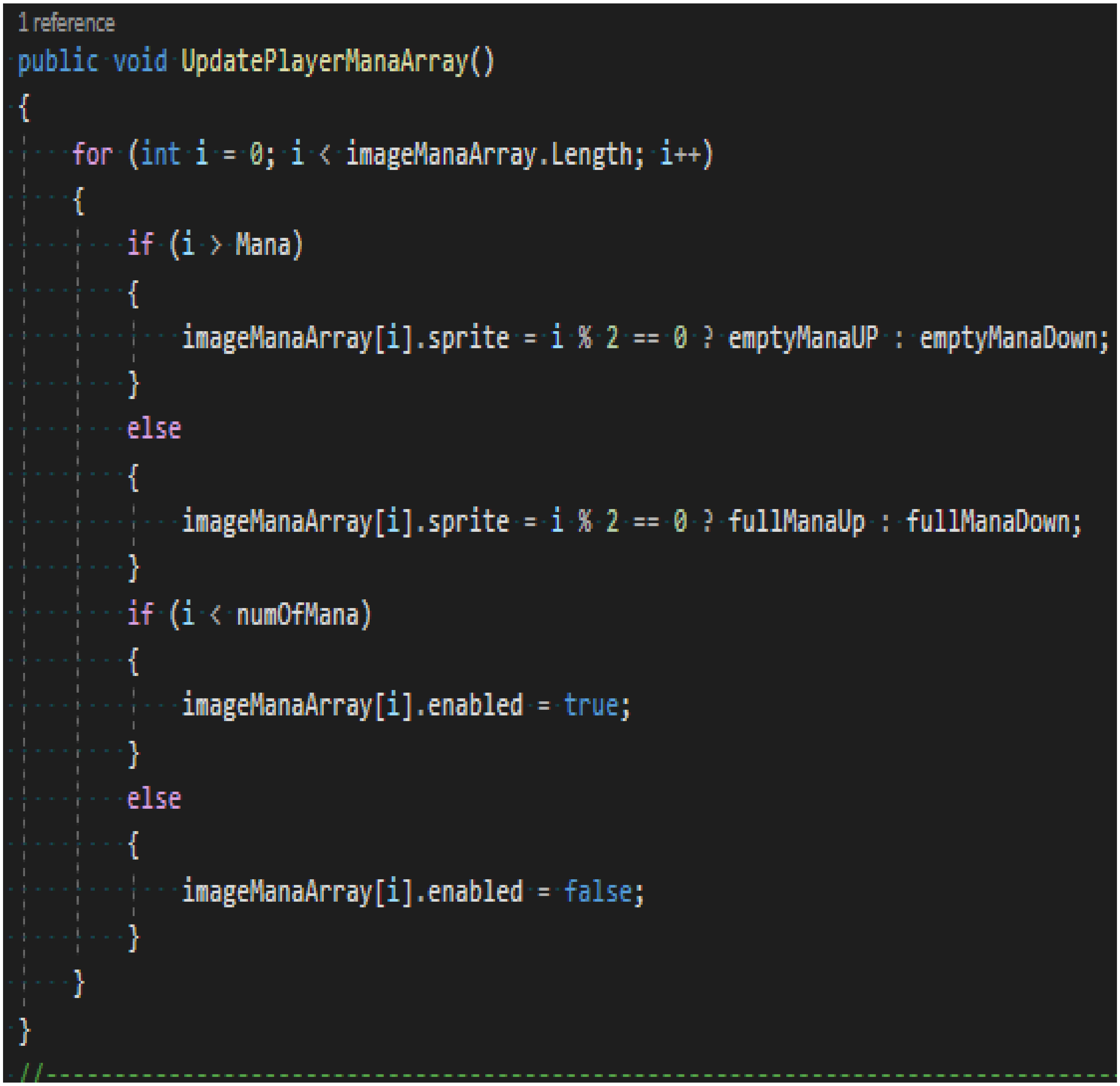
```
Unity Message | 0 references
void Update()
{
    CheckPlayerHealth();
    UpdatePlayerHealthArray(0);
    CheckPlayerMana();
    UpdatePlayerManaArray();
}
Health
//-------------------------------------------
Mana
```



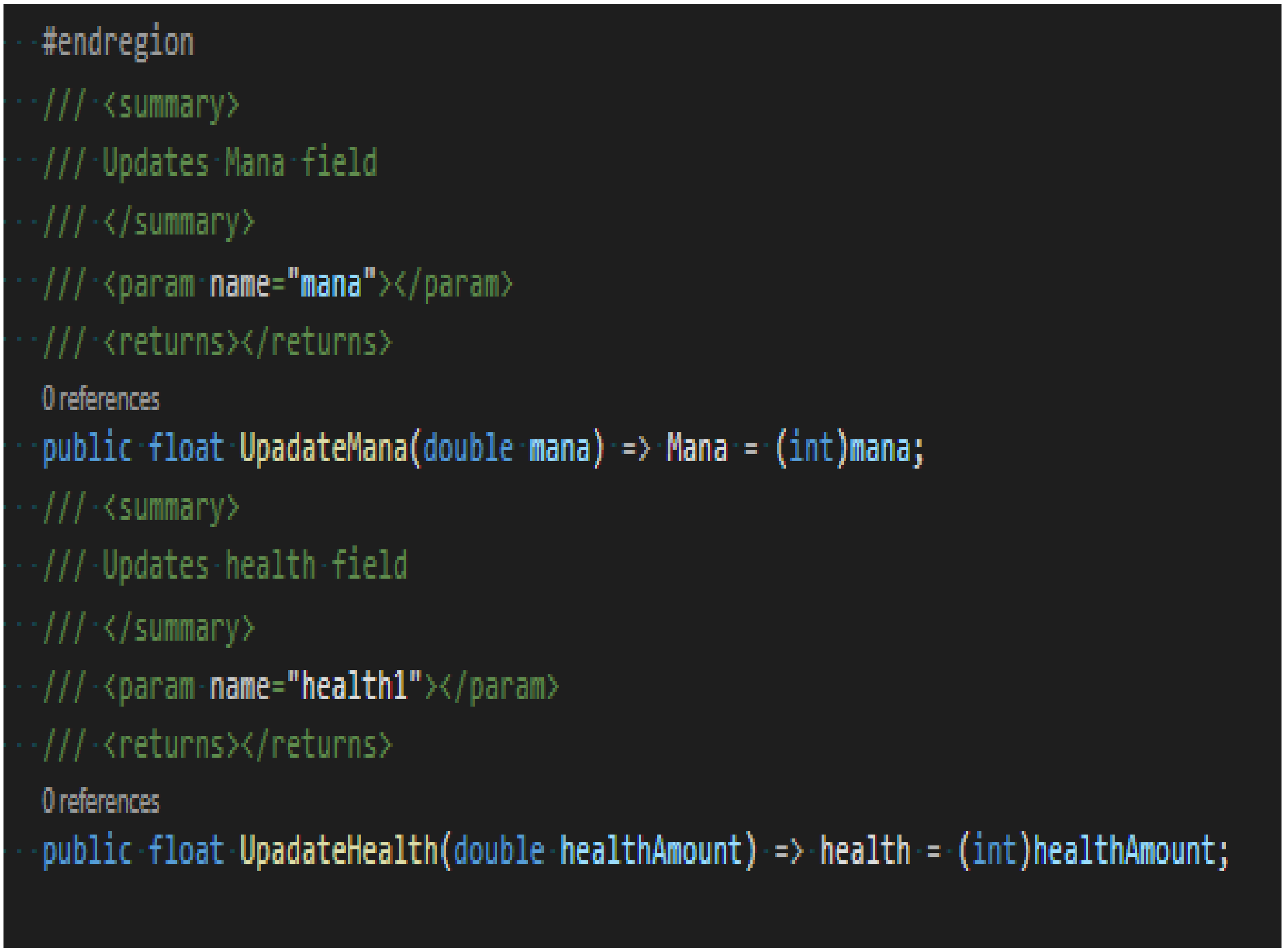
```
#region Health
/// <summary>
/// loads player health and manage the player if the player takes damage or gains healt
/// </summary>
1 reference
public void UpdatePlayerHealthArray(float dmg)
{
    for (int i = 0; i < imageArray.Length; i++)
    {
        //float healthRemaining = i*(health - dmg)/(health-dmg);//check of healthRemaining 0 is
        if (i > health)
        {
            imageArray[i].sprite = i % 2 == 0 ? emptyHealthDown : emptyHealthUP;
        }
        else
        {
            imageArray[i].sprite = i % 2 == 0 ? fullHealthDown : fullHealthUp;
        }
        if (i < numOfHearts)
        {
            imageArray[i].enabled = true;
        }
        else
        {
            imageArray[i].enabled = false;
        }
    }
}
/// <summary>
```



```
#region Mana
/// <summary>
/// check to see if health does not exceed the health container size
/// </summary>
1 reference
public void CheckPlayerMana()
{
    if (Mana > numOfMana)
    {
        Mana = numOfMana;
    }
}
/// <summary>
///  loads player health and manage the player if the player takes damage or gains healt
/// </summary>
1 reference
public void UpdatePlayerManaArray()
```



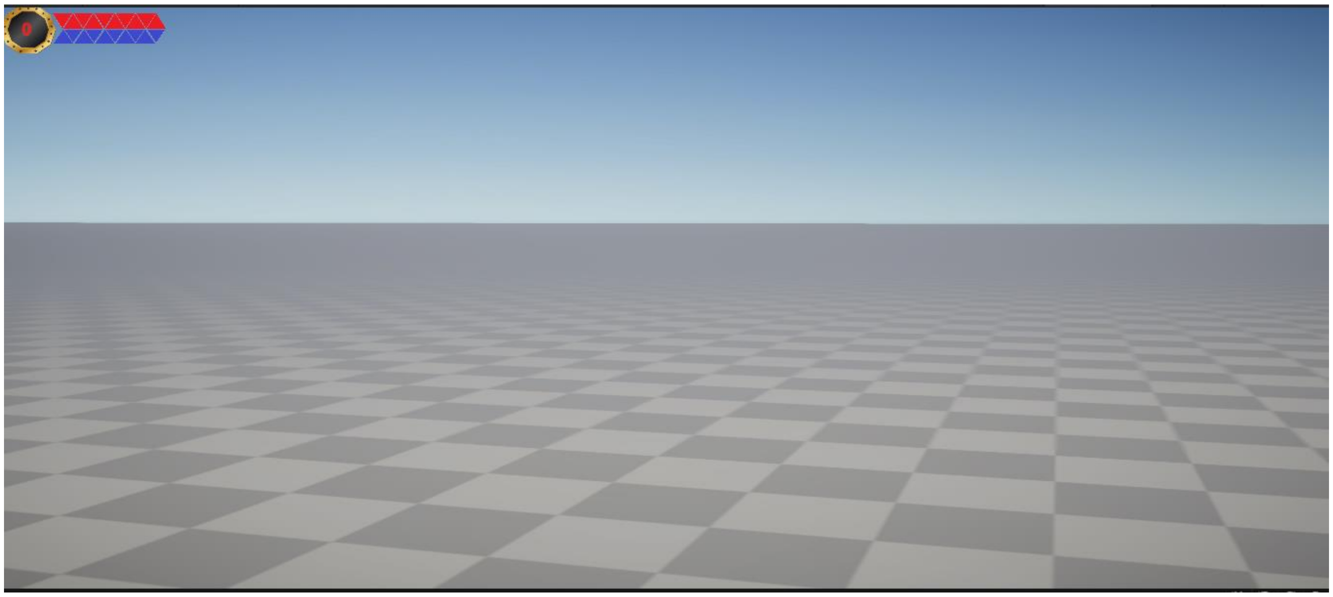
```
1 reference
public void UpdatePlayerManaArray()
{
    for (int i = 0; i < imageManaArray.Length; i++)
    {
        if (i > Mana)
        {
            imageManaArray[i].sprite = i % 2 == 0 ? emptyManaUP : emptyManaDown;
        }
        else
        {
            imageManaArray[i].sprite = i % 2 == 0 ? fullManaUp : fullManaDown;
        }
        if (i < numOfMana)
        {
            imageManaArray[i].enabled = true;
        }
        else
        {
            imageManaArray[i].enabled = false;
        }
    }
}
//---------------------------------------------------------------------------
```

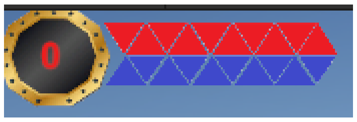


```
    #endregion
    /// <summary>
    /// Updates Mana field
    /// </summary>
    /// <param name="mana"></param>
    /// <returns></returns>
    0 references
    public float UpadateMana(double mana) => Mana = (int)mana;
    /// <summary>
    /// Updates health field
    /// </summary>
    /// <param name="health1"></param>
    /// <returns></returns>
    0 references
    public float UpadateHealth(double healthAmount) => health = (int)healthAmount;
```

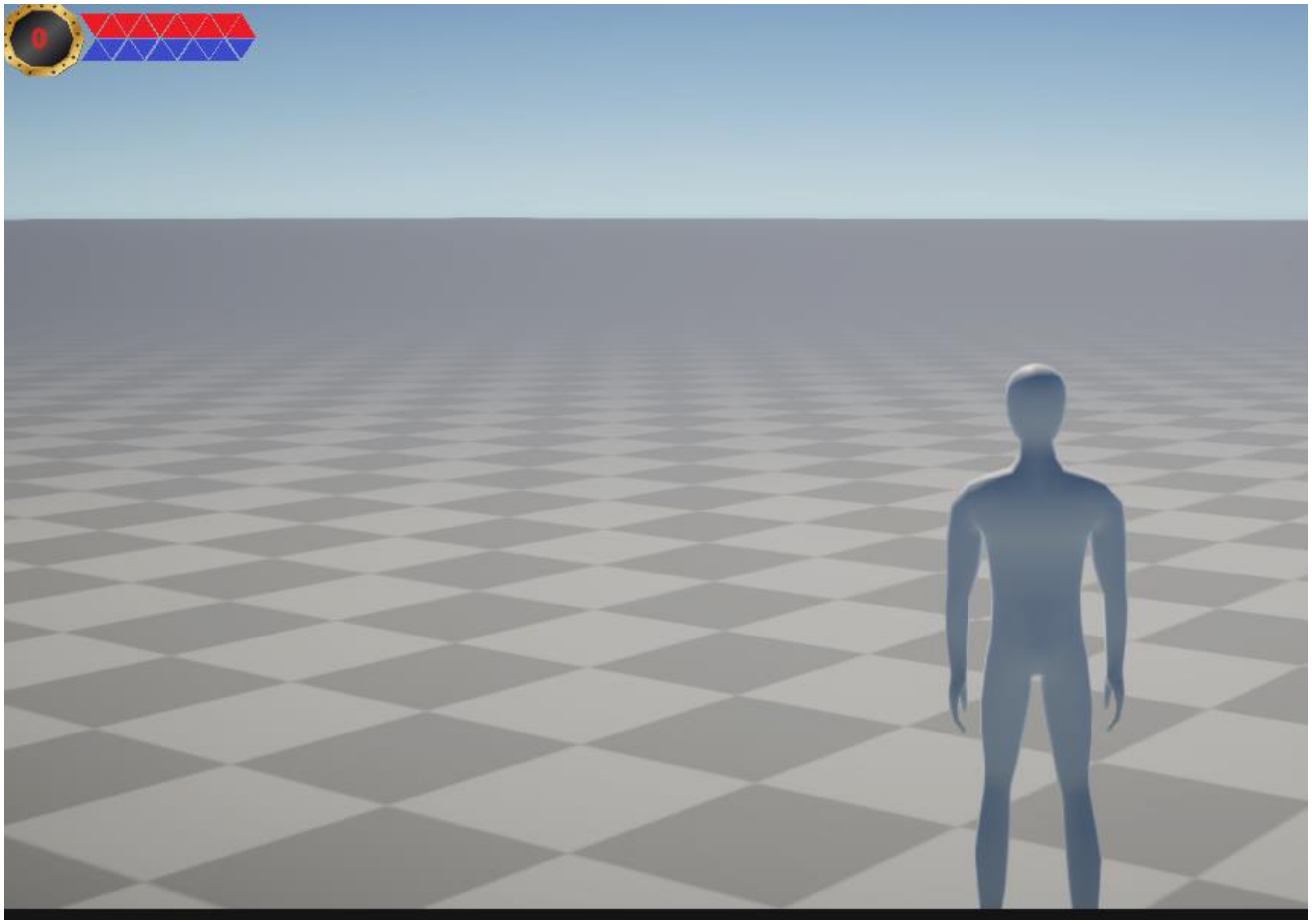
Images of game

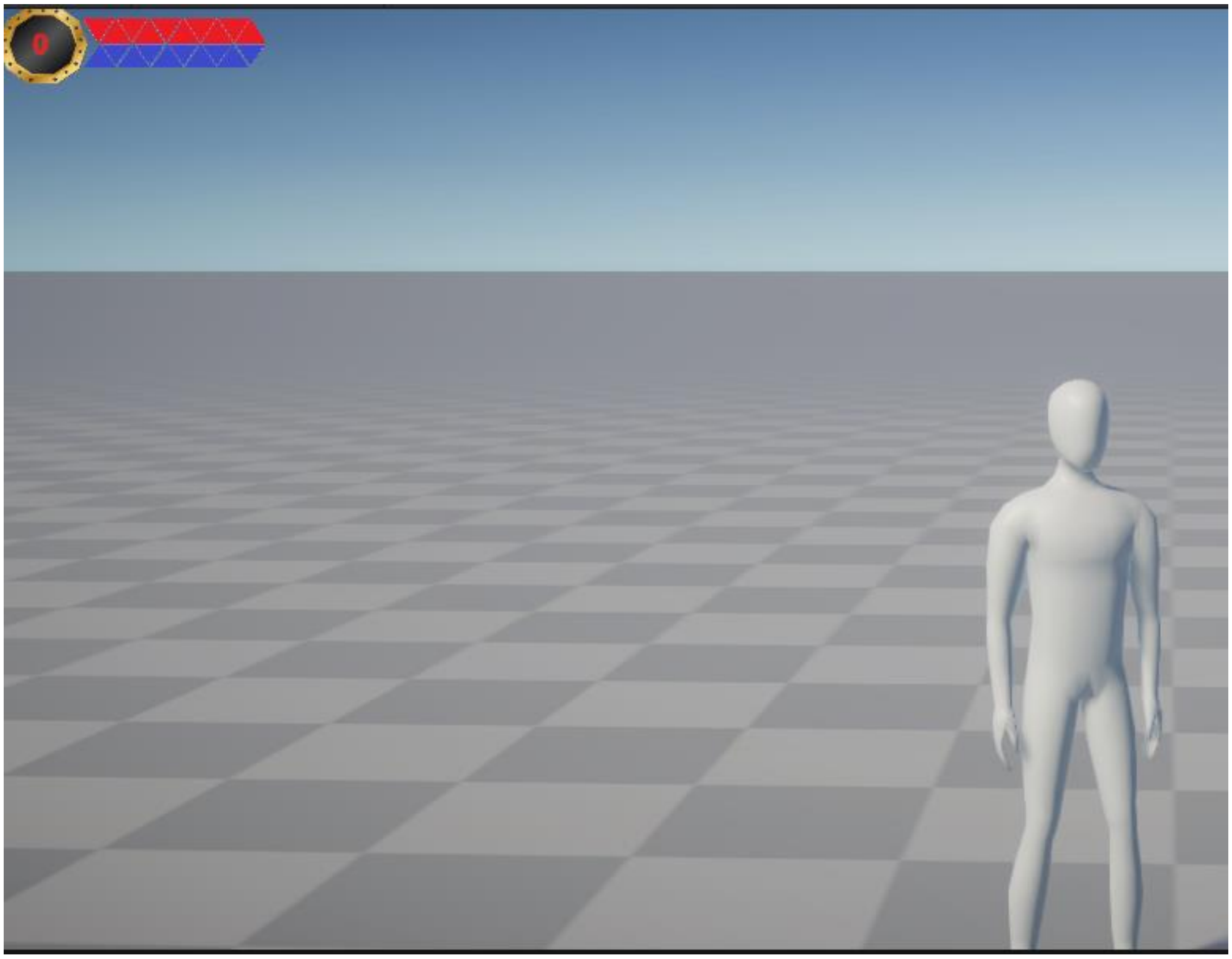
Fps Mode



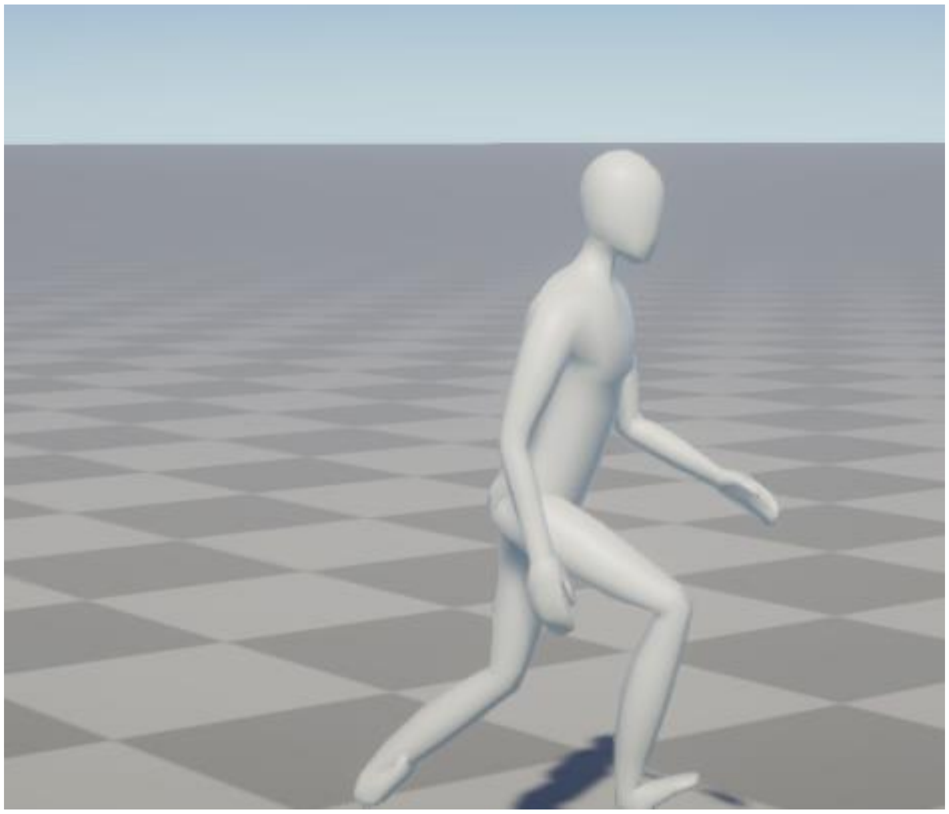


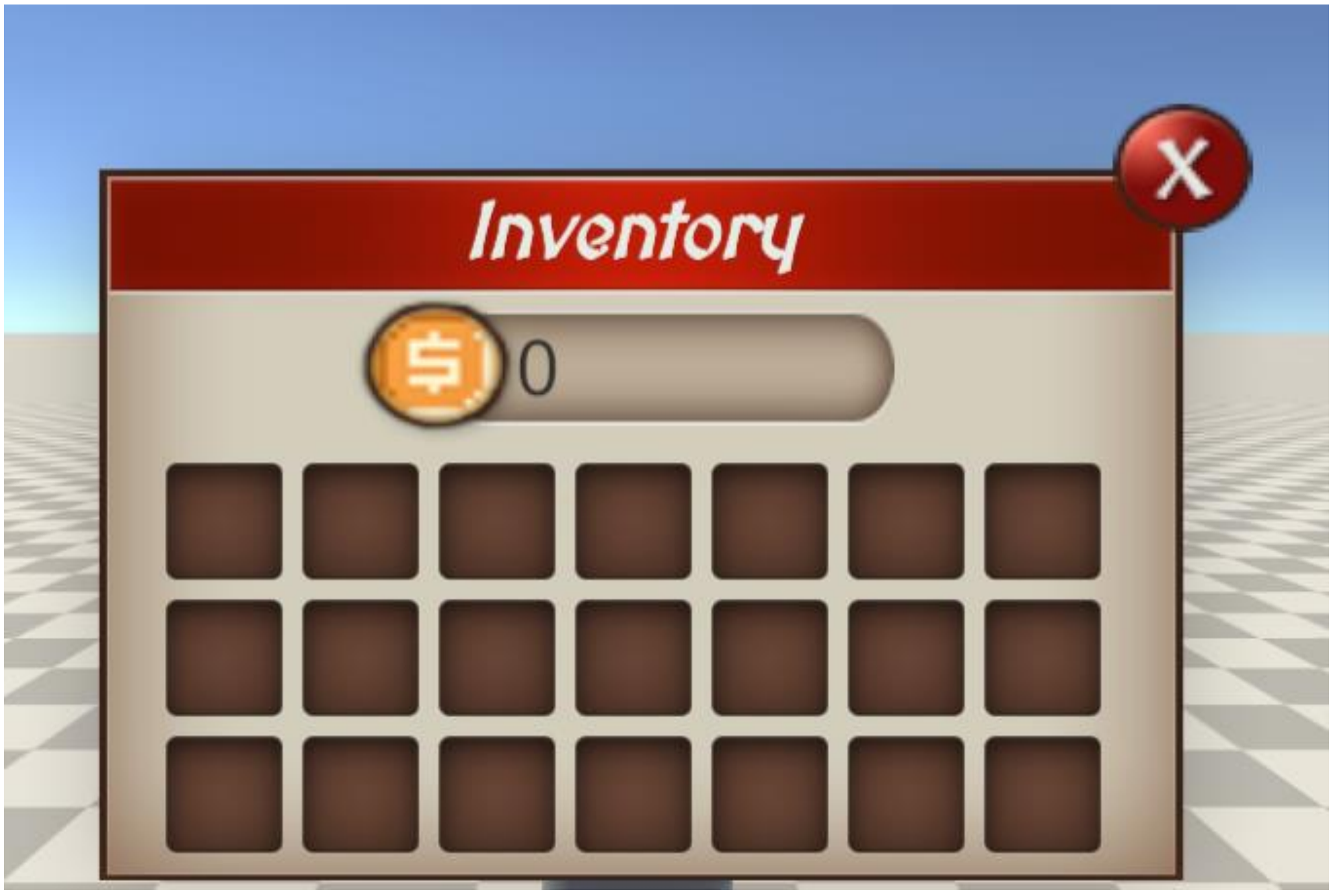
Third Person Mode

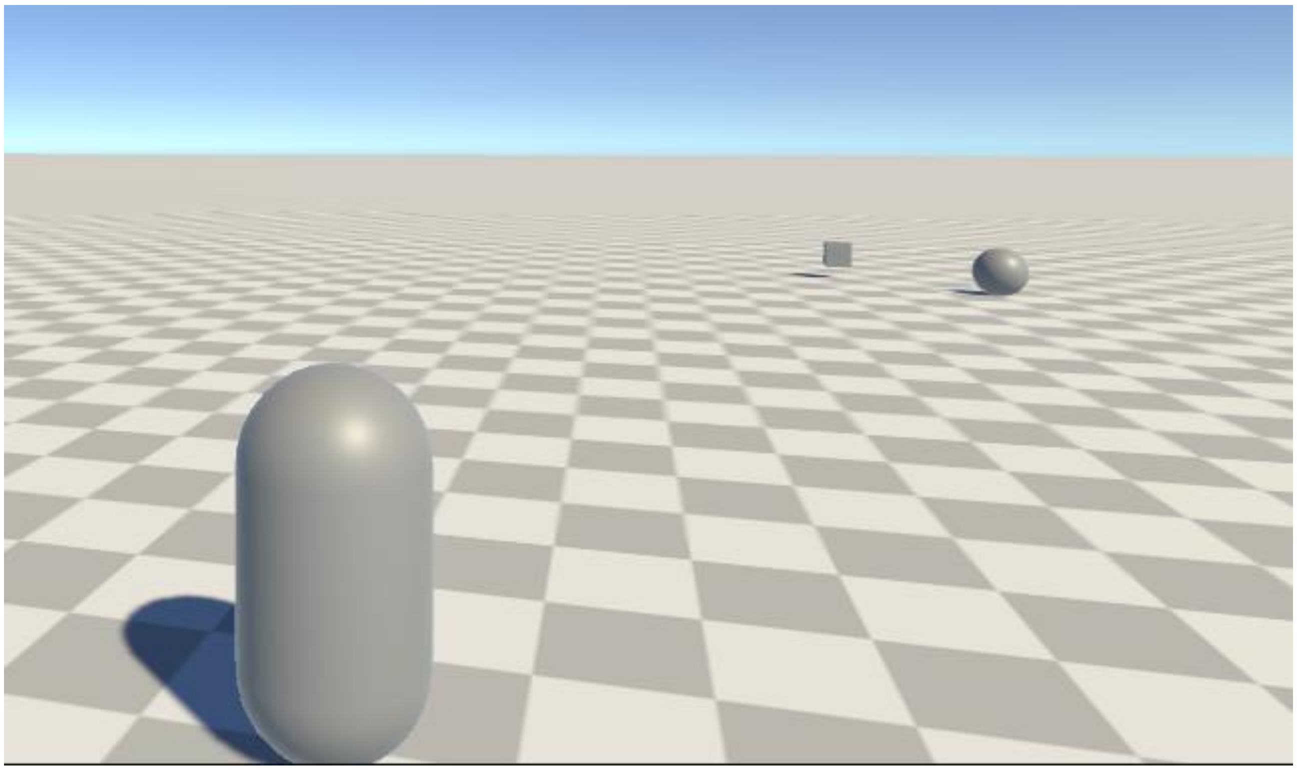
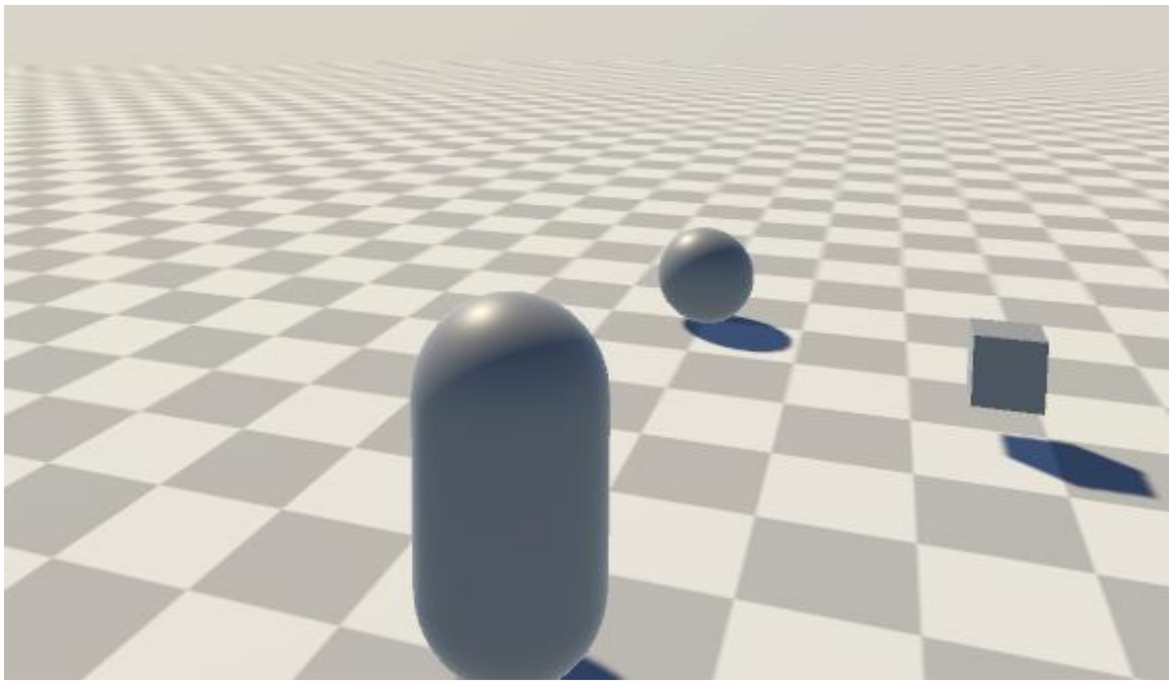



0

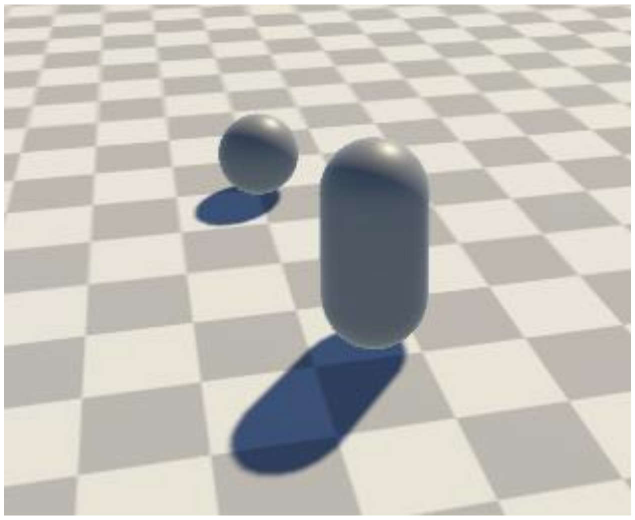
I am terrible at animation but hopefully I am good at programing.



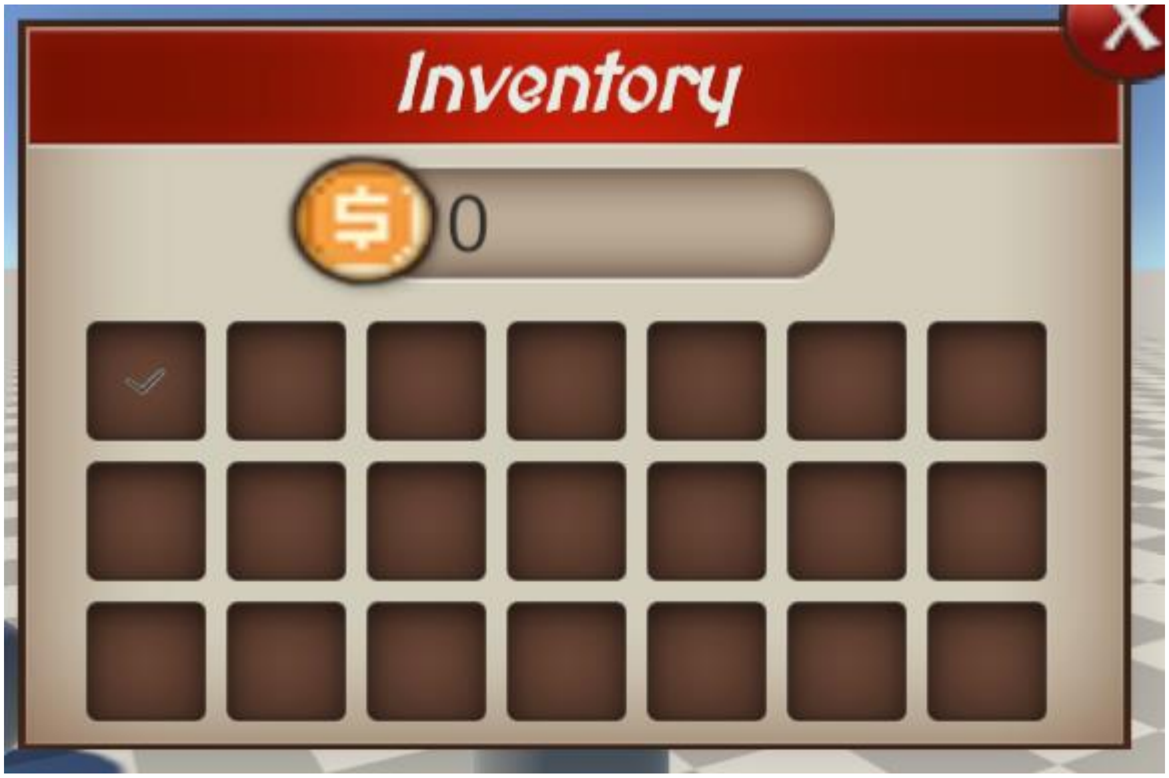


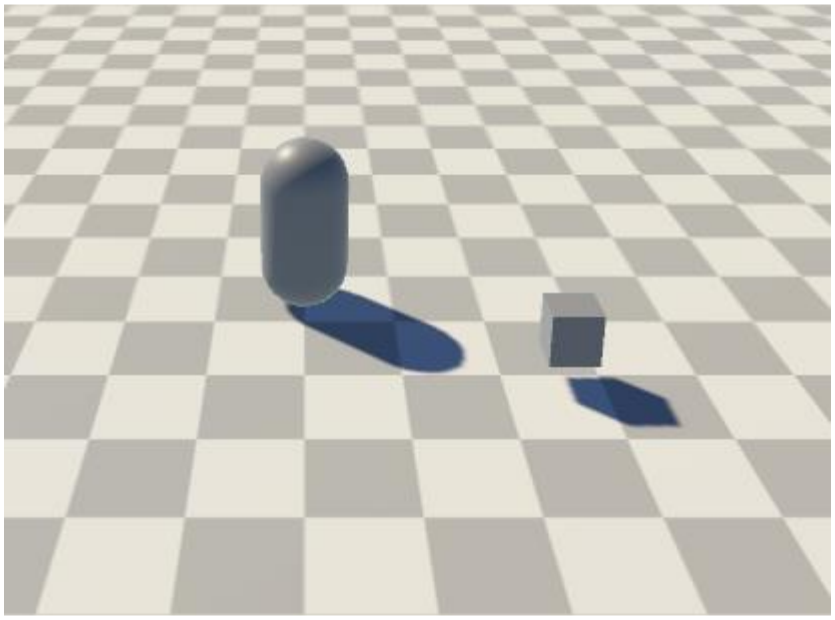
Cube is an item to pick up in game (like a sword), in this test scenario

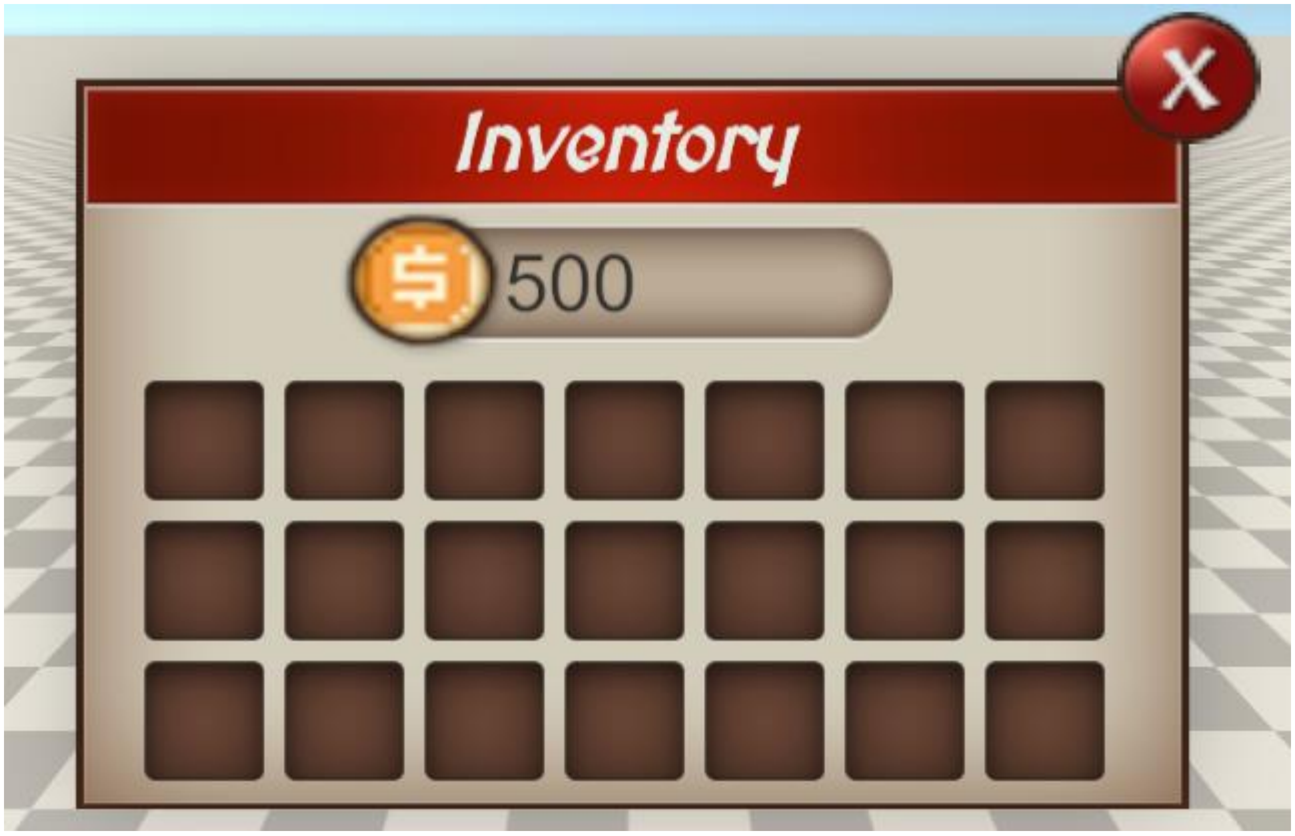


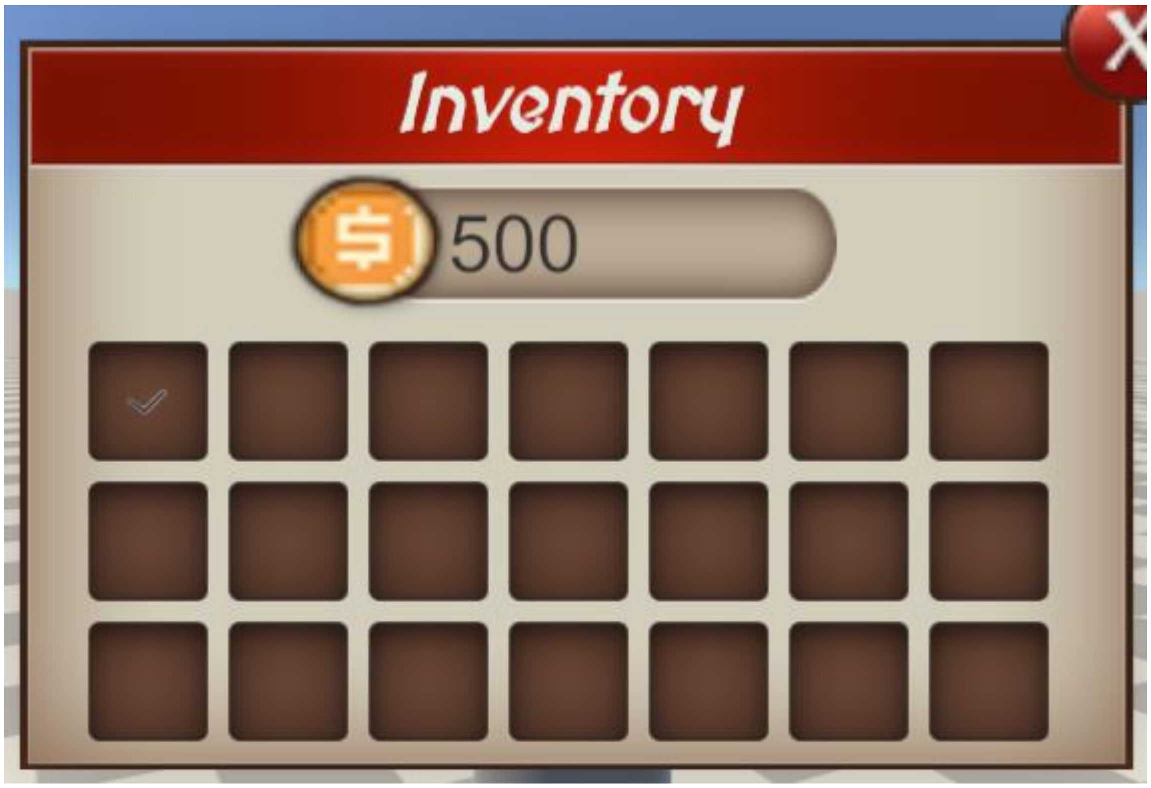
Item pickup

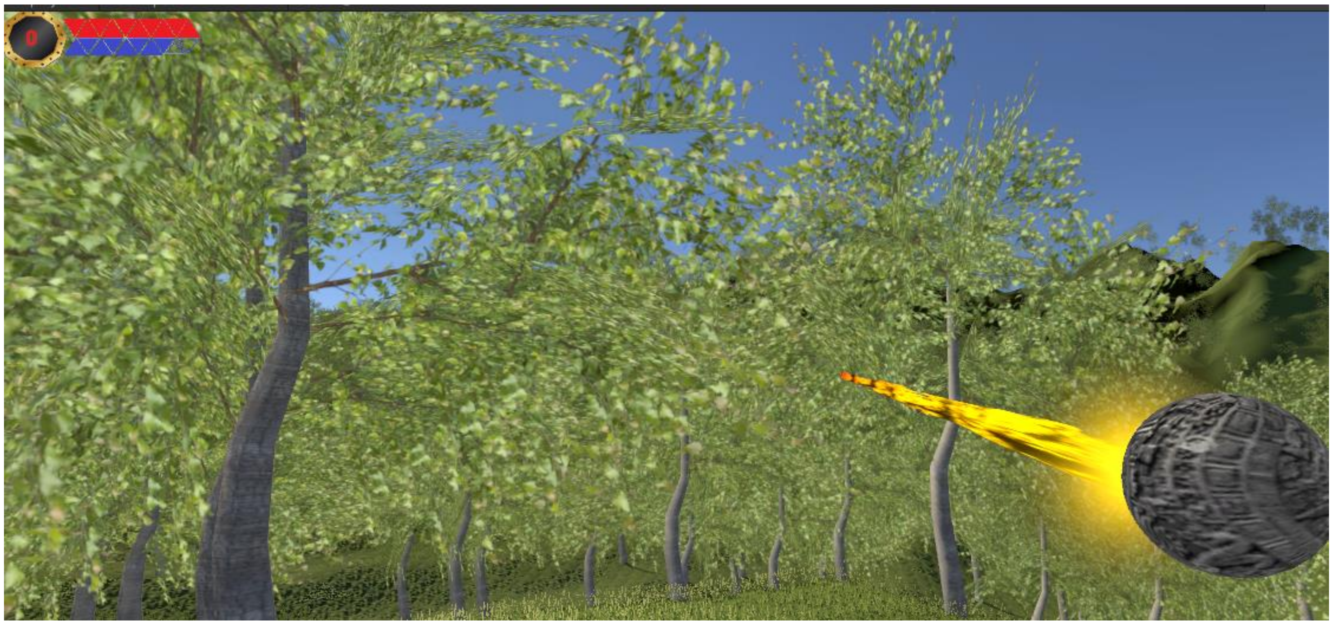


Ball is gold to Pick up in this test scenario



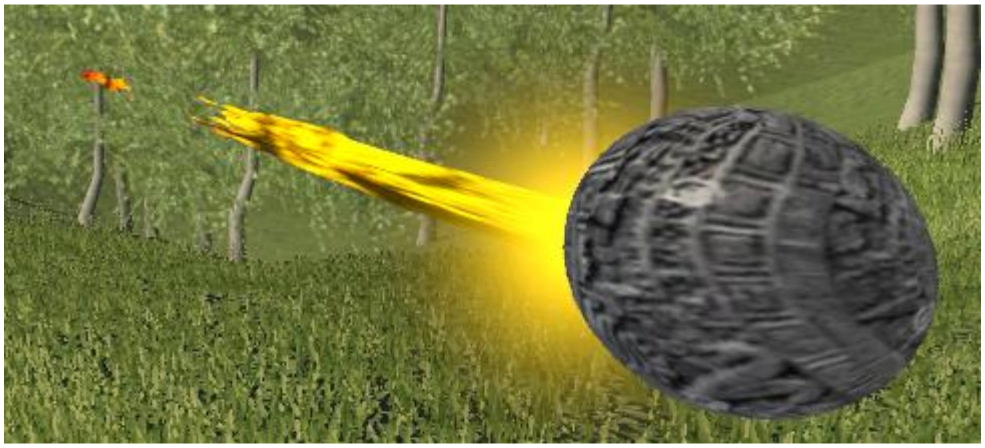



Inventory
X
500

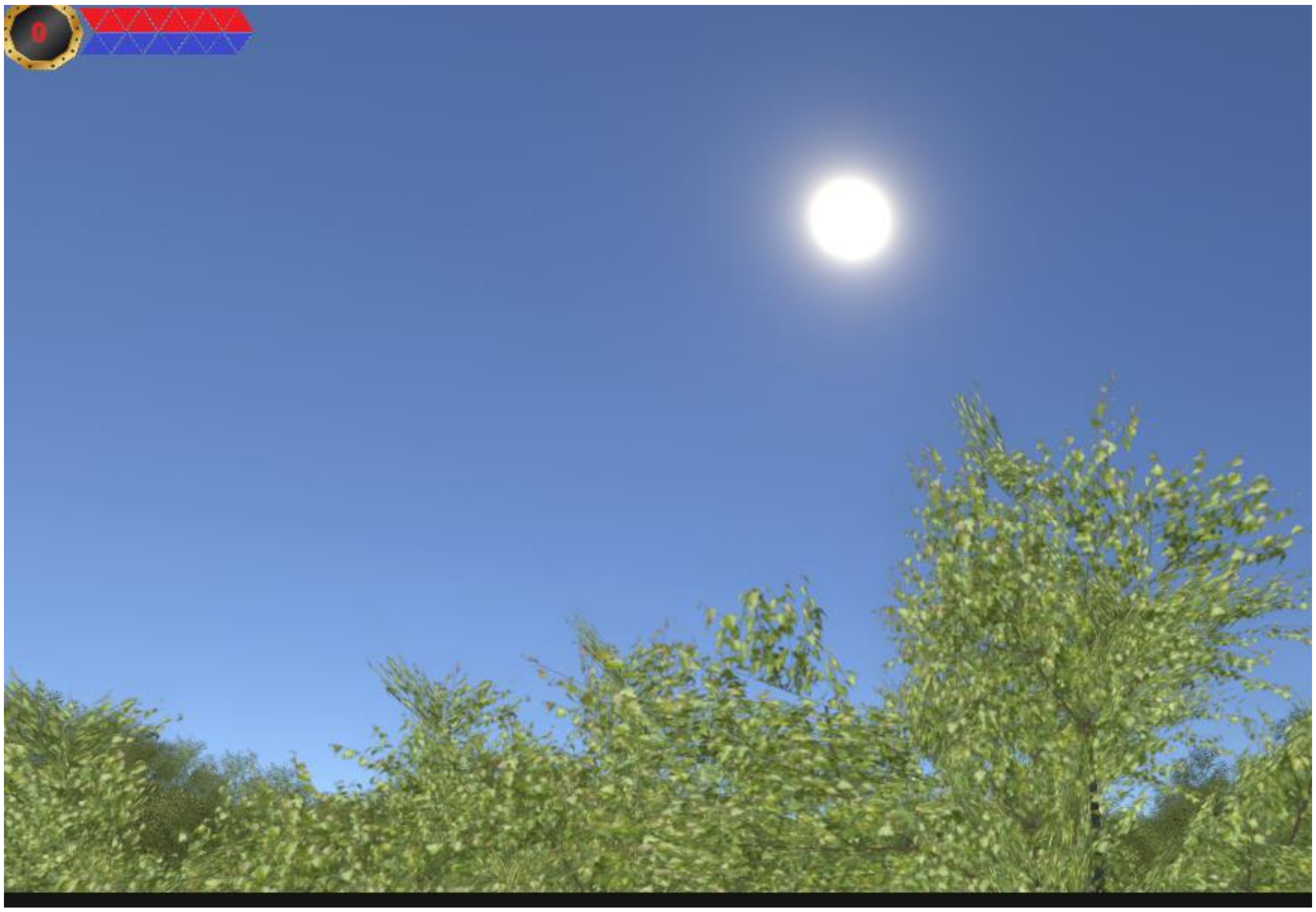

0

0

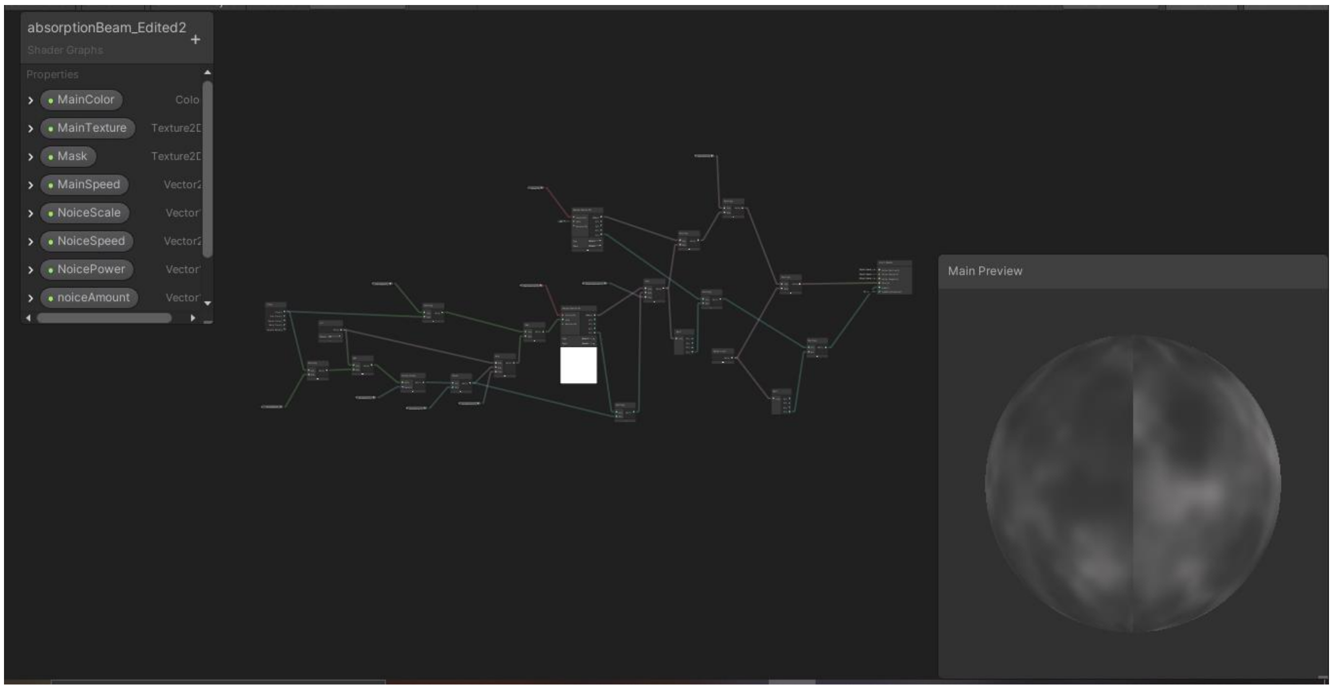
0

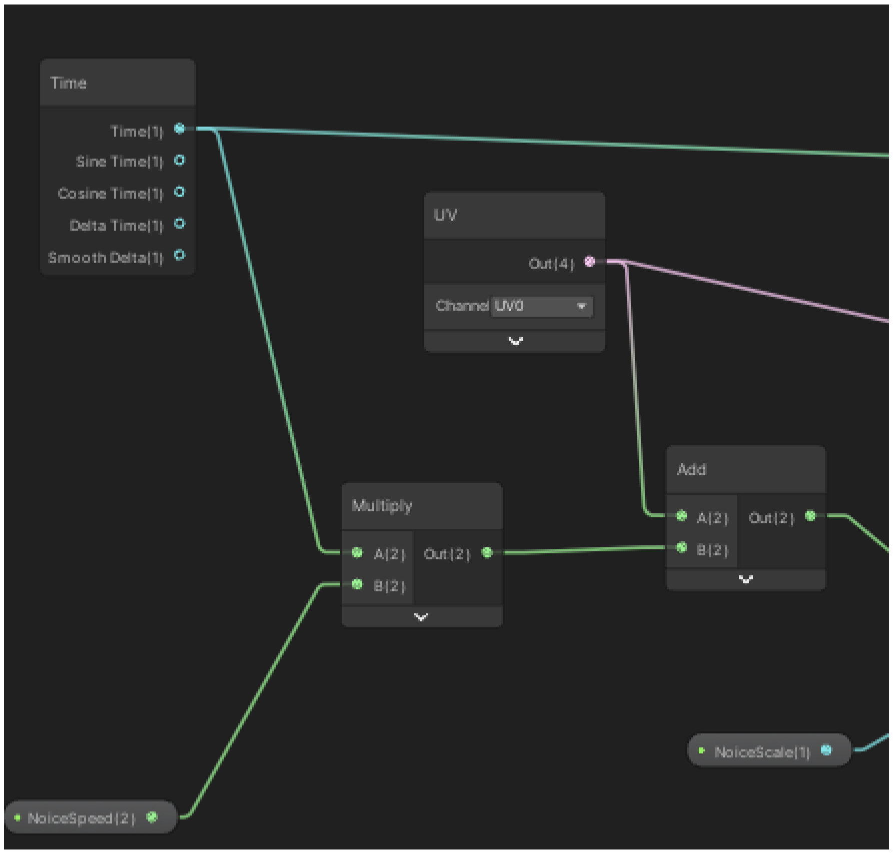


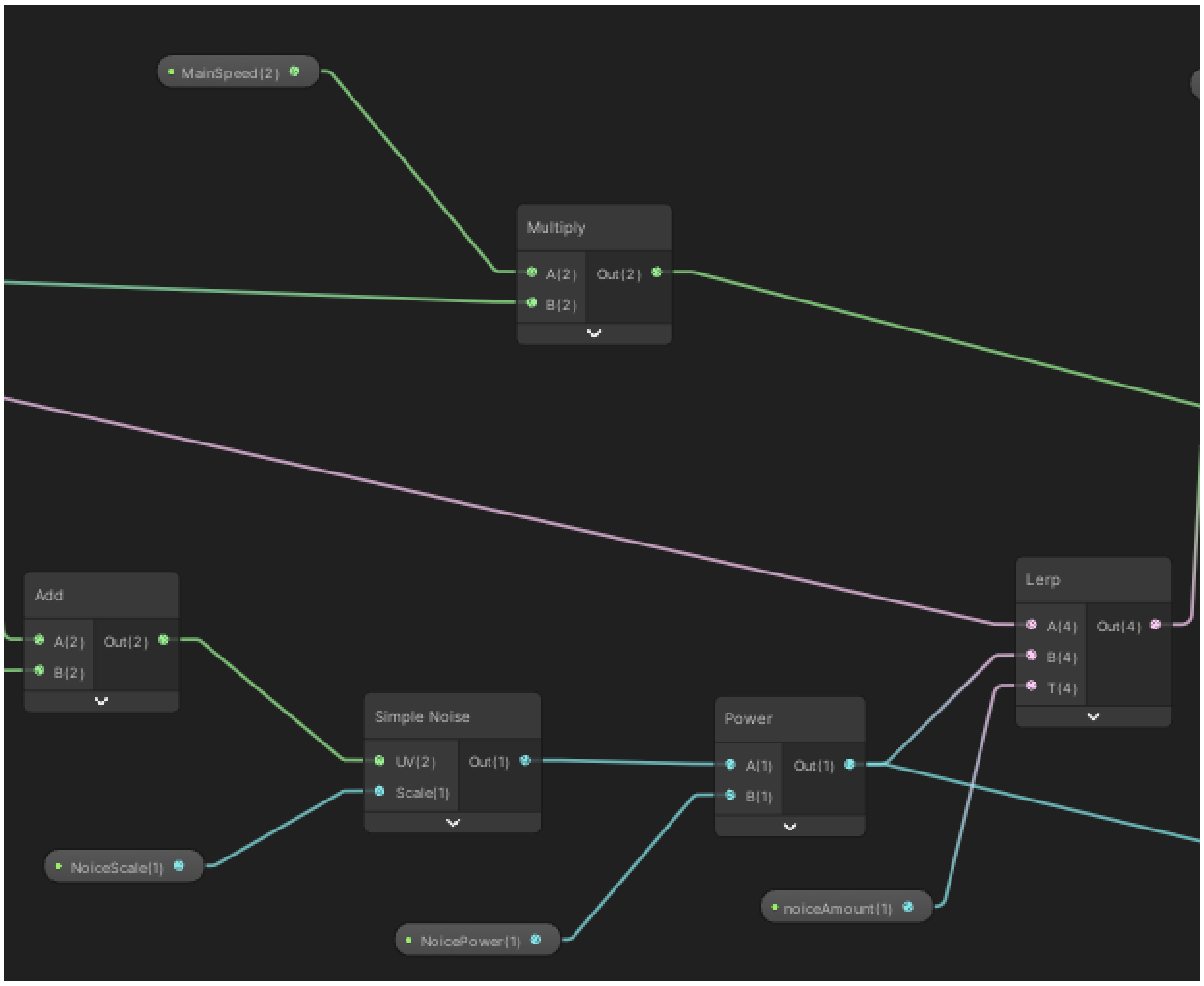
Ugly Trees I Know Right? Burn it!

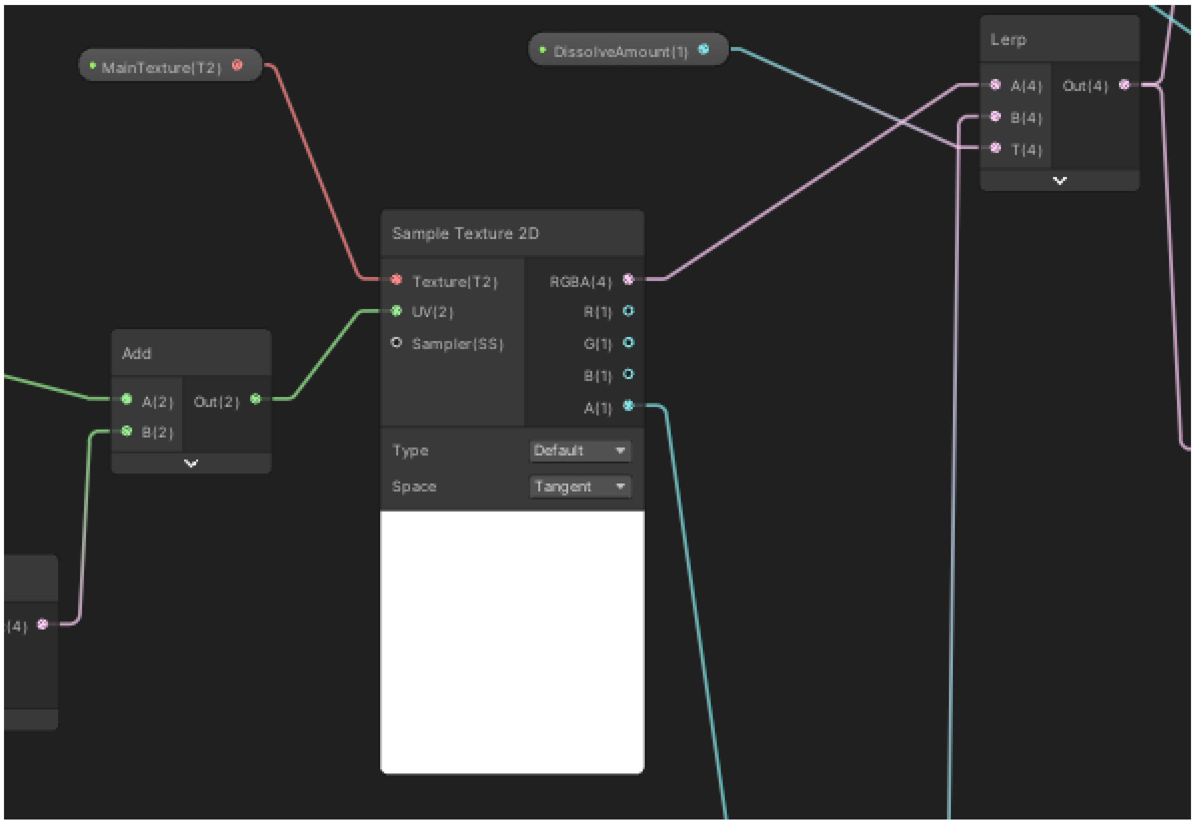

0

Shader Graph

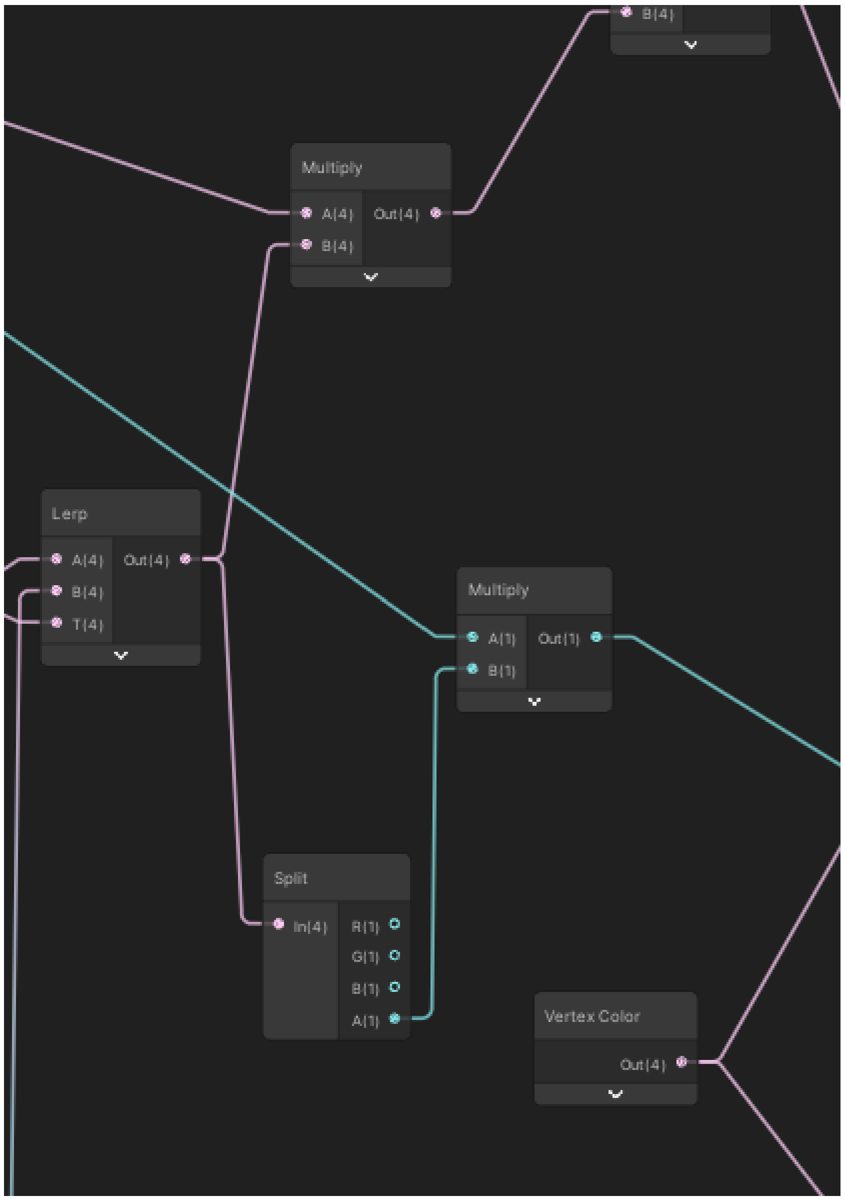


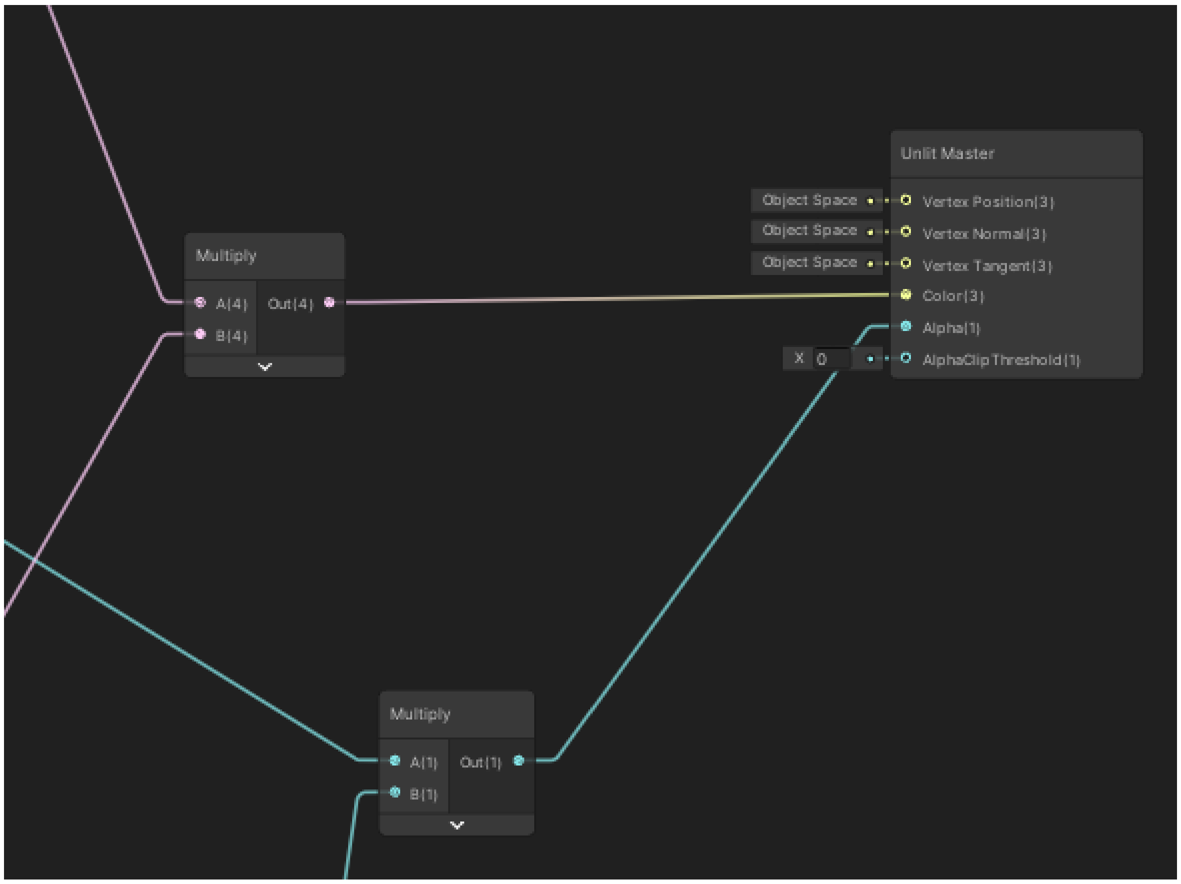


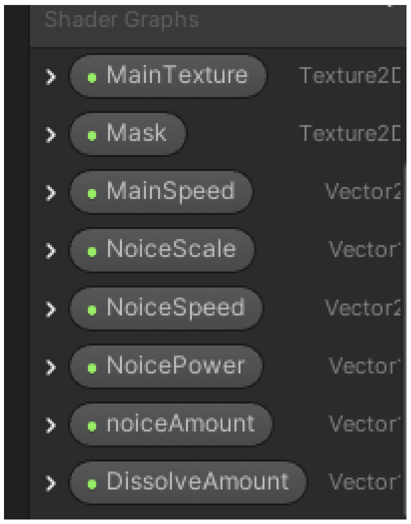

MainSpeed(2)
Multiply
A(2)
B(2)
Out(2)
Add
A(2)
B(2)
Out(2)
Lerp
A(4)
B(4)
T(4)
Out(4)
Simple Noise
UV(2)
Scale(1)
Out(1)
Power
A(1)
B(1)
Out(1)
NoiceScale(1)
NoicePower(1)
noiceAmount(1)



Type Default
Space Tangent

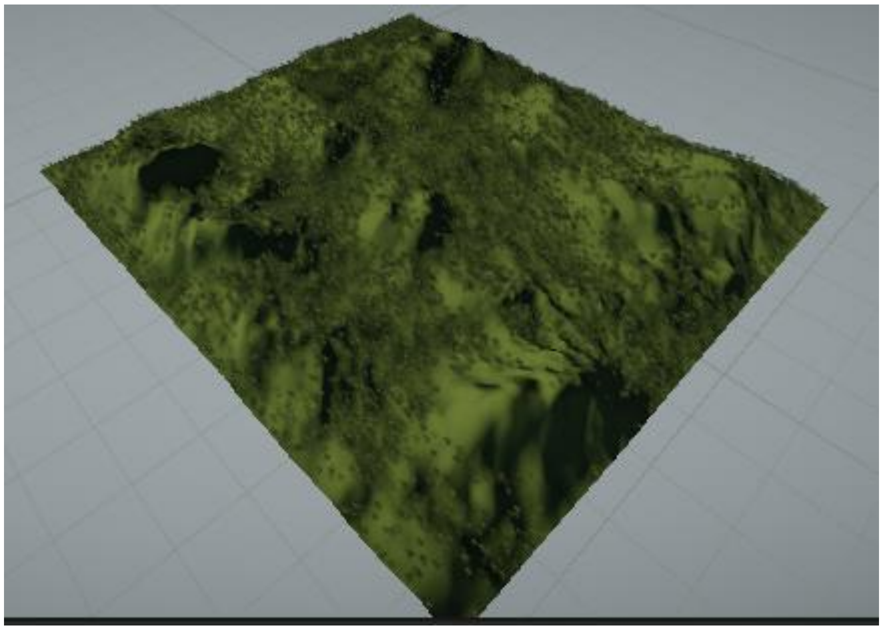
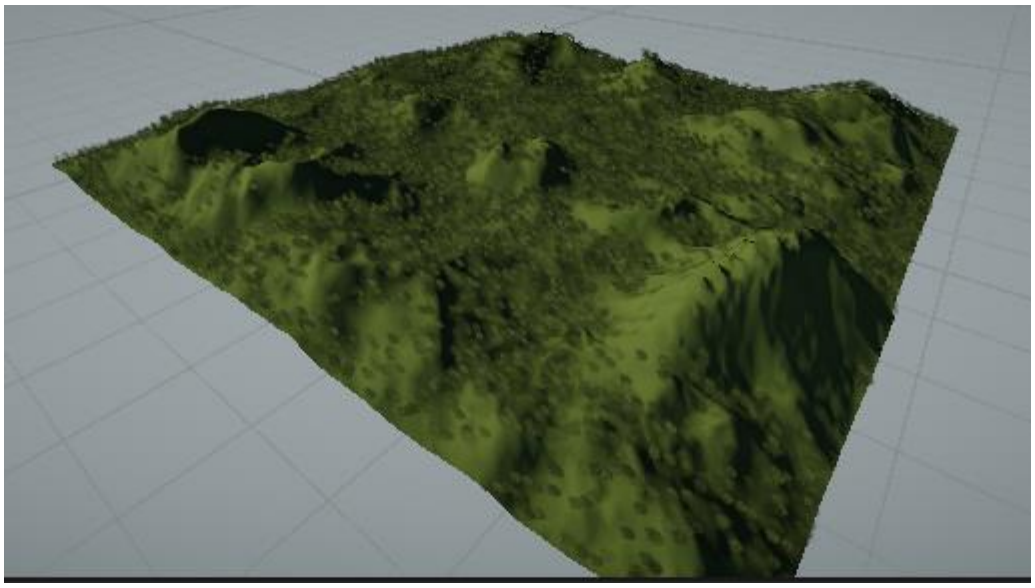

B(4)
Multiply
A(4)
B(4)
Out(4)
Lerp
A(4)
B(4)
T(4)
Out(4)
Multiply
A(1)
B(1)
Out(1)
Split
In(4)
R(1)
G(1)
B(1)
A(1)
Vertex Color
Out(4)

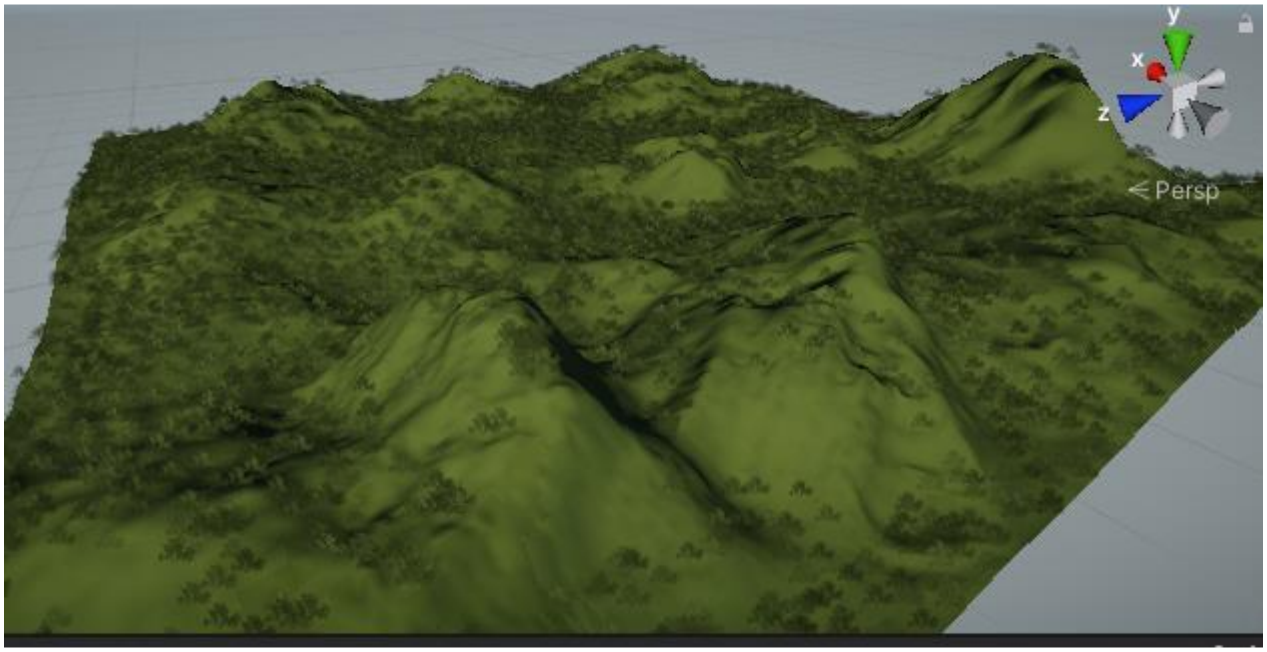
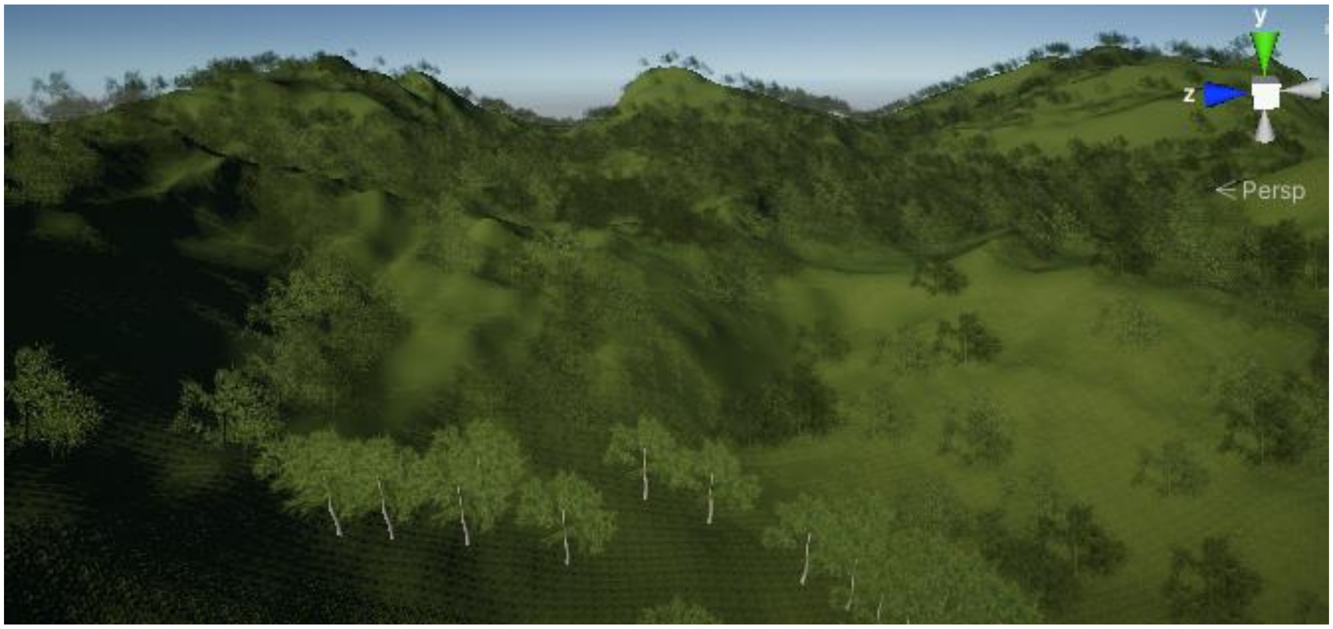



Shader Graphs

MainTexture	Texture2D
Mask	Texture2D
MainSpeed	Vector2
NoiceScale	Vector1
NoiceSpeed	Vector2
NoicePower	Vector1
noiceAmount	Vector1
DissolveAmount	Vector1

Level Design


y
x
z
Persp

y
z
Persp

Level number 2