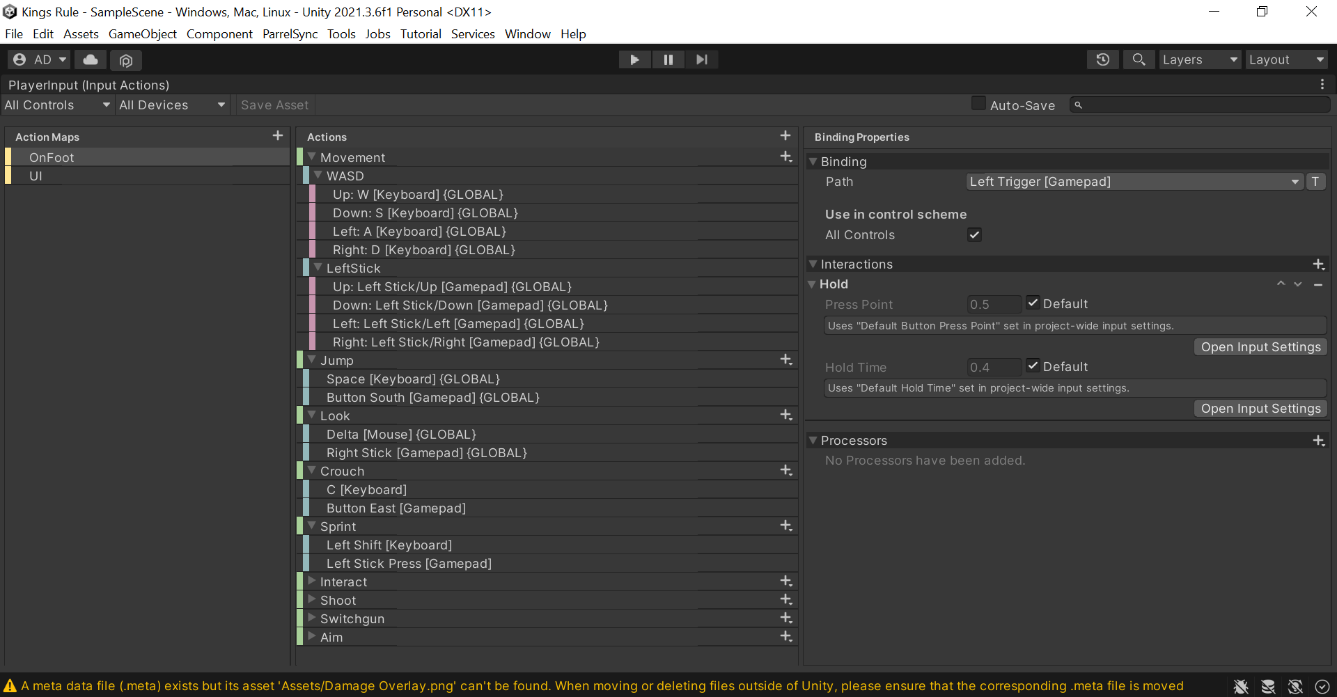
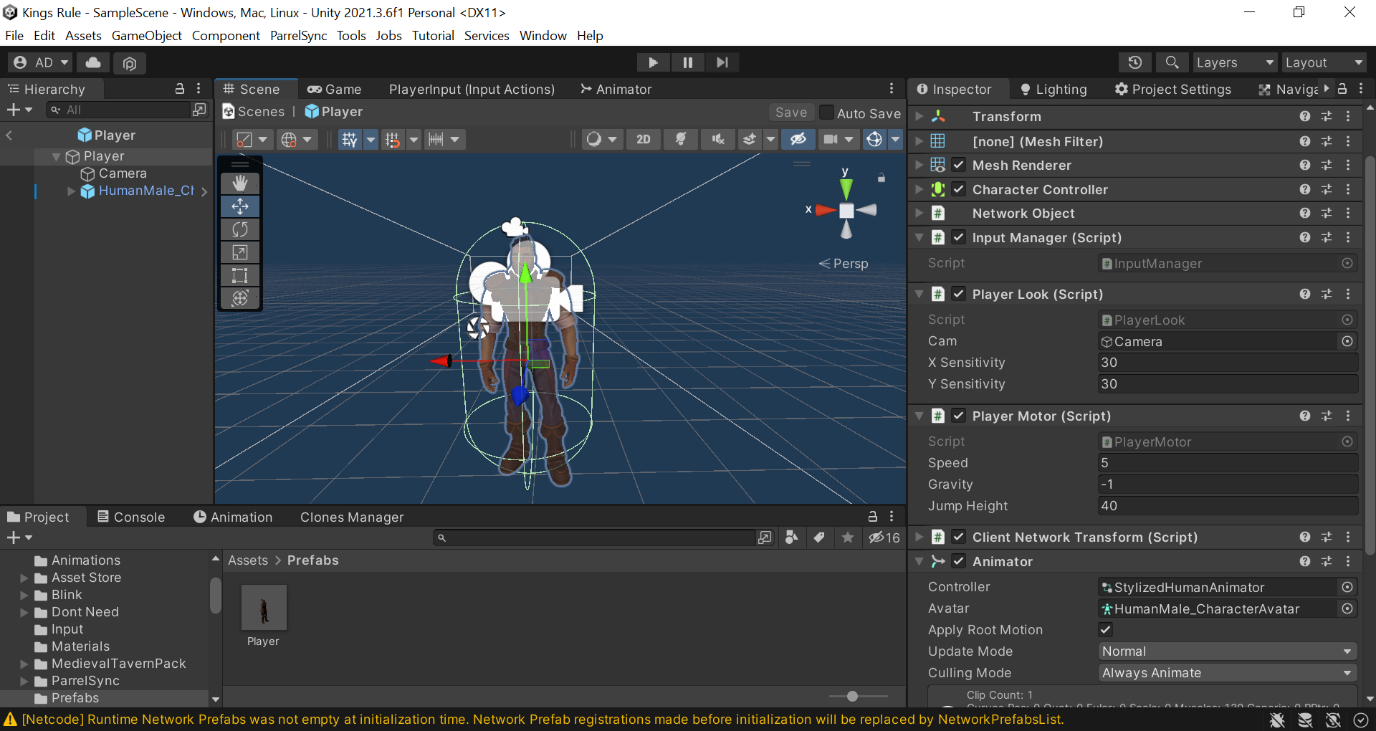
Drinkers Axe Scripts and descriptions

OnFoot Map – Use exact spelling as it plugs into the code in the following scripts

Player Model – This may be a little tricky, but you can see all the values to right and all the components I have plugged into it. The character controller is used for the Input Map. The Input manager, PlayerLook, And Player Motor are all pasted below, youll need to make slight edits, but look at my comments and figure out what is happening in those scripts, I have the values on screen. You can see I made the parent “Player” hold all the components, But I have the camera and character model underneath. I just used the male model we bought in the asset store if you wanna reuse it or find a new one and maybe make a character selector! You don’t need any of the network stuff, that’s just for multiplayer.

PlayerMotor – used to control the player through keyboard and controller inputs, Requires new input system to use

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using Unity.Netcode; // Wont need this, used for multiplayer

public class PlayerMotor : NetworkBehaviour // Change this back to MonoBehavior

{

private CharacterController controller;

private Vector3 playerVelocity;

private bool isGrounded;

public float speed = 5f;

public float gravity = -9.8f;

public float jumpHeight = 3f;

bool crouching = false;

float crouchTimer = 1;

bool lerpCrouch = false;

bool sprinting = false;

// Start is called before the first frame update

// public override void OnNetworkSpawn()

// {

// if (!IsOwner) Destroy(this);

// }

void Start()

{

controller = GetComponent<CharacterController>();

}

// Update is called once per frame

void Update()

{

if (!IsOwner) return; // May not need this line, used for multiplayer authority control

isGrounded = controller.isGrounded;

if (lerpCrouch)

{

crouchTimer += Time.deltaTime;

float p = crouchTimer / 1;

p \*= p;

if (crouching)

controller.height = Mathf.Lerp(controller.height, 1, p);

else

controller.height = Mathf.Lerp(controller.height, 2, p);

if (p > 1)

{

lerpCrouch = false;

crouchTimer = 0f;

}

}

}

//Recieves the inputs for out InputManager.cs and apply them to our Character controller

public void ProcessMove(Vector2 input)

{

Vector3 moveDirection = Vector3.zero;

moveDirection.x = input.x;

moveDirection.z = input.y;

controller.Move(transform.TransformDirection(moveDirection) \* speed \* Time.deltaTime);

playerVelocity.y += gravity + Time.deltaTime;

if (isGrounded && playerVelocity.y < 0)

playerVelocity.y = -2f;

controller.Move(playerVelocity \* Time.deltaTime);

}

public void Jump()

{

if (isGrounded)

{

playerVelocity.y = Mathf.Sqrt(jumpHeight \* -3.0f \* gravity);

}

}

public void Crouch()

{

crouching = !crouching;

crouchTimer = 0;

lerpCrouch = true;

}

public void Sprint()

{

sprinting = !sprinting;

if (sprinting)

speed = 8;

else

speed = 5;

}

}

PlayerLook – Used for mouse control and controller control

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using Unity.Netcode; // Don’t need this

public class PlayerLook : NetworkBehaviour // Change this to MonoBehavior

{

public GameObject cam;

private float xRotation = 0f;

public float xSensitivity = 30f;

public float ySensitivity = 30f;

// Start is called before the first frame update

void Update() // Don’t need this method, Can be deleted, or kept in, up to you

{

if (!IsOwner)

{

cam.SetActive(false);

}

return;

}

public void ProcessLook(Vector2 input)

{

float mouseX = input.x;

float mouseY = input.y;

//calculating camera rotation for looking up and down

xRotation -= (mouseY \* Time.deltaTime) \* ySensitivity;

xRotation = Mathf.Clamp(xRotation, -80f, 80f);

//apply this to our camera transform

cam.transform.localRotation = Quaternion.Euler(xRotation, 0, 0);

//rotate player to look left and right

transform.Rotate(Vector3.up \* (mouseX \* Time.deltaTime) \* xSensitivity);

}

}

Input Manager – This plugs the PlayerMotor and PlayerLook into the player, These 3 Components work together for basic player mobility

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.InputSystem;

using Unity.Netcode; // Don’t need this

public class InputManager : NetworkBehaviour // change to MonoBehavior

{

private PlayerInput playerInput;

public PlayerInput.OnFootActions onFoot;

private PlayerMotor motor;

private PlayerLook look;

// Start is called before the first frame update

void Awake()

{

playerInput = new PlayerInput();

onFoot = playerInput.OnFoot;

motor = GetComponent<PlayerMotor>();

look = GetComponent<PlayerLook>();

onFoot.Jump.performed += ctx => motor.Jump();

onFoot.Crouch.performed += ctx => motor.Crouch();

onFoot.Sprint.performed += ctx => motor.Sprint();

}

void Update()

{

if (!IsOwner) return; // Don’t need this, Mulitplayer function

}

// Update is called once per frame

void FixedUpdate()

{

//tell player motor to move using the value from our movement action

motor.ProcessMove(onFoot.Movement.ReadValue<Vector2>());

}

private void LateUpdate()

{

look.ProcessLook(onFoot.Look.ReadValue<Vector2>());

}

private void OnEnable()

{

onFoot.Enable();

}

private void OnDisable()

{

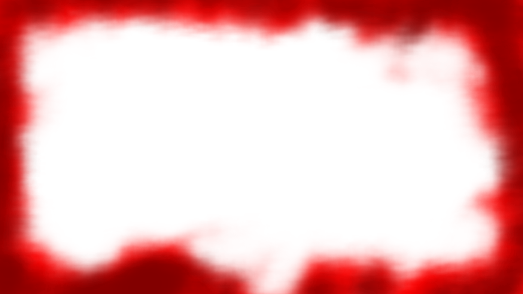
onFoot.Disable();

}

}

PlayerHealth – Not used in multiplayer, but fully functional, Just requires some bars and a little messing around with the ui to make it work

Damage- Overlays, Make sure they are Sprites, otherwise they are just images and the middle will be white instead of clear, might not work anyways but worth a shot.



using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using Unity.Netcode; //Not Needed

public class PlayerHealth : NetworkBehaviour // change to MonoBehavior

{

[Header("Health Bar")]

private float health;

private float lerpTimer;

[Header("Health Bar")]

public float maxHealth = 100f;

public float chipSpeed = 2;

public Image frontHealthBar;

public Image backHealthBar;

[Header("Damage Overlay")]

public Image overlay; //our DamageOverlay GameObject

public float duration; // how long the image stays fully opaque

public float fadeSpeed; // how quickly the image will fade

private float durationTimer; //timer to check against the duration

// Start is called before the first frame update

void Start()

{

health = maxHealth;

overlay.color = new Color(overlay.color.r, overlay.color.g, overlay.color.b, 0);

}

// Update is called once per frame

void Update()

{

if (!IsOwner) return;

health = Mathf.Clamp(health, 0, maxHealth);

UpdateHealthUI();

if(overlay.color.a > 0)

{

durationTimer += Time.deltaTime;

if(durationTimer > duration)

{

//fade the image

float tempAlpha = overlay.color.a;

tempAlpha -= Time.deltaTime \* fadeSpeed;

overlay.color = new Color(overlay.color.r, overlay.color.g, overlay.color.b, tempAlpha);

}

}

}

public void UpdateHealthUI()

{

Debug.Log(health);

float fillF = frontHealthBar.fillAmount;

float fillB = backHealthBar.fillAmount;

float hFraction = health / maxHealth;

if(fillB > hFraction)

{

frontHealthBar.fillAmount = hFraction;

backHealthBar.color = Color.red;

lerpTimer += Time.deltaTime;

float percentageComplete = lerpTimer / chipSpeed;

percentageComplete = percentageComplete \* percentageComplete;

backHealthBar.fillAmount = Mathf.Lerp(fillB, hFraction, percentageComplete);

}

if(fillF < hFraction)

{

backHealthBar.color = Color.green;

backHealthBar.fillAmount = hFraction;

lerpTimer += Time.deltaTime;

float percentComplete = lerpTimer / chipSpeed;

percentComplete = percentComplete \* percentComplete;

frontHealthBar.fillAmount = Mathf.Lerp(fillF, backHealthBar.fillAmount, percentComplete);

}

}

public void TakeDamage(float damage)

{

health -= damage;

lerpTimer = 0f;

durationTimer = 0;

overlay.color = new Color(overlay.color.r, overlay.color.g, overlay.color.b, 1);

}

public void RestorHealth(float healAmount)

{

health += healAmount;

lerpTimer = 0f;

}

}

Scoreboard – This is functional in multiplayer, Will need to make a few changes to work for single player against bots if desired, It is coded for 2 players to be plugged into it over a network but that’s obviously not needed for single player

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class Scoreboard : MonoBehaviour

{

//Timer

float currentTime = 0f;

float startingTime = 120f;

[SerializeField] Text countdownText;

//Score

[SerializeField] Text Player1Score;

[SerializeField] Text Player2Score;

float Player1currentScore = 0f;

float Player1startingScore = 0f;

float Player2currentScore = 0f;

float Player2startingScore = 0f;

void Start()

{

currentTime = startingTime;

Player1currentScore = Player1startingScore;

Player2currentScore = Player2startingScore;

}

void Update()

{

currentTime -= 1 \* Time.deltaTime;

countdownText.text = currentTime.ToString ("0");

Player1currentScore += 1 \*Time.deltaTime;

Player2currentScore += 1 \* Time.deltaTime;

Player1Score.text = Player1currentScore.ToString("0");

Player2Score.text = Player2currentScore.ToString("0");

countdownText.color = Color.green;

if(currentTime <= 10)

{

countdownText.color = Color.red;

}

if(currentTime <= 0)

{

currentTime = 0;

}

}

void ScorePoint()

{

}

}

GameManager – This is used for multiplayer but should be able to be used for single player with some changes

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class GameManager : MonoBehaviour

{

public static GameManager Instance;

public GameState State;

public static event Action<GameState> OnGameStateChanged;

// Start is called before the first frame update

void Awake()

{

Instance = this;

}

void Start()

{

UpdateGameState(GameState.SelectSide);

}

public void UpdateGameState(GameState newState)

{

State = newState;

switch (newState)

{

case GameState.SelectSide:

HandleSelectSide();

break;

case GameState.StartGame:

break;

case GameState.Player1Turn:

break;

case GameState.Player2Turn:

break;

case GameState.Victory:

break;

case GameState.Lose:

break;

default:

throw new ArgumentOutOfRangeException(nameof(newState), newState, null);

}

OnGameStateChanged?.Invoke(newState);

}

private void HandleSelectSide()

{

}

}

public enum GameState

{

SelectSide,

StartGame,

Player1Turn,

Player2Turn,

Victory,

Lose

}