**This Document is just a combination of all the scripts in one place:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Powerups : MonoBehaviour

{

[SerializeField] private int powerupID = 0;

// Applies a random power up to the player then destroys the object

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.tag == "Player")

{

Player P = collision.GetComponent<Player>();

powerupID = Random.Range(0, 3);

if (P != null)

{

if (powerupID == 0)

{

P.EatPowerUp();

}

else if (powerupID == 1)

{

P.SpeedBoostPowerUp();

}

else if (powerupID == 2)

{

P.DoublePointsPowerUp();

}

}

Destroy(this.gameObject);

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Points : MonoBehaviour

{

[SerializeField] GameObject GM;

void Start()

{

GM = GameObject.FindGameObjectWithTag("GM");

}

// When colliding with a point this adds the points to the Game Manager, updates points remaining, and destroys the object

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.tag == "Point")

{

GM.GetComponent<Score>().score += 100;

GM.GetComponent<WonLevel>().pointsRemaining--;

Destroy(collision.gameObject);

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class PlayerName : MonoBehaviour {

[SerializeField] private TMP\_Text inputText;

// Saves current name to the computer

public void SetName()

{

PlayerPrefs.SetString("CurrentName", inputText.text);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class Player : MonoBehaviour

{

//Variable for updating the slider UI

public Slider eatSlider;

[SerializeField] private float maxEatTime = 10f;

[SerializeField] private float currentEatTime = 0.0f;

[SerializeField] private bool canEat = false;

[SerializeField] private GameObject portals;

[SerializeField] private GameObject teleporterOne;

[SerializeField] private GameObject teleporterTwo;

[SerializeField] private GameObject last;

GameObject GM;

[SerializeField] private AudioSource source;

[SerializeField] private AudioClip deathSound;

private bool canTeleport = true;

public bool eatPower = false;

[SerializeField] private bool isDouble = false;

public Vector3 startPos;

// Sets up the player with initial values

void Start()

{

startPos = transform.position;

GM = GameObject.FindGameObjectWithTag("GM");

teleporterOne = portals.transform.GetChild(0).gameObject;

teleporterTwo = portals.transform.GetChild(1).gameObject;

Reset();

}

// Updates the eat functionality and slider

void Update()

{

//Checks if the player can eat and if not begins to fill the bar

if (currentEatTime >= maxEatTime)

{

canEat = true;

}

else

{

canEat = false;

currentEatTime += Time.deltaTime;

currentEatTime = Mathf.Clamp(currentEatTime, 0.0f, maxEatTime);

}

if (eatPower)

eatSlider.value = 1;

else

eatSlider.value = currentEatTime / maxEatTime;

}

// Checks collisions and conducts various functions according to the collision detected

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.tag == "Enemy" || collision.tag == "Clone")

{

if (canEat || eatPower)

{

collision.GetComponent<EnemyNavigation>().Die();

currentEatTime = 0f;

}

else

{

GM.GetComponent<Reset>().ResetEverything();

GM.GetComponent<GameOver>().decrementLives();

if (PlayerPrefs.GetInt("SoundEffects", 1) == 1)

source.PlayOneShot(deathSound);

}

}

else if (collision.tag == "GhostEnemy")

{

GM.GetComponent<Reset>().ResetEverything();

GM.GetComponent<GameOver>().decrementLives();

if(PlayerPrefs.GetInt("SoundEffects", 1) == 1)

source.PlayOneShot(deathSound);

}

else if (collision.tag == "Point" && isDouble)

{

GM.GetComponent<Score>().score += 100;

}

else if (collision.tag == "Portal")

{

if (canTeleport)

{

canTeleport = false;

if (collision.gameObject == teleporterOne)

{

transform.position = teleporterTwo.transform.position;

last = teleporterTwo;

}

else

{

transform.position = teleporterOne.transform.position;

last = teleporterOne;

}

this.GetComponent<CharacterMovement>().target = transform.position;

}

}

}

// Prevents the player from sling-shotting between teleporters

private void OnTriggerExit2D(Collider2D collision)

{

if(collision.tag == "Portal")

{

if(last == collision.gameObject)

canTeleport = true;

}

}

//Resets the player object to starting position

public void Reset()

{

transform.position = startPos;

currentEatTime = 0;

this.GetComponent<CharacterMovement>().target = this.transform.position;

}

public IEnumerator EatPowerDown()

{

yield return new WaitForSeconds(5.0f);

canEat = false;

eatPower = false;

currentEatTime = 10f;

}

// All the following functions are used for power up functionality

public void EatPowerUp()

{

canEat = true;

eatPower = true;

StartCoroutine(EatPowerDown());

}

public void SpeedBoostPowerUp()

{

this.GetComponent<CharacterMovement>().speed \*= 1.5f;

StartCoroutine(SpeedBoostPowerDown());

}

public IEnumerator SpeedBoostPowerDown()

{

yield return new WaitForSeconds(5.0f);

this.GetComponent<CharacterMovement>().speed /= 1.5f;

}

public void DoublePointsPowerUp()

{

isDouble = true;

StartCoroutine(DoublePointsPowerDown());

}

public IEnumerator DoublePointsPowerDown()

{

yield return new WaitForSeconds(5.0f);

isDouble = false;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MultiplyingEnemy : MonoBehaviour

{

[SerializeField] GameObject clone;

[SerializeField] float timeToSpawn;

private float currentTime = 0;

GameObject GM;

private void Start()

{

GM = GameObject.FindGameObjectWithTag("GM");

}

// Handles spawn times of enemies, slowing them temporarily to make sure they don't overlap in the game

void Update()

{

currentTime += Time.deltaTime;

if (currentTime >= timeToSpawn)

{

Spawn();

currentTime = 0;

}

if (currentTime <= 1f)

{

GetComponent<EnemyNavigation>().speed = Random.Range(0.2f, 1.2f);

}

else

{

GetComponent<EnemyNavigation>().speed = 1.875f;

}

}

// Spawns the enemies

private void Spawn()

{

GameObject cloneObj = Instantiate(clone, new Vector3(this.transform.position.x, this.transform.position.y, this.transform.position.z), Quaternion.identity);

GM.GetComponent<Reset>().SetCloneEnemies(cloneObj);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

using TMPro;

public class MenuScript : MonoBehaviour

{

[SerializeField] private TMP\_Text DisplayText;

[SerializeField] private TMP\_Text SoundEffects;

[SerializeField] private TMP\_Text Music;

[SerializeField] private GameObject Menu;

[SerializeField] private GameObject Setting;

[SerializeField] private AudioSource Source;

[SerializeField] private AudioClip MenuMusic;

// This method ensures that all initial values are set

void Start()

{

PlayerPrefs.SetInt("Music", PlayerPrefs.GetInt("Music", 1));

PlayerPrefs.SetInt("SoundEffects", PlayerPrefs.GetInt("SoundEffects", 1));

DisplayText.text = "High Score: " + PlayerPrefs.GetString("HighScoreName", "EMPTY") + ": "

+ PlayerPrefs.GetInt("HighScoreNumber", 0).ToString();

Music.text = "Menu Music: " + (PlayerPrefs.GetInt("Music", 1) == 1 ? "ON" : "OFF");

SoundEffects.text = "Sound Effects: " + (PlayerPrefs.GetInt("SoundEffects", 1) == 1 ? "ON" : "OFF");

if (PlayerPrefs.GetInt("Music", 1) == 1)

{

Source.Play();

//Source.PlayOneShot(MenuMusic);

Source.loop = true;

}

if(GameObject.Find("GameManager") != null)

Destroy(GameObject.Find("GameManager"));

}

// Updates music preference on computer

public void OnMusicButtonPress()

{

PlayerPrefs.SetInt("Music", PlayerPrefs.GetInt("Music", 1) == 1 ? 0 : 1);

Music.text = "Music: " + (PlayerPrefs.GetInt("Music", 1) == 1 ? "ON" : "OFF");

if(PlayerPrefs.GetInt("Music", 1) != 1)

{

Source.Stop();

}

else

{

Source.Play();

Source.loop = true;

}

}

// Updates sound effect preference on computer

public void OnSoundEffectsButtonPress()

{

PlayerPrefs.SetInt("SoundEffects", PlayerPrefs.GetInt("SoundEffects", 1) == 1 ? 0 : 1);

SoundEffects.text = "Sound Effects: " + (PlayerPrefs.GetInt("SoundEffects", 1) == 1 ? "ON" : "OFF");

}

// Loads level\_01

public void NewGame()

{

SceneManager.LoadScene("Level\_01", LoadSceneMode.Single);

}

// Opens settings menu

public void Settings()

{

Menu.SetActive(false);

Setting.SetActive(true);

}

// Returns to main menu

public void Return()

{

Menu.SetActive(true);

Setting.SetActive(false);

}

// Quits the application

public void QuitGame()

{

Application.Quit();

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class LevelManager : MonoBehaviour

{

GameObject GM;

// Ensures that with every level, the game is properly set

private void Start()

{

GM = GameObject.Find("GameManager");

spawn();

GM.GetComponent<WonLevel>().setGame();

}

// Ensures enemies are spawned

void spawn()

{

GM.GetComponent<SpawnEnemies>().Spawn();

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class GhostEnemy : MonoBehaviour

{

[SerializeField] private Transform player;

[SerializeField] private float speed;

GameObject GM;

public Vector3 startPos;

// Sets initial values

void Start()

{

startPos = transform.position;

player = GameObject.FindGameObjectWithTag("Player").transform;

GM = GameObject.FindGameObjectWithTag("GM");

}

// Moves enemy towards target position

void Update()

{

if (!GM.GetComponent<PausedMenu>().isPaused)

transform.position = Vector2.MoveTowards(transform.position, player.position, speed \* Time.deltaTime);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EnemyNavigation : MonoBehaviour

{

[SerializeField] private Transform player;

public float speed;

// Variables for determining the state of the enemy

[SerializeField] private bool isRoaming = false;

[SerializeField] private bool isChasing = false;

[SerializeField] private bool isStarting = true;

[SerializeField] private bool isDieing = false;

[SerializeField] private bool isResetting = false;

// Variables for time

[SerializeField] private float timeToStart = 5f;

[SerializeField] private float timeToRoam = 20f;

[SerializeField] private float currentTime = 0.0f;

[SerializeField] private bool down = true;

[SerializeField] private bool isFast = false;

// Variables for determining target

private Transform target;

private Transform lastTarget;

private Transform lastPath;

private GameObject start;

public Vector3 startPos;

GameObject GM;

// Sets initial values

void Start()

{

startPos = transform.position;

target = getTarget();

lastTarget = null;

player = GameObject.FindGameObjectWithTag("Player").transform;

start = GameObject.FindGameObjectWithTag("StartOne");

GM = GameObject.FindGameObjectWithTag("GM");

}

void Update()

{

currentTime += Time.deltaTime;

// If resetting, count down until start, if not:

// When you reach your target:

// Save your last two destinations, this will be used to prevent enemies going back and forth between two points and loop around instead

// Then set a new target

if (isResetting)

{

target = transform;

if (currentTime >= 1f)

{

isResetting = false;

isStarting = true;

}

}

else if (Vector2.Distance(transform.position, target.position) < .002f)

{

lastPath = lastTarget;

lastTarget = target;

target = getTarget();

}

// Moves enemy towards target position

if (!GM.GetComponent<PausedMenu>().isPaused)

transform.position = Vector2.MoveTowards(transform.position, target.position, speed \* Time.deltaTime);

// Switching between start, roam, and chase

if (currentTime >= timeToStart && isStarting)

{

SwitchToRoam();

}

else if (currentTime >= timeToRoam)

{

isRoaming = false;

isChasing = true;

}

}

// Handles a switch to the roam state

void SwitchToRoam()

{

target = GameObject.FindGameObjectWithTag("StartOne").transform;

isStarting = false;

isRoaming = true;

currentTime = 0;

}

// Sets the target for the enemy

Transform getTarget()

{

if (isStarting) return getTargetStartState();

// Send out a raycast in all directions

RaycastHit2D[] hitsUp = Physics2D.RaycastAll(transform.position, Vector2.up);

RaycastHit2D[] hitsDown = Physics2D.RaycastAll(transform.position, Vector2.down);

RaycastHit2D[] hitsLeft = Physics2D.RaycastAll(transform.position, Vector2.left);

RaycastHit2D[] hitsRight = Physics2D.RaycastAll(transform.position, Vector2.right);

// Compile the results into one list

RaycastHit2D[][] allDirs = { hitsUp, hitsDown, hitsLeft, hitsRight };

List<Transform> possiblePoints = new List<Transform>();

// Loop through the results one direction at a time

foreach (RaycastHit2D[] curDir in allDirs)

{

for (int i = 1; i < curDir.Length; i++)

{

// If you hit a wall, don't bother checking anymore hits in this direction

if (curDir[i].transform.tag == "Wall")

{

break;

}

// If you hit a node, check to make sure it isn't the node you're currently at

// and that it isn't the node you were just previously at

// If both conditions are met, add to the list of possible points and check another direction

else if (curDir[i].transform.tag == "Node")

{

if (curDir[i].transform != target)

{

possiblePoints.Add(curDir[i].transform);

break;

}

}

}

}

// if more than one option, don't go the same way you just came

// that is unless you're chasing the player

if (possiblePoints.Count > 1 && !isChasing)

{

for (int j = 0; j < possiblePoints.Count; j++)

{

if (possiblePoints[j] == lastPath)

possiblePoints.RemoveAt(j);

}

}

return getTargetOtherState(possiblePoints);

}

// Targeting is different for start state so it is its own method

private Transform getTargetStartState()

{

Transform node = null;

if (down)

{

node = Physics2D.OverlapBox(transform.position + new Vector3(0, -.75f, 0), new Vector2(0.1f, 0.1f), .5f).transform;

down = false;

}

else

{

down = true;

node = Physics2D.OverlapBox(transform.position + new Vector3(0, .75f, 0), new Vector2(0.1f, 0.1f), .5f).transform;

}

return node;

}

// Handles targeting for roaming, chasing, and dieing states

private Transform getTargetOtherState(List<Transform> possiblePoints)

{

Transform node = null;

if (isRoaming)

{

// Select a random point to travel to from the list of available points

int j = Random.Range(0, possiblePoints.Count);

node = possiblePoints[j];

}

else if (isChasing)

{

float closest = float.MaxValue;

foreach (Transform point in possiblePoints)

{

if (Vector2.Distance(point.transform.position, player.transform.position) < closest)

{

node = point;

closest = Vector2.Distance(point.transform.position, player.transform.position);

}

}

}

else if (isDieing)

{

if (Vector2.Distance(transform.position, start.transform.position) < .002f)

{

isStarting = true;

isDieing = false;

GetComponent<BoxCollider2D>().enabled = true;

speed /= 2f;

return getTargetStartState();

}

float closest = float.MaxValue;

foreach (Transform point in possiblePoints)

{

if (Vector2.Distance(point.transform.position, start.transform.position) < closest)

{

node = point;

closest = Vector2.Distance(point.transform.position, start.transform.position);

}

}

}

return node;

}

// Ensures all variables are properly set upon death

public void Die()

{

currentTime = -5;

isStarting = false;

isRoaming = false;

isChasing = false;

isDieing = true;

GetComponent<BoxCollider2D>().enabled = false;

down = true;

if (isFast) timeToRoam = 5;

else timeToRoam = 10;

speed \*= 2f;

}

// Ensures all variables are properly set upon reset

public void Reset()

{

isStarting = false;

isRoaming = false;

isChasing = false;

if (isDieing) speed /= 2;

isDieing = false;

isResetting = true;

currentTime = 0;

transform.position = startPos;

GetComponent<BoxCollider2D>().enabled = true;

lastPath = null;

lastTarget = null;

down = true;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using TMPro;

public class DisplayHighScore : MonoBehaviour

{

[SerializeField] TMP\_Text highScore;

// Changes high Score text to whatever the currently saved high score is

void Start()

{

highScore.text = "High Score: " + PlayerPrefs.GetInt("HighScoreNumber", 0).ToString();

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using TMPro;

public class DisplayCurrentScore : MonoBehaviour

{

[SerializeField] TMP\_Text currentScore;

GameObject GM;

// Finds the game manager to access the current score

void Start()

{

GM = GameObject.FindGameObjectWithTag("GM");

}

// Changes current Score text to whatever the current score is

private void Update()

{

currentScore.text = "Your Score: " + GM.GetComponent<Score>().score.ToString();

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using TMPro;

public class DisplayCurrentLives : MonoBehaviour

{

[SerializeField] TMP\_Text currentLives;

GameObject GM;

// Finds the game manager to access the current lifecount

void Start()

{

GM = GameObject.FindGameObjectWithTag("GM");

}

// Changes current lives text to whatever the current lifecount is

private void Update()

{

currentLives.text = "Lives: " + GM.GetComponent<GameOver>().lives.ToString();

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CharacterMovement : MonoBehaviour

{

public float speed;

public float distToTravel;

Vector2 potentialPos;

public Vector2 target;

bool movingUp = false;

bool movingDown = false;

bool movingLeft = false;

bool movingRight = false;

public float wallDetectorSize;

public bool hitWall = false;

public LayerMask Walls;

[SerializeField] private AudioSource source;

[SerializeField] private AudioClip moveSound;

// Sets initial values

void Start()

{

target = transform.position;

}

void Update()

{

// always move to your target

transform.position = Vector2.MoveTowards(transform.position, target, speed \* Time.deltaTime);

// if player input, move

if (movingDown || movingUp || movingLeft || movingRight)

{

StartCoroutine("Move");

}

// setting moving direction

if (Input.GetKeyDown(KeyCode.W))

{

movingUp = true;

movingDown = false;

movingRight = false;

movingLeft = false;

transform.rotation = Quaternion.Euler(0, 0, -90);

playMovementSound();

}

if (Input.GetKeyDown(KeyCode.S))

{

movingDown = true;

movingUp = false;

movingRight = false;

movingLeft = false;

transform.rotation = Quaternion.Euler(0, 0, 90);

playMovementSound();

}

if (Input.GetKeyDown(KeyCode.A))

{

movingLeft = true;

movingDown = false;

movingUp = false;

movingRight = false;

transform.rotation = Quaternion.Euler(0, 0, 0);

playMovementSound();

}

if (Input.GetKeyDown(KeyCode.D))

{

movingRight = true;

movingDown = false;

movingUp = false;

movingLeft = false;

transform.rotation = Quaternion.Euler(0, 0, 180);

playMovementSound();

}

}

// Plays the movement sound effect if it enabled in preferences and avoid overlapping sounds

void playMovementSound()

{

if(PlayerPrefs.GetInt("SoundEffects", 1) == 1 && source.isPlaying == false)

{

source.Play();

source.loop = true;

}

}

// this IEnumerator repeats every second to set character target

IEnumerator Move()

{

// If your at your target

if (Vector2.Distance(transform.position, target) < 0.002f)

{

// setting a new potential target position

if (movingUp)

potentialPos = new Vector2(transform.position.x, transform.position.y + distToTravel);

else if (movingDown)

potentialPos = new Vector2(transform.position.x, transform.position.y - distToTravel);

else if (movingRight)

potentialPos = new Vector2(transform.position.x + distToTravel, transform.position.y);

else if (movingLeft)

potentialPos = new Vector2(transform.position.x - distToTravel, transform.position.y);

// detecting wall collision

if (!Physics2D.OverlapBox(potentialPos, new Vector2(wallDetectorSize, wallDetectorSize), .5f, Walls))

{

target = potentialPos;

hitWall = false;

}

else

{

target = transform.position;

movingRight = false;

movingUp = false;

movingDown = false;

movingLeft = false;

source.Stop();

yield break;

}

}

yield return new WaitForSeconds(1);

}

// Just a visualizer for the editor so you can see the WallDetectorSize in action

private void OnDrawGizmos()

{

Gizmos.color = new Color(1, 0, 0, 0.5f);

Gizmos.DrawCube(potentialPos, new Vector3(wallDetectorSize, wallDetectorSize, wallDetectorSize));

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class GameOver : MonoBehaviour

{

[SerializeField] public int lives = 3;

public bool gameOver = false;

public GameObject PlayerInfoCanvas;

// Checks if the game should end

private void Update()

{

if (lives <= 0) EndGame();

}

// Sets all appropriate variables

public void EndGame()

{

gameOver = true;

GetComponent<PausedMenu>().isPaused = true;

GetComponent<GameOverMenu>().setOn();

int score = this.GetComponent<Score>().score;

PlayerInfoCanvas.SetActive(false);

if (score > PlayerPrefs.GetInt("HighScoreNumber", -1))

{

PlayerPrefs.SetInt("HighScoreNumber", score);

PlayerPrefs.SetString("HighScoreName", PlayerPrefs.GetString("CurrentName", "EMPTY"));

}

Time.timeScale = 0f;

}

// A call to remove a life from lifecount

public void decrementLives()

{

lives--;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class GameOverMenu : MonoBehaviour

{

public GameObject gameOverMenu;

// Ensures the menu is off upon load

public void Set()

{

gameOverMenu.SetActive(false);

}

// Turns on the menu

public void setOn()

{

gameOverMenu.SetActive(true);

}

// Returns to the main menu screen and sets proper variables

public void MainMenu()

{

Time.timeScale = 1f;

GetComponent<GameOver>().gameOver = false;

Destroy(this.gameObject);

SceneManager.LoadScene("MainMenu", LoadSceneMode.Single);

}

// Quits the application

public void QuitGame()

{

Application.Quit();

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PausedMenu : MonoBehaviour

{

public bool isPaused;

public GameObject pausedMenu;

// Don't display pause menu on start

public void Set()

{

pausedMenu.SetActive(false);

}

// Checks for escape button for pause menu

void Update()

{

if (Input.GetKeyDown(KeyCode.Escape) && !GetComponent<GameOver>().gameOver) {

if (isPaused) {

ResumeGame();

} else {

PausedGame();

}

}

}

// Pauses the game

public void PausedGame() {

pausedMenu.SetActive(true);

Time.timeScale = 0f;

isPaused = true;

}

// Resumes the game

public void ResumeGame()

{

pausedMenu.SetActive(false);

Time.timeScale = 1f;

isPaused = false;

}

// Quits the game

public void QuitGame() {

Application.Quit();

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Reset : MonoBehaviour

{

GameObject ghost;

public GameObject[] enemies;

public List<GameObject> cloneEnemies = null;

GameObject player;

// Ensures the time is moving properly

private void Start()

{

Time.timeScale = 1f;

}

// Sets initial values

public void SetValues()

{

ghost = GameObject.Find("GhostEnemy");

ghost = GameObject.Find("GhostEnemy");

enemies = GameObject.FindGameObjectsWithTag("Enemy");

player = GameObject.FindGameObjectWithTag("Player");

cloneEnemies = new List<GameObject>();

cloneEnemies.Add(GameObject.FindGameObjectWithTag("Clone"));

}

// Sets initial values for clone enemies

public void SetCloneEnemies(GameObject clone)

{

cloneEnemies.Add(clone);

}

// Ensures values are properly set whenever a restart is needed

public void ResetEverything()

{

ghost.transform.position = ghost.GetComponent<GhostEnemy>().startPos;

foreach(GameObject e in enemies)

{

e.GetComponent<EnemyNavigation>().Reset();

}

for (int i = cloneEnemies.Count - 1; i > 0; i--)

{

GameObject tmp = cloneEnemies[i];

cloneEnemies.RemoveAt(i);

Destroy(tmp);

}

if(cloneEnemies[0] != null)

cloneEnemies[0].GetComponent<EnemyNavigation>().Reset();

player.GetComponent<Player>().Reset();

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Score : MonoBehaviour

{

public int score;

// Score needs to be consistent throughout all levels, so do not get rid of this game object

private void Awake()

{

DontDestroyOnLoad(this.gameObject);

}

// Sets initial values

void Start()

{

score = 0;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class SpawnEnemies : MonoBehaviour

{

[SerializeField] private Transform startNodeParent;

[SerializeField] private Transform enemy;

[SerializeField] private Transform fastEnemy;

[SerializeField] private Transform multiplyingEnemy;

private Transform[] startNodes = new Transform[6];

public int level = 1;

// Handles spawning based on current level

public void Spawn()

{

startNodeParent = GameObject.FindGameObjectWithTag("StartNode").transform;

for (int i = 0; i < startNodeParent.childCount; i++)

{

startNodes[i] = startNodeParent.GetChild(i);

}

if (level == 1)

{

Instantiate(enemy, new Vector3(startNodes[0].position.x, startNodes[0].position.y, startNodes[0].position.z), Quaternion.identity);

Instantiate(fastEnemy, new Vector3(startNodes[2].position.x, startNodes[2].position.y, startNodes[2].position.z), Quaternion.identity);

Instantiate(enemy, new Vector3(startNodes[4].position.x, startNodes[4].position.y, startNodes[4].position.z), Quaternion.identity);

GetComponent<Reset>().SetValues();

} else if (level == 2)

{

Instantiate(fastEnemy, new Vector3(startNodes[0].position.x, startNodes[0].position.y, startNodes[0].position.z), Quaternion.identity);

Instantiate(fastEnemy, new Vector3(startNodes[2].position.x, startNodes[2].position.y, startNodes[2].position.z), Quaternion.identity);

Instantiate(fastEnemy, new Vector3(startNodes[4].position.x, startNodes[4].position.y, startNodes[4].position.z), Quaternion.identity);

GetComponent<Reset>().SetValues();

} else if (level >= 3)

{

Instantiate(fastEnemy, new Vector3(startNodes[0].position.x, startNodes[0].position.y, startNodes[0].position.z), Quaternion.identity);

Instantiate(multiplyingEnemy, new Vector3(startNodes[2].position.x, startNodes[2].position.y, startNodes[2].position.z), Quaternion.identity);

Instantiate(fastEnemy, new Vector3(startNodes[4].position.x, startNodes[4].position.y, startNodes[4].position.z), Quaternion.identity);

GetComponent<Reset>().SetValues();

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine.SceneManagement;

using UnityEngine;

public class WonLevel : MonoBehaviour

{

GameObject[] pointObjects;

public int pointsRemaining;

// Sets initial values and ensures that only one game manager is present in the scene

void Start()

{

Set();

if(GameObject.Find("GameManager") && GameObject.Find("GameManager") != this.gameObject)

{

Destroy(this.gameObject);

}

}

// Sets initial values

void Set()

{

pointObjects = GameObject.FindGameObjectsWithTag("Point");

pointsRemaining = pointObjects.Length;

}

// Checks if the level has been won

void Update()

{

if (pointsRemaining <= 0) WinLevel();

}

// Loads the next level

private void WinLevel()

{

GetComponent<SpawnEnemies>().level++;

if(SceneManager.GetActiveScene().name == "Level\_01")

SceneManager.LoadScene("Level\_02", LoadSceneMode.Single);

else if (SceneManager.GetActiveScene().name == "Level\_02")

SceneManager.LoadScene("Level\_01", LoadSceneMode.Single);

}

// Sets variables in the new level

public void setGame()

{

GetComponent<Reset>().SetValues();

Set();

GameObject gMenu = GetComponent<GameOverMenu>().gameOverMenu;

GetComponent<GameOver>().PlayerInfoCanvas = GameObject.Find("InfoCanvas");

GameObject pMenu = GetComponent<PausedMenu>().pausedMenu;

gMenu.SetActive(false);

pMenu.SetActive(false);

}

}