MOVE ENEMY

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MoveEnemy : MonoBehaviour {

// Use this for initialization

[HideInInspector]

public GameObject[] waypoints;

private int currentWaypoint = 0;

private float lastWaypointSwitchTime;

public float speed = 1.0f;

void Start () {

lastWaypointSwitchTime = Time.time;

}

// Update is called once per frame

void Update () {

// From the waypoints array, you retrieve the start and end position for the current path segment.

Vector3 startPosition = waypoints[currentWaypoint].transform.position;

Vector3 endPosition = waypoints[currentWaypoint + 1].transform.position;

/\* Calculate the time needed for the whole distance with the formula time = distance / speed,

then determine the current time on the path. Using Vector2.Lerp, you interpolate the current

position of the enemy between the segment's start and end positions. \*/

float pathLength = Vector3.Distance(startPosition, endPosition);

float totalTimeForPath = pathLength / speed;

float currentTimeOnPath = Time.time - lastWaypointSwitchTime;

gameObject.transform.position = Vector2.Lerp(startPosition, endPosition, currentTimeOnPath / totalTimeForPath);

// Check whether the enemy has reached the endPosition. If yes, handle these two possible scenarios:

if (gameObject.transform.position.Equals(endPosition))

{

if (currentWaypoint < waypoints.Length - 2)

{

/\* The enemy is not yet at the last waypoint, so increase currentWaypoint and update

lastWaypointSwitchTime. Later, you'll add code to rotate the enemy so it points in the

direction it's moving, too.\*/

currentWaypoint++;

lastWaypointSwitchTime = Time.time;

//TODO: Rotate into move direction

}

else

{

/\* The enemy reached the last waypoint, so this destroys it and triggers a sound effect.

Later you'll add code to decrease the player's health, too. \*/

Destroy(gameObject);

AudioSource audioSource = gameObject.GetComponent<AudioSource>();

AudioSource.PlayClipAtPoint(audioSource.clip, transform.position);

// TODO: deduct health

}

}

}

MOVE ENEMY (PART II)

private void RotateIntoMoveDirection()

{

//Calculate the bug’s current movement direction by subtracting the current waypoint’s position from that of the next waypoint.

Vector3 newStartPosition = waypoints[currentWaypoint].transform.position;

Vector3 newEndPosition = waypoints[currentWaypoint + 1].transform.position;

Vector3 newDirection = (newEndPosition - newStartPosition);

//Use Mathf.Atan2 to determine the angle toward which newDirection points, in radians, assuming zero points to the right. Multiplying the result by 180 / Mathf.PI converts the angle to degrees.

float x = newDirection.x;

float y = newDirection.y;

float rotationAngle = Mathf.Atan2(y, x) \* 180 / Mathf.PI;

//Finally, retrieve the child named Sprite and rotates it rotationAngle degrees along the z-axis. Note that you rotate the child instead of the parent so the health bar — we'll add later — remains horizontal.

GameObject sprite = gameObject.transform.Find("Sprite").gameObject;

sprite.transform.rotation = Quaternion.AngleAxis(rotationAngle, Vector3.forward);

}

GameManager Behavior

private int wave;

public int Wave

{

get

{

return wave;

}

set

{

wave = value;

if (!gameOver)

{

for (int i = 0; i < nextWaveLabels.Length; i++)

{

nextWaveLabels[i].GetComponent<Animator>().SetTrigger("nextWave");

}

}

waveLabel.text = "WAVE: " + (wave + 1);

}

}

SPAWN ENEMY

[System.Serializable]

public class Wave

{

public GameObject enemyPrefab;

public float spawnInterval = 2;

public int maxEnemies = 20;

}

public Wave[] waves;

public int timeBetweenWaves = 5;

private GameManagerBehavior gameManager;

private float lastSpawnTime;

private int enemiesSpawned = 0;

void Start () {

//Instantiate(testEnemyPrefab).GetComponent<MoveEnemy>().waypoints = waypoints;

lastSpawnTime = Time.time;

gameManager =

GameObject.Find("GameManager").GetComponent<GameManagerBehavior>();

}

void Update () {

//Get the index of the current wave, and check if it’s the last one.

int currentWave = gameManager.Wave;

if (currentWave < waves.Length)

{

//If so, calculate how much time passed since the last enemy spawn and whether it’s time to spawn an enemy.

//Here you consider two cases. If it’s the first enemy in the wave, you check whether timeInterval is bigger than timeBetweenWaves.

//Otherwise, you check whether timeInterval is bigger than this wave’s spawnInterval.

//In either case, you make sure you haven’t spawned all the enemies for this wave.

float timeInterval = Time.time - lastSpawnTime;

float spawnInterval = waves[currentWave].spawnInterval;

if (((enemiesSpawned == 0 && timeInterval > timeBetweenWaves) ||

timeInterval > spawnInterval) &&

enemiesSpawned < waves[currentWave].maxEnemies)

{

//If necessary, spawn an enemy by instantiating a copy of enemyPrefab. You also increase the enemiesSpawned count.

lastSpawnTime = Time.time;

GameObject newEnemy = (GameObject)

Instantiate(waves[currentWave].enemyPrefab);

newEnemy.GetComponent<MoveEnemy>().waypoints = waypoints;

enemiesSpawned++;

}

//You check the number of enemies on screen. If there are none and it was the last enemy in the wave you spawn the next wave.

//You also give the player 10 percent of all gold left at the end of the wave.

if (enemiesSpawned == waves[currentWave].maxEnemies &&

GameObject.FindGameObjectWithTag("Enemy") == null)

{

gameManager.Wave++;

gameManager.Gold = Mathf.RoundToInt(gameManager.Gold \* 1.1f);

enemiesSpawned = 0;

lastSpawnTime = Time.time;

}

//Upon beating the last wave this runs the game won animation.

}

else

{

gameManager.gameOver = true;

GameObject gameOverText = GameObject.FindGameObjectWithTag("GameWon");

gameOverText.GetComponent<Animator>().SetBool("gameOver", true);

}

}

GameBehavior Part II

private int health;

public int Health

{

get

{

return health;

}

set

{

//If you're reducing the player's health, use the CameraShake component to create a nice shake effect.

//This script was created by a former student - Mike Jasper.

if (value < health)

{

Camera.main.GetComponent<CameraShake>().Shake();

}

//Update the private variable and the health label in the top left corner of the screen.

health = value;

healthLabel.text = "HEALTH: " + health;

//If health drops to 0 and it's not yet game over, set gameOver to true and trigger the GameOver animation.

if (health <= 0 && !gameOver)

{

gameOver = true;

GameObject gameOverText = GameObject.FindGameObjectWithTag("GameOver");

gameOverText.GetComponent<Animator>().SetBool("gameOver", true);

}

//Remove one of the monsters from the cookie. If it just disabled them, this bit could be written more simply,

//but it also supports re-enabling them when you add health.

for (int i = 0; i < healthIndicator.Length; i++)

{

if (i < Health)

{

healthIndicator[i].SetActive(true);

}

else

{

healthIndicator[i].SetActive(false);

}

}

}

}

MoveEnemy Part III

GameManagerBehavior gameManager =

GameObject.Find("GameManager").GetComponent<GameManagerBehavior>();

gameManager.Health -= 1;

HEALTHBAR.CS

public float maxHealth = 100;

public float currentHealth = 100;

private float originalScale;

void Start () {

originalScale = gameObject.transform.localScale.x;

}

// Update is called once per frame

void Update () {

Vector3 tmpScale = gameObject.transform.localScale;

tmpScale.x = currentHealth / maxHealth \* originalScale;

gameObject.transform.localScale = tmpScale;

}

ENEMYDESTRUCTIONDELEGATE.CS

public delegate void EnemyDelegate(GameObject enemy);

public EnemyDelegate enemyDelegate;

void OnDestroy()

{

if (enemyDelegate != null)

{

enemyDelegate(gameObject);

}

}

SHOOT ENEMY

void OnEnemyDestroy(GameObject enemy)

{

enemiesInRange.Remove(enemy);

}

void OnTriggerEnter2D(Collider2D other)

{

//You then add the enemy to the list of enemiesInRange and add OnEnemyDestroy to the EnemyDestructionDelegate.

//This makes sure that OnEnemyDestroy is called when the enemy is destroyed.

if (other.gameObject.tag.Equals("Enemy"))

{

enemiesInRange.Add(other.gameObject);

EnemyDestructionDelegate del =

other.gameObject.GetComponent<EnemyDestructionDelegate>();

del.enemyDelegate += OnEnemyDestroy;

}

}

//In OnTriggerExit2D you remove the enemy from the list and unregister your delegate.

//Now you know which enemies are in range.

void OnTriggerExit2D(Collider2D other)

{

if (other.gameObject.tag.Equals("Enemy"))

{

enemiesInRange.Remove(other.gameObject);

EnemyDestructionDelegate del =

other.gameObject.GetComponent<EnemyDestructionDelegate>();

del.enemyDelegate -= OnEnemyDestroy;

}

}

Move Enemy Part IV

public float DistanceToGoal()

{

float distance = 0;

distance += Vector2.Distance(

gameObject.transform.position,

waypoints[currentWaypoint + 1].transform.position);

for (int i = currentWaypoint + 1; i < waypoints.Length - 1; i++)

{

Vector3 startPosition = waypoints[i].transform.position;

Vector3 endPosition = waypoints[i + 1].transform.position;

distance += Vector2.Distance(startPosition, endPosition);

}

return distance;

}

BulletBehavior

// Use this for initialization

public float speed = 10;

public int damage;

public GameObject target;

public Vector3 startPosition;

public Vector3 targetPosition;

private float distance;

private float startTime;

private GameManagerBehavior gameManager;

void Start () {

startTime = Time.time;

distance = Vector2.Distance(startPosition, targetPosition);

GameObject gm = GameObject.Find("GameManager");

gameManager = gm.GetComponent<GameManagerBehavior>();

}

// Update is called once per frame

void Update () {

//Calculate the new bullet position using Vector3.Lerp to interpolate between start and end positions.

float timeInterval = Time.time - startTime;

gameObject.transform.position = Vector3.Lerp(startPosition, targetPosition, timeInterval \* speed / distance);

//If the bullet reaches the targetPosition, you verify that target still exists.

if (gameObject.transform.position.Equals(targetPosition))

{

if (target != null)

{

//You retrieve the target's HealthBar component and reduce its health by the bullet's damage.

Transform healthBarTransform = target.transform.Find("HealthBar");

HealthBar healthBar =

healthBarTransform.gameObject.GetComponent<HealthBar>();

healthBar.currentHealth -= Mathf.Max(damage, 0);

//If the health of the enemy falls to zero, you destroy it, play a sound effect and reward the player for marksmanship.

if (healthBar.currentHealth <= 0)

{

Destroy(target);

AudioSource audioSource = target.GetComponent<AudioSource>();

AudioSource.PlayClipAtPoint(audioSource.clip, transform.position);

gameManager.Gold += 50;

}

}

Destroy(gameObject);

}

}

SHOOT ENEMY PART II

void Shoot(Collider2D target)

{

GameObject bulletPrefab = monsterData.CurrentLevel.bullet;

//Get the start and target positions of the bullet. Set the z-Position to that of bulletPrefab.

//Earlier, you set the bullet prefab's z position value to make sure the bullet appears behind the monster

//firing it, but in front of the enemies.

Vector3 startPosition = gameObject.transform.position;

Vector3 targetPosition = target.transform.position;

startPosition.z = bulletPrefab.transform.position.z;

targetPosition.z = bulletPrefab.transform.position.z;

//Instantiate a new bullet using the bulletPrefab for MonsterLevel. Assign the startPosition and targetPosition of the bullet.

GameObject newBullet = (GameObject)Instantiate(bulletPrefab);

newBullet.transform.position = startPosition;

BulletBehavior bulletComp = newBullet.GetComponent<BulletBehavior>();

bulletComp.target = target.gameObject;

bulletComp.startPosition = startPosition;

bulletComp.targetPosition = targetPosition;

//Make the game juicier: Run a shoot animation and play a laser sound whenever the monster shoots.

Animator animator =

monsterData.CurrentLevel.visualization.GetComponent<Animator>();

animator.SetTrigger("fireShot");

AudioSource audioSource = gameObject.GetComponent<AudioSource>();

audioSource.PlayOneShot(audioSource.clip);

}

SHOOT ENEMY PART III

//In OnEnemyDestroy, you remove the enemy from enemiesInRange.

//When an enemy walks on the trigger around your monster OnTriggerEnter2D is called.

GameObject target = null;

//Determine the target of the monster. Start with the maximum possible distance in the minimalEnemyDistance.

//Iterate over all enemies in range and make an enemy the new target if its distance to the cookie is smaller

//than the current minimum.

float minimalEnemyDistance = float.MaxValue;

foreach (GameObject enemy in enemiesInRange)

{

float distanceToGoal = enemy.GetComponent<MoveEnemy>().DistanceToGoal();

if (distanceToGoal<minimalEnemyDistance)

{

target = enemy;

minimalEnemyDistance = distanceToGoal;

}

}

//Call Shoot if the time passed is greater than the fire rate of your monster and set lastShotTime to the current time.

if (target != null)

{

if (Time.time - lastShotTime > monsterData.CurrentLevel.fireRate)

{

Shoot(target.GetComponent<Collider2D>());

lastShotTime = Time.time;

}

//Calculate the rotation angle between the monster and its target. You set the rotation of the monster to this angle.

//Now it always faces the target.

Vector3 direction = gameObject.transform.position - target.transform.position;

gameObject.transform.rotation = Quaternion.AngleAxis(

Mathf.Atan2 (direction.y, direction.x) \* 180 / Mathf.PI,

new Vector3(0, 0, 1));

}