**SOURCE CODE**



**Muhammad Syahrul Anwar**

**(1995114013)**

**PROGRAM STUDI S1 TEKNIK INFORMATIKA**

**FAKULTAS TEKNOLOGI INFORMASI**

**UNIVERSITAS HASYIM ASY’ARI**

**TEBUIRENG JOMBANG**

**2023**

**PlayerInput.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.Events;

public class PlayerInput : MonoBehaviour

{

[SerializeField]

private Camera mainCamera;

public UnityEvent OnShoot = new UnityEvent();

public UnityEvent<Vector2> OnMoveBody = new UnityEvent<Vector2>();

public UnityEvent<Vector2> OnMoveTurret = new UnityEvent<Vector2>();

private void Awake()

{

if (mainCamera == null)

mainCamera = Camera.main;

}

//Update is called once per frame

void Update()

{

GetBodyMovement();

GetTurretMovement();

GetShootingInput();

}

private void GetShootingInput()

{

if(Input.GetMouseButtonDown(0))//GetMouseButtonDown(0))

{

OnShoot?.Invoke();

}

}

private void GetTurretMovement()

{

OnMoveTurret?.Invoke(GetMousePosition());

}

private Vector2 GetMousePosition()

{

Vector3 mousePosition = Input.mousePosition;

mousePosition.z = mainCamera.nearClipPlane;

Vector2 mouseWorldPosition = mainCamera.ScreenToWorldPoint(mousePosition);

return mouseWorldPosition;

}

private void GetBodyMovement()

{

Vector2 movementVector = new Vector2(Input.GetAxisRaw("Horizontal"), Input.GetAxisRaw("Vertical"));

OnMoveBody?.Invoke(movementVector.normalized);

}

}

**TankController.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class TankController : MonoBehaviour

{

public AimTurret aimTurret;

public TankMover tankMover;

public Turret[] turrets;

private void Awake()

{

if(tankMover == null)

tankMover = GetComponentInChildren<TankMover>();

if(aimTurret == null)

aimTurret = GetComponentInChildren<AimTurret>();

if(turrets == null || turrets.Length == 0)

{

turrets = GetComponentsInChildren<Turret>();

}

}

public void HandleShoot()

{

foreach (var turret in turrets)

{

turret.Shoot();

}

}

public void HadleMoveBody(Vector2 movementVector)

{

tankMover.Move(movementVector);

}

public void HandleTurretMovement(Vector2 pointerPosition)

{

aimTurret.Aim(pointerPosition);

}

}

**TankMover.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.Events;

public class TankMover : MonoBehaviour

{

public Rigidbody2D rb2d;

public TankMovementData movementData;

private Vector2 movementVector;

private float currentSpeed = 0;

private float currentForewardDirection = 1;

public UnityEvent<float> OnSpeedChange = new UnityEvent<float>();

private void Awake()

{

rb2d = GetComponentInParent<Rigidbody2D>();

}

public void Move(Vector2 movementVector)

{

this.movementVector = movementVector;

CalculateSpeed(movementVector);

OnSpeedChange?.Invoke(this.movementVector.magnitude);

if(movementVector.y > 0)

currentForewardDirection = 1;

else if(movementVector.y < 0)

currentForewardDirection = -1;

}

private void CalculateSpeed(Vector2 movementVector)

{

if (Mathf.Abs(movementVector.y) > 0)

{

currentSpeed += movementData.acceleration \* Time.deltaTime;

}

else

{

currentSpeed -= movementData.deacceleration \* Time.deltaTime;

}

currentSpeed = Mathf.Clamp(currentSpeed, 0, movementData.maxSpeed);

}

private void FixedUpdate()

{

rb2d.velocity = (Vector2)transform.up \* currentSpeed \* currentForewardDirection \* Time.fixedDeltaTime;

rb2d.MoveRotation(transform.rotation \* Quaternion.Euler(0, 0, -movementVector.x \*

movementData.rotationSpeed \* Time.fixedDeltaTime));

}

}

**Turret.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.Events;

[RequireComponent(typeof(ObjectPool))]

public class Turret : MonoBehaviour

{

public List<Transform> turretBarrels;

public TurretData turretData;

private bool canShoot = true;

private Collider2D[] tankColliders;

private float currentDelay = 0;

private ObjectPool bulletPool;

[SerializeField]

private int bulletPoolCount = 10;

public UnityEvent OnShoot, OnCantShoot;

public UnityEvent<float> OnReloading;

private void Awake()

{

tankColliders = GetComponentsInParent<Collider2D>();

bulletPool = GetComponent<ObjectPool>();

}

private void Start()

{

bulletPool.Initialize(turretData.bulletPrefab, bulletPoolCount);

OnReloading?.Invoke(currentDelay);

}

private void Update()

{

if (canShoot == false)

{

currentDelay -= Time.deltaTime;

OnReloading?.Invoke(currentDelay/ turretData.reloadDelay);

if (currentDelay <= 0)

{

canShoot = true;

}

}

}

public void Shoot()

{

if (canShoot)

{

canShoot = false;

currentDelay = turretData.reloadDelay;

foreach (var barrel in turretBarrels)

{

//GameObject bullet = Instantiate(bulletPrefabs);

GameObject bullet = bulletPool.CreateObject();

bullet.transform.position = barrel.position;

bullet.transform.localRotation = barrel.rotation;

bullet.GetComponent<Bullet>().Initialize(turretData.bulletData);

foreach (var collider in tankColliders)

{

Physics2D.IgnoreCollision(bullet.GetComponent<Collider2D>(), collider);

}

}

OnShoot?.Invoke();

OnReloading?.Invoke(currentDelay);

}

else

{

OnCantShoot?.Invoke();

}

}

}

**AimTurret.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class AimTurret : MonoBehaviour

{

public float turretRotationSpeed = 150;

public void Aim(Vector2 inputPointerPosition)

{

var turretDirection = (Vector3)inputPointerPosition - transform.position;

var desiredAngle = Mathf.Atan2(turretDirection.y, turretDirection.x) \* Mathf.Rad2Deg;

var rotationStep = turretRotationSpeed \* Time.deltaTime;

transform.rotation = Quaternion.RotateTowards(transform.rotation, Quaternion.Euler(0, 0,

desiredAngle), rotationStep);

}

}

**Bullet.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.Events;

public class Bullet : MonoBehaviour

{

public BulletData bulletData;

private Vector2 startPosition;

private float conquaredDistance = 0;

private Rigidbody2D rb2d;

public UnityEvent OnHit = new UnityEvent();

private void Awake()

{

rb2d = GetComponent<Rigidbody2D>();

}

public void Initialize(BulletData bulletData)

{

this.bulletData = bulletData;

startPosition = transform.position;

rb2d.velocity = transform.up \* this.bulletData.speed;

}

private void Update()

{

conquaredDistance = Vector2.Distance(transform.position, startPosition);

if (conquaredDistance >= bulletData.maxDistance)

{

DisableObject();

}

}

private void DisableObject()

{

rb2d.velocity = Vector2.zero;

gameObject.SetActive(false);

}

private void OnTriggerEnter2D(Collider2D collision)

{

Debug.Log("Collider" + collision.name);

OnHit?.Invoke();

var damageable = collision.GetComponent<Damageable>();

if(damageable != null)

{

damageable.Hit(bulletData.damage);

}

DisableObject();

}

}

**ObjectPool.cs**

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ObjectPool : MonoBehaviour

{

[SerializeField]

protected GameObject objectToPool;

[SerializeField]

protected int poolSize = 10;

protected Queue<GameObject> objectPool;

public Transform spawnedObjectsParent;

public bool alwaysDestroy = false;

private void Awake()

{

objectPool = new Queue<GameObject>();

}

public void Initialize(GameObject objectToPool, int poolSize = 10)

{

this.objectToPool = objectToPool;

this.poolSize = poolSize;

}

public GameObject CreateObject()

{

CreatObjectParentIfNeeded();

GameObject spawnedObject = null;

if (objectPool.Count < poolSize)

{

spawnedObject = Instantiate(objectToPool, transform.position, Quaternion.identity);

spawnedObject.name = transform.root.name + "\_" + objectToPool.name + "\_" + objectPool.Count;

spawnedObject.transform.SetParent(spawnedObjectsParent);

spawnedObject.AddComponent<DestroyIfDisabled>();

}

else

{

spawnedObject = objectPool.Dequeue();

spawnedObject.transform.position = transform.position;

spawnedObject.transform.rotation = Quaternion.identity;

spawnedObject.SetActive(true);

}

objectPool.Enqueue(spawnedObject);

return spawnedObject;

}

private void CreatObjectParentIfNeeded()

{

if (spawnedObjectsParent == null)

{

string name = "ObjectPool\_" + objectToPool.name;

var parentObject = GameObject.Find(name);

if (parentObject != null)

spawnedObjectsParent = parentObject.transform;

else

{

spawnedObjectsParent = new GameObject(name).transform;

}

}

}

private void OnDestroy()

{

foreach(var item in objectPool)

{

if ( item == null)

continue;

else if (item.activeSelf == false || alwaysDestroy)

Destroy(item);

else

item.GetComponent<DestroyIfDisabled>().SelfDestructionEnabled = true;

}

}

}