Лабораторная работа № 31-32

Тема: Разработка игрового проекта Arkanoid

Цель: разработать игровой проект Arkanoid

Ход работы

1. Выполнение работы
2. Создал проект и разместил в него ресурсы по соответствующим папкам.
3. Добавил объект Background и настроил его Layer.

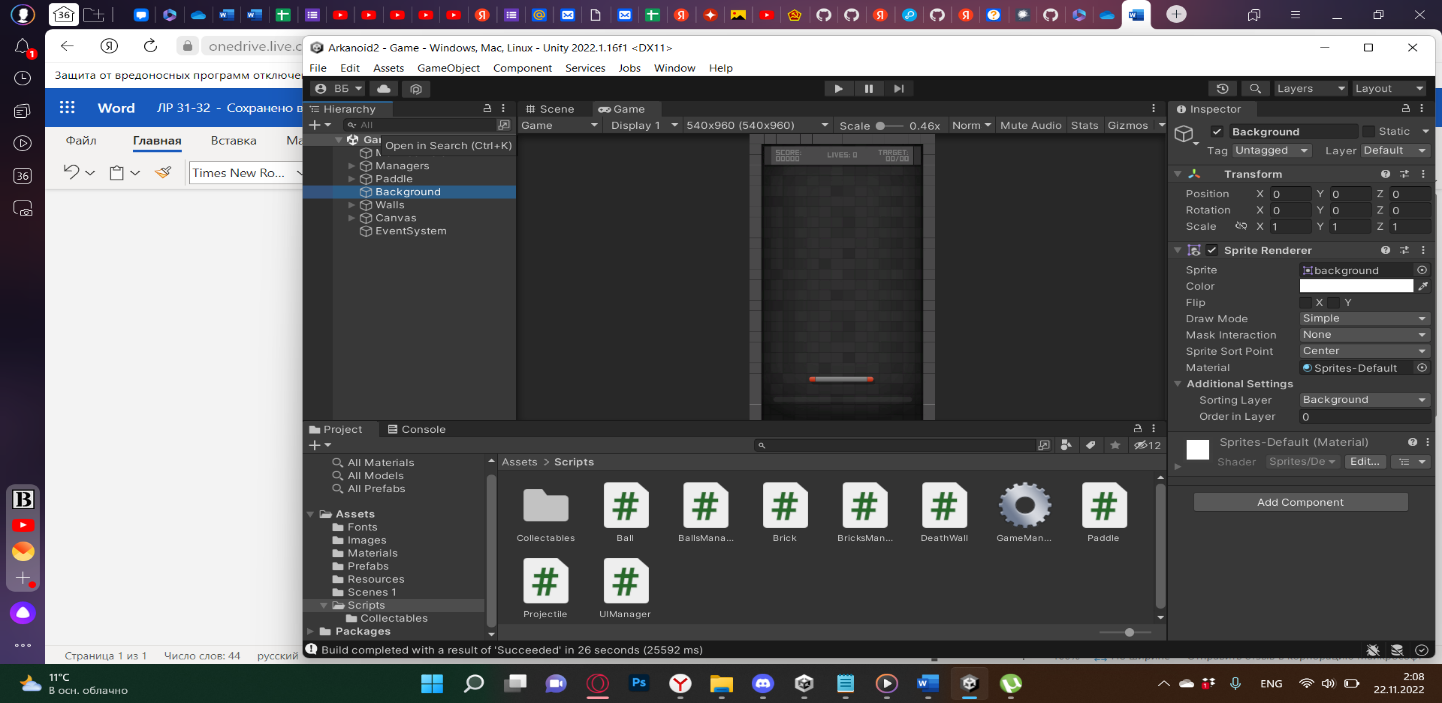


Рис. 31.1 - Инспектор Background

1. Добавил объект Paddle. Привязал к нему Box Collider, настроил его.

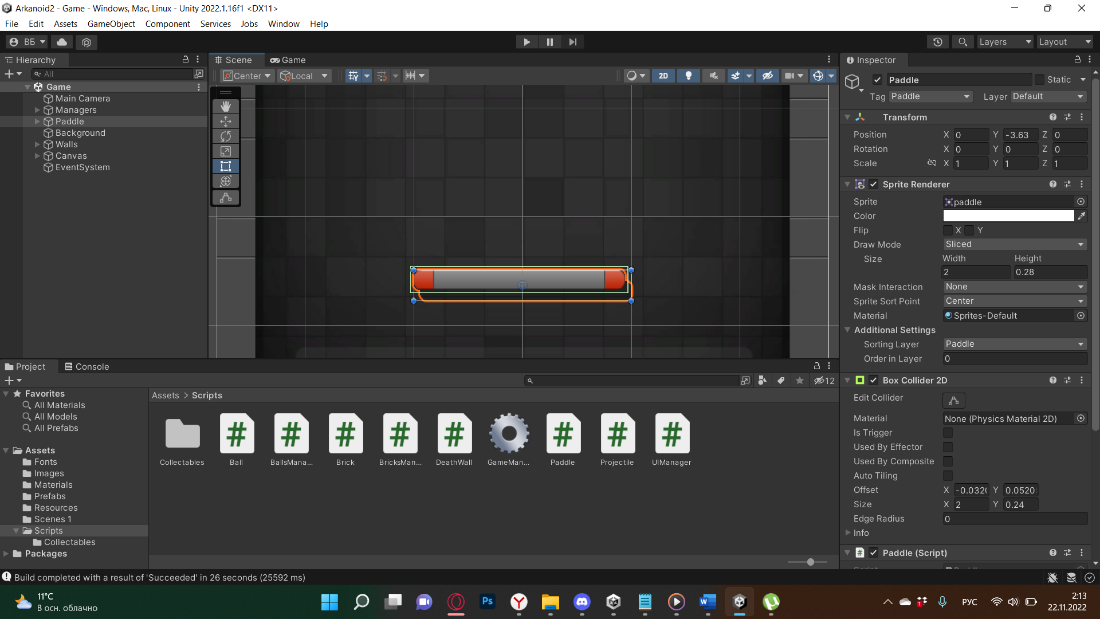


Рис. 31.2 - Инспектор Paddle

1. Создал скрипт Paddle, отвечающий за движение объекта, привязал к нему.

Листинг Paddle.cs

using System;

using System.Collections;

using UnityEngine;

public class Paddle : MonoBehaviour

{

#region Singleton

private static Paddle \_instance;

public static Paddle Instance => \_instance;

public bool PaddleIsTransforming { get; set; }

private void Awake()

{

if (\_instance != null)

{

Destroy(gameObject);

}

else

{

\_instance = this;

}

}

#endregion

private Camera mainCamera;

private float paddleInitialY;

private float defaultPaddleWidthInPixels = 200;

private float defaultLeftClamp = 135;

private float defaultRightClamp = 410;

private SpriteRenderer sr;

private BoxCollider2D boxCol;

// Shooting

public bool PaddleIsShooting { get; set; }

public GameObject leftMuzzle;

public GameObject rightMuzzle;

public Projectile bulletPrefab;

public float extendShrinkDuration = 10;

public float paddleWidth = 2;

public float paddleHeight = 0.28f;

private void Start()

{

mainCamera = FindObjectOfType<Camera>();

paddleInitialY = this.transform.position.y;

sr = GetComponent<SpriteRenderer>();

boxCol = GetComponent<BoxCollider2D>();

}

private void Update()

{

PaddleMovement();

UpdateMuzzlePosition();

}

private void UpdateMuzzlePosition()

{

leftMuzzle.transform.position = new Vector3(this.transform.position.x - (this.sr.size.x / 2) + 0.1f, this.transform.position.y + 0.2f, this.transform.position.z);

rightMuzzle.transform.position = new Vector3(this.transform.position.x + (this.sr.size.x / 2) - 0.153f, this.transform.position.y + 0.2f, this.transform.position.z);

}

public void StartWidthAnimation(float newWidth)

{

StartCoroutine(AnimatePaddleWidth(newWidth));

}

public IEnumerator AnimatePaddleWidth(float width)

{

this.PaddleIsTransforming = true;

this.StartCoroutine(ResetPaddleWidthAfterTime(this.extendShrinkDuration));

if (width > this.sr.size.x)

{

float currentWidth = this.sr.size.x;

while (currentWidth < width)

{

currentWidth += Time.deltaTime \* 2;

this.sr.size = new Vector2(currentWidth, paddleHeight);

boxCol.size = new Vector2(currentWidth, paddleHeight);

yield return null;

}

}

else

{

float currentWidth = this.sr.size.x;

while (currentWidth > width)

{

currentWidth -= Time.deltaTime \* 2;

this.sr.size = new Vector2(currentWidth, paddleHeight);

boxCol.size = new Vector2(currentWidth, paddleHeight);

yield return null;

}

}

this.PaddleIsTransforming = false;

}

private IEnumerator ResetPaddleWidthAfterTime(float seconds)

{

yield return new WaitForSeconds(seconds);

this.StartWidthAnimation(this.paddleWidth);

}

private void PaddleMovement()

{

float paddleShift = (defaultPaddleWidthInPixels - ((defaultPaddleWidthInPixels / 2) \* this.sr.size.x)) / 2;

float leftClamp = defaultLeftClamp - paddleShift;

float rightClamp = defaultRightClamp + paddleShift;

float mousePositionPixels = Mathf.Clamp(Input.mousePosition.x, leftClamp, rightClamp);

float mousePositionWorldX = mainCamera.ScreenToWorldPoint(new Vector3(mousePositionPixels, 0, 0)).x;

this.transform.position = new Vector3(mousePositionWorldX, paddleInitialY, 0);

}

private void OnCollisionEnter2D(Collision2D coll)

{

if (coll.gameObject.tag == "Ball")

{

Rigidbody2D ballRb = coll.gameObject.GetComponent<Rigidbody2D>();

Vector3 hitPoint = coll.contacts[0].point;

Vector3 paddleCenter = new Vector3(this.gameObject.transform.position.x, this.gameObject.transform.position.y);

ballRb.velocity = Vector2.zero;

float difference = paddleCenter.x - hitPoint.x;

if (hitPoint.x < paddleCenter.x)

{

ballRb.AddForce(new Vector2(-(Mathf.Abs(difference \* 200)), BallsManager.Instance.initialBallSpeed));

}

else

{

ballRb.AddForce(new Vector2((Mathf.Abs(difference \* 200)), BallsManager.Instance.initialBallSpeed));

}

}

}

public void StartShooting()

{

if (!this.PaddleIsShooting)

{

this.PaddleIsShooting = true;

StartCoroutine(StartShootingRoutine());

}

}

public IEnumerator StartShootingRoutine()

{

float fireCooldown = .5f; // TODO: extract this into unity variable

float fireCooldownLeft = 0;

float shootingDuration = 10; // TODO: extract this into unity variable

float shootingDurationLeft = shootingDuration;

//Debug.Log("START SHOOTING");

while (shootingDurationLeft >= 0)

{

fireCooldownLeft -= Time.deltaTime;

shootingDurationLeft -= Time.deltaTime;

if (fireCooldownLeft <= 0)

{

this.Shoot();

fireCooldownLeft = fireCooldown;

//Debug.Log($"Shoot at {Time.time}");

}

yield return null;

}

//Debug.Log("STOP SHOOTING!");

this.PaddleIsShooting = false;

leftMuzzle.SetActive(false);

rightMuzzle.SetActive(false);

}

private void Shoot()

{

leftMuzzle.SetActive(false);

rightMuzzle.SetActive(false);

leftMuzzle.SetActive(true);

rightMuzzle.SetActive(true);

this.SpawnBullet(leftMuzzle);

this.SpawnBullet(rightMuzzle);

}

private void SpawnBullet(GameObject muzzle)

{

Vector3 spawnPosition = new Vector3(muzzle.transform.position.x, muzzle.transform.position.y + 0.2f, muzzle.transform.position.z);

Projectile bullet = Instantiate(bulletPrefab, spawnPosition, Quaternion.identity);

Rigidbody2D bulletRb = bullet.GetComponent<Rigidbody2D>();

bulletRb.AddForce(new Vector2(0, 450f));

}

}

1. Создал объект Ball с дочерним Graphic, привязал к нему Sprite Renderer, а к родительскому - Rigidbody 2D с Circle Collider. Затем сделал его префабом.

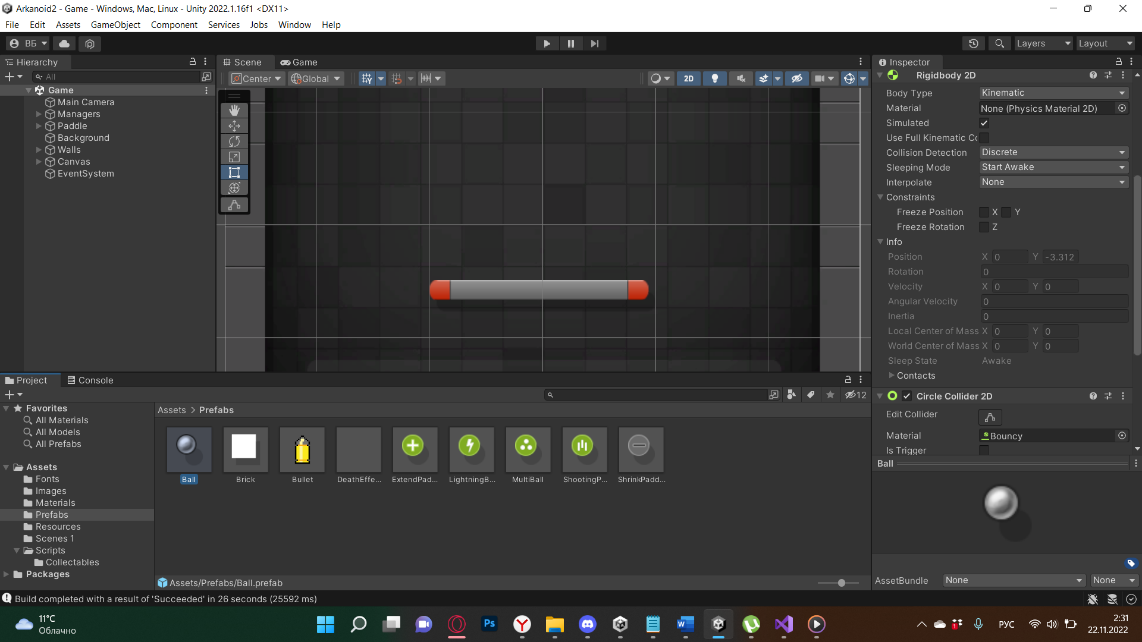


Рис. 31.3 Инспектор Ball

1. Создал скрипт Ball, отвечающий за физику объекта.

Листнинг Ball.cs

using System;

using System.Collections;

using UnityEngine;

public class Ball : MonoBehaviour

{

private SpriteRenderer sr;

public bool isLightningBall;

public ParticleSystem lightningBallEffect;

public float lightningBallDuration = 10;

public static event Action<Ball> OnBallDeath;

public static event Action<Ball> OnLightningBallEnable;

public static event Action<Ball> OnLightningBallDisable;

private void Awake()

{

this.sr = GetComponentInChildren<SpriteRenderer>();

}

public void Die()

{

OnBallDeath?.Invoke(this);

Destroy(gameObject, 1);

}

public void StartLightningBall()

{

if (!this.isLightningBall)

{

this.isLightningBall = true;

this.sr.enabled = false;

lightningBallEffect.gameObject.SetActive(true);

StartCoroutine(StopLightningBallAfterTime(this.lightningBallDuration));

OnLightningBallEnable?.Invoke(this);

}

}

private IEnumerator StopLightningBallAfterTime(float seconds)

{

yield return new WaitForSeconds(seconds);

StopLightningBall();

}

private void StopLightningBall()

{

if (this.isLightningBall)

{

this.isLightningBall = false;

this.sr.enabled = true;

lightningBallEffect.gameObject.SetActive(false);

OnLightningBallDisable?.Invoke(this);

}

}

}

1. Создал стены через коллизию, что бы Ball не вылетал из поля игры. Создал объект Brick, настроил спрайт и Box Collider. В будущем объект станет префабом.

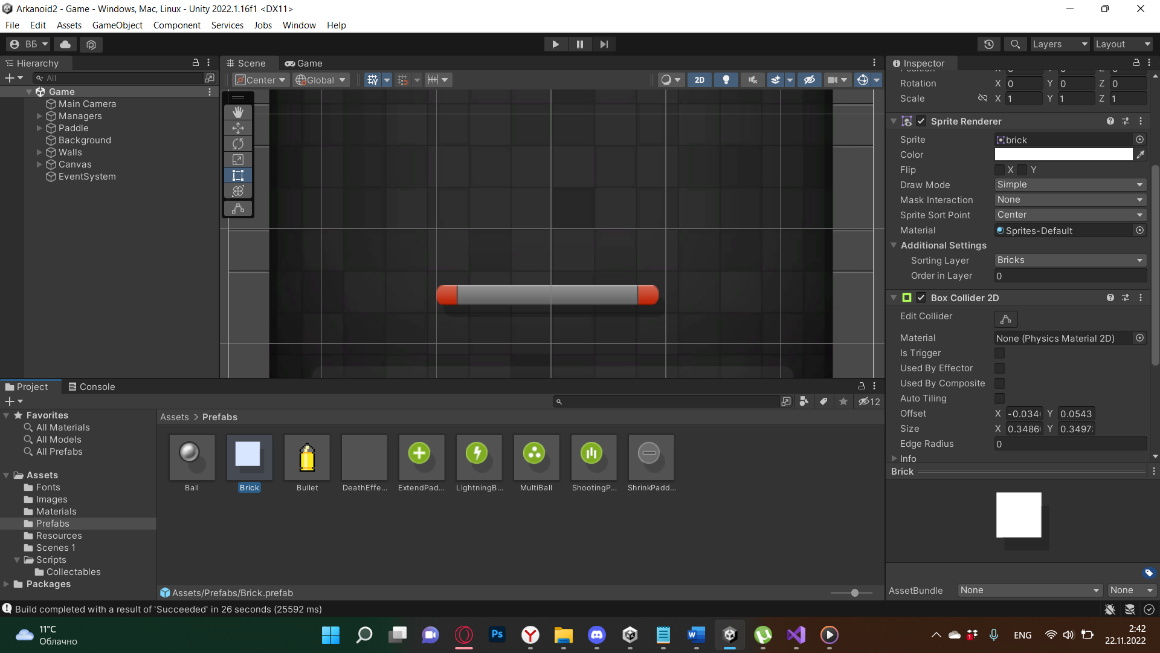


Рис 31.4 Инспектор Brick

1. Создал скрипт, отвечающий за поведение объекта Brick.

Листинг Brick.cs

using System;

using System.Collections.Generic;

using UnityEngine;

using static UnityEngine.ParticleSystem;

public class Brick : MonoBehaviour

{

private SpriteRenderer sr;

private BoxCollider2D boxCollider;

public int Hitpoints = 1;

public ParticleSystem DestroyEffect;

public static event Action<Brick> OnBrickDestruction;

private void Awake()

{

this.sr = this.GetComponent<SpriteRenderer>();

this.boxCollider = this.GetComponent<BoxCollider2D>();

Ball.OnLightningBallEnable += OnLightningBallEnable;

Ball.OnLightningBallDisable += OnLightningBallDisable;

}

private void OnLightningBallDisable(Ball obj)

{

if (this != null)

{

this.boxCollider.isTrigger = false;

}

}

private void OnLightningBallEnable(Ball obj)

{

if (this != null)

{

this.boxCollider.isTrigger = true;

}

}

private void OnCollisionEnter2D(Collision2D collision)

{

bool instantKill = false;

if (collision.collider.tag == "Ball")

{

Ball ball = collision.gameObject.GetComponent<Ball>();

instantKill = ball.isLightningBall;

}

if (collision.collider.tag == "Ball" || collision.collider.tag == "Projectile")

{

this.TakeDamage(instantKill);

}

}

private void OnTriggerEnter2D(Collider2D collision)

{

bool instantKill = false;

if (collision.tag == "Ball")

{

Ball ball = collision.gameObject.GetComponent<Ball>();

instantKill = ball.isLightningBall;

}

if (collision.tag == "Ball" || collision.tag == "Projectile")

{

this.TakeDamage(instantKill);

}

}

private void TakeDamage(bool instantKill)

{

this.Hitpoints--;

if (this.Hitpoints <= 0 || instantKill)

{

BricksManager.Instance.RemainingBricks.Remove(this);

OnBrickDestruction?.Invoke(this);

OnBrickDestroy();

SpawnDestroyEffect();

Destroy(this.gameObject);

}

else

{

this.sr.sprite = BricksManager.Instance.Sprites[this.Hitpoints - 1];

}

}

private void OnBrickDestroy()

{

float buffSpawnChance = UnityEngine.Random.Range(0, 100f);

float deBuffSpawnChance = UnityEngine.Random.Range(0, 100f);

bool alreadySpawned = false;

if (buffSpawnChance <= CollectablesManager.Instance.BuffChance)

{

alreadySpawned = true;

Collectable newBuff = this.SpawnCollectable(true);

}

if (deBuffSpawnChance <= CollectablesManager.Instance.DebuffChance && !alreadySpawned)

{

Collectable newDebuff = this.SpawnCollectable(false);

}

}

private Collectable SpawnCollectable(bool isBuff)

{

List<Collectable> collection;

if (isBuff)

{

collection = CollectablesManager.Instance.AvailableBuffs;

}

else

{

collection = CollectablesManager.Instance.AvailableDebuffs;

}

int buffIndex = UnityEngine.Random.Range(0, collection.Count);

Collectable prefab = collection[buffIndex];

Collectable newCollectable = Instantiate(prefab, this.transform.position, Quaternion.identity) as Collectable;

return newCollectable;

}

private void SpawnDestroyEffect()

{

Vector3 brickPos = gameObject.transform.position;

Vector3 spawnPosition = new Vector3(brickPos.x, brickPos.y, brickPos.z - 0.2f);

GameObject effect = Instantiate(DestroyEffect.gameObject, spawnPosition, Quaternion.identity);

MainModule mm = effect.GetComponent<ParticleSystem>().main;

mm.startColor = this.sr.color;

Destroy(effect, DestroyEffect.main.startLifetime.constant);

}

public void Init(Transform containerTransform, Sprite sprite, Color color, int hitpoints)

{

this.transform.SetParent(containerTransform);

this.sr.sprite = sprite;

this.sr.color = color;

this.Hitpoints = hitpoints;

}

private void OnDisable()

{

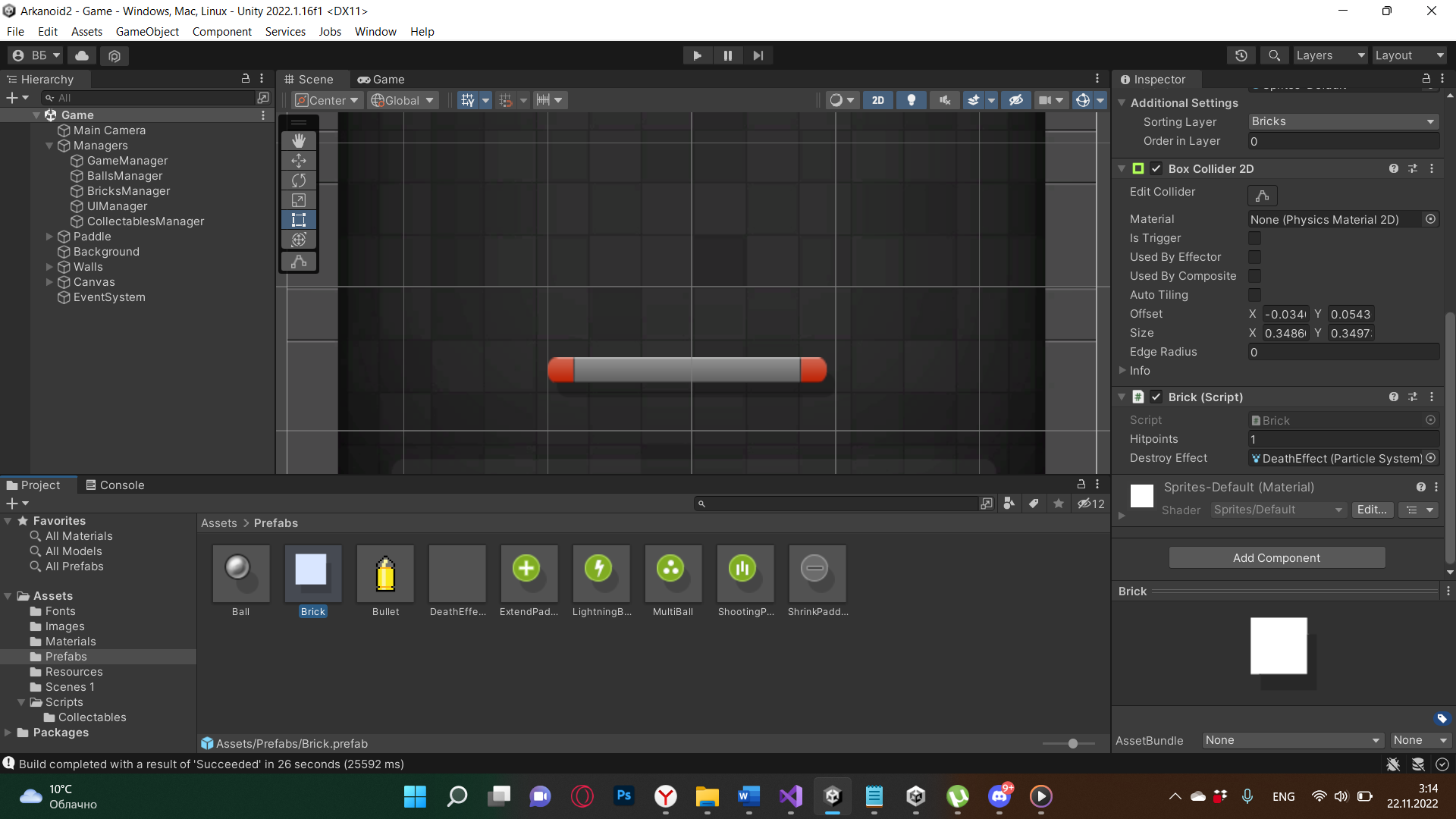
Ball.OnLightningBallEnable -= OnLightningBallEnable;

Ball.OnLightningBallDisable -= OnLightningBallDisable;

}

}

1. К объекту Bricks был привязан эффект Particle System, DeathEffect.



1. Создал скрипты и объекты Game, Balls и Brick Manager, являющиеся дочерними объекту Managers. Отвечают за взаимодействие объектов между друг другом. BrickManager отвечает за генерацию объектов “Brick”.

Листинг GameManager.cs

using System;

using UnityEngine;

using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour

{

#region Singleton

private static GameManager \_instance;

public static GameManager Instance => \_instance;

private void Awake()

{

if (\_instance != null)

{

Destroy(gameObject);

}

else

{

\_instance = this;

}

}

#endregion

public GameObject gameOverScreen;

public GameObject victoryScreen;

public int AvailibleLives = 3;

public int Lives { get; set; }

public bool IsGameStarted { get; set; }

public static event Action<int> OnLiveLost;

private void Start()

{

this.Lives = this.AvailibleLives;

Screen.SetResolution(540, 960, false);

Ball.OnBallDeath += OnBallDeath;

Brick.OnBrickDestruction += OnBrickDestruction;

}

private void OnBrickDestruction(Brick obj)

{

if (BricksManager.Instance.RemainingBricks.Count <= 0)

{

BallsManager.Instance.ResetBalls();

GameManager.Instance.IsGameStarted = false;

BricksManager.Instance.LoadNextLevel();

}

}

public void RestartGame()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);

}

private void OnBallDeath(Ball obj)

{

if (BallsManager.Instance.Balls.Count <= 0)

{

this.Lives--;

if (this.Lives < 1)

{

gameOverScreen.SetActive(true);

}

else

{

OnLiveLost?.Invoke(this.Lives);

BallsManager.Instance.ResetBalls();

IsGameStarted = false;

BricksManager.Instance.LoadLevel(BricksManager.Instance.CurrentLevel);

}

}

}

internal void ShowVictoryScreen()

{

victoryScreen.SetActive(true);

}

private void OnDisable()

{

Ball.OnBallDeath -= OnBallDeath;

}

}

Листинг BallsManager.cs

using System;

using System.Collections.Generic;

using System.Linq;

using UnityEngine;

public class BallsManager : MonoBehaviour

{

#region Singleton

private static BallsManager \_instance;

public static BallsManager Instance => \_instance;

private void Awake()

{

if (\_instance != null)

{

Destroy(gameObject);

}

else

{

\_instance = this;

}

}

#endregion

[SerializeField]

private Ball ballPrefab;

private Ball initialBall;

private Rigidbody2D initialBallRb;

public float initialBallSpeed = 250;

public List<Ball> Balls { get; set; }

private void Start()

{

InitBall();

}

private void Update()

{

if (!GameManager.Instance.IsGameStarted)

{

// Align ball position to the paddle position

Vector3 paddlePosition = Paddle.Instance.gameObject.transform.position;

Vector3 ballPosition = new Vector3(paddlePosition.x, paddlePosition.y + .27f, 0);

initialBall.transform.position = ballPosition;

if (Input.GetMouseButtonDown(0))

{

initialBallRb.isKinematic = false;

initialBallRb.AddForce(new Vector2(0, initialBallSpeed));

GameManager.Instance.IsGameStarted = true;

}

}

}

public void SpawnBalls(Vector3 position, int count, bool isLightningBall)

{

for (int i = 0; i < count; i++)

{

Ball spawnedBall = Instantiate(ballPrefab, position, Quaternion.identity) as Ball;

if (isLightningBall)

{

spawnedBall.StartLightningBall();

}

Rigidbody2D spawnedBallRb = spawnedBall.GetComponent<Rigidbody2D>();

spawnedBallRb.isKinematic = false;

spawnedBallRb.AddForce(new Vector2(0, initialBallSpeed));

this.Balls.Add(spawnedBall);

}

}

public void ResetBalls()

{

foreach (var ball in this.Balls.ToList())

{

Destroy(ball.gameObject);

}

InitBall();

}

private void InitBall()

{

Vector3 paddlePosition = Paddle.Instance.gameObject.transform.position;

Vector3 startingPosition = new Vector3(paddlePosition.x, paddlePosition.y + .27f, 0);

initialBall = Instantiate(ballPrefab, startingPosition, Quaternion.identity);

initialBallRb = initialBall.GetComponent<Rigidbody2D>();

this.Balls = new List<Ball>

{

initialBall

};

}

}

Листинг BricksManager.cs

using System;

using System.Collections.Generic;

using System.Linq;

using UnityEngine;

public class BricksManager : MonoBehaviour

{

#region Singleton

private static BricksManager \_instance;

public static BricksManager Instance => \_instance;

public static event Action OnLevelLoaded;

private void Awake()

{

if (\_instance != null)

{

Destroy(gameObject);

}

else

{

\_instance = this;

}

}

#endregion

private int maxRows = 17;

private int maxCols = 12;

private GameObject bricksContainer;

private float initialBrickSpawnPositionX = -1.96f;

private float initialBrickSpawnPositionY = 3.325f;

private float shiftAmount = 0.365f;

public Brick brickPrefab;

public Sprite[] Sprites;

public Color[] BrickColors;

public List<Brick> RemainingBricks { get; set; }

public List<int[,]> LevelsData { get; set; }

public int InitialBricksCount { get; set; }

public int CurrentLevel;

private void Start()

{

this.bricksContainer = new GameObject("BricksContainer");

this.LevelsData = this.LoadLevelsData();

this.GenerateBricks();

}

public void LoadNextLevel()

{

this.CurrentLevel++;

if (this.CurrentLevel >= this.LevelsData.Count)

{

GameManager.Instance.ShowVictoryScreen();

}

else

{

this.LoadLevel(this.CurrentLevel);

}

}

public void LoadLevel(int level)

{

this.CurrentLevel = level;

this.ClearRemainingBricks();

this.GenerateBricks();

}

private void ClearRemainingBricks()

{

foreach (Brick brick in this.RemainingBricks.ToList())

{

Destroy(brick.gameObject);

}

}

private void GenerateBricks()

{

this.RemainingBricks = new List<Brick>();

int[,] currentLevelData = this.LevelsData[this.CurrentLevel];

float currentSpawnX = initialBrickSpawnPositionX;

float currentSpawnY = initialBrickSpawnPositionY;

float zShift = 0;

for (int row = 0; row < this.maxRows; row++)

{

for (int col = 0; col < this.maxCols; col++)

{

int brickType = currentLevelData[row, col];

if (brickType > 0)

{

Brick newBrick = Instantiate(brickPrefab, new Vector3(currentSpawnX, currentSpawnY, 0.0f - zShift), Quaternion.identity) as Brick;

newBrick.Init(bricksContainer.transform, this.Sprites[brickType - 1], this.BrickColors[brickType], brickType);

this.RemainingBricks.Add(newBrick);

zShift += 0.0001f;

}

currentSpawnX += shiftAmount;

if (col + 1 == this.maxCols)

{

currentSpawnX = initialBrickSpawnPositionX;

}

}

currentSpawnY -= shiftAmount;

}

this.InitialBricksCount = this.RemainingBricks.Count;

OnLevelLoaded?.Invoke();

}

private List<int[,]> LoadLevelsData()

{

TextAsset text = Resources.Load("levels") as TextAsset;

string[] rows = text.text.Split(new string[] { Environment.NewLine }, StringSplitOptions.RemoveEmptyEntries);

List<int[,]> levelsData = new List<int[,]>();

int[,] currentLevel = new int[maxRows, maxCols];

int currentRow = 0;

for (int row = 0; row < rows.Length; row++)

{

string line = rows[row];

if (line.IndexOf("--") == -1)

{

string[] bricks = line.Split(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries);

for (int col = 0; col < bricks.Length; col++)

{

currentLevel[currentRow, col] = int.Parse(bricks[col]);

}

currentRow++;

}

else

{

// end of current level

// add the matrix to the last and continue the loop

currentRow = 0;

levelsData.Add(currentLevel);

currentLevel = new int[maxRows, maxCols];

}

}

return levelsData;

}

}

1. Генерация работает через специальную сетку, состоящую из 0, 1, 2, 3. Соответственно, где ноль, там отсутствует “Brick”. При единице, он появляется, имея 1 hitpoint. При двойке соответственно 2 и так далее.

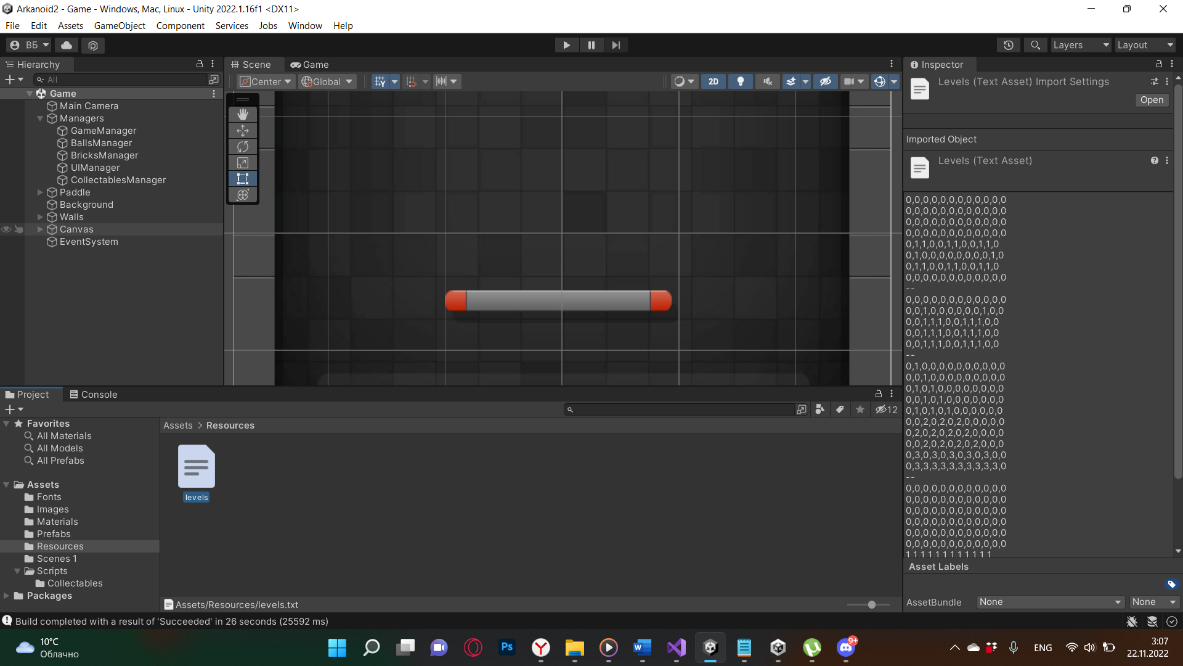


Рис. 31.5 Лист уровней

1. Данная сетка находится в папке Resources, позволяющей брать из неё данные. Например txt файла. Создал Canvas, дочерний GameOverPanel и GameOverText. Вставил шрифт из ассетов.

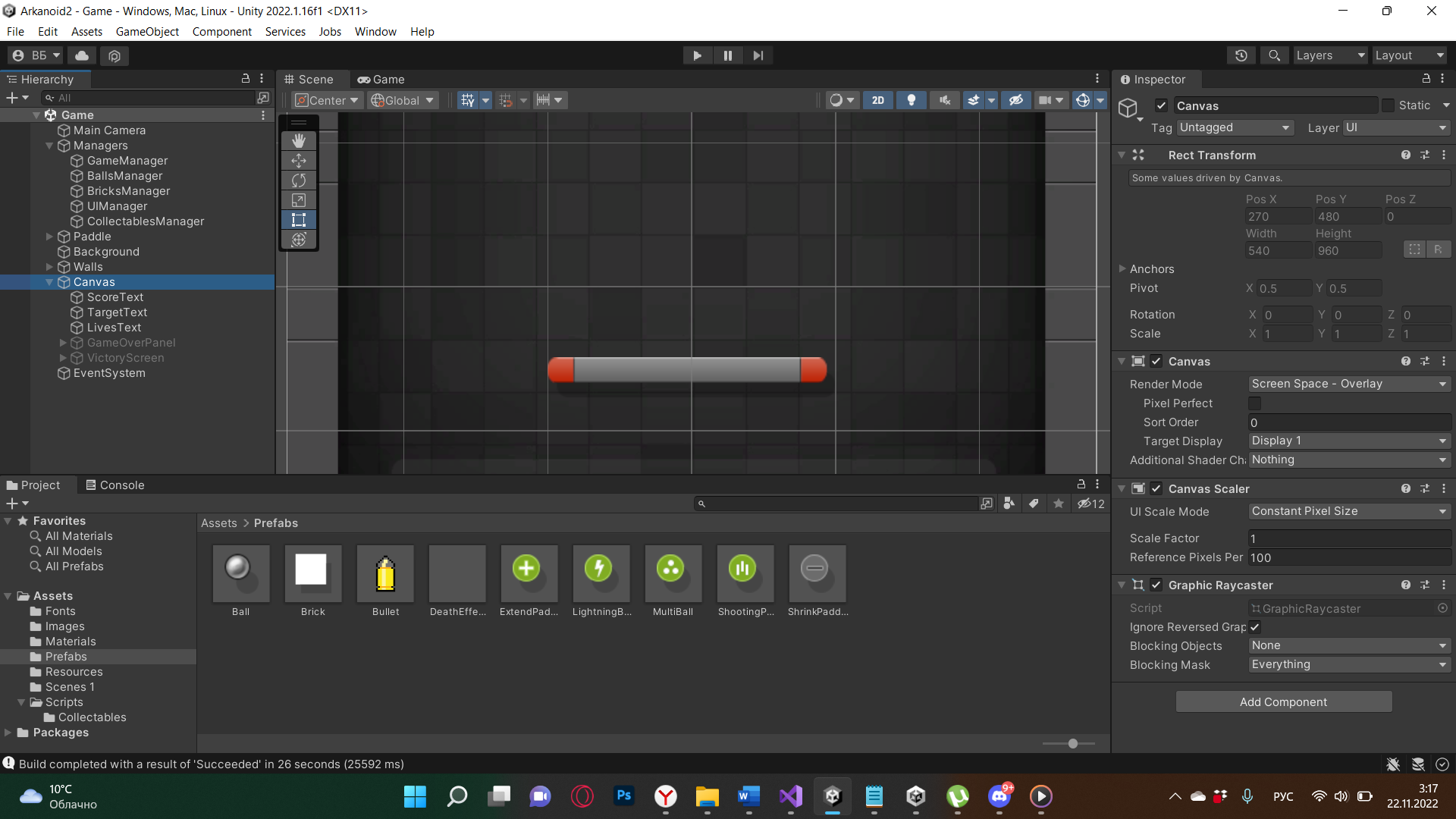


Рис. 31.6 Иерархия Canvas

1. Создал скрипт DeathWall, отвечающий за “смерть” объекта Ball.

Листинг DeathWall.cs

using UnityEngine;

public class DeathWall : MonoBehaviour

{

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.tag == "Ball")

{

Ball ball = collision.GetComponent<Ball>();

BallsManager.Instance.Balls.Remove(ball);

ball.Die();

}

}

}

1. В GameManager добавил логику смену уровней, траты жизней и экран “GameOver”

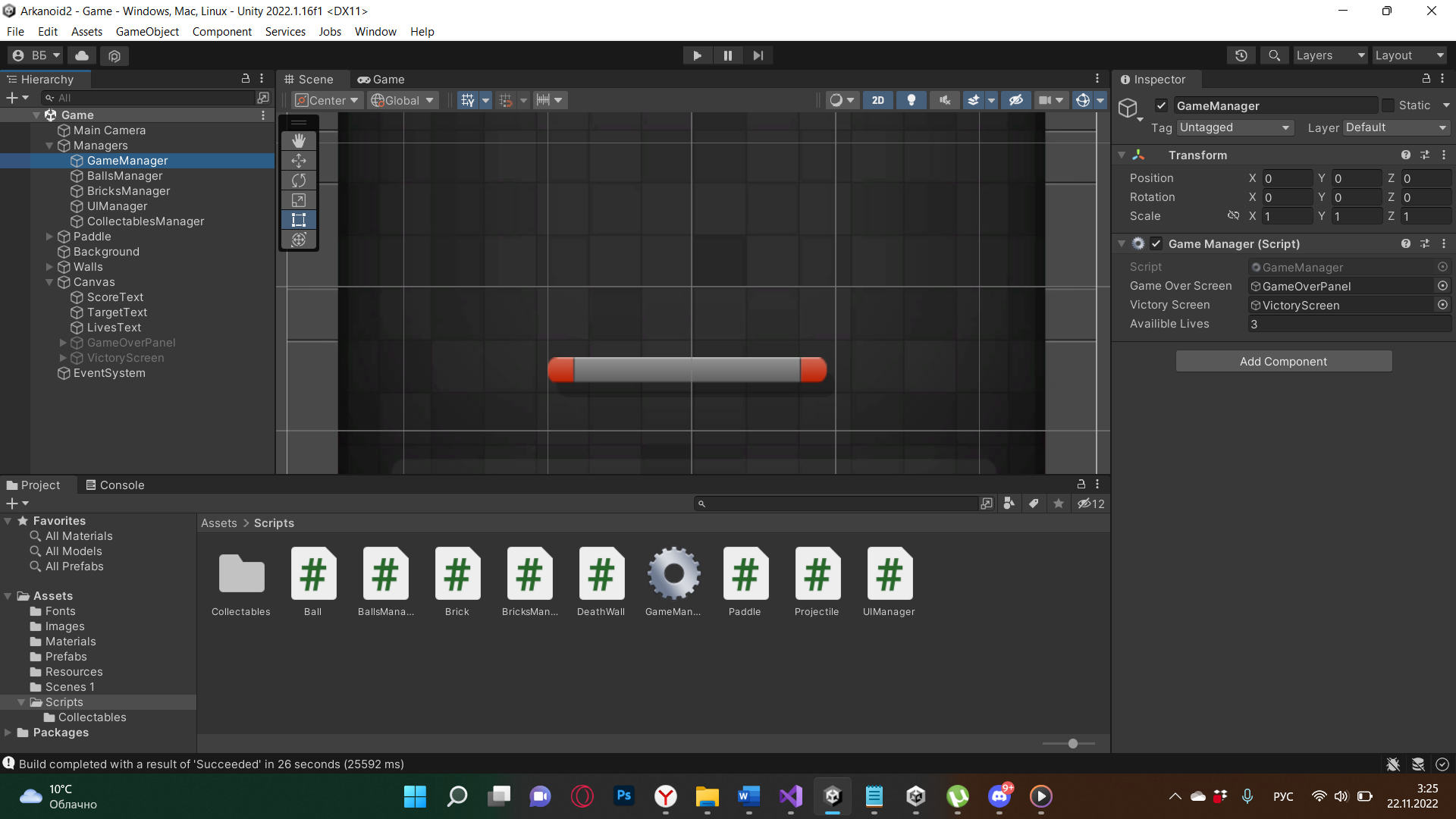


Рис. 31.7 Инспектор GameManager

1. Создал кнопку “Рестарта”. Создал Canvas, отвечающие за набор “очков” и скрипт UIManager

Листинг UIManager.cs

using System;

using UnityEngine;

using UnityEngine.UI;

public class UIManager : MonoBehaviour

{

public Text Target;

public Text ScoreText;

public Text LivesText;

public int Score { get; set; }

private void Awake()

{

Brick.OnBrickDestruction += OnBrickDestruction;

BricksManager.OnLevelLoaded += OnLevelLoaded;

GameManager.OnLiveLost += OnLiveLost;

}

private void Start()

{

OnLiveLost(GameManager.Instance.AvailibleLives);

}

private void OnLiveLost(int remainingLives)

{

LivesText.text = $"LIVES: {remainingLives}";

}

private void OnLevelLoaded()

{

UpdateRemainingBricksText();

UpdateScoreText(0);

}

private void UpdateScoreText(int increment)

{

this.Score += increment;

string scoreString = this.Score.ToString().PadLeft(5, '0');

ScoreText.text = $"SCORE:{Environment.NewLine}{scoreString}";

}

private void OnBrickDestruction(Brick obj)

{

UpdateRemainingBricksText();

UpdateScoreText(10);

}

private void UpdateRemainingBricksText()

{

Target.text = $"TARGET:{Environment.NewLine}{BricksManager.Instance.RemainingBricks.Count} / {BricksManager.Instance.InitialBricksCount}";

}

private void OnDisable()

{

Brick.OnBrickDestruction -= OnBrickDestruction;

BricksManager.OnLevelLoaded -= OnLevelLoaded;

GameManager.OnLiveLost -= OnLiveLost;

}

}

1. Создал “Collectables”

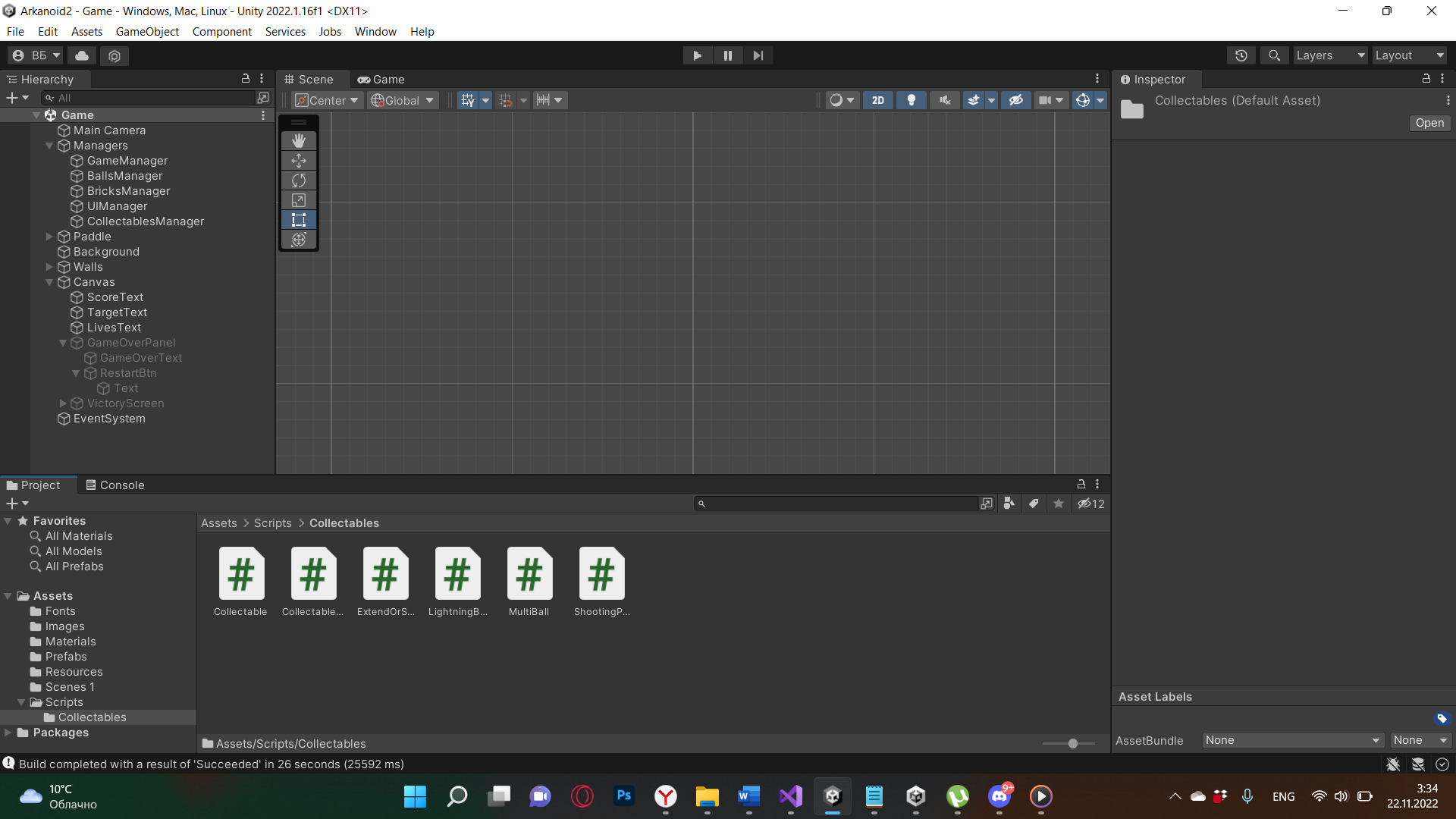


Рис. 31.8 Скрипты папки “Collectables”

Листинг Collectable.cs

using UnityEngine;

public abstract class Collectable : MonoBehaviour

{

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.tag == "Paddle")

{

this.ApplyEffect();

}

if (collision.tag == "Paddle" || collision.tag == "DeathWall")

{

Destroy(this.gameObject);

}

}

protected abstract void ApplyEffect();

}

1. Отвечает либо за использование эффекта при соприкосновении с пересечением коллайдера “Paddle”, либо за удаление объекта при пересечении “DeathWall”

Листинг MultiBall.cs

using System.Linq;

public class MultiBall : Collectable

{

protected override void ApplyEffect()

{

foreach (Ball ball in BallsManager.Instance.Balls.ToList())

{

BallsManager.Instance.SpawnBalls(ball.gameObject.transform.position, 2, ball.isLightningBall);

}

}

}

1. Создал CollectablesManager

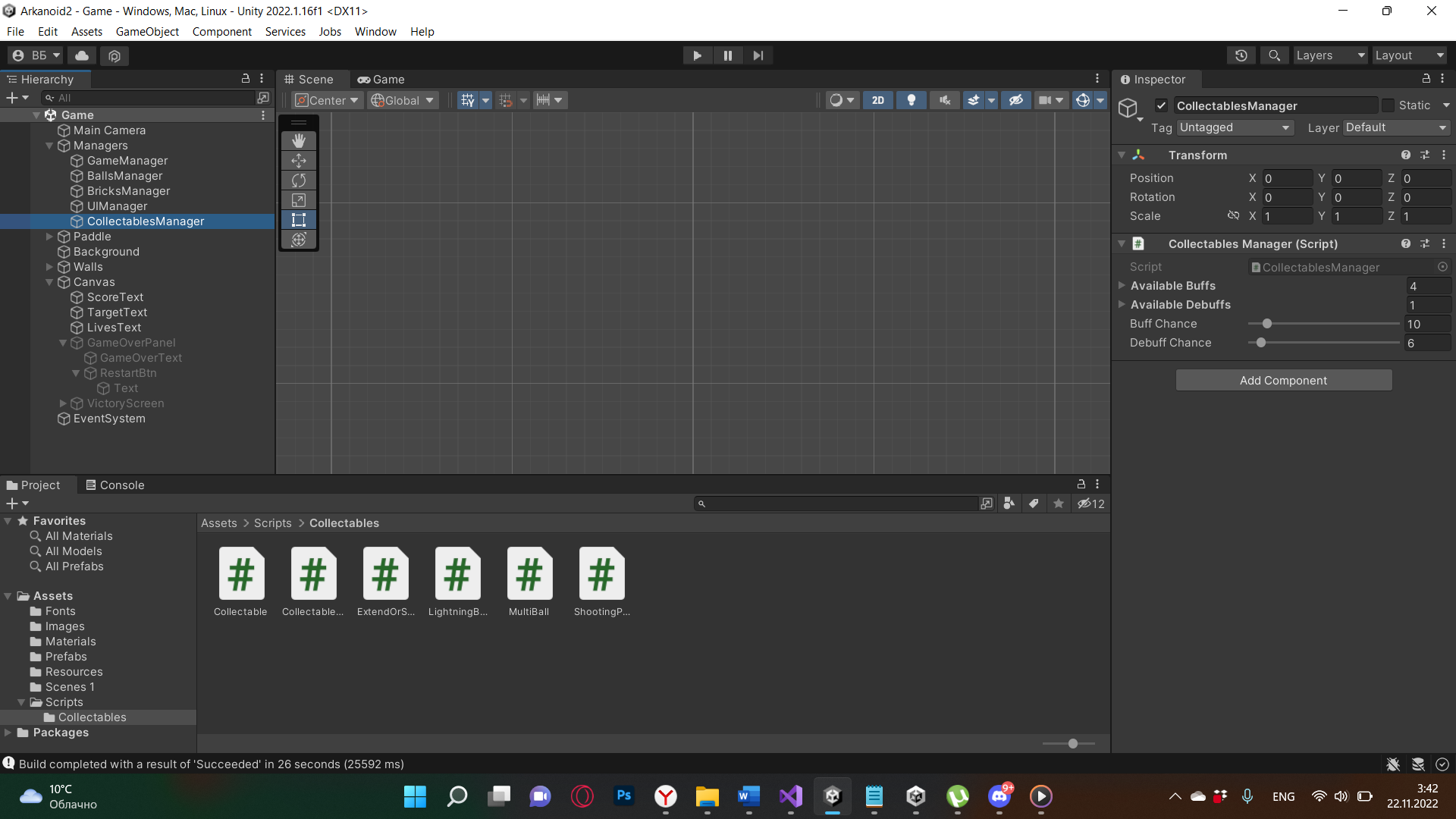


Рис. 31.9 Инспектор CollectablesManager

Листинг CollectablesManager

using System.Collections.Generic;

using UnityEngine;

public class CollectablesManager : MonoBehaviour

{

#region Singleton

private static CollectablesManager \_instance;

public static CollectablesManager Instance => \_instance;

private void Awake()

{

if (\_instance != null)

{

Destroy(gameObject);

}

else

{

\_instance = this;

}

}

#endregion

public List<Collectable> AvailableBuffs;

public List<Collectable> AvailableDebuffs;

[Range(0, 100)]

public float BuffChance;

[Range(0, 100)]

public float DebuffChance;

}

1. Создал ExtendOrShrink. Отвечает за увеличение или уменьшение “Paddle”.

Листинг ExtendOrShrink.cs

public class ExtendOrShrink : Collectable

{

public float NewWidth = 2.5f;

protected override void ApplyEffect()

{

if (Paddle.Instance != null && !Paddle.Instance.PaddleIsTransforming)

{

Paddle.Instance.StartWidthAnimation(NewWidth);

}

}

}

1. Создал “Lightning Ball”. Отвечает за усиление наносимых “Ball” hitpoints. При активации, так же воспроизводит Particle System.

Листинг LightningBall.cs

public class LightningBall : Collectable

{

protected override void ApplyEffect()

{

foreach (var ball in BallsManager.Instance.Balls)

{

ball.StartLightningBall();

}

}

}

1. Создал ShootingPaddle

Листинг ShootingPaddle.cs

public class ShootingPaddle : Collectable

{

protected override void ApplyEffect()

{

Paddle.Instance.StartShooting();

}

}

1. Cоздал билд проекта

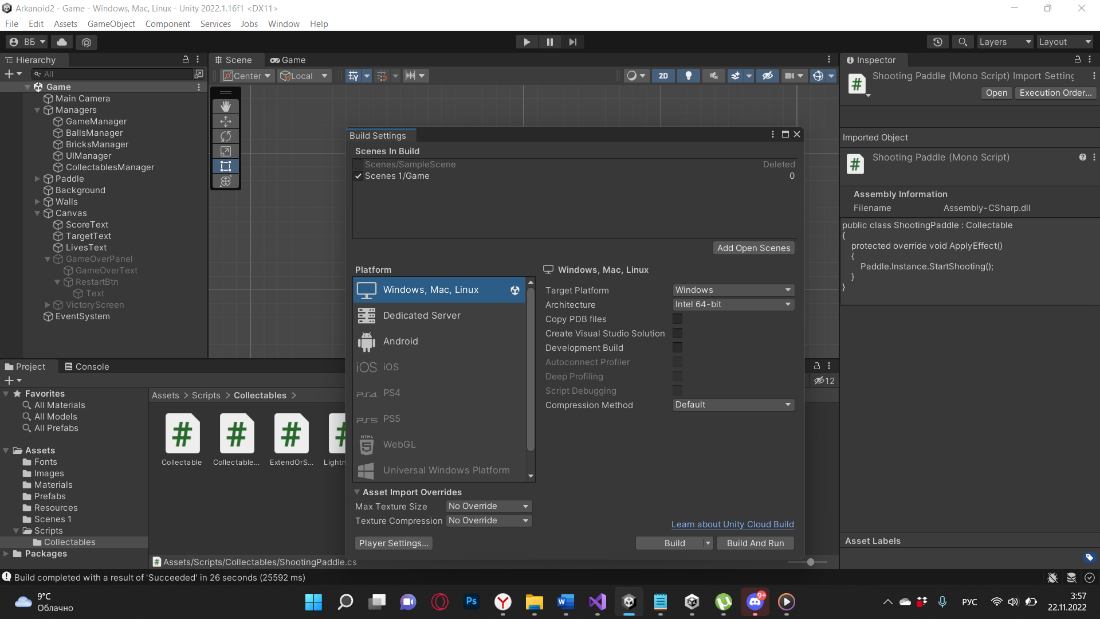


Рис. 31.10 Build Settings

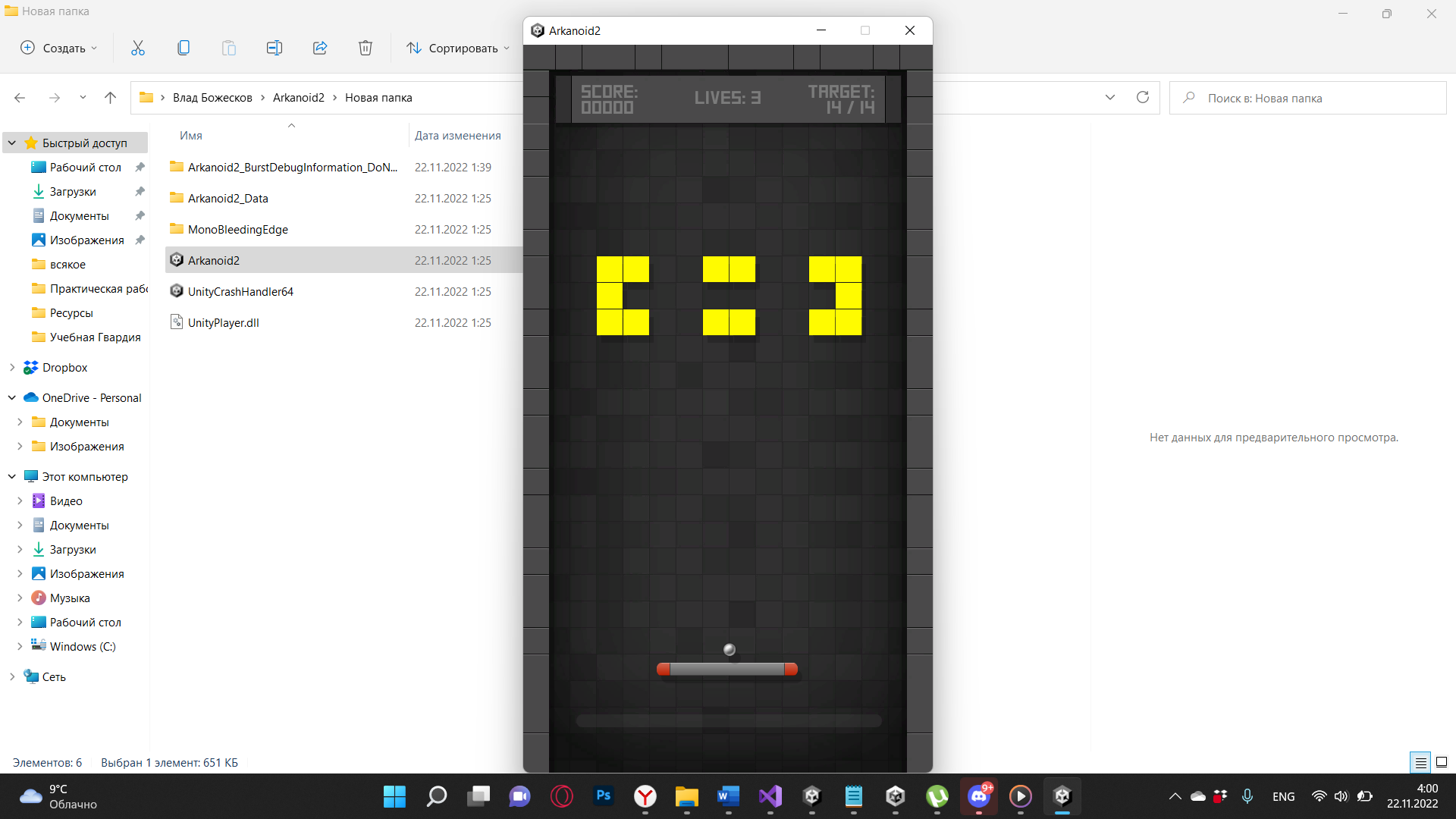


Рис. 31.11 Билд проекта.

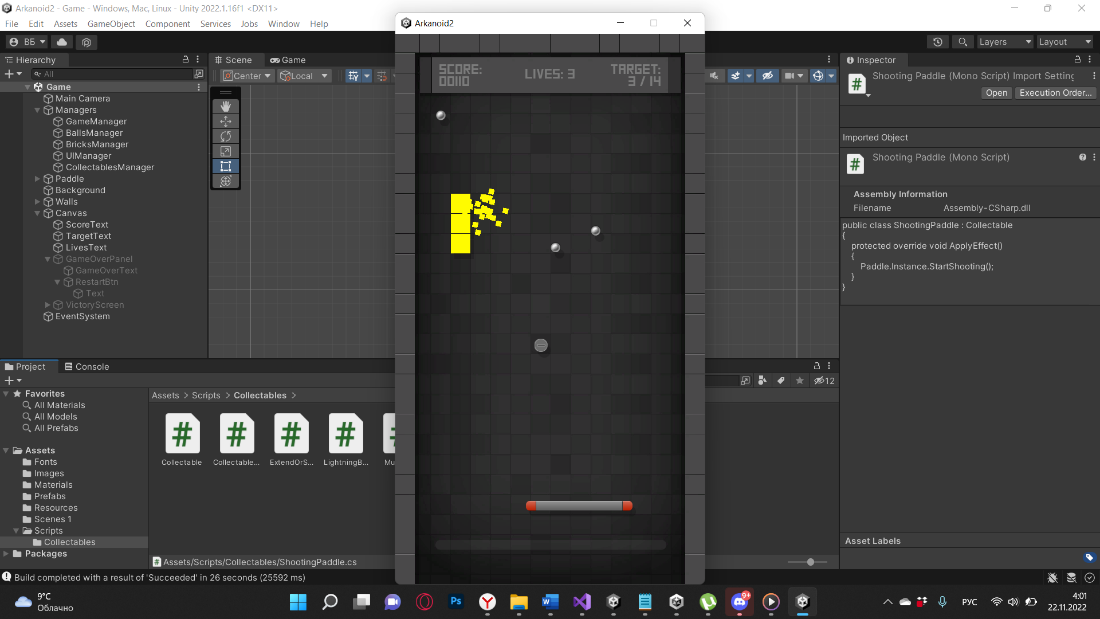


Рис. 31.12 Работа эффектов в проекте.

1. Вывод

В ходе проделанной работы был разработан игровой проект Arkanoid.