Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный университет»

Школа развития цифровых компетенций «Digital Up» (цифровая кафедра)

**ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ**

**на итоговый проект «Разработка**

**платформера «Escape»» по ДПП ПП «Основы Gamedev и VR-разработки»**

|  |  |  |  |
| --- | --- | --- | --- |
| п/п  № | Задание | Исполнитель | Рабочий график (план) выполнения |
| 1 | Провести исследование и анализ современных платформеров | Ваверко И.С | 02.05.2025 – 13.05.2025 |
| 2 | Разработать собственную концепию платформера, основываясь анализе | 14.05.2025 – 15.05.2025 |
| 3 | Разработать опытный образец платформера | 15.05.2025 – 30.05.2025 |
| 4 | Сделать выводы исходя из результатов работы | 30.05.2025 – 3.06.2025 |

Руководитель проекта   
преподаватель Осыкин Д.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / «\_\_\_ » \_\_\_\_\_\_\_\_\_ 2025 г.

(подпись)

СОГЛАСОВАНО:

Руководитель ДПП ПП

канд. физ.-мат. наук, доцент Козлов Д.Ю.

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2025 г.

(подпись)

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный университет»

Школа развития цифровых компетенций «Digital Up» (цифровая кафедра)

Отчет о выполнении группового итогового проекта по ДПП ПП

«Основы Gamedev и VR-разработки»

**«Разработка платформера “Escape”»**

Исполнители:

Ваверко И. С.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

Руководитель проекта

преподаватель Осыкин Д.А.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

г. Барнаул, 2025

# Цель проекта

Цель данного проекта создание динамичного 2D-платформера с акцентом на точное управление и сложные испытания.

# Задачи проекта и исполнители

Для достижения поставленной цели — создания динамичного 2D-платформера с управлением дроном — необходимо решить следующие задачи:

1. Провести исследование и анализ современных платформеров.
2. Разработать собственную концепию платформера, основываясь анализе.
3. Разработать опытный образец платформера.
4. Сделать выводы исходя из результатов работы.

# Актуальность и востребованность проекта

Актуальность данного проекта обусловлена стремительным ростом игровой индустрии, доступностью современных инструментов и растущим спросом на уникальные игровые проекты. Глобальный рынок видеоигр продолжает расширяться, достигая многомиллиардных оборотов, при этом цифровые платформы, такие как Steam, itch.io и Epic Games Store, значительно снизили барьеры для входа, позволяя небольшим студиям и одиночным разработчикам находить свою аудиторию. Технологический прогресс в виде мощных и бесплатных движков (Unity, Godot, Unreal Engine с роялти-моделью), а также обилие обучающих материалов дают возможность создавать качественные игры даже с ограниченным бюджетом. Кроме того, современные тренды, такие как пиксель-арт, нарративные инди-игры и экспериментальные механики, открывают пространство для творчества, ведь игроки всё чаще ценят оригинальность и душевность проектов больше, чем графику и масштабы ААА-тайтлов. Монетизация через краудфандинг (Kickstarter), ранний доступ или платформы подписки (Xbox Game Pass, PlayStation Plus) также предоставляет финансовые возможности для реализации своих идей. Важным фактором остаётся и комьюнити-поддержка: социальные сети, стриминговые платформы (Twitch, YouTube) и форумы позволяют напрямую общаться с аудиторией, получать фидбек и продвигать игру без огромных маркетинговых бюджетов. Таким образом, разработка даннго проекта остаётся перспективным направлением, где талант, креативность и умение работать с аудиторией могут привести к успеху даже в условиях ограниченных ресурсов.

# Общие сведения о проделанной работе

Этапы выполнения проекта:

* Исследование современных платформеров
  + Были проанализированы популярные игры жанра (Celeste, Super Meat Boy, Sunblaze) с точки зрения механик управления, уровня сложности и визуального стиля. Это помогло сформировать концепцию проекта, выделив ключевые особенности — точное управление и сложные, но справедливые испытания.
* Разработать опытный образец
  + На основе Unity был разработан прототип, включающий базовые механики
  + Для визуализации использовались ассеты, созданные в Inkscape, а звуковые эффекты были подобраны из Freesound.
* Испытать опытный образец
* Сделать выводы по проделанной работе

Для реализации проекта динамичного 2D-платформера был выбран следующий стек технологий:

* + В качестве основного игрового движка использовался Unity, поскольку он предоставляет мощные инструменты для разработки 2D-игр, включая физику, анимацию, систему коллизий и удобный редактор сцен. Unity поддерживает кроссплатформенную разработку, что позволяет в будущем выпустить игру на ПК, мобильные устройства и консоли. Кроме того, в Unity есть встроенная поддержка C# — языка, который хорошо подходит для создания игровой логики и сложных механик.
  + Для создания векторной графики и ассетов использовался Inkscape — бесплатный и мощный редактор, позволяющий разрабатывать векторный-арт: спрайты, фоны и интерфейсы. Его преимущество — гибкость в работе с форматами SVG и PNG, что важно для оптимизации игровых ресурсов.
  + Звуковое оформление игры было собрано с использованием ресурсов Freesound — открытой базы звуковых эффектов и музыки. Это позволило найти качественные аудиофайлы для различных игровых событий без необходимости записи с нуля.

Основная идея Escape строится вокруг мрачной, загадочной атмосферы заброшенного комплекса, где игроку не дают прямых ответов. Кто он? Почему оказался здесь? Что произошло с этим местом? Вместо явных объяснений игра оставляет пространство для размышлений, усиливая ощущение тревоги и неизвестности. Саундтрек, выполненный в жанрах future garage и industrial, дополняет гнетущую обстановку: эмбиентные шумы, редкие музыкальные фрагменты и механические звуки создают эффект погружения в пустую, но небезопасную среду.

Игрок управляет не самим персонажем, а дроном, через чей интерфейс и ведётся наблюдение. Камера демонстрирует мир так, будто это система видеонаблюдения, что сокращает дистанцию между игроком и происходящим, добавляя реализма. Такой подход также усиливает чувство изоляции — герой один в чужом, враждебном пространстве, а игрок лишь косвенно влияет на его судьбу.

Escape сочетает классический платформер с видом сбоку с нестандартными элементами. Ключевые механики включают:

* Головоломки с отражением лучей — некоторые поверхности перенаправляют лучи, что требует логики и точности.
* Движущиеся платформы и ловушки — уровень сложности растёт за счёт необходимости идеального расчёта времени.
* Ограниченный запас энергии — полоска энергии постоянно уменьшается, вынуждая игрока действовать быстро и добавляя напряжённости.

Эти элементы создают баланс между исследованием, решением задач и выживанием, удерживая игрока в состоянии постоянного напряжения. Для воплощения этой концепции были разработаны модульные и гибкие компоненты, подходящие для решения разнообразных задач.

# Результаты проекта

Результатом проекта стал функциональный прототип игры, который наглядно демонстрирует все задуманные ключевые особенности: мрачную атмосферу, геймплейные механики, звуковое оформление и нестандартный подход к повествованию. Этот прототип позволяет оценить как сильные стороны концепта, такие как инновационное управление через дрон и напряжённый игровой процесс, так и выявить слабые места — баланс сложности, удобство управления или необходимость дополнительных визуальных подсказок.

При этом прототип построен с расчётом на масштабирование — его архитектура позволяет легко добавлять новые уровни, механики и контент, что делает его готовым к дальнейшей доработке и полноценному релизу. Таким образом, проект уже сейчас представляет собой устойчивую основу для будущей игры, сохраняя потенциал для развития и финальной полировки перед выходом. 

Рисунок 1.

**Приложение 1.**Программный код

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerModel : MonoBehaviour

{

[Header("Movement Settings")]

[SerializeField] private float moveForce = 10f;

[SerializeField] private float jumpForce = 7f;

[SerializeField] private float rotationSpeed = 5f;

[SerializeField] private float maxSpeed = 5f;

[Header("Ground Check")]

[SerializeField] private LayerMask groundLayer;

private Rigidbody2D rb;

private bool isGrounded;

private Collider2D ballCollider;

private void Awake()

{

rb = GetComponent<Rigidbody2D>();

ballCollider = GetComponent<Collider2D>();

}

private void FixedUpdate()

{

if (rb.velocity.magnitude > maxSpeed)

{

rb.velocity = rb.velocity.normalized \* maxSpeed;

}

}

private void OnCollisionStay2D(Collision2D collision)

{

if (((1 << collision.gameObject.layer) & groundLayer) != 0)

{

isGrounded = true;

}

}

private void OnCollisionExit2D(Collision2D collision)

{

if (((1 << collision.gameObject.layer) & groundLayer) != 0)

{

isGrounded = false;

}

}

public void Move(float direction)

{

if (Mathf.Abs(direction) > 0.1f)

{

rb.AddForce(new Vector2(direction \* moveForce, 0), ForceMode2D.Impulse);

}

}

public void Jump()

{

if (isGrounded)

{

rb.AddForce(new Vector2(0, jumpForce), ForceMode2D.Impulse);

}

}

public bool IsGrounded()

{

return isGrounded;

}

public void Rotate(float Direction)

{

transform.Rotate(Vector3.forward, rotationSpeed \* Time.deltaTime\*Direction);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerController : MonoBehaviour

{

private PlayerModel ballModel;

private float moveInput;

private void Awake()

{

ballModel = GetComponent<PlayerModel>();

}

private void Update()

{

moveInput = Input.GetAxis("Horizontal");

if (Input.GetButtonDown("Jump"))

{

ballModel.Jump();

}

}

private void FixedUpdate()

{

ballModel.Move(moveInput);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class WaypointFollower : MonoBehaviour

{

public enum MovementType

{

Loop,

PingPong,

Once

}

[Header("Waypoints")]

public List<Vector2> waypoints = new List<Vector2>();

public MovementType movementType = MovementType.Loop;

public float movementSpeed = 3f;

public float reachThreshold = 0.1f;

[Header("Rotation Settings")]

public bool rotateToDirection = true;

public float rotationSpeed = 5f;

private int currentWaypointIndex = 0;

private bool movingForward = true;

private Rigidbody2D rb;

private void Awake()

{

rb = GetComponent<Rigidbody2D>();

if (waypoints.Count == 0)

{

waypoints.Add(transform.position);

}

}

private void FixedUpdate()

{

if (waypoints.Count <= 1) return;

Vector2 targetPosition = waypoints[currentWaypointIndex];

Vector2 moveDirection = (targetPosition - (Vector2)transform.position).normalized;

rb.velocity = moveDirection \* movementSpeed;

if (rotateToDirection && rb.velocity.magnitude > 0.1f)

{

float angle = Mathf.Atan2(moveDirection.y, moveDirection.x) \* Mathf.Rad2Deg;

Quaternion targetRotation = Quaternion.AngleAxis(angle, Vector3.forward);

transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, rotationSpeed \* Time.deltaTime);

}

if (Vector2.Distance(transform.position, targetPosition) < reachThreshold)

{

GetNextWaypoint();

}

}

private void GetNextWaypoint()

{

switch (movementType)

{

case MovementType.Loop:

currentWaypointIndex = (currentWaypointIndex + 1) % waypoints.Count;

break;

case MovementType.PingPong:

if (movingForward)

{

if (currentWaypointIndex >= waypoints.Count - 1)

{

movingForward = false;

currentWaypointIndex--;

}

else

{

currentWaypointIndex++;

}

}

else

{

if (currentWaypointIndex <= 0)

{

movingForward = true;

currentWaypointIndex++;

}

else

{

currentWaypointIndex--;

}

}

break;

case MovementType.Once:

if (currentWaypointIndex < waypoints.Count - 1)

{

currentWaypointIndex++;

}

else

{

rb.velocity = Vector2.zero;

enabled = false;

}

break;

}

}

private void OnDrawGizmosSelected()

{

if (waypoints.Count > 0)

{

Gizmos.color = Color.cyan;

for (int i = 0; i < waypoints.Count; i++)

{

Gizmos.DrawSphere(waypoints[i], 0.2f);

if (i < waypoints.Count - 1)

{

Gizmos.DrawLine(waypoints[i], waypoints[i + 1]);

}

else if (movementType == MovementType.Loop)

{

Gizmos.DrawLine(waypoints[i], waypoints[0]);

}

}

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.Events;

public class ButtonTrigger : MonoBehaviour

{

[SerializeField] GameObject TriggerCollider;

public UnityEvent onButtonTiggerEntered;

public void Invoke()

{

onButtonTiggerEntered?.Invoke();

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Destructable : MonoBehaviour

{

[SerializeField] GameObject EventBusObject;

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject.GetComponent<Destroyer>()!=null)

{

EventBusObject.GetComponent<EventBus>().onPlayerDeath?.Invoke();

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class FollowerCamera : MonoBehaviour

{

[Header("Target Settings")]

[SerializeField] private Transform target;

[Header("Follow Settings")]

[SerializeField] private Vector2 followSpeeds = new Vector2(5f, 3f);

[SerializeField] private float smoothTime = 0.3f;

[Header("Look Ahead Settings")]

[SerializeField] private bool lookAhead = true;

[SerializeField] private float lookAheadDistance = 2f;

[SerializeField] private float lookAheadSpeed = 1f;

[Header("Camera Bounds")]

[SerializeField] private bool useBounds = false;

[SerializeField] private Vector2 minBounds;

[SerializeField] private Vector2 maxBounds;

private Vector3 targetPosition;

private Vector3 lookAheadOffset;

private Vector3 velocity = Vector3.zero;

private Rigidbody2D targetRb;

private bool isSetingField = false;

private float setField;

private void Start()

{

if (target != null)

{

targetRb = target.GetComponent<Rigidbody2D>();

}

if (target != null)

{

transform.position = new Vector3(

target.position.x,

target.position.y,

transform.position.z

);

}

}

private void Update()

{

if (isSetingField)

{

Camera.main.fieldOfView = Mathf.Lerp(Camera.main.fieldOfView, setField, Time.deltaTime \* 0.5f);

}

if (target == null) return;

targetPosition = target.position;

targetPosition.z = transform.position.z;

if (lookAhead && targetRb != null)

{

Vector2 currentVelocity = targetRb.velocity;

if (currentVelocity.magnitude > 0.1f)

{

Vector2 targetLookAhead = currentVelocity.normalized \* lookAheadDistance;

lookAheadOffset = Vector3.Lerp(

lookAheadOffset,

targetLookAhead,

lookAheadSpeed \* Time.deltaTime

);

}

else

{

lookAheadOffset = Vector3.Lerp(

lookAheadOffset,

Vector3.zero,

lookAheadSpeed \* Time.deltaTime

);

}

targetPosition += lookAheadOffset;

}

if (useBounds)

{

targetPosition.x = Mathf.Clamp(targetPosition.x, minBounds.x, maxBounds.x);

targetPosition.y = Mathf.Clamp(targetPosition.y, minBounds.y, maxBounds.y);

}

float posX = Mathf.SmoothDamp(

transform.position.x,

targetPosition.x,

ref velocity.x,

smoothTime / followSpeeds.x

);

float posY = Mathf.SmoothDamp(

transform.position.y,

targetPosition.y,

ref velocity.y,

smoothTime / followSpeeds.y

);

transform.position = new Vector3(posX, posY, transform.position.z);

}

public void SetBounds(Vector2 min, Vector2 max)

{

minBounds = min;

maxBounds = max;

useBounds = true;

}

public void DisableBounds()

{

useBounds = false;

}

private void OnDrawGizmosSelected()

{

if (useBounds)

{

Gizmos.color = Color.green;

Vector3 center = new Vector3(

(minBounds.x + maxBounds.x) \* 0.5f,

(minBounds.y + maxBounds.y) \* 0.5f,

0

);

Vector3 size = new Vector3(

maxBounds.x - minBounds.x,

maxBounds.y - minBounds.y,

1

);

Gizmos.DrawWireCube(center, size);

}

}

public void setCameraFieldOfView(float field)

{

isSetingField = true;

setField = field;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.Events;

public class Lasertrigger : MonoBehaviour

{

public int countUse = 1;

private int currentUse = 0;

private float Timer = 0;

private bool inLaser = false;

public UnityEvent OnLaserEntered;

public void LaserEntered()

{

inLaser = true;

if (currentUse <= countUse)

{

OnLaserEntered?.Invoke();

currentUse++;

}

}

private void Update()

{

Timer += Time.deltaTime;

Timer = Timer % 2;

if(inLaser==false && Timer < 1)

{

currentUse = 0;

}

inLaser = false;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class LaserController : MonoBehaviour

{

[SerializeField] Laser laser;

[SerializeField] Rigidbody2D playerBody;

[SerializeField] PlayerController playerControll;

[SerializeField] PlayerModel player;

private float moveInput;

private void OnEnable()

{

laser.enabled = true;

playerBody.angularVelocity = 0;

playerBody.velocity = Vector3.zero;

playerBody.isKinematic = true;

playerControll.enabled = false;

}

private void OnDisable()

{

laser.enabled = false;

playerBody.isKinematic = false;

playerControll.enabled = true;

}

private void Update()

{

moveInput = Input.GetAxis("Horizontal");

if (Input.GetButtonDown("Jump"))

{

this.enabled = false;

}

}

private void FixedUpdate()

{

player.Rotate(moveInput);

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using TMPro;

public class TextMassage : MonoBehaviour

{

[SerializeField] private float typingSpeed = 0.05f;

[SerializeField] private TextMeshProUGUI textComponent;

[SerializeField] private string newText;

private string fullText;

private void Start()

{

fullText = textComponent.text;

textComponent.text = "";

StartCoroutine(TypeText());

}

private IEnumerator TypeText()

{

foreach (char letter in fullText.ToCharArray())

{

textComponent.text += letter;

yield return new WaitForSeconds(typingSpeed);

}

}

public void StartTyping()

{

fullText = newText;

textComponent.text = "";

StopAllCoroutines();

StartCoroutine(TypeText());

}

private void Awake()

{

StartTyping();

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class GloabalData : MonoBehaviour

{

[SerializeField] private float \_maxTime = 2000f;

[SerializeField] private float \_currentTime = 0f;

[SerializeField] private TextMeshProUGUI \_timeText;

[SerializeField] private TextMeshProUGUI \_levelText;

[SerializeField] private TextMeshProUGUI \_end;

[SerializeField] private Slider \_energySlider;

[Header("Settings")]

[SerializeField] private float \_energyDrainSpeed = 1f;

[SerializeField] private bool \_countDown = true;

private static GameObject instance;

private void Awake()

{

if (instance == null)

{

instance = this.gameObject;

DontDestroyOnLoad(gameObject);

}

else

{

Destroy(gameObject);

}

}

private void Update()

{

\_currentTime += Time.deltaTime;

UpdateTimeDisplay();

UpdateEnergy();

if (\_countDown && \_currentTime >= \_maxTime)

{

TimeExpired();

}

}

private void UpdateTimeDisplay()

{

int minutes = Mathf.FloorToInt(\_currentTime / 60f);

int seconds = Mathf.FloorToInt(\_currentTime % 60f);

int milliseconds = Mathf.FloorToInt((\_currentTime \* 1000f) % 1000f);

\_timeText.text = string.Format("{0:00}:{1:00}.{2:000}", minutes, seconds, milliseconds);

}

private void UpdateEnergy()

{

float energyPercentage = 1 - (\_currentTime / \_maxTime);

\_energySlider.value = Mathf.Clamp(energyPercentage, 0f, 1f);

}

private void ShowEnd()

{

\_end.enabled = true;

}

private void TimeExpired()

{

Debug.Log("Время вышло!");

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Laser : MonoBehaviour

{

[SerializeField] LineRenderer mainLine;

[SerializeField] int reflections;

[SerializeField] float maxRayDistance;

[SerializeField] LayerMask layerDetection;

private void Start()

{

Physics2D.queriesStartInColliders = false;

}

private void OnDisable()

{

mainLine.enabled = false;

}

private void OnEnable()

{

mainLine.enabled = true;

}

private void Update()

{

mainLine.positionCount = 1;

mainLine.SetPosition(0, transform.position);

RaycastHit2D hitInfo = Physics2D.Raycast(transform.position, transform.up, maxRayDistance, layerDetection);

bool isMirror = false;

Vector2 mirrorHitPoint = Vector2.zero;

Vector2 mirrorHitNormal = Vector2.zero;

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

{

mainLine.positionCount += 1;

if (hitInfo.collider != null)

{

mainLine.SetPosition(mainLine.positionCount - 1, hitInfo.point);

isMirror = false;

var LaserTrigg = hitInfo.collider.gameObject.GetComponent<Lasertrigger>();

if (LaserTrigg != null)

{

LaserTrigg.LaserEntered();

}

if (hitInfo.collider.CompareTag("Mirror"))

{

mirrorHitPoint = (Vector2)hitInfo.point;

mirrorHitNormal = (Vector2)hitInfo.normal;

hitInfo = Physics2D.Raycast(mirrorHitPoint, Vector2.Reflect(mirrorHitPoint, mirrorHitNormal), maxRayDistance, layerDetection);

isMirror = true;

}

else

break;

}

else

{

if (isMirror)

{

mainLine.SetPosition(mainLine.positionCount - 1, mirrorHitPoint + Vector2.Reflect(mirrorHitPoint, mirrorHitNormal) \* maxRayDistance);

break;

}

else

{

mainLine.SetPosition(mainLine.positionCount - 1, transform.position + transform.up \* maxRayDistance);

}

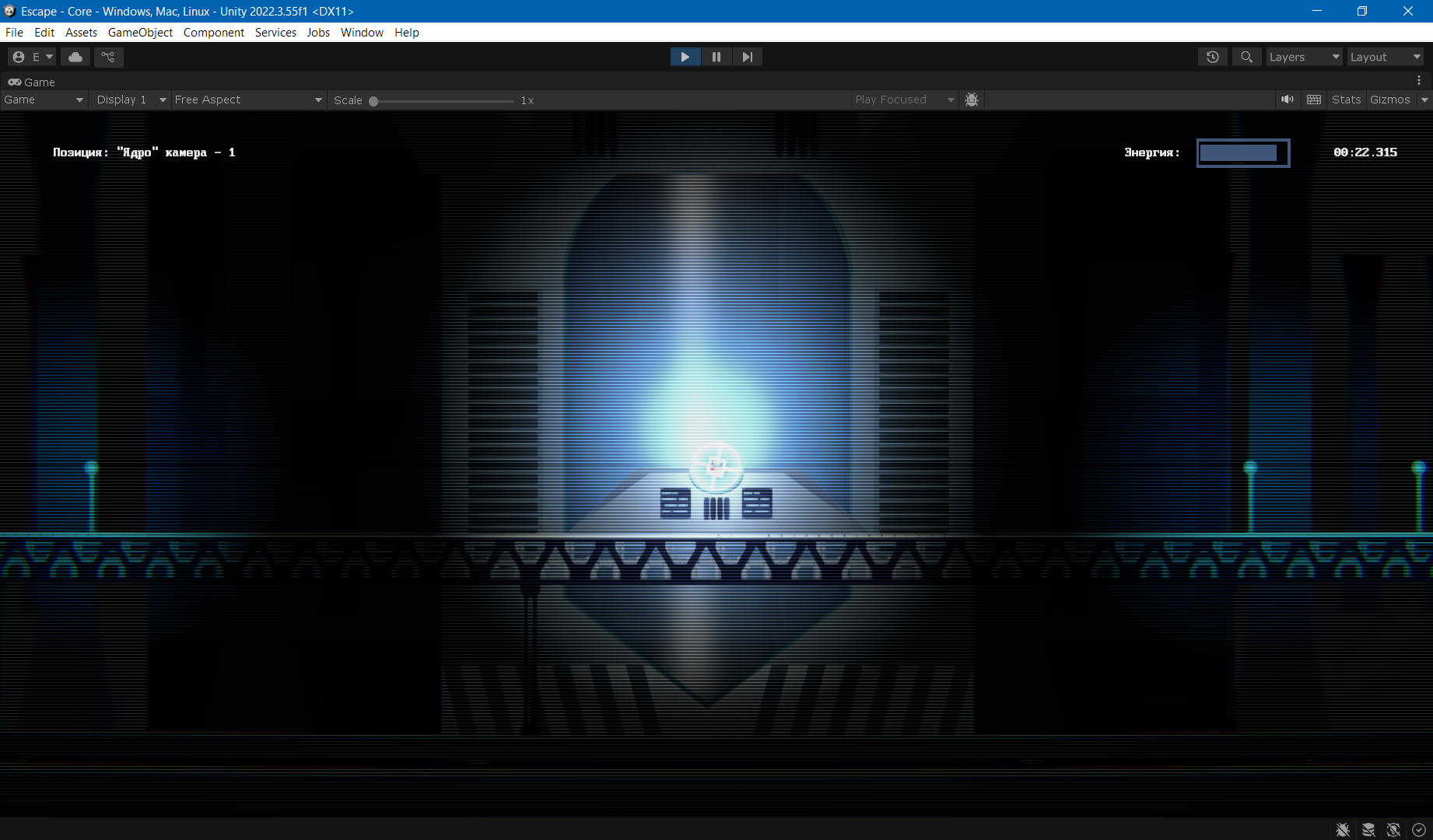
}

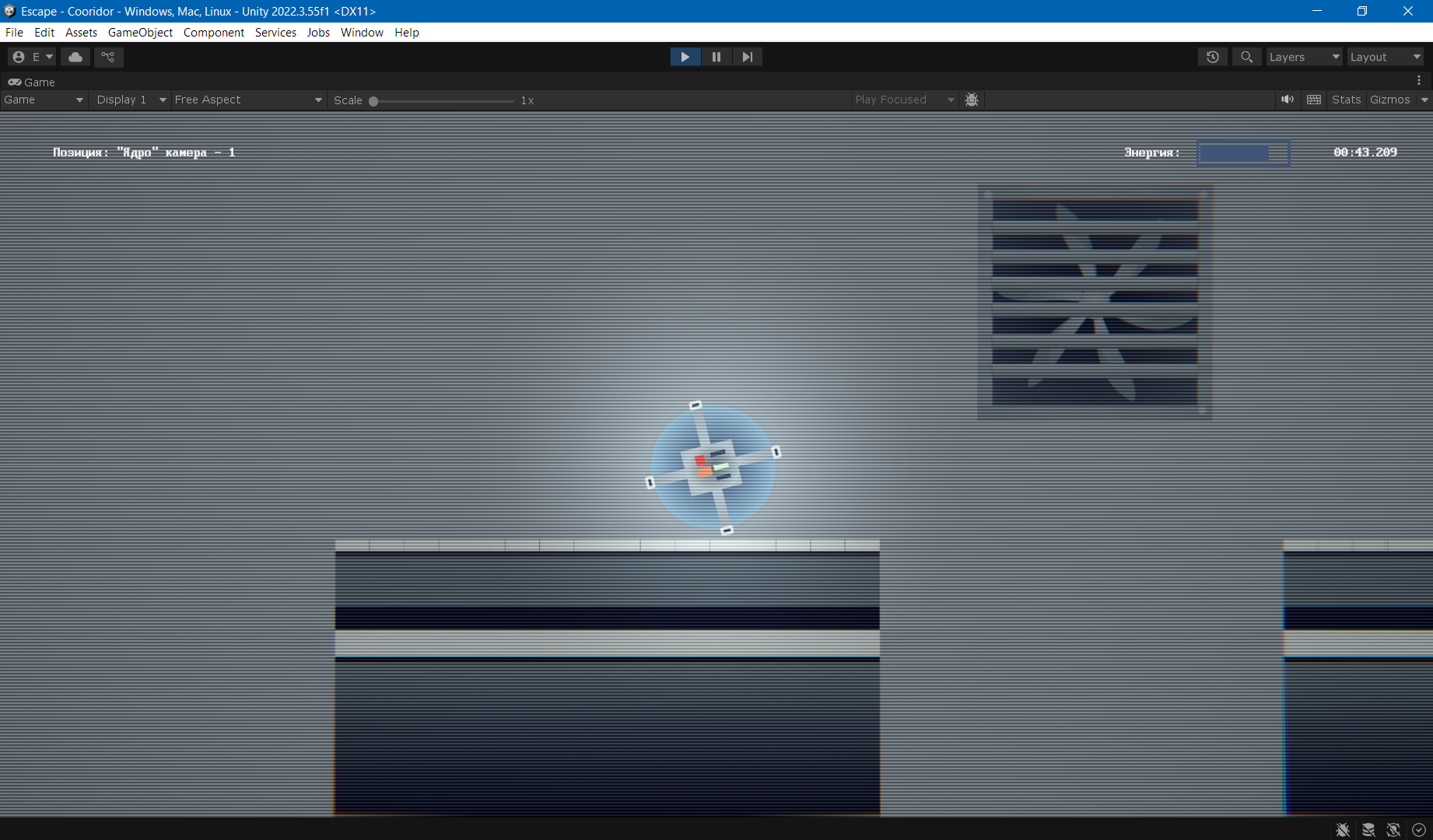
}

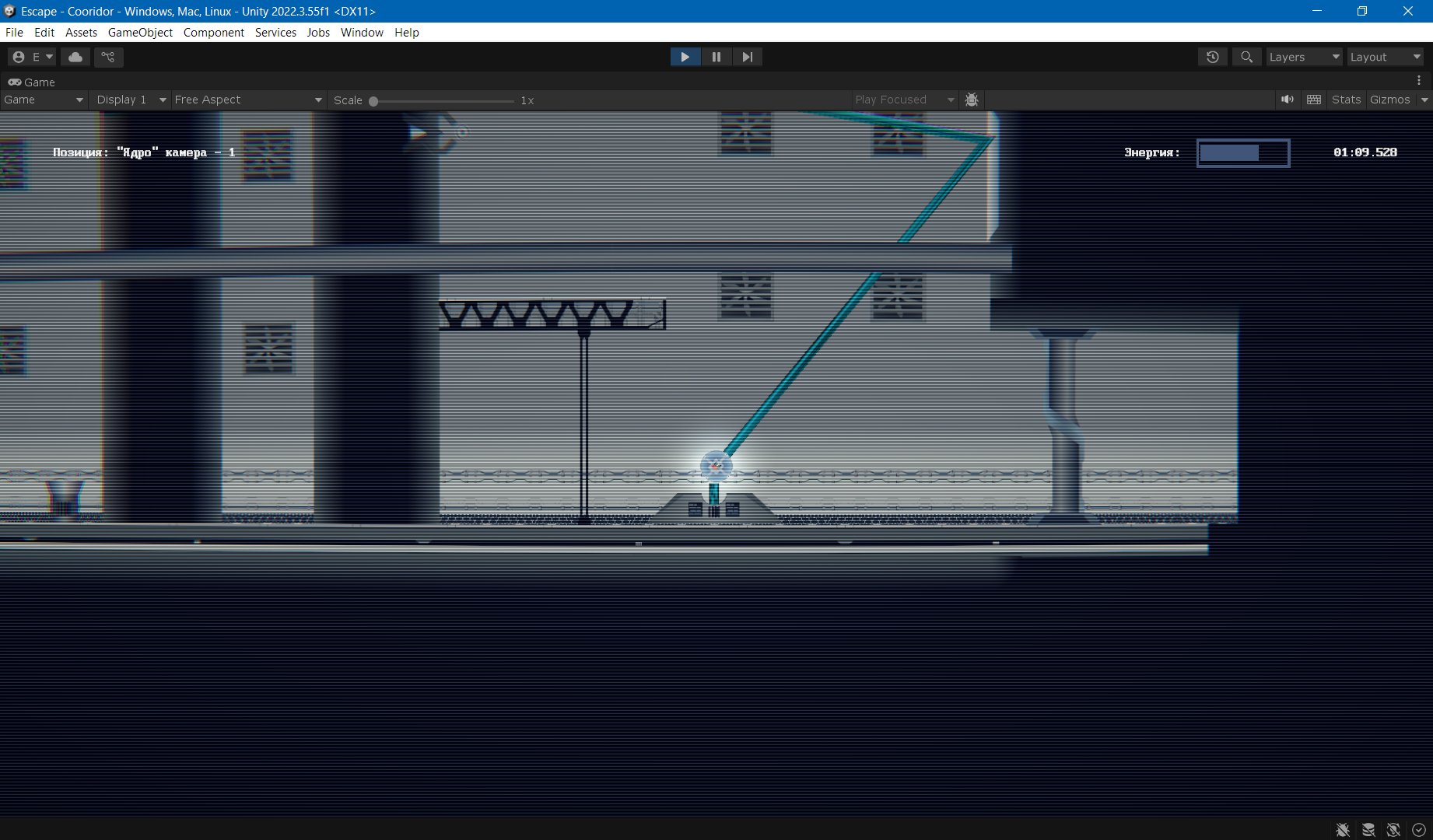
}

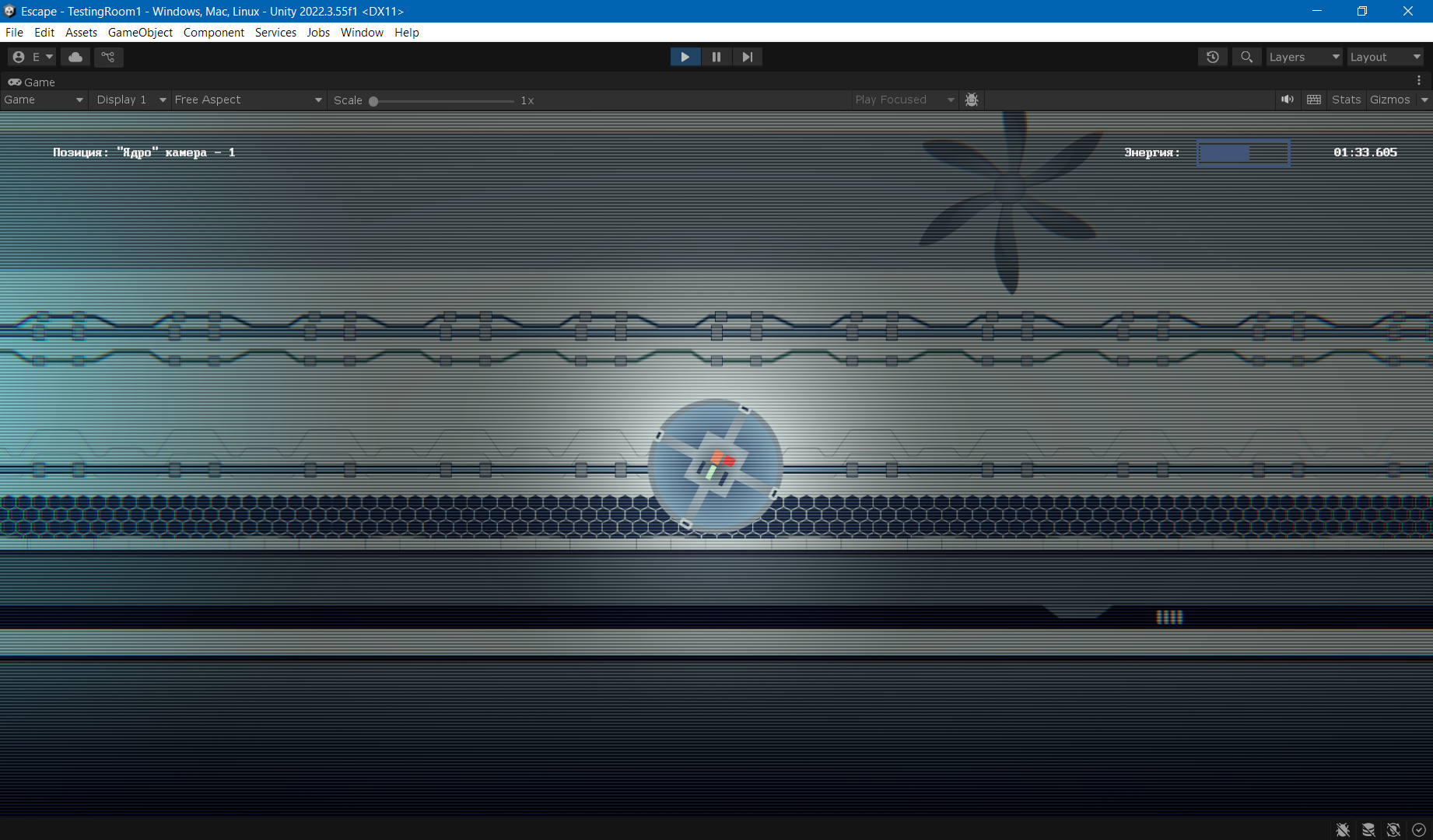
}

**Приложение 2.**









**Приложение 3.**

Ссылка на презентацию: https://github.com/Ersoil/Escape/tree/main/Report

Ссылка на видеоотчёт: https://1drv.ms/v/c/1fd28d83461ec201/EZW7Lzj1IVxIuVCdc20-2sAB9NDtv8IYlc9-7d8SrPtqiQ?e=amwsCL