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 игры в жанре «Головоломка» Big Puzzle Maze»**

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

|  |  |  |  |
| --- | --- | --- | --- |
| п/п  № | Задание | Исполнитель | Рабочий график (план) выполнения |
| 1 | Разработать главную цель игры и продумать механики геймплея | Корсун С.М | 01.05.2025-02.05.2025 |
| 2 | Создать уровни и проработать их наполнение, написать скрипты для механик геймплея. | Корсун С.М | 03.05.2025-10.06.2025 |
| 3 | Подготовка отчета и видео-  презентации | Корсун С.М | 11.06.2025-12.06.2025 |

Руководитель проекта   
Шостак Е. В.

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

(подпись)

СОГЛАСОВАНО:

Руководитель ДПП ПП

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

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2024 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 игры в жанре «Головоломка» Big Puzzle Maze»**

Исполнители:

Корсун С. М.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

Руководитель проекта

Шостак Е. В.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 г.

г. Барнаул, 2025

# Цель проекта

Цель данного проекта разработка игры в жанре «Головоломка» Big Puzzle Maze

# Задачи проекта и исполнители

Для достижения поставленной цели требовалось решить ряд задач:

1. Разработать главную цель игры и продумать механики геймплея.
2. Создать уровни и проработать их наполнение, написать скрипты для механик геймплея.
3. Подготовка отчёта и видео-презентации.

# Актуальность и востребованность проекта

Актуальность данного проекта достаточно высока, так как гейминг в наше время активно продвигается во все сферы жизни, например: образование, виртуальная реальность и другие отрасли. Согласно совместному исследованию Организации развития видеоигровой индустрии и Аналитического центра НАФИ, 60% россиян — около 88 млн человек — регулярно или эпизодически играют в видеоигры. С 2018 по 2022 год число геймеров в стране увеличилось более чем в три раза. Чаще всего видеоиграми увлекаются молодые люди до 24 лет. Доля мужчин в составе российских геймеров выше (54%), чем женщин (46%). Востребованность проекта также высока, потому что многие люди стремятся развивать свои интеллектуальные способности. Игры в жанре головоломок способствуют улучшению навыков решения проблем, критического мышления и внимания. Современные пользователи всё чаще ищут способы расслабиться и отвлечься от повседневной суеты. Игры, которые требуют концентрации и обдуманных решений, помогают справляться со стрессом и напряжением.

# Общие сведения о проделанной работе

Этапы выполнения проекта:

* Разработать главную цель игры и продумать механики геймплея.
* Создать уровни и проработать их наполнение, написать скрипты для механик геймплея.
* Подготовка отчёта и видео-презентации.

Для разработки была выбрана программа Unity и язык программирования C#, так как он содержит все необходимые для проекта библиотеки, он достаточно быстр и легок в освоении.

На первом этапе необходимо определить основную тему и концепцию игры, выбрать визуальный стиль и атмосферу. Важно сформулировать главную цель игры, чтобы игрок понимал, чего он должен достичь, будь то прохождение уровней или решение загадок. Также нужно разработать механики геймплея, описывающие взаимодействие игрока с объектами и уникальные элементы, добавляющие интерес.

На втором этапе требуется спроектировать уровни, определив их количество и сложность, а также продумать расположение препятствий и предметов. Наполнение уровней включает выбор и создание элементов окружения, разработку головоломок для каждого уровня, а также внедрение системы подсказок. Обязательно важно протестировать уровни на наличие ошибок и собрать отзывы от тестировщиков, что поможет внести необходимые улучшения.

На финальном этапе нужно собрать всю информацию о процессе разработки, включая цели, выполненные задачи и результаты тестирования. После этого оформляется отчет с графиками и ключевыми выводами, описывающий механики геймплея и уровни. Для видео-презентации снимается игровой процесс, показывающий ключевые механики и уровни с голосовым сопровождением, объясняющим особенности игры. Финальные правки для отчета и видео позволят сделать материалы четкими и привлекательными для представления перед инвесторами или потенциальными игроками.

# Результаты проекта

В результате проекта была разработана игра-головоломка, которая начинается с главного меню (рисунок 1), где игрок может выбрать одну из следующих опций: начать игру, выбрать уровни или выйти из программы.

Изображение выглядит как мультфильм, снимок экрана, Анимация, Компьютерная игра

Содержимое, созданное искусственным интеллектом, может быть неверным.

1 рисунок

Игра начинается с первого уровня, который вовлекает игрока в процесс (рисунок 2). В начале уровня игрок получает задание найти ключи для открытия дверей замка. Это создает мотивацию для исследования и взаимодействия с окружающей средой.

Изображение выглядит как небо, Компьютерная игра, Программное обеспечение для видеоигр, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

2 рисунок

По пути игрок сталкивается с различными головоломками, которые нужно решить для получения ключей. Каждая головоломка предлагает уникальный вызов, что делает игровой процесс разнообразным и интересным. Это не только развивает логическое мышление игроков, но и создает позитивный опыт вовлеченности в игру (рисунок 3, 4, 5, 6).

Изображение выглядит как небо, снимок экрана, Компьютерная игра, Программное обеспечение для видеоигр

Содержимое, созданное искусственным интеллектом, может быть неверным.

3 рисунок

Изображение выглядит как небо, снимок экрана, Компьютерная игра, Программное обеспечение для видеоигр

Содержимое, созданное искусственным интеллектом, может быть неверным.

4 рисунок

Изображение выглядит как небо, снимок экрана, мультфильм

Содержимое, созданное искусственным интеллектом, может быть неверным.

5 рисунок

Изображение выглядит как небо, Программное обеспечение для видеоигр, Компьютерная игра, Майнкрафт

Содержимое, созданное искусственным интеллектом, может быть неверным.

6 рисунок

Весь проект фокусируется на построении увлекательных механик, которые удерживают внимание игрока и стимулируют его продолжать проходить уровни. Игра обладает интуитивно понятным интерфейсом и продуманным вариантом взаимодействия, что делает её доступной для широкой аудитории. В результате проект полностью соответствует поставленным целям и задачам, обеспечивая игрокам интересный и увлекательный опыт!

**Приложение 1.**Программный код

Ссылка на репозиторий: https://github.com/Lismen96/-\_-.git

Скрипт Acceleration:

using UnityEngine;

using SimpleFPS;

public class Acceleration : MonoBehaviour

{

[SerializeField] private float bonus;

private void OnTriggerEnter(Collider other)

{

FirstPersonController fps = other.GetComponent<FirstPersonController>();

if (fps != null)

{

fps.m\_JumpSpeed += bonus;

fps.m\_WalkSpeed += bonus;

fps.m\_RunSpeed += bonus;

}

}

private void OnTriggerExit(Collider other)

{

FirstPersonController fps = other.GetComponent<FirstPersonController>();

if (fps != null)

{

fps.m\_JumpSpeed -= bonus;

fps.m\_WalkSpeed -= bonus;

fps.m\_RunSpeed -= bonus;

}

}

}

Скрипт Bag:  
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.Events;

public class Bag : MonoBehaviour

{

private int amountKey;

public UnityEvent ChangeAmountKey;

public void AddKey(int amount)

{

amountKey += amount;

ChangeAmountKey.Invoke();

}

public bool DrawKey(int amount)

{

if (amountKey - amount < 0) return false;

amountKey -= amount;

ChangeAmountKey.Invoke();

return true;

}

public int GetAmountKey()

{

return amountKey;

}

}

Скрипт DamageZone:

using SimpleFPS;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using static UnityEngine.GraphicsBuffer;

public class DamageZone : MonoBehaviour

{

[SerializeField]private int damage;

[SerializeField]private float damageRate;

private Destructible destructible;

private float timer;

private void Update()

{

timer += Time.deltaTime;

if(timer >= damageRate)

{

if (destructible != null)

{

destructible.AppltDamage(damage);

}

timer = 0;

}

}

private void OnTriggerEnter(Collider other)

{

destructible = other.GetComponent<Destructible>();

}

private void OnTriggerExit(Collider other)

{

if (other.GetComponent<Destructible>() == destructible) destructible = null;

}

}

Скрипт Destructible:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.Events;

public class Destructible : MonoBehaviour

{

[SerializeField] private int maxHitPoints;

public UnityEvent Die;

public UnityEvent ChangeHitPoints;

[HideInInspector] public int hitPoints;

private void Start()

{

hitPoints = maxHitPoints;

ChangeHitPoints.Invoke();

}

public void AppltDamage(int damage)

{

ChangeHitPoints.Invoke();

hitPoints -= damage;

if (hitPoints <= 0)

{

Kill();

}

}

public void Kill()

{

hitPoints = 0;

ChangeHitPoints.Invoke();

Die.Invoke();

}

public int GetHitPoints()

{

return hitPoints;

}

public int GetMaxHitPoints()

{

return maxHitPoints;

}

}

Скрипт HealthPoints:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class HealthPoints : Pikup

{

[SerializeField] private int healing;

private Destructible destructible;

protected override void OnTriggerEnter(Collider other)

{

base.OnTriggerEnter(other);

destructible = other.GetComponent<Destructible>();

if (destructible != null)

{

destructible.hitPoints += healing;

destructible.ChangeHitPoints.Invoke();

}

}

}

Скрипт ImpactAffect:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ImpactAffect : MonoBehaviour

{

[SerializeField] private float timer;

private void Start()

{

Destroy(gameObject,timer);

}

}

Скрипт Key:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Key : Pikup

{

[SerializeField] private GameObject ImpactEffect;

protected override void OnTriggerEnter(Collider other)

{

base.OnTriggerEnter(other);

Bag bag = other.GetComponent<Bag>();

if(bag != null)

{

bag.AddKey(1);

Instantiate(ImpactEffect);

}

}

}

Скрипт KeyTrigger:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.Events;

public class KeyTrigger : MonoBehaviour

{

[SerializeField] private GameObject massegeBox;

[SerializeField] private UnityEvent enter;

[SerializeField] private int AmountKeyActive;

private bool isActive = false;

protected void OnTriggerEnter(Collider other)

{

if (isActive == true) return;

Bag bag = other.GetComponent<Bag>();

if (bag != null)

{

if(bag.DrawKey(AmountKeyActive) == true)

{

enter.Invoke();

isActive = true;

}

else

{

massegeBox.SetActive(true);

}

}

}

private void OnTriggerExit(Collider other)

{

massegeBox.SetActive(false);

}

}

Скрипт KillZone:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class KillZone : MonoBehaviour

{

private void OnTriggerEnter(Collider other)

{

Destructible destructible = other.GetComponent<Destructible>();

if (destructible != null)

{

destructible.Kill();

}

}

}

Скрипт MoveTo:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MoveTo : MonoBehaviour

{

[SerializeField] public float speed;

[SerializeField] public Transform target;

private void Update()

{

transform.position = Vector3.MoveTowards(transform.position, target.position, speed \* Time.deltaTime);

}

}

Скрипт PauseManager:

using UnityEngine;

public class PauseManager : MonoBehaviour

{

private bool isPaused = false;

[SerializeField] private GameObject pauseMenu;

void Update()

{

if (Input.GetKeyDown(KeyCode.P))

{

TogglePause();

}

}

public void TogglePause()

{

if (isPaused)

{

Resume();

}

else

{

Pause();

}

}

public void Pause()

{

Time.timeScale = 0f;

isPaused = true;

pauseMenu.SetActive(true);

Cursor.visible = true;

Cursor.lockState = CursorLockMode.None;

}

public void Resume()

{

Time.timeScale = 1f;

isPaused = false;

pauseMenu.SetActive(false);

Cursor.visible = false;

Cursor.lockState = CursorLockMode.Locked;

}

public void MainMenu()

{

Time.timeScale = 0f;

Cursor.visible = true;

Cursor.lockState = CursorLockMode.None;

pauseMenu.SetActive(false);

}

public bool IsPaused()

{

return isPaused;

}

}

Скрипт Pikup:

using UnityEngine;

using SimpleFPS;

public class Pikup : MonoBehaviour

{

protected virtual void OnTriggerEnter(Collider other)

{

FirstPersonController fps = other.GetComponent<FirstPersonController>();

if (fps !=null)

{

Destroy(gameObject);

}

}

}

Скрипт Potion:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Potion : Pikup

{

[SerializeField] private GameObject ImpactEffect;

protected override void OnTriggerEnter(Collider other)

{

base.OnTriggerEnter(other);

Instantiate(ImpactEffect);

}

}

Скрипт Rotate:

using UnityEngine;

public class Rotate : MonoBehaviour

{

[SerializeField] public float speed;

[SerializeField] public Vector3 target;

private void Update()

{

transform.localRotation = Quaternion.RotateTowards(transform.localRotation, Quaternion.Euler(target), speed \* Time.deltaTime);

}

}

Скрипт Rotator:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Rotator : MonoBehaviour

{

[SerializeField] public Vector3 speedRotate;

[SerializeField] public float amplitude;

[SerializeField] public float frequency;

private Vector3 startPosition;

private float newY;

private void Start()

{

startPosition = transform.position;

}

private void Update()

{

transform.Rotate(speedRotate \* Time.deltaTime);

newY = startPosition.y + Mathf.Sin(Time.time \* frequency) \* amplitude;

transform.position = new Vector3(transform.position.x, newY, transform.position.z);

}

}

Скрипт SceneHelper:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class SceneHelper : MonoBehaviour

{

public void RestartLevel()

{

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

}

public void LoadLevel(int buildIndex)

{

SceneManager.LoadScene(buildIndex);

}

public void Quit()

{

Application.Quit();

}

public void Play()

{

Time.timeScale = 1f;

Cursor.visible = false;

Cursor.lockState = CursorLockMode.Locked;

}

}

Скрипт SpringJump:

using SimpleFPS;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[RequireComponent(typeof(AudioSource))]

public class SpringJump : MonoBehaviour

{

[SerializeField] private float jumpForce;

[SerializeField] private new AudioSource audio;

[SerializeField] private new Transform transform;

private bool isExitingTrigger;

private float previusJump;

private void Start()

{

audio = GetComponent<AudioSource>();

}

private void OnTriggerEnter(Collider other)

{

FirstPersonController fps = other.GetComponent<FirstPersonController>();

if (fps != null)

{

previusJump = fps.m\_JumpSpeed;

fps.m\_JumpSpeed += jumpForce;

fps.m\_Jump = true;

transform.localPosition -= new Vector3(0, 0.05f, 0);

audio.Play();

}

}

private void OnTriggerExit(Collider other)

{

FirstPersonController fps = other.GetComponent<FirstPersonController>();

if (fps != null)

{

fps.m\_JumpSpeed = previusJump;

isExitingTrigger = true;

}

}

private void Update()

{

if (isExitingTrigger == true && transform.localPosition.y < 0.132f)

{

transform.localPosition += new Vector3(0, 0.01f, 0);

}

else if (transform.localPosition.y >= 0.132f)

{

isExitingTrigger = false; // Остановить подъем, если достигли нужной высоты

}

}

}

Скрипт Teleport:

using SimpleFPS;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[RequireComponent(typeof(AudioSource))]

public class Teleport : MonoBehaviour

{

[SerializeField] private Teleport target;

[SerializeField] private new AudioSource audio;

[HideInInspector] public bool IsReceive;

private void Start()

{

audio = GetComponent<AudioSource>();

}

private void OnTriggerEnter(Collider other)

{

if (IsReceive) return;

FirstPersonController fps = other.GetComponent<FirstPersonController>();

if (fps != null)

{

target.IsReceive = true;

fps.transform.position = target.transform.position;

audio.Play();

}

}

private void OnTriggerExit(Collider other)

{

FirstPersonController fps = other.GetComponent<FirstPersonController>();

if (fps != null)

{

IsReceive = false;

}

}

}

Скрипт Trigger:

using SimpleFPS;

using UnityEngine;

using UnityEngine.Events;

public class Trigger : MonoBehaviour

{

public UnityEvent enter;

public UnityEvent exit;

private void OnTriggerEnter(Collider other)

{

FirstPersonController fps = other.GetComponent<FirstPersonController>();

if (fps != null)

{

enter.Invoke();

}

}

private void OnTriggerExit(Collider other)

{

FirstPersonController fps = other.GetComponent<FirstPersonController>();

if (fps != null)

{

exit.Invoke();

}

}

}

Скрипт UIHealth:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class UIHealth : MonoBehaviour

{

[SerializeField] private Destructible destructible;

[SerializeField] private Image image;

private void Start()

{

destructible.ChangeHitPoints.AddListener(OnChangeHitPoints);

}

private void OnDestroy()

{

destructible.ChangeHitPoints.RemoveListener(OnChangeHitPoints);

}

private void OnChangeHitPoints()

{

image.fillAmount =(float) (destructible.GetHitPoints() / (float)destructible.GetMaxHitPoints());

}

}

Скрипт UIKey:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class UIKey : MonoBehaviour

{

[SerializeField] private Bag bag;

[SerializeField] private Text text;

private void Start()

{

bag.ChangeAmountKey.AddListener(OnChangeKey);

}

private void OnDestroy()

{

bag.ChangeAmountKey.RemoveListener(OnChangeKey);

}

private void OnChangeKey()

{

text.text = bag.GetAmountKey().ToString();

}

}