СОДЕРЖАНИЕ

[1. Постановка задачи 2](#_Toc135334696)

[2. Реализация 3](#_Toc135334697)

[3. Скриншоты работы программы 8](#_Toc135334698)

[3. Листинг 12](#_Toc135334699)

[Hero.cs 12](#_Toc135334700)

[HeroInputReader.cs 17](#_Toc135334701)

[GameTimer.cs 18](#_Toc135334702)

[AnimatedWindow.cs 18](#_Toc135334703)

[SelectionWindow.cs 19](#_Toc135334704)

[PauseMenu.cs 20](#_Toc135334705)

[MainMenuWindow.cs 21](#_Toc135334706)

[DamageComponent.cs 22](#_Toc135334707)

[HealthComponent.cs 22](#_Toc135334708)

[DestroyObjectComponent.cs 23](#_Toc135334709)

[EnterCollisionComponent.cs 23](#_Toc135334710)

[EnterTriggerComponent.cs 23](#_Toc135334711)

[InteractableComponent.cs 24](#_Toc135334712)

[ReloadLevelComponent.cs 24](#_Toc135334713)

[SpawnCoin.cs 25](#_Toc135334714)

[SpriteAnimation.cs 26](#_Toc135334715)

[SwithComponent.cs 27](#_Toc135334716)

[TeleportComponent.cs 27](#_Toc135334717)

[HudController.cs 28](#_Toc135334718)

[ProgressBar.cs 28](#_Toc135334719)

# 1. Постановка задачи

В рамках выполнения расчетно-графической работы необходимо было реализовать игровое приложение с помощью кроссплатформенной среды разработки игр Unity на языке C#. В качестве жанра была выбрана аркада «Сбор монет на время».

# 2. Реализация

Суть игры заключается в сборе появляющихся на уровне монет на время. При исследовании уровня игроку будут попадаться закрытые двери, которые можно открыть, найдя переключатель, предметы с физикой (бочки), а также шипы, делающие что и должно — наносящие урон. Для удобства перемещения, игрок может карабкаться по стенам и делать второй прыжок в воздухе. В самой отдалённой точке на уровне можно найти портал, который телепортирует игрока ближе к центру.

Что бы не умереть раньше окончания таймера, игрок может использовать зелья лечения, которые разбросаны по уровню, а что бы продлить время надо находить фиолетовые монеты, которые будут прибавлять к таймеру дополнительные 15 сек.

Так же на уровне появляются: просто большие монеты, приносящие больше очков, монеты мультипликаторы, которые на время умножают все подбираемые монеты на 2, красные монеты-ловушки, которые отнимают здоровье сродни шипам, и самые опасные — черные монеты, мгновенной смерти.

Монеты появляются на заранее заготовленных точках спавна, у каждого вида монет свои точки спавна, в зависимости от их полезности: обычные монеты для набора очков появляются по всему уровню, особые монеты появляются реже и в таких местах куда нужно постараться забраться и не получить урон от шипов.

Для реализации всех игровых механик были написаны скрипты c# на основе компонентной ориентированного подхода (КОП). Вместо того, чтобы создавать объекты в виде классов, как это делается в других движках, в Unity объекты создаются в виде набора компонентов, которые могут быть добавлены и удалены в любое время. Каждый компонент отвечает за определенную функциональность игрового объекта, такую как графика, физика и т.д.

Скрипты заключают в себе следующую игровую логику:

* Открытие уровней, главного меню, меню паузы и меню конца игры;

PauseMenu.cs , SelectionWindow, MainMenuWindow.cs — скрипты для обработки нажатий на кнопки соответствующих названиям меню. Объект (Панель на слое Canvas), содержащий меню конца игры становится активным (отображается на экране) когда игрок теряет все жизни или таймер заканчивается. Пауза вызывается на кнопку паузы или на клавишу Esc. Во время паузы игровой процесс останавливается при помощи Time.timeScale = 0f;.

* Реализация движений игрока по нажатию кнопок как на клавиатуре, так и по нажатию кнопок на экране.

Для этого установлен модуль InputAction который регистрирует нажатия клавиш клавиатуры и передает координаты Vector2 в функцию класса игрока SetDirection которая обновляет координаты положения игрока в мире. Для кнопок на экране установлен компонент, который эмулирует нажатия кнопок клавиатуры «On-Screen Button». Обработка нажатий происходит в HeroInputReader.cs.

* Логика особых монет (дополнительное время, бонус к множителю монет, ловушки).

Своих скриптов у них нет, но в классах игрока и таймера предусмотрены функции, которые вызываются при сборе таких монет. Такие фукнции как — GamerTimer.AddTime добавляющий время таймеру и Hero.ActivateMultyplyCoins активирующий флаг запуска таймера умножения собираемых далее монет на 2.

* Определение коллизий игрока с игровыми объектами и уровнем.

Определение положения игрока (на земле или в воздухе) определяется при помощи создания невидимой сферы (Physics2D.CircleCast), которая проверяет есть ли пересечение со слоем уровня или с другими объектами под персонажем, это проверяется в функции Hero.IsGrounded().

Для определения коллизии с объектами по типу монеток или шипов на персонажа устанавливается капсульный коллайдер. Пересечение коллайдера персонажа с коллайдерами других объектов определяется скриптами EnterCollisionComponent и EnterTriggerComponent. При срабатывании этих скриптов, у объектов компонентами которых они являются, выполняются функции последствия этого столкновения (например получение урона/лечения, или использование предмета).

* Логика контроля здоровья персонажа, нанесения ему урон и фиксация его смерти.

Если предмет способен наносить урон персонажу ему устанавливается компонент DamageComponent, в котором указывается является действие уроном или исцелением и его значение. Далее эта информация попадает в HealthComponent который отвечает за контроль здоровья и определение смерти персонажа.

* Игровой таймер, который ведет обратный отчет, по истечению которого вызывается меню конца игры и выводится финальный счет.

Таймер контролируется компонентом GameTimer.cs. В нем указывается начальное значение таймера и объекты в которые будет отправляется эта информация для вывода на экран (таймер посередине экрана).

* Отображение текущего уровня здоровья игрока, количество собранных монет и текущее время на таймере.

Объекты типа «Текст» принимают значения текущее количество здоровья и монет из компонента Hero.cs, и обновленную информацию выводят на экран.

Для обновления информации о здоровье используется компоненты HudController — для обновления информации, и ProgressBar для отображения.

* Уничтожение объекта при соприкосновении с игроком (Относится к монетам и зелью).

При регистрации коллизии игрока и объекта, на котором установлен компонент DestroyObjectComponent этот объект удалиться со сцены.

* Смерть, если игрок упал за край карты.

За это отвечает коллайдер внизу уровня, соприкосновение игрока с которым происходит вызов компонента EnterTriggerComponent который инициализирует срабатывание функции смерти игрока Hero.OnDie.

* Настройка телепорта.

Есть два объекта сцены, один пустышка другой спрайт, спрайту присваиваем компонент TeleportComponent и записываем туда объект пустышку, который станет точкой куда телепортируется игрок.

* Логика предметов, с которыми игрок может взаимодействовать (переключатель и зелье) по нажатию на кнопку взаимодействия.

За это отвечает компонент InteractableComponent, при коллизии игрока с таким объектом вызываются функции указанные в событиях.

* Логика постоянного переключения спрайтов (анимации объектов, таких как пальмы, блики на воде, вращение монет и зелья, открытие закрытие двери и вращение переключателя).

Для этого написан компонент SpriteAnimation.cs, в котором указывается набор спрайтов, повторяется ли анимация, промежуток между сменой кадров и события при окончании анимации. Функция внутри этого компонента просто меняет спрайт на следующий из набора.

* Логика случайной генерации монет по уровню.

Происходит в компоненте SpawnCoin.cs. Суть заключается в том, что мы передаем компоненту набор пустых объектов сцены, которые будут отвечать за точки спавна монет, затем передаем ссылки на префабы объектов, которые будут появлятся в случайно выбранной точке из указанного набора с указанным промежутком. Скрипт контролирует какие из точек свободны, а какие готовы для появления в них новых объектов.

# 3. Скриншоты работы программы

Рис.1 — Главное меню

Рис.2 — Меню выбора уровня

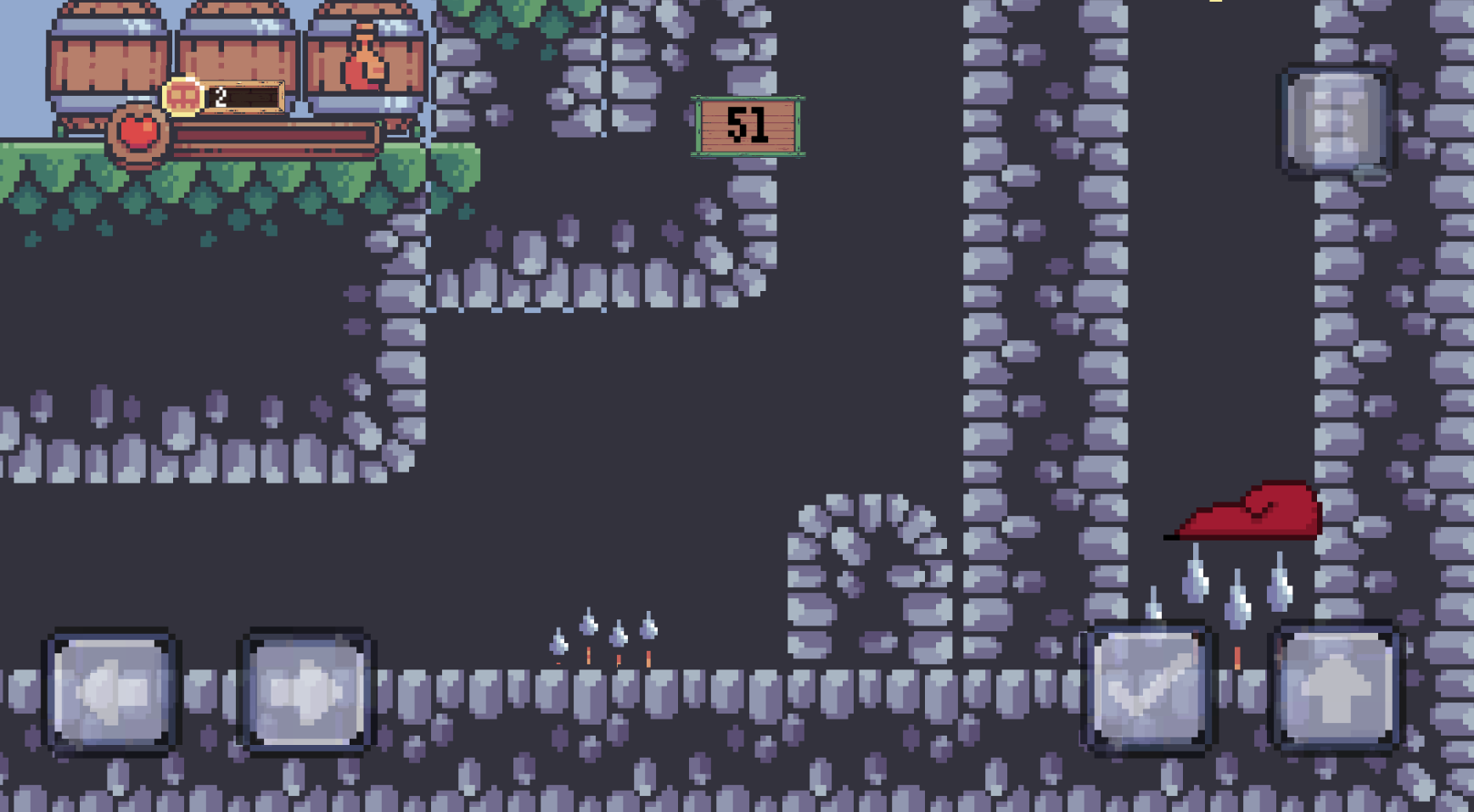
Рис.3 — Игровой процесс

Рис.4 — Смерть персонажа на шипах

Рис.5 — Игровой процесс 

Рис.6 — Персонаж избежал смерти предварительно скинув бочку на шипы

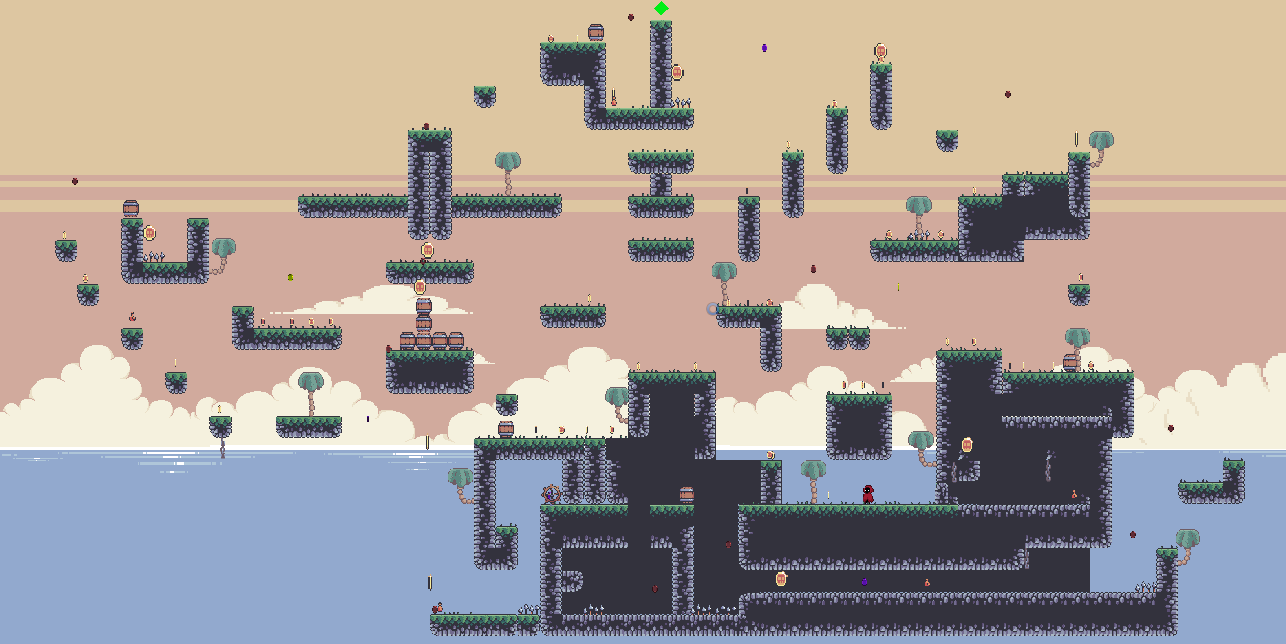
Рис.7 — Меню паузы

Рис.8 — Первый игровой уровень

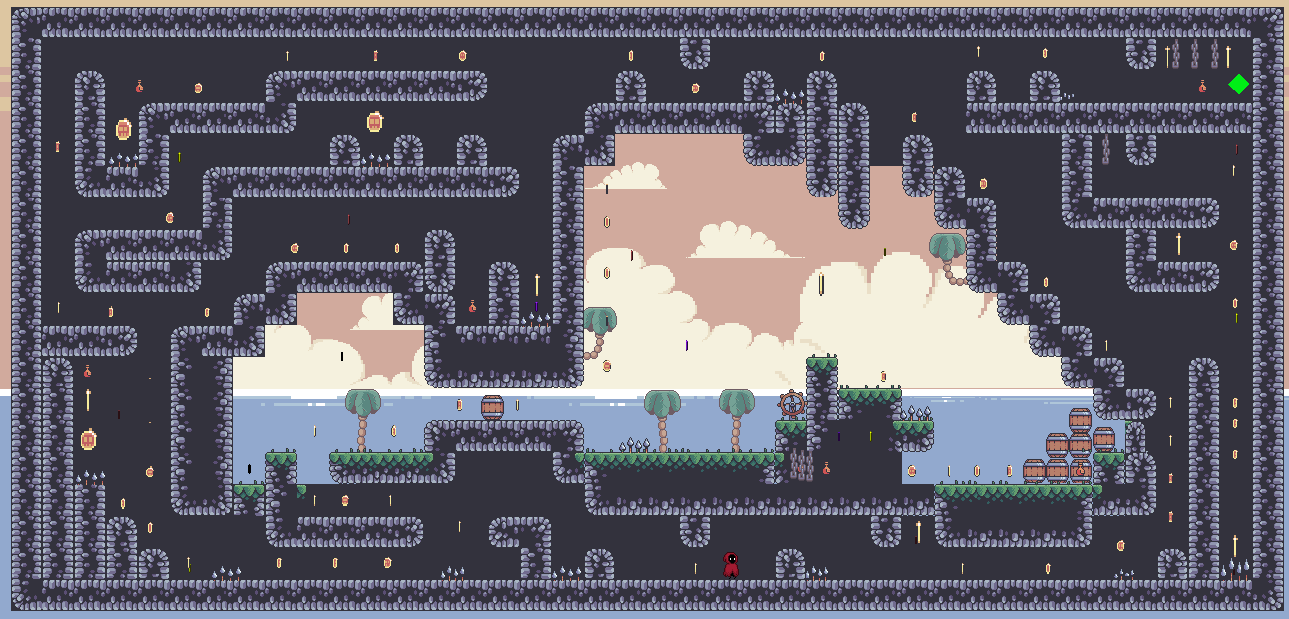


Рис.9 — Второй игровой уровень

# 3. Листинг

**Hero.cs**

using PixelCrew.Components;

using UnityEngine;

using UnityEngine.Events;

using UnityEngine.UI;

namespace PixelCrew

{

public class Hero : MonoBehaviour

{

[SerializeField] private float \_speed; // Виден в юнити как параметр

[SerializeField] private float \_jumpSpeed; // С какой скоростью прыгаем наверх

[SerializeField] private float \_damageJumpSpeed;

[SerializeField] private LayerMask \_groundLayer; // Слой земли

[SerializeField] private float \_groundCheckRadius;

[SerializeField] private Vector3 \_groundCheckPositionDelta; // Для отрисовки сферы под персонажем

[SerializeField] private float \_interactionRadius;

[SerializeField] private LayerMask \_interactionLayer;

[SerializeField] public Text coinsText;

[SerializeField] public Text multyTimerText;

[SerializeField] public Text finalScoreText;

[SerializeField] public GameObject multyTimerPanel;

[SerializeField] private GameObject \_activateButton;

[SerializeField] private UnityEvent \_onDie;

private Vector2 \_direction;

private Collider2D[] \_interactionResult = new Collider2D[1];

private Rigidbody2D \_rigidbody;

private Animator \_animator;

private SpriteRenderer \_sprite;

public int coinsCount;

public float multiplyTime = 0;

private bool \_isGounded;

private bool \_allowDoubleJump;

private bool \_isDie = false;

private static readonly int IsGroundKey = Animator.StringToHash("is-ground"); // что бы не вызывалось дальше каждый раз

private static readonly int VerticalVelocityKey = Animator.StringToHash("vertical-velocity");

private static readonly int IsRunningKey = Animator.StringToHash("is-running");

private static readonly int IsHittingKey = Animator.StringToHash("hit");

private static readonly int IsDieKey = Animator.StringToHash("die");

private void Awake()

{

\_rigidbody = GetComponent<Rigidbody2D>();

\_animator = GetComponent<Animator>();

\_sprite = GetComponent<SpriteRenderer>();

}

private void Start()

{

coinsText.text = coinsCount.ToString();

multyTimerText.text = Mathf.Round(multiplyTime).ToString();

}

public void SetDirection(Vector2 dir)

{

if (!\_isDie) \_direction = dir;

}

private void FixedUpdate() // Физика обрабатывается тут в FixedUpdate

{

var xVelocity = \_direction.x \* \_speed;

var yVelocity = CalculateYVelocity();

\_rigidbody.velocity = new Vector2(xVelocity, yVelocity);

if (!\_isDie)

{

\_animator.SetBool(IsGroundKey, \_isGounded); // Вызов анимаций

\_animator.SetFloat(VerticalVelocityKey, \_rigidbody.velocity.y);

\_animator.SetBool(IsRunningKey, \_direction.x != 0);

}

UpdateSpriteDirection();

}

private void Update()

{

if (multiplyTime > 0)

{

multiplyTime -= Time.deltaTime;

multyTimerText.text = Mathf.Round(multiplyTime).ToString();

}

else

{

if(multyTimerPanel.active) multyTimerPanel.active = false;

}

\_isGounded = IsGrounded();

}

private void UpdateSpriteDirection()

{

if(\_direction.x > 0)

{

\_sprite.flipX = false;

}

else if(\_direction.x< 0)

{

\_sprite.flipX = true;

}

}

public void ActivateMultiplyCoins()

{

multiplyTime += 10f;

}

public void GetCoins(int count)

{

if (multiplyTime > 0)

{

count \*= 2;

}

coinsCount+= count;

//Debug.Log(coinsCount);

coinsText.text = coinsCount.ToString();

}

public void TakeDamage()

{

if (!\_isDie)

{

\_animator.SetTrigger(IsHittingKey);

\_rigidbody.velocity = new Vector2(\_rigidbody.velocity.x, \_damageJumpSpeed);

}

}

public void OnDie()

{

\_isDie = true;

//\_animator.SetTrigger(IsHittingKey);

\_animator.SetTrigger(IsDieKey);

}

public void AfterDeathEvents()

{

if (\_isDie)

{

Time.timeScale = 0f;

finalScoreText.text = coinsCount.ToString();

//Debug.Log("GAME OVER");

\_onDie?.Invoke();

}

}

private float CalculateYVelocity()

{

var yVelocity = \_rigidbody.velocity.y;

var isJumpPressing = \_direction.y > 0;

if (\_isGounded) \_allowDoubleJump = true; // 2 прыжок будет доступен только после приземления

if (isJumpPressing)

{

yVelocity = CalculateJumpVelocity(yVelocity);

}

else if (\_rigidbody.velocity.y > 0) // Если отпустим кнопку прыжка то замедляемся

{

yVelocity \*= 0.5f;

}

return yVelocity;

}

private float CalculateJumpVelocity(float yVelocity)

{

var isFalling = \_rigidbody.velocity.y <= 0.001f;

if (!isFalling) return yVelocity;

// Логика 2го прыжка

if (\_isGounded)

{

yVelocity += \_jumpSpeed;

}

else if(\_allowDoubleJump)

{

yVelocity = \_jumpSpeed;

\_allowDoubleJump = false;

}

return yVelocity;

}

private bool IsGrounded() // Перисекается ли наш объект с землей

{

//var hitGround = Physics2D.Raycast(transform.position, Vector2.down, 1, \_groundLayer); // Raycast - Откладывает луч и возвращает с чем луч пересекся

var hitGround = Physics2D.CircleCast(transform.position + \_groundCheckPositionDelta, \_groundCheckRadius, Vector2.down, 0, \_groundLayer);

// Создаем сферу для определения стоит ли перс на земле

return hitGround.collider != null;

}

private void OnDrawGizmos()

{

//Debug.DrawRay(transform.position, Vector3.down, IsGrounded() ? Color.green : Color.red); //Проверка задевает ли луч землю

Gizmos.color = IsGrounded() ? Color.green : Color.red;

Gizmos.DrawSphere(transform.position + \_groundCheckPositionDelta, \_groundCheckRadius);

// Рисуем сферу для наглядности

}

public void Interact()

{

var size = Physics2D.OverlapCircleNonAlloc(transform.position, \_interactionRadius,

\_interactionResult, \_interactionLayer); // получаем все пересекающиеся объекты

// создаст сферу и все что в нее попадет запишет в \_interactionResult

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

{

var interactable = \_interactionResult[i].GetComponent<InteractableComponent>();

if(interactable != null) // Если находим

{

interactable.Interact(); // Запускаем интерактивное действие этого объекта

}

}

}

}

}

**HeroInputReader.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.InputSystem;

namespace PixelCrew

{

public class HeroInputReader : MonoBehaviour

{

[SerializeField] private Hero \_hero;

public void Movement(InputAction.CallbackContext context)

{

Vector2 dir = context.ReadValue<Vector2>(); // Срабатывает по событию нажатия именно когда нажали на кнопку

\_hero?.SetDirection(dir);

}

public void OnInteract(InputAction.CallbackContext context)

{

if (context.canceled)

{

\_hero?.Interact();

}

}

}

}

**GameTimer.cs**

using UnityEngine;

using UnityEngine.Events;

using UnityEngine.UI;

namespace PixelCrew.Components

{

public class GameTimer : MonoBehaviour

{

[SerializeField] private float \_timeStart = 60;

[SerializeField] private Text timerText;

[SerializeField] private UnityEvent \_timerEndAction;

private bool stopTimer = false ;

private void Start()

{

timerText.text = \_timeStart.ToString();

}

private void Update()

{

if (\_timeStart <= 0f && stopTimer == false )

{

\_timerEndAction?.Invoke();

stopTimer = true ;

}

else

{

\_timeStart -= Time.deltaTime;

timerText.text = Mathf.Round(\_timeStart).ToString();

}

}

public void AddTime(float seconds)

{

\_timeStart += seconds;

}

}

}

**AnimatedWindow.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

namespace PixelCrew.UI

{

public class AnimatedWindow : MonoBehaviour

{

private Animator \_animator;

private static readonly int Show = Animator.StringToHash("Show");

private static readonly int Hide = Animator.StringToHash("Hide");

private void Start()

{

\_animator = GetComponent<Animator>();

\_animator.SetTrigger(Show);

}

public void Close()

{

\_animator.SetTrigger(Hide);

}

public virtual void OnCloseAnimationCompete()

{

Destroy(gameObject);

}

}

}

**SelectionWindow.cs**

using UnityEngine.SceneManagement;

namespace PixelCrew.UI.SelectionWindow

{

public class SelectionWindow : AnimatedWindow

{

private string \_level;

public void OnStartLevel1()

{

\_level = "Level1";

Close();

}

public void OnStartLevel2()

{

\_level = "Level2";

Close();

}

public override void OnCloseAnimationCompete()

{

base.OnCloseAnimationCompete();

if(\_level!=null) SceneManager.LoadScene(\_level);

}

}

}

**PauseMenu.cs**

using UnityEngine;

using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour

{

public static bool GameIsPaused = false;

public GameObject pauseMenuUI;

public void Update()

{

if (Input.GetKeyDown(KeyCode.Escape))

{

if (GameIsPaused)

{

Resume();

}

else

{

Pause();

}

}

}

public void Resume()

{

pauseMenuUI.SetActive(false);

Time.timeScale = 1f;

GameIsPaused = false;

}

public void Pause()

{

if (GameIsPaused)

{

Resume();

}

else

{

pauseMenuUI.SetActive(true);

Time.timeScale = 0f;

GameIsPaused = true;

}

}

public void GoToMainMenu()

{

Time.timeScale = 1f;

SceneManager.LoadScene("MainMenu");

}

public void QuitGame()

{

Application.Quit();

Debug.Log("Quit");

}

}

**MainMenuWindow.cs**

using System;

using UnityEngine;

using UnityEngine.SceneManagement;

namespace PixelCrew.UI.MainMenu

{

public class MainMenuWindow : AnimatedWindow

{

public void OnShowLevels()

{

var window = Resources.Load<GameObject>("UI/SelectionWindow");

var canvas = FindObjectOfType<Canvas>();

Instantiate(window, canvas.transform);

}

public void OnStartGame()

{

Close();

}

public void OnExit()

{

Application.Quit();

}

public override void OnCloseAnimationCompete()

{

base.OnCloseAnimationCompete();

SceneManager.LoadScene("Level1");

}

}

}

**DamageComponent.cs**

using UnityEngine;

namespace PixelCrew.Components

{

public class DamageComponent : MonoBehaviour

{

[SerializeField] private int \_damage;

[SerializeField] private bool \_itsHeal;

public void ApplyDamage(GameObject target)

{

var healthComponent = target.GetComponent<HealthComponent>();

if (\_itsHeal) \_damage \*= -1;

if (healthComponent != null)

{

healthComponent.ApplyDamage(\_damage);

}

}

}

}

**HealthComponent.cs**

using PixelCrew.UI.Hud;

using UnityEngine;

using UnityEngine.Events;

namespace PixelCrew.Components

{

public class HealthComponent : MonoBehaviour

{

[SerializeField] private int \_health = 5;

[SerializeField] private UnityEvent \_onDamage;

[SerializeField] private UnityEvent \_onDie;

[SerializeField] private HudController \_hud;

public void ApplyDamage(int damageValue)

{

\_health -= damageValue;

if(damageValue>0) \_onDamage?.Invoke(); // вызываем эвент

\_hud.UpdateHP(\_health);

if (\_health <= 0)

{

\_onDie?.Invoke();

}

}

}

}

**DestroyObjectComponent.cs**

using UnityEngine;

namespace PixelCrew.Components

{

public class DestroyObjectComponent : MonoBehaviour

{

[SerializeField] private GameObject \_objectToDestroy;

[SerializeField] private float \_inputTimer;

public void DestroyObject()

{

Destroy(\_objectToDestroy, \_inputTimer);

}

}

}

**EnterCollisionComponent.cs**

using UnityEngine;

using UnityEngine.Events;

namespace PixelCrew.Components

{

public class EnterCollisionComponent : MonoBehaviour

{

[SerializeField] private string \_tag;

[SerializeField] private UnityEvent<GameObject> \_action;

private void OnCollisionEnter2D(Collision2D other)

{

if (other.gameObject.CompareTag(\_tag))

{

\_action?.Invoke(other.gameObject);

}

}

}

}

**EnterTriggerComponent.cs**

using UnityEngine;

using UnityEngine.Events;

namespace PixelCrew.Components

{

public class EnterTriggerComponent : MonoBehaviour

{

[SerializeField] private string \_tag;

[SerializeField] private UnityEvent \_action;

[SerializeField] private UnityEvent \_onComplete;

private void OnTriggerEnter2D(Collider2D other)

{

if (other.gameObject.CompareTag(\_tag))

{

\_action?.Invoke();

\_onComplete?.Invoke();

}

}

}

}

**InteractableComponent.cs**

using UnityEngine;

using UnityEngine.Events;

namespace PixelCrew.Components

{

public class InteractableComponent : MonoBehaviour

{

[SerializeField] private UnityEvent \_action;

public void Interact()

{

\_action?.Invoke();

}

}

}

**ReloadLevelComponent.cs**

using UnityEngine;

using UnityEngine.SceneManagement;

namespace PixelCrew.Components

{

public class ReloadLevelComponent : MonoBehaviour

{

public void Reload() // Перезагрузим сцену на изначальную

{

var scene = SceneManager.GetActiveScene(); // Получаем активную сцену

SceneManager.LoadScene(scene.name);

}

}

}

**SpawnCoin.cs**

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using UnityEngine;

namespace PixelCrew.Components

{

public class SpawnCoin : MonoBehaviour

{

public float spawnInterval = 2f; // промежуток времени между спавнами

public GameObject[] spawnPoints; // массив точек спавна

public GameObject[] spawnObjects; // массив объектов, которые будут спавниться

private Dictionary<Vector2, bool> usedPositions = new Dictionary<Vector2, bool>(); // словарь позиций, на которых уже есть объекты

void Start()

{

// заполняем словарь позиций, указывая, что в начале на этих позициях объектов нет

foreach (GameObject spawnPoint in spawnPoints)

{

usedPositions.Add(spawnPoint.transform.position, false);

}

// вызываем метод SpawnObject через заданный промежуток времени каждый раз

InvokeRepeating("SpawnObject", 0f, spawnInterval);

}

void SpawnObject()

{

// выбираем случайную точку спавна из массива точек

int spawnIndex = Random.Range(0, spawnPoints.Length);

GameObject spawnPoint = spawnPoints[spawnIndex];

if (!usedPositions[spawnPoint.transform.position])

{

// выбираем случайный объект для спавна

int objectIndex = Random.Range(0, spawnObjects.Length);

GameObject objectToSpawn = spawnObjects[objectIndex];

// спавним объект на выбранной точке

Instantiate(objectToSpawn, spawnPoint.transform.position, Quaternion.identity);

// помечаем позицию, как использованную

usedPositions[spawnPoint.transform.position] = true;

}

}

public void FreePOsition(GameObject position)

{

usedPositions[position.transform.position] = false;

}

}

}

**SpriteAnimation.cs**

using UnityEngine;

using UnityEngine.Events;

namespace PixelCrew

{

[RequireComponent(typeof(SpriteRenderer))]

public class SpriteAnimation : MonoBehaviour // Для удобства анимирования простых анимаций

{

[SerializeField] private int \_frameRate;

[SerializeField] private bool \_loop;

[SerializeField] private Sprite[] \_sprites;

[SerializeField] private UnityEvent \_onComplete;

private SpriteRenderer \_renderer;

private float \_secondsPerFrame;

private int \_currentSpriteIndex;

private float \_nextFrameTime;

private bool \_isPlaying = true;

private void Start()

{

\_renderer = GetComponent<SpriteRenderer>();

}

private void OnEnable()

{

\_secondsPerFrame = 1f / \_frameRate;

\_nextFrameTime = Time.time + \_secondsPerFrame;

\_currentSpriteIndex = 0;

}

private void Update()

{

if (\_nextFrameTime > Time.time) return;

if(\_currentSpriteIndex >= \_sprites.Length)

{

if (\_loop)

{

\_currentSpriteIndex = 0;

}

else

{

enabled = false;

\_onComplete?.Invoke();

return;

}

}

\_renderer.sprite = \_sprites[\_currentSpriteIndex];

\_nextFrameTime += \_secondsPerFrame;

\_currentSpriteIndex++;

}

}

}

**SwithComponent.cs**

using UnityEngine;

namespace PixelCrew.Components

{

public class SwithComponent : MonoBehaviour

{

[SerializeField] private Animator \_animator;

[SerializeField] private bool \_state;

[SerializeField] private string \_animatorKey;

public void Switch()

{

\_state = !\_state;

\_animator.SetBool(\_animatorKey, \_state);

}

}

}

**TeleportComponent.cs**

using UnityEngine;

namespace PixelCrew.Components

{

public class TeleportComponent : MonoBehaviour

{

[SerializeField] private Transform \_destTransform;

public void Teleport(GameObject targer)

{

targer.transform.position = \_destTransform.position;

}

}

}

**HudController.cs**

using UnityEngine;

namespace PixelCrew.UI.Hud

{

public class HudController : MonoBehaviour

{

[SerializeField] private ProgressBar \_healthBar;

public void UpdateHP(int newValue)

{

\_healthBar.SetProgress(newValue);

}

}

}

**ProgressBar.cs**

using UnityEngine;

using UnityEngine.UI;

namespace PixelCrew.UI.Hud

{

public class ProgressBar : MonoBehaviour

{

[SerializeField] private int \_maxHP;

[SerializeField] private Image \_bar;

private float newValue;

public void SetProgress(int progress)

{

newValue = ((progress \* 100f) / \_maxHP) / 100f;

\_bar.fillAmount = newValue;

Debug.Log(newValue);

}

}

}