Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Основы алгоритмизации и программирования

Отчёт
по учебной практике(ознакомительной)
на тему:

Игра Platformer (Wall Kickers)

Студент Быбко А.А.

Руководитель Владымцев В.Д.

МИНСК 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ……..4

1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ……..5

2. ОБЗОР ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ……...5

3. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ……...6

3.1. Схема головного модуля…….....6

3.2. Диаграмма классов.....6

3.3 Use-case диаграмма.....6

4. КОД ПРОГРАММЫ.....7

4.1 Hero.cs.....7

4.2 CameraController.cs.....10

4.3 ChangeScenes.cs…….11

4.4 Menu.cs……...12

4.5 Leaper.cs…….12

4.6 Thorns.cs……13

4.7 Sofa.cs……13

4.8 VerticalThorn.cs……14

4.9 HorizontalThorn.cs…….15

4.10 DontDestroy.cs…….16

5. РЕЗУЛЬТАТ……...17

ЗАКЛЮЧЕНИЕ……..18

ПРИМЕНЕНИЕ……..19

СПИСОК ЛИТЕРАТУРЫ…….20

ПРИЛОЖЕНИЕ А……...21

ПРИЛОЖЕНИЕ Б……...22

ПРИЛОЖЕНИЕ В……...23

ВВЕДЕНИЕ

Для многих компьютерные игры — это настоящий процесс творения, а именно создание мира, в котором могут быть как люди, так и чудовища, различные континенты, галактики. Каждый выбирает для себя сам, что ему нравится. Любая компьютерная игра – это отдельный мир, который уникален по-своему. Игрок может не только наблюдать, а также принимать участие в действиях, пробовать себя в разных ролях.

Большинство людей во время игры отдыхают от повседневной и рутинной работы, просто хотят уйти от реальности в миры интересов, которые наполнены невероятные приключениями и ощущениями.

Одним из самых популярных жанров игр является платформер. Платформеры — это своеобразный, довольно легко узнаваемый жанр компьютерной игры. Он заключается в том, что у персонажа есть задача — пройти по разным уровням, преодолевая препятствия.

Платформеры часто характеризуются, но не определяются, тем, что они интенсивно используют прыжки и лазание, чтобы ориентироваться в среде игрока и достигать своей цели. Уровни и окружение, как правило, имеют неровную местность и подвешенные платформы разной высоты, что требует использования способностей персонажа игрока для перемещения. Самый распространенный объединяющий элемент игр этого жанра - умение прыгать.

В какой-то момент платформеры были самым популярным жанром видеоигр. На пике своей популярности, по оценкам, от четверти до одной трети консольных игр были платформерами, но с тех пор их вытеснили шутеры от первого лица. По состоянию на 2006 год этот жанр стал гораздо менее доминирующим, составляя два процента рынка по сравнению с пятнадцатью процентами в 1998 году; однако этот жанр все еще коммерчески жизнеспособен, и ряд игр продается миллионными тиражами. С 2010 года множество бесконечно работающих платформеров для мобильных устройств вернули этому жанру популярность.

1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Разработать игру-платформер, а именно аналог игры *Wall Kickers*.

Выбор на эту игру пал из-за её, в какой-то степени, уникальности. Сама техника игры зациклена на передвижениях по стенках. В обычных играх-платформерах стенки используются больше, как дополнительная возможность персонажа.

2. ОБЗОР ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ

Unity – межплатформенная среда разработки компьютерных игр, разработанная американской компанией Unity Technologies. Unity позволяет создавать приложения, работающие на более чем 25 различных платформах, включающих персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие. Основными преимуществами Unity являются наличие визуальной среды разработки, межплатформенной поддержки и модульной системы компонентов.

Visual Studio – это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений. Интегрированная среда разработки (IDE) представляет собой многофункциональную программу, которую можно использовать для различных аспектов разработки программного обеспечения.

Adobe Photoshop – многофункциональный графический редактор, разрабатываемый и распространяемый компанией Adobe Systems. В основном работает с растровыми изображениями, однако имеет некоторые векторные инструменты. Продукт является лидером рынка в области коммерческих средств редактирования растровых изображений и наиболее известной программой разработчика.

3. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

1. ***Схема головного модуля:***

Блок-схема головного модуля представлен в **ПРИЛОЖЕНИИ А.**

2. ***Диаграмма классов:***

Диаграмма представлена в **ПРИЛОЖЕНИИ Б.**

3. ***Use-case диаграмма:***

Диаграмма представлена в **ПРИЛОЖЕНИИ В.**

4 КОД ПРОГРАММЫ

Весь код программы представлен в виде C#-скриптов.

4.1 Код Hero.cs (движение игрока)

```
using System;
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;
using System.Collections.Generic;

public class Hero : MonoBehaviour
{
    [SerializeField] private float speed = 5f;
    [SerializeField] private float jumpForce = 5f;
    [SerializeField] private LayerMask groundLayer;
    [SerializeField] private LayerMask wallLayer;
    [SerializeField] private LayerMask timerWallLayer;

    private Rigidbody2D body;
    public static Animator animator;
    [SerializeField] private SpriteRenderer sprite;
    [SerializeField] private AudioSource sound;
    private BoxCollider2D box_collider;

    public bool doubleJump = true;
    private bool wasInAir = false;

    private int maxScoreValue = 0;
    private float timeLeft = 3;

    private IEnumerator coroutine;

    public states state
    {
        get { return (states)animator.GetInteger("state"); }
        set { animator.SetInteger("state", (int)value); }
    }


    private void Awake()
    {
        body = GetComponent<Rigidbody2D>();
        animator = GetComponent<Animator>();
        box_collider = GetComponent<BoxCollider2D>();
    }

    private void Update()
    {
        timeLeft -= Time.deltaTime;
        if (timeLeft < 0)
        {
            if (onTimerWall())
            {
                if (sprite.flipX)
                {
                    transform.position = new Vector2(transform.position.x - 0.6f,
transform.position.y);
                }
                else
                {
```

```
                    transform.position = new Vector2(transform.position.x + 0.6f,
transform.position.y);
                }

                sprite.flipX = !sprite.flipX;
            }
            timeLeft = 3;
        }

        checkScore();

        if (!checkHero())
        {
            state = states.death;
            Debug.Log("Death!!!!!!!!!!!!!!");

            coroutine = ExecuteAfterTime(1.5f);
            StartCoroutine(coroutine);
        }

        if (state == states.death || state == states.sit) return; // ??

        if (onGround()) state = states.idle;
        else if (onWall() || onTimerWall()) state = states.on_wall;
        else state = states.jump;

        if (!onWall() && !onTimerWall()) Physics2D.gravity = new Vector2(0f, -9.81f);

        if (!onGround() && !onWall() && !onTimerWall()) wasInAir = true;

        if (onGround() || onWall() || onTimerWall()) doubleJump = true;

        if (onWall() && wasInAir)
        {
            Flip();
            body.velocity = Vector2.zero;
            wasInAir = false;
            Physics2D.gravity = new Vector2(0f, -0.1f);
        }

        if (onTimerWall() && wasInAir)
        {
            Flip();
            body.velocity = Vector2.zero;
            wasInAir = false;
            Physics2D.gravity = Vector2.zero;
        }

        if (onGround() && Input.GetButton("Horizontal")) Run();
        if (Input.GetKeyDown(KeyCode.Space) && (doubleJump))
        {
            if (!onGround() && !onWall() && !onTimerWall()) doubleJump = false;
            sound.Play();
            Jump();
        }
    }

    private bool checkHero()
    {
        float difference = Camera.main.transform.position.y - (body.position.y + 7.81f);
        return ((int)difference < 6);
    }

    private void checkScore()
```

```
    {
        int cScore = (int)body.position.y + 10;
        cScore /= 2;
        maxScoreValue = Math.Max(maxScoreValue, cScore);
        ScoreSystem.scoreValue = maxScoreValue;
    }

    private void Run()
    {
        state = states.run;

        Vector3 distance = transform.right * Input.GetAxis("Horizontal");
        transform.position = Vector3.MoveTowards(transform.position, transform.position +
distance, speed * Time.deltaTime);

        sprite.flipX = distance.x < 0.0f;
    }

    public void Jump()
    {
        if (onGround())
        {
            if (sprite.flipX) body.velocity = new Vector2(-0.5f * jumpForce, jumpForce);
            else body.velocity = new Vector2(0.5f * jumpForce, jumpForce);
        }
        else if (onWall())
        {
            if (sprite.flipX) body.velocity = new Vector2(0.5f * jumpForce, jumpForce);
            else body.velocity = new Vector2(-0.5f * jumpForce, jumpForce);
            sprite.flipX = !sprite.flipX;
        }
        else
        {
            if (!sprite.flipX) body.velocity = new Vector2(-0.5f * jumpForce, jumpForce);
            else body.velocity = new Vector2(0.5f * jumpForce, jumpForce);
            sprite.flipX = !sprite.flipX;
        }
    }

    private bool onGround()
    {
        RaycastHit2D raycast_hit = Physics2D.BoxCast(box_collider.bounds.center,
box_collider.bounds.size, 0, Vector2.down, 0.1f, groundLayer);

        return (raycast_hit.collider != null);
    }

    private bool onWall()
    {
        RaycastHit2D raycast_hit1 = Physics2D.BoxCast(box_collider.bounds.center, new
Vector2(0.8425932f, 0.1f), 0, Vector2.right, 0.1f, wallLayer);
        RaycastHit2D raycast_hit2 = Physics2D.BoxCast(box_collider.bounds.center, new
Vector2(0.8425932f, 0.1f), 0, Vector2.left, 0.1f, wallLayer);

        return (raycast_hit1.collider != null) || (raycast_hit2.collider != null);
    }
    private bool onTimerWall()
    {
        RaycastHit2D raycast_hit1 = Physics2D.BoxCast(box_collider.bounds.center, new
Vector2(0.8425932f, 0.1f), 0, Vector2.right, 0.1f, timerWallLayer);
        RaycastHit2D raycast_hit2 = Physics2D.BoxCast(box_collider.bounds.center, new
Vector2(0.8425932f, 0.1f), 0, Vector2.left, 0.1f, timerWallLayer);

        return (raycast_hit1.collider != null) || (raycast_hit2.collider != null);
```

```
    }

    private void Flip()
    {
        if (!onWall()) return;
        RaycastHit2D raycast_hit1 = Physics2D.BoxCast(box_collider.bounds.center,
box_collider.bounds.size, 0, Vector2.right, 0.1f, wallLayer);
        RaycastHit2D raycast_hit2 = Physics2D.BoxCast(box_collider.bounds.center,
box_collider.bounds.size, 0, Vector2.left, 0.1f, wallLayer);

        if (raycast_hit1.collider != null) sprite.flipX = false;
        if (raycast_hit2.collider != null) sprite.flipX = true;

    }

    IEnumerator ExecuteAfterTime(float timeInSec)
    {
        yield return new WaitForSeconds(timeInSec);

        GameObject obj = GameObject.FindGameObjectWithTag("music");

        if (obj)
        {
            Destroy(obj);
        }

        SceneManager.LoadScene(0);
    }
}

public enum states { idle, run, jump, on_wall, death, sit }
```

4.2 Код CameraController.cs (контроль основной камеры)

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraController : MonoBehaviour
{
    [SerializeField] private Transform player;
    private Vector3 position;
    private float maxY = 0f;

    private void Awake() { if (!player) player = FindObjectOfType<Hero>().transform; }

    private void Update()
    {
        position = player.position;
        position.z = -10f;

        position.y = Math.Max(maxY, position.y + 7.82f);
        maxY = position.y;

        position.x += 2.37f;

        transform.position = Vector3.Lerp(transform.position, position, Time.deltaTime);
    }
}
```

4.3 Код ChangeScenes.cs (смена сцен игры)

```
using UnityEngine;
using UnityEngine.SceneManagement;
using System;

public class ChangeScenes : MonoBehaviour
{
    private int sceneNumber;

    void Update()
    {
        sceneNumber = SceneManager.GetActiveScene().buildIndex;
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        GameObject collisionGameObject = collision.gameObject;

        if (collisionGameObject.name == "Hero")
        {
            GetSceneNumber();

            if(sceneNumber==0)
            {
                GameObject obj = GameObject.FindGameObjectWithTag("music");

                if (obj)
                {
                    Destroy(obj);
                }
            }

            SceneManager.LoadScene(sceneNumber);
        }
    }

    void GetSceneNumber()
    {
        if (sceneNumber < 5) sceneNumber++;
        else
        {
            sceneNumber = 0;
            //System.Random rnd = new System.Random();
            //sceneNumber = rnd.Next(1, 4);
            //if (sceneNumber == SceneManager.GetActiveScene().buildIndex)  sceneNumber =
rnd.Next(1, 4);
        }
    }
}
```

4.4 Код Menu.cs (меню игры)

```
using UnityEngine;

public class Menu : MonoBehaviour
{
    public GameObject panel;
    [SerializeField] private AudioSource sound;

    public void Pause()
    {
        sound.Play();

        panel.SetActive(true);
        Time.timeScale = 0;
    }

    public void Continue()
    {
        sound.Play();

        panel.SetActive(false);
        Time.timeScale = 1f;
    }

    public void Exit()
    {
        sound.Play();

        Application.Quit();
    }
}
```

4.5 Код Leaper.cs (платформа-попрыгунчик)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Leaper : MonoBehaviour
{
    public GameObject player;
    private Rigidbody2D body;

    private void Start()
    {
        body = player.GetComponent<Rigidbody2D>();
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        GameObject collisionGameObject = collision.gameObject;

        if (collisionGameObject.name == "Hero")
        {
            player.GetComponent<Hero>().Jump();
            player.GetComponent<Hero>().doubleJump = true;
        }
    }
}
```

4.6 Код Thorns.cs (платформа-колючки)

```
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;
using System.Collections.Generic;

public class Thorns : MonoBehaviour
{
    private IEnumerator coroutine;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        GameObject collisionGameObject = collision.gameObject;

        if (collisionGameObject.name == "Hero")
        {
            Hero.animator.SetInteger("state", 4);
            Debug.Log("Death!!!!!!!!!!!!!!");

            coroutine = ExecuteAfterTime(1.5f);
            StartCoroutine(coroutine);
        }
    }

    IEnumerator ExecuteAfterTime(float timeInSec)
    {
        yield return new WaitForSeconds(timeInSec);

        GameObject obj = GameObject.FindGameObjectWithTag("music");

        if (obj)
        {
            Destroy(obj);
        }

        SceneManager.LoadScene(0);
    }

}
```

4.7 Код Sofa.cs (взаимодействие с диваном в финальной сцене)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Sofa : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        GameObject collisionGameObject = collision.gameObject;

        if (collisionGameObject.name == "Hero")
        {
            Hero.animator.SetInteger("state", 5);
        }
    }
}
```

4.8 Код VerticalThorn.cs (вертикальная ловушка)

```
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;
using System.Collections.Generic;

public class VerticalThorn : MonoBehaviour
{
    public float speed = 3f;
    bool thornMove = true;
    float minY;
    float maxY;

    private Rigidbody2D body;

    private void Start()
    {
        body = GetComponent<Rigidbody2D>();
        minY = transform.position.y - 6f;
        maxY = transform.position.y;
    }

    void Update()
    {
        if (transform.position.y > maxY)
        {
            thornMove = false;
        }
        else if (transform.position.y < minY)
        {
            thornMove = true;
        }


        if (thornMove)
        {
            transform.position = new Vector2(transform.position.x, transform.position.y +
speed * Time.deltaTime);
        }
        else
        {
            transform.position = new Vector2(transform.position.x, transform.position.y -
speed * Time.deltaTime);

        }
    }

    private IEnumerator coroutine;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        GameObject collisionGameObject = collision.gameObject;

        if (collisionGameObject.name == "Hero")
        {
            Hero.animator.SetInteger("state", 4);
            Debug.Log("Death!!!!!!!!!!!!!");

            coroutine = ExecuteAfterTime(1.5f);
            StartCoroutine(coroutine);
        }
    }
```

```
    IEnumerator ExecuteAfterTime(float timeInSec)
    {
        yield return new WaitForSeconds(timeInSec);

        GameObject obj = GameObject.FindGameObjectWithTag("music");

        if (obj)
        {
            Destroy(obj);
        }

        SceneManager.LoadScene(0);
    }
}
```

4.9 Код HorizontalThorn.cs (горизонтальная ловушка)

```
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;
using System.Collections.Generic;

public class HorizontalThorn : MonoBehaviour
{
    public float speed = 3f;
    bool thornMove = true;
    float minX;
    float maxX;

    private Rigidbody2D body;

    private void Start()
    {
        body = GetComponent<Rigidbody2D>();
        minX = transform.position.x;
        maxX = transform.position.x + 6f;
    }

    void Update()
    {
        if (transform.position.x > maxX)
        {
            thornMove = false;
        }
        else if (transform.position.x < minX)
        {
            thornMove = true;
        }


        if (thornMove)
        {
            transform.position = new Vector2(transform.position.x + speed *
Time.deltaTime, transform.position.y);
        }
        else
        {
            transform.position = new Vector2(transform.position.x - speed *
Time.deltaTime, transform.position.y);

        }
    }
```

```
    private IEnumerator coroutine;
    private void OnTriggerEnter2D(Collider2D collision)
    {
        GameObject collisionGameObject = collision.gameObject;

        if (collisionGameObject.name == "Hero")
        {
            Hero.animator.SetInteger("state", 4);
            Debug.Log("Death!!!!!!!!!!!!!!");

            coroutine = ExecuteAfterTime(1.5f);
            StartCoroutine(coroutine);
        }
    }

    IEnumerator ExecuteAfterTime(float timeInSec)
    {
        yield return new WaitForSeconds(timeInSec);

        GameObject obj = GameObject.FindGameObjectWithTag("music");

        if (obj)
        {
            Destroy(obj);
        }

        SceneManager.LoadScene(0);
    }
}
```

4.10 Код DontDestroy.cs (нейтрализация уничтожения объекта при смене сцен)

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class DontDestroy : MonoBehaviour
{
    void Start()
    {
        DontDestroyOnLoad(gameObject);
    }
}
```

5. РЕЗУЛЬТАТЫ

В результате мы получили игру, состоящую из 6 сцен: стартовая, 4 обычные и финальная.

В стартовой сцене находятся обычные блоки, чтобы пользователь понял всю технику игру. Далее с наступлением каждой сцены появляются новые блоки или ловушки. В первой – платформы-колючки, во второй— платформы-попрыгунчики, в третьей – вертикальные и горизонтальные ловушки, в четвёртой – поворачивающиеся платформы.

Смена сцен сопровождается новыми локациями и связана порталом, через который персонаж и попадает на другие уровни.

Финальная сцена даёт игроку выбор: отдохнуть или повторить игру ещё раз.

Движение персонажа включает: передвижение влево и вправо, прыжок наискосок (если на полу), прыжок со стенки на стенку, двойной прыжок при нажатии пробела при нахождении героя в воздухе.

Дизайн включает в себя: локации, спрайты различных предметов и блоков, анимация персонажа и других объектов, дополнительные объекты для декорации. Также присутствуют музыка, звуки при нажатии некоторых клавиш или кнопок, меню и небольшая инструкция для игрока.

ЗАКЛЮЧЕНИЕ

Так как игра создавалась на игровом движке Unity, а это, как мы знаем, межплатформенная среда разработки компьютерных игр, то уже по умолчанию игра создавалась для Windows, Mac, Linux. Если собрать проект для Android,будет работать и на нём.

Главной целью этой работы было создать игру, похожую по механике на игру Wall Kickers. Я считаю, что нам более чем удалось это сделать. Конечно, от багов игра не застрахована, как и сам оригинал, но работает она полноценно и имеет хороший внешний вид.

ПРИМЕНЕНИЕ

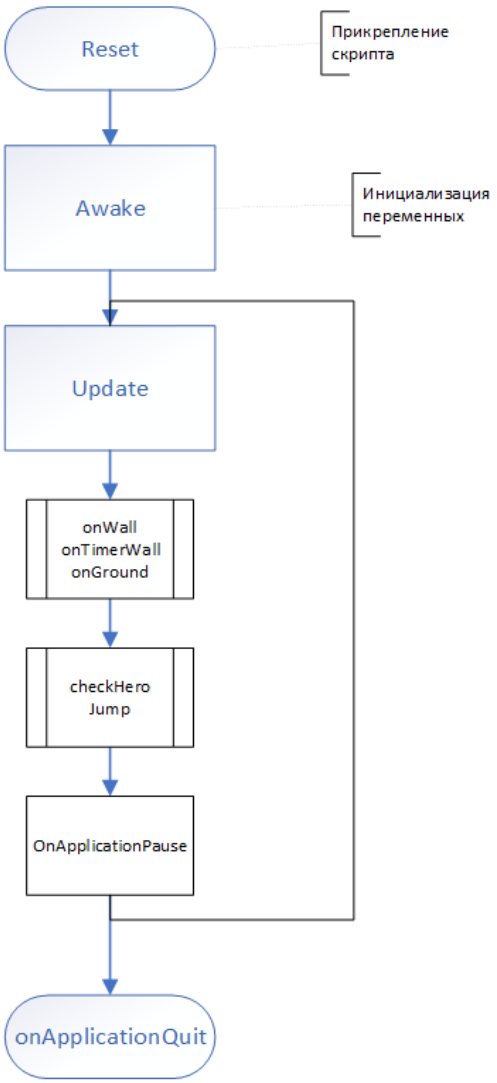
Хоть мы и знаем, что игры-платформеры были популярны очень давно, но они не утратили свою популярность. Сейчас начали появляться, как и старые добрые платформеры только с более хорошей графикой, которые отличаются простотой и позитивным настроением, так и платформеры с очень глубоким смыслом и интересным сюжетом.

Поэтому для любого человека найдётся, то, что он любит. Если всё-таки выберете классический вид платформера, то когда будете играть, разгрузите все свои мысли и отдохнёте.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Unity Documentation [Электронный ресурс]. – Режим доступа: https://docs.unity.com/
2. Unity 2D Platformer for Complete Beginners [Электронный ресурс]. – Режим доступа: https://www.youtube.com/playlist?list=PLgOEwFbvGm5o8hayFB6skAfa8Z-mw4dPV
3. Уроки Unity C# [Электронный ресурс]. – Режим доступа: https://www.youtube.com/playlist?list=PLDyJYA6aTY1mKdPdoxBW_UAiI-I7enTnb
4. Изучение движка Unity [Электронный ресурс]. – Режим доступа: https://www.youtube.com/playlist?list=PLDyJYA6aTY1k_-3fFiMVoYY04jCr-QY55
5. Интерфейсы IEnumerable и IEnumerator [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/tutorial/4.11.php
6. MonoBehaviour.StartCoroutine [Электронный ресурс]. – Режим доступа: https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html
7. Physics2D.gravity [Электронный ресурс]. – Режим доступа: https://docs.unity3d.com/ScriptReference/Physics2D-gravity.html

ПРИЛОЖЕНИЕ А.

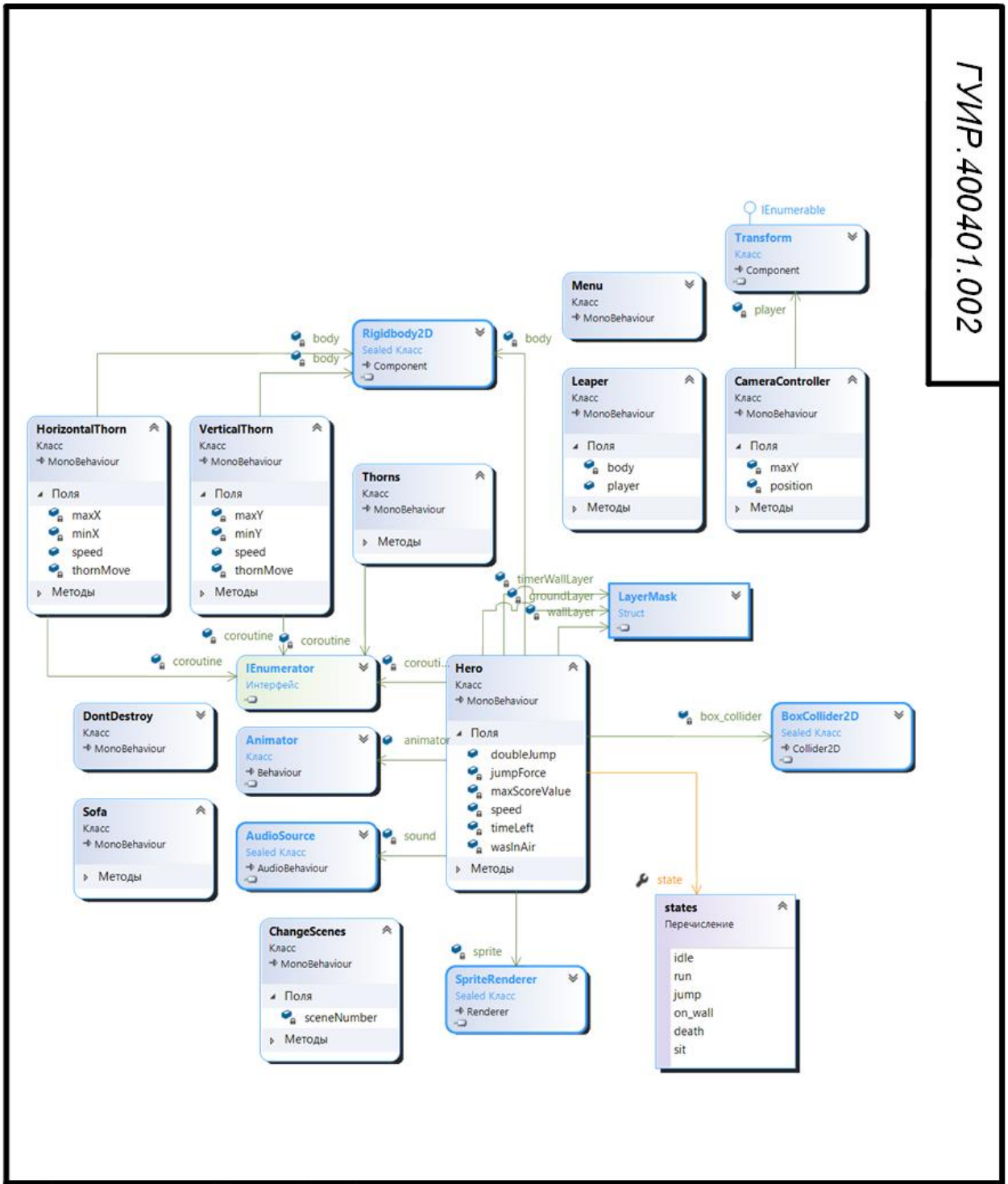


ГУИР.400401.001

					ГУИР.400401.001		
Изм.	Лист	№ докум.	Подп.	Дата			
Разраб.		Быбко			Схема головного модуля	Лит.	Масса / Масштаб
Пров.		Владымцев				У	1:1
						Лист	Листов 1
						ИиТП, гр. 153501	

Формат А4

ПРИЛОЖЕНИЕ Б.



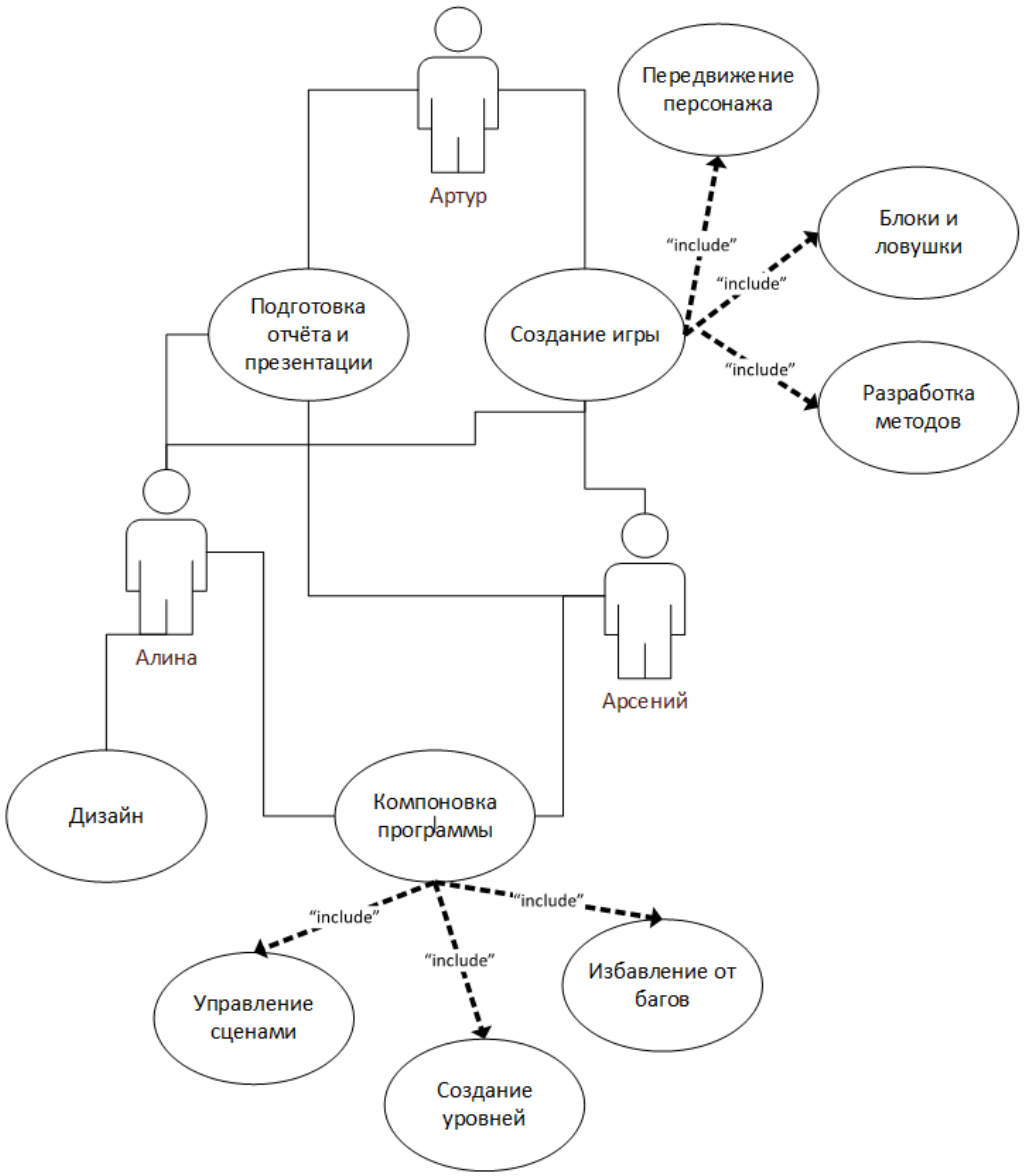
ГУИР.400401.002

Изм.	Лист	№ докум.	Подп.	Дата	ГУИР.400401.002		
Разраб.		Быбко			Диаграмма классов	Лит.	Масса / Масштаб
Пров.		Владымцев				У	1:1
						Лист	Листов 1
						ИиТП, гр. 153501	

Формат А4

ПРИЛОЖЕНИЕ В.

ГУИР.400401.003



					ГУИР.400401.003	
Изм.	Лист	№ докум.	Подп.	Дата		
Разраб.		Быбко			Use-case диаграмма	Лит. У / Масса / Масштаб 1:1
Пров.		Владымцев				Лист / Листов 1
						ИиТП, гр. 153501

Формат А4