Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный
университет»

УТВЕРЖДАЮ	СОГЛАСОВАНО
Декан ФКТ	Заведующий кафедрой ПМИ
_____Я.Ю. Григорьев	_____ С.А. Гордин
«____» _____ 2019 г.	«____» _____ 2019 г.

Программное обеспечение «Surf GO»

Руководитель СКБ Е.П. Жарикова

Подпись/дата

Ответственный исполнитель Д.А Плетнёв

Подпись/дата

Комсомольск-на-Амуре 2019

Карточка проекта

Название	Программное обеспечение «Surf GO»
Тип проекта	Инициативный (инициативный, по заказу, в рамках конкурса, учебная работа, другое)
Исполнители	Д.А Плетнёв – 9ВТб-1 ответственный исполнитель
Срок реализации	08.2019-10.2019 Месяц, год

Использованные программные средства

Наименование	**Версия**
Unity 3D	**2019.2.16**
C#	**7**

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

ЗАДАНИЕ

на разработку

Выдано студентам:
Д.А Плетнёв

Название проекта:

Программное обеспечение «Surf GO»

Назначение: Программное обеспечение «Surf GO» предназначено для игровой развлекательной деятельности.

Применение: в качестве отдыха и уменьшения стресса.

Функциональное описание Программного обеспечения:

Взаимодействие с интерфейсом происходит посредствам нажатия кнопок и движения пальцами по экрану мобильного устройства.

Требования: Программное обеспечение должно обеспечить решение следующих задач: реализацию комфортной игровой сессии, с звуковыми и графическими эффектами на мобильном устройстве под управлением операционной системы Android.

План работ:

Этап	Дата начала	Дата окончания
Формирование требований к программному обеспечению	15.08.19	21.08.19
Разработка структуры программного обеспечения	21.08.19	30.08.19
Разработка и утверждение технического задания	30.08.19	15.09.19
Программная реализация	15.10.19	20.10.19

Тестирование и отладка системы	21.10.19	23.10.19
Подготовка документации	23.08.19	25.10.19
Опытная эксплуатация	25.10.19	30.11.19

Руководитель СКБ Е.П. Жарикова

Подпись/дата

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный
университет»

Руководство администратора

Программное обеспечение «Surf GO»

Руководитель СКБ Е.П. Жарикова

Подпись/дата

Ответственный исполнитель Д.А Плетнёв

Подпись/дата

Комсомольск-на-Амуре 2019

Содержание

1 Общие положения

Настоящий документ представляет собой руководство администратора программного обеспечения «Surf GO» далее ПО.

Руководство определяет порядок установки, настройки и администрирования ПО.

Перед установкой и эксплуатацией системы рекомендуется внимательно ознакомиться с настоящим руководством.

Документ подготовлен в соответствии с РД 50-34.698-90 - в части структуры и содержания документов, и в соответствии с ГОСТ 34.201-89 - в части наименования и обозначения документов.

1.1 Наименование программы

Наименование программного продукта: программное обеспечение для игровой развлекательной деятельности «Surf GO».

1.2 Наименования документов, на основании которых ведется проектирование системы

Создание ПО осуществляется на основании требований и положений следующих документов:

- задание.

1.3 Перечень организаций, участвующих в разработке системы

Разработчики: студент группы 9ВТб-1 Комсомольского-на-Амуре государственного технического университета Д.А Плетнёв.

Заказчик: Комсомольский-на-Амуре государственный университет, студенческое конструкторское бюро «Интеллектуальные технологии».

1.4 Сведения об использованных при проектировании нормативно-технических документах

При проектировании использованы следующие нормативно-технические документы:

- ГОСТ 19.101-77 – виды программ и программных документов;

- ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом

- ГОСТ 19.201-78 Техническое задание, требования к содержанию и оформлению;

- ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению

- ГОСТ 2.004-88. Единая система конструкторской документации. Общие требования к выполнению конструкторских технологических документов на печатающих и графических устройствах вывода ЭВМ.

2 Назначение и принцип действия

2.1 Назначение изделия

Программное обеспечение «Surf GO» предназначено для развлечения человека в игровой форме на устройстве Android.

2.2 Области использования изделия

ПО может применяться в качестве приложения для снятия стресса и тренировки скорости реакции.

2.3 Принцип действия

Взаимодействие с интерфейсом происходит посредствам нажатия кнопок «Начать игру» выбора уровня и перемещения по уровню посредством движения пальцем по экрану смартфона.

3 Описание программного обеспечения

3.1 Описание логической структуры

Для игрового сеанса используется устройство Android, с рабочим apk файлом, созданным при помощи «Unity 3D».

Взаимодействие между компонентами происходит по схеме представленной на рисунке 3.1.

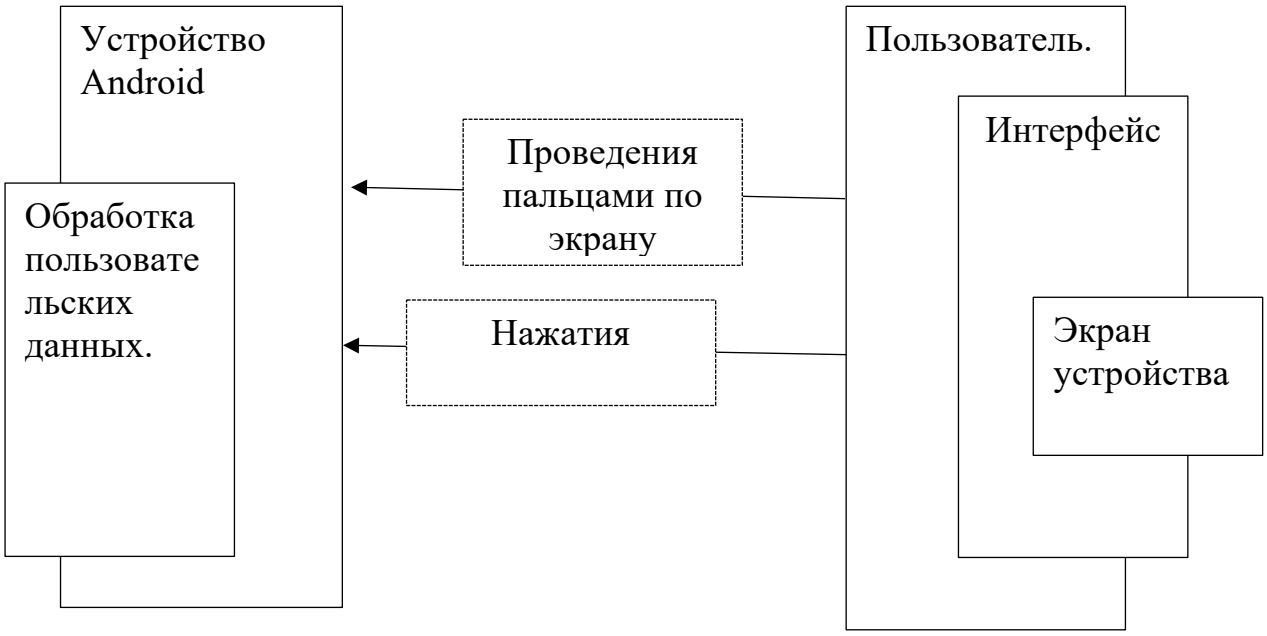

Рисунок 3.1 –схема работы ПО.

Взаимодействие с интерфейсом происходит посредствам нажатия кнопки «Начать игру» для начала игровой сессии и Перемещения пальцем по экрану для игры. Чтобы выйти необходимо нажать кнопку «Back» на устройстве.

Результат работы программы представлен на рисунке 3.2.

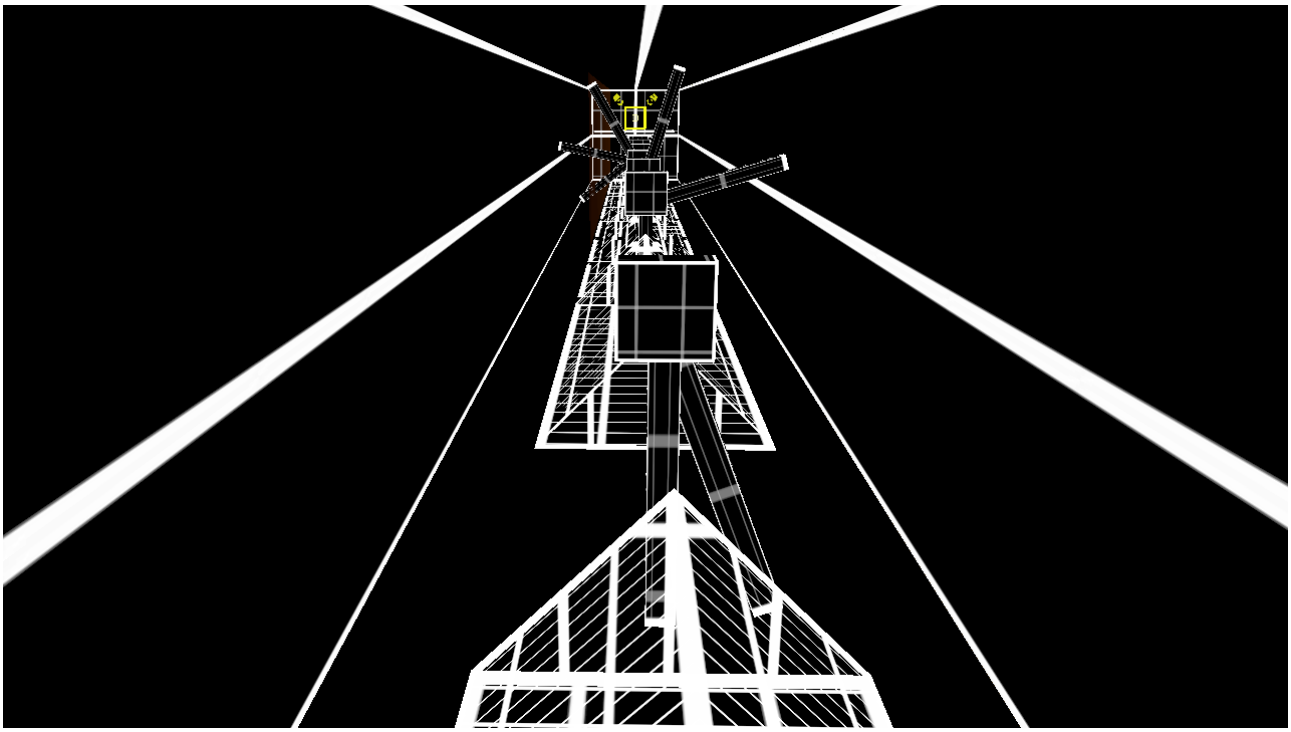

Рисунок 3.2 – игровой сеанс

3.3 Используемые технические средства

Техническое обеспечение необходимое для работы программы:

- оперативная память 512 Мб и выше;
- существование логических и/или физических дисков со свободным дисковым пространством 100 Мб и выше;
- устройство Android

3.4 Вызов и загрузка

Для функционирования программного обеспечения на устройстве Android должен быть установлен файл apk с игрой. Для вызова данного программного продукта «Surf GO» осуществляется запуск apk файла через нажатия на него.

3.5 Входные данные

Пользователь вводит данные посредством нажатия на. С помощью взаимодействия с интерфейсом определяет начало и окончание игрового сеанса.

3. 6 Выходные данные

Выходными данными программного обеспечения «Surf GO» является осуществление игрового сеанса.

4. ТЕКСТ ПРОГРАММЫ

Текст файла GameController.cs:

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameController : MonoBehaviour
{
    public static GameController instance;

    // Point where spawn player in start game
    public GameObject SpawnPoint;
    // Point where spawn player when bumps into an obstacle
    public GameObject SaveSpawnPoint;
    public Player Player;

    public GameObject TrainingWindow;


    // The window open when player win
    public WinWindow WinWindow;

    // The window open when player clicked pause button
    public GameObject WindowPause;
    public bool IsPause;

    private void Awake()
    {
        instance = this;
    }

    private void Start()
    {
        IsPause = true;
        SaveController.instance.CurrentAttempsOnLevel = 1;
        Time.timeScale = 1;

        SaveSpawnPoint = SpawnPoint;
        Player.transform.position = SpawnPoint.transform.position;

        if (!SaveController.instance.IsCompleteTraining)
        {
            TrainingWindow.SetActive(true);
        }
    }

    public void EndTraining()
    {
        TrainingWindow.SetActive(false);
        SaveController.instance.IsCompleteTraining = true;
    }

    public void BackInMenu()
    {
        AdInterstitial.instance.ShowInterstitial();
        SaveController.instance.CurrentAttempsOnLevel = 1;
        SaveController.instance.SaveAllDatas();
        SceneManager.LoadScene(0);
    }

    public void StartLevel()
    {
        UnPause();
    }
```

```
    public void RestartLevel()
    {
        if (WinWindow.gameObject.activeSelf)
        {
            // If player is win, when atempts restart
            SaveController.instance.CurrentAttempsOnLevel = 1;
        }
        AdInterstitial.instance.ShowInterstitial();

        SaveController.instance.SaveAllDatas();
        SceneManager.LoadScene(SaveController.instance.CurrentLevel);
    }

    // When player win level
    public void PlayerWin()
    {
        WinWindow.gameObject.SetActive(true);
        WinWindow.WinPlayer(SaveController.instance.CurrentAttempsOnLevel);
        // Pause
        IsPause = true;
        Time.timeScale = 0;
    }

    public void Pause()
    {
        WindowPause.SetActive(true);
        HudController.instance.ButtonStartLevel.SetActive(false);
        IsPause = true;
        Time.timeScale = 0;
    }

    public void UnPause()
    {
        WindowPause.SetActive(false);
        IsPause = false;
        Time.timeScale = 1;
    }
     }
```

Текст файла HUDController.cs:

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

public class HudController : MonoBehaviour
{
    public static HudController instance;

    public GameObject BoostIcon;
    public GameObject ButtonStartLevel;

    public Animator AnimationStageComplete;
    public Animator AnimationLoses;

    public GameObject WindowWin;
    public GameObject[] AllWindows;
    public bool IsStart;

    private void Awake()
    {
        instance = this;
    }

    private void Start()
    {
```

```
    if (SaveController.instance.ValueIsMegaBoostModeActive == 1)
    {
        BoostIcon.SetActive(true);
    } else
    {
        BoostIcon.SetActive(false);
    }
}

private void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        for (int i = 0; i < AllWindows.Length; i++)
        {
            if (AllWindows[i].activeSelf)
            {
                AllWindows[i].SetActive(false);

                if (Time.timeScale == 0 && !WindowWin.activeSelf)
                {
                    GameController.instance.UnPause();
                } else if (WindowWin.activeSelf)
                {
                    GameController.instance.BackInMenu();
                }
                return;
            }
        }
        GameController.instance.Pause();
    }
}

// Activate white window animation when player passed stage
public void ActivateWhiteWindow()
{
    AnimationStageComplete.Play("Animation Event");
}

// Activate white window animation when player loses
public void ActivateRedWindow()
{
    AnimationLoses.Play("Animation Event");
}

private void OnApplicationFocus(bool focus)
{
    if (!focus)
    {
        if (IsStart == true)
        {
            GameController.instance.Pause();
        } else
        {
            IsStart = true;
        }
    }
}

private void OnApplicationPause(bool pause)
{
    if (pause)
    {
        GameController.instance.Pause();
    }
}
```

Текст файла MenuController:

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

public class MenuController : MonoBehaviour
{
    public static MenuController instance;

    public GameObject ReviewWindow;
    public GameObject QuestionPlayServiciesWindow;
    public GameObject WindowExit;
    public GameObject[] AllWindows;
    public GameObject UIButtons;
    public GameObject UIGPS;

    private void Awake()
    {
        instance = this;
    }

    private void Start()
    {
        if (!SaveController.instance.IsGameReview &&
SaveController.instance.ValueStarsAllLevels >= 5)
        {
            ReviewWindow.SetActive(true);
        }
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            for (int i = 0; i < AllWindows.Length; i++)
            {
                if (AllWindows[i].activeSelf)
                {
                    AllWindows[i].SetActive(false);
                    UIButtons.SetActive(true);
                    UIGPS.SetActive(true);
                    return;
                }
            }
            WindowExit.SetActive(true);
        }
    }

    public void StartGame(int level)
    {
        if (SaveController.instance.IsCompleteTraining)
            AdInterstitial.instance.ShowInterstitial();
        SaveController.instance.CurrentLevel = level;
        SaveController.instance.CurrentAttempsOnLevel = 1;
        SaveController.instance.SaveAllDatas();
        SceneManager.LoadScene(level);
    }
}
```

Текст файла SaveController:

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

```
using UnityEngine;

public class SaveController : MonoBehaviour
{
    public static SaveController instance;

    // Names Save datas
    [Header("Names Save datas")]
    public string NameSettingsSensetivity;
    public string NameSettingsMusic;
    public string NameSettingsSounds;

    public string NameCurrentLevel;
    public string NameCurrentAttempsOnLevel;

    public string NameCountAttempsFor5Stars;
    public string NameCountAttempsFor4Stars;
    public string NameCountAttempsFor3Stars;
    public string NameCountAttempsFor2Stars;
    public string NameCountAttempsFor1Stars;

    public string NameIsCompleteTraining;
    public string NameIsGameReview;
    public string NameIsLogin;

    public string NameIsUnlockLevels;
    public string NameIsNotificationScheduled;

    // Level Stars
    public string NameStarsLevel1;
    public string NameStarsLevel2;
    public string NameStarsLevel3;
    public string NameStarsLevel4;
    public string NameStarsLevel5;
    public string NameStarsLevel6;
    public string NameStarsLevel7;

    public string NameAttemtsLevel1;
    public string NameAttemtsLevel2;
    public string NameAttemtsLevel3;
    public string NameAttemtsLevel4;
    public string NameAttemtsLevel5;
    public string NameAttemtsLevel6;
    public string NameAttemtsLevel7;

    // Mega boost mode
    public string NameIsMegaBoostModeActive;
    public string NameIsImmortalityModeActive;
    public string NameIsGodModeActive;
    public string NameValueMegaBoost;

    // Values Save datas
    [Header("Values Save datas")]
    [HideInInspector]
    public float ValueSettingsSensetivity;
    public float ValueSettingsMusic;
    public float ValueSettingsSounds;

    public int CurrentLevel;
    public int CurrentAttempsOnLevel;

    public int ValueCountEnergys;
    public float ValueTimeExit;

    public bool IsCompleteTraining;
    public bool IsGameReview;
    public bool IsLogin;
```

```
public int CountAttempsFor5Stars;
public int CountAttempsFor4Stars;
public int CountAttempsFor3Stars;
public int CountAttempsFor2Stars;
public int CountAttempsFor1Stars;

public int ValueStarsLevel1;
public int ValueStarsLevel2;
public int ValueStarsLevel3;
public int ValueStarsLevel4;
public int ValueStarsLevel5;
public int ValueStarsLevel6;
public int ValueStarsLevel7;

public int ValueStarsAllLevels;

public int ValueAttemptsLevel1;
public int ValueAttemptsLevel2;
public int ValueAttemptsLevel3;
public int ValueAttemptsLevel4;
public int ValueAttemptsLevel5;
public int ValueAttemptsLevel6;
public int ValueAttemptsLevel7;

// Mega boost mode
public int ValueIsMegaBoostModeActive;
public int ValueIsImmortalityModeActive;
public int ValueIsGodModeActive;
public float ValueMegaBoost;

public bool IsUnlockLevels;
public bool IsNotificationScheduled;


private void Awake()
{
    instance = this;

    InitializeDatasContinueGame();
}

// Initialize All data in start game if datas is null
public void InitializeDatasStartGame()
{
    SaveSystem.SetFloat(NameSettingsSensetivity, 0.25f);
    SaveSystem.SetFloat(NameSettingsMusic, 1);
    SaveSystem.SetFloat(NameSettingsSounds, 1);

    SaveSystem.SetInt(NameCurrentLevel, 0);
    SaveSystem.SetInt(NameCurrentAttempsOnLevel, 0);

    SaveSystem.SetBool(NameIsCompleteTraining, false);
    SaveSystem.SetBool(NameIsGameReview, false);
    SaveSystem.SetBool(NameIsLogin, false);

    SaveSystem.SetBool(NameIsUnlockLevels, false);
    SaveSystem.SetBool(NameIsNotificationScheduled, false);

    SaveSystem.SetInt(NameCountAttempsFor1Stars, 0);
    SaveSystem.SetInt(NameCountAttempsFor2Stars, 0);
    SaveSystem.SetInt(NameCountAttempsFor3Stars, 0);
    SaveSystem.SetInt(NameCountAttempsFor4Stars, 0);
    SaveSystem.SetInt(NameCountAttempsFor5Stars, 0);

    SaveSystem.SetInt(NameStarsLevel1, 0);
    SaveSystem.SetInt(NameStarsLevel2, 0);
```

```
        SaveSystem.SetInt(NameStarsLevel3, 0);
        SaveSystem.SetInt(NameStarsLevel4, 0);
        SaveSystem.SetInt(NameStarsLevel5, 0);
        SaveSystem.SetInt(NameStarsLevel6, 0);
        SaveSystem.SetInt(NameStarsLevel7, 0);

        SaveSystem.SetInt(NameAttemtsLevel1, 0);
        SaveSystem.SetInt(NameAttemtsLevel2, 0);
        SaveSystem.SetInt(NameAttemtsLevel3, 0);
        SaveSystem.SetInt(NameAttemtsLevel4, 0);
        SaveSystem.SetInt(NameAttemtsLevel5, 0);
        SaveSystem.SetInt(NameAttemtsLevel6, 0);
        SaveSystem.SetInt(NameAttemtsLevel7, 0);

        SaveSystem.SetInt(NameIsMegaBoostModeActive, 0);
        SaveSystem.SetInt(NameIsImmortalityModeActive, 0);
        SaveSystem.SetInt(NameIsGodModeActive, 0);
        SaveSystem.SetFloat(NameValueMegaBoost, 1);

        SaveSystem.SaveToDisk();
    }

    public void InitializeDatasContinueGame()
    {
        // If datas is null, then initialize their
        if (!SaveSystem.HasKey(NameSettingsSensetivity))
        {
            InitializeDatasStartGame();
        }
        if (!SaveSystem.HasKey(NameIsMegaBoostModeActive))
        {
            SaveSystem.SetInt(NameIsMegaBoostModeActive, 0);
            SaveSystem.SetInt(NameIsImmortalityModeActive, 0);
            SaveSystem.SetInt(NameIsGodModeActive, 0);
            SaveSystem.SetFloat(NameValueMegaBoost, 1);
        }

        ValueSettingsSensetivity = SaveSystem.GetFloat(NameSettingsSensetivity);
        ValueSettingsMusic = SaveSystem.GetFloat(NameSettingsMusic);
        ValueSettingsSounds = SaveSystem.GetFloat(NameSettingsSounds);

        CurrentLevel = SaveSystem.GetInt(NameCurrentLevel);
        CurrentAttempsOnLevel = SaveSystem.GetInt(NameCurrentAttempsOnLevel);

        IsCompleteTraining = SaveSystem.GetBool(NameIsCompleteTraining);
        IsGameReview = SaveSystem.GetBool(NameIsGameReview);
        IsLogin = SaveSystem.GetBool(NameIsLogin);

        IsUnlockLevels = SaveSystem.GetBool(NameIsUnlockLevels);
        IsNotificationScheduled = SaveSystem.GetBool(NameIsNotificationScheduled);

        CountAttempsFor1Stars = SaveSystem.GetInt(NameCountAttempsFor1Stars);
        CountAttempsFor2Stars = SaveSystem.GetInt(NameCountAttempsFor2Stars);
        CountAttempsFor3Stars = SaveSystem.GetInt(NameCountAttempsFor3Stars);
        CountAttempsFor4Stars = SaveSystem.GetInt(NameCountAttempsFor4Stars);
        CountAttempsFor5Stars = SaveSystem.GetInt(NameCountAttempsFor5Stars);

        ValueStarsLevel1 = SaveSystem.GetInt(NameStarsLevel1);
        ValueStarsLevel2 = SaveSystem.GetInt(NameStarsLevel2);
        ValueStarsLevel3 = SaveSystem.GetInt(NameStarsLevel3);
        ValueStarsLevel4 = SaveSystem.GetInt(NameStarsLevel4);
        ValueStarsLevel5 = SaveSystem.GetInt(NameStarsLevel5);
        ValueStarsLevel6 = SaveSystem.GetInt(NameStarsLevel6);
        ValueStarsLevel7 = SaveSystem.GetInt(NameStarsLevel7);

        ValueAttemptsLevel1 = SaveSystem.GetInt(NameAttemtsLevel1);
        ValueAttemptsLevel2 = SaveSystem.GetInt(NameAttemtsLevel2);
```

```
        ValueAttemptsLevel3 = SaveSystem.GetInt(NameAttemtsLevel3);
        ValueAttemptsLevel4 = SaveSystem.GetInt(NameAttemtsLevel4);
        ValueAttemptsLevel5 = SaveSystem.GetInt(NameAttemtsLevel5);
        ValueAttemptsLevel6 = SaveSystem.GetInt(NameAttemtsLevel6);
        ValueAttemptsLevel7 = SaveSystem.GetInt(NameAttemtsLevel7);

        ValueIsMegaBoostModeActive = SaveSystem.GetInt(NameIsMegaBoostModeActive);
        ValueIsImmortalityModeActive = SaveSystem.GetInt(NameIsImmortalityModeActive);
        ValueIsGodModeActive = SaveSystem.GetInt(NameIsGodModeActive);
        ValueMegaBoost = SaveSystem.GetFloat(NameValueMegaBoost);

        ValueStarsAllLevels = ValueStarsLevel1 + ValueStarsLevel2 + ValueStarsLevel3 +
ValueStarsLevel4 + ValueStarsLevel5 +
            ValueStarsLevel6 + ValueStarsLevel7;
    }

    public void SaveAllDatas()
    {
        SaveSystem.SetFloat(NameSettingsSensetivity, ValueSettingsSensetivity);
        SaveSystem.SetFloat(NameSettingsMusic, ValueSettingsMusic);
        SaveSystem.SetFloat(NameSettingsSounds, ValueSettingsSounds);

        SaveSystem.SetInt(NameCurrentLevel, CurrentLevel);
        SaveSystem.SetInt(NameCurrentAttempsOnLevel, 0);

        SaveSystem.SetBool(NameIsCompleteTraining, IsCompleteTraining);
        SaveSystem.SetBool(NameIsGameReview, IsGameReview);
        SaveSystem.SetBool(NameIsLogin, IsLogin);

        SaveSystem.SetBool(NameIsUnlockLevels, IsUnlockLevels);
        SaveSystem.SetBool(NameIsNotificationScheduled, IsNotificationScheduled);

        SaveSystem.SetInt(NameCountAttempsFor1Stars, CountAttempsFor1Stars);
        SaveSystem.SetInt(NameCountAttempsFor2Stars, CountAttempsFor2Stars);
        SaveSystem.SetInt(NameCountAttempsFor3Stars, CountAttempsFor3Stars);
        SaveSystem.SetInt(NameCountAttempsFor4Stars, CountAttempsFor4Stars);
        SaveSystem.SetInt(NameCountAttempsFor5Stars, CountAttempsFor5Stars);

        SaveSystem.SetInt(NameStarsLevel1, ValueStarsLevel1);
        SaveSystem.SetInt(NameStarsLevel2, ValueStarsLevel2);
        SaveSystem.SetInt(NameStarsLevel3, ValueStarsLevel3);
        SaveSystem.SetInt(NameStarsLevel4, ValueStarsLevel4);
        SaveSystem.SetInt(NameStarsLevel5, ValueStarsLevel5);
        SaveSystem.SetInt(NameStarsLevel6, ValueStarsLevel6);
        SaveSystem.SetInt(NameStarsLevel7, ValueStarsLevel7);

        SaveSystem.SetInt(NameAttemtsLevel1, ValueAttemptsLevel1);
        SaveSystem.SetInt(NameAttemtsLevel2, ValueAttemptsLevel2);
        SaveSystem.SetInt(NameAttemtsLevel3, ValueAttemptsLevel3);
        SaveSystem.SetInt(NameAttemtsLevel4, ValueAttemptsLevel4);
        SaveSystem.SetInt(NameAttemtsLevel5, ValueAttemptsLevel5);
        SaveSystem.SetInt(NameAttemtsLevel6, ValueAttemptsLevel6);
        SaveSystem.SetInt(NameAttemtsLevel7, ValueAttemptsLevel7);

        SaveSystem.SetInt(NameIsMegaBoostModeActive, ValueIsMegaBoostModeActive);
        SaveSystem.SetInt(NameIsImmortalityModeActive, ValueIsImmortalityModeActive);
        SaveSystem.SetInt(NameIsGodModeActive, ValueIsGodModeActive);
        SaveSystem.SetFloat(NameValueMegaBoost, ValueMegaBoost);

        SaveSystem.SaveToDisk();
    }

    private void OnApplicationFocus(bool focus)
    {
        if (!focus)
        {
            SaveAllDatas();
```

```
        }
    }
    private void OnApplicationPause(bool pause)
    {
        if (pause)
        {
            SaveAllDatas();
        }
    }

    private void OnApplicationQuit()
    {
        SaveAllDatas();
    }
}
}
```

Текст файла SettingsController:

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

public class SettingsController : MonoBehaviour
{
    public List<AudioSource> Musics;
    public List<AudioSource> Audios;

    private void Start()
    {
        // Find all sounds in scene
        AudioSource[] FindAudios = FindObjectsOfType<AudioSource>();
        for (int i = 0; i < FindAudios.Length; i++)
        {
            if (FindAudios[i].gameObject.tag == "Sound")
            {
                Audios.Add(FindAudios[i]);
            }
            else
            {
                Musics.Add(FindAudios[i]);
            }
        }

        for (int i = 0; i < Musics.Count; i++)
        {
            Musics[i].volume = SaveController.instance.ValueSettingsMusic;
        }

        for (int i = 0; i < Audios.Count; i++)
        {
            Audios[i].volume = SaveController.instance.ValueSettingsSounds;
        }

        gameObject.SetActive(false);
    }

    // Functions that change settings values;
    public void ChangeSensetivity(Slider Slider)
    {
        SaveController.instance.ValueSettingsSensetivity = Slider.value;
    }

    public void ChangeMusicVolume(Slider Slider)
    {
        SaveController.instance.ValueSettingsMusic = Slider.value;
```

```
        for(int i = 0; i < Musics.Count; i++)
        {
            Musics[i].volume = Slider.value;
        }
    }

    public void ChangeSoundsVolume(Slider Slider)
    {
        SaveController.instance.ValueSettingsSounds = Slider.value;
        for (int i = 0; i < Audios.Count; i++)
        {
            Audios[i].volume = Slider.value;
        }
    }
}
```

Текст файла WindowExitGameController:

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

public class WindowExitGameController : MonoBehaviour
{
    public void Exit()
    {
        Application.Quit();
    }
}
```

Текст файла LevelControl:

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

public class LevelControl : MonoBehaviour
{
    public int ValLevel;
    public int ValLevelAttempts;
    public int CountReceivedStars;

    public int NeedStars;
    public GameObject LockBlock;
    public TMPro.TextMeshProUGUI TextLockStars;

    public Image[] Stars;

    public Sprite IconEnableStar;
    public Sprite IconDisableStar;

    public TMPro.TextMeshProUGUI TextLevelAttempts;
    public TMPro.TextMeshProUGUI[] TextMinAttempts;
    public Animator[] AnimatorsStar;

    public int CountAttempsFor5Stars;
    public int CountAttempsFor4Stars;
    public int CountAttempsFor3Stars;
    public int CountAttempsFor2Stars;
    public int CountAttempsFor1Stars;

    public GameObject WindowUnlockLevels;
    public GameObject WindowSelectLevels;

    public void Start()
    {
```

```
        ActivateLevel();
    }

    public void ActivateLevel()
    {
        if (SaveController.instance.ValueStarsAllLevels < NeedStars &&
!SaveController.instance.IsUnlockLevels)
        {
            LockBlock.SetActive(true);
            TextLockStars.text = "<#FF1100>" +
SaveController.instance.ValueStarsAllLevels + "</color>"
                + "<#FFFFFF>" + " / " + "</color>"
                + NeedStars;
        } else
        {
            LockBlock.SetActive(false);
        }

        if (ValLevel == 1)
        {
            ValLevelAttempts = SaveController.instance.ValueAttemptsLevel1;
        }
        else if (ValLevel == 2)
        {
            ValLevelAttempts = SaveController.instance.ValueAttemptsLevel2;
        }
        else if (ValLevel == 3)
        {
            ValLevelAttempts = SaveController.instance.ValueAttemptsLevel3;
        }
        else if (ValLevel == 4)
        {
            ValLevelAttempts = SaveController.instance.ValueAttemptsLevel4;
        }
        else if (ValLevel == 5)
        {
            ValLevelAttempts = SaveController.instance.ValueAttemptsLevel5;
        }
        else if (ValLevel == 6)
        {
            ValLevelAttempts = SaveController.instance.ValueAttemptsLevel6;
        }
        else if (ValLevel == 7)
        {
            ValLevelAttempts = SaveController.instance.ValueAttemptsLevel7;
        }

        TextMinAttempts[0].text = CountAttempsFor1Stars.ToString();
        TextMinAttempts[1].text = CountAttempsFor2Stars.ToString();
        TextMinAttempts[2].text = CountAttempsFor3Stars.ToString();
        TextMinAttempts[3].text = CountAttempsFor4Stars.ToString();
        TextMinAttempts[4].text = CountAttempsFor5Stars.ToString();

        // Determinate count stars which received player
        if (ValLevelAttempts != 0 && ValLevelAttempts <= CountAttempsFor5Stars)
        {
            CountReceivedStars = 5;
            TextLevelAttempts.color = Color.yellow;
            TextMinAttempts[0].color = Color.yellow;
            TextMinAttempts[1].color = Color.yellow;
            TextMinAttempts[2].color = Color.yellow;
            TextMinAttempts[3].color = Color.yellow;
            TextMinAttempts[4].color = Color.yellow;
            AnimatorsStar[0].Play("Animation Star");
            AnimatorsStar[1].Play("Animation Star");
            AnimatorsStar[2].Play("Animation Star");
            AnimatorsStar[3].Play("Animation Star");
```

```
            AnimatorsStar[4].Play("Animation Star");
        }
        else if (ValLevelAttempts != 0 && ValLevelAttempts <= CountAttempsFor4Stars)
        {
            CountReceivedStars = 4;
            TextMinAttempts[0].color = Color.yellow;
            TextMinAttempts[1].color = Color.yellow;
            TextMinAttempts[2].color = Color.yellow;
            TextMinAttempts[3].color = Color.yellow;
            TextMinAttempts[4].color = Color.red;
            AnimatorsStar[0].Play("Animation Star");
            AnimatorsStar[1].Play("Animation Star");
            AnimatorsStar[2].Play("Animation Star");
            AnimatorsStar[3].Play("Animation Star");
        }
        else if (ValLevelAttempts != 0 && ValLevelAttempts <= CountAttempsFor3Stars)
        {
            CountReceivedStars = 3;
            TextMinAttempts[0].color = Color.yellow;
            TextMinAttempts[1].color = Color.yellow;
            TextMinAttempts[2].color = Color.yellow;
            TextMinAttempts[3].color = Color.red;
            TextMinAttempts[4].color = Color.red;
            AnimatorsStar[0].Play("Animation Star");
            AnimatorsStar[1].Play("Animation Star");
            AnimatorsStar[2].Play("Animation Star");
        }
        else if (ValLevelAttempts != 0 && ValLevelAttempts <= CountAttempsFor2Stars)
        {
            CountReceivedStars = 2;
            TextMinAttempts[0].color = Color.yellow;
            TextMinAttempts[1].color = Color.yellow;
            TextMinAttempts[2].color = Color.red;
            TextMinAttempts[3].color = Color.red;
            TextMinAttempts[4].color = Color.red;
            AnimatorsStar[0].Play("Animation Star");
            AnimatorsStar[1].Play("Animation Star");
        }
        else if (ValLevelAttempts != 0 && ValLevelAttempts <= CountAttempsFor1Stars)
        {
            CountReceivedStars = 1;
            TextMinAttempts[0].color = Color.yellow;
            TextMinAttempts[1].color = Color.red;
            TextMinAttempts[2].color = Color.red;
            TextMinAttempts[3].color = Color.red;
            TextMinAttempts[4].color = Color.red;
            AnimatorsStar[0].Play("Animation Star");
        }
        else
        {
            CountReceivedStars = 0;
            TextLevelAttempts.color = Color.red;
            TextMinAttempts[0].color = Color.red;
            TextMinAttempts[1].color = Color.red;
            TextMinAttempts[2].color = Color.red;
            TextMinAttempts[3].color = Color.red;
            TextMinAttempts[4].color = Color.red;
        }
        if (ValLevelAttempts == 0)
        {
            TextLevelAttempts.text = "-";
        }
        else
        {
            TextLevelAttempts.text = ValLevelAttempts.ToString();
        }
```

```
        // Set Up stars in win window
        for (int i = 0; i < CountReceivedStars; i++)
        {
            Stars[i].sprite = IconEnableStar;
        }
    }

    public void StartLevel()
    {
        SaveController.instance.CountAttempsFor1Stars = CountAttempsFor1Stars;
        SaveController.instance.CountAttempsFor2Stars = CountAttempsFor2Stars;
        SaveController.instance.CountAttempsFor3Stars = CountAttempsFor3Stars;
        SaveController.instance.CountAttempsFor4Stars = CountAttempsFor4Stars;
        SaveController.instance.CountAttempsFor5Stars = CountAttempsFor5Stars;

        if (SaveController.instance.ValueStarsAllLevels >= NeedStars ||
SaveController.instance.IsUnlockLevels)
        {
            MenuController.instance.StartGame(ValLevel);
        } else
        {
            // block level
            WindowUnlockLevels.SetActive(true);
            WindowSelectLevels.SetActive(false);
        }

        SaveController.instance.SaveAllDatas();
    }
}
```

Текст файла Transition:

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

public class Transition : MonoBehaviour
{
    public GameObject CurrentStage;
    public GameObject NextStage;
    public GameObject PointTransition;

    // Set new max speed on next stage
    public float NewMaxSpeedForPlayer;
    // if this transition is finish
    public bool IsFinish;

    public void Transit()
    {
        if (NewMaxSpeedForPlayer != 0)
        {
            GameController.instance.Player.MaxSpeed = NewMaxSpeedForPlayer;
            if (SaveController.instance.ValueIsMegaBoostModeActive == 1)
            {
                GameController.instance.Player.MaxSpeed *=
SaveController.instance.ValueMegaBoost;
            }
        }
        HudController.instance.ActivateWhiteWindow();
        CurrentStage.SetActive(false);
        NextStage.SetActive(true);
    }
    public void Finish()
    {
        HudController.instance.ActivateWhiteWindow();
        GameController.instance.PlayerWin();
    }
}
```

Текст файла Player:

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

public class Player : MonoBehaviour
{
    public float MaxSpeed;
    private float CurrentSpeed;
    public float StepUpSpeed;

    public AudioSource SoundDeath;
    public AudioSource SoundKnock;
    public AudioSource SoundTeleport;
    public AudioSource SoundWin;

    public float Gravity;
    public float PermissibleGravity;
    // Player should be on the platform and not fly into the air
    private bool IsOnPlatform;
    private bool IsHeadOnPlatform;
    private bool IsTransit;

    [HideInInspector]
    public float CurrentMagnitudeSpeed;
    private Vector3 CurrentPlatformOnView;

    public Camera Camera;
    private Rigidbody Rigidbody;

    private void Awake()
    {
        Rigidbody = GetComponent<Rigidbody>();
    }

    private void Start()
    {
        if (SaveController.instance.ValueIsMegaBoostModeActive == 1)
        {
            StepUpSpeed *= SaveController.instance.ValueMegaBoost;
            MaxSpeed *= SaveController.instance.ValueMegaBoost;
        }
    }

    private void FixedUpdate()
    {
        if (!GameController.instance.IsPause)
            ChangeSpeedPlayer();
    }

    /*public void CheckViewOnPlatform()
    {
        RaycastHit hit;
        Ray ray = Camera.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0f));

        if (Physics.Raycast(ray, out hit))
        {
            CurrentPlatformOnView = hit.point;

            // Player looks at the surface
            if (hit.transform.CompareTag("Untagged"))
            {
```

```
            }
        }
    } */

    private void ChangeSpeedPlayer()
    {
        if (IsTransit)
        {
            IsTransit = false;
            IsOnPlatform = false;
            IsHeadOnPlatform = false;
        }

        if (SaveController.instance.ValueIsMegaBoostModeActive == 1 &&
SaveController.instance.ValueIsGodModeActive == 1)
        {
            // Gradual speed increase
            if (CurrentSpeed < MaxSpeed)
            {
                CurrentSpeed += StepUpSpeed * 1.15f;
            }
            else
            {
                CurrentSpeed = MaxSpeed;
            }
        }

        if (SaveController.instance.ValueIsImmortalityModeActive == 1)
        {
            IsHeadOnPlatform = false;
        }

        if (IsHeadOnPlatform)
            IsOnPlatform = false;

        // Player move in direction camera
        if ((IsOnPlatform || (!IsOnPlatform && Camera.transform.forward.y * CurrentSpeed
< Rigidbody.velocity.y) && !IsHeadOnPlatform)
            || (SaveController.instance.ValueIsMegaBoostModeActive == 1 &&
SaveController.instance.ValueIsGodModeActive == 1))
        {
            Rigidbody.velocity = new Vector3(Camera.transform.forward.x * CurrentSpeed,
Camera.transform.forward.y * CurrentSpeed, Camera.transform.forward.z * CurrentSpeed);

            // Gradual speed increase
            if (CurrentSpeed < MaxSpeed)
            {
                CurrentSpeed += StepUpSpeed;
            }
            else
            {
                CurrentSpeed = MaxSpeed;
            }
        }
        else if (!IsOnPlatform && Camera.transform.forward.y * CurrentSpeed >
Rigidbody.velocity.y && !IsHeadOnPlatform)
        {
            // Gravity gradually pulls the player down, if he does not fall to the bottom

            Rigidbody.velocity = new Vector3(Camera.transform.forward.x * CurrentSpeed,
Rigidbody.velocity.y, Camera.transform.forward.z * CurrentSpeed);
        }
        else if (!IsOnPlatform && Camera.transform.forward.y * CurrentSpeed <
Rigidbody.velocity.y)
        {
            // Gradual speed decrease
```

```
            if (CurrentSpeed > MaxSpeed / 3)
            {
                CurrentSpeed -= StepUpSpeed;
            }
        }
        if (SaveController.instance.ValueIsGodModeActive != 1 ||
(SaveController.instance.ValueIsGodModeActive == 1 &&
SaveController.instance.ValueIsMegaBoostModeActive == 0))
        {
            if (Rigidbody.velocity.y > PermissibleGravity)
                Rigidbody.AddForce(new Vector3(0, -Gravity, 0));
            if (IsHeadOnPlatform)
                Rigidbody.AddForce(new Vector3(0, -Gravity * 10, 0));
        }
    }



    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag("Platform"))
        {
            if (!IsHeadOnPlatform)
            {
                IsOnPlatform = true;
                SoundKnock.PlayOneShot(SoundKnock.clip);
            }
        }
        else if (collision.gameObject.CompareTag("Obstacle") &&
(SaveController.instance.ValueIsImmortalityModeActive != 1
            || (SaveController.instance.ValueIsImmortalityModeActive == 1 &&
SaveController.instance.ValueIsMegaBoostModeActive == 0)))
        {
            SaveController.instance.CurrentAttempsOnLevel++;

            // Show ad interstitial
            if (SaveController.instance.CurrentAttempsOnLevel % 3 == 0 &&
AdInterstitial.instance.IsLoaded())
                AdInterstitial.instance.ShowInterstitial();
            else
                HudController.instance.ActivateRedWindow();

            // Unlock achivement
            if (Application.isMobilePlatform)

PlayGamesController.instance.UnlockStepsAchivement(GPGSIds.achievement_experience_with_pa
in, 1);

            // Player lose
            SoundDeath.PlayOneShot(SoundDeath.clip);

            IsOnPlatform = false;
            CurrentSpeed = 0;
            Camera.transform.localRotation =
GameController.instance.SaveSpawnPoint.transform.localRotation;
            transform.position =
GameController.instance.SaveSpawnPoint.transform.position;
        }
        else if (collision.gameObject.CompareTag("Transition"))
        {
            IsOnPlatform = false;
            IsHeadOnPlatform = false;
            IsTransit = true;

            Transition Transition = collision.gameObject.GetComponent<Transition>();

            if (Transition.IsFinish)
```

```
            {
                SoundWin.PlayOneShot(SoundWin.clip);
                Transition.Finish();
            }
            else
            {
                SoundTeleport.PlayOneShot(SoundTeleport.clip);
                // When player to go over in next zone, speed reset to zero
                CurrentSpeed = 0;
                Transition.Transit();
                // Player Teleported in next zone
                transform.position = Transition.PointTransition.transform.position;
                Camera.transform.localRotation =
Transition.PointTransition.transform.localRotation;
                GameController.instance.SaveSpawnPoint = Transition.PointTransition;
            }
        }
    }

    private void OnCollisionStay(Collision collision)
    {
        if (!IsHeadOnPlatform && collision.gameObject.CompareTag("Platform"))
        {
            // Player should be on the platform and not fly into the air
            IsOnPlatform = true;

        }
    }

    private void OnCollisionExit(Collision collision)
    {
        if (collision.gameObject.CompareTag("Platform"))
        {
            // Player should be on the platform and not fly into the air
            IsOnPlatform = false;
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Platform"))
        {
            // Player do not should touch head on platform
            IsHeadOnPlatform = true;
            IsOnPlatform = false;
        }
    }

    private void OnTriggerStay(Collider other)
    {
        if (other.CompareTag("Platform"))
        {
            // Player do not should touch head on platform
            IsHeadOnPlatform = true;
            IsOnPlatform = false;
        }
        else if (other.CompareTag("Wall"))
        {
            if (SaveController.instance.ValueIsGodModeActive != 1 ||
(SaveController.instance.ValueIsGodModeActive == 1 &&
SaveController.instance.ValueIsMegaBoostModeActive == 0))
            {
                Rigidbody.AddForce(new Vector3(0, -Gravity * 5, 0));
            }
        }
    }
```

```
    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Platform"))
        {
            // Player do not should touch head on platform
            IsHeadOnPlatform = false;
        }
    }
}
```

Текст файла TouchControl:

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

public class TouchControl : MonoBehaviour
{
    public GameObject Player;

    private float rotSpeedMouse;
    private bool IsDownMouse;

    private float rotSpeedTouch;

    private float xAngle;
    private float yAngle;
    private Vector2 TouchDist;
    private Vector2 PointerOld;

    private bool IsStartTouch;

    void Start()
    {
        xAngle = 0;
        yAngle = 0;
        IsStartTouch = false;
    }

    void Update()
    {
        // Set settings values
        rotSpeedMouse = SaveController.instance.ValueSettingsSensetivity;
        rotSpeedTouch = SaveController.instance.ValueSettingsSensetivity;

        if (!GameController.instance.IsPause && Input.touchCount > 0)
        {
            TouchPhase phase = Input.GetTouch(0).phase;
            // Whether the finger that turns the camera
            if (phase == TouchPhase.Began)
            {
                PointerOld = Input.GetTouch(0).position;
                IsStartTouch = true;
            }
            else if (phase == TouchPhase.Moved && IsStartTouch)
            {


                TouchDist = (Input.GetTouch(0).position - PointerOld);
                PointerOld = Input.GetTouch(0).position;

                //Rotate player and camera
                transform.Rotate(-TouchDist.y * rotSpeedTouch, TouchDist.x *
rotSpeedTouch, 0);
```

```
                xAngle = transform.rotation.eulerAngles.x;
                yAngle = transform.rotation.eulerAngles.y;

                transform.rotation = Quaternion.Euler(xAngle, yAngle, 0);
            }
            else if (phase == TouchPhase.Ended)
            {
                TouchDist = new Vector2();
                IsStartTouch = false;
            }
        }

        if (!GameController.instance.IsPause && !Application.isMobilePlatform &&
Input.GetMouseButton(0))
        {
            transform.Rotate(new Vector3(Input.GetAxis("Mouse Y") * rotSpeedMouse, -
Input.GetAxis("Mouse X") * rotSpeedMouse, 0));
            xAngle = transform.rotation.eulerAngles.x;
            yAngle = transform.rotation.eulerAngles.y;
            transform.rotation = Quaternion.Euler(xAngle, yAngle, 0);
        }
    }
}
```

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный
университет»

УТВЕРЖДАЮ	СОГЛАСОВАНО
Декан ФКТ	Заведующий кафедрой ПМИ
_____Я.Ю. Григорьев	_____ С.А. Гордин
«____» _____ 2019 г.	«____» _____ 2019 г.

АКТ

о приемке в эксплуатацию программного обеспечения

«Surf GO»

г. Комсомольск-на-Амуре « » _____ 2019 г.

Комиссия в составе представителей:
заказчика Е.П. Жариковой – руководителя СКБ ФКТ, С.А. Гордина – Заведующего кафедрой ПМИ, исполнителя Д.А Плетнёв – 9ВТб-1
составила акт о нижеследующем: «Исполнитель» передает программное обеспечение «Surf GO»
Программное обеспечение «Surf GO» прошло опытную эксплуатацию
с « » _____ по « » _____ 2019г. и признано годным к эксплуатации. Были протестированы все режимы функционирования, отказы системы, а также аварийные отключения по вине системы не наблюдались.

Руководитель СКБ	Ответственный исполнитель
_____/Е.П. Жарикова /	_____ / Д.А Плетнёв /

Таблица учета работ

Дисциплина	Форма учтенной работы (номер ЛР, КП, КР, РГЗ, зачет, итоговая оценка , экзамен)	Преподаватель (дата, ФИО, подпись)	Примечание (ЗУН полученные при выполнении проекта)