Министерство образования и науки Российской Федерации

Московский авиационный институт (национальный исследовательский университет)

ЖУРНАЛ

ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Наименование практики: *вычислительная*
Студенты: В. П. Будникова, И. С. Глушатов
Факультет №8, курс 2, группа 7

Практика с 29.06.21 по 12.07.21

Москва, 2021

ИНСТРУКЦИЯ

о заполнении журнала по производственной практике

Журнал по производственной практике студентов имеет единую форму для всех видов практик.

Задание в журнал вписывается руководителем практики от института в первые три-пять дней пребывания студентов на практике в соответствии с тематикой, утверждённой на кафедре до начала практики. Журнал по производственной практике является основным документом для текущего и итогового контроля выполнения заданий, требований инструкции и программы практики.

Табель прохождения практики, задание, а также технический отчёт выполняются каждым студентом самостоятельно.

Журнал заполняется студентом непрерывно в процессе прохождения всей практики и регулярно представляется для просмотра руководителям практики. Все их замечания подлежат немедленному выполнению.

В разделе «Табель прохождения практики» ежедневно должно быть указано, на каких рабочих местах и в качестве кого работал студент. Эти записи проверяются и заверяются цеховыми руководителями практики, в том числе мастерами и бригадирами. График прохождения практики заполняется в соответствии с графиком распределения студентов по рабочим местам практики, утверждённым руководителем предприятия. В разделе «Рационализаторские предложения» должно быть приведено содержание поданных в цехе рационализаторских предложений со всеми необходимыми расчётами и эскизами. Рационализаторские предложения подаются индивидуально и коллективно.

Выполнение студентом задания по общественно-политической практике заносятся в раздел «Общественно-политическая практика». Выполнение работы по оказанию практической помощи предприятию (участие в выполнении спецзаданий, работа сверхурочно и т.п.) заносятся в раздел журнала «Работа в помощь предприятию» с последующим письменным подтверждением записанной работы соответствующими цеховыми руководителями. Раздел «Технический отчёт по практике» должен быть заполнен особо тщательно. Записи необходимо делать чернилами в сжатой, но вместе с тем чёткой и ясной форме и технически грамотно. Студент обязан ежедневно подробно излагать содержание работы, выполняемой за каждый день. Содержание этого раздела должно отвечать тем конкретным требованиям, которые предъявляются к техническому отчёту заданием и программой практики. Технический отчёт должен показать умение студента критически оценивать работу данного производственного участка и отразить, в какой степени студент способен применить теоретические знания для решения конкретных производственных задач.

Иллюстративный и другие материалы, использованные студентом в других разделах журнала, в техническом отчёте не должны повторяться, следует ограничиваться лишь ссылкой на него. Участие студентов в производственно-технической конференции, выступление с докладами, рационализаторские предложения и т.п. должны заноситься на свободные страницы журнала.

Примечание. Синьки, кальки и другие дополнения к журналу могут быть сделаны только с разрешения администрации предприятия и должны подшиваться в конце журнала.

Руководители практики от института обязаны следить за тем, чтобы каждый цеховой руководитель практики перед уходом студентов из данного цеха в другой цех вписывал в журнал студента отзывы об их работе в цехе.

Текущий контроль работы студентов осуществляется руководители практики от института и цеховыми руководителями практики заводов. Все замечания студентам руководители делают в письменном виде на страницах журнала, ставя при этом свою подпись и дату проверки.

Результаты защиты технического отчёта заносятся в протокол и одновременно заносятся в ведомость и зачётную книжку студента.

Примечание. Нумерация чистых страниц журнала проставляется каждым студентом в своём журнале до начала практики.

С инструкцией о заполнении журнала ознакомились:

« » _____ 2021 г. Студент Будникова В. П. _____
(дата) (подпись)

« » _____ 2021 г. Студент Глушатов И. С. _____
(дата) (подпись)

ЗАДАНИЕ

кафедры 806 по вычислительной практике: научиться работать с межплатформенной средой разработки Unity 3d, освоить инструменты написания скриптов, работы с UI, добавления 3d-моделей объектов на сцену, их взаимодействия, и научиться планировать и составлять графы связи объектов друг с другом.

Руководитель практики от института:

« » _____ 2021 г. Кухтичев А. А. _____

(дата) (подпись)

ТАБЕЛЬ ПРОХОЖДЕНИЯ ПРАКТИКИ

Дата	**Содержание или наименование проделанной работы**	**Место работы**	**Время работы**		**Подпись цехового руководителя**
			Начало	Конец	
29.06.2021	Получение задания	МАИ	9:00	18:00	
01.07.2021	Освоили интерфес игрового движка, научились добавлять объекты на сцену, написали первые скрипты	МАИ	9:00	18:00	
02.07.2021	Планирование игрового мира, механики игры, способов взаимодействия игрока с персонажем	МАИ	9:00	18:00	
03.07.2021	Написание первых вариантов анимаций камеры и движения персонажа	МАИ	9:00	18:00	
04.07.2021	Работа с 3d-редактором объектов, добавление на сцену собственных 3d-моделей	МАИ	9:00	18:00	
05.07.2021	Проектирование первой головоломки. Добавление механики открывания дверей, знакомство с физическим взаимодействием объектов в Unity через скрипты	МАИ	9:00	18:00	
06.07.2021	Знакомство с Unity UI, добавление простого меню игры	МАИ	9:00	18:00	
07.07.2021	Проектирование второй комнаты с головоломками. Создание мини-игры "пятнашки"и головоломки со светом, написание их логики	МАИ	9:00	18:00	
09.07.2021	Моделирование дополнительной механики: стреляющей шляпки. Добавление механики ввода кода и подключения UI	МАИ	9:00	18:00	

Продолжение на следующей странице

Дата	**Содержание или наименование проделанной работы**	**Место работы**	**Время работы**		**Подпись цехового руководителя**
			Начало	Конец	
10.07.2021	Добавление возможности стрелять, создание третьей комнаты с врагами, возможность их уничтожения	МАИ	9:00	18:00	
11.07.2021	Тестирование, отладка, профайлинг и оптимизация написанных скриптов	МАИ	9:00	18:00	
12.07.2021	Сдача журнала	МАИ	9:00	18:00	

Отзывы цеховых руководителей практики

Презентация защищена на комиссии кафедры 806. Работа выполнена в полном объёме. Рекомендую на оценку « ». Все материалы сданы на кафедру.

ПРОТОКОЛ

ЗАЩИТЫ ТЕХНИЧЕСКОГО ОТЧЁТА

по *производственной практике*

студентами: Будниковой Валерией Павловной и Глушатовым Игорем Сергеевичем

Слушали:	**Постановили:**
Отчёт практиканта	считать практику выполненной и защищённой на Общая оценка: _____

Руководители: Зайцев В. Е. _____

Кухтичев А. А. _____

Дата: 12 июля 2021 г.

МАТЕРИАЛЫ ПО РАЦИОНАЛИЗАТОРСКИМ ПРЕДЛОЖЕНИЯМ

Научится лучше планировать взаимосвязи скриптов и объектов. Обрисовать и добавить дизайн игры, проработать меню, паузу игры, возможность сохранения игры. Проработать карту уровней.

ТЕХНИЧЕСКИЙ ОТЧЁТ ПО ПРАКТИКЕ

Архитектура

Архитектура проекта поддерживается самим движком Unity и инкапсулированна в нём. Каждый объект состоит из компонентов. Базовыми являются: координаты, коллайдеры, mesh-свойства, физические компоненты и т.д. Пользователь может писать собственные скрипты и прикреплять их в качестве новых компонент, а также редактировать свойства своих и базовых компонент.

Описание

Взаимодействие с пользователем осуществляется через устройства ввода (клавиатуру и мышь). Пользователь может управлять положением камеры, закрепленной за протагонистом, а также его положением в пространстве. Игрок может взаимодействовать с объектами игрового мира. За такое взаимодействие отвечают скрипты, написанные на языке C#. Данные скрипты обрабатывают ввод данных с клавиатуры и мыши, после чего просчитывается физические взаимодействия с объектами игрового мира.

Реализация

SPlayerMoving.cs Движение игрока

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

public class SPlayerMoving : MonoBehaviour
{
    public Transform target_rotation;

    public float player_speed = 0.7f;
    public float jump_size = 0.5f;
    public float max_jump_height = 2.0f;
    public float rotation_speed = 0.1f;

    private Rigidbody _player_rigidbody;

    void Start()
    {
        _player_rigidbody = GetComponent<Rigidbody>();

    }

    private Vector3 ProjectionOnXZ(Vector3 vector)
    {
        vector.y = 0f;
        return vector.normalized;
    }
```

```
    void FixedUpdate()
    {
        float horizontal = Input.GetAxisRaw("Horizontal");
        float vertical = Input.GetAxisRaw("Vertical");

        Vector3 move_vector = new Vector3(vertical, 0.0f, -horizontal).normalized

        if (move_vector.magnitude > 0.1f)
        {
            _player_rigidbody.MoveRotation(Quaternion.Slerp(transform.rotation, Q
        }
        _player_rigidbody.AddForce(((transform.right * -vertical) + (transform.fo

        if (Input.GetKey(KeyCode.Space) && _player_rigidbody.position.y < max_jur
        {
            _player_rigidbody.AddForce(Vector3.up * jump_size, ForceMode.Velocity
        }
    }
}
```

SProCamera.cs Движение камеры за игроком

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

public class SProCamera : MonoBehaviour
{
    public Transform target;
    public float speedX = 300.0f;
    public float speedY = 400.0f;
    public float limitY = 90.0f;
    public float length_camera_react = 100.0f;
    public float length_player_react = 10.0f;
    public float min_obst_dist = 1.8f;
    public bool true_move;

    public LayerMask active;
    public LayerMask obstacles;
    public LayerMask no_player;

    private float _min_distance = 2.0f;
    private float _max_distance;
    private Vector3 _local_posotion;
    private float _current_y_rotation;
    public LayerMask _origin_mask;

    private GameObject _prev_select_obj;

    private Vector3 _position
    {
        get { return transform.position; }
```

```
    set { transform.position = value; }
}

void Start()
{
    true_move = true;
    _prev_select_obj = null;
    _local_posotion = target.InverseTransformPoint(_position);
    _max_distance = Vector3.Distance(_position, target.position);
    _origin_mask = GetComponent<Camera>().cullingMask;
}

private void Update()
{
    CameraReact();
}

private void LateUpdate()
{
    _position = target.TransformPoint(_local_posotion);
    CameraRot();
    if (true_move)
    {
        CameraObst();
        CameraMinObstDist();
    }
    _local_posotion = target.InverseTransformPoint(_position);
}

void CameraRot()
{
    float move_x = Input.GetAxis("Mouse X");
    float move_y = -Input.GetAxis("Mouse Y");

    if (move_y != 0)
    {
        float tmp = Mathf.Clamp(_current_y_rotation + move_y * speedY * Time.
        if (tmp != _current_y_rotation)
        {
            float r = tmp - _current_y_rotation;
            transform.RotateAround(target.position, transform.right, r);
            _current_y_rotation = tmp;
        }
    }

    if (move_x != 0)
    {
        transform.RotateAround(target.position, transform.up, move_x * speedX
    }
```

```
    transform.LookAt(target);
}

void CameraObst()
{
    float distance = Vector3.Distance(_position, target.position);
    RaycastHit hit;

    if (Physics.Raycast(target.position, transform.position - target.position
    {
        _position = hit.point;
    } else if (distance < _max_distance && !Physics.Raycast(_position, -trans
    {
        _position -= transform.forward * 0.05f;
    }
}

void CameraMinObstDist()
{
    if (Vector3.Distance(_position, target.position) < min_obst_dist)
    {
        GetComponent<Camera>().cullingMask = no_player;
    } else
    {
        GetComponent<Camera>().cullingMask = _origin_mask;
    }
}

void CameraReact()
{
    Ray ray_forward = new Ray(transform.position, transform.forward.normalize
    RaycastHit hit;

    if (Physics.Raycast(ray_forward, out hit, length_camera_react, active) &&
    {

        GameObject obj = hit.collider.gameObject;

        if (_prev_select_obj && _prev_select_obj != obj)
        {
            if (_prev_select_obj.GetComponent<SSelect>() && _prev_select_obj.
            {
                _prev_select_obj.GetComponent<SSelect>().Deselect();
                _prev_select_obj = null;
            }
        }

        if (obj.GetComponent<SSelect>())
        {
            _prev_select_obj = obj;
```

```
                obj.GetComponent<SSelect>().Select();
            }
        }
        else if (_prev_select_obj && _prev_select_obj.GetComponent<SSelect>() &&
        {
            _prev_select_obj.GetComponent<SSelect>().Deselect();
            _prev_select_obj = null;
        }

    }

    Vector3 XZ(Vector3 vector)
    {
        vector.y = 0;
        return vector;
    }
}
```

SGun.cs Стрельба из оружия

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

public class SGun : MonoBehaviour
{
    public GameObject bullet;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.F))
        {
            GameObject new_bull = Instantiate(bullet, transform.position + new Ve
            new_bull.GetComponent<Rigidbody>().velocity = -new_bull.transform.rig
        }

    }
}
```

SBullet.cs Взаимодействие пули с врагами

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

public class SBullet : MonoBehaviour
{
```

```
    private void OnCollisionEnter(Collision target)
    {
        if (target.gameObject.tag == "GunTarget")
        {
            Destroy(target.gameObject);
        }
    }
}
```

SCapRotation.cs Вращение шляпки

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

public class SCapRotation : MonoBehaviour
{
    public float cap_speed = 2.0f;

    private Transform _player_transform;
    // Start is called before the first frame update
    void Start()
    {
        _player_transform = transform.parent;
    }

    // Update is called once per frame
    void Update()
    {
        transform.RotateAround(_player_transform.position, transform.up, cap_spee
    }
}
```

STargetForCamera.cs Перемещение цели для камеры за игроком

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

public class STargetForCamera : MonoBehaviour
{
    public Transform player;

    private Vector3 _local_posotion;
    // Start is called before the first frame update
    void Start()
    {
        _local_posotion = player.InverseTransformPoint(transform.position);
    }

    // Update is called once per frame
    void Update()
```

```
    {
        transform.position = player.TransformPoint(_local_posotion);
        _local_posotion = player.InverseTransformPoint(transform.position);
    }
}
```

SStartGame.cs Старт игры и переключение сцены

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

public class SStartGame : MonoBehaviour
{
    public void PlayPressed()
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
        SceneManager.LoadScene("Game");
    }

    public void ExitPressed()
    {
        Application.Quit();
    }
}
```

SSelect.cs Триггер объекта на приближение игрока

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

public class SSelect : MonoBehaviour
{
    public bool true_select;

    public void Start()
    {
        true_select = false;
    }

    public void Select()
    {
        true_select = true;
    }

    public void Deselect()
    {
        true_select = false;
    }
}
```

SOpenDoor.cs Открывание двери

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

public class SOpenDoor : MonoBehaviour
{
    public float speed_open = 1.0f;
    public bool is_open;
    public Transform panel;
    public Transform door;

    //private BoxCollider _box_door;
    private Quaternion _close_rotation;
    private Quaternion _open_rotation;

    private SSelect _select;

    private Color _panal_color;

    //Start is called before the first frame update
    void Start()
    {
        _select = GetComponent<SSelect>();
        _panal_color = panel.GetComponent<Renderer>().material.color;

        _close_rotation = door.rotation;
        _open_rotation = Quaternion.LookRotation(-door.transform.right);
        is_open = false;

    }

    // Update is called once per frame
    void Update()
    {
        if (_select.true_select)
        {
            if (Input.GetKeyDown(KeyCode.E))
            {
                is_open = !is_open;
            }
            else
            {
                if (is_open) panel.GetComponent<Renderer>().material.color = Colo
                if (!is_open) panel.GetComponent<Renderer>().material.color = Col
            }

        }
        else
        {
```

```
            panel.GetComponent<Renderer>().material.color = _panal_color;
        }

    }

    private void LateUpdate()
    {
        Rotate();
    }

    void Rotate()
    {
        if (door.rotation != _open_rotation && is_open)
            door.rotation = Quaternion.RotateTowards(door.rotation, _open_rotatio

        if (door.rotation != _close_rotation && !is_open)
            door.rotation = Quaternion.RotateTowards(door.rotation, _close_rotati
    }


}
```

SEnterCodeCanvas.cs Ввод кода в панель

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

public class SEnterCodeCanvas : MonoBehaviour
{
    [SerializeField]
    private Text _panel;
    public bool _entering;

    public void Start()
    {
        _entering = true;
    }
    public void Enter1(string text)
    {
        if (_entering)
        {
            _panel.text += text;
        }
    }
}
```

SEnterCode.cs Механика ввода кода

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

```
public class SEnterCode : MonoBehaviour
{
    [SerializeField]
    private GameObject _note;
    private Camera _camera;
    private SSelect _select;
    private bool _is_active;
    private float[] _speedXandY;

    private Material _note_material;

    void Start()
    {
        _select = GetComponent<SSelect>();
        _note.SetActive(false);
        _is_active = false;
        _note_material = GetComponent<Renderer>().material;
        _camera = Camera.main;
        _speedXandY = new float[2] { _camera.GetComponent<SProCamera>().speedX, _
    }

    void Update()
    {
        if (_select.true_select)
        {
            _note_material.EnableKeyword("_EMISSION");

            if (Input.GetKeyDown(KeyCode.E) && !_is_active)
            {
                Activate();
            }

            if (Input.GetKeyDown(KeyCode.Q) && _is_active)
            {
                Deactivate();
                Cursor.lockState = CursorLockMode.Locked;
                Cursor.visible = false;
            }

        }
        else
        {
            if (_is_active)
            {
                Cursor.lockState = CursorLockMode.Locked;
                Cursor.visible = false;
            }
            _note_material.DisableKeyword("_EMISSION");
            Deactivate();
```

```
        }
    }

    void Activate()
    {
        Cursor.lockState = CursorLockMode.Confined;
        Cursor.visible = true;
        _camera.GetComponent<SProCamera>().speedX = 0f;
        _camera.GetComponent<SProCamera>().speedY = 0f;

        _note.SetActive(true);
        _is_active = true;
    }

    void Deactivate()
    {
        _camera.GetComponent<SProCamera>().speedX = _speedXandY[0];
        _camera.GetComponent<SProCamera>().speedY = _speedXandY[1];

        _note.SetActive(false);
        _is_active = false;
    }
}
```

SCodeConfirm.cs Подтверждение ввода кода

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

public class SCodeConfirm : MonoBehaviour
{
    [SerializeField]
    private string _confirmed_text;
    [SerializeField]
    private GameObject _object_to_action;
    [SerializeField]
    private SEnterCodeCanvas _sEnterCodeCanvas_entering;
    private SOpenDoor _action;
    private Text _text;

    void Start()
    {
        _text = GetComponent<Text>();
        _action = _object_to_action.GetComponent<SOpenDoor>();
    }

    void Update()
    {
        if (_text.text == _confirmed_text)
        {
```

```
            _action.speed_open = 1.5f;
            _sEnterCodeCanvas_entering._entering = false;
            _text.text = _confirmed_text;
            _text.color = Color.green;
        }
        else if (_text.text.Length >= 4)
        {
            _text.text = "";
        }
    }
}
```

SReadNote.cs Чтение записки

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

public class SReadNote : MonoBehaviour
{
    [SerializeField]
    private GameObject _note;
    private SSelect _select;
    private bool _is_active;

    private Material _note_material;

    void Start()
    {
        _select = GetComponent<SSelect>();
        _note.SetActive(false);
        _is_active = false;
        _note_material = GetComponent<Renderer>().material;
    }

    void Update()
    {
        if (_select.true_select)
        {
            _note_material.EnableKeyword("_EMISSION");

            if (Input.GetKeyDown(KeyCode.E) && !_is_active)
            {
                Activate();
            }

            if (Input.GetKeyDown(KeyCode.Q) && _is_active)
            {
                Deactivate();
            }

        } else
```

```
        {
            _note_material.DisableKeyword("_EMISSION");
            Deactivate();
        }
    }

    void Activate()
    {
        _note.SetActive(true);
        _is_active = true;
    }

    void Deactivate()
    {
        _note.SetActive(false);
        _is_active = false;
    }
}
```

SNumber.cs Взаимодествие цифр из игры "пятнашки"друг с другом

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

public class SNumber : MonoBehaviour
{
    public Transform numbers;
    public LayerMask active;
    public Vector3 true_position;
    public bool act;
    public bool ok;

    private SSelect _select;
    private BoxCollider _box;
    private Color _origin_color;
    private float _min_distance;
    private Vector3 _move_x;
    private Vector3 _move_y;

    private Vector3 _1_position, _2_position, _3_position;
    private Vector3 _4_position, _5_position, _6_position;
    private Vector3 _7_position, _8_position;

    // Start is called before the first frame update
    void Start()
    {
        act = false; ok = false;
        _select = GetComponent<SSelect>();
        _box = GetComponent<BoxCollider>();

        true_position = new Vector3(transform.position.x, transform.position.y, t
```

```
    _1_position = numbers.Find("1").transform.position; _2_position = numbers
    _3_position = numbers.Find("3").transform.position; _4_position = numbers
    _5_position = numbers.Find("5").transform.position; _6_position = numbers
    _7_position = numbers.Find("7").transform.position; _8_position = numbers

    _origin_color = Color.HSVToRGB(22.0f, 77.0f, 32.0f);
    GetComponent<Renderer>().material.color = Color.white;
    _move_x = _1_position - _2_position;
    _move_y = _4_position - _7_position;
    _min_distance = transform.right.normalized.magnitude;
}

// Update is called once per frame
void Update()
{
    if (!ok && _select.true_select && act)
    {
        GetComponent<Renderer>().material.color = Color.green;
        Moving();

    }

    if (!ok && !_select.true_select && act)
    {
        GetComponent<Renderer>().material.color = _origin_color;
    }

    if (!act)
    {
        GetComponent<Renderer>().material.color = Color.white;

    }
    if (ok)
    {
        GetComponent<Renderer>().material.color = Color.magenta;
    }
}

void LateUpdate()
{
    if (!act)
        StartPosition();

}

void Moving()
{
    Ray ray_right = new Ray(transform.position, transform.right.normalized);
    Ray ray_left = new Ray(transform.position, -transform.right.normalized);
```

```
    Ray ray_up = new Ray(transform.position, transform.up.normalized);
    Ray ray_down = new Ray(transform.position, -transform.up.normalized);

    Debug.DrawRay(ray_right.origin, ray_right.direction, Color.red);
    Debug.DrawRay(ray_left.origin, ray_left.direction, Color.blue);
    Debug.DrawRay(ray_up.origin, ray_up.direction, Color.green);
    Debug.DrawRay(ray_down.origin, ray_down.direction, Color.yellow);

    if (Input.GetMouseButtonDown(0))
    {
        bool true_hit_right = Physics.Raycast(ray_right, _min_distance, activ
        bool true_hit_left = Physics.Raycast(ray_left, _min_distance, active)
        bool true_hit_up = Physics.Raycast(ray_up, _min_distance, active);
        bool true_hit_down = Physics.Raycast(ray_down, _min_distance, active)

        if (!true_hit_right) transform.position += _move_x;
        else if (!true_hit_left) transform.position -= _move_x;
        else if (!true_hit_up) transform.position += _move_y;
        else if (!true_hit_down) transform.position -= _move_y;
    }
}
void StartPosition()
{
    numbers.Find("1").transform.position = _1_position;
    numbers.Find("2").transform.position = _2_position + _right;
    numbers.Find("3").transform.position = _3_position + _down;
    numbers.Find("4").transform.position = _4_position;
    numbers.Find("5").transform.position = _5_position + _up;
    numbers.Find("6").transform.position = _6_position + _down;
    numbers.Find("7").transform.position = _7_position;
    numbers.Find("8").transform.position = _8_position + _up;
}
private void Check()
{
    if (transform.position == true_position)
    {
        GetComponent<Renderer>().material.color = Color.magenta;
    }
}

private Vector3 _right
{
    get { return -_move_x; }
}

private Vector3 _left
{
    get { return _move_x; }
}
```

```
    private Vector3 _down
    {
        get { return -_move_y; }
    }

    private Vector3 _up
    {
        get { return _move_y; }
    }
}
```

SPlayNumb.cs Механика игры "пятнашки"

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

public class SPlayNumb : MonoBehaviour
{
    public Transform panel;
    public Transform numbers;
    public GameObject lamp;
    public bool active;
    public bool finish;

    private SSelect _select;
    private Color _origin_color;

    private SNumber _1, _2, _3;
    private SNumber _4, _5, _6;
    private SNumber _7, _8;

    // Start is called before the first frame update
    void Start()
    {
        _select = GetComponent<SSelect>();
        _1 = Pos("1"); _2 = Pos("2"); _3 = Pos("3");
        _4 = Pos("4"); _5 = Pos("5"); _6 = Pos("6");
        _7 = Pos("7"); _8 = Pos("8");

        _origin_color = panel.GetComponent<Renderer>().material.color;
        active = false;
        finish = false;
        if (lamp.GetComponent<Light>().enabled)
            lamp.GetComponent<Light>().enabled = false;
    }

    // Update is called once per frame
    void Update()
    {
        if (_select.true_select)
        {
```

```
            if (!active) panel.GetComponent<Renderer>().material.color = Color.gr
            if (Input.GetKeyDown(KeyCode.E))
                active = !active;
        } else
        {
            panel.GetComponent<Renderer>().material.color = _origin_color;
        }

        if (active)
            panel.GetComponent<Renderer>().material.color = Color.magenta;
        else if (!_select)
            panel.GetComponent<Renderer>().material.color = _origin_color;

    }

    private void LateUpdate()
    {
        if (active)
        {
            Activation();
            Check();
        }
        else
        {
            Deactivation();
        }
    }

    private void Activation()
    {
        _1.act = true; _2.act = true; _3.act = true;
        _4.act = true; _5.act = true; _6.act = true;
        _7.act = true; _8.act = true;
    }

    private void Deactivation()
    {
        _1.act = false; _2.act = false; _3.act = false;
        _4.act = false; _5.act = false; _6.act = false;
        _7.act = false; _8.act = false;
    }

    private void Check()
    {
        if (InTruePosition(_1) && InTruePosition(_2) &&
            InTruePosition(_3) && InTruePosition(_4) &&
            InTruePosition(_5) && InTruePosition(_6) &&
            InTruePosition(_7) && InTruePosition(_8))
        {
            _1.ok = true; _2.ok = true; _3.ok = true;
```

```
            _4.ok = true; _5.ok = true; _6.ok = true;
            _7.ok = true; _8.ok = true;

            if (!finish)
                lamp.GetComponent<Light>().enabled = true;
            finish = true;
        }

    }

    private SNumber Pos(string num)
    {
        return numbers.Find(num).GetComponent<SNumber>();
    }

    private bool InTruePosition(SNumber num)
    {
        return num.transform.position == num.true_position;
    }



}
```

SPlayLmp.cs Игра со светом

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

public class SPlayLmp : MonoBehaviour
{
    public Transform lmp;
    public Transform light1;
    public GameObject light2;
    public GameObject light3;
    public GameObject first_lamp;
    public GameObject second_lamp;

    private SSelect _select;
    private Color _origin_color;
    private bool _end;
    // Start is called before the first frame update
    void Start()
    {
        _end = false;
        _select = GetComponent<SSelect>();
        _origin_color = GetComponent<Renderer>().material.color;

        if (first_lamp.GetComponent<Light>().enabled)
            first_lamp.GetComponent<Light>().enabled = false;
```

```
    if (second_lamp.GetComponent<Light>().enabled)
        second_lamp.GetComponent<Light>().enabled = false;

    if (light3.GetComponent<Light>().enabled)
        light3.GetComponent<Light>().enabled = false;
}

// Update is called once per frame
void Update()
{
    if (light2.GetComponent<Light>().enabled)
    {
        if (_select.true_select)
        {
            GetComponent<Renderer>().material.color = Color.grey;
            if (Input.GetKeyDown(KeyCode.E))
            {
                GetComponent<Renderer>().material.color = Color.blue;
                if (first_lamp.GetComponent<Light>().enabled)
                    first_lamp.GetComponent<Light>().enabled = false;
                else first_lamp.GetComponent<Light>().enabled = true;

                if (second_lamp.GetComponent<Light>().enabled)
                    second_lamp.GetComponent<Light>().enabled = false;
                else second_lamp.GetComponent<Light>().enabled = true;
            }

        }
        else
        {
            GetComponent<Renderer>().material.color = _origin_color;
        }

    }
}
void LateUpdate()
{
    Check();
    if (light2.GetComponent<Light>().enabled) Check();
}

void Check()
{
    bool ok = true;
    foreach (Transform l in lmp.GetComponentInChildren<Transform>())
    {
        if (!l.GetComponent<Light>().enabled)
            ok = false;
    }
    if (ok && !_end)
```

```
        {
            if (light2.GetComponent<Light>().enabled)
                light2.GetComponent<Light>().enabled = false;

            if (!light3.GetComponent<Light>().enabled)
                light3.GetComponent<Light>().enabled = true;

            light1.RotateAround(light1.position, light1.right, 175.0f);
            light1.GetComponent<Light>().intensity = light1.GetComponent<Light>()
            _end = true;
        }


    }

}
```

SPlace.cs Панель для игры в "пятнашки"

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

public class SPlace : MonoBehaviour
{
    public Camera cam;
    public Transform player;
    public GameObject panel_numb;

    private float _min_distanse = 0.5f;
    private float _origin_dist;
    private bool _act;

    private void Start()
    {
        _origin_dist = cam.GetComponent<SProCamera>().length_player_react;
        _act = true;
    }

    // Update is called once per frame
    void Update()
    {
        if (Vector3.Distance(player.position, transform.position) < _min_distanse
        {
            cam.GetComponent<SProCamera>().true_move = false;
            cam.GetComponent<SProCamera>().length_player_react = 50;
            cam.GetComponent<Camera>().cullingMask = cam.GetComponent<SProCamera>
        } else if (_act)
        {
            cam.GetComponent<Camera>().cullingMask = cam.GetComponent<SProCamera>
            cam.GetComponent<SProCamera>().length_player_react = _origin_dist;
            cam.GetComponent<SProCamera>().true_move = true;
```

```
        }

        if (panel_numb.GetComponent<SPlayNumb>().finish && _act)
        {
            cam.GetComponent<Camera>().cullingMask = cam.GetComponent<SProCamera>
            cam.GetComponent<SProCamera>().length_player_react = _origin_dist;
            cam.GetComponent<SProCamera>().true_move = true;
            _act = false;
        }
    }
}
```

SPanelKey.cs Поучение игрового трофея - оружия

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

public class SPanelKey : MonoBehaviour
{
    public GameObject light3;
    public GameObject key;
    public Transform key_cube;
    public GameObject gun;
    public GameObject cap;

    private GameObject new_key;
    private bool _get_key;
    private SSelect _select;
    private Color _origin_color;
    private bool _ok;
    private Color _origin_color_key;
    private float _min_dist = 2.0f;

    // Start is called before the first frame update
    void Start() {
        _get_key = false;
        _origin_color = GetComponent<Renderer>().material.color;
        _select = GetComponent<SSelect>();
        _ok = true;
    }

    // Update is called once per frame
    void Update()
    {
        if (light3.GetComponent<Light>().enabled)
        {
            if (_select.true_select)
            {
            GetComponent<Renderer>().material.color = Color.green;

            if (Input.GetKeyDown(KeyCode.E)) GetKey();
```

```
            }
            else
            {
                GetComponent<Renderer>().material.color = _origin_color;
            }
        }

        if (_get_key && _ok)
        {
            if (Vector3.Distance(new_key.transform.position, cap.transform.positi
            {
                new_key.GetComponent<Renderer>().material.color = Color.green;
                if (Input.GetKeyDown(KeyCode.E))
                {
                    PutOn();
                    _ok = false;
                }
            } else
            {
                new_key.GetComponent<Renderer>().material.color = _origin_color_k
            }
        }

    }

    void GetKey()
    {
        if (!_get_key)
        {
            float pos_z = key_cube.GetComponents<Collider>().Length;
            new_key = Instantiate(key, key_cube.position + new Vector3(0.0f, 0.0f
            new_key.GetComponent<Rigidbody>().velocity = key_cube.right * 4.0f;
            _origin_color_key = new_key.GetComponent<Renderer>().material.color;
            _get_key = true;
        }
    }

    void PutOn()
    {
        if (new_key.activeSelf) new_key.SetActive(false);
        if (cap.activeSelf) cap.SetActive(false);
        if (!gun.activeSelf) gun.SetActive(true);

    }

}
```

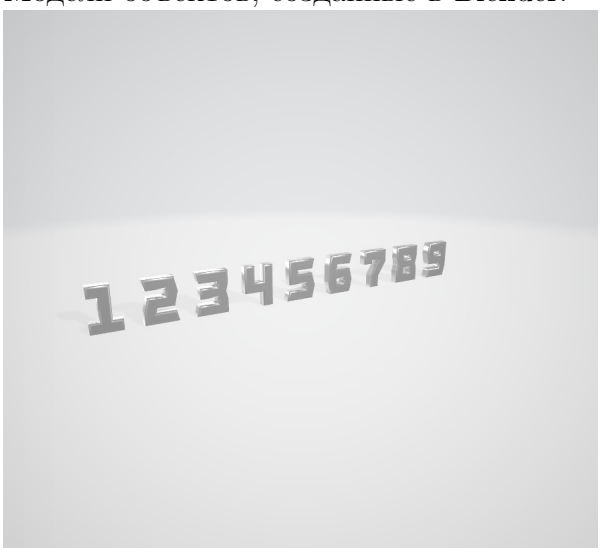
SEnemy.cs Движение врагов

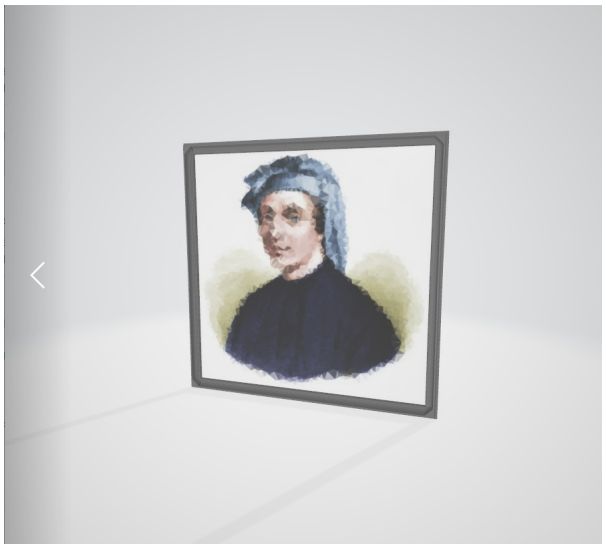
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

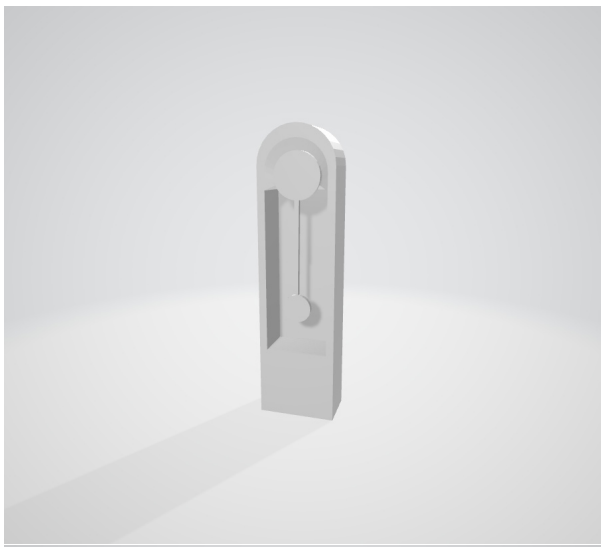
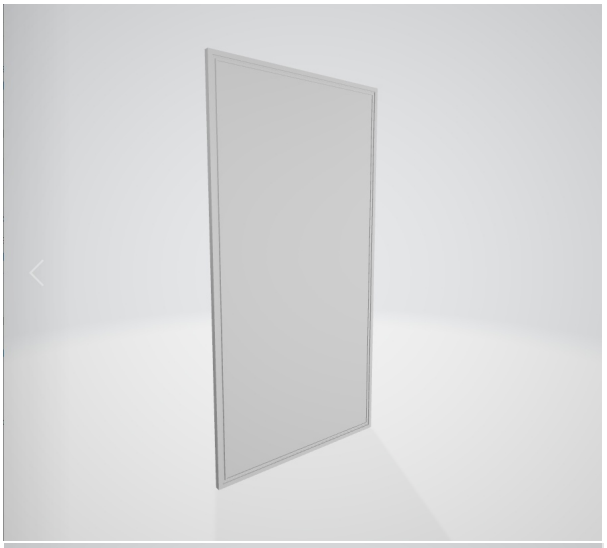
```
public class SEnemy : MonoBehaviour
{
    public Transform start_point;
    public float enemy_speed = 0.3f;
    private Vector3 move_vect;
    // Start is called before the first frame update
    void Start()
    {
        move_vect = transform.forward;
    }

    // Update is called once per frame
    void Update()
    {
        transform.position += move_vect * enemy_speed;
    }

    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.tag == "Obstacles")
        {
            if (move_vect == transform.forward)
                move_vect = -transform.forward;
            else
                move_vect = transform.forward;
        }

        if (collision.gameObject.tag == "Player")
        {
            collision.gameObject.transform.position = start_point.position;
        }
    }
}
```

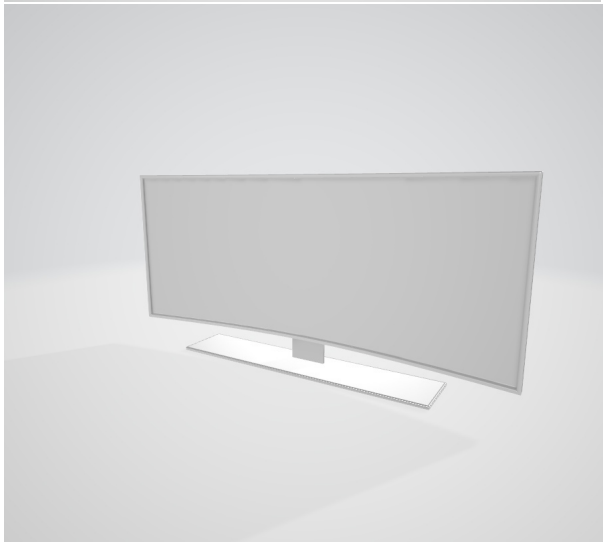
3D-модели:

Модели объектов, созданные в Blender.

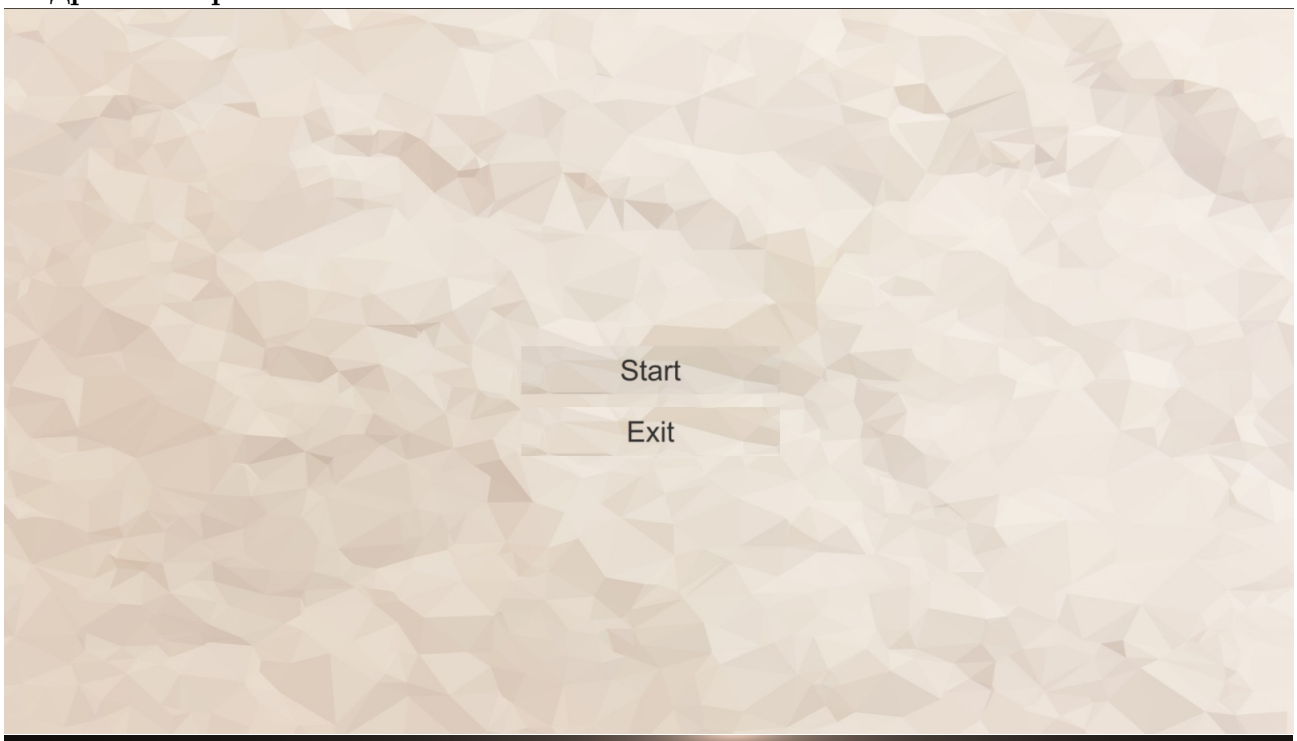




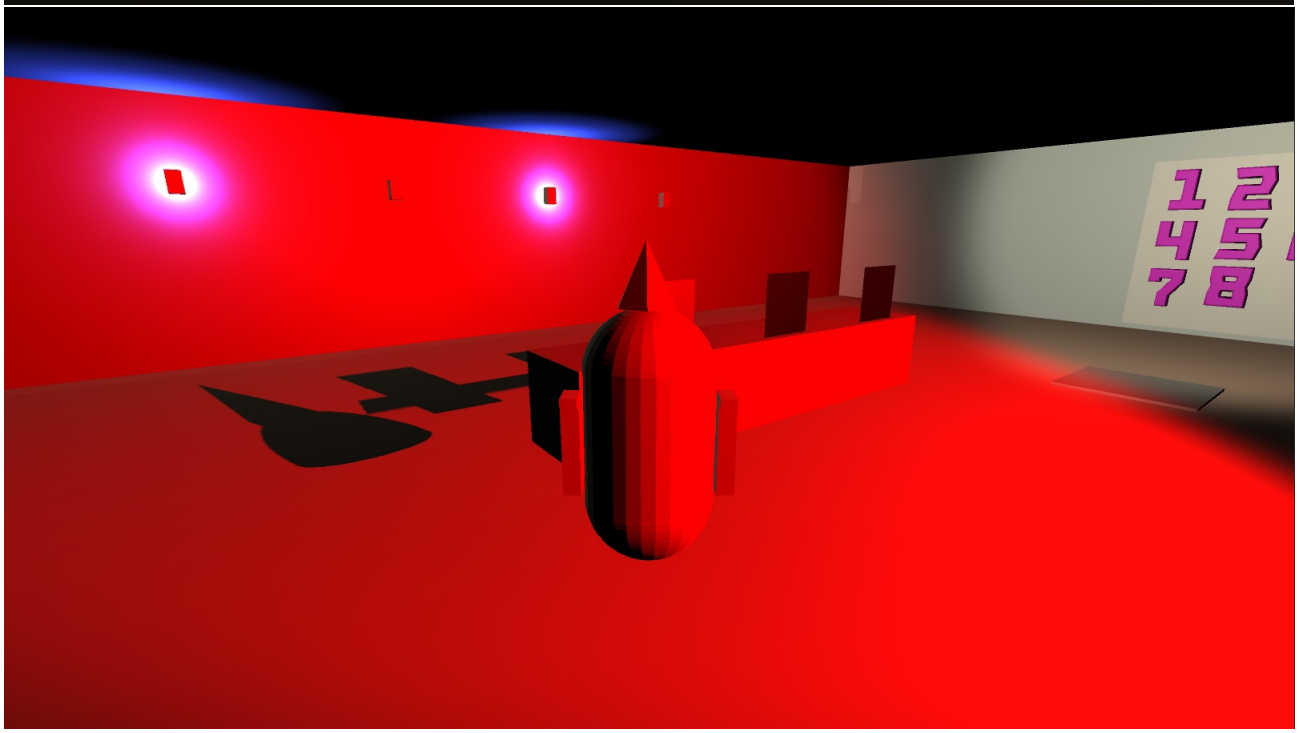

Кадры из игры:

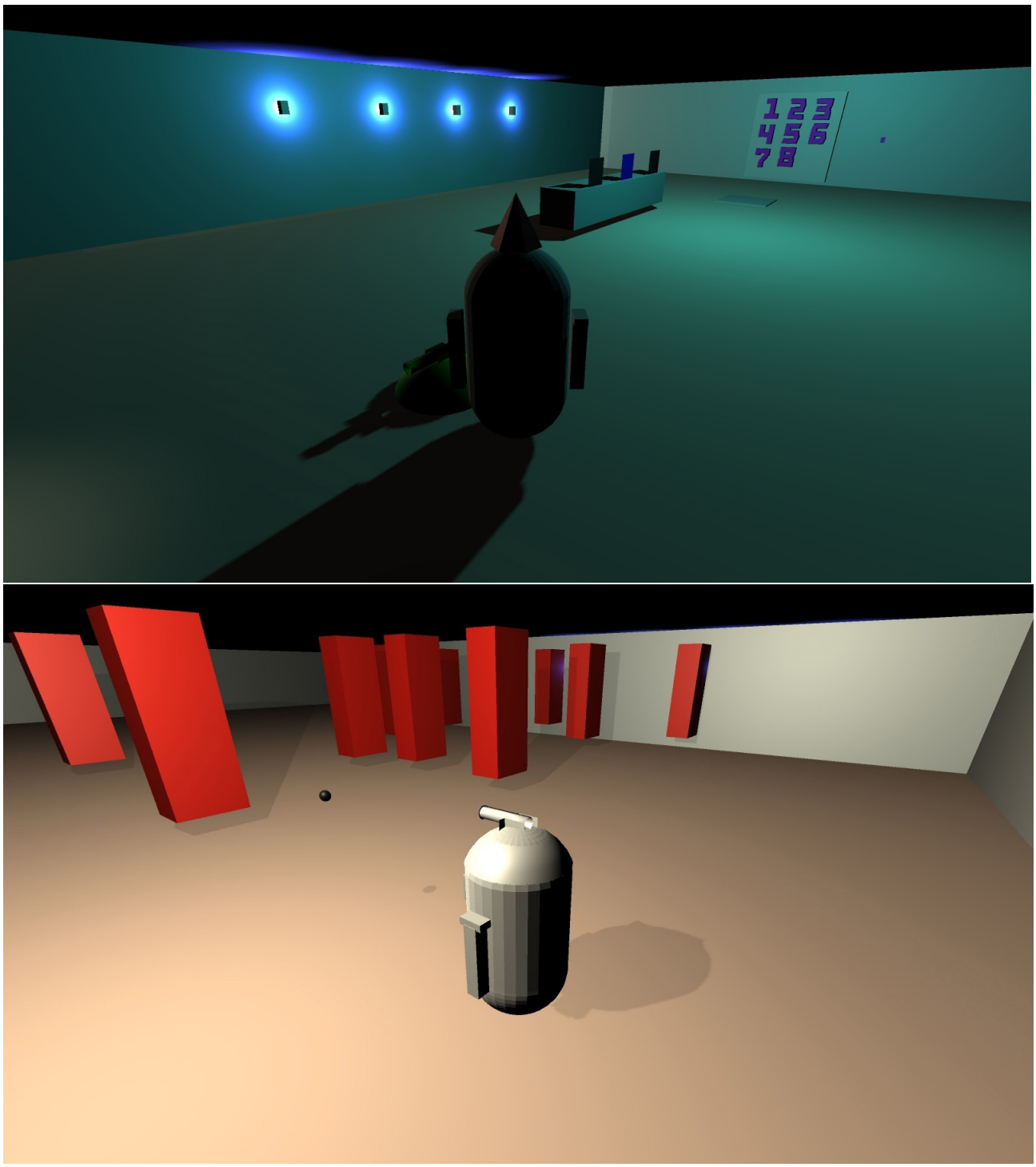



0112
1
2
3
4
5
6
7
8
9
0
Q - quit


Q - quit


1 5 2
4 3
7 8 6

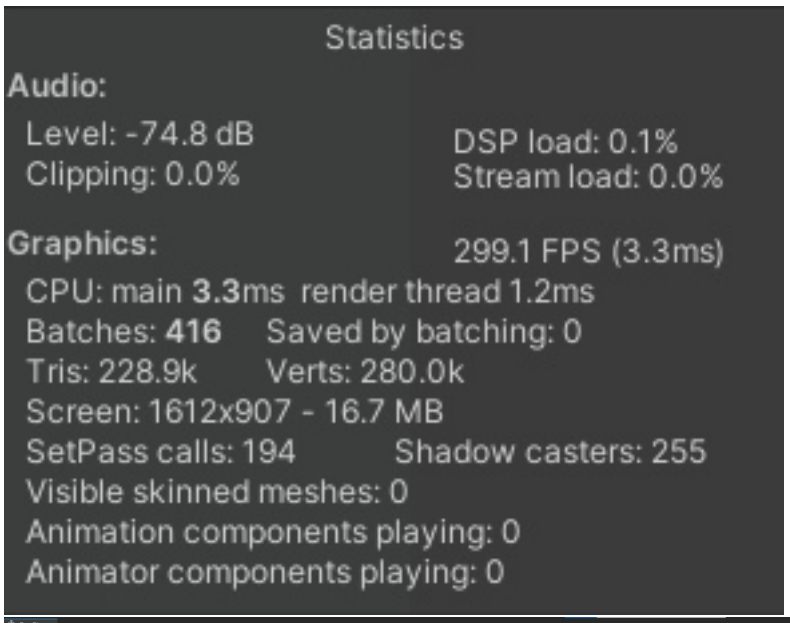

1 2
4 5
7 8

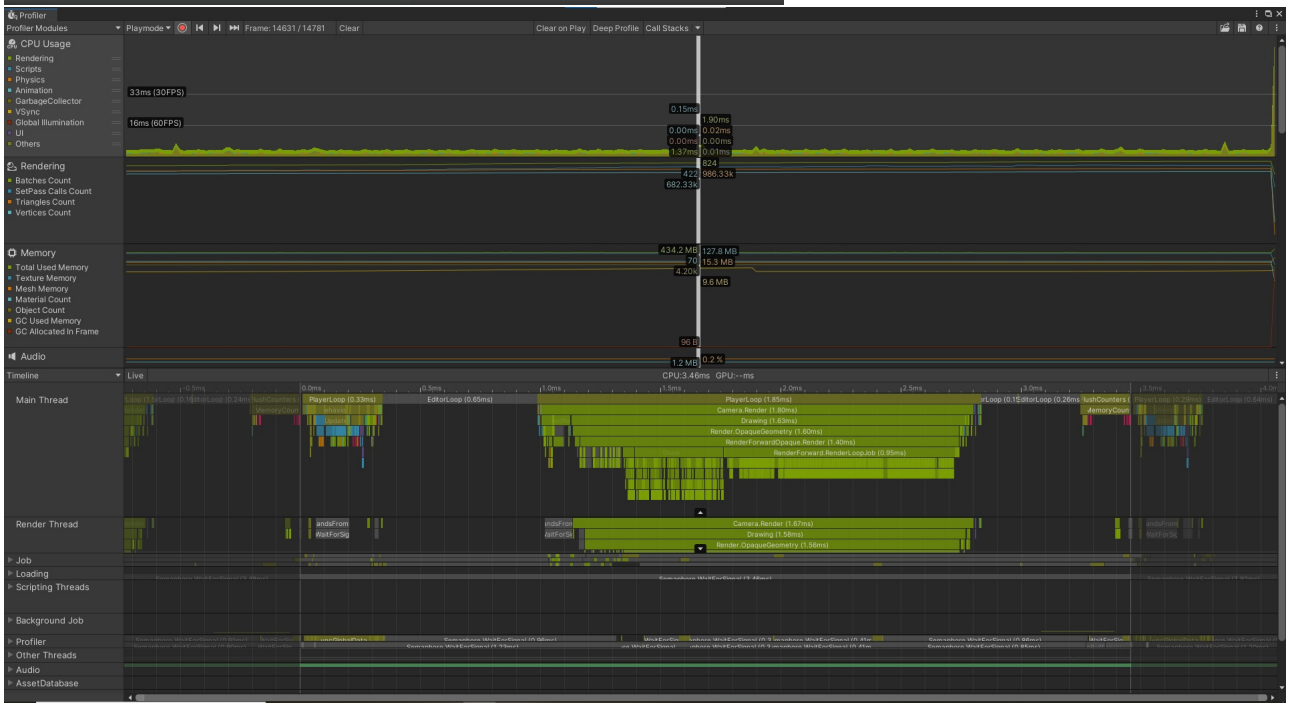


Тестирование

Профайлер:

На скриншотах показано, как большенство времени тратится на именно отрисовку объектов и просчет их физики, в то время как время работы скриптов составляет очень маленькую часть от всей времени отрисовки кадра. Средний фпс - 300 кадров в секунду.





Ссылка на GitHub

https://github.com/Igor743646/SummerPractise