МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Высшая школа информационных технологии и интеллектуальных систем

**ОТЧЕТ О ПРОХОЖДЕНИИ ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ**

\_\_\_\_\_Кафедра программной инженерии\_\_\_\_\_\_\_

название лаборатории

Студент группы \_11-708\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Иванаевский В.В\_\_\_\_

подпись (Ф.И.О.)

Руководитель практики от ИТИС:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_доцент к.н.\_\_\_\_Голицина И.Н.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность, звание, Ф.И.О.)

**Оглавление**

Введение…………………………………………………………………………………………..…3

Основная часть………………………………………………………………………..…………...4-9

Заключение…………………………………………………………………………..………………9

**Введение**

После распределения задач между группами появилась задача сделать игру в жанре хоррор, в которой обязательно должны присутствовать элементы таких тематик, как Fear of clowns и 8 eyes.

Для работы были выбраны данные программы:

1. В качестве игрового движка был выбран Unity.
2. Для работы с 3d была выбрана программа Autodesk Maya 2018.
3. Для рисования спрайтов был использован Adobe Photoshop CS6.

Для работы со звукозаписью и аудио – FL-Studio 12 и Adobe Audition CC 2017.

Появились задачи следующего плана: работать с механикой игры, писать скрипты, продумывать и создавать локации, рисовать спрайты, обрабатывать звуки

**Основная часть**

В начале практики было решено, что из-за представленных тем жанр игры будет хоррор-стелс. Учитывая это, было решено создать темный лабиринт с запеченным светом на точках без мирового освещения.



Рис. 1 Лабиринт

В данном лабиринте были расставлены клоуны и глаза. Весь лабиринт разделен на 4 логические части. В каждой из 4 частей есть разные функции:

1. Много клоунов

2. Тяжело найти предметы

3. Слабое освещение

4. В некоторые зоны можно пройти только предварительно набрав определенное количество предметов.

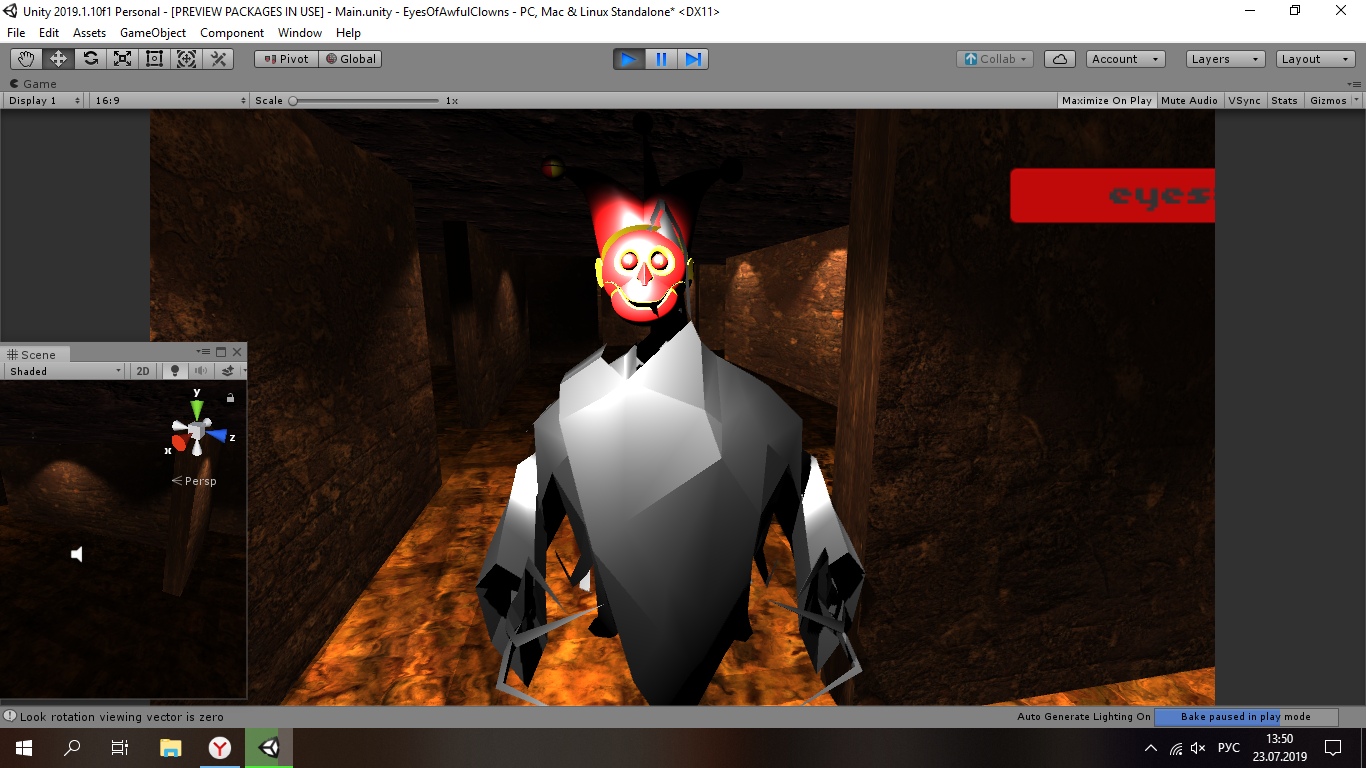


Рис.2 Клоун

**Клоун**

Клоуныдвижутся по маршруту с помощью Nav mesha и скрипта.

Каждый фрейм метод вычисляет расстояние до игрока. После этого происходит проверка. Если игрок находится в нужном радиусе, то булево значение будет true, иначе false. После этого происходит проверка количества точек передвижения и булева значения таргет. В зависимости от этих значений клоун движется по маршруту или за игроком. Если игрок выходит из радиуса преследования, то isTarget = false и клоун возвращается на маршрут.

void Update()

{

target = player.position;

float dis = Vector3.Distance(transform.position, player.position);

if (dis < activeDistance)

{

isTarget = true;

}

if (dis > disableDistance)

isTarget = false;

if (wayPoints.Length >= 2 && !isTarget)

{

agent.stoppingDistance = 0;

agent.SetDestination(wayPoints[wayCount].position);

if (!agent.hasPath)

{

SetRotation(defLook.position);

curTimeout += Time.deltaTime;

if (curTimeout > timeWait)

{

curTimeout = 0;

if (wayCount < wayPoints.Length - 1) wayCount++;

else wayCount = 0;

}

}

}

else if (wayPoints.Length == 0 || isTarget)

{

agent.stoppingDistance = stoppingDistance;

agent.SetDestination(target);

if (agent.velocity == Vector3.zero) SetRotation(target); } }

Модель клоуна состоит из ассетов и собственных наработок.

Голова клоуна (рис 2) всегда смотрит в направлении игрока с помощью скрипта:

if (player == null)

{

var \_player = GameObject.FindGameObjectWithTag("Player");

look = new Vector3(\_player.transform.position.x, transform.position.y, \_player.transform.position.z);

}

else

look = new Vector3(player.transform.position.x, transform.position.y, player.transform.position.z);

transform.LookAt(look);

Данный скрипт передает положение игрока по координатам х и z. После этого вызывается метод LookAt, который непосредственно поворачивает голову клоуна в направлении игрока.

**Персонаж**

Префаб персонажа состоит из множества скриптов с разной функциональностью:

1. Camera Controller

2. Player Movement

3. Footsteps

4. Pause menu

5. Taken eyes

6. Camera bobbing

**Camera controller** отвечает за поворот камеры:

if (axes == RotationAxis.MouseX)

transform.Rotate(0, Input.GetAxis("Mouse X") \* sensHorizontal, 0);

else if (axes == RotationAxis.MouseY)

{

\_rotaionX -= Input.GetAxis("Mouse Y") \* sensVertical;

\_rotaionX = Mathf.Clamp(\_rotaionX, minimumVert, maximumVert);

float rotationY = transform.localEulerAngles.y;

transform.localEulerAngles = new Vector3(\_rotaionX, rotationY, 0);

}

В зависимости от направления выполняет математические вычисления.

**Player movement**  отвечает за движение персонажа данным скриптом:

void Update()

{

float deltaX = Input.GetAxis("Horizontal") \* speed;

float deltaZ = Input.GetAxis("Vertical") \* speed;

Vector3 movement = new Vector3(deltaX, 0, deltaZ);

movement = Vector3.ClampMagnitude(movement, speed);

movement.y = gravity;

movement \*= Time.deltaTime;

movement = transform.TransformDirection(movement);

\_charController.Move(movement);

}

Передает информацию для определенных клавиш, добавляет некоторые вычисления и движет персонажем.

**Footsteps** отвечает за звук шагов во время движения. Т.е., когда местоположение объекта меняется, проигрывается звук шагов. Если передвижение прекращается, то прекращается и звук.

**Pause menu** за остановку игры, вывода меню и вызова функций для определенных кнопок. Каждый фрейм проверяется условие на нажатие клавиши esc. Если клавиша нажата, то приостанавливаются множество действий игры, такие как: передвижение персонажа, врагов, выключается скрипт вращения камерой. Вызывается курсор и UI меню с кнопками, на которых есть разная функциональность перехода на другие сцены, выход из игры или продолжения игры

**Taken eyes** отвечает за подбирание глаз и увеличения счетчика:

if (Input.GetMouseButtonDown(0))

{

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

RaycastHit hit;

if (Physics.Raycast(ray, out hit))

{

if (hit.transform.CompareTag("Eye"))

{

eyeCount++;

Destroy(hit.transform.gameObject);

text.text = "eyes: " + eyeCount;

}

}

if (eyeCount == finished)

{

Pause();

EndGame.GetComponent<Text>().text = "U WIN!!!";

EndGame.SetActive(true);

}

}

Таким образом, при нажатии клавиши мыши проверяется куда попадает луч. Если тэг объекта Eye, то увеличивается количество подобранных предметов, удаляется объект и обновляет текст с информацией. Дальше если удовлетворяет условию количества глаз, то останавливает игру и обновляет текст.

**Camera bobbing** отвечает за имитацию тряски камеры во время движения. В то время, как персонаж движется, камера движется по синусу и создает иллюзию тряски.

**Отображение UI**

В игре присутствует несколько разных UI. Первое отображение происходит при запуске игры на стартовом экране. На кнопках используются функции, вызывающие начальный уровень или же выход из игры. При наведении на разные кнопки выводятся разные картинки.

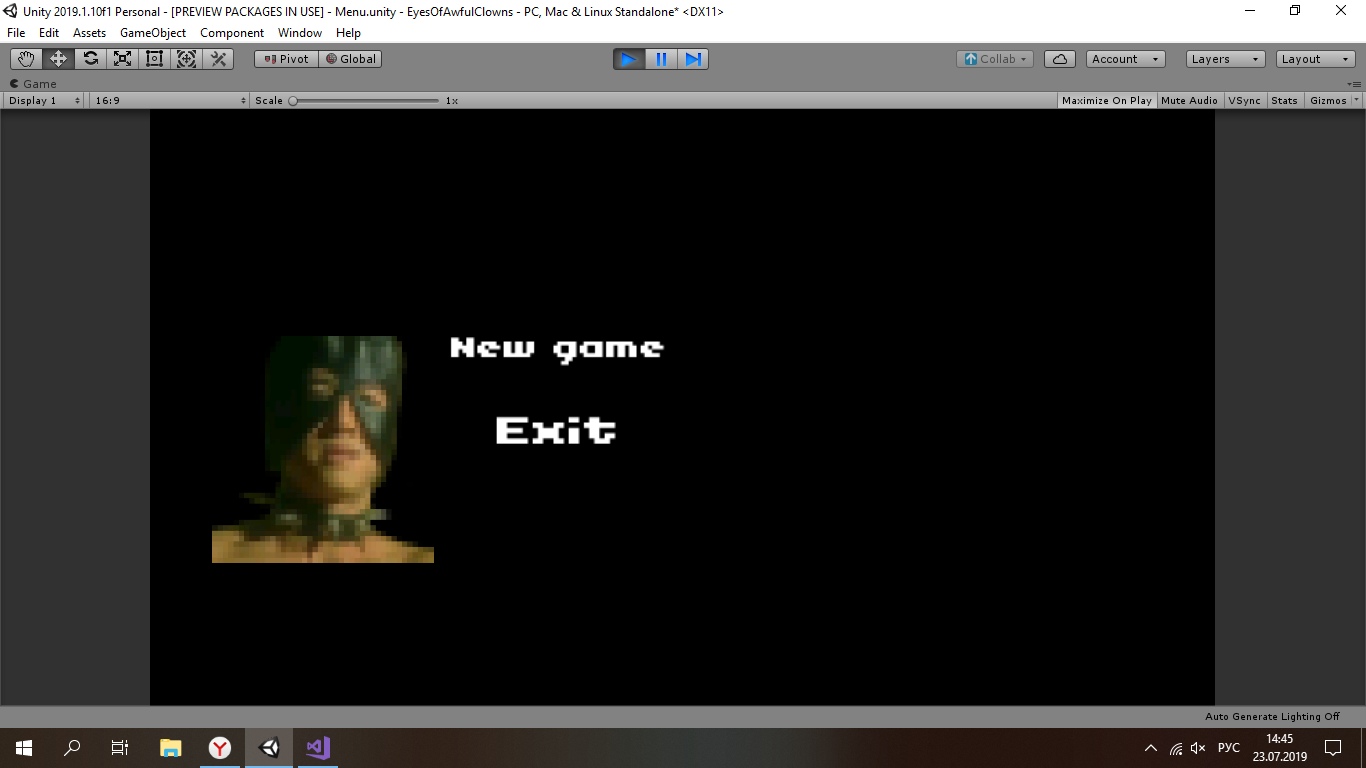


Рис. 3 Стартовое меню

второе UI: отображение количества подобранных глаз во время игры. После каждого подобранного предмета с нужным тегом обновляется текст с помощью специального метода. Обновление текста происходит данным образом:

text.text = “eyes: “ + countEyes; ,где countEyes переменная хранящая количество глаз.



Рис. 4. Счетчик

Следующее UI отображает победу при 8 подобранных предметах или проигрыш при тригере с клоуном.

После каждого поднятия предмета происходит проверка достаточно ли предметов для выигрыша или нет. Если достаточно, то выводится соответствующий текст и кнопка возвращения к главному меню, которая вызывает функцию переключения на другую сцену.

Если игрок столкнется с клоуном, то сработает триггер, который выведет этот же объект (рис. 5), но с измененным текстом, сообщающим о проигрыше.



Рис. 5 Уведомление о победе

Последнее UI отображает условия открытия дверей.



Рис.6 Условие открытия двери

Как работает открывание дверей на локации:

if (Input.GetMouseButtonDown(0))

{

if (doorUI.activeSelf)

doorUI.SetActive(false);

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

RaycastHit hit;

if (Physics.Raycast(ray, out hit))

{

if (hit.transform.CompareTag("Button"))

{

int eyesCount = player.GetComponent<TakenEye>().eyeCount;

if (eyesCount >= eyeC)

{

Destroy(door);

Destroy(gameObject);

}

else doorUI.SetActive(true);

}

}

}

При нажатии клавиши мыши проверяется куда попадает луч. Если тэг объекта Button, то проверяется количество подобранных предметов. Если удовлетворяет условию, то уничтожает объект, иначе сообщает сколько нужно для открытия.

**Заключение**

Благодаря кратким срокам для практики появились условия для полного погружения в работу и концентрации только на ней на всем протяжении практики: что днём, что ночью. Также был получен опыт использования облака в юнити. Правильный импорт и экспорт в этом облаке. Благодаря данной практике появилось понимание тонкостей движка Unity. На данный момент был получен огромный опыт написания скриптов, расстановки объектов на сцене, понятия функционала компонентов, практики в фотошопе, импорта объектов в юнити.