**Введение**

Моими целями в проекте Самосбор было написание кода на C# для создания разных механик и получение новых знаний в среде разработки Unity. Из тех задач что были выполнены мной это передвижение персонажа (ходьба, прыжок, присед, блокирование камеры на персонаже по углам наклона, перемещение камеры, хп), добавление оружие персонажу, скрипт по нанесению урона, триггеры (воспроизведение звука, создание объекта и перемещение объекта, появление задачи.), скрипт для патрона, перемещение по сценам, перемещение по сценам.

**Первый код под названием** [**Bullet**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/Bullet.cs)**.**

Нужен этот код для существования патрона. Мы публично задаём float переменную “BulletLife” сколько времени будет существовать патрон.

**Второй код под названием** [**CAAAAM**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/CAAAAM.cs)**.**

Данный код нужен для управления и поворота камерой. В этом классе объявляются несколько приватных полей (private), помеченных атрибутом [SerializeField]. Эти поля будут отображаться в инспекторе Unity, позволяя редактировать значения переменных без необходимости изменения кода. Полями являются:

\_sensivity: скорость, с которой камера реагирует на движения мыши.

\_player: ссылка на Transform игрока.

\_verticalLover: максимальный угол вращения камеры вниз.

\_verticalUpper: максимальный угол вращения камеры вверх.

\_currentVerticalAngle: текущий угол вращения камеры по вертикали

В методе Start() устанавливается состояние блокировки курсора (Cursor.lockState) и скрытие курсора (Cursor.visible).

В методе Update() происходит обновление камеры в каждом кадре:

Получается значение оси Y мыши (vertical) и значения оси X мыши (horizontal) с помощью методов Input.GetAxis().

Значение vertical умножается на чувствительность (\_sensivity) и на время кадра (Time.deltaTime), чтобы получить плавное изменение угла вращения камеры по вертикали.

Значение horizontal умножается на чувствительность (\_sensivity) и на время кадра (Time.deltaTime), чтобы получить плавное изменение угла вращения камеры по горизонтали.

Текущий угол вращения камеры по вертикали (\_currentVerticalAngle) ограничивается значениями максимального угла вращения вверх (\_verticalUpper) и вниз (\_verticalLover) с помощью метода Mathf.Clamp().

Вращение камеры по вертикали задается с помощью метода Transform.localRotation, преобразуя угол вращения в Quaternion.Euler().

Вращение камеры по горизонтали осуществляется путем поворота Transform игрока с помощью метода Transform.Rotate().

Этот код позволяет игроку вращать камеру вокруг себя по горизонтали

**Третьей код под названием** [**Control**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/DamageScripts.cs)**.**

Данный код - это скрипт на языке C# для игры, использующей движок Unity. Он содержит класс Control, который управляет поведением игрока в игре.

Переменные:

player - объект игрока, на котором висит данный скрипт.

weapon1 и weapon2 - два различных типа оружия, которые могут быть выбраны игроком.

speed - базовая скорость передвижения игрока.

\_speed - скорость передвижения игрока, используемая для восстановления скорости после использования ускорения.

DashSpeed - скорость, с которой игрок будет перемещаться при использовании приема "рывок".

isGrounded - флаг, указывающий, находится ли игрок на земле или нет.

IsDrawWeapon - флаг, указывающий, отображается ли у игрока в данный момент оружие.

Weapon - флаг, указывающий, какой тип оружия выбран у игрока.

Методы:

Start() - метод, вызываемый при запуске скрипта. Задает начальные значения переменных и определяет игрока.

Update() - метод, который вызывается каждый кадр. В данном случае используется для вызова метода Move(), который отвечает за передвижение игрока.

OnCollisionStay() - метод, вызываемый каждый раз, когда игрок сталкивается с объектом и остается на нем. Устанавливает флаг isGrounded в true.

Move() - метод, который отвечает за передвижение игрока в зависимости от нажатых клавиш. Если игрок нажимает клавишу W, то он начинает движение вперед. Если нажать клавишу S, то он начнет движение назад. Аналогично, клавиши A и D отвечают за движение влево и вправо соответственно. Если игрок нажимает клавишу Shift, то его скорость увеличится на два раза. Если нажать клавишу Space, то игрок сможет использовать прием "рывок", который позволит ему быстро переместиться на небольшое расстояние. Клавиша Tab отвечает за выбор оружия. Если оружие не отображается у игрока, то он может нажать Tab для его появления. Если оружие уже отображается, то нажатие клавиши Tab скроет его. Клавиши Alpha1 и Alpha2 позволяют выбрать первый и второй тип оружия

Если клавиша Tab была нажата и IsDrawWeapon равно false, то проверяем значение Weapon. Если оно равно false, то активируем weapon1 и деактивируем weapon2, и устанавливаем IsDrawWeapon равным true. Если значение Weapon равно true, то активируем weapon2 и деактивируем weapon1, и устанавливаем IsDrawWeapon равным true.

Если IsDrawWeapon равно true, то проверяем клавишу Tab. Если она была нажата, то деактивируем weapon1 и weapon2, и устанавливаем IsDrawWeapon равным false.

Если значение Weapon равно true, то устанавливаем speed равным \_speed, и проверяем клавишу Alpha1. Если она была нажата, то активируем weapon1 и деактивируем weapon2, и устанавливаем Weapon равным false.

Если значение Weapon равно false, то также устанавливаем speed равным \_speed, и проверяем клавишу Alpha2. Если она была нажата, то активируем weapon2 и деактивируем weapon1, и устанавливаем Weapon равным true.

**Четвёртый код под названием** [**DamageScripts**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/DamageScripts.cs)**.**

Этот код содержит класс DamageScripts, который используется для нанесения урона игроку при столкновении.

В классе определена публичная переменная DAMAge, которая устанавливает количество наносимого урона.

Метод OnCollisionEnter() вызывается, когда объект, на котором расположен этот скрипт, сталкивается с другим объектом. В данном случае, при столкновении вызывается метод GetDamage() класса Control, который принимает значение переменной DAMAge и наносит урон игроку.

Однако, для того чтобы данный код работал корректно, в классе Control должен быть реализован статический метод GetDamage(int damage) с соответствующей логикой.

**Пятый код под названием** [**Gun**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/Gun.cs)**.**

Этот код представляет класс Gun, который содержит несколько полей, отвечающих за параметры оружия, такие как количество пуль, максимальное количество пуль, модель пули, камера, из которой происходит выстрел, точка спавна пуль, скорость пуль, разброс, аудиоэффект выстрела и система частиц для эффектов выстрела.

Метод Update() вызывается каждый кадр и запускает метод Shoot(), который отвечает за процесс выстрела. Метод Shoot() проверяет, нажата ли кнопка мыши, и если quantityBullet не равно нулю, проигрывает звук выстрела и запускает эффект выстрела, уменьшает количество пуль на единицу и создает луч, направленный из центра камеры. Затем он проверяет, попала ли пуля в какой-либо объект, и если да, то берет координаты точки попадания в качестве targetPoint, в противном случае берет координаты точки на расстоянии 75 единиц от центра камеры.

Затем метод вычисляет вектор направления dirWithoutSpread от точки спавна пуль до targetPoint. Далее метод генерирует два случайных числа с плавающей точкой в диапазоне от -spread до spread, чтобы добавить случайный разброс к вектору направления, получив вектор направления с разбросом dirWithSpread. Затем метод создает игровой объект пули, устанавливает его направление на направление точки спавна пуль и назначает скорость пули в направлении dirWithSpread.

Если нажата клавиша R, количество пуль quantityBullet обновляется до MaxQuantityBullet.

В начале объявляются публичные переменные, такие как количество пуль (quantityBullet), максимальное количество пуль (MaxQuantityBullet), префаб пули (bullet), главная камера (mainCamera), место спауна пуль (spawnerBullet), скорость полета пули (BulletSpeed), звук выстрела (ShootMp3), компонент AudioSource для воспроизведения звука выстрела (\_AudioSource) и система частиц для эффекта выстрела (ShootEff).

В методе Update() вызывается метод Shoot(), который обрабатывает выстрелы из пушки.

Метод Shoot() проверяет, была ли нажата левая кнопка мыши и есть ли еще пули в запасе. Если условие выполняется, воспроизводится звук выстрела, запускается система частиц для создания эффекта выстрела, количество пуль уменьшается на 1.

Затем определяется направление выстрела с помощью луча, и, если луч сталкивается с каким-то объектом, координаты точки столкновения становятся целью выстрела. Если луч не сталкивается с объектом, то целью выстрела становится точка, находящаяся на расстоянии 75 единиц в направлении луча.

Далее определяется направление выстрела без учета разброса. Разброс вычисляется случайным образом в пределах заданного интервала (от -spread до spread) по осям x и y. Полученные значения добавляются к направлению выстрела без учета разброса, получая тем самым направление выстрела с учетом разброса. Создается новый объект пули, которая устанавливается на место спауна, направление пули устанавливается в направлении спауна, пуля получает начальную скорость в направлении выстрела.

Если игрок нажимает клавишу R, то количество пуль устанавливается на максимальное значение.

**Шестой код под названием** [**NewScene**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/NewScene.cs)**.**

В классе определен публичный метод "scene", который принимает на вход номер сцены (sceneNumber) в виде целочисленного значения и использует класс SceneManager из библиотеки UnityEngine для загрузки сцены с указанным номером.

Для загрузки новой сцены нужно создать объект класса NewScene и вызвать его метод "scene" с номером нужной сцены в качестве аргумента.

Пример использования данного класса: при нажатии на кнопку в интерфейсе игры можно вызвать метод "scene" с номером сцены, которая должна быть загружена при нажатии на данную кнопку**.**

**Седьмой код под названием** [**Zadacha**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/Zadacha.cs)**.**

Данный код относится к скрипту взаимодействия игрока с объектом, имеющим триггерный коллайдер. Когда игрок проходит через триггер, метод OnTriggerEnter(Collider Player) вызывается. В данном методе у объекта, содержащего скрипт, изменяются значения полей MissionText и NameMissionText, а затем объект уничтожается с помощью метода Destroy().

Конкретно, при прохождении игроком через триггерный коллайдер, текстовое поле Missiontmp внутри объекта, содержащего скрипт, устанавливается в значение MissionText, а поле NameMissiontmp устанавливается в значение NameMissionText. Затем объект, содержащий скрипт, уничтожается с помощью метода Destroy(), чтобы объект не срабатывал каждый раз при входе игрока в триггерный коллайдер.

**Восьмой код под названием** [**onColliderEnter**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/onColliderEnter.cs)**.**

Этот код представляет скрипт, который отслеживает, когда игрок входит в зону триггера, и выполняет несколько действий при входе в триггер.

Переменная Door отвечает за игровой объект двери, на которую будет влиять этот скрипт. Переменная Player отвечает за игрока, который входит в триггер. Переменная dim отвечает за объект, который должен быть уничтожен при входе в триггер. Переменная ZVYK отвечает за звук, который должен быть проигран при входе в триггер. Переменная ObjecticAudio отвечает за игровой объект, на котором находится компонент AudioSource для проигрывания звука.

Когда игрок входит в триггер, выполняются следующие действия:

Дверь открывается. В этом случае, на объекте двери вызывается метод Interact(), реализация которого находится в скрипте Door.

Проигрывается звук. Метод PlayOneShot() вызывается на компоненте AudioSource, который находится на игровом объекте ObjecticAudio.

Объект dim уничтожается.

Игровой объект, на котором находится скрипт, также уничтожается.

**Девятый код под названием** [**Perem**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/perem.cs)**.**

Данный код создает объект, который перемещается в заданном направлении со скоростью speed. Он также имеет переменную tine, которая определяет время жизни объекта. Когда это время заканчивается, объект удаляется.

В методе Start() определяется запуск корутины oble().

Метод FixedUpdate() вызывается каждый раз при физическом обновлении и перемещает объект в направлении, определяемом вектором dir.

Метод oble() ожидает в течение заданного времени (tine) и затем уничтожает объект с помощью метода Destroy(gameObject).

**Десятый код под названием** [**SetActive**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/setActive.cs)**.**

Данный код отвечает за активацию/деактивацию объекта при входе коллайдера в зону триггера.

При этом существует переменная obj - ссылка на объект, который будет активирован/деактивирован. Если переменная obj имеет значение false, то при входе коллайдера в зону триггера объект будет активирован и уничтожен скриптом Destroy(gameObject). Если же obj имеет значение true, то при входе коллайдера в зону триггера объект будет деактивирован и также уничтожен.

Здесь используется метод SetActive() для активации/деактивации объекта, который принимает булево значение в качестве параметра: true - активация, false - деактивация.

**Одиннадцатый код под названием** [**spawnObjj**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/spawnObjj.cs)**.**

Данный скрипт отслеживает столкновение объекта с коллайдером, к которому данный скрипт прикреплен. Если происходит столкновение с игроком (т.е. с объектом, который имеет компонент Collider), то создается новый объект ObjSpawn в точке SpawnPoint. Для создания нового объекта используется метод Instantiate, который принимает три аргумента: сам объект, который нужно создать (в данном случае ObjSpawn), его позицию и его поворот. Здесь мы используем позицию SpawnPoint и поворот по умолчанию (identity).

**Двенадцатый код под названием** [**stats**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/stats.cs)**.**

Данный код представляет собой скрипт для управления здоровьем персонажа в Unity. В нем определяется переменная максимального здоровья (maxHealth), статическая переменная здоровья (Health) и номер сцены (sceneL), куда перейти при смерти персонажа.

В методе Start() задается начальное значение здоровья, равное максимальному.

В методе Update() происходит вызов метода die(), который проверяет, упало ли здоровье до нуля и, если это так, загружает указанную сцену.

В скрипте также определен статический метод GetDamage(), который вызывается при получении персонажем урона. Этот метод вычитает из здоровья персонажа полученный урон (DAMAge)

**Тринадцатый код под названием** [**trg**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/trg.cs)**.**

Данный код написан на языке программирования C# и использует библиотеку Unity. Класс trg содержит метод OnTriggerEnter, который вызывается, когда объект с коллайдером входит в зону действия триггера. При этом происходят следующие действия:

Активируется объект hud (интерфейс игрока).

Запускается корутина Hides() - задержка на 3 секунды, после которой:

Объект hud деактивируется.

Текущий объект уничтожается.

**Четырнадцатый код под названием** [**trgWall**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/trgWall.cs)**.**

Данный код отслеживает событие, когда игрок (предположительно объект с компонентом Collider) сталкивается с определенной стеной в игре.

Когда это происходит, вызывается метод OnTriggerEnter(), который принимает объект Collider, который столкнулся со стеной. В данном случае этот параметр имеет имя "player", и его тип Collider.

Затем код устанавливает свойство "isClimbing" в компоненте waa в значение false. Для этого используется объект booleanController, который вероятно имеет ссылку на компонент waa.

Это может означать, что игрок не может карабкаться по данной стене (если это стена, по которой игрок может карабкаться).

**Пятнадцатый код под названием** [**trigSound**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/trigSound.cs)**.**

Данный скрипт реагирует на событие, когда коллайдер объекта (переменная Player) входит в триггер, к которому прикреплен данный скрипт. При срабатывании триггера проигрывается звуковой эффект, указанный в переменной SCaudio, с помощью компонента AudioSource, который прикреплен к объекту ObjectAudio. Затем сам объект, к которому прикреплен скрипт, уничтожается с помощью функции Destroy.

**Шестнадцатый код под названием** [**waa**](https://github.com/Zolotushka1/MorLovecraft/blob/Fokin-V/Assets/Scripts/Fokin/waa.cs)**.**

Этот код создает компонент, позволяющий игроку лазить по стенам. Скрипт реализует движение по горизонтали и вертикали и содержит следующие переменные:

climbSpeed - скорость движения во время лазания;

raycastDistance - дальность луча для определения объектов, по которым можно лазить;

targetLayer - слой объектов, по которым можно лазить;

displayText - компонент для отображения текста на экране;

customText - текст, отображаемый на экране при приближении к объекту;

componentToDisable - компонент, который нужно отключить во время лазания;

rb - компонент Rigidbody объекта;

mainCamera - камера, которую использует игрок;

isClimbing - флаг, указывающий, находится ли игрок в режиме лазания;

moveSpeed - скорость движения в обычном режиме.

Скрипт использует функции Start и Update для управления объектом и RaycastToScreenCenter для обнаружения объектов, по которым можно лазить. Также в скрипте присутствует функция для обработки столкновения с объектами.