Районный конкурс учебно-исследовательских работ

учащихся «Путь к успеху»

Исследовательский проект по информатике

**«Разработка компьютерной игры»**

Выполнила: Манюгин Захар,

учащийся 10 класса МБОУ

«СОШ с. Павло-Федоровка Кировского района»

Руководитель: Федосенко А.А.,

учитель информатики и математики.

С. Павло-Федоровка

2024

**Содержание.**

1. Введение 3
2. История появления компьютерных игр 4
3. Что такое «игровой движок» 6
4. Выбор «игрового движка» 7
5. Этапы создания игры 8
6. Простая сцена и физика персонажа 8
7. Графика и анимация персонажа 11
8. Первый враг 13
9. Система боя 16
10. Конец игры 18
11. Заключение 23
12. Список литературы 24

**Введение.**

В современном мире профессия программиста, геймдизайнера, аниматора очень популярны. Это связано с высокой заработной платой и общим престижем данной профессии. Научиться создавать игры довольно непросто, однако, каждый месяц в один только App Store загружают от 346 (февраль 2021 года) до 46 815 (сентябрь 2016 года) игр. Это означает то, что множество маленьких «инди» студий, состоящих из одного или нескольких людей, каждый день создают множество игр. Научиться создавать игры, изучить процесс и создать собственную видеоигру решил и я.

**Актуальность:**На данный момент, рынок компьютерных видеоигр испытывает сильный подъем, так общий объем игровой индустрии по итогам девяти месяцев 2020 года составил 174,9 миллиарда долларов (на 19,6% выше, чем в 2019-м). Почти половина (49%) пришлась на игры для смартфонов и планшетов — их объем вырос на 25,6% и составил 86,3 миллиарда долларов. ПК-игры заняли 21,4% рынка с 37,4 миллиарда долларов (+6,2%), а консольные — 29% рынка и 51,2 миллиард. Общий объем игровой индустрии сильно превысил объем киноиндустрии (100 миллиардов долларов) и объем индустрии спорта (75 миллиардов долларов).

Эти данные говорят о том, что компьютерные игры крайне востребованы в среде самых различных социальных групп, а также то, что игры стали обыденной частью жизни многих людей.

**Гипотеза:** Я предполагаю, что любой школьник может создать свою игру, при этом он может не знать азов программирования.

**Цель** исследовательской работы: показать процесс создания собственной игры, которая заинтересует сверстников.

**История появления компьютерных игр.**

Компьютерные игры стали настоящим культурным феноменом - возникнув как незамысловатый плод творческой мысли программистов, они с каждым годом приобретали всё большую популярность - и развились до того, что стали отдельной специфической спортивной дисциплиной - киберспортом. По всему миру выросли компании по разработке игр, а работа в этой сфере стала розовой мечтой для многих юных умов, желающих создавать любимые компьютерные игрушки. Некоторые игровые серии стали культовыми - например, DOOM, Fallout, Dragon Quest, Final Fantasy, Contra, WoW, Starcraft, Need for Speed, Grand theft Auto. Как минимум про одну из них наверняка слышал любой человек, который хоть раз сталкивался с компьютером.

Попытки создать простенькие игры на цифровых устройствах предпринимались ещё до начала Второй Мировой войны, а в 1947 уже была запрограммирована первая электронная игра, монитором для которой служил экран военного радара - это был симулятор вражеских ракет. Однако считается, что первой компьютерной игрой стала игра "Крестики нолики", которую в одиночку создал Сендри Дуглас в 1952 году, с минимально возможным полем 3х3 клетки. Сегодня это может показаться смешным, но в тот момент это было революционным новшеством. Крестики нолики - одна из самых популярных игр для маленьких детей.

В 1958 году в Нью-Йорке Уильям Хигинботэм порадовал пользователей новым опытным образцом. Им стала видеоигра «Теннис на двоих». Посетители его лаборатории могли поиграть в теннис на цифровом корте, управляя своими "ракетками" с помощью джойстиков. Несмотря на простоту функционала, эта игра стала очередным прорывом в нарождающемся мире геймеров.

Однако настоящий виртуальный взрыв произошел в 1962 году. Компания DEC разработала игровой контроллер и вместе с компьютером     PDP-1 стала распространять как тестовую программу, не имеющую прежде подобий, игру SpaceWar. Это была первая компьютерная игра, ставшая по-настоящему популярной.

Компьютеры в то время были еще громоздкими. Прошло почти 10 лет до того момента, когда возникли компактные платы на транзисторных схемах. В мае 1972 года была представлена Magnavox Odissey — первая игровая приставка для телевизора.

С этого момента игровой компьютерный мир стал продвигаться семимильными шагами. Развитие шло в четырех основных направлениях: непосредственно компьютеры, телевизионные игровые приставки, электронные игровые автоматы и карманные электронные игры.

В 1979 году американской компанией Milton Bradley была выпущена первая карманная игровая консоль, в которую было вложено сразу 12 игр. В 1980 году японская Nintendo, модернизировав игры на калькуляторе, осуществила массовый выпуск простейших монохромных консолей с серией игр Game&Watch. В Советском Союзе эти консоли стали прототипом продукции фирмы «Электроника» — игр «Тайны океана» и «Ну, погоди!», которыми были увлечены буквально все.

По мере развития технологий, виртуальный мир наполнялся звуком, совершенствовалась графика, добавлялось видео. Сегодня каждый желающий может скачать и установить игры на любой вкус: стрелялки и драки; гонки и спортивные игры, симуляторы; аркады, стратегии и приключения; логические, обучающие и развивающие игры.

Подобрать игру можно как для взрослого, так и для подростка, или даже для ребенка, который едва научился сидеть.

Процесс развития компьютерных игр уже нельзя остановить. Они становятся все более разнообразными и захватывающими. На смену персональным играм пришли игры браузерные, где в онлайн-режиме можно общаться, сообща решать пусть виртуальные, но важные задачи. Такие игры, помимо развлечения, дают возможность осваивать и развивать навыки общения, социализации, расширять кругозор. Мир компьютерных игр по-прежнему дает больше положительных эффектов, нежели отрицательных. Важно лишь правильно выбирать для себя подходящие игры и верно дозировать количество времени, проведенного у монитора.

**Что такое «игровой движок».**

Термин «игровой движок» является прямой копией английского «Game Engine». Фактически это объединенный в единое целое комплекс прикладных программ, с помощью которых обеспечивается графическая визуализация, звуковое сопровождение, перемещение внутриигровых персонажей, их действия в соответствии со скриптами, а также игра в сети, встроенные графические сцены, соблюдение физических эффектов и законов и многое другое. Разработчик получает готовый качественный инструмент с большим количеством библиотек. В результате ему не надо писать большую часть базового программного кода и можно сосредоточиться на реализации своих идей, графики, игровой механики и сюжета, не тратя время на написание кода с нуля. В результате ряд компаний занялся разработкой именно игровых движков, а разработчики игр стали покупать на них лицензии, как это получилось с Unreal Engine или id Tech 3. Стоимость лицензии может составлять от нескольких тысяч до миллионов долларов. Но при этом надо отметить, что для некоммерческого использования многие игровые движки, например, популярные Unity и Unreal Engine 4 доступны бесплатно. Остановимся на этих движках немного подробнее. Движки Unity и Unreal Engine 4 являются самыми популярными в среде разработчиков из-за их удобства, детальной проработки и большого количества дополнительных библиотек, что позволяет настраивать и реализовывать практически любые идеи, приходящие в голову дизайнерам и игроделам.

**Выбор «игрового движка».**

Unreal Engine 4 - движок смело можно назвать легендой. Его разработка началась в 1998 году и с тех пор он постоянно модернизируется, дополняется и совершенствуется. Современный Unreal Engine 4 — это движок, на котором пишут игры для любых платформ и операционных систем, начиная от ОС Windows и заканчивая всеми современными консолями — Playstation 5, Xbox One, а также мобильными платформами, в том числе и iOS.

Unity — одна из популярных платформ для разработчиков игровых приложений. Можно услышать, что этот движок называют самым молодым. Но тут надо отметить, что он появился в 2005 году и с тех пор успешно развивается.

Как и Unreal Engine, Unity — кроссплатформенная система. На ней пишут игры для любых платформ и в любых жанрах. На Unity делают как простые аркады и головоломки, так и шутеры от первого лица с достаточно сложным игровым миром. Пример — Dead Trigger. Другой пример - интересная стратегия Endless Legend. Движок бесплатен для некоммерческого использования, а при написании коммерческих игр требуется выплачивать комиссионный сбор авторам движка. Большой плюс Unity — простота его освоения. Минус — графика в играх, созданных на основе этого движка. Она выглядит проще и не настолько реалистична, как у Unreal Engine. Тем не менее, около половины всех мобильных игр, по заверениям разработчиков, написаны именно на этом движке. Для создания игры я буду использовать именно Unity, так как он больше подходит новичку.

**Этап создание игры.**

**Простая сцена и физика персонажа.**

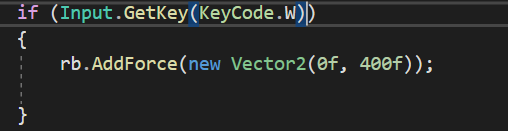
Для начала определимся с жанром нашей игры. Я решил, что нужно сделать что-то простое, поэтому мой выбор пал на 2D платформер.

Приступим к разработке, во-первых, нужно создать персонажа и платформу. В первое время будем использовать обычные геометрические фигуры – прямоугольник и капсула.

Но если сейчас запустить игру, то ничего не произойдет, ведь наш персонаж не имеет веса. Добавим компонент Rigidbody2D для работы с физикой. Теперь капсула просто провалилась сквозь платформу. Добавим на героя и платформу BoxCollider 2D, который сделает объекты «жесткими».

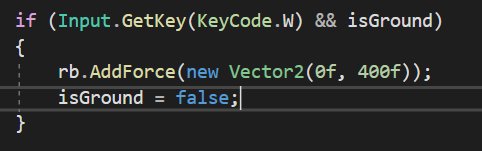
В коде мы будем использовать методы Update(Вызывается каждый кадр) и FixUpdate(Вызывается в определенный промежуток времени), а также метод Start(Вызывается 1 раз в начале). Сначала объявим публичную переменную speedX со значение 3, она будет отвечать за скорость, а так же в дальнейшем нам понадобится переменная horizontal со значением 0, создадим приватную переменную rb и присвоим ей Rigidbody2D с помощью GetComponent в методе Start.

Теперь напишем движение персонажа. Для этого будем использовать метод ввода Input и его функцию GetAxis(«Horizontal»), которую мы присвоим переменной horizontal. Она принимает значение -1 и 1. С его помощью мы будем указывать движение персонажа по оси X в зависимости от нажатой кнопки. Остается создать скорость движения персонажа с помощью rb.velocity. В нем мы пишем новый вектор 2 и перемножаем нашу скорость с переменной horizontal. Так же нужно умножить на Time.fixedDeltaTime, так как код физики пишется в FixedUpdate, который вызывается в определенный промежуток времени в отличии от Update, из-за этого FixedUpdate может вызваться чуть раньше или позже для этого и нужно умножить на Time.fixedDeltaTime. После создадим и домножим на переменную Multiply, которая равна 50. Дело в том, что Time.fixedDeltaTime равен 0,02 и мы увеличиваем это значение, иначе персонаж будет двигаться очень медленно.

Пришло время сделать прыжок. Прыжок – это сила с которой игрок отталкивается от земли. Эту силу мы сможем задать с помощью rb.AddForce(). Напишем код, что, если будет зажата клавиша W, наш персонаж подпрыгнет.

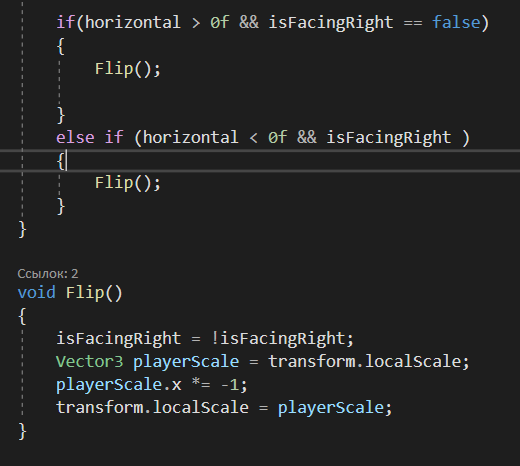
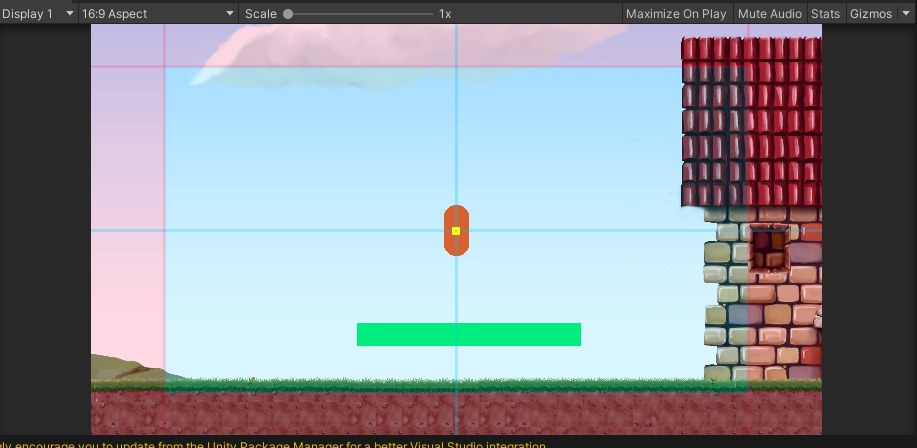
Но теперь наш герой может прыгать в полете, надо это исправить. Вводим новую булевую переменную isGround со значением false. Данная переменная будет говорить о том, коснулся персонаж земли или нет.

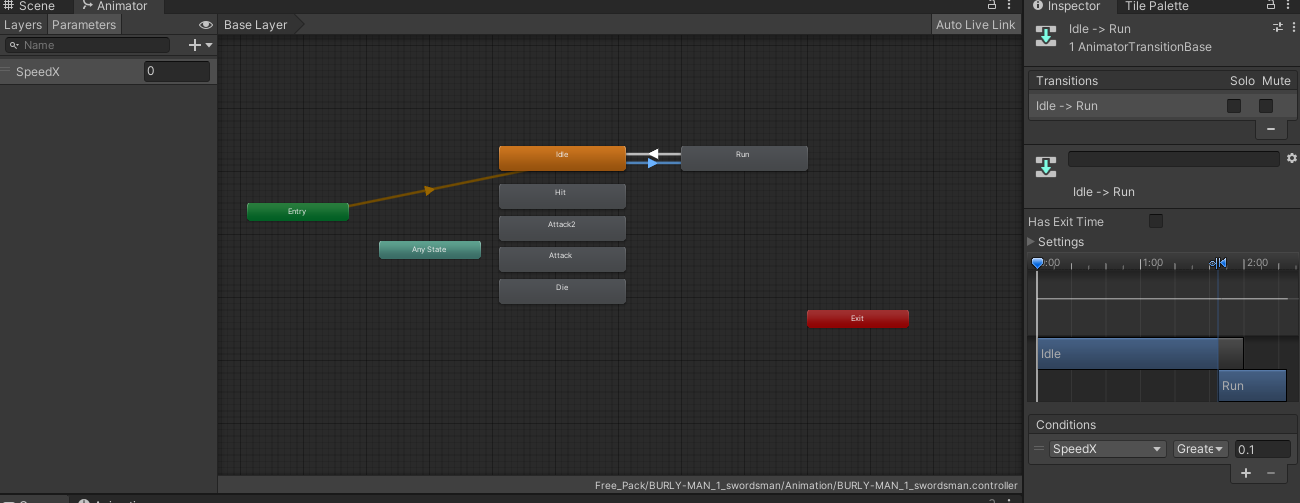
В первую очередь нужно дать тег Ground нашей платформе. Далее мы применим метод OnCollisionEnter2D. Он будет проверять, к чему прикасается Collider. Пропишем условие, что, если к нашему объекту с тегом Ground прикоснется объект имеющий Collider, то мы isGround делаем true. Допишем в коде прыжка еще одно условие, что isGround = true и в конце кода прыжка сделаем isGround = false. Теперь наш персонаж может бегать и прыгать.



**Графика и анимация персонажа.**

В первую очередь сделаем так, чтобы камера следила за персонажем. Это можно сделать разными путями, но лучше установить модуль Cinemachine. Cinemachine – это камера с гибкими настройками, она значительно упрастит нашу работу.

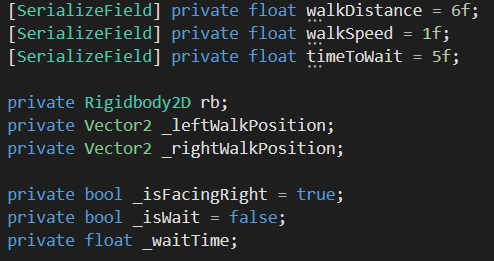
****Скачаем из Unity Asset Store бесплатные спрайты персонажа и окружения. Создаем простой уровень из ассетов, а вместо капсулы ставим героя. Необходимо перенести все компоненты: скрипты, теги, коллайдеры – нашим спрайтам. У нас есть персонаж с анимацией покоя, нужно сделать ему поворот, что если у него горизонтальная скорость меньше нуля он поворачивался в лево, а если больше , то в право. Введем переменную isFacingRight = false и будем проверять, если он идет в право и смотрит назад, то поворачиваем и наоборот.

У нас есть 6 анимаций, при запуске игры работает только idle(состояние покоя), сделаем переход из анимации idle в run и наоборот. Зайдем в аниматор и создадим из состояния покоя стрелочку к бегу и наоборот. Вводим переменную SpeedX и добовляем ее на каждую из стрелочек. Ставим условие, что если SpeedX больше 0.1, то срабатывает анимация бега, если меньше, то анимация покоя.

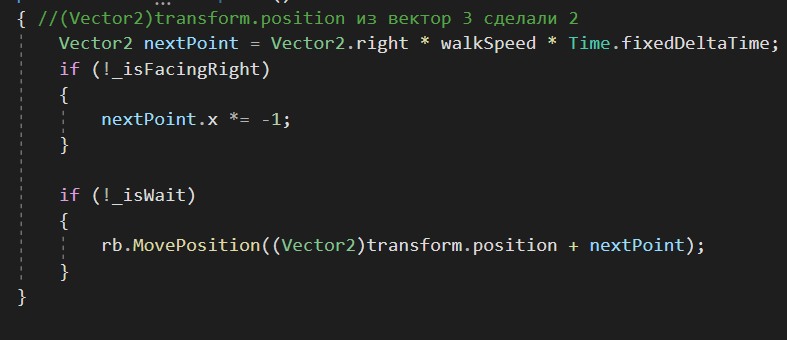
Теперь нашей переменной SpeedX надо передовать значение скорости, напишем код. Создаем переменную animator, которой мы через [SerializeField] присвоем доступ к функциям аниматора В методе Update через animator с помощью функции SetFloat обращаемся к перемменной SpeedX и присваиваем ей horizontal. Но так как horizontal иммеет значение -1 и 1, наш персонаж будет двигаться с анимацией только в одну сторону. Возьмем переменную horizontal в модуль (Abs), используя библиотеку Mathf.



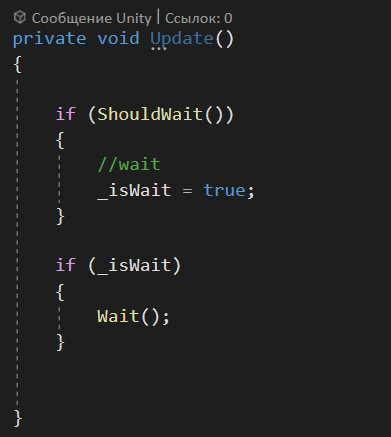
**Первый враг.**

Для начала научим нашего врага патрулировать местность. Он будет двигаться с одной точки до другой, разворачиваться, ждать и идти обратно. Создадим переменные:

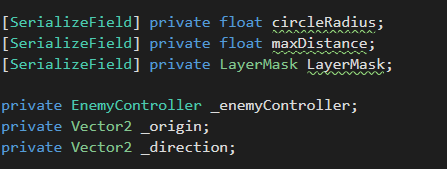
В методе FixedUpdate пропишим код движения. Создадим Vector2 nextPoint и присвоим ему новый Vector.2 со значением (1,0), умножим на скорость (walkSpeed), и на Time.fixedDeltaTime. Будем проверять, если !\_isFacingRight. Тогда nextPoint.x \*= -1. Добавим еще проверку для ожидания, если !\_isWait, тогда двигаем врага с помощью rb.MovePosition.



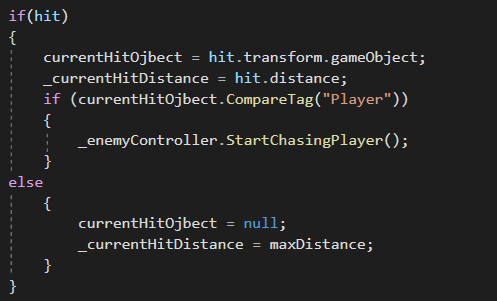
Создадим функцию ShouldWait и напишем две переменные типа bool, isOutofRightBoundery, который будет равен \_isFacingRight и при этом позиция больше или равна \_rightWalkPosition.x. Так же делаем со второй переменной, но ставим знак меньше или равно. В методе Start rb присвоим Rigidbody2D, \_leftWalkPosirion присвоим текущее положение, а \_rightWalkPosition присвоим текущее положение левого плюс walkDistance, и умножим на new Vector2(1, 0).

Пропишим код таймера ожидания создадим функцию Wait. От переменной \_waitTime отнимаем Time.deltaTime, и проверяем \_waitTime < 0f. Присваеваем \_waitTime время ожидания, после \_isWait = false, и поворачиваем его. В методе Update сделаем проверку, если наш враг дошел до крайней точки, то \_isWait = true. Ниже напишим еще одну проверку, если \_isWait = true, тогда используем функцию Wait.

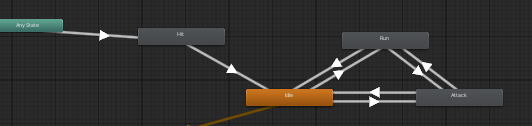
Теперь заставим врага приследовать, когда он видит нашего героя. Создадим зрение врагу с помощь невидемого луча RayCastHit2D. Для этого нам нужно 5 переменных.



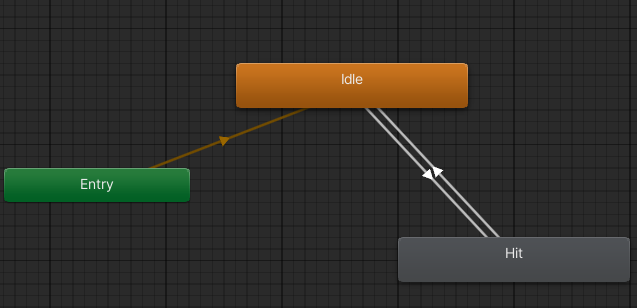
В методе Updete пропишем значение origin. Далее из кода EnemyController возьмем перемменую isFacingRight. Чтобы знать куда смотрит наш враг. Создадим условие, что если Enemy смотрит вправо, то \_direction присваеваем Vector2.right, и наоборот. Создадим лучь RaycastHit2D hit. В его значение укажем точку появления, длину, радиус, максимальную дистанцию и LayerMask. С помщью LayerMask лучь будет понимать, что перед ним находится герой.

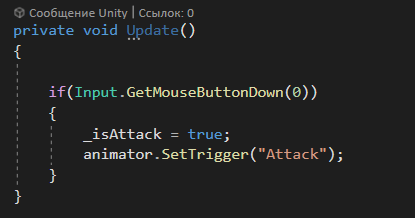
В коде EnemyController изменим немного nextPoint. Наша задача сделать так чтобы враг при виде героя начинал его приследовать, c большей скоростью. Если же враг потеряет нашего персонажа, то он перейдет обратно в режим патруля. В nextPoint будем менять переменную walkSpeed в зависимости от режима Enemy. Создаем условие в коде EnemyVision. Если перед лучем будет gameObject с тегом “Player”, тогда мы начинаем режим преследование, в ином случае включаем режим патруля.

**Система боя.**

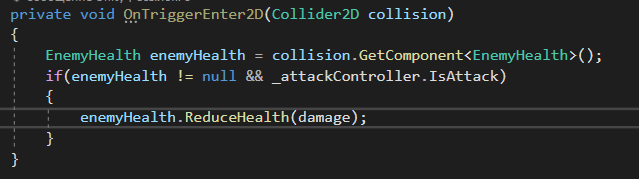
Сделаем так, чтобы герой и враг могли наносить друг другу урон при столновении Коллайдеров. Дла начала создадим анимации атаки и урона у нашего героя, а также это и для врага.

Пропишем две булевые перременные для перехода анимаций. Attack и takeDamage. А у Enemy создадим только переменную takeDamage.

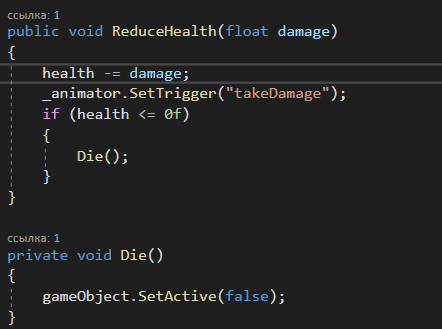


Сделаем два новых скрипта enemyHealth и PlayerAttack, а также нам понадобится еще один скрипт AttackController. В AttackController в методе Update напишем, что если нажата левая кнопка мышки, то мы переходим в режим атаки и IsAttact делаем true,а также проигрываем анимацию.

В скрипте PlayerAttack создадим переменную damage(урон) со значением 20. В методе OnTriggerEnter2D проверим. Если enemyHealth не равно нулю, и isAttack = true, тогда отнимаем здоровье.



В скрипте enemyHealth создадим функцию, которя принимает урон. А еще нам понадобится переменная health со значением 100. Напишим условие, что если health меньше 0, то переходим к функции Die. Ниже создаем ее и пишим gameObjectSetActive(false). С помощью этой строчки кода мы удалим объект.

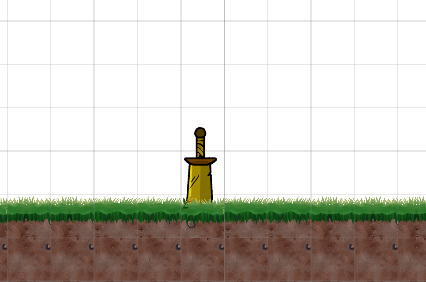


Точно такоже же код пишим для enemyAttack и PlayerHealth. И разбрасываем врагов по всей карте.

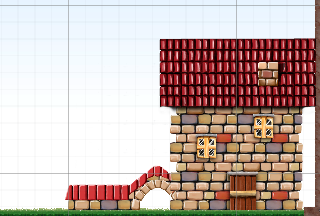
**Конец игры.**

Осталось сделать концовку, работать это будет так, на карте есть «рычаг», если подойти к нему и нажать букву «F», то при нажатии на «F» около «домика» игра завершится.

Рычаг:

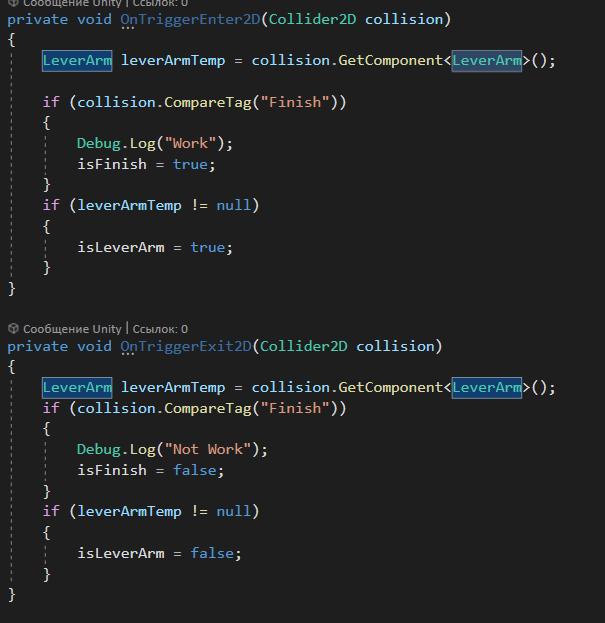


Выход:

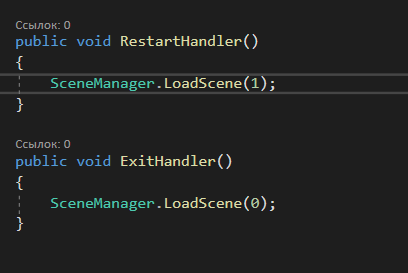


Создадим скрипт для рычага LeverArm, и для выхда Finish. В коде Controller напишем переменные isFinish, isLevelArm – равные false, а также приватные переменные finish, levelArm. В методе Update пропишем условие, что если нажата кнопка «F», то будут проверятся два условия. Если перед нами isFinish, то будем вызывать функцию FinishLevel(), и если перед нами isLevelArm, то будем вызывать функцию ActivateLevelArm().

Создаем метод OnTriggerEnter2D в котором будет два условия, если наш персонаж сопрекоснется с collision у которой тег «Finish», то isFinish ставим true. Напишим переменную leverArmTemp и с помощью GetComponent присваеваем ей LevelArm. Проверяем, если levelArmTemp != null, тогда isLevelArm присваеваем true. Во втором методе OnTriggerExit2D пишем точно такой же код, но isLevelArm и isFinish меняем на false.

  
Сейчас мы сделаем окно GameOver, a также меню, и окно победы. Создаем Convas GameOver с двумя кнопками Restart и Exit, а так же делаем уго прозрачным. В новом скрипте пишим код.



Устанавливаем библиотеку UnityEngine.SceneManagement. Пропишим два метода, RestartHandler() и ExitHandler(). Наша текущая сцена находится под индексом «1», а меню будет под «2». В RestartHandler загрузим «1» сцену.(SceneManager.LoadScene(1)). В методе Exithandler делаем тоже самое только загружаем «0» сцену. Привяжим наши функции к кнопкам, и в коде PlayerHealth в функции Die включаем gameOver.

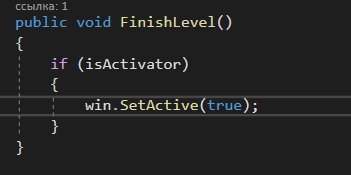
Нам понадобится новая сцена. В ней мы таким же образом создаем Convas и делаем меню. Но в ExitHandler прописываем Application.Quit(). С его помощью мы сможем выйти из игры.



Точно таким же образом делаем окно победы, но с одной кнопокой.



Доработаем финиш. Его код будет таким же, как и у «рычага». Создадим функцию FinishLevel и будем проверять условие, если isActivator = true, тогда делаем окно win видимым.



**Заключение.**

Рассматривая выполненную мной работу можно прийти к выводу, что гипотеза, которую я выдвинул в начале проектной работы, ошибочна. Трудно создать игру не зная азов программирования, но любой сможет этому научиться, приложив немного усилий. Лично для меня проект оказался крайне интересным, позволил понять глубже процесс создания игры, в ходе создания я узнал много нового. К сожалению, проект не выполнен в полной мере, т.к. процесс создания игры довольно трудоемкий и требует много времени.

**Список литературы.**

1. Информация о Unity(игровой движок): <https://ru.wikipedia.org/wiki/Unity_(%D0%B8%D0%B3%D1%80%D0%BE%D0%B2%D0%BE%D0%B9_%D0%B4%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA)>
2. История появления компьютерных игр: <https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%82%D0%BE%D1%80%D0%B8%D1%8F_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D1%85_%D0%B8%D0%B3%D1%80>
3. Обучающие видео по программированию на C#: <https://www.youtube.com/watch?v=mpdthsdB-W0&t=556s> <https://www.youtube.com/watch?v=Mwvkt4UqW-Q&t=47s>
4. Как пользоваться интерфейсом Unity: <https://www.youtube.com/watch?v=nRGOW9O7ARk&t=2349s>