**Проект**

**По Основам Программной Инженерии**

**Игра «ONCE IN THE GYM»**

**Белорусский Государственный Технологический Университет**

**Содержание**

1. Системное проектирование
   1. Игровой движок
   2. Игровые события
   3. Столкновение с противником (Событие «Битвы»)
   4. Сохранение и загрузка
2. Разработка программных модулей
   1. Класс **Vector2**
      1. Переменные
      2. Конструктор
      3. Функции **Length()**, **Dot()**, **Angle()**
      4. Операторы
   2. Класс **GameObject**
      1. Переменные
      2. Перечисление **ObjectType**
      3. Конструктор
   3. Класс **Enemy**
      1. Переменные
      2. Конструктор
   4. Класс **GameMap**
      1. Переменные
      2. Конструктор
      3. Функция **getSize()**
      4. Функция **addGameObject()**
      5. Функции **deleteGameObject()** и **deleteGameObjectById()**
      6. Функция **getGameObjectsCount()**
      7. Функции **getGameObjectByMapPosition(), getGameObjectByTag(), getGameObjectById()**
      8. Функции **getGameObjectsByMapPosition(), getGameObjectsByTag(), getGameObjectsByType()**
      9. Функция **Render()**
3. Тестирование и отладка приложения
   1. Ошибка C2061 и C2065
   2. ASCII проблема
   3. Глобальные переменные
4. Руководство пользователя
   1. Как играть?!
5. Заключение
6. Система проектирования

Программа была поделена на четыре куска, за каждый из которых кто-то взялся:

Окулич Дмитрий – Игровой движок

Илья Дмитрук – Игровые события

Саша Богданов – Столкновение с противником

Саша Рубашек – Сохранение и загрузка

* 1. Игровой движок

Игровой движок — это основа всей игры. Он используется как для отображения карты, так и для расчета векторов, физики столкновения и т.д. Само отображение работает по принципу матрицы из символов. Точнее мы имеем массив из игровых объектов и у каждого из них есть свой символ и для вывода карты, в зависимости от их позиции, берем их символы, а если на какой-то позиции объекта мы его заполняем стандартным символом (у каждой карты он может быть свой). А также для более удобной работы с двумерным пространством был введён класс Vector2. Он просто хранит два значения (x и y). Ещё объекты могут входит или быть рядом с другими объектами, для этого есть статический класс Collision. Благодаря ему мы можем считать входят ли объекты в друг друга или же находятся радом с друг другом (в этом случае можно получить сторону с которой находиться второй объект относительно первого).

* 1. Игровые события

Игровые события дополняют игровой мир. Благодаря им строится сюжет. Они запускают битвы, дают различные эффекты (увеличение или уменьшение жизни, брони, урона и т. д), а также загружают новые локации. События обычно происходят при взаимодействии с неким объектом. Есть три вида объекта: враги, вещи и переход на следующую карту. На картах они располагаются так, чтобы вы могли пройти игру. Например, в начале вы сможете найти компот (вещь, которая восстанавливает жизнь) после нескольких врагов. На протяжении игры вы будете встречать вещи, которые будут увеличивать ваш урон. И в конце игры ваш урон будет очень большим, и вы сможет убивать врагов за 2-3 удара, таким образом они не будут успевать нанести вам большой урон, и вам не нужен будет компот. И поэтому компота в конце будет мало. А как работают события с технической стороны? Мы создаём функцию с событием. В неё записываем то, что должно произойти. Затем создаём на карте объект, при взаимодействии с которым должно произойти событие. Затем в условном операторе делаем проверку: находится ли игрок в данном объекте. Если да, то вызывается функция с событием и событие происходит.

* 1. Столкновение с противником (Событие «Битвы»)

Битва является неотъемлемой частью геймплея OG. Я создал класс Enemy с количеством здоровьем, уроном, защитой, именем и описанием. Характеристики Enemy балансировались в случайном порядке. Характеристики игрока существуют в виде глобальных переменных, которые сбалансированы основываюсь на характеристиках Enemy. При попадании события битвы мы уведомляем что игрока атакуют. У нас открывается меню, где мы видим имя противника, его описание, здоровье и последний нанесённый урон по нам, наша кол-во здоровья, наша кол-во стамины, и нам предоставляется выбор: Fight(атаковать), Stat(Информация о противнике) и Rest(отдых – восстановление стамины). При выборе Fight нам открывается меню: Hand(удар рукой, при котором снимается 1 единица стамины), Leg(удар ногой, при котором снимается 3 единицы стамины), Cancel(вернуться в предыдущее меню). При выборе Stat мы получаем информацию о противнике: имя, описание, здоровье, защита. При выборе Rest мы получаем урон от противника, восстанавливаем стамину(если это нужно).

* 1. Сохранение и загрузка

Сохранения и загрузка данных являются очень и важным и самое главное удобным инструментом. Эти два инструмента позволяют не потерять прогресс, полученный игроком во время игры. Мы реализовали их с помощью отдельного заголовочного файла «SaveLoad.h», и там уже ввели соответствующие функции «Save»-отвечающий за сохранение, и «Load» за загрузку. Мы использовали метод сохранения в бинарном виде, это безопасней и функциональней. Сохранения происходят автоматически, при перемещении на следующую локацию. Мы сохраняем 6 значений: Максимальное здоровье, максимальную выносливость, здоровье, урон, броню и карту.

Возможность загрузить игру осуществляется в начале программы, где пользователь выбирает между началом новой игры и загрузкой старой.

1. Разработка программных модулей
   1. Класс **Vector2**

Это класс необходимый работы с двухмерный пространством, содержит лишь две переменные, 3 оператора, 3 статические функции и 2 перегрузки конструктора

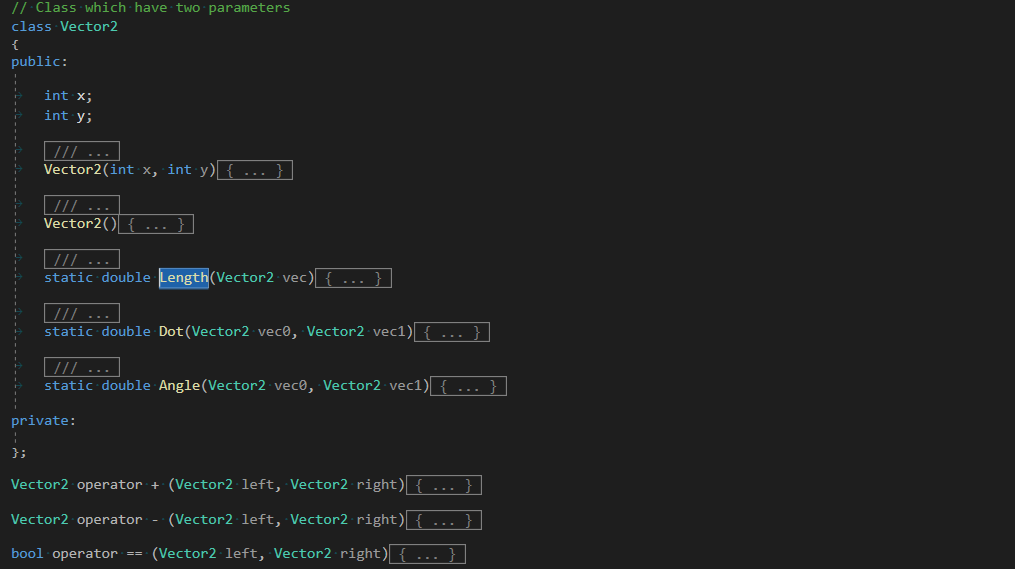
****

Рисунок 2.1

* + 1. Переменные

Данный класс содержит лишь две переменные:

X – Положение вектора по оси X

Y – Положение вектора по оси Y



Рисунок 2.1.1(1)

* + 1. Конструктор

Конструктор в данном классе имеет две перегрузки:

Перегрузка по умолчанию. Она инициализирует переменные класса забивая их нулями.



Рисунок 2.1.2 (1)

Вторая перегрузка конструктора будет требовать целочисленные переменные для координат по X и Y.



Рисунок 2.1.2 (2)

* + 1. Функции **Length**(), **Dot**(), **Angle**()

Это функции для работы с векторами:

Функция Length() запрашивает в виде аргумента один вектор(Vector2), а возвращает длину этого вектора в виде числа с плавающей точкой.

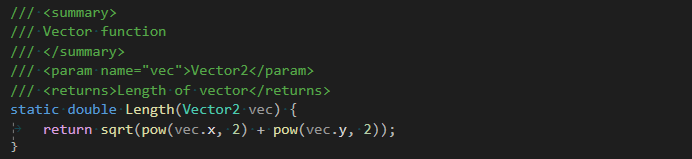


Рисунок 2.1.3(1)

Функция Dot() запрашивает в виде аргументов два вектора, а возвращает скалярное произведение этих векторов в виде числа с плавающей точкой.

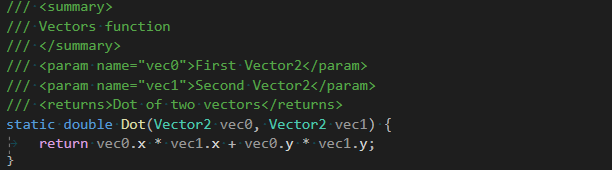


Рисунок 2.1.3(2)

Функция Angle() запрашивает в виде аргументов два вектора, а возвращает угол между ими в виде числа с плавающей точкой.

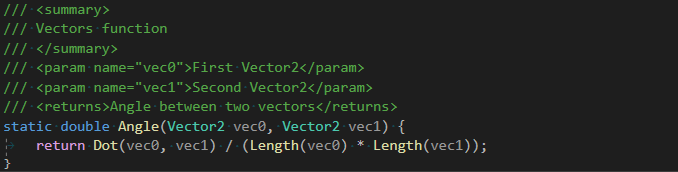


Рисунок 2.1.3(3)

* + 1. Операторы

Для этого класса есть лишь три оператора:

(+) - складывает координаты двух векторов.

(-) – Отнимает координаты двух векторов.

(= =) – Сравнивает координаты двух векторов и, если они совпадают возвращает True, иначе False.

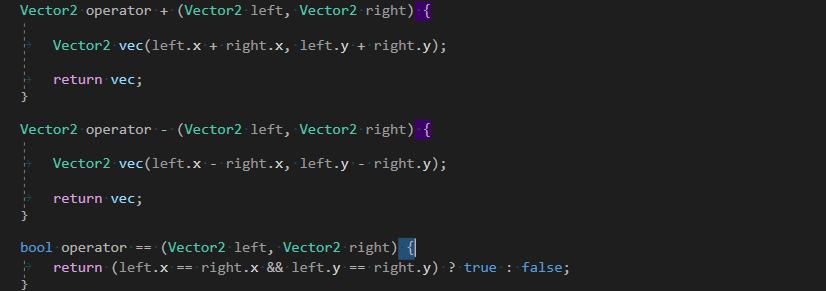


Рисунок 2.1.4

* 1. Класс **GameObject**

Это класс необходимый для хранения информации об игровом объекте

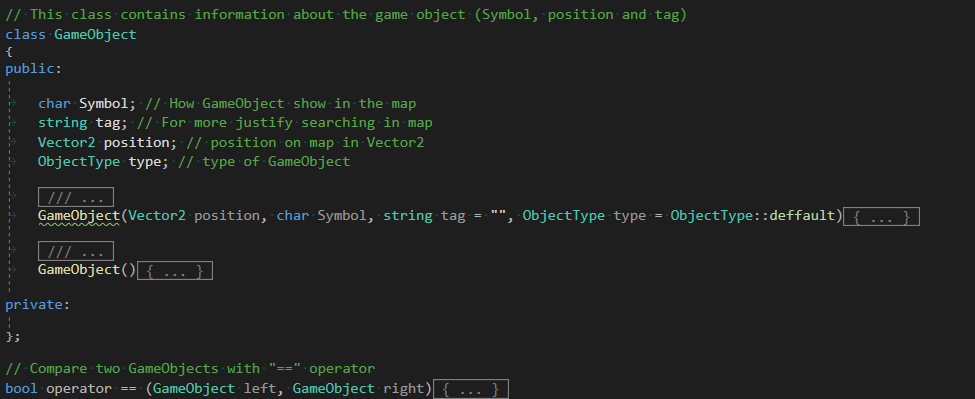
****

Рисунок 2.2

* + 1. Переменные

Игровой объект(GameObject) имеет 4 переменные:

Symbol – Это символьная переменная определяет как объект будет отображаться на карте(GameMap)

tag – Строковая переменная из внешней библиотеки <string> определяет тег объекта(кодовое имя объекта)

position – позиция объекта в пространстве через класс Vector2

type – тип объекта (задаётся при помощи перечисления ObjectType)

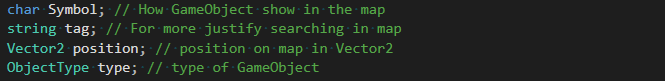


Рисунок 2.2.1

* + 1. Перечисление **ObjectType**

Это перечисление обозначает тип объекта:

Default – объект без каких либо особенностей

Wall – Обозначает что этот объект является стенной и не проходим

Action – Это значить что объект может быть объектом события



Рисунок 2.2.2

* + 1. Конструктор

Конструктор по умолчанию

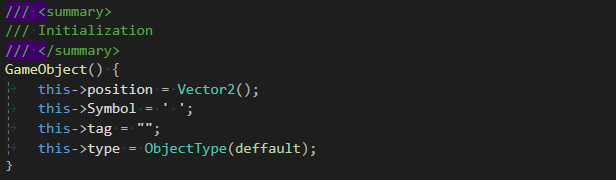


Рисунок 2.2.3(1)

Этот конструктор запаршивеет четыре аргумента. Из них два обязательных и два не обязательных.

Symbol – Это символьная переменная определяет как объект будет отображаться на карте(GameMap)

tag – Строковая переменная из внешней библиотеки <string> определяет тег объекта(кодовое имя объекта)

position – позиция объекта в пространстве через класс Vector2

type – тип объекта (задаётся при помощи перечисления ObjectType)

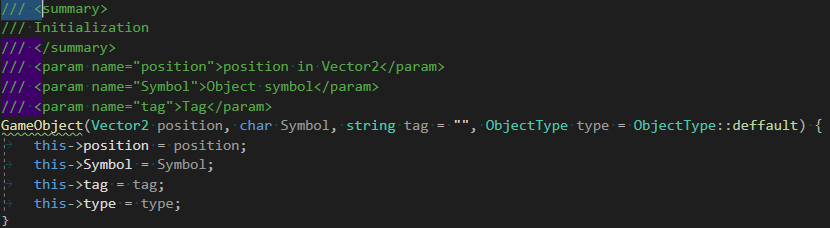


Рисунок 2.2.3(2)

* 1. Класс **Enemy**

Этот класс хранит информации о противнике, которого можно встретить по игре.

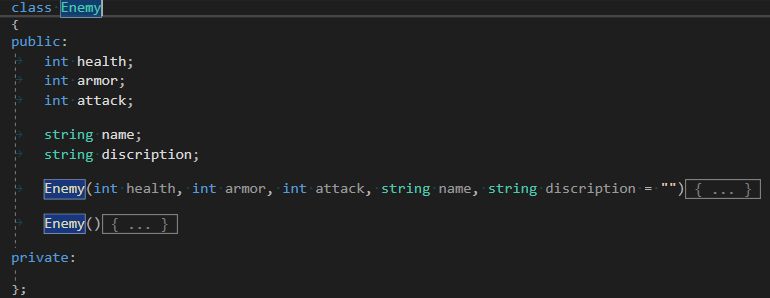


Рисунок 2.3

* + 1. Переменные

Данный класс содержит 3 целочисленных и 2 строковые (<string>) переменные:

Health – здоровье противника

Armor – Защита противника

Attack – Урон противника

Name – Имя противника

Discription – Описание противника



Рисунок 2.3.1

* + 1. Конструктор

Конструктор по умолчанию

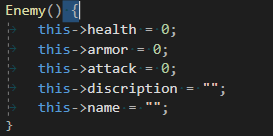


Рисунок 2.3.1(1)

Перегрузка, имеющая 4 обязательные и 1 не обязательный аргумент

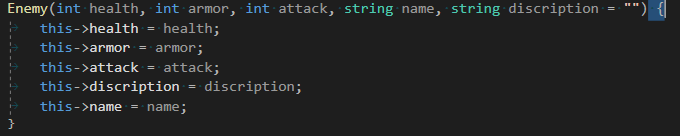


Рисунок 2.3.1(2)

* 1. Класс **GameMap**

Хранит внутри себя всю информацию об карте и его объектах

* + 1. Переменные

Этот класс содержит в себе 1 переменную с модификатором доступа public и 3 переменные с модификатором доступа private:

DefaultSymbol – Символьная переменная. Этим символом закрашиваются те позиции, где нет объектов.



Рисунок 2.4.1(1)

GameObjects\* – Это динамический массив(для массива использовался класс из внешней библиотеки <vector>) хранящий указатели всё игровые объекты которые есть на карте.

Map – Это двумерный динамический массив(<vector>) хранящий матрицу символов, которые буду выводиться на экран.

Size – Размер карты указанный в Vector2

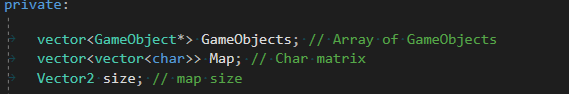


Рисунок 2.4.1(2)

* + 1. Конструктор

Конструктор по умолчанию:

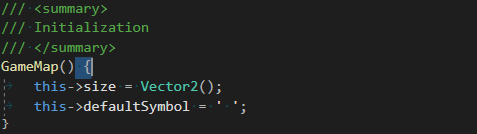


Рисунок 2.4.2(1)

Перегрузка, имеющая 2 обязательных аргумента. Он инициализирует символьную матрицу Map.

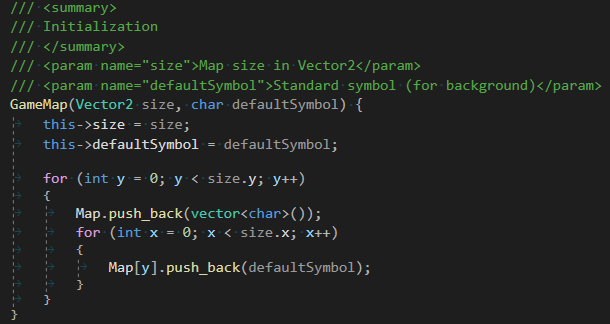


Рисунок 2.4.2(2)

* + 1. Функция **getSize()**

Возвращает размер карты в Vector2

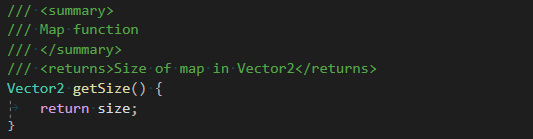


Рисунок 2.4.3

* + 1. Функция **addGameObject()**

Это функция принимает в виде аргумента указатель на объект и добавляет его в массив игровых объектов

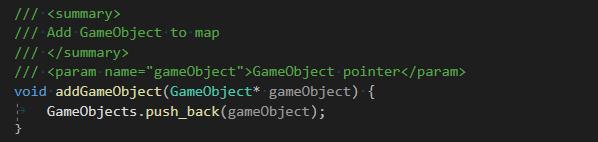
****

Рисунок 2.4.4

* + 1. Функции **deleteGameObject()** и **deleteGameObjectById()**

Это функции для удаления объектов из карты.

**deleteGameObjectById() –** принимает в виде аргумента идентификатор объекта. Удаляет объект под данным идентификатором.

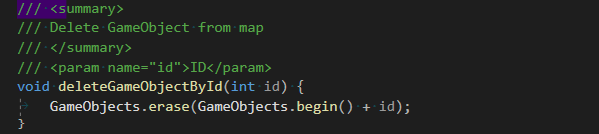


Рисунок 2.4.5(1)

**deleteGameObject()** **-** принимает в виде аргумента указатель на объект. Удаляет объект с таким же указателем.

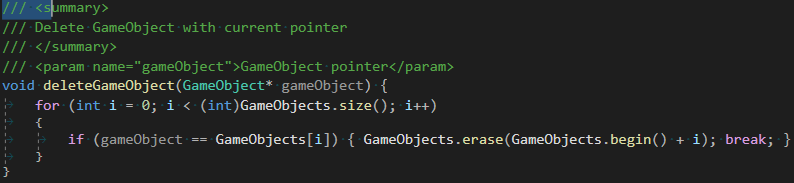


Рисунок 2.4.5(2)

* + 1. Функция **getGameObjectsCount()**

Возвращает кол-во объектов на карте

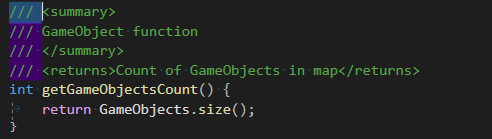


Рисунок 2.4.6

* + 1. Функции **getGameObjectByMapPosition(), getGameObjectByTag(), getGameObjectById()**

Эти функции для нахождения нужного объекта на карте. Все возвращают указатель на объект.

**getGameObjectById() –** За аргумент принимает идентификатор объекта.

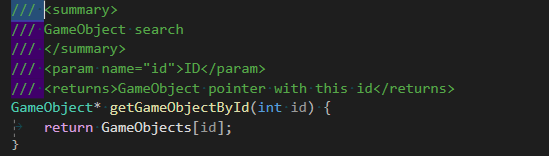


Рисунок 2.4.7(1)

**getGameObjectByTag() –** За аргументы принимает тег объекта(в виде string из <string>) и буллевую переменную которая нужна что бы отметить нужно ли выводить сообщение что объект не найден.

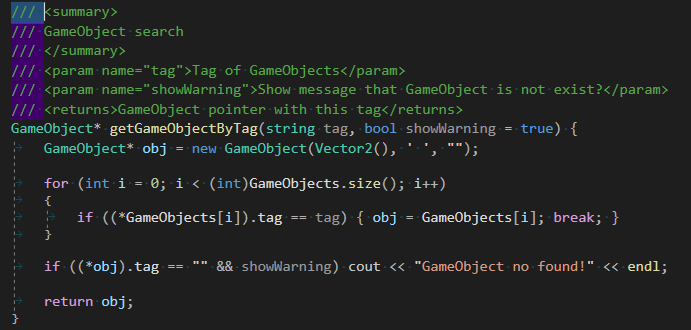


Рисунок 2.4.7(2)

**getGameObjectByMapPosition() –** За аргументы принимает позицию(в Vector 2) в которой может находиться объект и буллевую переменную которая нужна что бы отметить нужно ли выводить сообщение что объект не найден.

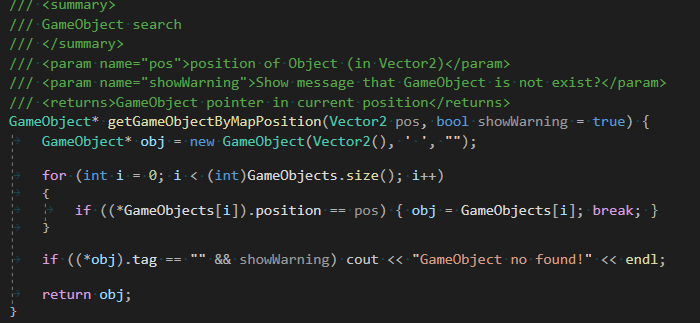


Рисунок 2.4.7(3)

* + 1. Функции **getGameObjectsByMapPosition(), getGameObjectsByTag(), getGameObjectsByType()**

Эти функции для нахождения нужных объектов на карте. Все возвращают динамический массив(дин. массив vector из <vector>) указателей на объекты

**getGameObjectsByTag() –** За аргументы принимает тег объектов(в виде string из <string>)

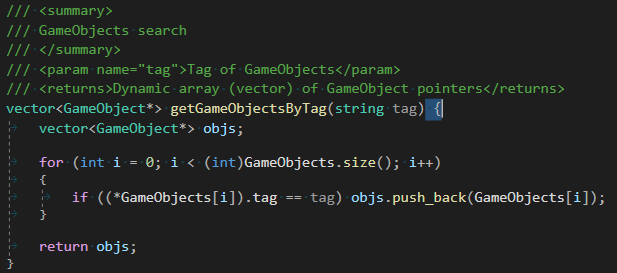


Рисунок 2.4.8(1)

**getGameObjectsByMapPosition() –** За аргументы принимает позицию(в Vector 2) в которой могут находиться объекты

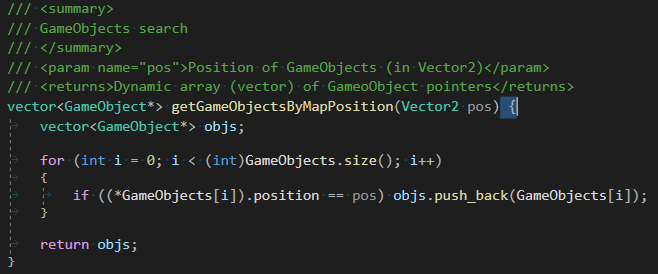


Рисунок 2.4.8(2)

**getGameObjectsByType() –** За аргумент принимает тип объектов(ObjectType)

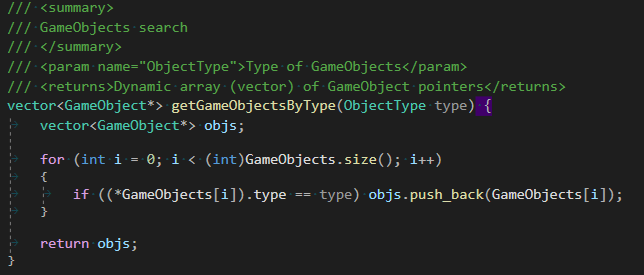


Рисунок 2.4.8(3)

* + 1. Функция **Render()**

Функция отрисовки. Работает по принципу прямого обращения к буферу консоли.

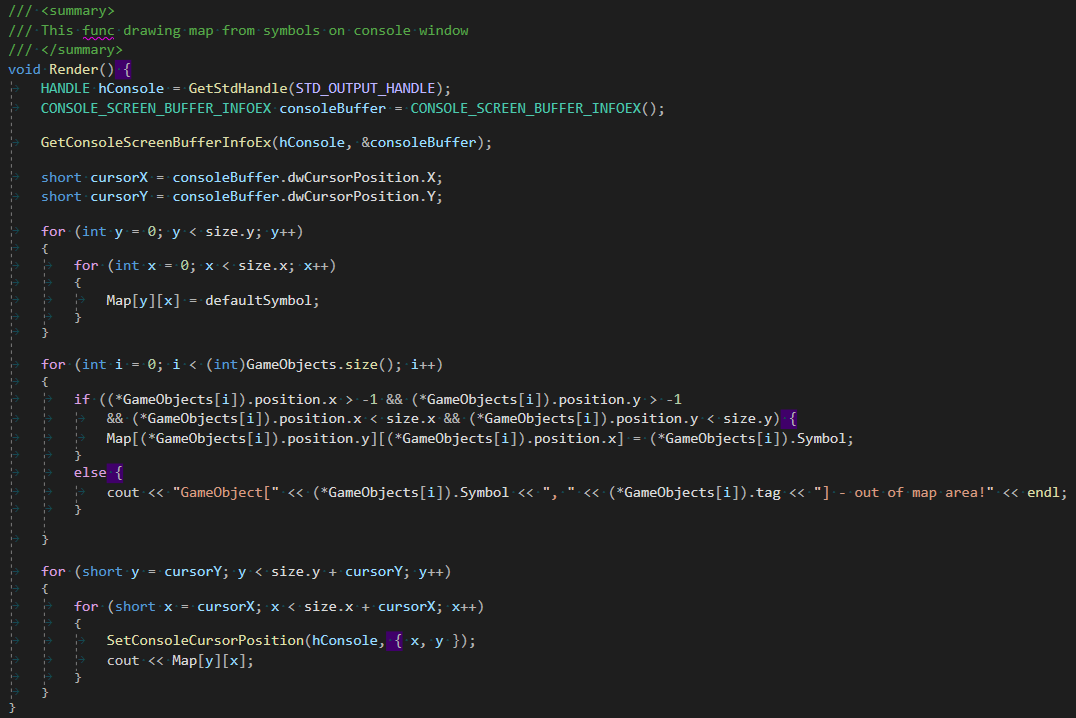


Рисунок 2.4.9(1)

Пример карты:

Рисунок 2.4.9(2)

1. Тестирование и отладка приложения

Во время создания и отладки игры мы столкнулись с некоторыми техническими проблемами.

* 1. Ошибка C2061 и C2065

Ошибка компилятора C2061 и C2065. С этой проблемой мы столкнулись, когда начали создавать много заголовочных файлов. Как мы узнали это ошибка встречается, когда компилятор обнаружил идентификатор, которого не ожидал. Мы перепробовали много способов как решить это проблему, но ничего не помогало.

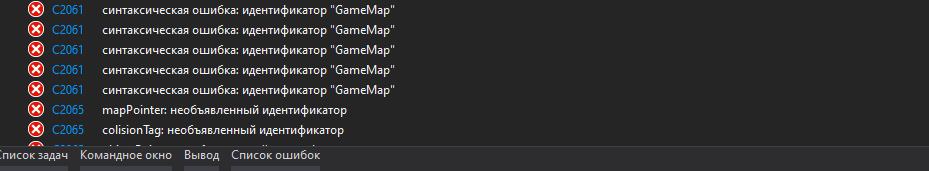


Рисунок 3.1(1)

Однако мы поняли всё-таки поняли в чём была проблема. Проблема заключалась в том, что файлы вызывали друг друга и компилятор обращался одновременно к первому и второму файлу и, похоже, происходил конфликт и программа сыпалась.

GraphicEngine.h Physics.h

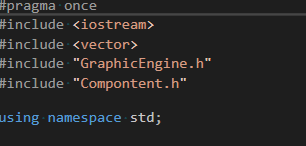
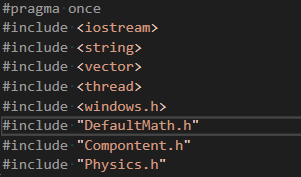


Рисунок 3.1(2) Рисунок 3.1(3)

* 1. ASCII проблема

Мы столкнулись с проблемой что GitHub не считывает ASCII(1251) он всё переводит в UTF-8 и это превращало все русские символы в кракозябры.

****

Рисунок 3.2

К сожалению, мы ничего лучше не придумали как просто на данной стадии разработки игры отказаться от русского языка и из-за этого сейчас управление не работает если у игрока включена русская раскладка клавиатуры.

* 1. Глобальные переменные

Мы решили создать отдельный файл для хранения глобальный переменных. Вроде всё хорошо, да? Сначала мы просто создали ещё одни файл исходного кода(.cpp) однако даже на этой стадии мы столкнулись с тем что .cpp файлы не так просто с друг другом связывать. Но даже когда мы смогли файл к главному файлу то мы все равно столкнулись с ошибкой компилятора C2065 (пункт 3.1).

Решение этой проблемы мы нашли в перемещении переменных в заголовочный файл(.h). Да, так просто.

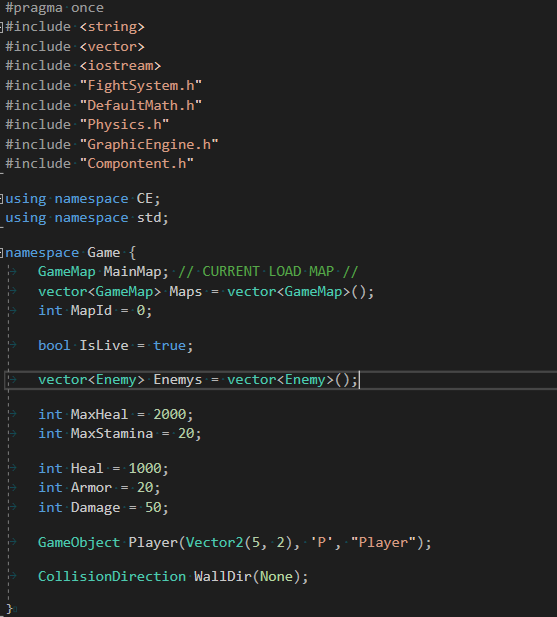


Рисунок 3.3

1. Руководство пользователя

После запуска игры вас встретит главное меню. Здесь вам нужно будет выбрать начать новую игру или загрузиться из уже существующего.

Менять выбор на клавиши A, D.

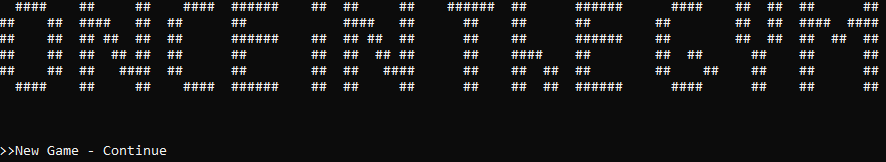


Рисунок 4(1)

Допустим вы начали новую игру. Вас перекинет на первую карту(карта на рисунке 4.1(2) выбрана было произвольно). Вам будет отображаться сама карта, ваше здоровье и защита. Перемещение по карте работает при помощи клавиш W,A,S,D.



Рисунок 4(2)

На протяжении игры вы можете натыкаться на разные события. Это может быть как битва, увеличение или уменьшение ваших игровых параметров так и просто какое не будь сообщение.



Рисунок 4(3)

Если вы вступите в битву вам будет отображаться здоровье и последний нанесённый по вам урон, а также ваше здоровье и выносливость(стамина). У вас будет выбор(Перемещение по выбору при помощи клавиш W, S): атаковать, проверить, отдохнуть. Если вы атакуете, то у вам ещё предстоит выбрать как атаковать и тогда вы нанесёте урон, но помните враг тоже будет атаковать. Если вы решите проверить, то вам покажут описание, урон, защиту противника, при этом это не считается за ход(противник не будет атаковать). После каждой атаки у вас будут сниматься очки выносливости и если их будет слишком мало, то вы не сможете атаковать. Что бы восстановить очки выносливости вам нужно выбрать «отдохнуть(rest)». Очки выносливости немного восстановятся, однако это считается за ход, а значит противник будет атаковать.

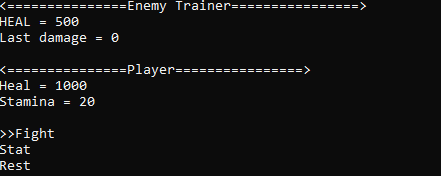


Рисунок 4(4)

Если ваше здоровье достигнет 0, то вы проиграете, и игра закроется.

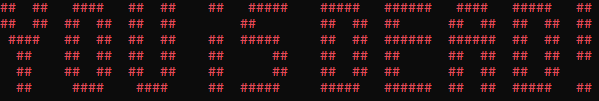


Рисунок 4(5)

1. Заключение

Работа над этим проектам научила нас работать в команде. Мы научились разрабатывать ПО по технологии канбан. Мы так же усовершенствовали свои умения пользования GitHub. Ещё мы открыли для себя новую библиотеку c++ <fstream> для работы с файлами. В ходе работы мы сталкивались с ошибками, которые в большинстве мы смогли решить. Улучшили навыки написания и понимания кода. Так же научились обращаться напрямую к буферу консоли. У нас появилось понимание о разработки ПО в команде. Если подытожить нашу работу то это был несомненно важный опыт.

