Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Программное игровое средство «3d shooter»

|  |  |  |
| --- | --- | --- |
| Студент |  | Е. Н. Траханов |
| Руководитель |  | Д.С. Шулицкий |

Минск 2019

СОДЕРЖАНИЕ

[Введение 5](#_Toc513665878)

[1 Анализ предметной области 7](#_Toc513665879)

[1.1 Обзор аналогов 7](#_Toc513665880)

[1.2 Постановка задачи 9](#_Toc513665881)

[2 Разработка программного средства 10](#_Toc513665882)

[2.1 Структура программы 10](#_Toc513665883)

[2.2 Интерфейс программного средства 10](#_Toc513665884)

[2.3 Работа со звуком](#_Toc513665885) 13

[2.4 Прорисовка карты и других объектов 1](#_Toc513665888)4

[2.5 Взаимодействие с объектами и игровая логика](#_Toc513665889) 18

[2.6 Работа с таблицей рекордов](#_Toc513665889) 21

[3 Тестирование программного средства](#_Toc513665890) 26

[4 Руководство пользователя](#_Toc513665891) 27

[4.1 Правила игры](#_Toc513665892) 27

[4.2 Интерфейс программы](#_Toc513665893) 27

[4.3 Противники](#_Toc513665893) 27

[Заключение](#_Toc513665894) 29

[Список использованных источников](#_Toc513665895) 30

[Приложение А. Исходный код программы](#_Toc513665896) 31

ВВЕДЕНИЕ

На сегодняшний день наблюдается бурное развитие компьютерной техники и информационных технологий в целом. В современном мире информационные технологии распространены повсеместно, что позволяет работать с большими объёма информации за относительно малое время, делая жизнь современного человека более удобной. Немалую роль в этом сыграли компьютерные игры, которые привили интерес к вычислительной технике более широкой группе людей, что стимулировало продажи, и, следовательно, развитие компьютерной техники. Нередко компьютерные игры становились целью, на пути к достижению которой разработчики создавали новые технологии, использование которых выходило за рамки компьютерной игры. Так, например, операционная система UNIX — прародитель многих современных операционных систем, таких как Linux, Android, Mac OS X была создана для запуска игры  «Space Travel», написанной Кеном Томпсоном.

Попытки создания игр 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 "ОХО" ("Крестики нолики"), в одиночку сделанная А.С. Дугласом в далёком 1952 году.

На сегодняшний день наиболее популярны трёхмерные игры, история которых началась в 1981 году.

Первой трёхмерной игрой для персональных компьютеров считается 3D Monster Maze. Стоит отметить, что графическое изображение в игре имеет низкое [разрешение](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_(%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0)) и выполнено символами [псевдографики](https://ru.wikipedia.org/wiki/%D0%9F%D1%81%D0%B5%D0%B2%D0%B4%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0).

Наиболее популярной первой трёхмерной игрой стала Wolfenstein 3D, где для отрисовки игрового мира использовался алгоритм ray-casting. Данный метод является одним из основных методов рендеринга трёхмерной графики. Первый алгоритм рейкастинга (не рейтрейсинга), используемый для рендеринга, был представлен Артуром Аппелем в 1968 году. В основе рейкастинга стоит идея испускать лучи из «глаз» наблюдателя, один луч на [пиксель](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D0%BA%D1%81%D0%B5%D0%BB%D1%8C), и находить самый близкий объект, который блокирует путь распространения этого луча.

Также для рендеринга трёхмерной графики используются такие алгоритмы, как растеризация, raytracing трассировка путей[1].

Наиболее похожий на ray casting является метод ray tracing, в котром также из точки наблюдения на объекты сцены направляются лучи, с помощью которых определяется цвет пиксела на двумерном экране. Отличие заключается в том, что  при этом луч не прекращает своё распространение, а разделяется на три луча-компонента, каждый из которых вносит свой вклад в цвет пикселя на двумерном экране: отражённый, теневой и преломлённый. Количество таких компонентов определяет глубину трассировки и влияет на качество и фотореалистичность изображения.

Целью данного курсового проекта является разработка игрового программного средства «3D Shooter», использующий технологию ray-casting и, подобно 3D Monster Maze, представляющий изображение с помощью символов псевдографики. Стоит также отметить, что данная программа будет полностью в консольном режиме, при этом используя только некоторые функции WinAPI. Таким образом данное ПО не имеет сильной привязки к конкретной библиотеке, что позволяет сосредоточиться на самом изучении принципов трёхмерной графики, а не на её реализации в какой-либо графической библиотеке, также это делает проект более гибким в плане дальнейшей визуализации.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

Пожалуй, наиболее популярными играми с использованием псевдотрёхмерных движков являются Wolfenstein 3D и Doom, разработанные студией id Software.

Особенностями игры Wolfenstein 3D является то, что один из её создателей, Джон Кармак, использовал необычный в то время подход с отображением графики с помощью [метода бросания лучей](https://ru.wikipedia.org/wiki/Ray_casting) (Ray casting, алгоритм которого будет рассмотрен далее в разделе 2.4), при котором требуется рассчитывать только поверхности, видимые игроку, а не всё окружающее его пространство.



Рисунок 1.1 – «Wolfenstein 3D», аналог игры «Shooter 3D»

Игра Doom (1993 г.) также создавалась под предводительством Джона Кармака. Движок создавался на [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) на рабочих станциях  [NeXT](https://ru.wikipedia.org/wiki/NeXT) на операционной системе [NEXTSTEP](https://ru.wikipedia.org/wiki/NEXTSTEP).



Рисунок 1.2 – «Doom», аналог «Shooter 3D»

Самой первой 3D игрой для персональных компьютеров было 3D Monster Maze, выпущенная в 1981 году.

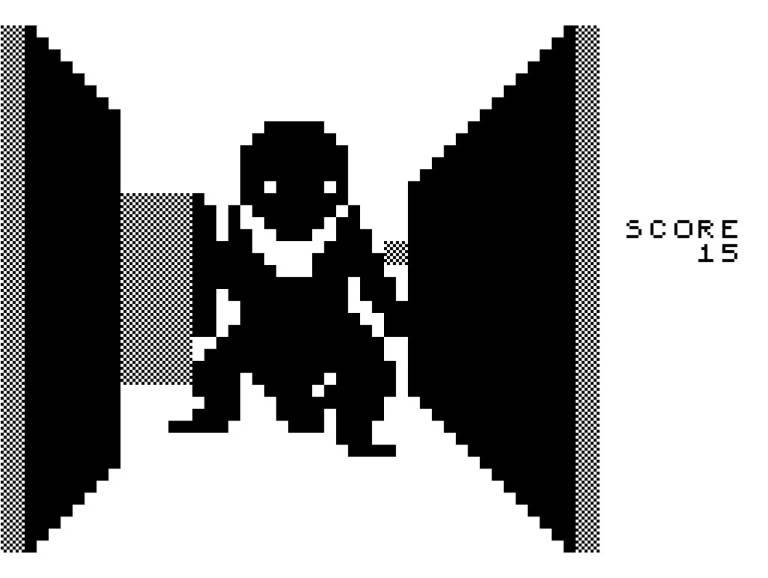


Рисунок 1.3 – “3D Monster Maze”, первая 3D игра

1.2 Постановка задачи

В рамках данного курсового проекта планируется разработка игрового программного средства «3D Shooter», адаптированного под запуск на современных операционных системах Windows.

В процессе реализации будут разработаны алгоритмы отрисовки карты и объектов на ней в трёхмерном представлении, а также игровой логики, а именно взаимодействие игрока с объектами карты и с самой картой.

В ходе разработки данного программного средства планируется решить следующие задачи:

* выбор уровней карты;
* отрисовка двумерной карты в трёхмерном представлении;
* отрисовка противников и других объектов;
* взаимодействие с другими объектами на карте;
* стрельба по противникам;
* поведение противников;
* меню с возможностью выбора уровней и настроек;
* визуализация таблицы рекордов.

Для разработки программного средства будет использоваться язык программирования C++ и функции WinAPI.

2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

2.1 Структура программы

В данном приложении использовались следующие классы и файлы:

–Enemy – класс, описывающий поведение и состояние базовых противников;

– EnemySoldier – класс, который расширяет класс Enemy;

– Bullet – класс, отвечающий за реализацию снарядов;

– EventHandler – класс, позволяющий отследить нажатие и момент поднятия клавиши;

– Menu – класс, содержащий модель главного меню;

– Pause – класс, содержащий модель меню паузы;

– Levels – класс, содержащий модель меню выбора уровней и саму информацию об уровнях;

– Sound – класс, обеспечивающий работу со звуком;

– EDS – класс, использующийся для подписи таблицы рекаордов;

– Engine.h – файл, содержащий основные константы и структуру, описывающую состояния игрока;

– Source.cpp – основной файл, в котором осуществляется прорисовка и всё остальное.

2.2 Интерфейс программного средства

Внешний вид и удобность в использовании являются одними из главных критериев качества программного средства. Поэтому взаимодействие приложения с пользователем необходимо организовать максимально интуитивно и просто.

Для реализации удобного меню была выбрана парадигма MVC. В данной реализации модель представляется с помощью отдельного класса, а контроль и отображение осуществляются с помощью функций в файле Source.cpp, при этом контроллер изменяет только состояние модели, а отображение обращается только к модели

Класс главного меню Menu имеет поле itemPointer перечислимого типа ItemPointer, который представляет собой указатель на конкретный пункт меню. Все значения перечислимого типа соответствуют пунктам меню. Далее представлено описание этого типа:

enum ItemPointer {ipStart, ipLevels, ipRecords, ipQuit, ipAmount};

Соответствующие пункты меню:

– «Start»;

– «Levels»;

– «Records»;

– «Quit».

Последнее значение перенумерованного типа ItemPointer используется для обозначения количества пунктов меню.

Отображение данного меню представлено на рисунке 2.1.



Рисунок 2.1 – Главное меню

С помощью класса EventHandler фиксируется поднятие клавиши “W”, “S” или пробела и вызывается функция Move класса Menu с соответствующим параметром. Данная функция изменяет указатель на пункт меню и возвращает значение перенумерованного типа GameState, которое определяет настоящее состояние игры. Описание типа GameState приведено ниже:

enum GameState {gsMainMenu, gsRun, gsPause, gsQuit, gsChooseLevel,

gsLoadLevel, gsShowRecords, gsGameOver};

При выборе пункта “start” выбирается карта, на которой враги генерируются случайно, при этом при убийстве одного врага генерируются дополнительные двое.

При выборе пункта “levels” переменная gameState, отвечающая за состояние игры, меняет своё значение на gsChooseLevel, и, соответственно далее идёт работа с классом Levels, который устроен аналогично классу Menu. Однако, данный класс также содержит в себе описание уровней. Каждый уровень описывается следующей структурой:

typedef struct \_Level

{

int ObjectsNum;//кол-во объектов на уровне

Object\* objects;//казатель на массив объектов

int MapWidth, MapHeight;//высота и ширина карты

std::string map;//сама карта

float PlayerX, PlayerY;//координаты, в которых появляется игрок при загрузке карты

} Level;

Object является структурой, которая содержит информацию о том, к какому типу принадлежит данный объект и его координаты на данной карте. Тип, который указывает, к какому виду принадлежит объект, является перенумерованным:

enum ObjectType {otEnemy, otEnemySoldier};

typedef struct \_Object

{

ObjectType type;

float x, y;

} Object;

При выборе пункта “records” аналогичным образом осуществляется показ таблицы рекордов. Более подробно работа с таблицей рекордов будет рассмотрена в пункте 2.6.

При выборе пункта “Quit” переменная состояния принимает значение gsQuit, для которого не предусмотрено никаких действий, кроме выхода из главного цикла игры, в ходе чего программа завершает работу.

Также стоит упомянуть о состояниях паузы и завершении игры.

При значении gsPause переменной состояния повляется меню, из которого можно выйти в главное, продолжить играть, или выйти из игры вовсе.

В состоянии gsGameOver показывается результат игрока за последнюю игру, который включает в себя время игры и количество уничтоженных врагов. Из данного меню можно выйти в главное меню или из самой программы.

2.3 Работа со звуком

Работу со звуком обеспечивает класс Sound. При этом при запуске программы происходит загрузка аудиофайлов в динамическую память для более быстрого и удобного доступа к ним в дальнейшем. Для удобства работы с данными аудиофайлов была определена следующая структура:

typedef struct \_SoundData

{

char \*sndBuf;//буфер, куда записываются данные аудифайла

int size;//размер буфера

bool isLoaded;//флаг, который показывает загружен ли аудифайл в динамичсекую память

} SoundData;

Также для описания того, какой именно звук содержит данная структура был создан перенумерованный тип SoundName.

enum SoundName {sndShoot, sndGameOver, sndMain, sndAmount};

При создании объекта данного класса в конструкторе Sound происходит загрузка аудиофайлов в динамическую память, при этом, если загрузка не удалась, флаг isLoaded принимает значение false, иначе - true. Далее представлен код процедуры, реализующей эту загрузку:

Sound::Sound()

{

int size;

for (int i = 0; i < sndAmount; i++)

{

std::ifstream f;

switch (i)

{

case sndShoot:

f.open(FileShoot, std::ios::binary);

break;

case sndGameOver:

f.open(FileGameOver, std::ios::binary);

break;

case sndMain:

f.open(FileMain, std::ios::binary);

break;

}

if (f.is\_open())

{

f.seekg(0, std::ios\_base::end);

size = f.tellg();

sounds[i].size = size;

sounds[i].sndBuf = new char[size];

f.seekg(0, std::ios\_base::beg);

f.read(sounds[i].sndBuf, size);

f.close();

sounds[i].isLoaded = true;

}

else

sounds[i].isLoaded = false;

}

}

Проигрывание загруженных аудиофайлов осуществляется в функции Play с помощью API функции PlaySound. Данная функция принимает в качестве параметра значение перенумерованного типа, которое определяет какой звук надо проиграть.

void Sound::Play(SoundName Name)

{

if (sounds[Name].isLoaded)

sndPlaySound((LPCSTR)sounds[Name].sndBuf, SND\_MEMORY |SND\_ASYNC);

}

2.4 Прорисовка карты и других объектов

Для прорисовки карты используется такой метод рендеринга, как ray casting (метод “бросания лучей”), при котором изображение строится на основе замеров пересечения лучей с визуализируемой поверхностью. Работа данного алгоритма представлена на рисунке 2.2.

Игровой мир построен на квадратной сетке из стен однородной высоты и ровных сплошных полов и потолков. Для отрисовки игрового уровня из области наблюдения «выпускается» луч, который проходит через каждый столбец символов на экране. Далее алгоритм проверяет, не пересёк ли луч стену, и если пересёк, то в соответствии с этим выбирает и масштабирует текстуру стены согласно тому, где на уровне луч «столкнулся» со стеной и насколько далеко он распространился до этого.

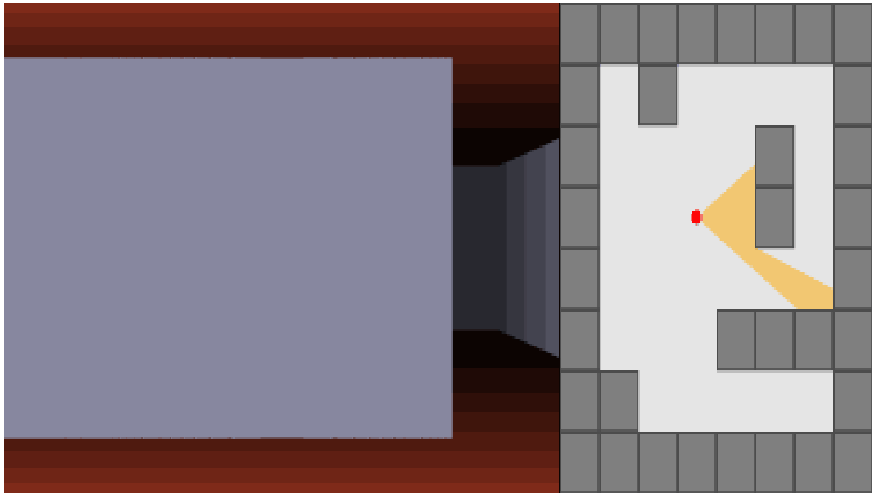


Рисунок 2.2 – Работа алгоритма ray-casting

Для реализации данного алгоритма в структуре Player есть поле fAngle, которое обозначает угол камеры с положительным направлением оси 0X и принимает значения от 0 до 2π. Данное значение изменяется при нажатии клавиш “A” или “W”, т.е. при нажатии данных клавиш происходит поворот камеры вправо или влево. Назначение некоторых ключевых полей структуры Player изображено на рисунке ниже.

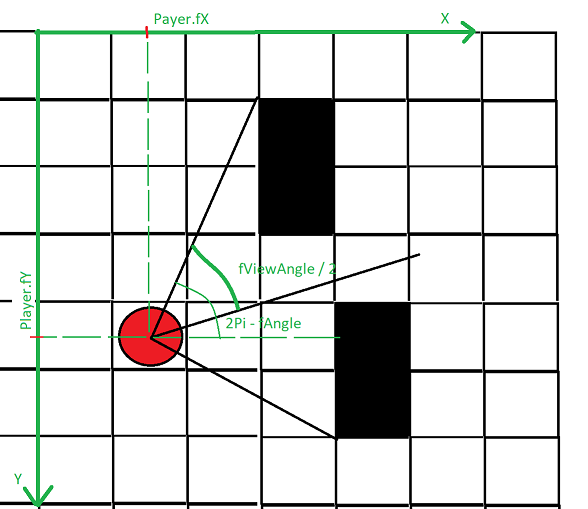


Рисунок 2.3 – Назначение полей структуры Player.

Далее в цикле для всех столбцов матрицы символов “выпускаются” лучи и производятся необходимые измерения. Ниже приведена часть кода, реализующая основную часть алгоритма “бросания” лучей:

for (x = 0; x < ScreenWidth; x++)

{

/\*вычисляем угол луча относительно основной системы координат\*/

float fRayAngle = (player.fAngle - fViewAngle / 2.0f)

+ ((float)x / ScreenWidth) \* fViewAngle;

float fWallDistance = 0;

bool isWall = false;

while (!isWall && fWallDistance < player.fViewRad)

{

fWallDistance += 0.1f;//"наращиваем" луч

/\*вычисляем координаты конца отрезка луча \*/

int TestX = (int)((player.fX + cosf(fRayAngle) \* fWallDistance) + MapWidth) % MapWidth;

int TestY = (int)((player.fY + sinf(fRayAngle) \* fWallDistance) + MapHeigth) % MapHeigth;

/\* если луч вышел за рамки карты, останавливаем вычисление \*/

if (TestX < 0 || TestX >= MapWidth || TestY < 0 || TestY >= MapHeigth)

{

isWall = true;

fWallDistance = fDepth;

}

else

{

/\* фиксируем столкновение луча со стеной и останавливаем вычисления\*/

if (map[TestY \* MapWidth + TestX] == '#')

isWall = true;

}

}

DepthBuf[x] = fWallDistance;//заносим значение расстояния до стены в буфер глубины

/\*вычисляются координаты верхней границы стены в символах\*/

int Ceiling = (float)(ScreenHeight / 2.0) - ScreenHeight / ((float)fWallDistance);

/\*вычисляются координтаты нижней границы стены в символах \*/

int Floor = ScreenHeight - Ceiling;

…

}

Для того, чтобы изображение выглядело более трёхмерным, для отрисовки стен используются разные текстуры, в зависимости от расстояния до них:

if (fWallDistance <= 5)

WallShade = 178;

else

if (fWallDistance <= 10)

WallShade = 177;

else

WallShade = 176;

Отрисовка противников и других объектов происходит немного иначе. Сначала проверяется, попадает ли противник в сектор обзора игрока и, если попадает, вычисляем расстояние до него и выполняем соответствующую отрисовку. Далее приведён код, реализующий данную идею:

/\*вычисляем вектор от игрока до противника\*/

float alpha = atan2(enemy->getY() - player.fY, enemy->getX() - player.fX);

/\*проверяем, попадает ли противник в сектор обзора\*/

if ((alpha > CheckLow && alpha < CheckHigh) || isVisual)

{

int Ceiling = ScreenHeight / 2.0f - ScreenHeight / (fEnemyDistance);

int Floor = ScreenHeight - Ceiling;

int Height = Floor - Ceiling;

…

}

Результат такой отрисовки представлен на рисунке ниже:

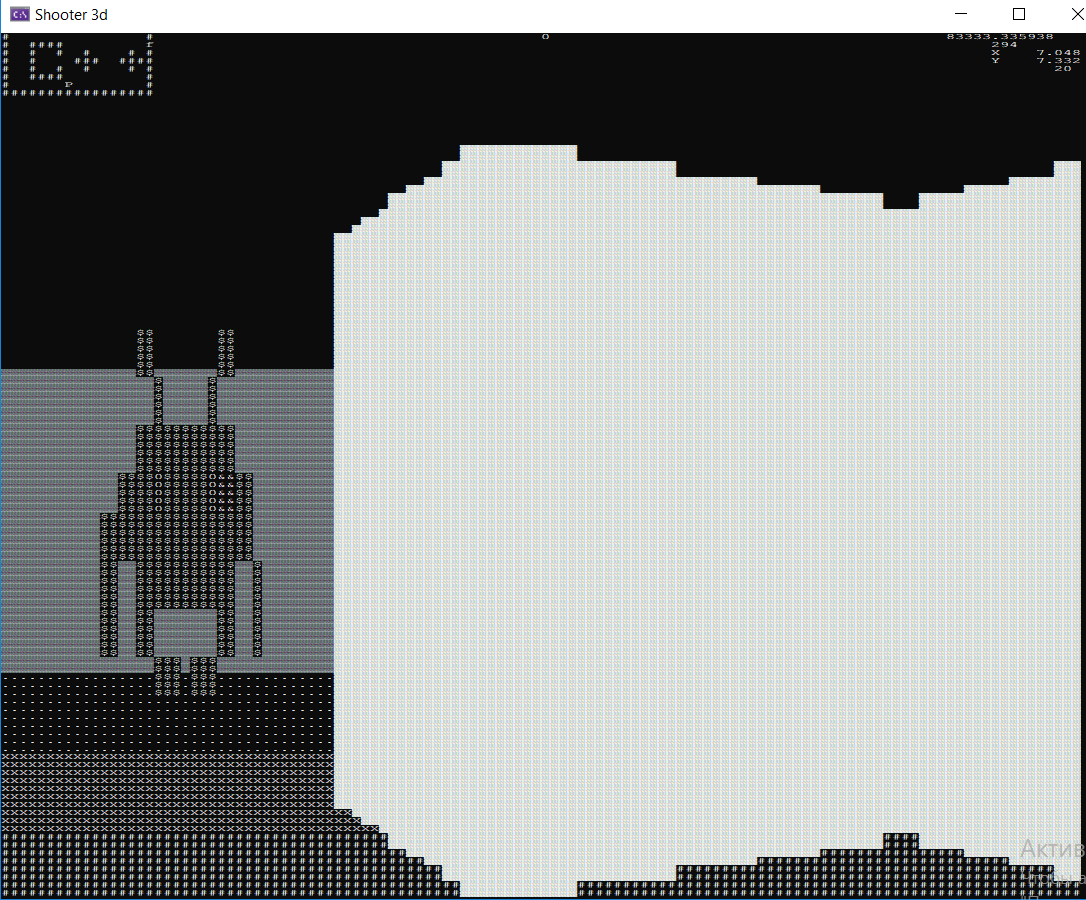


Рисунок 2.4 – Результат работы алгоритмов отрисовки карты и противников

2.5 Взаимодействие с объектами и игровая логика

Все игровые события и отрисовка происходят внутри главного игрового цикла, выход из которого осуществляется только в том случае, если переменная состояния приняла значение gsQuit.

Основной игровой цикл выполняется при значении переменной состояния gsRun. В данном цикле происходит обработка нажатий клавиш, вызываются функции, которые инициируют поведение противников, полёт пуль, отрисовка карты и интерфейса, на котором изображено количество кадров в секунду, координаты игрока, угол поворота камеры и запас здоровья. В конце цикла происходит проверка запаса здоровья игрока, и если оно меньше либо равно нулю, то переменная gameState принимает значение gsGameOver и происходит отрисовка соответствующего меню и запись в таблицу рекордов.

Для хранения объектов карты используется класс-контейнер list, представляющий двусвязный список[2]. Класс-контейнер list был выбран по той причине, что в ходе игры будет часто происходить удаление объектов, а данная структура позволяет выполнить данную операцию за O(1).

В файле Source объявлены список объектов класса Bullet и список указателей на объекты класса Enemy. Для противников использовался список указателей на объекты, а не список самих объектов для того, чтобы можно было работать с экземплярами классов, производных от класса Enemy. Наиболее важным методом класса Enemy и его производных является метод void Move(float time), который обеспечивает поведение противников. Т.к. интерфейс данного метода не отличается от интерфейса в производных классах и данный метод надо будет вызывать для экземпляров разных классов, то удобно объявить метод Move как виртуальную функцию[3].

Как объекты класса Bullet, так и объекты Enemy и его производных имеют поле isDestroy, которое определяет, нужно ли удалять объект из списка или нет. В функциях UpdateBullet проверяется. Попала ли пуля в стену или в противника, и, если попала, то поле isDestroy принимает значение true. Функция UpdateEnemyes() проверяет, попала ли пуля в противника и также выставляет значение поля isDestroy объекта класса Enemy и его производных в зависимости от этого. Данная функция представлена ниже:

void UpdateEnemyes()

{

enemyes.sort([](const Enemy\* en1, const Enemy\* en2) {return \*en1 < \*en2; });

//SortEnemyes();

float size;

for (Enemy\* pEnemy : enemyes)

{

for (Bullet &bullet : bullets)

{

float size = pEnemy->getSize();

if (bullet.getX() > pEnemy->getX() - size

&& bullet.getX() < pEnemy->getX() + size

&& bullet.getY() < pEnemy->getY() + size

&& bullet.getY() > pEnemy->getY() - size)

{

KillsCounter++;

pEnemy->setDestroy(true);

bullet.setDestroy(true);

}

}

}

enemyes.remove\_if([](Enemy\* en) {

if (en->getDestroy())

{

delete(en);

return true;

}

return false; });

}

Стоит также отметить механизм обработки событий. Для получения информации о том, нажата ли клавиша в данный момент, достаточно вызвать WinAPI функцию GetAsyncKeyState, передав в качестве параметра виртуальный код анализируемой клавиши. Данная функция возвращает переменную типа SHORT, старший бит у которой установлен в единицу, если данная клавиша была нажата после предыдущего вызова данной функции. Иногда важно учитывать, что данная функция достаёт событие нажатия клавиши из очереди сообщений, поэтому, если в другой программе вызывается данная функция, то информация о состоянии клавиши может быть некорректной[4].

Однако в WinAPI нет подобной специальной функции для определения события поднятия клавиши, что необходимо для навигации пользователя в меню. Для решения этой проблемы был создан класс для обработки событий EventHandler.

Для работы данного класса была определена структура KeyState, состоящая из трёх полей: Key – содержит код виртуальной клавиши, isDown – определяет нажата ли клавиша isUp – была ли клавиши поднята пере последним вызовом метода Update, который будет также описан далее.

Класс EventHandler содержит массив из 256 структур KeyState (по структуре на каждую виртуальную клавишу).

Наиболее важным методом данного класса является метод Update, который заполняет данный массив структур в соответствии с состоянием клавиатуры на момент вызова данного метода. Код данного метода представлен ниже:

void EventHandler::Update()

{

unsigned char i;

for (i = 0; i < 255; i++)

if (GetAsyncKeyState(KeysState[i].Key))

{

KeysState[i].isKeyDown = true;

KeysState[i].isKeyUp = false;

}

else

{

if (KeysState[i].isKeyDown)

KeysState[i].isKeyUp = true;

else

KeysState[i].isKeyUp = false;

KeysState[i].isKeyDown = false;

}

}

Для определения того, была ли клавиши поднята или нажата на момент последнего вызова метода Update используются геттеры isKeyUp и isKeyDown соответственно. Данные функции принимают в качестве аргумента виртуальный код клавиши, которую надо анализировать.

2.6 Работа с таблицей рекордов

Работа с таблицей рекордов реализована с помощью класса Records.

Данный класс содержит структуру Record, которая хранит в себе результат игрока для определённого уровня. Как было указано ранее, результат состоит из двух элементов: времени и количества убитых врагов. Следовательно, структура имеет следующий вид:

typedef struct \_Record

{

int kills;

int time;

} Record;

Сами рекорды хранятся в массиве values. Также определено значение результата по умолчанию, на случай если рекорд для данного уровня ещё не поставлен.

Также для проверки подлинности таблицы рекордов при загрузке их из файла используется цифровая подпись[5], работу с которой обеспечивает класс EDS. В данном классе реализована хеш-функция, которая в дальнейшем в качестве своих аргументов будет принимать указатель на массив рекордов и размер этого массива в байтах. Таким образом, в общем случае для разных значений рекордов значение хеш-функции будет различно, однако не стоит забывать про вероятность коллизии. Для быстроты вычислений используется довольно простая хеш-функция, код которой приведён ниже:

#define sqr(a) a \* a

int EDS::GetHash(char\* data, int size)

{

int h0 = SecretStartValue;

int h;

for (int i = 0; i < size; i++)

{

h = (sqr(h0 + (unsigned char) data[i])) % R;

h0 = h;

}

return h0;

}

Сама цифровая подпись будет представлять значение данной хеш-функции. Также для определения факта наличия подписи непосредственно перед самой подписью будет ставиться заранее определённый непечатный символ.

При запуске программы происходит попытка чтения рекордов из файла. При этом проверяется наличие специального непечатного символа, который определяет, существует ли цифровая подпись. Если файл успешно прочитан и цифровая подпись найдена, то на основании прочитанных рекордов вычисляется значение хеш-функции, которое сравнивается с прочитанным из файла значением и, если они равны, подпись признаётся действительной. Если хотя бы один из этих этапов прошёл некорректно (не удалось обнаружить файл, нет цифровой подписи и т.д.), то рекорды заполняются значением по умолчанию. Далtе приведён код, реализующий данные действия:

void LoadMyRecords()

{

bool isCorrect = true;

char temp;

int size;

EDS eds;

Record tempRec;

Record tempRecs[4];

std::ifstream fin(RecordsFileName);

if (fin.is\_open())

{

for (int i = 0; i < 4; i++)

{

tempRec = records.GetRecord(0);

if (fin >> tempRec.kills && fin >> tempRec.time)

records.SetRecord(i, tempRec);

else

isCorrect = false;

}

if (!(fin >> temp) && isCorrect)

isCorrect = false;

if (temp == EDSSign && isCorrect)

{

if (isCorrect && !(fin >> Siganture))

isCorrect = false;

for (int i = 0; i < 4; i++)

{

tempRecs[i] = records.GetRecord(i);

}

if (isCorrect && eds.GetHash((char\*)tempRecs, 4 \* sizeof(Record)) != Siganture)

isCorrect = false;

}

else

{

isCorrect = false;

}

}

else

{

isCorrect = false;

}

fin.close();

if (!isCorrect)

{

records.SetDefaultRecords();

}

}

По завершению уровня или смерти игрока сравниваются результаты игрока на данном уровне с предыдущими результатами на данном и, если новый результат оказался лучше, то происходит запись в таблицу рекордов. Из двух результатов лучшим оказывается тот, у которого больше значение убитых противников, или, если данные значения равны, тот, у которого меньше значение времени. Код, осуществляющий сравнение:

Record tempRec = records.GetRecord(levels.GetLevelNum());

if (KillsCounter > tempRec.kills ||

(KillsCounter == tempRec.kills && TotalTime / 1000 < tempRec.time))

{

tempRec.kills = KillsCounter;

tempRec.time = TotalTime / 1000;

records.SetRecord(levels.GetLevelNum(), tempRec);

}

SaveRecords();

При записи рекордов в файл также осуществляется вызов метода GetHash объекта класса EDS, который генерирует значение подписи в зависимости от новых результатов. Блок-схема функции, выполняющей сохранение, приведена ниже:



Рисунок 2.5 – Блок-схема функции SaveRecords

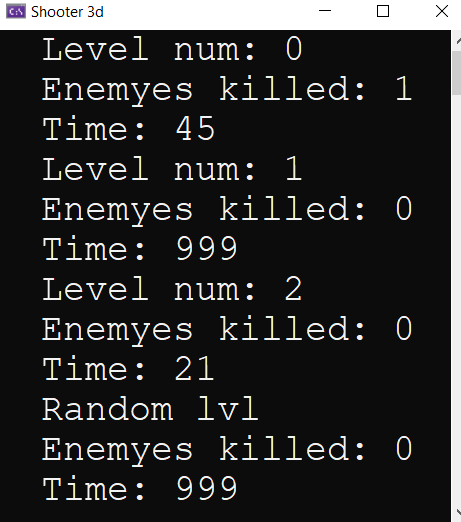


Рисунок 2.6 – Таблица рекордов

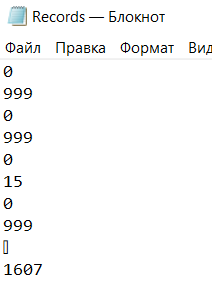


Рисунок 2.7 – Подписанный файл с таблицей рекордов

Таким образом, файл рекордов имеет минимальную защиту от подделки рекордов.

3 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Тестирование программного обеспечения является неотъемлемой частью разработки любого проекта, которое напрямую влияет на его качество. В ходе тестирования проверяется соответствие между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов.

В ходе тестирования приложения были выявлены следующие недостатки данного программного средства.

Была обнаружена проблема при прорисовке противников, стоящих друг за другом. А именно, противник, который находиться дальше от игрока, может прорисовывается поверх противника, стоящего ближе к игроку. Для устранения данной проблемы был создан так называемый буфер глубины, хранящий расстояние до ближайшего объекта или стены для каждого столбца матрицы символов screen. Использование данного буфера также обеспечивает то, что противники и другие объекты не будут прорисовываться через стену. Ниже приведен код, описывающий принцип работы буфера глубины.

DepthBuf[x] = fWallDistance;

char Shade = enemy->texture[textureY][textureX];

if (Shade != EMPTY && DepthBuf[x] > fEnemyDistance)

{

screen[y \* ScreenWidth + x] = Shade;

DepthBuf[x] = fEnemyDistance;

}

Также были проблемы, связанные с удалением противников. Как было указано в разделе 2.5, для обеспечения функционирования врагов был создан список указателей на объекты классов, базовым для которых является Enemy. И при удалении из списка врагов, освобождалась память, которую занимает только указатель на объект, но не сам объект. Ниже приведён код, решающий данную проблему:

enemyes.remove\_if([](Enemy\* en) {

if (en->getDestroy())

{

delete(en);

return true;

}

return false; });

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

4.1 Правила игры

` Суть данной игры заключается в том, чтобы за наименьшее время уничтожить как можно больше противников. Игра заканчивается в том случае, если игрока убивают или игрок доходит до специального места, которое на карте обозначено символом ‘f’ и нажимает клавишу ‘f’. И в том, и в другом случае появляется меню GameOver, на котором изображены результаты.

4.2 Управление

Во время самой игры перемещение осуществляется по нажатию клавиш ‘W’ для перемещения вперёд, ‘S’ – для перемещения назад. Поворот камеры влево и вправо по нажатию клавиш ‘A’ и ‘D’ соответственно. При нажатии пробела осуществляется выстрел. Также во время самой игры можно вызвать меню паузы, откуда потом можно выйти в главное меню, продолжить играть или вовсе закрыть программу.

Взаимодействие с пунктами меню осуществляется по нажатию клавиш ‘W’, ‘S’, пробел и esc. При нажатии клавиши ‘W’ указатель на пункт меню перемещается вверх, при нажатии ‘S’ – вниз. Выбор пункта меню осуществляется по нажатию пробела. Если меню не главное, то при нажатии клавиши esc происходит выход в главное меню.

4.3 Противники

В игре 2 типа противников, представленные классами Enemy и EnemySoldier.

Противник, являющийся объектом класса Enemy, не представляет реальной угрозы для игрока. Его поведение организовано таким образом, что при появлении игрока в определённом радиусе, данный противник начинает движение в сторону игрока, при этом скорость противиника ниже скорости игрока. Не смотря на то, что данный противник не представляет угрозы, за его убийство также даются очки, которые влияют на конечный результат. Данный противник представлен на рисунке 4.1.

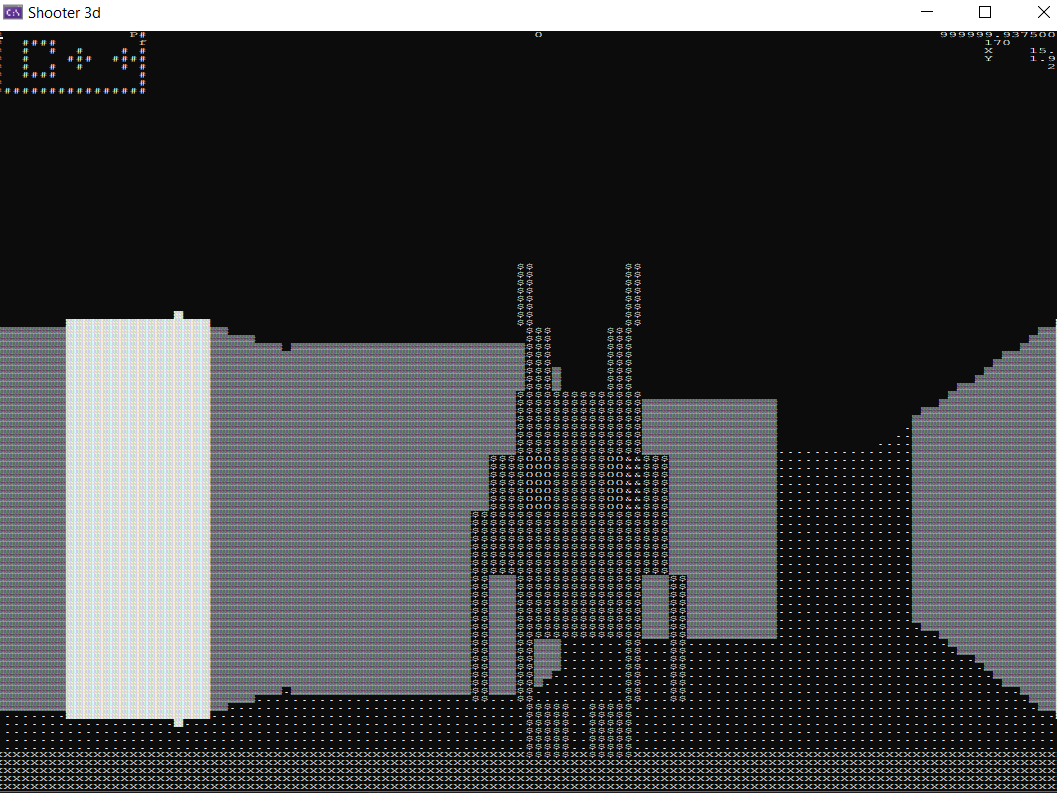


Рисунок 4.1 – Противник класса Enemy

Второй противник представлен классом EnemySoldier. Данный противник ведёт себя таким образом, что при определённом расстоянии о игрока противник начинает стрелять в направлении игрока. При этом противник класса EnemySoldier не меняет своего местоположения. Внешний вид данного противника продемонстрирован на рисунке 4.2.

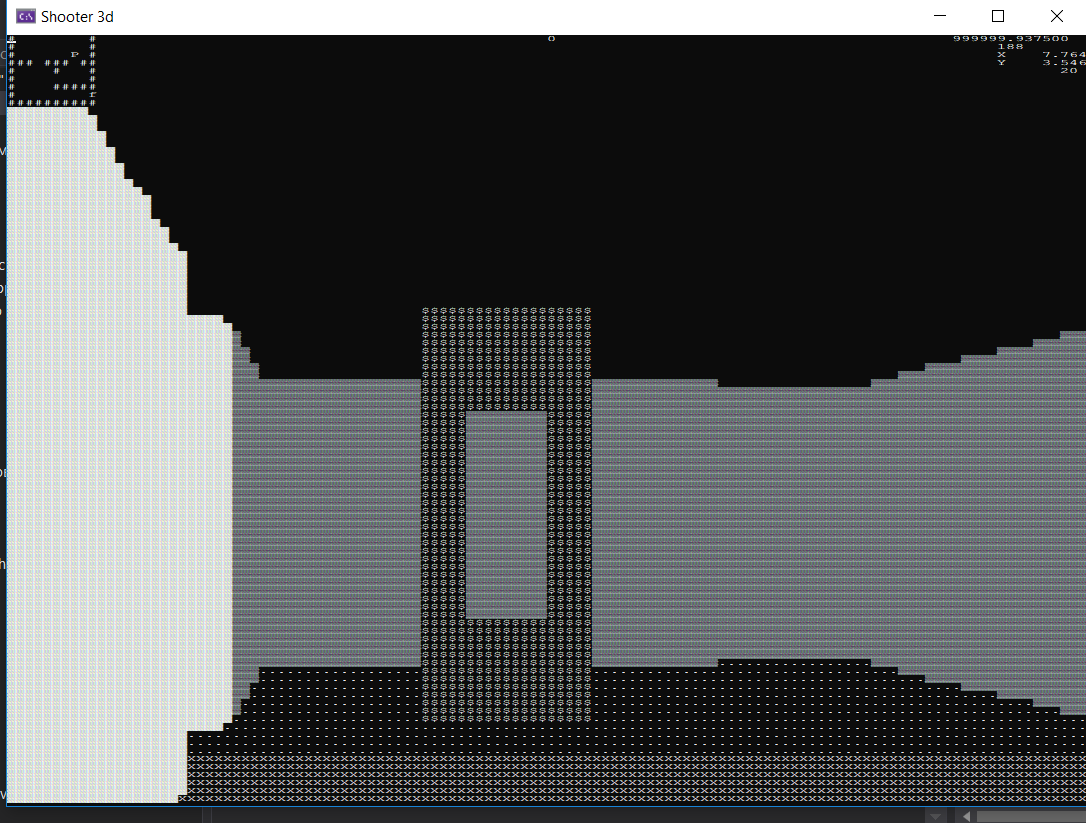


Рисунок 4.2 – Противник класса EnemySodier

В дальнейшем планируется добавить ещё некоторые типы противников, расширяющих класс Enemy.

ЗАКЛЮЧЕНИЕ

На сегодняшний день с развитием компьютерной техники интерес к компьютерным играм увеличивается всё больше и больше. Большинство популярных современных игр являются трёхмерными. Для реализации трёхмерной графики используются различные методы, среди которых растеризация, ray-casting, ray-tracing и трассировка пути. В данном ПО был реализован один из выше перечисленных методов.

В рамках данного курсового проекта было разработано игровое программное средство «3D Shooter». Согласно поставленным задачам, в данном приложении были реализованы следующие функции:

* выбор уровней карты;
* отрисовка двумерной карты в трёхмерном представлении;
* отрисовка противников и других объектов;
* взаимодействие с другими объектами на карте;
* стрельба по противникам;
* поведение противников;
* меню с возможностью выбора уровней и настроек;
* визуализация таблицы рекордов.

Для успешной реализации вышеперечисленных функций потребовалось изучить объектно-ориентированную парадигму и возможности её применения в С++, концепцию MVC(Model-View-Controller)

Существуют множество вариантов для улучшения данного программного обеспечения, например, добавление новых противников, видов оружия и различных бонусов, дающие некоторые преимущества. Ещё одним вариантом развития является использование в данном проекте различных графических библиотек, что увеличит качество визуализации, однако может снизить производительность.

Данное программное обеспечение позволяет развить пространственное мышление, реакцию и просто провести время за игрой. Также данный проект показывает то, что для создания трёхмерной (в данном случае, псевдотрёхмерной) графики не обязательно наличие каких-либо специальных графических библиотек, а важна сама идея того, что в них реализовано. Таким образом, создание трёхмерной игры возможно даже в командной строке, что и демонстрирует данный проект.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Компьютерная графика. Динамика, реалистические изображения/Шикин Е. В., Боресков А. В . – М.: ДИАЛОГ МИФИ, 1996. – 288 с.

[2] Алгоритмы. Теория и практическое применение/Род Стивенс – Москва: Издательство «Э», 2016. – 544 с.

[3] Самоучитель C++/Герберт Шилдт – Санкт-Петербург: Издательство «БХВ-Петербург». 2003. -689 с

[4] Библиотека MSDN[Электронный ресурс]. – Электронные данные. – режим доступа: <https://docs.microsoft.com/ru-ru/>

.[5] Криптография и охрана коммерческой информации. Методическое пособие по выполнению лабораторных работ для студентов специальностей 1-40 01 02 – 02 и 1-26 02 03 дневной и заочной форм обучения/ C. В. Ярмолик, В. Н. Ярмолик. – Минск: БГУИР, 2010. – 35 с.

ПРИЛОЖЕНИЕ А

Исходный код программы

Основной файл Source.cpp

#include "Engine.h"

#include "Enemy.h"

#include "EnemySoldier.h"

#include "Bullet.h"

#include "EventHadler.h"

#include "Menu.h"

#include "Levels.h"

#include "Pause.h"

#include "GameOver.h"

#include "Records.h"

#include "EDS.h"

#include "Sound.h"

#define \_CRT\_SECURE\_NO\_WARNINGS

//TODO: add to Engine.h

int MapWidth = 16;

int MapHeigth = 16;

float DepthBuf[ScreenWidth];

Player player;

Sound \*sound;

int KillsCounter;

float TotalTime;

int Siganture;

Records records;

CONSOLE\_FONT\_INFOEX font;

HANDLE hConsole;

char\* sndBuf;

const char\* SoundFileName = "gun-gunshot-01.wav";

const char\* RecordsFileName = "Records.txt";

GameState gameState;

//---------------------------

using namespace std;

string map = "";

list<Enemy\*> enemyes;

list<Bullet> bullets;

DWORD note;

void CreateMap()

{

map += "################";

map += "#...#..........#";

map += "#...#..........#";

map += "#...#..........#";

map += "#..............#";

map += "#..............#";

map += "#.....###......#";

map += "#..............#";

map += "#..............#";

map += "#.......#......#";

map += "#..............#";

map += "#.........#....#";

map += "#.........#....#";

map += "#.........######";

map += "#..............f";

map += "################";

}

void DrawMap(char\* screen, int MapHeight, int MapWidth, int ScreenWidth)

{

int y, x, i;

i = 0;

for (y = 0; y < MapHeight; y++)

{

for (x = 0; x < MapWidth; x++)

{

screen[i] = map[y \* MapWidth + x];

i++;

}

i = y \* ScreenWidth;

}

screen[(int)player.fY \* ScreenWidth - ScreenWidth + (int)player.fX] = 'P';

}

bool isCollision(float x, float y)

{

bool isCollision = 0;

list<Enemy\*>::iterator p = enemyes.begin();

while ((p != enemyes.end()) && (!isCollision) && (enemyes.size() > 0))

{

float size = (\*p)->getSize();

if (x > (\*p)->getX() - size && x < (\*p)->getX() + size

&& y >(\*p)->getY() - size && y < (\*p)->getY() + size)

isCollision = true;

p++;

}

return isCollision;

}

bool IsWall(float x, float y)

{

return map[(int)y \* MapWidth + (int)x] == '#'

|| map[(int)y \* MapWidth + (int)x] == 'f';

}

void MovePlayer(float time)

{

if (GetAsyncKeyState('F'))

{

float testX = player.fX + 0.1f \* cosf(player.fAngle);

float testY = player.fY + 0.1f \* sinf(player.fAngle);

if (map[(int)testY \* MapWidth + (int)testX] == 'f')

{

gameState = gsGameOver;

}

}

if (GetAsyncKeyState(VK\_ESCAPE))

gameState = gsPause;

if (GetAsyncKeyState('A'))

player.fAngle -= player.fAngleSpeed \* time;

if (GetAsyncKeyState('D'))

player.fAngle += player.fAngleSpeed \* time;

if (GetAsyncKeyState('W'))

{

float testX = player.fX + player.fSpeed \* cosf(player.fAngle) \* time;

float testY = player.fY + player.fSpeed \* sinf(player.fAngle) \* time;

if (!IsWall(testX, testY) && !isCollision(testX, testY))

{

player.fX = testX;

player.fY = testY;

}

}

if (GetAsyncKeyState('S'))

{

float testX = player.fX - player.fSpeed \* cosf(player.fAngle) \* time;

float testY = player.fY - player.fSpeed \* sinf(player.fAngle) \* time;

if (!IsWall(testX, testY) && !isCollision(testX, testY))

{

player.fX = testX;

player.fY = testY;

}

}

if (GetAsyncKeyState(' '))

{

// Player.tmAfterShoot += time;

if (player.tmReload < player.tmAfterShoot)

{

sound->Play(sndShoot);

float b\_x = player.fX + ( player.fSize + 0.1f) \* cosf(player.fAngle);

float b\_y = player.fY + (player.fSize + 0.1f) \* sinf(player.fAngle);

Bullet bullet(b\_x, b\_y, cosf(player.fAngle), sinf(player.fAngle));

bullets.push\_back(bullet);

player.tmAfterShoot = 0;

}

}

player.tmAfterShoot += time;

}

void LoadMyRecords()

{

bool isCorrect = true;

char temp;

int size;

EDS eds;

Record tempRec;

Record tempRecs[4];

std::ifstream fin(RecordsFileName);

if (fin.is\_open())

{

for (int i = 0; i < 4; i++)

{

tempRec = records.GetRecord(0);

if (fin >> tempRec.kills && fin >> tempRec.time)

records.SetRecord(i, tempRec);

else

isCorrect = false;

}

if (!(fin >> temp) && isCorrect)

isCorrect = false;

if (temp == EDSSign && isCorrect)

{

if (isCorrect && !(fin >> Siganture))

isCorrect = false;

for (int i = 0; i < 4; i++)

{

tempRecs[i] = records.GetRecord(i);

}

if (isCorrect && eds.GetHash((char\*)tempRecs, 4 \* sizeof(Record)) != Siganture)

isCorrect = false;

}

else

{

isCorrect = false;

}

}

else

{

isCorrect = false;

}

fin.close();

if (!isCorrect)

{

records.SetDefaultRecords();

}

}

void SaveRecords()

{

FILE\* f;

if (fopen\_s(&f, RecordsFileName, "w"))

{

EDS eds;

Record tempRecs[4];

Record tempRec;

for (int i = 0; i < 4; i++)

{

tempRecs[i] = records.GetRecord(i);

}

Siganture = eds.GetHash((char\*)tempRecs, 4 \* sizeof(Record));

for (int i = 0; i < 4; i++)

{

tempRec = records.GetRecord(i);

fprintf\_s(f, "%d\n", tempRec.kills);

fprintf\_s(f, "%d\n", tempRec.time);

}

fprintf\_s(f, "%c\n", EDSSign);

fprintf\_s(f, "%d\n", Siganture);

fclose(f);

}

}

void LoadSound()

{

int size;

ifstream f(SoundFileName);

if (f.is\_open())

{

f.seekg(0, ios\_base::end);

size = f.tellg();

sndBuf = new char[size];

f.seekg(0, ios\_base::beg);

f.read(sndBuf, size);

f.close();

}

}

void PrintLine(char\* screen, int ScreenWidth, char \*strBuf, int left, int height)

{

int i, j = 0;

for (i = ScreenWidth - left; i < ScreenWidth && strBuf[j] != '\0'; i++, j++)

screen[i + height \* ScreenWidth] = strBuf[j];

}

void DrawInterface(char\* screen, float time)

{

if (!time)

time = 1;

time = time / CLOCKS\_PER\_SEC;

float FPS = 1000 / time;

char strBuf[20] = " ";

sprintf\_s(strBuf, "%f", FPS);

int i = 0, j = 0;

PrintLine(screen, ScreenWidth, strBuf, 15, 0);

int angle = player.fAngle \* 180 / PI;

j = 0;

\_itoa\_s(angle, strBuf, 10);

PrintLine(screen, ScreenWidth, strBuf, 10, 1);

sprintf\_s(strBuf, "%f", player.fX);

screen[ScreenWidth \* 2 + ScreenWidth - 10] = 'X';

PrintLine(screen, ScreenWidth, strBuf, 5, 2);

screen[ScreenWidth \* 3 + ScreenWidth - 10] = 'Y';

sprintf\_s(strBuf, "%f", player.fY);

PrintLine(screen, ScreenWidth, strBuf, 5, 3);

sprintf\_s(strBuf, "%d", player.HP);

PrintLine(screen, ScreenWidth, strBuf, 3, 4);

}

void DrawWalls(char\* screen)

{

int x;

for (x = 0; x < ScreenWidth; x++)

{

float fRayAngle = (player.fAngle - fViewAngle / 2.0f) + ((float)x / ScreenWidth) \* fViewAngle;

float fWallDistance = 0;

bool isWall = false;

int TestX;

int TestY;

while (!isWall)

{

fWallDistance += 0.1f;

TestX = (int)(player.fX + cosf(fRayAngle) \* fWallDistance);

TestY = (int)(player.fY + sinf(fRayAngle) \* fWallDistance);

if (TestX < 0 || TestX >= MapWidth || TestY < 0 || TestY >= MapHeigth)

{

isWall = true;

fWallDistance = fDepth;

}

else

{

if (IsWall((float)TestX, (float)TestY))

isWall = true;

}

}

DepthBuf[x] = fWallDistance;

int Ceiling = (float)(ScreenHeight / 2.0) - ScreenHeight / ((float)fWallDistance);

int Floor = ScreenHeight - Ceiling;

int y;

char WallShade;

if (fWallDistance <= 5)

WallShade = 178;

else

if (fWallDistance <= 10)

WallShade = 177;

else

WallShade = 176;

if (map[(int)TestY \* MapWidth + (int)TestX] == 'f')

WallShade = 'F';

for (y = 0; y < ScreenHeight; y++)

{

if (y <= Ceiling)

screen[y \* ScreenWidth + x] = ' ';

else if (y > Ceiling && y <= Floor && fWallDistance <= fViewRad)

screen[y \* ScreenWidth + x] = WallShade;

else

{

char FloorShade;

if (y >= ScreenHeight - ScreenHeight / (2 \* 3))

FloorShade = '#';

else

if (y >= ScreenHeight - ScreenHeight / 4)

FloorShade = 'X';

else

FloorShade = '.';

screen[y \* ScreenWidth + x] = FloorShade;

}

}

}

}

void DrawEnemy(char\* screen, Enemy\* enemy)

{

while (player.fAngle > 2 \* PI)

player.fAngle -= 2 \* PI;

while (player.fAngle < 0)

player.fAngle += 2 \* PI;

float MainAlpha = atan2(enemy->getY() - player.fY, enemy- >getX() - player.fX);

if (MainAlpha < 0)

MainAlpha += 2 \* PI;

bool isVisual = false;

float fEnemyDistance = sqrt(pow(enemy->getX() - player.fX, 2) +

pow(enemy->getY() - player.fY, 2));

float j;

float N = fEnemyDistance / (120.0f + 60.0f);//replace 60.0f

for (j = -enemy->getSize(); j < enemy->getSize(); j += N)

{

float alpha = atan2(j, fEnemyDistance) + MainAlpha;

float CheckLow = player.fAngle - fViewAngle / 2.0f;

float CheckHigh = player.fAngle + fViewAngle / 2.0f;

if (CheckLow < 0)

if (alpha < PI / 2.0f && alpha < CheckHigh)

isVisual = true;

else

if (alpha > CheckLow + 2 \* PI)

isVisual = true;

if ((alpha > CheckLow && alpha < CheckHigh) || isVisual)

{

screen[ScreenWidth / 2] = '0';

int Ceiling = ScreenHeight / 2.0f - ScreenHeight / (fEnemyDistance);

int Floor = ScreenHeight - Ceiling;

int Height = Floor - Ceiling;

int textureX = (j + enemy->getSize()) \* enemy- >getTextureWidth()

/ (2 \* enemy->getSize());

int y;

int x = (alpha - CheckLow) \* ScreenWidth / (CheckHigh - CheckLow);

for (y = Ceiling; y < Floor; y++)

{

if (!(y > ScreenHeight || y < 0))

{

int textureY = (y - Ceiling) \* enemy ->getTextureHeight() / Height;

char Shade = enemy->texture[textureY][textureX];

if (Shade != EMPTY && DepthBuf[x] > fEnemyDistance)

{

screen[y \* ScreenWidth + x] = Shade;

}

}

}

}

}

}

void DrawEnemyes(char\* screen)

{

for (Enemy\* pEnemy : enemyes)

{

DrawEnemy(screen, pEnemy);

}

}

void SortEnemyes()

{

Enemy\* temp = new Enemy(0.0f, 0.0f, 0.0f);

for (Enemy\* pEnemyI : enemyes)

{

for (Enemy\* pEnemyJ : enemyes)

if (\*pEnemyI > \* pEnemyJ)

{

\*temp = \*pEnemyI;

\*pEnemyI = \*pEnemyJ;

\*pEnemyJ = \*temp;

}

}

}

void UpdateEnemyes()

{

enemyes.sort([](const Enemy\* en1, const Enemy\* en2) {return \*en1 < \*en2; });

float size;

for (Enemy\* pEnemy : enemyes)

{

for (Bullet &bullet : bullets)

{

float size = pEnemy->getSize();

if (bullet.getX() > pEnemy->getX() - size

&& bullet.getX() < pEnemy->getX() + size

&& bullet.getY() < pEnemy->getY() + size

&& bullet.getY() > pEnemy->getY() - size)

{

KillsCounter++;

pEnemy->setDestroy(true);

bullet.setDestroy(true);

}

}

}

enemyes.remove\_if([](Enemy\* en) {

if (en->getDestroy())

{

delete(en);

return true;

}

return false; });

}

void DrawCircle(int x, int y, int Rad, char shade, char\* screen)

{

int i, width;

for (i = 0; i < Rad && i < (ScreenHeight / 2); i++)

{

width = (int)sqrt(Rad \* Rad - i \* i);

int j;

for (j = 0; j < width && j < (ScreenWidth / 2); j++)

{

if (j + x < ScreenWidth)

{

screen[(i + y) \* ScreenWidth + j + x] = shade;

screen[(-i + y) \* ScreenWidth + j + x] = shade;

}

if (x - j > 0)

{

screen[(i + y) \* ScreenWidth - j + x] = shade;

screen[(-i + y) \* ScreenWidth - j + x] = shade;

}

}

}

}

void UpdateBullets()

{

for (Bullet& bullet : bullets)

{

if (bullet.getX() > player.fX - player.fSize

&& bullet.getX() < player.fX + player.fSize

&& bullet.getY() > player.fY - player.fSize

&& bullet.getY() < player.fY + player.fSize)

{

if (!bullet.getDestroy())

{

bullet.setDestroy(true);

player.HP -= bullet.GetDamage();

}

}

}

bullets.remove\_if([](Bullet &bullet){return bullet.getDestroy();});

}

void DrawBullets(char\* screen)

{

float x, y;

for (Bullet bullet: bullets)

{

x = bullet.getX();

y = bullet.getY();

while (player.fAngle > 2 \* PI)

player.fAngle -= 2 \* PI;

while (player.fAngle < 0)

player.fAngle += 2 \* PI;

float alpha = atan2(y - player.fY, x - player.fX);

if (alpha < 0)

alpha += 2 \* PI;

bool isVisual = false;

float fBulletDistance = sqrt(pow(x - player.fX, 2) + pow(y - player.fY, 2));

float CheckLow = player.fAngle - fViewAngle / 2.0f;

float CheckHigh = player.fAngle + fViewAngle / 2.0f;

if (CheckLow < 0)

if (alpha < PI / 2.0f && alpha < CheckHigh)

isVisual = true;

else

if (alpha > CheckLow + 2 \* PI)

isVisual = true;

if ((alpha > CheckLow && alpha < CheckHigh) || isVisual)

{

int Ceiling = ScreenHeight / 2.0f - ScreenHeight / (fBulletDistance );

int Floor = ScreenHeight - Ceiling;

int Radius = (Floor - Ceiling) / 2;

int x = (alpha - CheckLow) \* ScreenWidth / (CheckHigh - CheckLow);

DrawCircle(x, ScreenHeight / 2, Radius, BULLET\_SHADE , screen);

}

}

}

void MoveEnemyes(float time)

{

for (Enemy\* pEnemy : enemyes)

pEnemy->Move(player.fX, player.fY, time, map);

}

void MoveBullets(float time)

{

for (Bullet &bullet : bullets)

bullet.Move(map, time, enemyes);

}

void DrawRect(char \*screen, int ScreenWidth, int x,

int y, int Width, int Height, char Fill)

{

int i, j;

for (i = 0; i < Height; i++)

for (j = 0; j < Width; j++)

screen[(y + i) \* ScreenWidth + j + x] = Fill;

}

void DrawMainMenu(char \*MainMenuScreen, Menu &menu)

{

SMALL\_RECT rect = { 0, 0, MenuWidth, MenuHeight };

SetConsoleWindowInfo(hConsole, true, &rect);

DWORD dwBytes = 0;

memset(MainMenuScreen, ' ', MenuWidth \* MenuHeight);

DrawRect(MainMenuScreen, MenuWidth, 2, 2 \* (int)menu.GetIP(), 9, 3, '#');

char strBuf[20] = "start";

PrintLine(MainMenuScreen, MenuWidth, strBuf, 9, 1);

sprintf\_s(strBuf, "%s", "levels");

PrintLine(MainMenuScreen, MenuWidth, strBuf, 9, 3);

sprintf\_s(strBuf, "%s", "records");

PrintLine(MainMenuScreen, MenuWidth, strBuf, 9, 5);

sprintf\_s(strBuf, "%s", "Quit");

PrintLine(MainMenuScreen, MenuWidth, strBuf, 9, 7);

WriteConsoleOutputCharacter(hConsole, MainMenuScreen, MenuWidth \* MenuHeight, { 0, 0 }, &dwBytes);

}

void DrawLevelsMenu(char\* screen, Levels &levels)

{

DWORD dwBytes = 0;

memset(screen, ' ', LvlMenuWidth \* LvlMenuHeight);

DrawRect(screen, LvlMenuWidth, 2, 2 \* levels.GetLevelNum(), 9, 3, '#');

int i;

char strBuf[20] = "Level 1";

for (i = 0; i < levels.GetMaxLevel(); i++)

{

PrintLine(screen, LvlMenuWidth, strBuf, 9, i \* 2 + 1);

sprintf\_s(strBuf, "Level %d", i + 2);

}

WriteConsoleOutputCharacter(hConsole, screen,

MenuWidth \* MenuHeight, { 0, 0 }, &dwBytes);

}

void LoadLevel(Levels& levels)

{

int LevelNum = levels.GetLevelNum();

Level level = levels.GetLevel();

int i;

float x, y;

for (i = 0; i < level.ObjectsNum; i++)

{

x = level.objects[i].x;

y = level.objects[i].y;

switch (level.objects[i].type)

{

case otEnemy:

enemyes.push\_back(new Enemy(x, y, 0.3f));

break;

case otEnemySoldier:

enemyes.push\_back(new EnemySoldier(x, y, 0.3f));

break;

}

}

map = level.map;

MapWidth = level.MapWidth;

MapHeigth = level.MapHeight;

player.fX = level.PlayerX;

player.fY = level.PlayerY;

player.HP = 20;

TotalTime = 0.0f;

KillsCounter = 0;

sound->Play(sndMain);

}

void DrawPauseMenu(char \*screen,Pause &pause)

{

SMALL\_RECT rect = { 0, 0, PauseWidth, PauseHeight};

SetConsoleWindowInfo(hConsole, true, &rect);

DWORD dwBytes = 0;

memset(screen, ' ', MenuWidth \* MenuHeight);

DrawRect(screen, MenuWidth, 2, 2 \* (int)pause.GetIPP(), 10, 3, '#');

char strBuf[20] = "Continue";

PrintLine(screen, MenuWidth, strBuf, 9, 1);

sprintf\_s(strBuf, "%s", "Main menu");

PrintLine(screen, MenuWidth, strBuf, 9, 3);

sprintf\_s(strBuf, "%s", "Quit");

PrintLine(screen, MenuWidth, strBuf, 9, 5);

WriteConsoleOutputCharacter(hConsole, screen, MenuWidth \* MenuHeight, { 0, 0 }, &dwBytes);

}

void DrawGameOver(char\* screen, GameOver& gameOver)

{

SMALL\_RECT rect = { 0, 0, GameOverWidth, GameOverHeight};

SetConsoleWindowInfo(hConsole, true, &rect);

DWORD dwBytes = 0;

memset(screen, ' ', GameOverHeight \* GameOverWidth);

DrawRect(screen, GameOverWidth, 2, 2 \* (int)gameOver.GetGOIP() + 6, 10 , 3, '#');

char strBuf[20] = "Game Over";

PrintLine(screen, GameOverWidth, strBuf, 18, 1);

sprintf\_s(strBuf, "%s %d", "Enemyes killed:", KillsCounter);

PrintLine(screen, GameOverWidth, strBuf, 18, 3);

sprintf\_s(strBuf, "Time: %d s", (int)TotalTime / 1000);

PrintLine(screen, GameOverWidth, strBuf, 18, 5);

sprintf\_s(strBuf, "%s", "Main menu");

PrintLine(screen, GameOverWidth, strBuf, 18, 7);

sprintf\_s(strBuf, "%s", "Quit");

PrintLine(screen, GameOverWidth, strBuf, 18, 9);

WriteConsoleOutputCharacter(hConsole, screen, GameOverWidth \* GameOverHeight, { 0, 0 }, &dwBytes);

}

void DrawRecords(char\* screen/\*, Records& records\*/)

{

SMALL\_RECT rect = { 0, 0, RecordsWidth, RecordsHeight };

SetConsoleWindowInfo(hConsole, true, &rect);

DWORD dwBytes = 0;

char strBuf[20] = "Game Over";

Record tempRec;

memset(screen, ' ', RecordsWidth \* RecordsHeight);

for (int i = 0; i < 4; i++)

{

tempRec = records.GetRecord(i);

if (i == 3)

sprintf\_s(strBuf, "Random lvl");

else

sprintf\_s(strBuf, "Level num: %d", i);

PrintLine(screen, RecordsWidth, strBuf, 18, 3 \* i);

sprintf\_s(strBuf, "Enemyes killed: %d", tempRec.kills);

PrintLine(screen, RecordsWidth, strBuf, 18, 3 \* i + 1);

sprintf\_s(strBuf, "Time: %d", tempRec.time);

PrintLine(screen, RecordsWidth, strBuf, 18, 3 \* i + 2);

}

WriteConsoleOutputCharacter(hConsole, screen, GameOverWidth \* GameOverHeight, { 0, 0 }, &dwBytes);

}

int main() {

char Titles[BUF\_SIZE];

gameState = gsMainMenu;

SetConsoleTitle("Shooter 3d");

CreateMap();

char\* screen = new char[ScreenWidth \* ScreenHeight];

char\* MainMenuScreen = new char[MenuWidth \* MenuHeight];

char\* LvlMenuScreen = new char[LvlMenuWidth \* LvlMenuHeight];

char\* PausMenuScreen = new char[PauseWidth \* PauseHeight];

char\* GameOverScreen = new char[GameOverWidth \* GameOverHeight];

char\* RecordsScreen = new char[RecordsWidth \* RecordsHeight];

hConsole = CreateConsoleScreenBuffer(GENERIC\_READ | GENERIC\_WRITE,

0, NULL, CONSOLE\_TEXTMODE\_BUFFER, NULL);

DWORD dwBytes = 0;

SetConsoleActiveScreenBuffer(hConsole);

GetConsoleTitle(Titles, BUF\_SIZE);

font.cbSize = sizeof(CONSOLE\_FONT\_INFOEX);

font.nFont = 0;

font.FontFamily = 0;

font.FontWeight = 400;

font.dwFontSize.X = 32;

font.dwFontSize.Y = 64;

SetCurrentConsoleFontEx(hConsole, true, &font);

EventHandler event;

int tm1 = clock();

int tm2 = clock();

Menu menu(map);

Levels levels;

Pause pause;

GameOver gameOver;

sound = new Sound();

LoadMyRecords();

gameState = gsMainMenu;

while (gameState != gsQuit)

{

font.cbSize = sizeof(CONSOLE\_FONT\_INFOEX);

font.nFont = 0;

font.FontFamily = 0;

font.FontWeight = 400;

font.dwFontSize.X = 32;

font.dwFontSize.Y = 64;

SetCurrentConsoleFontEx(hConsole, true, &font);

while (gameState == gsMainMenu)

{

DrawMainMenu(MainMenuScreen, menu);

event.Update();

if (event.isKeyUp(' '))

gameState = menu.Move(IN);

if (event.isKeyUp('W'))

gameState = menu.Move(DOWN);

if (event.isKeyUp('S'))

gameState = menu.Move(UP);

}

while (gameState == gsChooseLevel)

{

DrawLevelsMenu(LvlMenuScreen, levels);

event.Update();

if (event.isKeyUp(' '))

gameState = levels.Move(IN);

if (event.isKeyUp('W'))

gameState = levels.Move(DOWN);

if (event.isKeyUp('S'))

gameState = levels.Move(UP);

if (event.isKeyUp(VK\_ESCAPE))

gameState = levels.Move(BACK);

}

while (gameState == gsLoadLevel)

{

LoadLevel(levels);

SMALL\_RECT rect = { 0, 0, ScreenWidth, ScreenHeight };

SetConsoleWindowInfo(hConsole, true, &rect);

gameState = gsRun;

}

while (gameState == gsPause)

{

DrawPauseMenu(PausMenuScreen, pause);

event.Update();

if (event.isKeyUp('W'))

gameState = pause.Move(DOWN);

if (event.isKeyUp('S'))

gameState = pause.Move(UP);

if (event.isKeyUp(' '))

gameState = pause.Move(IN);

}

font.FontFamily = 0;

font.FontWeight = 400;

font.dwFontSize.X = 16;

font.dwFontSize.Y = 32;

SetCurrentConsoleFontEx(hConsole, true, &font);

while (gameState == gsShowRecords)

{

DrawRecords(RecordsScreen);

event.Update();

if (event.isKeyUp(VK\_ESCAPE))

gameState = records.Move(BACK);

}

while (gameState == gsGameOver)

{

DrawGameOver(GameOverScreen, gameOver);

event.Update();

if (event.isKeyUp('W') || event.isKeyUp('S'))

gameState = gameOver.Move(UP);

if (event.isKeyUp(' '))

gameState = gameOver.Move(IN);

}

font.cbSize = sizeof(CONSOLE\_FONT\_INFOEX);

font.nFont = 0;

font.FontFamily = 0;

font.FontWeight = 400;

font.dwFontSize.X = 8;

font.dwFontSize.Y = 8;

SetCurrentConsoleFontEx(hConsole, true, &font);

while (gameState == gsRun)

{

tm1 = clock();

float time = (tm1 - tm2);

TotalTime += time;

tm2 = tm1;

MovePlayer(time);

MoveEnemyes(time);

UpdateBullets();

UpdateEnemyes();

MoveBullets(time);

DrawWalls(screen);

DrawEnemyes(screen);

DrawBullets(screen);

DrawMap(screen, MapHeigth, MapWidth, ScreenWidth);

DrawInterface(screen, time);

WriteConsoleOutputCharacter(hConsole, screen, ScreenWidth \* ScreenHeight, { 0, 0 }, &dwBytes);

if (player.HP <= 0)

{

sound->Play(sndGameOver);

gameState = gsGameOver;

bool isUpdate = false;

Record tempRec = records.GetRecord(levels.GetLevelNum());

if (KillsCounter > tempRec.kills ||

(KillsCounter == tempRec.kills && TotalTime / 1000 < tempRec.time))

{

tempRec.kills = KillsCounter;

tempRec.time = TotalTime / 1000;

records.SetRecord(levels.GetLevelNum(), tempRec);

}

SaveRecords();

}

}

}

return 0;

}

Файл Engine.h, содержащий основные константы и структуру Player

#ifndef ENGINE\_H

#define ENGINE\_H

#pragma comment(lib, "winmm.lib")

#include <fstream>

#include <iostream>

#include <cmath>

#include <ctime>

#include <list>

#include <Windows.h>

#define PI 3.14159265

#define BULLET\_SHADE '@'

#define RECORD "Records.txt"

#define max(a, b) ((a) > (b)) ? a : b

#define BUF\_SIZE 30

#define BACK 2

enum GameState {gsMainMenu, gsRun, gsPause, gsQuit, gsChooseLevel,

gsLoadLevel, gsShowRecords, gsGameOver};

void makeScreenShoot(char\* screen);

const int ScreenWidth = 120;

const int ScreenHeight = 120;

const float fViewAngle = PI / 4.0f;

const float fViewRad = 10.0f;

const float fDepth = 16.0f;

const char EDSSign = 1;

typedef struct \_Player

{

int HP;

float fX;

float fY;// = 8.0f;

float fAngle;// = 0.0f;

float fSpeed;// = 0.001f;

float fAngleSpeed;// = 0.001f;

float tmReload;// = 1000.0f;

float tmAfterShoot;// = 1000.0f;

float fSize;

\_Player()

{

HP = 20;

fX = 8.0f;

fY = 8.0f;

fAngle = 0.0f;

fSize = 0.3f;

fSpeed = 0.001f;

fAngleSpeed = 0.002f;

tmReload = 1000.0f;

tmAfterShoot = 1000.0f;

}

} Player;

#endif

Класс Bullet

#ifndef BULLET\_H

#define BULLET\_H

#include "Enemy.h"

#include "Engine.h"

#include <iostream>

#include <list>

class Bullet

{

int Damage;

float x, y;

float fspeed;

float dx, dy;

bool isDestroy;

public:

Bullet(float x, float y, float dx, float dy);

void Move(const std::string& Map, float time, std::list<Enemy\*> enemyes);

float getX()

{

return x;

}

float getY()

{

return y;

}

bool getDestroy()

{

return isDestroy;

}

void setDestroy(bool value)

{

isDestroy = value;

}

int GetDamage()

{

return Damage;

}

};

#endif

#include "Bullet.h"

Bullet::Bullet(float x, float y, float dx, float dy)

{

this->x = x;

this->y = y;

this->dx = dx;

this->dy = dy;

Damage = 5;

fspeed = 0.002f;

isDestroy = false;

}

extern int MapWidth, MapHeigth;

void Bullet::Move(const std::string& Map, float time, std::list<Enemy\*> enemyes)

{

if (!isDestroy)

{

x += (float)(dx \* fspeed \* time);

y += (float)(dy \* fspeed \* time);

if (x > MapWidth && y > MapHeigth)

isDestroy = true;

else

if (Map[int(y) \* MapWidth + int(x)] == '#')

isDestroy = true;

}

}

Класс Enemy

#ifndef ENEMY\_H

#define ENEMY\_H

#include <iostream>

const char EMPTY = ' ';

class Enemy

{

protected:

float x, y;

float fspeed;

float size;

float fViewRad;

int textureWidth;

int textureHeight;

bool isDestroy;

int FrameNum, FrameAmount;

float tmLastFrame, tmFramePeriod;

char \*\*\*Frames;

virtual void InitFrames();

void Animate(float time);

public:

bool operator==(const Enemy &en2) const;

bool operator>=(const Enemy &en2)const;

bool operator < (const Enemy &en2)const;

bool operator>(const Enemy &en2)const;

char \*\*texture;

virtual void Move(float plX, float plY, float time, std::string &map);

virtual void InitTexture();

float getX() const

{

return x;

}

float getY() const

{

return y;

}

int getTextureWidth() const

{

return textureWidth;

}

int getTextureHeight() const

{

return textureHeight;

}

void setDestroy(bool value)

{

isDestroy = value;

}

bool getDestroy()

{

return isDestroy;

}

float getSize()

{

return size;

}

};

#endif

#include <cmath>

#include "Enemy.h"

#include "Engine.h"

Enemy::Enemy(float x, float y, float size)

{

this->x = x;

this->y = y;

this->size = size;

fspeed = 0.0005f;

textureWidth = 11;

textureHeight = 8;

fViewRad = 3.0f;

isDestroy = false;

tmLastFrame = 0.0f;

tmFramePeriod = 1000.0f;

FrameAmount = 2;

InitTexture();

InitFrames();

}

void Enemy::InitTexture()

{

texture = new char\*[textureHeight];

for (int i = 0; i < textureWidth; i++)

texture[i] = new char[textureWidth];

char tempTexture[8][11] = { {' ', ' ', '$', ' ', ' ', ' ', ' ', ' ', '$', ' ', ' '},

{' ', ' ', ' ', '$', ' ', ' ', ' ', '$', ' ', ' ', ' '},

{' ', ' ', '$', '$', '$', '$', '$', '$', '$', ' ', ' '},

{' ', '$', '$', '0', '$', '$', '$', '0', '&', '$', ' '},

{'$', '$', '$', '$', '$', '$', '$', '$', '$', '$', ' '},

{'$', ' ', '$', '$', '$', '$', '$', '$', '$', ' ', '$'},

{'$', ' ', '$', ' ', ' ', ' ', ' ', ' ', '$', ' ', '$'},

{' ', ' ', ' ', '$', '$', ' ', '$', '$', ' ', ' ', ' '} };

for (int i = 0; i < textureHeight; i++)

for (int j = 0; j < textureWidth; j++)

{

texture[i][j] = tempTexture[i][j];

}

}

void Enemy::InitFrames()

{

Frames = new char\*\*[2];

for (int i = 0; i < 2; i++)

{

Frames[i] = new char\*[textureHeight];

for (int j = 0; j < textureHeight; j++)

{

Frames[i][j] = new char[textureWidth];

}

}

char TempFrame[8][11] = { {' ', ' ', '$', ' ', ' ', ' ', ' ', ' ', '$', ' ', ' '},

{' ', ' ', ' ', '$', ' ', ' ', ' ', '$', ' ', ' ', ' '},

{' ', ' ', '$', '$', '$', '$', '$', '$', '$', ' ', ' '},

{' ', '$', '$', '0', '$', '$', '$', '0', '&', '$', ' '},

{'$', '$', '$', '$', '$', '$', '$', '$', '$', '$', ' '},

{'$', ' ', '$', '$', '$', '$', '$', '$', '$', ' ', '$'},

{'$', ' ', '$', ' ', ' ', ' ', ' ', ' ', '$', ' ', '$'},

{' ', ' ', ' ', '$', '$', ' ', '$', '$', ' ', ' ', ' '} };

for (int i = 0; i < textureHeight; i++)

for (int j = 0; j < textureWidth; j++)

Frames[0][i][j] = TempFrame[i][j];

char TempFrame2[8][11] = { {' ', ' ', ' ', ' ', '$', ' ', '$', ' ', ' ', ' ', ' '},

{' ', ' ', ' ', '$', ' ', ' ', ' ', '$', ' ', ' ', ' '},

{'$', ' ', '$', '$', '$', '$', '$', '$', '$', ' ', '$'},

{'$', '$', '$', '0', '$', '$', '$', '0', '&', '$', '$'},

{'$', '$', '$', '$', '$', '$', '$', '$', '$', '$', '$'},

{' ', ' ', '$', '$', '$', '$', '$', '$', '$', ' ', ' '},

{' ', ' ', '$', ' ', ' ', ' ', ' ', ' ', '$', ' ', ' '},

{' ', ' ', ' ', '$', '$', ' ', '$', '$', ' ', ' ', ' '} };

for (int i = 0; i < textureHeight; i++)

for (int j = 0; j < textureWidth; j++)

Frames[1][i][j] = TempFrame2[i][j];

}

void Enemy::Animate(float time)

{

if (tmLastFrame > tmFramePeriod)

{

if (++FrameNum >= FrameAmount)

{

FrameNum = 0;

}

texture = Frames[FrameNum];

tmFramePeriod = 0.0f;

}

else

tmLastFrame += time;

}

void Enemy::Move(float plX, float plY, float time, std::string &map)

{

if (!isDestroy)

{

float distance = sqrt(pow(x - plX, 2) + pow(y - plY, 2));

if (distance < fViewRad)

if (distance > size + 0.2f)

{

float dx = (plX - x) / distance;

float dy = (plY - y) / distance;

float testX = x + dx \* fspeed \* time;

float testY = y + dy \* fspeed \* time;

if (map[testY \* MapWidth + testX] != '#')

{

x = testX;

y = testY;

}

}

Animate(time);

}

}

extern Player player;

bool Enemy::operator < (const Enemy &en2) const

{

float dist1 = sqrt(pow(player.fX - this->getX(), 2) + pow(player.fY - this->getY(), 2));

float dist2 = sqrt(pow(player.fX - en2.getX(), 2) + pow(player.fY - en2.getY(), 2));

return dist1 < dist2;

}

bool Enemy::operator>(const Enemy &en2) const

{

float dist1 = sqrt(pow(player.fX - this->getX(), 2) + pow(player.fY - this->getY(), 2));

float dist2 = sqrt(pow(player.fX - en2.getX(), 2) + pow(player.fY - en2.getY(), 2));

return dist1 > dist2;

}

bool Enemy::operator==(const Enemy &en2) const

{

float dist1 = sqrt(pow(player.fX - this->getX(), 2) + pow(player.fY - this->getY(), 2));

float dist2 = sqrt(pow(player.fX - en2.getX(), 2) + pow(player.fY - en2.getY(), 2));

return dist1 == dist2;

}

Класс EnemySoldier

#ifndef ENEMY\_SOLDIER\_H

#define ENEMY\_SOLDIER\_H

#include "Bullet.h"

#include "Enemy.h"

class EnemySoldier: public Enemy

{

protected:

float tmAfterShoot, tmReload;

void Attack(float plX, float plY, float time);

public:

EnemySoldier(float x, float y, float size): Enemy(x, y, size)

{

tmAfterShoot = 1000.0f;

tmReload = 1000.0f;

textureHeight = 4;

textureWidth = 4;

InitTexture();

}

void Move(float plX, float plY, float time, std::string &map);

void InitTexture();

void InitFrames();

};

#endif

#include "EnemySoldier.h"

#include <cmath>

void EnemySoldier::InitTexture()

{

char tempTexture[4][4] = { {'$', '$', '$', '$'},

{'$', ' ', ' ', '$'},

{'$', ' ', ' ', '$'},

{'$', '$', '$', '$'} };

int i;

int j;

for (i = 0; i < 4; i++)

for (j = 0; j < 4; j++)

{

texture[i][j] = tempTexture[i][j];

}

}

void EnemySoldier::InitFrames()

{

}

extern std::list<Bullet> bullets;

void EnemySoldier::Attack(float plX, float plY, float time)

{

if (tmAfterShoot > tmReload)

{

float distance = sqrt(pow(plX - x, 2) + pow(plY - y, 2));

float dx = (plX - x) / distance;

float dy = (plY - y) / distance;

float b\_x = x + (size + 0.1f) \* dx;

float b\_y = y + (size + 0.1f) \* dy;

//Bullet \*myBullet = new Bullet(x, y, dx, dy);

bullets.push\_back(Bullet(b\_x,

b\_y, dx, dy));

tmAfterShoot = 0;

}

else

tmAfterShoot += time;

}

void EnemySoldier::Move(float plX, float plY, float time, std::string &map)

{

if (!isDestroy)

{

float distance = sqrt(pow(plX - x, 2) + pow(plY - y, 2));

if ((distance < fViewRad) && (distance > size + 0.2f))

{

Attack(plX, plY, time);

}

}

}

Класс EventHandler

#ifndef EVENT\_HANDLER\_H

#define EVENT\_HANDLER\_H

#include <windows.h>

/\*This is not an event handler per se.

The isKeyDown property determines whether the key is currently pressed.

The isKeyUp property determines whether the key was omitted at the time

of the last call to the Update function\*/

typedef struct \_KeyState

{

char Key;

bool isKeyDown;

bool isKeyUp;

} KeyState;

class EventHandler

{

KeyState KeysState[256];

public:

EventHandler();

bool isKeyDown(char Key);

bool isKeyUp(char Key);

void Update();

};

#endif

#include "EventHadler.h"

EventHandler::EventHandler()

{

unsigned char i;

for (i = 0; i < 255; i++)

{

KeysState[i].isKeyDown = false;

KeysState[i].isKeyUp = false;

KeysState[i].Key = i;

}

}

bool EventHandler::isKeyDown(char Key)

{

return KeysState[Key].isKeyDown;

}

bool EventHandler::isKeyUp(char Key)

{

return KeysState[Key].isKeyUp;

}

void EventHandler::Update()

{

unsigned char i;

for (i = 0; i < 255; i++)

if (GetAsyncKeyState(KeysState[i].Key))

{

KeysState[i].isKeyDown = true;

KeysState[i].isKeyUp = false;

}

else

{

if (KeysState[i].isKeyDown)

KeysState[i].isKeyUp = true;

else

KeysState[i].isKeyUp = false;

KeysState[i].isKeyDown = false;

}

}

Класс Sound

#pragma once

enum SoundName {sndShoot, sndGameOver, sndMain, sndAmount};

typedef struct \_SoundData

{

char \*sndBuf;//буфер, куда записываются данные аудифайла

int size;//размер буфера

bool isLoaded;//флаг, который показывает загружен ли аудифайл в динамичсекую память

} SoundData;

class Sound

{

const char\* FileShoot = "gun1.wav";

const char\* FileGameOver = "Game Over.wav";

const char\* FileMain = "Main Theme.wav";

SoundName sndName;

SoundData sounds[sndAmount];

public:

Sound();

void Play(SoundName Name);

};

#include "Sound.h"

#include <Windows.h>

#include <iostream>

#include <fstream>

Sound::Sound()

{

int size;

for (int i = 0; i < sndAmount; i++)

{

std::ifstream f;

switch (i)

{

case sndShoot:

f.open(FileShoot, std::ios::binary);

break;

case sndGameOver:

f.open(FileGameOver, std::ios::binary);

break;

case sndMain:

f.open(FileMain, std::ios::binary);

break;

}

if (f.is\_open())

{

f.seekg(0, std::ios\_base::end);

size = f.tellg();

sounds[i].size = size;

sounds[i].sndBuf = new char[size];

f.seekg(0, std::ios\_base::beg);

f.read(sounds[i].sndBuf, size);

f.close();

sounds[i].isLoaded = true;

}

else

sounds[i].isLoaded = false;

}

}

void Sound::Play(SoundName Name)

{

if (sounds[Name].isLoaded)

sndPlaySound((LPCSTR)sounds[Name].sndBuf, SND\_MEMORY |SND\_ASYNC);

}

Класс EDS

#pragma once

class EDS

{

const int R = 2827;

const int SecretStartVaue = 133;

public:

int GetHash(char \*date, int size);

};

#include "EDS.h"

#define sqr(a) a \* a

int EDS::GetHash(char\* data, int size)

{

int h0 = SecretStartVaue;

int h;

for (int i = 0; i < size; i++)

{

h = (sqr(h0 + (unsigned char) data[i])) % R;

h0 = h;

}

return h0;

}

Класс Records

#pragma once

#include "Engine.h"

const int RecordsWidth = 20;

const int RecordsHeight = 12;

typedef struct \_Record

{

int kills;

int time;

} Record;

class Records

{

Record defaultRecord = { 0, 999 };

Record values[4];

const int MaxLevelNum = 4;

public:

Records();

void SetDefaultRecords();

void SetRecord(int n, Record value);

Record GetRecord(int n);

GameState Move(int direction);

};

#include "Records.h"

Records::Records()

{

}

Record Records::GetRecord(int n)

{

if (n < 0 || n >= MaxLevelNum)

n = 0;

return values[n];

}

void Records::SetDefaultRecords()

{

for (int i = 0; i < MaxLevelNum; i++)

{

values[i] = defaultRecord;

}

}

GameState Records::Move(int direction)

{

GameState gameState = gsShowRecords;

if (direction == BACK)

{

gameState = gsMainMenu;

}

return gameState;

}

void Records::SetRecord(int n, Record value)

{

if (n < 0 || n >= MaxLevelNum)

n = 0;

values[n] = value;

}

Класс Menu

#pragma once

#include "Engine.h"

#include <iostream>

#define DOWN -1

#define UP 1

#define IN 0

const int MenuWidth = 12;

const int MenuHeight = 9;

enum ItemPointer {ipStart, ipLevels, ipRecords, ipQuit, ipAmount};

class Menu

{

ItemPointer itemPointer;

std::string map;

public:

Menu(std::string &map);

GameState Move(int dir);

ItemPointer GetIP();

};

#include "Menu.h"

Menu::Menu(std::string &map)

{

this->map = map;

itemPointer = ipStart;

}

GameState Menu::Move(int dir)

{

GameState gameState = gsMainMenu;

if (dir)

{

itemPointer = (ItemPointer)((itemPointer + dir + ipAmount) % ipAmount);

}

else

{

switch (itemPointer)

{

case ipStart:

gameState = gsRun;

break;

case ipQuit:

gameState = gsQuit;

break;

case ipLevels:

gameState = gsChooseLevel;

break;

case ipRecords:

gameState = gsShowRecords;

break;

default:

break;

}

}

return gameState;

}

ItemPointer Menu::GetIP()

{

return itemPointer;

}

Класс Levels

#pragma once

#include <iostream>

#include "Engine.h"

const int LvlMenuWidth = 12;

const int LvlMenuHeight = 9;

enum ObjectType {otEnemy, otEnemySoldier};

typedef struct \_Object

{

ObjectType type;

float x, y;

} Object;

typedef struct \_Level

{

int ObjectsNum;//кол-во объектов на уровне

Object\* objects;//казатель на массив объектов

int MapWidth, MapHeight;//высота и ширина карты

std::string map;//сама карта

float PlayerX, PlayerY;//координаты, в которых появляется игрок при загрузке карты

} Level;

class Levels

{

const int MaxLevel = 3;

int LevelNum;

Level \*LevelList;

public:

Levels();

int GetLevelNum();

int GetMaxLevel();

Level GetLevel();

GameState Move(int direction);

};

#include "Levels.h"

Levels::Levels()

{

LevelNum = 0;

LevelList = new Level[MaxLevel];

LevelList[0].ObjectsNum = 2;

LevelList[0].objects = new Object[2];

LevelList[0].objects[0] = { otEnemy, 8.0f, 2.0f };

LevelList[0].objects[1] = { otEnemySoldier, 4.0f, 5.0f };

LevelList[0].MapWidth = 17;

LevelList[0].MapHeight = 9;

LevelList[0].PlayerX = 1;

LevelList[0].PlayerY = 7;

LevelList[0].map += "#################";

LevelList[0].map += "# #";

LevelList[0].map += "# #### f";

LevelList[0].map += "# # # # # #";

LevelList[0].map += "# # ### ####";

LevelList[0].map += "# # # # # #";

LevelList[0].map += "# #### #";

LevelList[0].map += "# #";

LevelList[0].map += "#################";

LevelList[1].ObjectsNum = 0;

LevelList[1].MapHeight = 5;

LevelList[1].MapWidth = 5;

LevelList[1].map += "#####";

LevelList[1].map += "# #";

LevelList[1].map += "# f";

LevelList[1].map += "# #";

LevelList[1].map += "#####";

LevelList[1].PlayerX = 2;

LevelList[1].PlayerY = 2;

LevelList[2].ObjectsNum = 1;

LevelList[2].objects = new Object[1];

LevelList[2].objects[0] = {otEnemySoldier, 3.0f, 3.0f};

LevelList[2].MapHeight = 10;

LevelList[2].MapWidth = 10;

LevelList[2].PlayerX = 7;

LevelList[2].PlayerY = 7;

LevelList[2].map += "##########";

LevelList[2].map += "# #";

LevelList[2].map += "# #";

LevelList[2].map += "# #";

LevelList[2].map += "### ### ##";

LevelList[2].map += "# # #";

LevelList[2].map += "# #";

LevelList[2].map += "# #####";

LevelList[2].map += "# f";

LevelList[2].map += "##########";

}

Level Levels::GetLevel()

{

return LevelList[LevelNum];

}

int Levels::GetLevelNum()

{

return LevelNum;

}

GameState Levels::Move(int direction)

{

if (direction)

{

if (direction == BACK)

return gsMainMenu;

LevelNum = (LevelNum + MaxLevel + direction) % MaxLevel;

return gsChooseLevel;

}

else

return gsLoadLevel;

}

int Levels::GetMaxLevel()

{

return MaxLevel;

}

Класс GameOver

#pragma once

#include "Engine.h";

const int GameOverHeight = 12;

const int GameOverWidth = 20;

enum GOItemPointer{goipMainMenu, goipQuit};

class GameOver

{

GOItemPointer itemPointer;

public:

GameOver();

GameState Move(int direction);

GOItemPointer GetGOIP();

};

#include "GameOver.h"

GameOver::GameOver()

{

itemPointer = goipMainMenu;

}

GameState GameOver::Move(int direction)

{

GameState gameState = gsGameOver;

if (direction)

{

itemPointer = (GOItemPointer)!itemPointer;

}

else

{

if (!itemPointer)

gameState = gsMainMenu;

else

gameState = gsQuit;

}

return gameState;

}

GOItemPointer GameOver::GetGOIP()

{

return itemPointer;

}

Класс Pause

#pragma once

#include "Engine.h"

const int PauseWidth = 12;

const int PauseHeight = 9;

enum ItemPausePointer {ippContinue, ippMainMenu, ippQuit, ippAmount};

class Pause

{

ItemPausePointer itemPointer;

public:

Pause();

GameState Move(int direction);

ItemPausePointer GetIPP();

};

#include "Pause.h"

Pause::Pause()

{

itemPointer = ippContinue;

}

GameState Pause::Move(int direction)

{

GameState gameState = gsPause;

if (direction)

{

itemPointer = (ItemPausePointer)((itemPointer + direction + ippAmount) % ippAmount);

}

else

{

switch (itemPointer)

{

case ippMainMenu:

gameState = gsMainMenu;

break;

case ippContinue:

gameState = gsRun;

break;

case ippQuit:

gameState = gsQuit;

break;

}

}

return gameState;

}

ItemPausePointer Pause::GetIPP()

{

return itemPointer;

}