**СОДЕРЖАНИЕ**

1.ПОСТАНОВКА ЗАДАЧИ……....…………………………………….2

2.ВВЕДЕНИЕ…………………..………………………………………...3

3.ОБЗОР ЛИТЕРАТУРЫ……………………………………………......4

3.1.Обзор методов и решений поставленной задачи…….……………4

3.2.Обзор библиотеки SFML…….……………………………………...5

3.3 Обзор аналогов игры “Пакман”…..………………………………...5

4.Функциональное проектирование….………….…………………......6

4.1.Структура входных и выходных данных.……………..…………...6

4.2.Разработка диаграммы классов…………………………...………...6

4.3.Классы игры………………………………………………………….

4.3.1.Классы игры………………………………………………..………7

4.3.2.Класс сущности……………………………………………………..

4.3.3.Класс игрока………………………………………………………..

4.3.4.Класс врага………………………………………………………….

4.3.5.Класс-обработчик врагов…………………………………………..

4.3.6.Класс сущностей, которые могут перемещаться…………………

4.3.7.Класс сцены…………………………………………………………

4.3.8.Класс уровня………………………………………………………..

4.3.9.Класс худа…………………………………………………………..

5.Разработка программных модулей……………………………………..

5.1.Разработка схем алгоритмов………………………………………….

5.2.Разработка алгоритмов………………………………………………..

5.2.1.Разработка алгоритма метода Update()……………………………

5.2.2.Разработка алгоритма метода MainLoop()………………………..

6.Результаты работы………………………………………………………

Заключение………………………………………………………………

Список литературы………………………………………………………

ПРИЛОЖЕНИЕ А……………………………………………...…….…10

ПРИЛОЖЕНИЕ Б…………………………………………………………

ПРИЛОЖЕНИЕ В…………………………………………………………

ПРИЛОЖЕНИЕ Г…………………………………………………………

**1 ПОСТАНОВКА ЗАДАЧИ**

В проекте будет реализована игровая логика классической игры «Пакман»; карта, по которой перемещаются игрок и враги, а так же содержащая стены и «точки», которые ест пакман, и которые нужно собрать полностью для успешного прохождения уровня. В игре будет реализовано меню, в котором можно будет выбрать один из трёх предложенных уровней для прохождения; пункт настроек, в котором можно будет поменять цвет стен а так же цвета игрового персонажа и врагов. Игрок будет иметь 3 жизни для прохождения уровня а так же суммарный счет, который будет формироваться из съеденных точек. В случае столкновения с врагом игрок будет терять одну жизнь, возрождаться в начальной точке, а все съеденные им точки будут вновь отображены на карте. Графическая составляющая проекта будет реализована посредством использования библиотеки SFML.

**2 ВВЕДЕНИЕ**

Язык программирования C++ представляет высокоуровневый компилируемый язык программирования общего назначения со статической типизацией, который подходит для создания самых различных приложений. На сегодняшний день C++ является одним из самых популярных и распространенных языков.

Фактически вначале C++ просто дополнял язык C некоторыми

возможностями объектно-ориентированного программирования. И поэтому сам Страуструп вначале называл его как “C with classes”(“Си с классами”). Впоследствии новый язык стал набирать популярность. В него были добавлены новые возможности, которые делали его не просто дополнением к Си, а совершенно новым языком программирования. В итоге “Си с классами” был переименован в C++. И с тех пор оба языка стали развиваться независимо друг от друга.

С++ является мощным языком, унаследовав от Си богатые возможности по работе с памятью. Поэтому нередко С++ находит свое применение в системном программировании, в частности, при создании операционных систем, драйверов, различных утилит, антивирусов и т.д. Но только системным программированием применение данного языка не ограничивается. С++ можно использовать в программах любого уровня, где важны скорость работы и производительность. Нередко он применяется для создания графических приложений, различных прикладных программ. Также особенно часто его используют для создания игр с богатой насыщенной визуализацией. Кроме того, в последнее время набирает ход мобильное направление, где С++ тоже нашел свое применение. И даже в веб-разработке также можно использовать С++ для создания веб-приложений или каких-то вспомогательных сервисов, которые обслуживают веб-приложения. В общем С++ - язык широкого пользования, на котором можно создавать практически любые виды программы. С++ является компилируемым языком, а это значит, что компилятор транслирует исходный код на С++ в исполняемый файл, который содержит набор машинных инструкций. Но разные платформы имеют свои особенности, поэтому скомпилированные программы нельзя просто перенести с одной платформы на другую и там уже запустить. Однако на уровне исходного кода программы на С++ по большей степени обладают переносимостью, если не используются какие-то специфичные для текущей ос функции. А наличие компиляторов, библиотек и инструментов разработки почти под все распространенные платформы позволяет компилировать один и тот же исходный код на С++ в приложения под эти платформы.

**3 ОБЗОР ЛИТЕРАТУРЫ**

**3.1 Обзор методов и алгоритмов решения поставленной задачи**

Для реализации игрового процесса используется класс Game, который содержит в себе поле типа Scene, которое содержит текущую сцену, на которой размещены игрок, враги, непосредственно сама карта и худ игры, поле типа RenderWindow, которое хранит в себе текущее окно, в котором отрисовывается игра, поле типа std::vector<Map>, которое хранит в себе все загруженные карты, а также поле типа std::list<sf::Texture>, которое хранит в себе все загруженные текстуры. Главный цикл отрисовки реализован при помощи метода void MainLoop(), который будет отрисовывать каждый кадр игры до тех пор, пока пользователь не закроет окно с игрой. Внутри метода MainLoop() используется метод Update(), который обрабатывает всю логику игры, а так же метод Redraw(), который отвечает за отрисовку сцены в окне.

**3.2 Обзор библиотеки SFML**

За реализацию графической составляющей игры отвечает библиотека SFML. SFML(Simple and Fast Multimedia Library) – свободная кроссплатформенная мультимедийная библиотека. Представляет собой объектно-ориентированный аналог SDL. SFML содержит ряд модулей для простого программирования игр и мультимедиа приложений. Основные модули, которые будут использоваться для реализации: System, Window, Graphics. Модуль System управляет временем и потоками а так же является обязательным, так как все модули зависят от него. Модуль Window используется для управления окнами и взаимодействия с пользователем. Модуль Graphics служит для отображения графических примитивов и изображений.[1]

* 1. **Обзор аналогов игры “Пакман”**

1. “Dig Dug”(1982) – игра, разработанная компанией “MAMCO”.

Главной задачей в игре является уничтожение монстров, живущих под землей, раздувая их, или пока они не лопнут, или сбрасывая на них камни. В игре есть два вида врагов: Pookas, круглые красные монстры, которые носят желтые очки, и Fygas, зеленые драконы, которые могут выдыхать пламя

1. “Gauntlet”(1985) – игра, разработанная “Atari games” и “Tengen”.

Целью игрока является поиск обозначенного на каждом уровне «выхода», до которого нужно дотронуться. На каждом уровне можно найти много различных предметов, которые восстанавливают здоровье игрока, открывают двери, дают очки или уничтожают всех врагов на экране.

1. “Dodge ‘Em”(1980) – еще один продукт “Atari games”. Игрок управляет одной машиной и должен двигаться против часовой стрелки, избегая машин, управляемых компьютером, единственная цель которых - вызвать лобовое столкновение. Каждая проезжая часть лабиринта имеет четыре пробела вверху, внизу, слева и справа от экрана. Игрок может использовать промежутки для смены полосы движения, чтобы собирать другие точки или избегать машин, управляемых компьютером.

Автомобиль игрока может двигаться на двух скоростях, нормальная

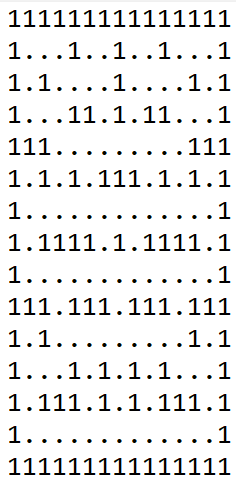
скорость равна скорости компьютера. -управляемые автомобили, или повышенная (удвоенная) скорость, активируемая нажатием кнопки контроллера. У машин с компьютерным управлением только одна скорость. Игроки меняют полосу движения, толкая контроллер в нужном направлении, когда их машина находится возле одного из пробелов на проезжей части.Википедия site:wiki5.ru

**4 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

В данном разделе описываются входные и выходные данные программы, диаграмма классов, а также приводится описание используемых классов и их методов.

**4.1 Структура входных и выходных данных**

Рис.1 – файл level1.txt(карта стен для уровня, где “1” – стена, “.” – “точка”):



**4.2 Разработка диаграммы классов**

Диаграмма классов для курсового проекта приведена в Приложении А

**4.3 Классы игры**

**4.3.1 Класс игры**

Для реализации используется класс Game, который отвечает за работу игры. Он хранит в себе загруженные уровни, текстуры, а также выполняет главный игровой цикл, который содержит в себе методы работы с графикой и игровой логикой

Описание полей класса Game:

sf::RenderWindow window – окно игры

std::vector<Map> – массив загруженных карт

list<sf::Texture> – список загруженных текстур

Описание методов класса Game:

Game() – конструктор

~Game() – деструктор

void Start() – запуск игры

void StartLevel(Map& map) – начало игры на выбранном уровне

void MainLoop() – главный цикл отрисовки графики и выполнения игровой логики

sf::RenderWindow& getWindow() – получение окна, в котором отрисовывается игра

void Update(float deltaTime) – обработка игровой логики

void Redraw(sf::RenderWindow& window) – отрисовка игры

Map& getMap() – получение выбранного уровня

Scene& getScene() – получение текущей сцены

static Game\* getInstance() – получение указателя на инстанс игры

sf::Texture& loadTexture(const std::string& path) – загрузка текстуры в массив текстур

**4.3.2 Класс сущности**

В программе используется базовый абстрактный класс Entity, от которого в дальнейшем наследуются классы Imovable, EnemyHandler. Класс содержит два виртуальных метода, которые перегружаются под нужды наследуемых классов.

Описание полей класса Entity:

sf::Vector2f – координаты сущности на карте

sf::Sprite entitySprite – спрайт сущности

Описание методов класса Entity:

virtual ~Entity() – деструктор по умолчанию

virtual void draw(sf::RenderWindow& window) – метод отрисовки сущности в окне

virtual void update(float deltaTime) – функция обработки логики

void setPosition(sf::Vector2f pos) – установка позиции сущности

sf::Vector2f getPosition()– получение позиции сущности

**4.3.3 Класс игрока**

Наследуется от класса Imovable

Класс Player

Описание полей класса Player:

int score – игровой счет

int countOfHealth – количество жизней игрока

Описание методов класса Player:

Player() – конструктор по умолчанию

Player(sf::Sprite& playerSprite) - конструктор

void update(float deltaTime) override – обновление логики игрока

int getScore() – получение счета игрока

void setScore(int value) – установка счета игрока

int getHealth() - получение количества жизней игрока

void setHealth(int value) – установка количества жизней игрока

bool checkCollisionWithEnemies() – проверка столкновения игрока с врагами

sf::Vector2f getPlayerPosition() - получение позиции игрока

**4.3.4 Класс врага**

Наследуется от класса Imovable

Класс Enemy:

Описание полей класса Enemy:

bool isDirAvailable[4] – массив возможных направлений передвижения

Описание методов класса Enemy:

Enemy(sf::Sprite enemySprite) - конструктор

void update(float deltaTime) override – обновление логики врага

void draw(sf::RenderWindow& window) override – отрисовка врага

void updateDir() – обновление направления движения врага

MoveDirection newDir(float deltaTime) – выбор нового направления движения

**4.3.5 Класс-обработчик врагов**

Данный класс хранит в себе массив врагов, а также служит для обработки логики всех врагов

Класс EnemyHandler – класс обработки врагов

Описание полей класса EnemyHandelr:

std::vector<Enemy\*> – массив врагов

Описание методов класса EnemyHandler:

EnemyHandler(size\_t enemiesCount, std::vector<sf::Vector2f>& enemiesPositions, sf::Sprite enemySprite) – конструктор

void update(float deltaTime) override – обновление логики

игрока

bool checkCollisionWithPlayer(Player\* player) – проверка столкновения врага с игроком

**4.3.6 Класс сущностей, которые могут перемещаться**

Данный класс является наследником класса Entity и содержит в себе поля текущего и следующего направлений перемещения сущности, а также скорость перемещения.

Класс Imovable – класс двигающейся сущности

Описание полей класса Imovable:

MoveDirection currDir – текущее направление движения

MoveDirection nextDir – следующее направление движения

float speed – скорость движения

sf::Vector2f size – размер спрайта

Описание методов класса Imovable:

void update(float deltaTime) override – обновление логики сущности

bool checkCollision(sf::Vector2f position) – проверка на столкновение

static sf::Vector2f getOffset(MoveDirection dir) – получение пары чисел, которые определяют направление движения

void setDir(MoveDirection dir) – установка направления движения

**4.3.7 Класс сцены**

Данный класс содержит в себе все, что происходит на игровой сцене – карта, игрок, враги, худ, а также является основным классом, при отрисовке которого можно видеть графическую составляющую игры.

Класс Scene

Описание полей класса Scene:

std::vector<std::unique\_ptr<Entity>> – массив сущностей, которые есть на карте

Map\* map – карта

Описание методов класса Scene:

Inline Map& getMap() – обновление логики сущности

void update(float deltaTime) override – обновление логики сцены

void draw(sf::RenderWindow& window) override – отрисовка сцены

void setMap(Map& map) – установка выбранного уровня на сцену

Entity\* addEntity(std::unique\_ptr<Entity>&& entity) – добавление сущности в массив сущностей

Void destroyEntity(Entity\* entity) – удаление сущности

T\* getEntityOfType() – получение указателя на сущность нужного типа

**4.3.8 Класс уровня**

Данный класс содержит карту уровня, который выбран пользователем

Класс Map – класс сцены

Описание полей класса Map:

std::vector<std::string> – массив сущностей, которые есть на карте

std::vector<sf::Vector2f> enemiesPositions – массив координат врагов

std::Vector2f playerPosition – координаты игрока

Описание методов класса Map:

Map() – конструктор по умолчанию

Map(std::stream& stream) – конструктор

Inline const std::string& operator[](size\_t index) – перегрузка оператора []

char getCell(sf::Vector2f position) const – получение значения, хранящегося в ячейке карты

void draw(sf::RenderWindow& window) – отрисовка карты

void setEmemiesPositions() – установка позиций врагов

void setPlayerPosition() – установка позиции игрока

std::vector<sf::Vector2f>& getEnemiesPositions() – получение позиций врагов

void setCell(char value, sf::Vector2u pos) – установка значения в клетку карты

std::vector<std::string>\* getMap() – получение карты

**4.3.9 Класс худа**

Данный класс содержит спрайты жизней игрока а также текст, который отображает набранный игроком счёт

Класс HoodElements – класс худа

Описание полей класса HoodElements:

sf::Sprite heart – спрайт сердца

sf::Text scoreText – объект класса Text, который отображает счет игрока

Описание методов класса HoodElements:

void createHood() – создание худа

void drawHood(sf::RenderWindow& window, Player player) – отрисовка худа

**5 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

**5.1 Разработка схем алгоритмов**

Блок-схемы алгоритмов приведены в Приложениях Б, В соответственно

**5.2 Разработка алгоритмов**

**5.2.1 Разработка алгоритма метода Update() у класса Player**

Метод Update() у игрока отвечает за управление персонажем, а также проверяет столкновение игрока с врагами

Шаг 1. Начало.

Шаг 2. Для всех четырех возможных направлений движения проверяем, нажата ли соответствующая кнопка. Если была нажата кнопка управления, то переходим к Шагу 3. Если нет – к Шагу 4.

Шаг 3. Переменной nextDir присваиваем значение следующего направления.

Шаг 4. Переменной nextDir присваивается значение None.

Шаг 5. Создаем указатель на карту и присваиваем ему значение посредством вызова метода getMap() для инстанса игры.

Шаг 6. Создаем переменную playerPos и присваиваем ей значение посредством преобразования координат игрока к типу unsigned int.

Шаг 7. Вызываем метод CheckCollisionWithEnemies() для проверки столкновения игрока с врагами.

Шаг 8. В случае, если игрок столкнулся с врагом, переходим к Шагу 9. Если нет – к Шагу 12.

Шаг 9. Уменьшаем количество жизней игрока на 1.

Шаг 10. Присваиваем игроку стартовую позицию.

Шаг 11. Присваиваем переменной currDir значение None.

Шаг 12. Проверяем значение, которое хранится в клетке с координатами, равными преобразованным координатам игрока

Шаг 13. Если в клетке хранится символ ‘.’, увеличиваем значение счета игрока на 10.

Шаг 14. Меняем значение клетки на символ ‘ ’(пробел).

Шаг 15. Вызываем метод Update() для класса Imovable.

Шаг 16. Конец.

**5.2.2 Разработка алгоритма метода MainLoop() для класса Game**

Метод MainLoop() отвечает за обработку и отрисовку одного кадра.

Шаг 1. Начало.

Шаг 2. Создаём переменную deltaTime типа Clock.

Шаг 3 .Создаём переменную time типа float.

Шаг 4. Запуск цикла while, внутри которого происходит обработка игровой логики и отрисовка одного кадра.

Шаг 5. Присваиваем переменной time значение, равное времени, прошедшему с момента отрисовки последнего кадра

Шаг 6. Очищаем окно, в котором отображается игра.

Шаг 7. Обновляем игровую логику.

Шаг 8. Отрисовываем новый кадр.

Шаг 9. Выводим новый кадр в окно игры.

Шаг 10. Создаем переменную event типа Event.

Шаг 11. Запускаем цикл, внутри которого происходит проверка, нажата ли кнопка закрытия окна.

Шаг 12. Если кнопка нажата, закрываем окно.

Шаг 13. Конец.

**6 РЕЗУЛЬТАТЫ РАБОТЫ**

(Меню допилю напишу про меню) рис такой то

После выбора пользователем уровня открывается окно, в котором запускается игра и пользователь может пройти уровень

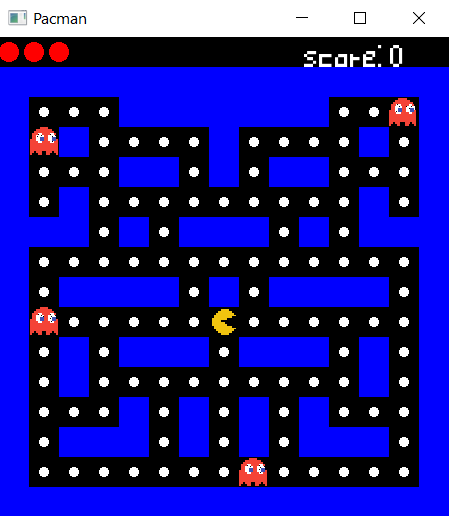
****

рис такой то

В случае успешного прохождения уровня на экран выводится сообщение

«картинка с сообщением »

Рисунок такой то

В случае, если игрок не успел собрать все точки и у него закончились жизни, на экран выводится сообщение

«картинка с сообщением»

Рисунок такой то

(доделать)

**ЗАКЛЮЧЕНИЕ**

В процессе выполнения курсовой работы были закреплены знания основ программирования и алгоритмизации. Были приобретены знания в объектно-ориентированном программировании. Также я познакомился с созданием полноценных проектов при помощи использования библиотеки SFML и убедился, что она подходит для создания несложных игр и проектов с удобным интерфейсом.

В процессе разработки игры “Пакман” были реализованы алгоритмы обновления игровой логики, отрисовки кадров, проверка всевозможных коллизий и т. д. Как упоминалось раннее, моя игра является неполной версией оригинальной игры “Пакман”. Однако она открыта для модернизации. Например, можно создать новый уровень, добавить режим игры “Все или ничего” и т.д.

Игра оказалась простой в изучении, имеет удобный интерфейс.

На основе полученных знаний при создании игры можно будет создать проекты, где нужно будет отрисовывать графику, нажимать кнопки и т.д. Такими примерами могут быть приложение для ведения заметок, ежедневник и т.д.

Создание игры было выполнено на ОС Windows 10, используя среду разработки Visual Studio 2022.

Код программы приведен в приложении Г.

**СПИСОК ЛИТЕРАТУРЫ**

**ПРИЛОЖЕНИЕ А**

(*обязательное*)

Диаграмма классов

**ПРИЛОЖЕНИЕ Б**

(*обязательное*)

Схема алгоритма Update()

**ПРИЛОЖЕНИЕ В**

(*обязательное*)

Схема алгоритма MainLoop()

**ПРИЛОЖЕНИЕ Г**

(*обязательное*)

Код программы

Файл Scene.cpp:

#include "Scene.h"

void Scene::update(float deltaTime) {

for (auto& entity : entities)

entity->update(deltaTime);

}

void Scene::draw(sf::RenderWindow& window) {

Player\* player = getEntityOfType<Player>();

HoodElements hood;

hood.createHood();

hood.drawHood(window, (\*player));

if (map)

map->draw(window);

for (auto& entity : entities)

entity->draw(window);

}

void Scene::setMap(Map& map) {

this->map = &map;

}

Entity\* Scene::addEntity(std::unique\_ptr<Entity>&& entity) {

entities.push\_back(std::move(entity));

return entities.back().get();

}

std::vector<std::unique\_ptr<Entity>>& Scene::getEntities() {

return entities;

}

void Scene::destroyEntity(Entity\* entity) {

for (auto& ent : entities) {

if (ent.get() == entity) {

std::remove(entities.begin(), entities.end(), ent);

break;

}

}

}

Файл Scene.h:

#pragma once

#include "Entity.h"

#include "Map.h"

#include <SFML/Graphics.hpp>

#include "HoodElements.h"

class Scene {

private:

std::vector<std::unique\_ptr<Entity>> entities;

Map\* map;

public:

inline Map& getMap() {

return \*map;

}

void update(float deltaTime);

void draw(sf::RenderWindow& window);

void setMap(Map& map);

Entity\* addEntity(std::unique\_ptr<Entity>&& entity);

std::vector<std::unique\_ptr<Entity>>& getEntities();

void destroyEntity(Entity\* entity);

template<typename T>

T\* getEntityOfType() {

for (auto& entity : entities) {

if (dynamic\_cast<T\*>(entity.get()) != nullptr) {

return reinterpret\_cast<T\*>(entity.get());

}

}

return nullptr;

}

};

Файл Map.cpp:

#include "Map.h"

Map::Map(std::istream& stream) {

std::string tempLine;

while (std::getline(stream, tempLine)) {

data.push\_back(tempLine);

}

setEnemiesPositions();

}

static float GetNearestToHalf(float value) {

return std::abs((value - (unsigned int)value) - 0.5f);

}

void Map::setEnemiesPositions() {

enemiesPositions.resize(4);

enemiesPositions[0] = { 1.5f, 1.5f };

enemiesPositions[1] = { 13.5f, 1.5f };

enemiesPositions[2] = { 1.5f, 13.5f };

enemiesPositions[3] = { 13.5f, 13.5f };

}

std::vector<sf::Vector2f>& Map::getEnemiesPositions() {

return this->enemiesPositions;

}

char Map::getCell(sf::Vector2f position) const {

sf::Vector2u iPos = { (unsigned int)position.x, (unsigned int)position.y };

if (iPos.y >= data.size())

return true;

if (iPos.x >= data[iPos.y].size())

return true;

return data[iPos.y][iPos.x];

}

void Map::setPlayerPosition(sf::Vector2f player\_Position) {

playerPosition = player\_Position;

}

void Map::draw(sf::RenderWindow& window) {

sf::RectangleShape brick(sf::Vector2f(30, 30));

sf::CircleShape point(5.f);

brick.setFillColor(sf::Color::Blue);

point.setFillColor(sf::Color::White);

for (int i = 0; i < 15; i++) {

for (int j = 0; j < 15; j++) {

brick.setPosition(j \* 30, 30 + i \* 30);

point.setPosition(j \* 30 + 10, i \* 30 + 40);

if (data[i][j] == '1' && data[i][j] != ' ') {

window.draw(brick);

}

else if (data[i][j] == '.' && data[i][j] != ' ') {

window.draw(point);

}

}

}

}

void Map::setCell(char value, sf::Vector2u pos) {

data[pos.y][pos.x] = value;

}

std::vector<std::string>\* Map::getMap() {

return &data;

}

Файл Map.h:

#pragma once

#include <iostream>

#include <SFML/Graphics.hpp>

#include <time.h>

#include <fstream>

class Map {

private:

std::vector<std::string> data;

std::vector<sf::Vector2f> enemiesPositions;

sf::Vector2f playerPosition;

public:

Map() = default;

Map(std::istream& stream);

inline const std::string& operator[](size\_t index) const {

return data[index];

}

char getCell(sf::Vector2f position) const;

void draw(sf::RenderWindow& window);

void setEnemiesPositions();

void setPlayerPosition(sf::Vector2f player\_Position);

std::vector<sf::Vector2f>& getEnemiesPositions();

void setCell(char value, sf::Vector2u pos);

std::vector<std::string>\* getMap();

};

Файл Entity.cpp:

#pragma once

#include "Entity.h"

void Entity::draw(sf::RenderWindow& window) {

entitySprite.setPosition(position.x \* 30, (position.y + 1) \* 30);

window.draw(entitySprite);

}

void Entity::setPosition(sf::Vector2f pos) {

position = pos;

}

sf::Vector2f Entity::getPosition() {

return position;

}

Файл Entity.h:

#pragma once

#include <iostream>

#include "SFML\Graphics.hpp"

#include "Map.h"

class Entity {

protected:

sf::Vector2f position;

sf::Sprite entitySprite;

public:

virtual ~Entity() = default;

virtual void draw(sf::RenderWindow& window);

virtual void update(float deltaTime) = 0;

void setPosition(sf::Vector2f pos);

sf::Vector2f getPosition();

};

Файл Player.cpp:

#pragma once

#include "Player.h"

#include "Game.h"

#include "SFML/Graphics/Text.hpp"

#include "EnemyHandler.h"

#include "Map.h"

void Player::update(float deltaTime) {

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {

nextDir = MoveDirection::LEFT;

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {

nextDir = MoveDirection::RIGHT;

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {

nextDir = MoveDirection::UP;

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {

nextDir = MoveDirection::DOWN;

}

std::vector<std::string>\* currMap = Game::getInstance()->getMap().getMap();

sf::Vector2u playerPos = { (unsigned int)position.x, (unsigned int)position.y };

if (checkCollisionWithEnemies()) {

countOfHealth--;

currDir = MoveDirection::NONE;

sf::Vector2f newPos = { 7.5, 8.5 };

setPosition(newPos);

return;

}

if ((\*currMap)[playerPos.y][playerPos.x] == '.') {

setScore(10);

(\*currMap)[playerPos.y][playerPos.x] = ' ';

}

Imovable::update(deltaTime);

}

void Player::setHealth(int value) {

countOfHealth = value;

}

int Player::getHealth() {

return this->countOfHealth;

}

void Player::setScore(int value) {

score += value;

}

int Player::getScore(){

return this->score;

}

sf::Vector2f Player::getPlayerPosition() {

return position;

}

bool Player::checkCollisionWithEnemies() {

Scene& scene = Game::getInstance()->getScene();

EnemyHandler\* handler = scene.getEntityOfType<EnemyHandler>();

return handler->checkCollisionWithPlayer(this);

}

Файл Player.h:

#pragma once

#include "IMovable.h"

#include "EnemyHandler.h"

class Player : public Imovable {

protected:

int score = 0;

int countOfHealth = 3;

public:

Player() = default;

Player(sf::Sprite& playerSptire) {

entitySprite = playerSptire;

position = { 7.5, 8.5 };

}

void update(float deltaTime) override;

int getScore();

void setScore(int value);

int getHealth();

void setHealth(int value);

bool checkCollisionWithEnemies();

sf::Vector2f getPlayerPosition();

};

Файл Enemy.cpp:

#pragma once

#include "Enemy.h"

#include "Game.h"

#include <ctime>

#define POS\_LEFT 0

#define POS\_UP 2

#define POS\_RIGHT 1

#define POS\_DOWN 3

Imovable::MoveDirection Enemy::newDir(float deltaTime) {

std::vector<MoveDirection> newDir;

sf::Vector2f expVector = position;

int currIndex = 0;

Map& map = Game::getInstance()->getMap();

expVector.x += deltaTime \* speed;

if (map.getCell(expVector) != '1') {

newDir.push\_back(MoveDirection::DOWN);

expVector = position;

}

if (expVector.x >= deltaTime \* speed) {

expVector.x -= deltaTime \* speed;

if (map.getCell(expVector) != '1') {

newDir.push\_back(MoveDirection::UP);

expVector = position;

}

}

expVector.y += deltaTime \* speed;

if (map.getCell(expVector) != '1') {

newDir.push\_back(MoveDirection::LEFT);

expVector = position;

}

if (expVector.y >= deltaTime \* speed) {

expVector.y -= deltaTime \* speed;

if (map.getCell(expVector) != '1') {

newDir.push\_back(MoveDirection::RIGHT);

}

}

return newDir[rand() % newDir.size()];

}

void Enemy::updateDir() {

currDir = newDir(1 / speed);

}

void Enemy::update(float deltaTime) {

Map& map = Game::getInstance()->getMap();

static float hold = 2.f;

bool isDirAvailNow[4];

std::vector<MoveDirection> avail;

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

MoveDirection dir = static\_cast<MoveDirection>(i + 1);

isDirAvailNow[i] = map.getCell(position + getOffset(dir)) == '1';

if (isDirAvailNow[i] == false && isDirAvailable[i] != isDirAvailNow[i])

avail.push\_back(dir);

isDirAvailable[i] = isDirAvailNow[i];

}

hold -= deltaTime;

if (hold <= 0.f && nextDir == MoveDirection::NONE && avail.empty() == false) {

nextDir = avail[rand() % avail.size()];

hold = 2.f;

}

else if (checkCollision(position + (getOffset(currDir) \* speed \* deltaTime))) {

nextDir = newDir(deltaTime);

}

Imovable::update(deltaTime);

}

void Enemy::draw(sf::RenderWindow& window) {

Entity::draw(window);

}

Файл Enemy.h:

#pragma once

#include "IMovable.h"

class Enemy : public Imovable {

private:

bool isDirAvailable[4] = { true };

public:

Enemy(sf::Sprite enemySprite) {

entitySprite = enemySprite;

}

void update(float deltaTime) override;

void draw(sf::RenderWindow& window) override;

void updateDir();

MoveDirection newDir(float deltaTime);

};

Файл Imovable.cpp:

#include "IMovable.h"

#include "Game.h"

bool isCloseEnoughToHalf(float value) {

return std::abs((value - (unsigned int)value) - 0.5f) < 0.05f;

}

sf::Vector2f Imovable::getOffset(MoveDirection dir) {

switch (dir) {

case(MoveDirection::LEFT):

return { -1, 0 };

break;

case(MoveDirection::RIGHT):

return { 1, 0 };

break;

case(MoveDirection::UP):

return { 0, -1 };

break;

case(MoveDirection::DOWN):

return { 0, 1 };

break;

default:

return { 0, 0 };

break;

}

}

void Imovable::update(float deltaTime) {

static constexpr const float timerDefault = 0.3f;

static float timer = timerDefault;

if (nextDir != MoveDirection::NONE && timer > 0) {

timer -= deltaTime;

sf::Vector2f offset = getOffset(nextDir) \* speed \* deltaTime;

sf::Vector2f tempPos = position;

if (isCloseEnoughToHalf(tempPos.x))

tempPos.x = (unsigned int)position.x + 0.5f;

if (isCloseEnoughToHalf(tempPos.y))

tempPos.y = (unsigned int)position.y + 0.5f;

if (checkCollision(tempPos + offset) == false) {

currDir = nextDir;

nextDir = MoveDirection::NONE;

}

}

else {

timer = timerDefault;

nextDir = MoveDirection::NONE;

}

sf::Vector2f offset = getOffset(currDir) \* speed \* deltaTime;

sf::Vector2f expectedPosion = position + offset;

if (checkCollision(expectedPosion)) {

position.x = (unsigned int)position.x + 0.5f;

position.y = (unsigned int)position.y + 0.5f;

}

else {

position = expectedPosion;

}

}

float roundToUpper(float value){

float fraction = value - (int)value;

return fraction == 0 ? value : (unsigned int)(value + 1);

}

void Imovable::setDir(MoveDirection dir) {

nextDir = dir;

}

bool Imovable::checkCollision(sf::Vector2f position) {

Map& map = Game::getInstance()->getMap();

sf::Vector2f pointLeftUp, pointRightDown, pointLeftDown, pointRightUp;

pointLeftUp.x = position.x - size.x / 2;

pointLeftUp.y = position.y - size.y / 2;

if (map.getCell(pointLeftUp) == '1')

return true;

pointRightDown.x = roundToUpper(position.x + size.x / 2) - 1;

pointRightDown.y = roundToUpper(position.y + size.y / 2) - 1;

if (map.getCell(pointRightDown) == '1')

return true;

pointLeftDown.x = pointLeftUp.x;

pointLeftDown.y = pointRightDown.y;

if (map.getCell(pointLeftDown) == '1')

return true;

pointRightUp.x = pointRightDown.x;

pointRightUp.y = pointLeftUp.y;

if (map.getCell(pointRightUp) == '1')

return true;

return false;

}

Файл Imovable.h:

#pragma once

#include "Entity.h"

#include "Map.h"

class Imovable : public Entity {

protected:

enum class MoveDirection {

NONE,

LEFT,

RIGHT,

UP,

DOWN

};

MoveDirection currDir = MoveDirection::NONE;

MoveDirection nextDir = MoveDirection::NONE;

float speed = 4.f;

sf::Vector2f size = { 1, 1 };

public:

void update(float deltaTime) override;

bool checkCollision(sf::Vector2f position);

static sf::Vector2f getOffset(MoveDirection dir);

void setDir(MoveDirection dir);

};

Файл EnemyHandler.cpp:

#include "EnemyHandler.h"

#include "Game.h"

#include "Player.h"

EnemyHandler::EnemyHandler(size\_t enemiesCount, std::vector<sf::Vector2f>& enemiesPositions, sf::Sprite enemySprite) {

Scene& scene = Game::getInstance()->getScene();

for (int i = 0; i < enemiesCount; ++i) {

Enemy\* enemy = reinterpret\_cast<Enemy\*>(scene.addEntity(std::make\_unique<Enemy>(enemySprite)));

enemy->setPosition(enemiesPositions[i]);

enemy->updateDir();

enemies.push\_back(enemy);

}

}

void EnemyHandler::update(float deltaTime) {

}

bool EnemyHandler::checkCollisionWithPlayer(Player\* player)

Map map = Game::getInstance()->getMap();

std::vector<sf::Vector2f> enPos = map.getEnemiesPositions();

sf::Vector2f playerPos = player->getPosition();

sf::Rect<float> playerRect(playerPos - sf::Vector2f{ 0.5f,0.5f }, sf::Vector2f{ 1,1 });

for (auto& enemy : enemies) {

sf::Rect<float> enemyRect(enemy->getPosition() - sf::Vector2f{0.5f,0.5f}, sf::Vector2f{1,1});

if (enemyRect.intersects(playerRect)) {

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

enemies[i]->setPosition(enPos[i]);

}

return true;

}

}

return false;

}

Файл EnemyHandler.h:

#pragma once

#include "Entity.h"

#include "Enemy.h"

class Player;

class EnemyHandler : public Entity {

private:

std::vector<Enemy\*> enemies;

public:

EnemyHandler(size\_t enemiesCount, std::vector<sf::Vector2f>& enemiesPositions, sf::Sprite enemySprite);

void update(float deltaTime) override;

bool checkCollisionWithPlayer(Player\* player);

};

Файл Game.cpp:

#pragma once

#include "Game.h"

#include "Player.h"

#include "Map.h"

#include "EnemyHandler.h"

#include <Windows.h>

#include "HoodElements.h"

Game\* Game::instance = nullptr;

void Game::Redraw(sf::RenderWindow& window) {

scene.draw(window);

}

void Game::Start() {

std::ifstream file("lvl1.txt");

loadedMaps.push\_back(file);

file.close();

StartLevel(loadedMaps.back());

}

void Game::StartLevel(Map& map) {

scene = Scene();

sf::Texture& playerIcon = loadTexture("kkk.png");

sf::Sprite playerSprite;

playerSprite.setTexture(playerIcon);

playerSprite.setScale(0.025, 0.025);

playerSprite.setOrigin(playerIcon.getSize().x \* 0.5f, playerIcon.getSize().y \* 0.5f);

scene.setMap(map);

scene.addEntity(std::make\_unique<Player>(playerSprite));

sf::Texture& enemyIcon = loadTexture("jjj.png");

sf::Sprite enemySprite;

enemySprite.setTexture(enemyIcon);

enemySprite.setScale(0.03, 0.03);

enemySprite.setOrigin(enemyIcon.getSize().x \* 0.5f, enemyIcon.getSize().y \* 0.5f);

std::srand(std::time(nullptr));

scene.addEntity(std::make\_unique<EnemyHandler>(4, map.getEnemiesPositions(), enemySprite));

}

void Game::MainLoop() {

sf::Clock deltaTime;

float time = 0;

while (window.isOpen()) {

float timer = 0;

time = deltaTime.restart().asSeconds();

window.clear();

Update(time);

Redraw(window);

window.display();

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) {

window.close();

}

}

}

}

sf::RenderWindow& Game::getWindow() {

return this->window;

}

void Game::Update(float deltaTime) {

scene.update(deltaTime);

}

Map& Game::getMap() {

return scene.getMap();

}

Scene& Game::getScene() {

return scene;

}

Game\* Game::getInstance() {

return instance;

}

sf::Texture& Game::loadTexture(const std::string& path){

loadedTextures.push\_back(sf::Texture());

loadedTextures.back().loadFromFile(path);

return loadedTextures.back();

}

Файл Game.h:

#pragma once

#include <iostream>

#include "Player.h"

#include "Map.h"

#include "Enemy.h"

#include "Scene.h"

#include <list>

class Game{

private:

Scene scene;

sf::RenderWindow window;

std::vector<Map> loadedMaps;

std::list<sf::Texture> loadedTextures;

static Game\* instance;

public:

Game() : window(sf::VideoMode(450, 480), "Pacman") { instance = this; };

void Start();

void StartLevel(Map& map);

void MainLoop();

sf::RenderWindow& getWindow();

void Update(float deltaTime);

void Redraw(sf::RenderWindow& render);

Map& getMap();

Scene& getScene();

static Game\* getInstance();

sf::Texture& loadTexture(const std::string& path);

};

Файл HoodElements.cpp:

#include "HoodElements.h"

void HoodElements::createHood() {

heart.setRadius(10.f);

heart.setFillColor(sf::Color::Red);

}

void HoodElements::drawHood(sf::RenderWindow& window, Player player) {

for (int i = 0; i < player.getHealth(); i++) {

heart.setPosition(i \* 25, 5);

window.draw(heart);

}

sf::Font font;

font.loadFromFile("pixel\_perfect.ttf");

scoreText.setFont(font);

std::string scoreMessage = { "score: " + std::to\_string(player.getScore()) };

scoreText.setString(scoreMessage);

scoreText.setCharacterSize(25);

scoreText.setPosition(305, 5);

window.draw(scoreText);

}

Файл HoodElements.h:

#pragma once

#include "Player.h"

class HoodElements{

sf::CircleShape heart;

sf::Text scoreText;

public:

void createHood();

void drawHood(sf::RenderWindow& window, Player player);

};

Файл main.cpp:

#include <iostream>

#include "Game.h"

int main() {

Game pacman;

pacman.Start();

pacman.MainLoop();

return 0;

}