МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Балтийский федеральный университет имени Иммануила Канта»

ОНК «Институт высоких технологий»

ОТЧЁТ О ПРОХОЖДЕНИИ

УЧЕБНОЙ ТЕХНОЛОГИЧЕСКОЙ (ПРОЕКТНО-ТЕХНОЛОГИЧЕСКОЙ) ПРАКТИКИ

на базе Высшей школы компьютерных наук и искусственного интеллекта

Выполнил Кузнецов Станислав Витальевич

студент очной формы обучения 1 курса

направления подготовки 01.03.02 Прикладная математика и информатика

профиль обучения «Информатика и программирование»

Руководитель практики

Ст. преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тарачков М.В.

г. Калининград 2024 г.

Оглавление

[ВВЕДЕНИЕ 4](#_Toc172065846)

[ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 5](#_Toc172065847)

[Базовые типы данных 5](#_Toc172065848)

[Библиотека SFML 6](#_Toc172065849)

[ГЛАВА 2. ЗАДАНИЕ НА ПРАКТИКУ 7](#_Toc172065850)

[Задача №1 «Знакомство с библиотекой SFML» 7](#_Toc172065851)

[Задача №2 «Создание игрового пространства» 7](#_Toc172065852)

[Задача №3 «Игрок» 7](#_Toc172065853)

[Задача №4 «Постройки» 7](#_Toc172065854)

[Задача №5 «Враги» 8](#_Toc172065855)

[Глава 3. Выполнение заданий на практику 9](#_Toc172065856)

[Решение задачи №1 9](#_Toc172065857)

[Решение задачи №2 9](#_Toc172065858)

[Решение задачи №3 10](#_Toc172065859)

[Решение задачи №4 10](#_Toc172065860)

[Решение задачи №5 11](#_Toc172065861)

[ЗАКЛЮЧЕНИЕ 13](#_Toc172065862)

[СПИСОК ЛИТЕРАТУРЫ 14](#_Toc172065863)

[ПРИЛОЖЕНИЕ 1 16](#_Toc172065864)

[ПРИЛОЖЕНИЕ 2 17](#_Toc172065865)

[ПРИЛОЖЕНИЕ 3 17](#_Toc172065866)

[ПРИЛОЖЕНИЕ 4 17](#_Toc172065867)

[ПРИЛОЖЕНИЕ 5 18](#_Toc172065868)

[ПРИЛОЖЕНИЕ 6 19](#_Toc172065869)

[ПРИЛОЖЕНИЕ 7 19](#_Toc172065870)

[ПРИЛОЖЕНИЕ 8 20](#_Toc172065871)

[ПРИЛОЖЕНИЕ 9 20](#_Toc172065872)

[ПРИЛОЖЕНИЕ 10 21](#_Toc172065873)

[ПРИЛОЖЕНИЕ 11 21](#_Toc172065874)

[ПРИЛОЖЕНИЕ 12 22](#_Toc172065875)

[ПРИЛОЖЕНИЕ 13 23](#_Toc172065876)

[ПРИЛОЖЕНИЕ 14 24](#_Toc172065877)

[ПРИЛОЖЕНИЕ 15 25](#_Toc172065878)

[ПРИЛОЖЕНИЕ 16 26](#_Toc172065879)

[ПРИЛОЖЕНИЕ 17 26](#_Toc172065880)

[ПРИЛОЖЕНИЕ 18 27](#_Toc172065881)

[ПРИЛОЖЕНИЕ 19 27](#_Toc172065882)

[ПРИЛОЖЕНИЕ 20 29](#_Toc172065883)

[ПРИЛОЖЕНИЕ 21 29](#_Toc172065884)

[ПРИЛОЖЕНИЕ 22 29](#_Toc172065885)

[ПРИЛОЖЕНИЕ 23 30](#_Toc172065886)

[ПРИЛОЖЕНИЕ 24 31](#_Toc172065887)

[ПРИЛОЖЕНИЕ 25 32](#_Toc172065888)

[ПРИЛОЖЕНИЕ 26 32](#_Toc172065889)

**ВВЕДЕНИЕ**

Вид практики – Учебная технологическая (проектно-технологическая) практика (далее Учебная практика).

Цель учебной практики: получение первичных профессиональных умений навыков.

Задачи учебной практики:

1. Закрепление и углубление теоретических знаний в области информационных технологий;
2. Приобретение и развитие первичных профессиональных навыков и умений в области прикладной математики и информатики.

**ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

**Базовые типы данных**

Тип переменной – характеристика, определяющая формат представления данных в памяти компьютера, множество допустимых значений этих данных и совокупность операций над ними. Зная тип переменной, компилятор выделит для нее необходимое количество ячеек памяти. Основные типы переменных приведены в таблице 1.

*Таблица 1*

**Типы данных в С++**

|  |  |  |
| --- | --- | --- |
| **Тип** | **Размер**  **(в байтах)** | **Диапазон** |
| unsigned char | 1 | 0 – 255 |
| char | 1 | -128 – 127 |
| unsigned short | 2 | 0 – 65535 |
| short | 2 | -32 768 – 32 767 |
| unsigned int (16 разрядов) | 2 | 0 – 65 535 |
| unsigned int (32 разряда) | 4 | 0 – 4 294 967 295 |
| int (16 разрядов) | 2 | -32 768 – 32 767 |
| int (32 разряда) | 4 | -2 147 483 648 – 2 147 483 647 |
| unsigned long | 4 | 0 – 4294967295 |
| long | 4 | -2 147 483 648 – 2 147 483 647 |
| unsigned long long (С99) | 8 | 0 – 18 446 744 073 709 551 615 |
| long long (С99) | 8 | -9 223 372 036 854 775 808 –  9 223 372 036 854 775 807 |
| float | 4 | 1.2e-38 – 3.4e+38 |
| double | 8 | 2.2e-308 – 1.7e+308 |
| long double | 10 | 1.7e-4932 – 1.7e+4932 |

**Библиотека SFML**

Библиотека SFML обеспечивает простой интерфейс для различных компонентов вашего ПК, упрощая разработку игр и мультимедийных приложений. Он состоит из пяти модулей, приведенных в таблице 2.

*Таблица 2*

**Модули библиотеки SFML**

|  |  |
| --- | --- |
| **Название модуля** | **Особенности** |
| System | Обеспечивает базовые функции и типы данных, такие как время, строки, массивы и т.д. |
| Window | Управляет окнами и их событиями, а также предоставляет доступ к вводу (клавиатура, мышь, джойстики). |
| Graphics | Предоставляет функции для отрисовки 2D-графики, включая текстуры, спрайты, формы, текст и т.д. |
| Audio | Обеспечивает возможности для воспроизведения звуков и музыки. |
| Network | Предоставляет средства для сетевого взаимодействия, такие как TCP и UDP сокеты. |

**ГЛАВА 2. ЗАДАНИЕ НА ПРАКТИКУ**

**Задача №1 «Знакомство с библиотекой SFML»**

Изучите библиотеку SFML и создайте окно приложения. Научитесь обрабатывать события, такие как нажатие кнопок клавиатуры и мыши, а также отображать простые графические примитивы (круги и прямоугольники) на экране. Добавьте возможность перемещения с помощью мыши или клавиатуры.

**Задача №2 «Создание игрового пространства»**

Реализуйте класс, который будет случайным образом генерировать игровое пространство. Убедитесь, что пространство включает различные элементы, такие как препятствия и открытые области, для создания разнообразной среды.

**Задача №3 «Игрок»**

Реализуйте в игре игрока, который может перемещаться по игровому пространству. Обеспечьте управление игроком с использованием клавиатуры и мыши, и добавьте отображение спрайта игрока на экране, а также добавьте физику для персонажа, чтобы он не мог ходить сквозь препятствий.

**Задача №4 «Постройки»**

Реализуйте в игре систему добычи ресурсов, а также систему постройки зданий. Убедитесь, что игрок может собирать ресурсы из окружающей среды и использовать их для создания различных типов построек.

**Задача №5 «Враги»**

Добавьте в игру врагов, которые будут взаимодействовать с игроком и игровым пространством. Реализуйте базовое поведение врагов, такое как передвижение и атака, а также взаимодействие с игроком (например, снижение здоровья при столкновении).

**Глава 3. Выполнение заданий на практику**

**Решение задачи №1**

Был создан класс Engine, который создает окно приложения, обрабатывает события, а также отрисовывает 2Д графику. Он содержит методы CrankUp() для запуска бесконечного цикла, в котором выполняется вся логика игры; IsPlaying() для определения запущена ли игра; UpdateView() обновляет размеры и положение камеры. Заголовок класса расположен в приложении №1.

Кроме того, были созданы классы IRenderable (Класс, содержащий данные для отрисовки), IStartable (Класс, содержащий виртуальный метод, который вызывается единожды, при вызове метода CrankUp() ) и IUpdateable (Класс, содержащий виртуальный метод, который вызывается каждый кадр). Заголовки классов расположены в приложении №2-4 соответственно.

Также были созданы классы Input, который хранит информацию об клавиатуре и мыши, и Camera, используется для перемещения камеры по игровому пространству. Заголовки классов расположены в приложении №5-6 соответственно.

**Решение задачи №2**

Для реализации этой задачи были созданы следующие классы:

* Класс Vector2, для работы с векторами в двухмерном пространстве.
* Класс Noise, который использует Vector2 и создает массив псевдослучайных чисел от 0 до 1, используя Шум Перлина.
* Класс GameObject, который используется при создании игровых объектов, таких как деревья, месторождения, камни, постройки и т.д.
* Класс Tile, который содержит перечисление типов «почвы» (вода, грязь, песок и т.д.), а также содержит методы: MoveSpeed(…) возвращает скорость передвижения по данному типу почвы; Image(...) возвращает sf::Image, используем для отображения спрайтов.
* Класс IHittable, который используется для создания разрушаемых объектов.
* Класс MapGenerator, который на основе данных от класса Noise создает псевдослучайное игровое пространство Также этот класс реализует метод getPlayerCoordsSpawn(), который возвращает координаты свободного места для появления игрока.
* Класс Collision, который обрабатывает все столкновения игровых объектов.

Заголовки классов расположены в приложении №7-13 соответственно.

**Решение задачи №3**

Для реализации этой задачи были созданы классы:

* Класс Singleton, содержащий ссылки на классы MapGenerator и Player. Используется для быстрого доступа к данным классам.
* Класс Player, который реализует методы: ChangeTool() обрабатывает смену текущего инструмента в руках; Attack() обрабатывает атаку игрока; Move() обрабатывает передвижение игрока; UI() обрабатывает отображение пользовательского интерфейса.

Заголовки классов расположены в приложении №14-15 соответственно.

**Решение задачи №4**

Для реализации этой задачи были созданы классы:

* Класс Resources, который содержит перечисление всех видов добываемых ресурсов, а также содержит методы: MoveModification(…) возвращает скорость передвижения по этому ресурсу; Image(…) возвращает sf::Image, используем для отображения спрайтов.
* Класс IMineable, который используется для создания добываемых объектов.
* Класс Deposit, который наследуется от классов GameObject, IMineable и используется при создании деревьев, месторождений и камней в классе MapGenerator.
* Класс Building, который содержит все виды построек; кол-во ресурсов, необходимых для их строительства; текстуры, используемые для создания спрайтов построек. Также реализует методы Build(…) для постройки здания и Destroy(…) для уничтожения здания.
* Класс Wall, который используется для создания и взаимодействия со стеной.
* Класс WorkStation, который используется для создания и взаимодействия со зданиями, которые добывают ресурсы.
* Класс Archer, который используется для создания лучника

Заголовки классов расположены в приложении №16-22 соответственно.

**Решение задачи №5**

Для реализации этой задачи были созданы следующие классы:

* Класс PathFinding, который реализует метод CalculatePath(…). Этот метод возвращает путь от точки a до точки b.
* Класс Enemy, который используется для создания врагов и реализует их поведение.
* Класс Waves, который создает новую волну врагов каждые n секунд.

При выполнении этой задачи, я столкнулся с долгим выполнением метода CalculatePath (Примерно 0.5 секунды при нахождении пути из точки (0,0) до точки (200,200)). Так как я ожидал, что врагов будет много, такой алгоритм занял бы много времени в однопоточном приложении. Поэтому я решил использовать библиотеки thread, mutex и condition\_variable для создания класса ThreadPool, который реализует методы enqueue(…) для добавления задачи в очередь.

Заголовки классов расположены в приложении №23-26 соответственно.

**ЗАКЛЮЧЕНИЕ**

В ходе практики были изучены основы языка программирования С++. Задачи были направлены на закрепление теоретического материала по типам данных, арифметическим, логическим операторам. Были изучены условия и циклы. Особое внимание уделялось работе с библиотекой SFML.

В результате практики были усовершенствованы мои компетенции, закреплены теоретические навыки.

В ходе учебной практики я научился: выполнять поставленную задачу, следуя условию, изменять готовое решения по мере нахождения ошибок, искать альтернативные пути решения задач.

Также я освоил новую для меня среду программирования Visual Studio и работу с GitHub (создание репозиториев для выгрузки решений задач). Я научился использовать функции и циклы, обрабатывать строки, изучил библиотеки SFML, iostream (для ввода и вывода данных), vector (для работы с динамическим массивом), string (для работы со строками), thread (для работы с потоками), list и forward\_list, (для работы со списком), functional (для работы с функциями), queue (для работы с очередями), unordered\_set (для работы с контейнером, содержащий уникальные элементы), map (для работы с контейнером типа «ключ-значение»), cmath (для работы с математическими операторами), ctime (для работы со временем), выполнил работу с разными видами массивов и приемами работы с ними.

По мере прохождения учебно-технологической практики я выполнил пять задач, в которых использовал полученные знания и закрепил навыки работы с C++.

В течение практики задачи были выполнены, а цели достигнуты.

# СПИСОК ЛИТЕРАТУРЫ

**Перечень учебной литературы ресурсов сети «Интернет», необходимой для проведения практики**

1. Варфоломеева, Т. Н. Структуры данных и основные алгоритмы их обработки : учебное пособие / Т. Н. Варфоломеева. - Москва : ФЛИНТА, 2017. - 159 с. - ISBN 978-5-9765-3691-3. - Текст : электронный. - URL: https://znanium.com/catalog/product/1860018 (дата обращения: 19.01.2023). – Режим доступа: по подписке.
2. Гданский, Н. И. Основы теории и алгоритмы на графах : учебное пособие / Н. И. Гданский. — Москва : ИНФРА-М, 2020. — 206 с. — (Высшее образование: Бакалавриат). - ISBN 978-5-16-014386-6. - Текст : электронный. - URL: https://znanium.com/catalog/product/978686 (дата обращения: 19.01.2023). – Режим доступа: по подписке.
3. Затонский, А. В. Программирование и основы алгоритмизации. Теоретические основы и примеры реализации численных методов: учебное пособие / А.В. Затонский, Н.В. Бильфельд. — 2-е изд. — Москва: РИОР : ИНФРА-М, 2022. — 167 с. — (Высшее образование). — DOI: https: //www.dx.doi.org/10.12737/20468. - ISBN 978-5-369-01195-9. - Текст: электронный. - URL: https://znanium.com/catalog/product/1860435 (дата обращения: 16.02.2023). – Режим доступа: по подписке.
4. Воронцова, Е. А. Программирование на С++ с погружением: практические задания и примеры кода - Москва: НИЦ ИНФРА-М, 2016. - 80 с. ISBN 978-5-16-105159-7. - Текст: электронный. - URL: https://znanium.com/catalog/product/563294 (дата обращения: 16.02.2023). – Режим доступа: по подписке.

**Перечень ресурсов информационно-телекоммуникационной сети «Интернет»**

1. Колдаев, В. Д. Структуры и алгоритмы обработки данных [Электронный ресурс]: учеб. пособие для вузов/ В. Д. Колдаев. - Москва: РИОР; Москва: ИНФРА-М, 2014. - 1 эл. опт. диск (CD-ROM), 294 с.: ил., табл. - (Высшее образование - бакалавриат). - Библиогр.: с. 285. - Лицензия до 23.06.2020 г. - ISBN 978-5-369-01264-2. - ISBN 978-5-16-009012-2: 15100.00 р. Имеются экземпляры в отделах /There are copies in departments: всего /all 2: ЭБС Кантиана(1), ч.з.N1(1) Свободны / free: ЭБС Кантиана(1), ч.з.N1(1)
2. Основы программирования. – режим доступа: http://www.intuit.ru/studies/courses/648/504/info
3. Видеолекции по курсу «Основы программирования». – режим доступа: http://www.youtube.com/watch?v=pxR3UoO9c9w
4. Сайт «Структуры и алгоритмы». – режим доступа: http://www.structur.h1.ru/
5. Структуры и алгоритмы обработки данных: Учебное пособие. – режим доступа: http://window.edu.ru/resource/820/44820
6. Документация по различным библиотекам C++ - режим доступа: https://en.cppreference.com/w/
7. Алгоритм A\* - режим доступа: https://ru.wikipedia.org/wiki/A\*
8. Шум Перлина – режим доступа: https://en.wikipedia.org/wiki/Perlin\_noise
9. Многопоточность – режим доступа: https://habr.com/ru/companies/otus/articles/549814/
10. Документация по библиотеке SFML – режим доступа: https://www.sfml-dev.org/tutorials/2.6/

## ПРИЛОЖЕНИЕ 1

#pragma once

#include <SFML/Graphics.hpp>

#include <SFML/Audio.hpp>

#include <cmath>

#include <ctime>

#include <iostream>

#include <list>

#include <Input.hpp>

#include <Vector2.hpp>

#include <IUpdateable.hpp>

#include <IStartable.hpp>

#include <IRenderable.hpp>

#include <GameObject.hpp>

#include <MapGenerator.hpp>

#include <EventHolder.hpp>

#include <ThreadPool.hpp>

#include <Collision.hpp>

class Engine

{

bool isPlaying = false;

public:

static inline Engine\* instance{nullptr};

float deltaTime;

int m\_gameWidth;

int m\_gameHeight;

my::ThreadPool\* threadPool;

Collision\* m\_collision;

Input\* m\_input;

sf::View\* m\_view;

sf::RenderWindow\* m\_window;

std::list<IStartable\*> m\_start;

std::list<IUpdateable\*> m\_update;

std::list<ILateUpdateable\*> m\_lateUpdate;

std::list<IRenderable\*> m\_render;

std::list<sf::Drawable\*> m\_manualRender;

std::list<BaseEventHolder\*> m\_events;

std::list<sf::Texture\*> m\_textures;

Engine(int gameWidth, int gameHeight, sf::Vector2i gridSizeCollision, sf::Vector2i mapSize);

Engine(const Engine& engine) = delete;

Engine& operator=(const Engine& other) = delete;

~Engine();

template<typename T>

static void ClearListWithAutoRemove(std::list<T>& lst)

{

while (!lst.empty())

{

delete (\*(lst.begin()));

}

lst.clear();

}

template<typename T>

static void ClearList(std::list<T>& lst)

{

for (auto iter = lst.begin(); iter != lst.end(); ++iter)

{

delete (\*iter);

}

lst.clear();

}

void CrankUp();

bool IsPlaying();

void UpdateView();

};

## ПРИЛОЖЕНИЕ 2

#pragma once

#include <SFML/Graphics.hpp>

struct IRenderable

{

sf::Sprite\* render;

IRenderable(sf::Texture\* t = nullptr);

virtual ~IRenderable();

void ClearRender();

};

## ПРИЛОЖЕНИЕ 3

#pragma once

struct IStartable

{

IStartable();

virtual ~IStartable();

virtual void Start() = 0;

};

## ПРИЛОЖЕНИЕ 4

#pragma once

struct IUpdateable

{

IUpdateable();

virtual ~IUpdateable();

virtual void Update() = 0;

};

struct ILateUpdateable

{

ILateUpdateable();

virtual ~ILateUpdateable();

virtual void LateUpdate() = 0;

};

## ПРИЛОЖЕНИЕ 5

#pragma once

#include <map>

#include <SFML/Graphics.hpp>

class Input

{

private:

friend class Engine;

std::map<sf::Keyboard::Key, bool> keyboardPressed;

std::map<sf::Keyboard::Key, bool> keyboardReleased;

std::map<sf::Keyboard::Key, bool> keyboardHold;

std::map<sf::Mouse::Button, bool> mousePressed;

std::map<sf::Mouse::Button, bool> mouseReleased;

std::map<sf::Mouse::Button, bool> mouseHold;

sf::Vector2f mousePosInWindow;

sf::Vector2f mousePosInWorld;

float mouseWheelDelta;

public:

Input();

Input(const Input& other) = delete;

Input& operator=(const Input& other) = delete;

bool isPressed(sf::Keyboard::Key key);

bool isPressed(sf::Mouse::Button key);

bool isReleased(sf::Keyboard::Key key);

bool isReleased(sf::Mouse::Button key);

bool isHold(sf::Keyboard::Key key);

bool isHold(sf::Mouse::Button key);

bool isHasFocus();

sf::Vector2f getMousePosInWindow();

sf::Vector2f getMousePosInWorld();

float getWheelDelta();

private:

void Reset();

void Check(sf::Event& e);

};

#include <Engine.hpp>

## ПРИЛОЖЕНИЕ 6

#pragma once

#include <SFML/Graphics.hpp>

class Camera

{

public:

void SetSize(sf::Vector2f size);

void SetPosition(sf::Vector2f pos);

void Move(sf::Vector2f vector);

void Move(float offsetX, float offsetY);

};

## ПРИЛОЖЕНИЕ 7

#pragma once

#include <cmath>

struct Vector2

{

float x;

float y;

Vector2() = default;

Vector2(float x, float y);

Vector2& operator+(const Vector2& other);

Vector2& operator+=(const Vector2& other);

Vector2& operator-(const Vector2& other);

Vector2& operator-=(const Vector2& other);

Vector2& operator/(float num);

Vector2& operator/=(float num);

Vector2& operator\*(float num);

Vector2& operator\*=(float num);

Vector2 Normalize();

static float Dot(Vector2 a, Vector2 b);

};

struct Vector2I

{

int x;

int y;

Vector2I() = default;

Vector2I(int x, int y);

Vector2I& operator-(const Vector2I& other);

Vector2I& operator-=(const Vector2I& other);

Vector2I& operator/(int num);

Vector2I& operator/=(int num);

bool operator==(Vector2I& other);

bool operator!=(Vector2I& other);

Vector2I& Normalize();

static float Dot(Vector2I a, Vector2I b);

};

## ПРИЛОЖЕНИЕ 8

#pragma once

#include <Vector2.hpp>

#include <cstdint>

namespace my

{

struct Noise\_Output;

class Perlin\_Noise

{

public:

/// <summary>

/// Шум Перлина

/// </summary>

/// <param name="seed">Случайное значение, которое будет использоваться для генерации данных</param>

/// <param name="gridSize">Размер сетки</param>

/// <param name="texSize">Размер выходных данных</param>

/// <returns>Возращяет массив размером [dataSize.x, dataSize.y]</returns>

static Noise\_Output\* GetNoise(int seed, Vector2I gridSize, Vector2I dataSize);

};

struct Noise\_Output

{

private:

bool m\_isMain;

Vector2I m\_size;

float\*\* m\_data;

public:

Noise\_Output() = default;

Noise\_Output(Vector2I size);

Noise\_Output(const Noise\_Output& other);

Noise\_Output& operator=(const Noise\_Output& other);

~Noise\_Output();

float\* operator[](std::size\_t idx);

const Vector2I& getSize();

friend static Noise\_Output\* Perlin\_Noise::GetNoise(int, Vector2I, Vector2I);

};

}

## ПРИЛОЖЕНИЕ 9

#pragma once

#include <IUpdateable.hpp>

#include <IStartable.hpp>

#include <IRenderable.hpp>

#include <Vector2.hpp>

class GameObject : public IUpdateable, public IStartable, public IRenderable

{

friend class Engine;

public:

GameObject();

GameObject(const sf::Vector2f position, sf::Texture\* texture);

GameObject(const GameObject& other);

GameObject& operator=(const GameObject& other);

void Update() override {}

void Start() override {}

~GameObject() override = default;

};

## ПРИЛОЖЕНИЕ 10

#pragma once

#include <SFML/Graphics.hpp>

struct Tile

{

enum class Type : char

{

water,

soil,

grass,

sand,

road,

rock,

rocky\_surface,

snow,

};

Type type;

float speed;

Tile(Type type);

static float MoveSpeed(Type type);

static sf::Image\* Image(Type type);

friend class MapGenerator;

private:

void resetSpeed();

};

## ПРИЛОЖЕНИЕ 11

#pragma once

class IHittable

{

protected:

int m\_health;

public:

IHittable(int health) : m\_health{ health } {}

virtual ~IHittable() = default;

//true - is dead

virtual bool Hit(int damage) = 0;

};

## ПРИЛОЖЕНИЕ 12

#pragma once

#include <Vector2.hpp>

#include <Tile.hpp>

#include <Noise.hpp>

#include <list>

#include <GameObject.hpp>

class MapGenerator

{

int m\_seed;

my::Noise\_Output\* m\_dataHeight;

my::Noise\_Output\* m\_dataTemp;

my::Noise\_Output\* m\_dataDeposit;

Vector2I m\_size;

std::vector<Tile\*> m\_ground;

std::vector<GameObject\*> m\_objects;

GameObject\* m\_groundObject;

public:

const int m\_texturesSize = 16;

MapGenerator() = default;

MapGenerator(Vector2I size, int textureSize);

MapGenerator(const MapGenerator& other) = delete;

MapGenerator& operator=(const MapGenerator& other) = delete;

~MapGenerator();

my::Noise\_Output& getDataHeight() { return \*m\_dataHeight; }

my::Noise\_Output& getDataTemp() { return \*m\_dataTemp; }

my::Noise\_Output& getDataDeposit() { return \*m\_dataDeposit; }

Tile\* const getTile(int x, int y);

Tile\* const getTile(sf::Vector2i pos);

GameObject\*& getObj(int x, int y);

GameObject\*& getObj(sf::Vector2i pos);

bool IsBuilding(int x, int y);

const Vector2I& getSize();

sf::Vector2f GridCoordsToCenterCoords(int x, int y);

sf::Vector2f GridCoordsToCoords(int x, int y);

sf::Vector2f CoordsToGridCoords(sf::Vector2f pos);

sf::Vector2f getPlayerCoordsSpawn();

};

## ПРИЛОЖЕНИЕ 13

#pragma once

#include <SFML/Graphics.hpp>

#include <list>

#include <vector>

#include <GameObject.hpp>

#define DEBUG\_COLLISION

class Collision

{

static inline Collision\* instance{ nullptr };

public:

struct BaseCollider;

struct BoxCollider;

struct CircleCollider;

private:

sf::Vector2i gridSize;

sf::Vector2i mapSize;

std::list<BaseCollider\*>\* grid;

std::list<BaseCollider\*>& getList(sf::Vector2i gridPos);

void checkCollisions(BoxCollider\* collider, std::list<BaseCollider\*>& where\_check, std::list<BaseCollider\*>& saveTo);

void checkCollisions(CircleCollider\* collider, std::list<BaseCollider\*>& where\_check, std::list<BaseCollider\*>& saveTo);

public:

Collision() = default;

Collision(sf::Vector2i gridSize, sf::Vector2i mapSize);

void UpdateCollider(BoxCollider\* collider);

void UpdateCollider(CircleCollider\* collider);

void RemoveCollider(BoxCollider\* collider);

void RemoveCollider(CircleCollider\* collider);

std::list<BaseCollider\*> getCollisions(BoxCollider\* collider);

std::list<BaseCollider\*> getCollisions(CircleCollider\* collider);

bool CheckCollision(const CircleCollider\* circle, const BoxCollider\* box);

bool CheckCollision(const BoxCollider\* box1, const BoxCollider\* box2);

bool CheckCollision(const CircleCollider\* circle1, const CircleCollider\* circle2);

bool CheckCollision(const CircleCollider\* circle, BaseCollider\* collider);

bool CheckCollision(const BoxCollider\* box, BaseCollider\* collider);

bool CheckCollision(BaseCollider\* collider1, BaseCollider\* collider2);

struct BaseCollider

{

GameObject\* object;

sf::Vector2f center;

bool isTrigger;

BaseCollider() : object{ nullptr }, center{}, isTrigger{ false } {}

virtual ~BaseCollider() = default;

BaseCollider(GameObject\* obj, sf::Vector2f center, bool isTrigger = false);

};

struct BoxCollider : public BaseCollider

{

sf::Vector2f border;

float angle;

BoxCollider() = default;

BoxCollider(GameObject\* obj, sf::Vector2f center, sf::Vector2f border, bool isTempCollider = false);

void Update(sf::Vector2f center, float angle);

private:

sf::Vector2f actualBorder;

friend class Collision;

};

struct CircleCollider : public BaseCollider

{

float radius;

#ifdef DEBUG\_COLLISION

sf::CircleShape\* c;

#endif // DEBUG\_COLLISION

CircleCollider() = default;

~CircleCollider() override;

CircleCollider(GameObject\* obj, sf::Vector2f center, float radius, bool isTrigger = false, bool isTempCollider = false);

void Update(sf::Vector2f center);

};

};

## ПРИЛОЖЕНИЕ 14

#pragma once

#include <MapGenerator.hpp>

#include <Player.hpp>

struct Singleton

{

static inline Singleton\* instance{ nullptr };

MapGenerator\* map;

Player\* player;

Singleton() : map{nullptr}, player{nullptr}

{

if (instance != nullptr)

{

std::cerr << "Singleton already exist!" << std::endl;

throw new std::runtime\_error("Singleton already exist!");

}

instance = this;

}

};

## ПРИЛОЖЕНИЕ 15

#pragma once

#include <GameObject.hpp>

#include <Camera.hpp>

#include <Input.hpp>

#include <Resources.hpp>

#include <Collision.hpp>

#include <IHittable.hpp>

#include <Keep.hpp>

#include <unordered\_set>

#include <Building.hpp>

class Player : public GameObject, public IHittable

{

float speed;

float sprintMultiple;

Camera m\_camera;

Input\* input;

BuildSystem m\_buildSystem;

BuildSystem::Buildings typeOfBuilding = BuildSystem::Buildings::wall\_1;

bool isBuilding;

GameObject sword;

GameObject pickaxe;

GameObject axe;

GameObject\* currentToolObj;

Collision::CircleCollider playerCollider;

Collision::CircleCollider attackTrigger;

sf::Vector2f defaultToolPos;

//UI

sf::Text\* resourcesUI;

sf::RectangleShape\* ui\_w\_1;

sf::RectangleShape\* ui\_w\_2;

sf::RectangleShape\* ui\_w\_3;

sf::RectangleShape\* ui\_keep;

sf::RectangleShape\* ui\_cave;

sf::RectangleShape\* ui\_lumber;

sf::RectangleShape\* ui\_archerBarrack;

sf::RectangleShape\* ui\_archer;

//Attack settings

const float attackDur;

const float attackRadius;

//Attack variables

bool isAttack;

float attackTimer;

float attackAngle;

std::unordered\_set<GameObject\*> attackObj;

public:

Resources resource;

Player(float speed, float sprintMultiple);

Player(const Player& other) = delete;

Player& operator=(const Player& other) = delete;

~Player() override;

void ChangeTool();

void Attack();

void Move();

void UI();

bool Hit(int damage) override;

void Update() override;

void Start() override;

const BuildSystem& getBuildSystem();

};

## ПРИЛОЖЕНИЕ 16

#pragma once

#include <map>

#include <Engine.hpp>

class Resources

{

public:

enum class resource

{

wood,

iron,

gold,

rock,

archer,

nothing,

};

std::map<resource, int> resources;

Resources(int wood = 0, int iron = 0, int gold = 0, int rock = 0, int archer = 0);

bool operator<=(Resources& other);

Resources& operator-=(Resources& other);

static sf::Image\* Image(resource type);

static float MoveModification(resource type);

};

## ПРИЛОЖЕНИЕ 17

#pragma once

#include <Resources.hpp>

class IMineable

{

protected:

const int m\_maxHealth;

const int m\_maxResource;

int m\_resourceCount;

int m\_health;

public:

const Resources::resource type;

IMineable(int resourceCount, int health, Resources::resource type);

virtual ~IMineable() = default;

virtual int Mine(int damage) = 0;

};

## ПРИЛОЖЕНИЕ 18

#pragma once

#include <GameObject.hpp>

#include <IMineable.hpp>

#include <Collision.hpp>

class Deposit : public GameObject, public IMineable

{

Collision::CircleCollider collider;

bool isDestroyed;

public:

Deposit(const sf::Vector2f position, sf::Texture\* texture, int health, int resourceCount, Resources::resource typeResource);

~Deposit() override;

int Mine(int damage) override;

bool IsDestroyed();

};

## ПРИЛОЖЕНИЕ 19

#pragma once

#include <SFML/Graphics.hpp>

#include <GameObject.hpp>

#include <IHittable.hpp>

#include <Resources.hpp>

#include <unordered\_map>

#include <Collision.hpp>

#include <Keep.hpp>

class BuildSystem

{

sf::Image resources;

sf::Image wall\_1;

sf::Image wall\_2;

sf::Image wall\_3;

sf::Image keep;

sf::Image barracks;

sf::Image archers;

GameObject\* getGameObjectInGrid(sf::Vector2f pos);

GameObject\* getGameObjectInGrid(int x, int y);

public:

enum class Buildings

{

wall\_1,

wall\_2,

wall\_3,

cave,

lumber,

archerBarrack,

archer,

//swordsmanBarrack

keep\_1,

keep\_2,

keep\_3,

nothing,

};

std::unordered\_map<Buildings, Resources> resourceRequire =

{

{Buildings::wall\_1, Resources(3)},

{Buildings::wall\_2, Resources(5,0,0,3)},

{Buildings::wall\_3, Resources(5,0,0,6)},

{Buildings::keep\_1, Resources(20)},

{Buildings::keep\_2, Resources(50, 10, 15, 30)},

{Buildings::keep\_3, Resources(100, 50, 100, 100)},

{Buildings::cave, Resources(15)},

{Buildings::lumber, Resources(10)},

{Buildings::archerBarrack, Resources(30, 15, 80, 15)},

{Buildings::archer, Resources(0, 0, 0, 0, 1)}

//{Buildings::swordsmanBarrack, Resources(30, 20, 50, 15)}

};

int buildLevel;

GameObject\* building;

GameObject\* keepObj;

sf::Vector2f prevPos;

sf::Texture wall\_1\_t;

sf::Texture wall\_1\_v;

sf::Texture wall\_1\_h;

sf::Texture wall\_2\_t;

sf::Texture wall\_2\_v;

sf::Texture wall\_2\_h;

sf::Texture wall\_3\_t;

sf::Texture wall\_3\_v;

sf::Texture wall\_3\_h;

sf::Texture keep\_1;

sf::Texture keep\_2;

sf::Texture keep\_3;

sf::Texture cave;

sf::Texture lumber;

sf::Texture archerBarrack;

sf::Texture archer;

//sf::Texture swordsmanBarrack;

BuildSystem();

bool CheckBuild(sf::Vector2f pos, Buildings type, GameObject\*& building);

void PlaceCurrentBuilding(Buildings type);

void Build(sf::Vector2f pos, Buildings type, bool isBuild);

void Destroy(sf::Vector2f pos);

void Upgrade(sf::Vector2f pos);

};

## ПРИЛОЖЕНИЕ 20

class Wall : public GameObject, public IHittable

{

Archer\* archer;

Collision::CircleCollider collider;

friend class BuildSystem;

public:

Wall(const sf::Vector2f position, sf::Texture\* texture, int health);

~Wall() override;

bool PlaceArcher(sf::Texture& texture);

bool Hit(int damage) override;

};

## ПРИЛОЖЕНИЕ 21

class WorkStation : public GameObject, public IHittable

{

Resources::resource m\_typeOfResource;

Collision::CircleCollider gatherCollider;

Collision::CircleCollider collider;

int m\_resourcesPer15sec;

float m\_timer;

bool isWorking;

friend class BuildSystem;

public:

WorkStation(const sf::Vector2f position, sf::Texture\* texture, int health);

WorkStation(const sf::Vector2f position, sf::Texture\* texture, int health, Resources::resource typeOfResource, int resourcesPer15sec);

~WorkStation() override = default;

void Update() override;

bool Hit(int damage) override;

};

## ПРИЛОЖЕНИЕ 22

class Archer : public GameObject, public IHittable

{

Collision::CircleCollider collider;

Collision::CircleCollider attackTrigger;

friend class Wall;

friend class BuildSystem;

public:

Archer(const sf::Vector2f position, sf::Texture\* texture, int health);

~Archer() override = default;

bool Hit(int damage) override;

};

## ПРИЛОЖЕНИЕ 23

#pragma once

#include <Vector2.hpp>

#include <MapGenerator.hpp>

#include <Singleton.hpp>

#include <unordered\_set>

namespace my

{

namespace pathFinding

{

struct Node

{

Vector2I pos; //Координаты

int g; //Расстояние от начала до этой точки

int h; //Расстояние от конца до этой точки

int f; //Сумма g+h

Node\* parent; //Родительский узел

Node() = default;

Node(int x, int y);

void calculate();

bool operator >(const Node& other) const;

bool operator==(const Node& other) const;

};

//Объявляем свои функцию, по которой будет считаться хэш и сравниваться указатели на Node

struct NodePtrHash

{

size\_t operator()(const my::pathFinding::Node\* f) const;

};

struct NodePtrEqual

{

bool operator()(const my::pathFinding::Node\* lhs, const my::pathFinding::Node\* rhs) const;

};

class PathFinding

{

float distance(Vector2I a, Vector2I b);

public:

std::unordered\_set<Node\*, NodePtrHash, NodePtrEqual> checked;

std::unordered\_set<Node\*, NodePtrHash, NodePtrEqual> awaits;

PathFinding() = default;

void Clear();

~PathFinding();

Node\* CalculatePath(Vector2I a, Vector2I b, float mul);

};

}

}

## ПРИЛОЖЕНИЕ 24

#pragma once

#include <IHittable.hpp>

#include <GameObject.hpp>

#include <PathFinding.hpp>

#include <mutex>

#include <Collision.hpp>

#include <forward\_list>

//#define DEBUG\_PATHFINDING

class Enemy : public GameObject, public IHittable, public ILateUpdateable

{

float moveTimer;

float attackTimer;

//1 = одна клетка в секунду при условии, что скорость данной клетки равна 1

float speed;

float attackCooldown;

int damage;

bool isDead;

bool isCalculatingPath;

my::pathFinding::PathFinding finder;

sf::Vector2f playerPos;

sf::Vector2f currentPos;

sf::Vector2f startPos;

sf::Vector2i currentPosTarget;

std::forward\_list<sf::Vector2i> path;

std::mutex m;

Collision::CircleCollider collider;

Collision::CircleCollider attackTrigger;

#ifdef DEBUG\_PATHFINDING

std::list<sf::Drawable\*> tempList;

#endif // DEBUG\_PATHFINDING

public:

Enemy(const sf::Vector2f position, sf::Texture\* texture, int health, int damage, float moveSpeed, float attackCooldown);

~Enemy() override = default;

void Update() override;

void Start() override;

void LateUpdate() override;

bool Hit(int damage) override;

void Attack();

void Attack(int x, int y);

};

## ПРИЛОЖЕНИЕ 25

#pragma once

#include <IUpdateable.hpp>

#include <SFML/Graphics.hpp>

class Waves : IUpdateable

{

float timer;

const float multiplyEnemys = 1.2f;

int currentCount;

sf::Texture\* enemyTexture;

public:

Waves();

~Waves() override = default;

void Update() override;

};

## ПРИЛОЖЕНИЕ 26

#pragma once

#include <thread>

#include <mutex>

#include <condition\_variable>

#include <queue>

#include <vector>

#include <functional>

#include <future>

namespace my

class ThreadHolder :

{

std::function<void()> task;

public:

ThreadHolder(std::function<void()> task);

~ThreadHolder() = default;

void Execute();

};

class ThreadPool

{

bool stop;

std::vector<std::thread> workers;

std::queue<BaseThreadDataHolder\*> tasks;

std::mutex queue\_mutex;

std::condition\_variable condition;

friend class Engine;

public:

ThreadPool(size\_t num\_threads);

~ThreadPool();

void enqueue(std::function<void()> task);

void worker\_thread(size\_t id);

};

}