МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Объектно ориентированное программирование»

Тема: Конструкторы и деструкторы

Студент гр. 2383 _____ Борисов И.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2023

Цель работы.

Целью данной лабораторной работы является создание классов клетки игрового поля, а также самого игрового поля. Создание конструкторов копирования и перемещения, а также соответствующих им операторов присваивания. В класс управления игрока добавить взаимодействие с полем.

Задание.

а) Создать класс клетки игрового поля. Клетка игрового поля может быть проходимой или нет, тем самым определяя возможность игрока встать на эту клетку. Возможность задать проходимость клетки должна быть реализована через конструктор и через метод клетки. В будущем в клетке будет храниться указатель на интерфейс события.

б) Создать класс игрового поля. Игровое поле представляет собой прямоугольник из клеток (двумерный массив). В учебных целях, клетки хранятся как чистый массив на указателях (использовать контейнеры stl запрещено в этой лаб. работе). Размер поля передается в конструктор поля, в котором динамически выделяется память под массив клеток. Также должна быть возможность вызвать конструктор поля без аргументов. Так как происходит выделение память, то необходимо реализовать деструктор в котором будет происходить очистка память. Также для класса поля необходимо реализовать конструкторы копирования и перемещения, а также соответствующие им операторы присваивания.

Также добавить в игровое поле добавьте информацию о входе (где в начале появляется игрок) и выходе (куда игрок должен дойти)

в) В класс управления игрока добавить взаимодействия с полем. При перемещении игрока должна быть проверка на проходимость клетки, если клетка непроходима, то перемещение не должно производиться.

Примечания:

- Так как в клетке будет храниться указатель, то при копировании и перемещении должен быть предусмотрен механизм копирования объекта хранящегося по указателю.
- Убедитесь, что присутствует проверка контроля размера поля, чтобы его нельзя было сделать слишком маленьким или с отрицательными размерами.
- В конструкторе перемещения и соответствующем ему операторе присваивания не должно происходить никакого копирования данных.
- Через класс поля должен быть доступ к каждой индивидуальной клетке. Создавать метод, который возвращает указатель на весь массив или указатель на каждую клетку, плохая практика, так как появляется возможность очистки памяти.

Основные теоретические положения.

В языке C++ lvalues и rvalues - это два типа выражений, определяющих время жизни и изменяемость объектов. Под lvalue понимается объект, который имеет постоянный адрес в памяти и которому можно присваивать значения, а под rvalue - временный объект, который не имеет постоянного адреса и обычно используется в качестве значения или операнда в выражении.

Rvalue ссылки широко используются для реализации семантики перемещения, когда право собственности на ресурсы эффективно передается от одного объекта к другому.

Идиома RAII расшифровывается как Resource Acquisition Is Initialization. Это программный прием, используемый в языке C++ для управления ресурсами и обеспечения их правильной очистки. Основная идея RAII заключается в том, что получение ресурса должно быть связано с инициализацией объекта, а освобождение ресурса - с уничтожением объекта.

В языке C++ RAII обычно реализуется с помощью конструкторов и деструкторов классов. При создании объекта вызывается его конструктор, и происходит выделение или приобретение необходимых ресурсов. При уничтожении объекта вызывается его деструктор, и все необходимые операции по выделению или освобождению ресурсов происходят автоматически.

Выполнение работы.

Создан класс Cell. Внутри себя данный класс содержит публичный enum подкласс Type, который отвечает за тип клетки, а также приватное поле *Type type_*. Перегружены конструкторы копирования и перемещения, а также соответствующие им операторы присваивания. По умолчанию клетка создаётся проходимой.

Метод *set_type(Type new_type)* позволяет явно задать тип клетки.

Методы *is_entrance, is_exit, is_movable* позволяют определить проходимость данной клетки.

В дальнейшем в приватном поле будет храниться указатель на интерфейс события.

Создан класс Map. Данный класс имеет приватные поля *Dimension dimensions_, Position start_, Position finish_.* Данные поля хранят размерности

поля, а также точку старта и финиша. По умолчанию старт и финиш имеют координаты (-1, -1). Приватное поле *Cell **map_* содержит матрицу клеток.

Метод *clear_map()* очищает память выделенную под map_, а также зануляет указатель.

Метод *Cell **allocate_map(const Dimension &dimensions)* выделяет в куче память под новую карту и возвращает указатель на эту память.

Реализованы конструкторы копирования, перемещения и соответствующие операторы присваивания.

Конструкторы позволяют создавать карту с данными размерностями. Если размерности меньше чем нижний лимит (10), то выбрасывается исключение *std::logic_error("Unexpected dimensions for map (too small or too big)").*

Созданы методы *resert_start, reset_finish, build_wall, destroy_wall,* которые принимают позицию (*const Position &point)* и позволяют менять заданную точку на карте.

Методы *void set_cell(const Position &point, const Cell &new_cell)* позволяют явно изменить заданную клетку.

Реализованы методы *const Position &get_start_point() const, const Position &get_finish_point() const* возвращают ссылку на константу во избежании копирования и позволяют узнать позицию старта и финиша на карте.

Метод *Cell &get_cell(const Position &point) const* позволяет получить доступ к клетке по ссылке.

Методы *is_on_map* и *is_adjacent_to_movable* позволяют определить находится ли данная клетка на карте и связана ли данная клетка с хотя бы одной movable клеткой.

Далее был реализован класс MapHandler, который наследуется от HandlerInterface. В дальнейшем благодаря этому наследованию будет производиться взаимодействие с событием на клетке.

Приватное поле всего одно — *Map *map_* - указатель на карту. Владение объектом будет происходить до конца жизни объекта MapHandler, т. е. реализуется RAII.

Также есть метод *get_cell* аналогичный методу из класса Map, а также вспомогательный метод *bool can_move(const Position &point) const* который позволяет проверить возможно ли передвижение в данную точку.

Для всех классов в программе была создана UML диаграмма (Рис. 1)

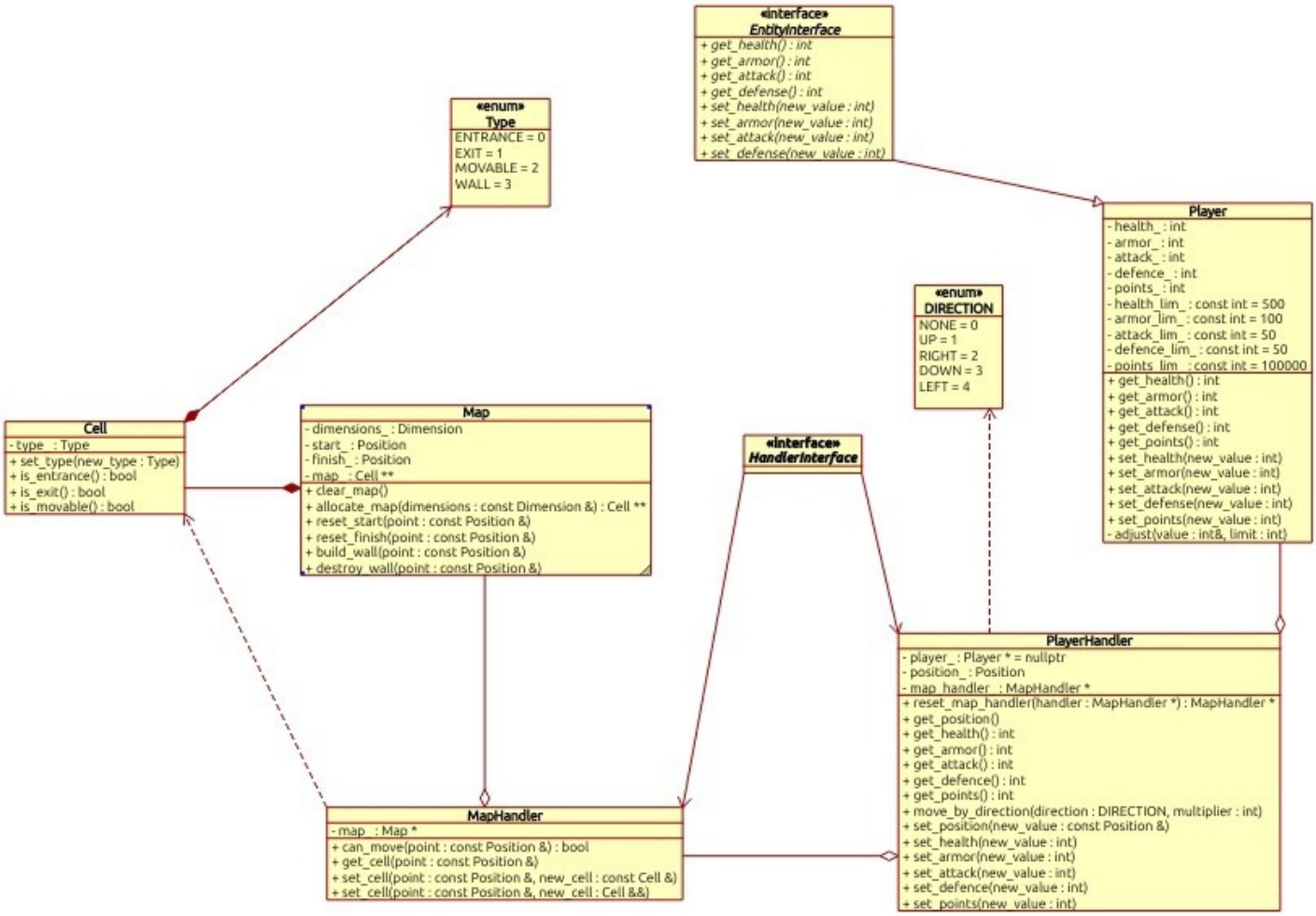

Рисунок 1 - Диаграмма

Для тестирования программы были использованы гугл тесты.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Cell cell;	EXPECT_EQ(cell.is_movable(), true);	Дефолтный конструктор клетки
2.	Cell cell(Cell::Type::WALL);	EXPECT_EQ(cell.is_movable(), false);	Конструктор типа
3.	Map map;	EXPECT_EQ(map.get_dimensions(), Dimension(10, 10)); EXPECT_EQ(map.get_start_point(), Position(-1, -1)); EXPECT_EQ(map.get_finish_point(), Position(-1, -1));	Дефолтный конструктор
4.	Map map(11, 11); Map copy = std::move(map);	EXPECT_EQ(copy.get_dimensions(), Dimension(11, 11)); EXPECT_EQ(copy.get_start_point(), Position(-1, -1)); EXPECT_EQ(copy.get_finish_point(), Position(-1, -1));	Конструктор карты
5.	Map *map = new Map; MapHandler map_handler(map); PlayerHandler handler(new Player, &map_handler); map->build_wall({0, 1});	EXPECT_EQ(map->get_cell({0, 1}).is_movable(), false); EXPECT_EQ(map->is_on_map({0, 1}), true); EXPECT_EQ(map_handler.can_move({0, 1}), false);	Проверка передвижения игрока

Выводы.

Разработаны классы клетки игрового поля, а также самого игрового поля. Созданы конструкторы копирования и перемещения, а также соответствующих им операторов присваивания. В класс управления игрока добавлено взаимодействие с полем. Новый функционал протестирован при помощи гугл тестов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: src/Entities/Interface/EntityInterface.hpp

```
#pragma once
#include "../../Math/Vector2.hpp"
#include "../../Movement/Aliases.hpp"

class EntityInterface {
 public:
  virtual ~EntityInterface() = default;

  [[nodiscard]] virtual int32_t get_health() const = 0;
  [[nodiscard]] virtual int32_t get_armor() const = 0;
  [[nodiscard]] virtual int32_t get_attack() const = 0;
  [[nodiscard]] virtual int32_t get_defense() const = 0;

  virtual void set_health(int32_t new_value) = 0;
  virtual void set_armor(int32_t new_value) = 0;
  virtual void set_attack(int32_t new_value) = 0;
  virtual void set_defense(int32_t new_value) = 0;
};
```

Название файла: src/Entities/Player/Player.hpp

```
#pragma once
#include "../Interface/EntityInterface.hpp"

class Player : public EntityInterface {
 private:
  const int32_t health_lim_ = 500;
  const int32_t armor_lim_ = 100;
  const int32_t attack_lim_ = 50;
  const int32_t defence_lim_ = 50;
  const int32_t points_lim_ = 100000;

  int32_t health_;
  int32_t armor_;
  int32_t attack_;
  int32_t defence_;
  int32_t points_;

  void adjust(int32_t &value, int32_t limit);
 public:
  Player(const Player &player);
  Player(Player &&player) noexcept;
  Player &operator=(const Player &player);
  Player &operator=(Player &&player) noexcept;
```

```
  explicit Player(int32_t health = 100,
                        int32_t armor = 0,
                        int32_t attack = 10,
                        int32_t defence = 10,
                        int32_t points = 0);

  [[nodiscard]] int32_t get_health() const override;
  [[nodiscard]] int32_t get_armor() const override;
  [[nodiscard]] int32_t get_attack() const override;
  [[nodiscard]] int32_t get_defense() const override;
  [[nodiscard]] int32_t get_points() const;

  void set_health(int32_t new_value) override;
  void set_armor(int32_t new_value) override;
  void set_attack(int32_t new_value) override;
  void set_defense(int32_t new_value) override;
  void set_points(int32_t new_value);
};
```

Название файла: src/Entities/Player/Player.cpp

```
#include "Player.hpp"

Player::Player(int32_t health,
                    int32_t armor,
                    int32_t attack,
                    int32_t defence,
                    int32_t points) :
                    health_(health),
                    armor_(armor),
                    attack_(attack),
                    defence_(defence),
                    points_(points)
{
}
Player::Player(const Player &player)
{
     *this = player;
}
Player::Player(Player &&player) noexcept
{
     *this = std::move(player);
}
Player &Player::operator=(const Player &player)
{
     if (this != &player)
     {
          health_ = player.health_;
          armor_ = player.armor_;
          attack_ = player.attack_;
          defence_ = player.defence_;
          points_ = player.points_;
     }
```

```
	return *this;
}
Player &Player::operator=(Player &&player) noexcept
{
	if (this != &player)
	{
		health_ = std::move(player.health_);
		armor_ = std::move(player.armor_);
		attack_ = std::move(player.attack_);
		defence_ = std::move(player.defence_);
		points_ = std::move(player.points_);
	}

	return *this;
}
int32_t Player::get_health() const
{
	return health_;
}
int32_t Player::get_armor() const
{
	return armor_;
}
int32_t Player::get_attack() const
{
	return attack_;
}
int32_t Player::get_defense() const
{
	return defence_;
}
int32_t Player::get_points() const
{
	return points_;
}
void Player::set_health(int32_t new_value)
{
	health_ = new_value;
	adjust(health_, health_lim_);
}
void Player::set_armor(int32_t new_value)
{
	armor_ = new_value;
	adjust(armor_, armor_lim_);
}
void Player::set_attack(int32_t new_value)
{
	attack_ = new_value;
	adjust(attack_, attack_lim_);
}
void Player::set_defense(int32_t new_value)
{
	defence_ = new_value;
	adjust(defence_, defence_lim_);
```

```
}
void Player::set_points(int32_t new_value)
{
     points_ = new_value;
     adjust(points_, points_lim_);
}
void Player::adjust(int32_t &value, int32_t limit)
{
     value = std::max(0, value);
     value = std::min(value, limit);
}
```

Название файла: src/Handlers/PlayerHandler/PlayerHandler.hpp

```
#pragma once
#include "../../Entities/Player/Player.hpp"
#include "../../Movement/Direction.hpp"
#include "../../World/Map.hpp"
#include "../Interface/HandlerInterface.hpp"
#include "Handlers/MapHandler/MapHandler.hpp"

class PlayerHandler : public HandlerInterface {
 private:
  Player *player_;
  Position position_;
  MapHandler *map_handler_;
 public:
  ~PlayerHandler() override;
  PlayerHandler() = delete;
   explicit PlayerHandler(Player *player, MapHandler *handler =
nullptr);
  MapHandler *reset_map_handler(MapHandler *handler);

  [[nodiscard]] const Position &get_position() const;
  [[nodiscard]] int32_t get_health() const;
  [[nodiscard]] int32_t get_armor() const;
  [[nodiscard]] int32_t get_attack() const;
  [[nodiscard]] int32_t get_defense() const;
  [[nodiscard]] int32_t get_points() const;

       void move_by_direction(DIRECTION direction, int32_t
multiplier);
  void set_position(const Position &new_value);
  void set_health(int32_t new_value);
  void set_armor(int32_t new_value);
  void set_attack(int32_t new_value);
  void set_defense(int32_t new_value);
  void set_points(int32_t new_value);
};
```

Название файла: src/Handlers/PlayerHandler/PlayerHandler.cpp

```
#include "PlayerHandler.hpp"

PlayerHandler::~PlayerHandler()
{
     delete player_;
}
const Position &PlayerHandler::get_position() const
{
     return position_;
}
int32_t PlayerHandler::get_health() const
{
     return player_->get_health();
}
int32_t PlayerHandler::get_armor() const
{
     return player_->get_armor();
}
int32_t PlayerHandler::get_attack() const
{
     return player_->get_attack();
}
int32_t PlayerHandler::get_defense() const
{
     return player_->get_defense();
}
int32_t PlayerHandler::get_points() const
{
     return player_->get_points();
}
void PlayerHandler::set_position(const Position &new_value)
{
     position_ = new_value;
}
void PlayerHandler::set_health(int32_t new_value)
{
     player_->set_health(new_value);
}
void PlayerHandler::set_armor(int32_t new_value)
{
     player_->set_armor(new_value);
}
void PlayerHandler::set_attack(int32_t new_value)
{
     player_->set_attack(new_value);
}
void PlayerHandler::set_defense(int32_t new_value)
{
     player_->set_defense(new_value);
}
void PlayerHandler::set_points(int32_t new_value)
{
     player_->set_points(new_value);
}
```

```
	void	PlayerHandler::move_by_direction(DIRECTION	direction,
int32_t multiplier)
	{
		for (int32_t i = 0; i < multiplier; ++i)
		{
			auto			new_position			=
Direction::getInstance().calculate_position(position_, direction);

			if (map_handler_ == nullptr)
			{
				throw	std::invalid_argument("MapHandler	was	not
initialized");
			}
			if (map_handler_->can_move(new_position))
			{
				set_position(new_position);

				// TODO activate events
			}
		}

		// TODO notify subscribers about player move
	}
	PlayerHandler::PlayerHandler(Player *player, MapHandler *handler)
: player_(player), map_handler_(handler)
	{
		if (player_ == nullptr)
		{
			throw	std::invalid_argument("Nullptr	passed	to
PlayerHandler");
		}
	}
	MapHandler *PlayerHandler::reset_map_handler(MapHandler *handler)
	{
		auto *old = map_handler_;
		map_handler_ = handler;
		return old;
	}
```

Название файла: src/Handlers/Interface/HandlerInterface.hpp

```
	#pragma once

	class HandlerInterface {
	 public:
	  virtual ~HandlerInterface() = default;
	};
```

Название файла: src/Handlers/Interface/MapHandler.hpp

```
	#pragma once
	#include "../Interface/HandlerInterface.hpp"
	#include "World/Map.hpp"
```

```
class MapHandler : public HandlerInterface {
 private:
  Map *map_;
 public:
  MapHandler() = delete;
  explicit MapHandler(Map *map);
  ~MapHandler();

  [[nodiscard]] bool can_move(const Position &point) const;
  [[nodiscard]] Cell &get_cell(const Position &point) const;
   void set_cell(const Position &point, const Cell &new_cell)
const;
  void set_cell(const Position &point, Cell &&new_cell) const;
};
```

Название файла: src/Handlers/Interface/MapHandler.cpp

```
#include "MapHandler.hpp"

MapHandler::MapHandler(Map *map) : map_(map)
{
      if (map_ == nullptr)
      {
            throw    std::invalid_argument("Nullptr    passed    to
MapHandler");
      }
}
Cell &MapHandler::get_cell(const Position &point) const
{
      return map_->get_cell(point);
}
void  MapHandler::set_cell(const  Position  &point,  const  Cell
&new_cell) const
{
      map_->set_cell(point, new_cell);
}
void MapHandler::set_cell(const Position &point, Cell &&new_cell)
const
{
      map_->set_cell(point, std::move(new_cell));
}
bool MapHandler::can_move(const Position &point) const
{
      // TODO check for other npc?
      return        map_->is_on_map(point)        &&        map_-
>get_cell(point).is_movable();
}
MapHandler::~MapHandler()
{
      delete map_;
}
```

Название файла: src/Math/Vector2.hpp

```
#pragma once
#include <algorithm>

template<typename T>
class Vector2 {
 private:
  T x_, y_;
 public:
  // getters/setters
  T get_y() const;
  T get_x() const;

  void set_y(const T &y);
  void set_x(T x);

  // constructors
  Vector2(const T &ix, const T &iy);
  Vector2(const Vector2 &other);
  Vector2();

  // operators
  Vector2 operator+(const Vector2 &other) const;
  Vector2 operator-(const Vector2 &other) const;
  Vector2 operator*(const Vector2 &other) const;
  Vector2 operator/(const Vector2 &other) const;
  Vector2 &operator+=(const Vector2 &other);
  Vector2 &operator-=(const Vector2 &other);
  Vector2 &operator*=(const Vector2 &other);
  Vector2 &operator/=(const Vector2 &other);
  Vector2 &operator=(const Vector2 &other);
  Vector2 &operator=(Vector2 &&other) noexcept;
  bool operator==(const Vector2 &other) const;
  bool operator<(const Vector2 &other) const;
};

template<typename T>
Vector2<T>::Vector2(const T &ix, const T &iy): x_(ix), y_(iy)
{
}
template<typename T>
Vector2<T>::Vector2(): Vector2(0, 0)
{
}
template<typename T>
Vector2<T>::Vector2(const Vector2 &other) : Vector2()
{
     *this = other;
}
template<typename T>
Vector2<T> Vector2<T>::operator+(const Vector2<T> &other) const
{
```

```
	Vector2<T> temp(*this);
	temp += other;
	return temp;
}
template<typename T>
Vector2<T> Vector2<T>::operator-(const Vector2<T> &other) const
{
	Vector2<T> temp(*this);
	temp -= other;
	return temp;
}
template<typename T>
Vector2<T> Vector2<T>::operator*(const Vector2<T> &other) const
{
	Vector2<T> temp(*this);
	temp *= other;
	return temp;
}
template<typename T>
Vector2<T> Vector2<T>::operator/(const Vector2<T> &other) const
{
	Vector2<T> temp(*this);
	temp /= other;
	return temp;
}
template<typename T>
Vector2<T> &Vector2<T>::operator+=(const Vector2<T> &other)
{
	x_ += other.x_;
	y_ += other.y_;
	return *this;
}
template<typename T>
Vector2<T> &Vector2<T>::operator-=(const Vector2<T> &other)
{
	x_ -= other.x_;
	y_ -= other.y_;
	return *this;
}
template<typename T>
Vector2<T> &Vector2<T>::operator*=(const Vector2<T> &other)
{
	x_ *= other.x_;
	y_ *= other.y_;
	return *this;
}
template<typename T>
Vector2<T> &Vector2<T>::operator/=(const Vector2<T> &other)
{
	if (other.x_ != 0 && other.y_ != 0)
	{
		x_ /= other.x_;
		y_ /= other.y_;
	}
	return *this;
```

```
}
template<typename T>
T Vector2<T>::get_y() const
{
     return y_;
}
template<typename T>
void Vector2<T>::set_y(const T &y)
{
     y_ = y;
}
template<typename T>
T Vector2<T>::get_x() const
{
     return x_;
}
template<typename T>
void Vector2<T>::set_x(T x)
{
     x_ = x;
}
template<typename T>
Vector2<T> &Vector2<T>::operator=(const Vector2 &other)
{
     if (this != &other)
     {
          x_ = other.x_;
          y_ = other.y_;
     }
     return *this;
}
template<typename T>
Vector2<T> &Vector2<T>::operator=(Vector2 &&other) noexcept
{
     if (this != &other)
     {
          x_ = std::move(other.x_);
          y_ = std::move(other.y_);
     }
     return *this;
}
template<typename T>
bool Vector2<T>::operator==(const Vector2 &other) const
{
     return x_ == other.x_ && y_ == other.y_;
}
template<typename T>
bool Vector2<T>::operator<(const Vector2 &other) const
{
     if (x_ == other.x_)
     {
          return y_ < other.y_;
     }
     return x_ < other.x_;
}
```

Название файла: src/Movement/Aliases.hpp

```
#pragma once
#include "../Math/Vector2.hpp"
#include <cstdint>

using Position = Vector2<int32_t>;
using Dimension = Vector2<int32_t>;
```

Название файла: src/Movement/Direction.hpp

```
#pragma once
#include "../Math/Vector2.hpp"
#include "../Entities/Interface/EntityInterface.hpp"
#include <vector>

enum DIRECTION {
     NONE, UP, RIGHT, DOWN, LEFT
};

class Direction {
 private:
  const std::vector<Vector2<int32_t>> possible_moves_ = {{0,  0},

       {-1, 0},

       {0,  1},

       {1,  0},

       {0,  -1}};
  Direction() = default;
 public:
  Direction(const Direction &) = delete;
  void operator=(const Direction &) = delete;

  static Direction &getInstance()
  {
       static Direction instance;
       return instance;
  }
    Position  calculate_position(const  Position  &old,  DIRECTION
direction);
};
```

Название файла: src/Movement/Direction.cpp

```
#include "Direction.hpp"

Position   Direction::calculate_position(const   Position   &old,
DIRECTION direction)
```

```
{
     return old + possible_moves_[direction];
}
```

Название файла: src/World/Cell.hpp

```
#pragma once

class Cell {
 public:
  enum Type {
       ENTRANCE, EXIT, MOVABLE, WALL
  };
 public:
  Cell();
  ~Cell();
  explicit Cell(Type type);
  Cell(const Cell &other);
  Cell(Cell &&other) noexcept;
  Cell &operator=(const Cell &other) noexcept;
  Cell &operator=(Cell &&other) noexcept;
  bool operator==(const Cell &other) const;

  void set_type(Type new_type);

  [[nodiscard]] bool is_entrance() const;
  [[nodiscard]] bool is_exit() const;
  [[nodiscard]] bool is_movable() const;

 private:
  Type type_;
};
```

Название файла: src/World/Cell.cpp

```
#include <algorithm>
#include "Cell.hpp"
```

```
Cell::Cell() : Cell(Type::MOVABLE)
{
}
Cell::Cell(Type type) : type_(type), event_(nullptr)
{
}
Cell::~Cell()
{
	// release event resources
}
bool Cell::is_entrance() const
{
	return type_ == Type::ENTRANCE;
}
bool Cell::is_exit() const
{
	return type_ == Type::EXIT;
}
bool Cell::is_movable() const
{
	return type_ == Type::MOVABLE || is_exit() || is_entrance();
}
Cell::Cell(const Cell &other) : Cell()
{
	*this = other;
}
Cell &Cell::operator=(Cell &&other) noexcept
{
	if (this != &other)
	{
		type_ = std::move(other.type_);
		other.event_ = nullptr;
	}
	return *this;
}
Cell::Cell(Cell &&other) noexcept: Cell()
{
```

```
      *this = std::move(other);
}
Cell &Cell::operator=(const Cell &other) noexcept
{
      if (this != &other)
      {
            type_ = other.type_;
      }
      return *this;
}
void Cell::set_type(Type new_type)
{
      type_ = new_type;
}
bool Cell::operator==(const Cell &other) const
{
      // TODO check for events
      return type_ == other.type_;
}
```

Название файла: src/World/Map.hpp

```
#pragma once
#include <cstdint>
#include "../Math/Vector2.hpp"
#include "../Movement/Aliases.hpp"
#include "Cell.hpp"

#define MAP_DIMENSION_UPPER_BOUND 1000
#define MAP_DIMENSION_LOWER_BOUND 10

class Map {
 private:
  Dimension dimensions_;
  Position start_;
  Position finish_;
```

```
        Cell **map_;

        void clear_map();
        Cell **allocate_map(const Dimension &dimensions);
      public:
        Map();
        Map(int32_t dim_x, int32_t dim_y);
        explicit Map(const Dimension &dimensions);

        Map(const Map &other);
        Map(Map &&other) noexcept;

        Map &operator=(const Map &other);
        Map &operator=(Map &&other) noexcept;

        void reset_start(const Position &point);
        void reset_finish(const Position &point);
        void build_wall(const Position &point);
        void destroy_wall(const Position &point);

        void set_cell(const Position &point, Cell &&new_cell);
        void set_cell(const Position &point, const Cell &new_cell);

        [[nodiscard]] const Position &get_start_point() const;
        [[nodiscard]] const Position &get_finish_point() const;
        [[nodiscard]] Cell &get_cell(const Position &point) const;
        [[nodiscard]] const Dimension &get_dimensions() const;
        [[nodiscard]] bool is_on_map(const Position &point) const;
           [[nodiscard]]  bool  is_adjacent_to_movable(const  Position
&point) const;

        ~Map();
      };
```

Название файла: src/World/Map.cpp

```
#include "Map.hpp"
#include <ostream>

Map::Map(const Dimension &dimensions) : dimensions_(dimensions), start_(-1, -1), finish_(-1, -1), map_(nullptr)
{
	if (dimensions_.get_x() < MAP_DIMENSION_LOWER_BOUND || dimensions_.get_y() < MAP_DIMENSION_LOWER_BOUND || dimensions_.get_x() > MAP_DIMENSION_UPPER_BOUND
		|| dimensions_.get_y() > MAP_DIMENSION_UPPER_BOUND)
	{
		throw std::logic_error("Unexpected dimensions for map (too small or too big)");
	}
	map_ = allocate_map(dimensions_);
}
Map::Map(int32_t dim_x, int32_t dim_y) : Map(Vector2<int32_t>{dim_x, dim_y})
{
}
Map::~Map()
{
	clear_map();
}
Map::Map() : Map(MAP_DIMENSION_LOWER_BOUND, MAP_DIMENSION_LOWER_BOUND)
{
}
Map::Map(const Map &other) : Map()
{
	*this = other;
}
Map::Map(Map &&other) noexcept: Map()
{
	*this = std::move(other);
}
```

```
Map &Map::operator=(const Map &other)
{
      if (&other != this)
      {
            clear_map();
            map_ = allocate_map(other.dimensions_);

            dimensions_ = other.dimensions_;
            start_ = other.start_;
            finish_ = other.finish_;

            for (int32_t i = 0; i < dimensions_.get_x(); ++i)
            {
                  for (int32_t j = 0; j < dimensions_.get_y(); ++j)
                  {
                        this->map_[i][j] = other.map_[i][j];
                  }
            }
      }
      return *this;
}
Map &Map::operator=(Map &&other) noexcept
{
      if (&other != this)
      {
            clear_map();
            dimensions_ = std::move(other.dimensions_);
            start_ = std::move(other.start_);
            finish_ = std::move(other.finish_);
            map_ = std::move(other.map_);

            // manually zeroing ptr because we moved it
            other.map_ = nullptr;
      }
      return *this;
}
void Map::clear_map()
```

```
{
	if (map_ == nullptr)
	{
		return;
	}

	for (int32_t i = 0; i < dimensions_.get_x(); ++i)
	{
		for (int32_t j = 0; j < dimensions_.get_y(); ++j)
		{
			// map_[i][j].remove_event();
		}
		delete[] map_[i];
	}
	delete[] map_;
	map_ = nullptr;
}
Cell **Map::allocate_map(const Dimension &dimensions)
{
	auto map = new Cell *[dimensions.get_x()];

	for (int32_t i = 0; i < dimensions.get_x(); ++i)
	{
		map[i] = new Cell[dimensions.get_y()];
	}
	return map;
}
Cell &Map::get_cell(const Position &point) const
{
	if (!is_on_map(point))
	{
		throw std::out_of_range("Point is out of map");
	}
	return map_[point.get_x()][point.get_y()];
}
void Map::set_cell(const Position &point, Cell &&new_cell)
{
```

```
            if (!is_on_map(point))
            {
                  return;
            }
            // TODO release resources of old cell ?
            map_[point.get_x()][point.get_y()] = std::move(new_cell);
      }
      void Map::set_cell(const Position &point, const Cell &new_cell)
      {
            if (!is_on_map(point))
            {
                  return;
            }
            map_[point.get_x()][point.get_y()] = new_cell;
      }
      const Dimension &Map::get_dimensions() const
      {
            return dimensions_;
      }
      void Map::reset_start(const Position &point)
      {
            if (is_on_map(point) && get_cell(point).is_movable())
            {
                  if (start_ != Vector2<int32_t>(-1, -1))
                  {
                        map_[start_.get_x()]
[start_.get_y()].set_type(Cell::Type::MOVABLE);
                  }
                  start_ = point;
                  map_[start_.get_x()]
[start_.get_y()].set_type(Cell::Type::ENTRANCE);
            }
      }
      void Map::reset_finish(const Vector2<int32_t> &point)
      {
            if (is_on_map(point) && get_cell(point).is_movable())
            {
```

```
                if (finish_ != Vector2<int32_t>(-1, -1))
                {
                     map_[finish_.get_x()]
[finish_.get_y()].set_type(Cell::Type::MOVABLE);
                }
                finish_ = point;
                map_[finish_.get_x()]
[finish_.get_y()].set_type(Cell::Type::EXIT);
           }
     }
     void Map::build_wall(const Position &point)
     {
           if (is_on_map(point) && get_cell(point).is_movable())
           {
                map_[point.get_x()]
[point.get_y()].set_type(Cell::Type::WALL);
           }
     }
     void Map::destroy_wall(const Position &point)
     {
           if (is_on_map(point) && !get_cell(point).is_movable())
           {
                map_[point.get_x()]
[point.get_y()].set_type(Cell::Type::MOVABLE);
           }
     }
     bool Map::is_on_map(const Position &point) const
     {
           return  point.get_x()  >=  0  &&  point.get_y()  >=  0  &&
point.get_x() < dimensions_.get_x()
                  && point.get_y() < dimensions_.get_y();
     }
     const Position &Map::get_start_point() const
     {
           return start_;
     }
     const Position &Map::get_finish_point() const
```

```
	{
		return finish_;
	}
	bool Map::is_adjacent_to_movable(const Position &point) const
	{
		for (int i = 1; i <= 4; ++i)
		{
			Position
neighbor(Direction::getInstance().calculate_position(point,
static_cast<DIRECTION>(i)));

			if (is_on_map(neighbor) &&
get_cell(neighbor).is_movable())
			{
				return true;
			}
		}
		return false;
	}
```