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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Объектно-ориентированное программирование»
Тема: Добавление класса управления игрой

Студент гр. 9382 Демин В.В.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Создать класс игры, через который пользователь взаимодействует с игрой.

Задание.

Создать класс игры, через который пользователь взаимодействует с игрой. Управление игроком, начало новой игры, завершение игры. Могут быть созданы дополнительные необходимые классы, которые отвечают отдельно за перемещение, создание игры и.т.д. Но пользователь должен взаимодействовать через интерфейс одного класса.

Обязательные требования:

- Создан класс управления игрой
- Взаимодействие сохраняет инвариант

Дополнительные требования:

- Пользователь взаимодействует с использованием паттерна **Команды**

  Взаимодействие с компонентами происходит через паттерн **Фасад**

Выполнение работы.

Был создан класс игры через который пользователь взаимодействует с игрой. В котором реализованы методы управления игроком, начало новой игры, завершение игры. Так, например, создав этот класс можно использовать различные методы для управления игроком, вне зависимости от того, какой способ будет реализован. Они будут работать независим. Нет конкретной привязки к технологии управления.

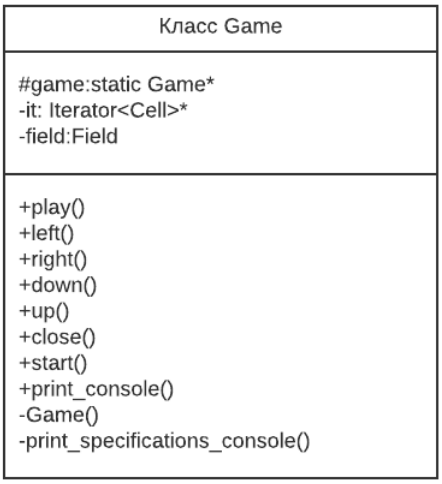

Рисунок 1. Uml диаграмма

Выводы.

В ходе выполнения лабораторной работы был создан класс игры для взаимодействия пользователя с игрой.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Game.h

```
//
// Created by vikto on 13.09.2020.
//


#ifndef MYGAME_GAME_H
#define MYGAME_GAME_H

#include "StatusEnum.h"
#include <iostream>
#include <cstdlib>
#include <conio.h>
#include <windows.h>
#include "Field/Field.h"
#include "Player/Player.h"
#include "../Tools/MyCollections/collections.h"
#include "../Tools/AbstractFactory/AbstractFactory.h"
#include "../Tools/AbstractFactory/ChestFactory.h"
#include "../Tools/AbstractFactory/CoinFactory.h"
#include "../Tools/AbstractFactory/MushroomFactory.h"
#include "../Tools/Logger/Logger.h"
#include "../Tools/Logger/LogFile.h"
#include "../Tools/Logger/LogConsole.h"
#include "Enemy/Enemy.h"
#include "../Tools/State/State.h"


class Game {
protected:

    static Game *game_;

    Game(Game &other) = delete;

    void operator=(const Game &) = delete;

public:
    class Memento {
    public:
        Memento(Field *field);
```

```
    const Field *field;
};

~Game();

void play();

void left();

void right();

void down();

void up();

void close();

void print_console();

void start();

void set_state(StateMove *state);

StateMove *get_state();

void move_player();//запихать в field
-
Memento save_memento();

void load_memento(Memento *load);

static Game *get_instance();


private:

Game();

StateMove *state;

Iterator<Cell> *it;

Field *field;
```

```
    void print_specifications_console();

};


#endif //MYGAME_GAME_H


#include "Game.cpp"

//
// Created by vikto on 13.09.2020.
//

#include "Game.h"
#include "Commands/Controller.h"
#include "States/StateMovePlayer.h"

Game *Game::game_ = nullptr;

Game::Game() {
    field = Field::get_instance();
    state = new StateMovePlayer(this);
}


Game *Game::get_instance() {
    if (game_ == nullptr) {
        game_ = new Game();
    }
    return game_;
}

void Game::play() {
    field->set_status(PLAY);

}

void Game::print_console() {
    system("cls");
    field->get_player()->log_cord_player();
    int size_y = field->get_size_y();
    int size_x = field->get_size_x();
    Cell **cells = field->get_cells();
```

```
CellCollection cont(cells, size_x, size_y);

it = cont.create_iterator();

while ((*(CellIterator *) it) != *(CellIterator *) cont.end()) {
   if (field->test_cord_of_player(dynamic_cast<CellIterator *>(it))) {
      std::cout << "+";
      ++(*it);
      continue;
   }

   if (field->test_cord_of_enemy(dynamic_cast<CellIterator *>(it)->get_cord_x(),
                dynamic_cast<CellIterator *>(it)->get_cord_y(), field->get_big_enemy())) {
      field->get_big_enemy()->print_enemy();
      ++(*it);
      continue;
   }
   if (field->test_cord_of_enemy(dynamic_cast<CellIterator *>(it)->get_cord_x(),
                dynamic_cast<CellIterator *>(it)->get_cord_y(), field->get_quick_enemy())) {
      field->get_quick_enemy()->print_enemy();
      ++(*it);
      continue;
   }
   if (field->test_cord_of_enemy(dynamic_cast<CellIterator *>(it)->get_cord_x(),
                dynamic_cast<CellIterator *>(it)->get_cord_y(), field->get_simple_enemy())) {
      field->get_simple_enemy()->print_enemy();
      ++(*it);
      continue;
   }
   if ((**it).get_element() != nullptr) {
      (**it).get_element()->print_element();
      ++(*it);
      continue;
   }
   if ((**it).get_type() == EMPTY) {
      std::cout << " ";
   }
   if ((**it).get_type() == HWALL) {
```

```
            std::cout << "_";
        }
        if ((**it).get_type() == VWALL || (**it).get_type() == CWALL) {
            std::cout << "|";
        }

        if ((**it).get_type() == EXIT) {
            std::cout << "$";
        }
        if ((**it).get_type() == ENTRANCE) {
            std::cout << "#";
        }
        if (((CellIterator *) it)->get_cord_x() == size_x - 1) {
            std::cout << "\n";
        }
        ++(*it);

    }
    std::cout << "|" << std::endl;
    print_specifications_console();
}

Game::~Game() {
    delete (CellIterator *) it;
    delete field;
    delete game_;
}

void Game::print_specifications_console() {
    std::cout << "Health:" << field->get_player()->get_health_player() <<
"\n";
    std::cout << "Coins:" << field->get_player()->get_coins_player() << "\n";
    if (field->get_count_coins() - field->get_player()->get_coins_player() < 0) {
        std::cout << "You need coins to win:" << 0 << "\n";
        return;
    }
    std::cout << "You need coins to win:" << field->get_count_coins() - field-
>get_player()->get_coins_player() << "\n";
}

void Game::left() {
    if (field->get_status() == PLAY) {
        if (field->test_cell(-1, 0)) {
            field->get_player()->set_cord_x(field->get_player()->get_cord_x()    -
1);
```

```
            }
            field->interaction_with_elements();
        }
    }

    void Game::right() {
        if (field->get_status() == PLAY) {
            if (field->test_cell(1, 0)) {
                field->get_player()->set_cord_x(field->get_player()->get_cord_x() +
1);
                field->interaction_with_elements();
            } else {
                if (field->get_cells()[field->get_player()->get_cord_y()][field-
>get_player()->get_cord_x() +
                                                        1].get_type() == EXIT &&
                    field->get_player()->get_coins_player() >= field-
>get_count_coins()) {
                    field->set_status(WIN);
                    field->get_player()->set_cord_x(field->get_player()->get_cord_x()
+ 1);
                } else {
                    std::cout << "Not enough coins!" << std::endl;
                }
            }
        }
    }

    void Game::down() {
        if (field->get_status() == PLAY) {
            if (field->test_cell(0, 1)) {
                field->get_player()->set_cord_y(field->get_player()->get_cord_y() +
1);
            }
            field->interaction_with_elements();
        }
    }

    void Game::up() {
        if (field->get_status() == PLAY) {
            if (field->test_cell(0, -1)) {
                field->get_player()->set_cord_y(field->get_player()->get_cord_y() -
1);
            }
            field->interaction_with_elements();
        }
```

```
}

void Game::close() {
   field->set_status(CLOSE);
}

void Game::start() {
   if (field->get_status() == WAIT) {
      field->set_status(PLAY);
   }
}


void Game::set_state(StateMove *state) {
   delete this->state;
   this->state = state;
}

void Game::move_player() {
   Controller controller(this);
   controller.inputCommand();
}


Game::Memento Game::save_memento() {
   //прописать get из поля
   Game::Memento save(field);
   return save;
}

void Game::load_memento(Game::Memento *load) {
   /*
    delete field;
   delete player_one;
   delete big_enemy;
   delete quick_enemy;
   delete simple_enemy;
      field = load->field;
   player_one = load->player_one;
   big_enemy = load->big_enemy;
   quick_enemy = load->quick_enemy;
   simple_enemy = load->simple_enemy;
   count_coins = load->count_coins;
   status = load->status;*/
}
```

```
Game::Memento::Memento(Field *field) : field(field) {

}

StateMove *Game::get_state() {
    return state;
}
```