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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «ООП»

Тема: Добавления игрока и элементов для поля

Студент гр. 9383 Моисейченко К.А.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Добавить класс игрока и классы элементов поля. Реализовать взаимодействие игрока с элементами. Изучить и реализовать паттерн Абстрактная фабрика.

Задание.

Создан класс игрока, которым управляет пользователь. Объект класса игрока может перемещаться по полю, а также взаимодействовать с элементами поля. Для элементов поля должен быть создан общий интерфейс и должны быть реализованы 3 разных класса элементов, которые по-разному взаимодействуют с игроком. Для взаимодействия игрока с элементом должен использоваться перегруженный оператор (Например, оператор +). Элементы поля могут добавлять очки игроку/замедлять передвижения/и.т.д.

Обязательные требования:

- Реализован класс игрока
- Реализованы три класса элементов поля
- Объект класса игрока появляется на клетке со входом
- Уровень считается пройденным, когда объект класса игрока оказывается на клетке с выходом (и при определенных условиях: например, набрано необходимое кол-во очков)
- Взаимодействие с элементами происходит через общий интерфейс
- Взаимодействие игрока с элементами происходит через перегруженный оператор

Дополнительные требования:

- Для создания элементов используется паттерн Фабричный метод/Абстрактная фабрика
- Реализовано динамическое изменение взаимодействия игрока с элементами через паттерн Стратегия. Например, при взаимодействии с определенным количеством элементов, игрок не может больше с ними взаимодействовать

Выполнение работы.

Класс Character:

Класс отвечает за реализацию персонажей, как игрока, так и врагов.

Поля:

int x, y - координаты на поле

Методы:

+ int GetX()

+ int GetY()

+ void Move(int, int) - изменяет поля координат на заданное перемещение

Класс Player:

Класс наследуется от Character, отвечает за реализацию игрока

Поля:

x, y

- int health - кол-во здоровья у игрока

- int energy - кол-во энергии у игрока (позволяет использовать прыжок)

- int points - кол-во очков у игрока

Методы:

+ Player(int, int) - конструктор, присваивает переданные значения полям координат

+ int GetHealth()

+ int GetEnergy()

+ int GetPoints()

+ void TakeItemHealth() - добавляет 1 единицу здоровья игроку

+ void TakeItemEnergy() - добавляет 1 единицу энергии игроку

+ void TakeItemPoint() - добавляет 1 очко игроку

+ void TakeDamage(int x) - уменьшает здоровье игрока на заданное значение

+ void UseJump() - уменьшает энергию игрока на единицу

Класс Item:

Интерфейс предметов игры:

Методы:
virtual void operator+(Player*) - взаимодействие с игроком
virtual int GetIndex() - индекс элемента

Класс ItemFactory:
Общий интерфейс создания элементов.
Методы:
+ Item* CreateItem()

Класс ItemHealth:
Наследуется от Item
Методы:
void operator+(Player*) - взаимодействие с игроком
int GetIndex() - индекс элемента

Класс ItemEnergy:
Наследуется от Item
Методы:
void operator+(Player*) - взаимодействие с игроком
Int GetIndex() - индекс элемента

Класс ItemPoint:
Наследуется от Item
Методы:
void operator+(Player*) - взаимодействие с игроком
Int GetIndex() - индекс элемента

Классы ItemHealthFactory, ItemEnergyFactory, ItemPointFactory:
Наследуются от ItemFactory. Создают соответственые элементы.
Методы:

+ Item* CreateItem()

Класс Cell:

Добавленные поля:

- bool isPlayerSet
- bool isItemSet
- Player* player
- Item* item

Добавленные методы:

+ void PlacePlayer(Player*)
+ void RemovePlayer()
+ bool IsPlayerSet()
+ void SetItem(Item*)
+ void DeleteItem()
+ bool IsItemSet()
+ Item* GetItem()

Класс Field:

Добавленные поля:

- Player* player

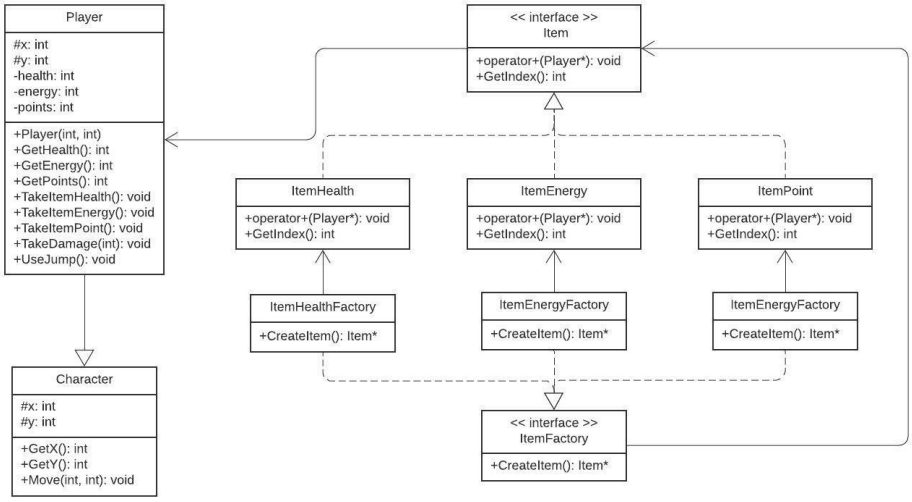
Добавленные методы:

+ bool IsPlayerSet(int, int);
+ bool IsItemSet(int, int);
+ Item* GetItem(int, int);
+ void SetPlayer(Player*);
+ Player* GetPlayer();

Также были созданы классы GameManager и UserInterface, которые отвечают за взаимодействие пользователя с игрой. Классы будут подробно описаны в лабораторной работе №4.

Разработанный программный код см. в приложении А.

UML-диаграмма.



Выводы.

В ходе работы были добавлены классы игрока и элементов поля.

Так же изучен и реализован паттерн Абстрактная фабрика.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: character.h

```
#pragma once

class Character {
protected:
    int x, y;

public:
    int GetX();
    int GetY();
    void Move(int dx, int dy);
};
```

Название файла: character.cpp

```
#include "character.h"

int Character::GetX()
{
    return x;
}

int Character::GetY()
{
    return y;
}

void Character::Move(int dx, int dy)
{
    this->x += dx;
    this->y += dy;
}
```

Название файла: player.h

```
#pragma once
#include "character.h"

class Player : public Character
{
private:
    int health;
    int energy;
    int points;
public:
    Player(int x, int y);
    int GetHealth();
    int GetEnergy();
    int GetPoints();
    void TakeItemHealth();
    void TakeItemEnergy();
    void TakeItemPoint();
```

```
    void TakeDamage(int x);
    void UseJump();
};
```

Название файла: player.cpp

```
#include "player.h"

Player::Player(int x, int y)
{
this->x = x;
this->y = y;
health = 6;
energy = 3;
points = 0;
}

int Player::GetHealth()
{
return health;
}

int Player::GetEnergy()
{
return energy;
}

int Player::GetPoints()
{
return points;
}

void Player::TakeItemHealth()
{
this->health++;
}

void Player::TakeItemEnergy()
{
this->energy++;
}

void Player::TakeItemPoint()
{
this->points++;
}

void Player::TakeDamage(int x)
{
this->health -= x;
}

void Player::UseJump() {
this->energy--;
}
```

Название файла: item.h

```
#pragma once

#include "../Characters/player.h"

class Player;

class Item {
public:
    virtual void operator+(Player* player) = 0;
    virtual int GetIndex() = 0;
};
```

Название файла: itemFactory.h

```
#pragma once

#include "item.h"

class ItemFactory
{
public:
virtual Item* CreateItem() = 0;
};
```

Название файла: itemHealth.h

```
#pragma once

#include "../item.h"

class ItemHealth : public Item {
public:
    void operator+(Player* player);
    int GetIndex();
};
```

Название файла: itemHealth.cpp

```
#include "itemHealth.h"

void ItemHealth::operator+(Player* player)
{
player->TakeItemHealth();
}

int ItemHealth::GetIndex()
{
return 0;
}
```

Название файла: itemHealthFactory.h

```
#pragma once
```

```
#include "../itemFactory.h"
#include "itemHealth.h"

class ItemHealthFactory : public ItemFactory {
public:
    Item* CreateItem();
};
```

Название файла: healthpointFactory.cpp

```
#include "itemHealthFactory.h"

Item* ItemHealthFactory::CreateItem() {
    return new ItemHealth;
}
```

Название файла: itemEnergy.h

```
#pragma once

#include "../item.h"

class ItemEnergy : public Item {
public:
    void operator+(Player* player);
    int GetIndex();
};
```

Название файла: itemEnergy.cpp

```
#include "itemEnergy.h"

void ItemEnergy::operator+(Player* player)
{
player->TakeItemEnergy();
}

int ItemEnergy::GetIndex()
{
return 1;
}
```

Название файла: itemEnergyFactory.h

```
#pragma once

#include "../itemFactory.h"
#include "itemEnergy.h"

class ItemEnergyFactory : public ItemFactory {
public:
    Item* CreateItem();
};
```

Название файла: itemEnergyFactory.cpp

```
#include "itemEnergyFactory.h"

Item* ItemEnergyFactory::CreateItem() {
    return new ItemEnergy;
}
```

Название файла: itemPoint.h

```
#pragma once

#include "../item.h"

class ItemPoint : public Item {
public:
    void operator+(Player* player);
    int GetIndex();
};
```

Название файла: itemPoint.cpp

```
#include "itemPoint.h"

void ItemPoint::operator+(Player* player)
{
player->TakeItemPoint();
}

int ItemPoint::GetIndex()
{
return 2;
}
```

Название файла: itemPointFactory.h

```
#pragma once

#include "../itemFactory.h"
#include "itemPoint.h"

class ItemPointFactory : public ItemFactory {
public:
    Item* CreateItem();
};
```

Название файла: pointFactory.cpp

```
#include "pointFactory.h"

Element* PointFactory::CreateElement() {
```

```
    return new Point;
}
```

Название файла: cell.h

```
#pragma once

#include "../Items/item.h"
#include "../Characters/player.h"

enum class CellType
{
    PATH,
    WALL,
    START,
    END
};

class Cell
{
private:
    CellType cellType;
    bool isPlayerSet;
    bool isItemSet;
    Player* player = nullptr;
    Item* item = nullptr;
public:
    Cell();
    CellType GetCellType();
    void SetCellType(CellType cellType);
    void PlacePlayer(Player* player);
    void RemovePlayer();
    bool IsPlayerSet();
    void SetItem(Item* item);
    void DeleteItem();
    bool IsItemSet();
    Item* GetItem();
    ~Cell();
};
```

Название файла: cell.cpp

```
#include "cell.h"

Cell::Cell()
{
    cellType = CellType::PATH;
    isPlayerSet = false;
    isItemSet = false;
}

CellType Cell::GetCellType()
{
    return cellType;
}
```

```
void Cell::SetCellType(CellType cellType)
{
    this->cellType = cellType;
}

void Cell::PlacePlayer(Player* player)
{
    isPlayerSet = true;
    this->player = player;
}

void Cell::RemovePlayer()
{
    isPlayerSet = false;
    this->player = nullptr;
}

bool Cell::IsPlayerSet()
{
    return isPlayerSet;
}

void Cell::SetItem(Item* item) {
    isItemSet = true;
    this->item = item;
}

void Cell::DeleteItem()
{
    isItemSet = false;
    if (item)
    {
        delete item;
        item = nullptr;
    }
}

bool Cell::IsItemSet()
{
    return isItemSet;
}

Item* Cell::GetItem()
{
    return item;
}

Cell::~Cell()
{
    DeleteItem();
}
```

Название файла: field.h

```
#pragma once

#include <iostream>
```

```
#include "../Cell/cell.h"
#include "../Characters/player.h"

class Cell;

class Field
{
private:
    static Field* instance;
    int height;
    int width;
    Cell** field = nullptr;
    Player* player = nullptr;
    std::pair<int, int> start;
    std::pair<int, int> end;
    Field();
    Field(const Field& other);
    Field(Field&& other);

    Field& operator = (const Field& other);
    Field& operator = (Field&& other);

    void cleanCells();

public:
    static Field* GetInstance();
    int GetHeight();
    int GetWidth();
    Cell** GetField();
    std::pair<int, int> GetStart();
    std::pair<int, int> GetEnd();
    bool IsWall(int x, int y);
    bool IsStartOrEnd(int x, int y);
    bool IsPlayerSet(int x, int y);
    bool IsItemSet(int x, int y);
    Item* GetItem(int x, int y);
    void SetPlayer(Player* player);
    Player* GetPlayer();
    void DeleteField();
};
```

Название файла: field.cpp

```
#include "field.h"

Field::Field()
{
    this->height = 22;
    this->width = 19;
    if (!field)
    {
        field = new Cell * [height];
        for (int i = 0; i < height; i++)
        {
            field[i] = new Cell[width];
        }
```

```
            int                     myField[22][19]                     =
{ {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1},

{1,0,1,0,0,1,1,1,0,1,0,1,1,1,0,0,1,0,1},

{1,0,1,1,0,0,1,1,0,1,0,1,1,0,0,1,1,0,1},

{1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},

{1,0,1,1,0,1,0,1,1,1,1,1,0,1,0,1,1,0,1},

{1,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,1},

{1,1,1,1,0,1,1,1,0,1,0,1,1,1,0,1,1,1,1},

{1,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,1},

{1,0,1,1,0,1,0,1,1,0,1,1,0,1,0,1,1,0,1},

{1,2,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,3,1},

{1,0,1,1,0,1,0,1,1,0,1,1,0,1,0,1,1,0,1},

{1,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,1},

{1,1,1,1,0,1,0,1,1,1,1,1,1,0,1,0,1,1,1,1},

{1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1},

{1,0,1,1,0,1,1,1,0,1,0,1,1,1,0,1,1,0,1},

{1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1},

{1,1,0,1,0,1,0,1,1,1,1,1,1,0,1,0,1,0,1,1},

{1,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,1},

{1,0,1,1,1,1,1,1,1,0,1,0,1,1,1,1,1,1,0,1},

{1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1} };
            start = std::make_pair(1, 10);
            end = std::make_pair(17, 10);
            for (int i = 0; i < height; i++)
            {
                for (int j = 0; j < width; j++)
                {
                    if (myField[i][j] == 0)
                        field[i][j].SetCellType(CellType::PATH);
                    if (myField[i][j] == 1)
                        field[i][j].SetCellType(CellType::WALL);
                    if (myField[i][j] == 2)
                        field[i][j].SetCellType(CellType::START);
                    if (myField[i][j] == 3)
                        field[i][j].SetCellType(CellType::END);
```

```
            }
        }
    }
}

Field::Field(const Field& other)
{
    this->width = other.width;
    this->height = other.height;
    this->field = new Cell * [height];
    for (int i = 0; i < height; i++)
    {
        this->field[i] = new Cell[width];
        for (int j = 0; j < width; j++)
        {
            this->field[i][j] = other.field[i][j];
        }
    }
}

Field::Field(Field&& other)
{
    this->width = other.width;
    this->height = other.height;
    this->field = other.field;
    other.field = nullptr;
}

Field& Field::operator=(const Field& other)
{
    if (&other == this)
        return *this;
    cleanCells();
    this->width = other.width;
    this->height = other.height;
    this->field = new Cell * [height];
    for (int i = 0; i < height; i++)
    {
        this->field[i] = new Cell[width];
        for (int j = 0; j < width; j++)
        {
            this->field[i][j] = other.field[i][j];
        }
    }
    return *this;
}

Field& Field::operator=(Field&& other)
{
    cleanCells();
    this->width = other.width;
    this->height = other.height;
    this->field = other.field;
    return *this;
}

void Field::cleanCells()
{
```

```
        if (field)
        {
            for (int i = 0; i < height; i++)
            {
                delete[] field[i];
            }
            delete[] field;
            field = nullptr;
        }
    }

    Field* Field::GetInstance()
    {
        if (instance == nullptr)
            instance = new Field;
        return instance;
    }

    int Field::GetHeight()
    {
        return height;
    }

    int Field::GetWidth()
    {
        return width;
    }

    Cell** Field::GetField()
    {
        return field;
    }

    std::pair<int, int> Field::GetStart()
    {
        return start;
    }

    std::pair<int, int> Field::GetEnd()
    {
        return end;
    }

    bool Field::IsWall(int x, int y)
    {
        if (x < 0 || y < 0 || x >= width || y >= height)
            return true;
        return field[y][x].GetCellType() == CellType::WALL;
    }

    bool Field::IsStartOrEnd(int x, int y)
    {
        if (x < 0 || y < 0 || x >= width || y >= height)
            return false;
        return   (field[y][x].GetCellType()   ==   CellType::START   ||
field[y][x].GetCellType() == CellType::END);
    }
```

```
bool Field::IsPlayerSet(int x, int y)
{
    return field[y][x].IsPlayerSet();
}

bool Field::IsItemSet(int x, int y)
{
    return field[y][x].IsItemSet();
}

Item* Field::GetItem(int x, int y)
{
    return field[y][x].GetItem();
}

void Field::SetPlayer(Player* player)
{
    this->player = player;
}
Player* Field::GetPlayer()
{
    return player;
}

void Field::DeleteField()
{
    cleanCells();
    delete instance;
    instance = nullptr;
}

Field* Field::instance = nullptr;
```

Название файла: GameManager.h

```
#pragma once

#include <stdlib.h>
#include <time.h>
#include <iostream>
#include "../Field/field.h"
#include "../Items/ItemHealth/itemHealth.h"
#include "../Items/ItemHealth/itemHealthFactory.h"
#include "../Items/ItemEnergy/itemEnergy.h"
#include "../Items/ItemEnergy/itemEnergyFactory.h"
#include "../Items/ItemPoint/itemPoint.h"
#include "../Items/ItemPoint/itemPointFactory.h"
#include "../UserInterface/userInterface.h"

class GameManager
{
private:
Field* field;
Player* player;
UserInterface* userInterface;
int numOfItemPoint;
int numOfItemHealth;
```

```
int numOfItemEnergy;
int pointsToWin;
std::pair<int, int> randomCell();
void setGameObjects();
void parseMove();
void openWin();
void nextMove(int dx, int dy);
public:
GameManager();
void OpenNew();
~GameManager();
};
```

Название файла: GameManager.cpp

```
#include "gameManager.h"

GameManager::GameManager()
{
field = nullptr;
player = nullptr;
userInterface = new UserInterface;
}

std::pair<int, int> GameManager::randomCell()
{
int x = rand() % field->GetWidth();
int y = rand() % field->GetHeight();

while (field->GetField()[y][x].GetCellType() != CellType::PATH ||
field->IsItemSet(x, y))
{
x = rand() % field->GetWidth();
y = rand() % field->GetHeight();
}
return std::make_pair(x, y);
}

void GameManager::setGameObjects()
{
ItemFactory* itemFactory;
std::pair<int, int> randPos = randomCell();
itemFactory = new ItemPointFactory;
for (int i = 0; i < numOfItemPoint; i++)
{
randPos = randomCell();
field->GetField()[randPos.second][randPos.first].SetItem(itemFactor
y->CreateItem());
}
delete itemFactory;
itemFactory = new ItemHealthFactory;
for (int i = 0; i < numOfItemHealth; i++)
{
randPos = randomCell();
field->GetField()[randPos.second][randPos.first].SetItem(itemFactor
y->CreateItem());
}
delete itemFactory;
```

```
      itemFactory = new ItemEnergyFactory;
      for (int i = 0; i < numOfItemEnergy; i++)
      {
      randPos = randomCell();
      field->GetField()[randPos.second][randPos.first].SetItem(itemFactor
y->CreateItem());
      }
      delete itemFactory;
      }

      void GameManager::OpenNew()
      {
      if (player)
      delete player;
      if (field)
      field->DeleteField();
      field = Field::GetInstance();
      player         =         new         Player(field->GetStart().first,
field->GetStart().second);
      srand(time(0));
      numOfItemPoint = rand() % 4 + 4;
      pointsToWin = numOfItemPoint;
      numOfItemHealth = rand() % 7 + 6;
      numOfItemEnergy = rand() % 6 + 10;
      field->SetPlayer(player);
      field->GetField()[field->GetStart().second][field->GetStart().first
].PlacePlayer(player);
      setGameObjects();
      parseMove();
      }

      void GameManager::parseMove()
      {
      userInterface->PrintGame(player, pointsToWin);
      std::string userCommand = userInterface->ScanCommand();
      if (userCommand == "a" || userCommand == "left")
      nextMove(-1, 0);
      else if (userCommand == "d" || userCommand == "right")
      nextMove(1, 0);
      else if (userCommand == "w" || userCommand == "up")
      nextMove(0, -1);
      else if (userCommand == "s" || userCommand == "down")
      nextMove(0, 1);
      else if (userCommand == "a2" || userCommand == "left2" || userCommand
== "2a" || userCommand == "2left")
      nextMove(-2, 0);
      else if (userCommand == "d2" || userCommand == "right2" || userCommand
== "2d" || userCommand == "2right")
      nextMove(2, 0);
      else if (userCommand == "w2" || userCommand == "up2" || userCommand
== "2w" || userCommand == "2up")
      nextMove(0, -2);
      else if (userCommand == "s2" || userCommand == "down2" || userCommand
== "2s" || userCommand == "2down")
      nextMove(0, 2);
      else if (userCommand != "q" && userCommand != "quit")
      {
      std::cout << "Invalid command!\n";
```

```
parseMove();
}
}

void GameManager::openWin()
{
userInterface->PrintWin();
std::string userCommand = userInterface->ScanCommand();
if (userCommand == "n" || userCommand == "new" || userCommand ==
">>")
OpenNew();
else if (userCommand != "q" && userCommand != "quit")
{
std::cout << "Invalid command!\n";
openWin();
}
}

void GameManager::nextMove(int dx, int dy)
{
if (field->IsWall(player->GetX() + dx, player->GetY() + dy)) {
std::cout << "That's not a valid move!\n";
parseMove();
return;
}
if (abs(dx + dy) == 2)
{
if (player->GetEnergy() > 0)
{
player->UseJump();
}
else
{
std::cout << "You have no energy!\n";
parseMove();
return;
}
}
field->GetField()[player->GetY()][player->GetX()].RemovePlayer();
player->Move(dx, dy);
field->GetField()[player->GetY()][player->GetX()].PlacePlayer(playe
r);
if (field->IsItemSet(player->GetX(), player->GetY()))
{
*(field->GetItem(player->GetX(), player->GetY())) + player;
switch (field->GetItem(player->GetX(), player->GetY())->GetIndex())
{
case 0:
numOfItemHealth--;
break;
case 1:
numOfItemEnergy--;
break;
case 2:
numOfItemPoint--;
break;
default:
break;
```

```
      }
      field->GetField()[player->GetY()][player->GetX()].DeleteItem();
      }
      if  (field->GetField()[player->GetY()][player->GetX()].GetCellType()
== CellType::END)
      {
      if (player->GetPoints() == pointsToWin)
      {
      openWin();
      return;
      }
      else
      {
      std::cout << "You haven't collected all the points!\n";
      parseMove();
      return;
      }
      }
      parseMove();
      }

      GameManager::~GameManager()
      {
      if (player)
      delete player;
      if (field)
      field->DeleteField();
      }
```

Название файла: UserInterface.h

```
      #pragma once
      #include <iostream>
      #include <string>
      #include <algorithm>
      #include <cctype>
      #include "../Field/field.h"

      class UserInterface
      {
      private:
      Field* field;
      public:
      UserInterface();
      std::string ScanCommand();
      void PrintGame(Player* player, int pointsToWin);
      void PrintWin();
      };
```

Название файла: UserInterface.cpp

```
      #include "userInterface.h"

      UserInterface::UserInterface()
      {
      field = Field::GetInstance();
```

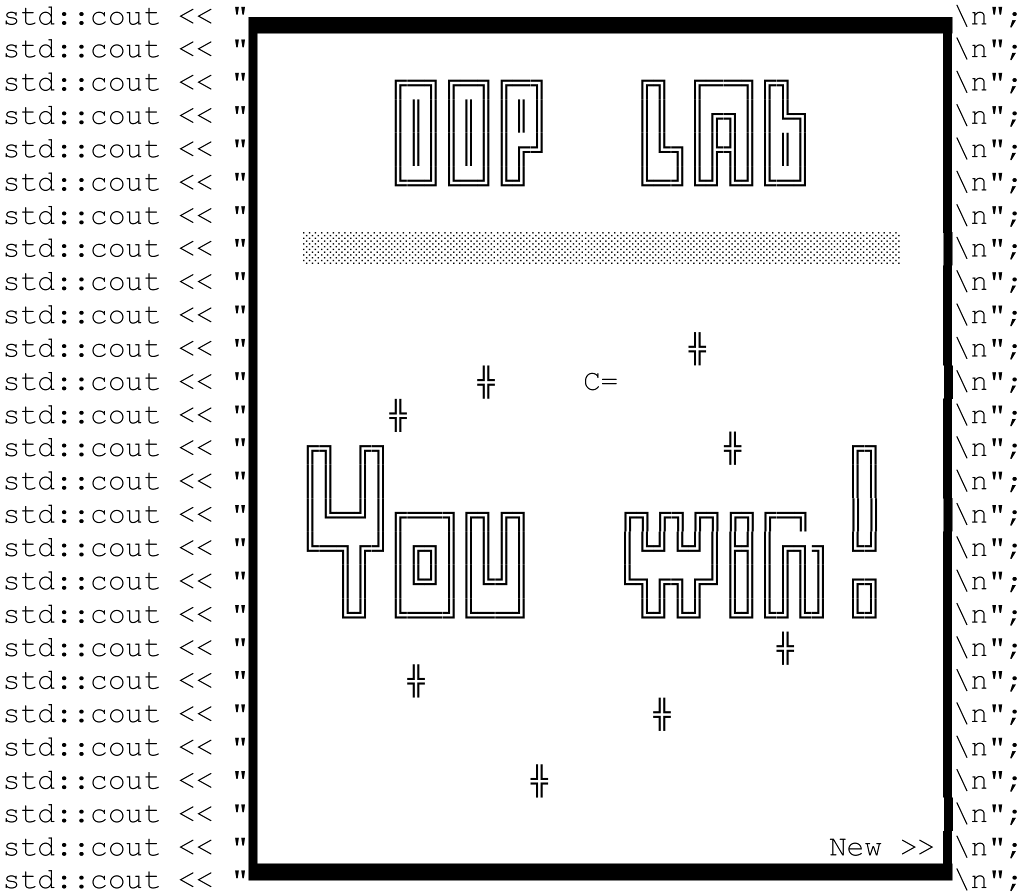
```
}

std::string UserInterface::ScanCommand() {
std::cout << "Enter your command: ";
std::string userCommand;
std::getline(std::cin, userCommand);
std::transform(userCommand.begin(),              userCommand.end(),
userCommand.begin(),
[](unsigned char c) { return std::tolower(c); });
return userCommand;
}

void UserInterface::PrintGame(Player* player, int pointsToWin) {
std::cout << "┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓\n";
std::cout << "┃                                      ┃\n";
std::cout << "┃ Health<3 " << player->GetHealth();
if (player->GetHealth() < 10)
std::cout  <<  "   Points{+  "  <<  player->GetPoints()  <<  '/'  <<
pointsToWin;
else
std::cout  <<  "  Points{+  "  <<  player->GetPoints()  <<  '/'  <<
pointsToWin;
if(player->GetEnergy() < 10)
std::cout << "  Energy~@ " << player->GetEnergy() << "  ┃\n";
else
std::cout << " Energy~@ " << player->GetEnergy() << "  ┃\n";
std::cout << "┃                                      ┃\n";
for (int i = 0; i < field->GetHeight(); i++)
{
std::cout << "┃ ";
for (int j = 0; j < field->GetWidth(); j++)
{
if (field->IsPlayerSet(j, i))
std::cout << "(=";
else if (field->IsItemSet(j, i))
{
switch (field->GetItem(j, i)->GetIndex())
{
case 0:
std::cout << "<3";
break;
case 1:
std::cout << "~@";
break;
case 2:
std::cout << "{+";
break;
default:
break;
}
}
else {
switch (field->GetField()[i][j].GetCellType())
{
case CellType::PATH:
std::cout << "  ";
break;
case CellType::WALL:
```

```
std::cout << "░";
break;
case CellType::START:
std::cout << "->";
break;
case CellType::END:
std::cout << ">>";
break;
default:
break;
}
}
}
std::cout << "█\n";
}
std::cout << "████████████████████████████████████████\n";
}

void UserInterface::PrintWin()
{
```




```
}
```

Название файла: main.cpp

```
#include "../GameManager/gameManager.h"

int main() {
    GameManager* gameManager = new GameManager;
    gameManager->OpenNew();
    if (gameManager)
        delete gameManager;
```

```
    return 0;
}
```

Название файла: Makefile

```
all: main.o cell.o field.o character.o player.o item.o itemFactory.o itemHealth.o itemHealthFactory.o itemEnergy.o itemEnergyFactory.o itemPoint.o itemPointFactory.o gameManager.o userInterface.o
	g++ main.o cell.o field.o character.o player.o item.o itemFactory.o itemHealth.o itemHealthFactory.o itemEnergy.o itemEnergyFactory.o itemPoint.o itemPointFactory.o gameManager.o userInterface.o -o oop_lab

main.o: Main/main.cpp
	g++ -c Main/main.cpp

cell.o: Cell/cell.cpp Cell/cell.h
	g++ -c Cell/cell.cpp

field.o: Field/field.cpp Field/field.h
	g++ -c Field/field.cpp

character.o: Characters/character.cpp Characters/character.h
	g++ -c Characters/character.cpp

player.o: Characters/player.cpp Characters/player.h
	g++ -c Characters/player.cpp

item.o: Items/item.cpp Items/item.h
	g++ -c Items/item.cpp

itemFactory.o: Items/itemFactory.cpp Items/itemFactory.h
	g++ -c Items/itemFactory.cpp

itemHealth.o: Items/ItemHealth/itemHealth.cpp Items/ItemHealth/itemHealth.h
	g++ -c Items/ItemHealth/itemHealth.cpp

itemHealthFactory.o: Items/ItemHealth/itemHealthFactory.cpp Items/ItemHealth/itemHealthFactory.h
	g++ -c Items/ItemHealth/itemHealthFactory.cpp

itemEnergy.o: Items/ItemEnergy/itemEnergy.cpp Items/ItemEnergy/itemEnergy.h
	g++ -c Items/ItemEnergy/itemEnergy.cpp

itemEnergyFactory.o: Items/ItemEnergy/itemEnergyFactory.cpp Items/ItemEnergy/itemEnergyFactory.h
	g++ -c Items/ItemEnergy/itemEnergyFactory.cpp

itemPoint.o: Items/ItemPoint/itemPoint.cpp Items/ItemPoint/itemPoint.h
	g++ -c Items/ItemPoint/itemPoint.cpp

itemPointFactory.o: Items/ItemPoint/itemPointFactory.cpp Items/ItemPoint/itemPointFactory.h
	g++ -c Items/ItemPoint/itemPointFactory.cpp
```

```
	gameManager.o: GameManager/gameManager.cpp GameManager/gameManager.h
	g++ -c GameManager/gameManager.cpp

	userInterface.o:                    UserInterface/userInterface.cpp
UserInterface/userInterface.h
	g++ -c UserInterface/userInterface.cpp

	clean:
	@rm -rf *.o
	echo "\nClean complete"
```