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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «ООП»
Тема: Добавления врагов

Студент гр. 9383 _____ Моисейченко К.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Создать шаблонный класс врага. Изучить и реализовать паттерн Состояния.

Задание.

Создать шаблонный класс врага. Параметр шаблона должен определять поведение врага (параметров шаблона может быть несколько, например отдельный параметр для политики передвижения и для политики атаки). Класс врага должен препятствовать игроку. Класс игрока должен иметь возможность взаимодействовать с врагом и наоборот.

Обязательные требования:

- Создан шаблонный класс врага
- Создано не менее 3 типа поведения врагов
- Взаимодействие происходит через перегруженный оператор

Дополнительные требования:

- Передача хода между игроком и врагами происходит с использованием паттерна Состояния в классе игры

Выполнение работы.

Класс Enemy:

Шаблонный класс, наследуется от класса Character. Является реализацией класса врага, поведение врага зависит от шаблона

Поля:

int x - координата X, унаследовано от Character

int y - координата Y, унаследовано от Character

- T* policy - указатель на класс поведения

Методы:

+ Enemy(int, int) - задает поля координат, динамически выделяет память под поле с классом поведения

+ void Move(int, int) - добавляет полям координат заданные значения

+ T* GetPolicy() - возвращает указатель на класс поведения

+ void MakeMove() - в зависимости от класса поведения запускает нужный метод перемещения врага

+ ~Enemy() - освобождает память, выделенную динамически под поле policy

- void moveChaser() - метод перемещения врага Chaser. Может переместить врага на одну клетку вверх/влево/вправо/вниз. Для нахождения кротчайшего пути до игрока был реализован волновой алгоритм Ли

- void moveAmbusher() - метод перемещения врага Ambusher. Если игрок находится в радиусе одной клетки от врага, перемещает к игроку. Иначе определяет, в какой четверти поля находится игрок и перемещает в случайную свободную клетку в соответствующей четверти.

- void moveJumper() - метод перемещения врага Jumper. Враг перемещается как конь из шахмат. Выбирает наиболее выгодную позицию в зависимости от того, где находится игрок и какие клетки свободны.

Класс Policy:

Интерфейс класса поведения врага

Методы:

+ virtual int GetIndex()

+ virtual void operator+(Player* player)

Классы PolicyChaser, PolicyAmbusher, PolicyJumper:

Методы

+ int GetIndex() - для PolicyChaser: возвращает 0, для PolicyAmbusher: возвращает 1, для PolicyJumper: возвращает 2

+ void operator+(Player* player) - для PolicyChaser: наносит игроку 2 урона, для PolicyAmbusher: наносит игроку 3 урона, для PolicyJumper: наносит игроку 1 урон

Класс Cell:

Добавлены поля и методы, связанные со врагами

Поля:

- bool isEnemySet

- Enemy<PolicyChaser>* chaser

- Enemy<PolicyAmbusher>* ambusher

- Enemy<PolicyJumper>* jumper

Методы:

+ bool IsEnemySet() - возвращает значение поля isEnemySet

+ void PlaceChaser(Enemy<PolicyChaser>*) - помещает в клетку врага Chaser

+ void RemoveChaser() - убирает врага Chaser из клетки

+ void PlaceAmbusher(Enemy<PolicyAmbusher>*) - помещает в клетку врага Ambusher

+ void RemoveAmbusher() - убирает врага Ambusher из клетки

+ void PlaceJumper(Enemy<PolicyJumper>*) - помещает в клетку врага Jumper

+ void RemoveJumper() - убирает врага Jumper из клетки

+ int GetEnemyIndex() - возвращает индекс врага в клетке

Класс Field:

Добавлен метод + bool IsEnemySet(int, int)

Класс GameManager:

Добавлены поля chaser, ambusher, jumper

В конструктор добавлена инициализация новых полей

В метод setGameObjects() добавлено создание трех различных врагов

В деструктор добавлено освобождение выделенной динамически памяти под поля врагов

В метод nextMove() добавлены перемещения врагов и взаимодействие врагов с игроком

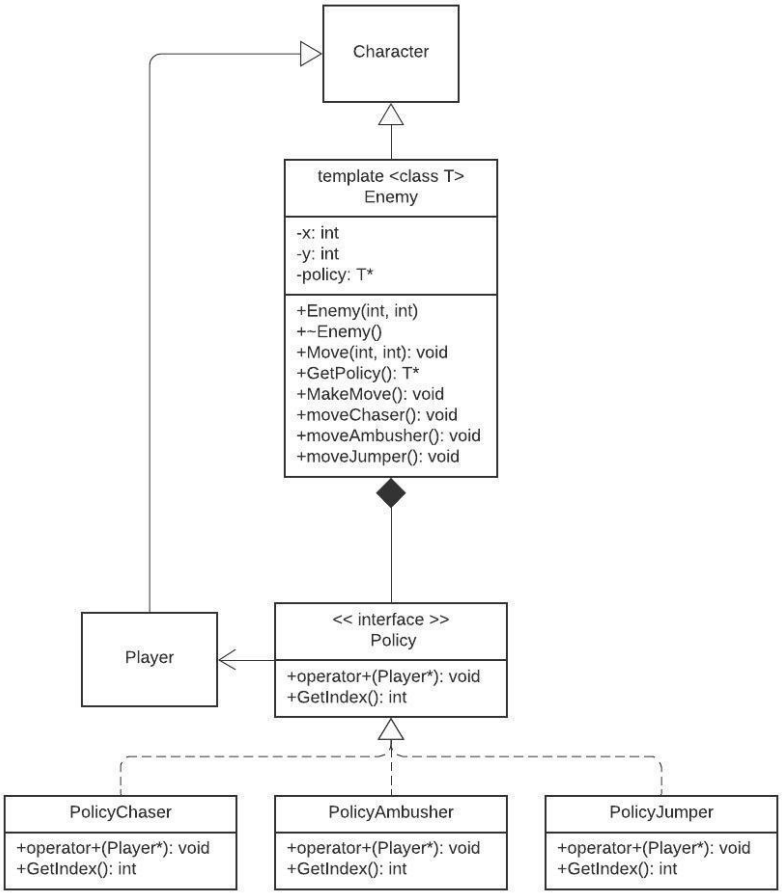
Добавлено условие проигрыша игры

Добавлены методы открытия новых окон интерфейса, связанных со врагами

Класс UserInterface:
Добавлена отрисовка врагов на поле
Добавлена отрисовка новых окон интерфейса, связанных со врагами
Старые окна интерфейса обновлены

Разработанный программный код см. в приложении А.

UML-диаграмма.



Выводы.

В ходе работы был создан шаблонный класс врага. Изучен паттерн Состояния.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: enemy.h

```
#pragma once

#include "character.h"
#include "player.h"
#include "../Field/field.h"

class Field;

template <class T>
class Enemy : public Character
{
private:
T* policy;

void moveChaser()
{
Field* field = field->GetInstance();
Player* player = field->GetPlayer();
int width = field->GetWidth();
int height = field->GetHeight();
int** pathfinder = new int* [height];
for (int i = 0; i < height; i++)
{
pathfinder[i] = new int[width];
for(int j = 0; j < width; j++)
{
if (field->IsWall(j, i))
pathfinder[i][j] = -2;
else
pathfinder[i][j] = -1;
}
}
pathfinder[y][x] = 0;
pathfinder[player->GetY()][player->GetX()] = -3;
int wave = 0;
bool foundEnd = false;
bool foundEmpty = false;
while (!foundEnd)
{
foundEmpty = false;
for (int i = 0; i < height; i++)
{
for (int j = 0; j < width; j++)
{
if (pathfinder[i][j] == wave)
{
if (i < height - 1)
{
if (pathfinder[i + 1][j] == -3)
{
pathfinder[i + 1][j] = wave + 1;
foundEnd = true;
```

```
break;
}
else if (pathfinder[i + 1][j] == -1)
{
pathfinder[i + 1][j] = wave + 1;
foundEmpty = true;
}
}

if (i > 0)
{
if (pathfinder[i - 1][j] == -3)
{
pathfinder[i - 1][j] = wave + 1;
foundEnd = true;
break;
}
else if (pathfinder[i - 1][j] == -1)
{
pathfinder[i - 1][j] = wave + 1;
foundEmpty = true;
}
}

if (j < width - 1)
{
if (pathfinder[i][j + 1] == -3)
{
pathfinder[i][j + 1] = wave + 1;
foundEnd = true;
break;
}
else if (pathfinder[i][j + 1] == -1)
{
pathfinder[i][j + 1] = wave + 1;
foundEmpty = true;
}
}

if (j > 0)
{
if (pathfinder[i][j - 1] == -3)
{
pathfinder[i][j - 1] = wave + 1;
foundEnd = true;
break;
}
else if (pathfinder[i][j - 1] == -1)
{
pathfinder[i][j - 1] = wave + 1;
foundEmpty = true;
}
}
}
}
}
if (!foundEnd && !foundEmpty)
break;
```

```
      wave++;
      }
      int curX = player->GetX();
      int curY = player->GetY();
      bool isMoved = false;
      int destinationX = 0;
      int destinationY = 0;
      while (curX != x || curY != y)
      {
      isMoved = false;
      for (int dy = -1; dy <= 1 && !isMoved; dy++) {
      for (int dx = -1; dx <= 1 && !isMoved; dx++) {
      if (abs(dy) + abs(dx) != 1 || curX + dx > width || curY + dy > height
|| curX + dx < 0 || curY + dy < 0)
      continue;
      if (pathfinder[curY + dy][curX + dx] < 0)
      continue;
      if (pathfinder[curY + dy][curX + dx] == pathfinder[curY][curX] - 1)
      {
      curX += dx;
      curY += dy;
      destinationX = -1 * dx;
      destinationY = -1 * dy;
      isMoved = true;
      break;
      }
      }
      }
      }
      if(!field->IsEnemySet(x   +   destinationX,   y   +   destinationY)
&& !(field->IsStartOrEnd(player->GetX(), player->GetY()) && player->GetX()
== x + destinationX && player->GetY() == y + destinationY))
      Move(destinationX, destinationY);

      for (int i = 0; i < height; i++) {
      delete[] pathfinder[i];
      }
      delete[] pathfinder;
      }

      void moveAmbusher()
      {
      Field* field = field->GetInstance();
      Player* player = field->GetPlayer();
      if  (player->GetX()  >=  x  -  1  &&  player->GetX()  <=  x  +  1  &&
player->GetY() >= y - 1 && player->GetY() <= y + 1)
      {
      if (!(field->IsStartOrEnd(player->GetX(), player->GetY())))
      Move(player->GetX() - x, player->GetY() - y);
      return;
      }
      int marginX;
      int marginY;
      if (player->GetX() < field->GetWidth() / 2)
      marginX = 0;
      else
      marginX = 1;
```

```
     if (player->GetY() < field->GetHeight() / 2)
     marginY = 0;
     else
     marginY = 1;

     int newX = rand() % (field->GetWidth() / 2) + marginX *
field->GetWidth() / 2;
     int newY = rand() % (field->GetHeight() / 2) + marginY *
field->GetHeight() / 2;
     while (field->IsWall(newX, newY) || field->IsEnemySet(newX, newY) ||
field->IsPlayerSet(newX, newY))
     {
     newX = rand() % (field->GetWidth() / 2) + marginX * field->GetWidth()
/ 2;
     newY = rand() % (field->GetHeight() / 2) + marginY *
field->GetHeight() / 2;
     }
     Move(newX - x, newY - y);
     }

     void moveJumper()
     {
     Field* field = field->GetInstance();
     Player* player = field->GetPlayer();
     if (player->GetX() == x + 1 && player->GetY() == y + 2)
     {
     if (!(field->IsStartOrEnd(player->GetX(), player->GetY())))
     Move(1, 2);
     return;
     }
     if (player->GetX() == x + 2 && player->GetY() == y + 1)
     {
     if (!(field->IsStartOrEnd(player->GetX(), player->GetY())))
     Move(2, 1);
     return;
     }
     if (player->GetX() == x - 1 && player->GetY() == y + 2)
     {
     if (!(field->IsStartOrEnd(player->GetX(), player->GetY())))
     Move(-1, 2);
     return;
     }
     if (player->GetX() == x - 2 && player->GetY() == y + 1)
     {
     if (!(field->IsStartOrEnd(player->GetX(), player->GetY())))
     Move(-2, 1);
     return;
     }
     if (player->GetX() == x + 2 && player->GetY() == y - 1)
     {
     if (!(field->IsStartOrEnd(player->GetX(), player->GetY())))
     Move(2, -1);
     return;
     }
     if (player->GetX() == x + 1 && player->GetY() == y - 2)
     {
     if (!(field->IsStartOrEnd(player->GetX(), player->GetY())))
     Move(1, -2);
```

```
return;
}
if (player->GetX() == x - 1 && player->GetY() == y - 2)
{
if (!(field->IsStartOrEnd(player->GetX(), player->GetY())))
Move(-1, -2);
return;
}
if (player->GetX() == x - 2 && player->GetY() == y - 1)
{
if (!(field->IsStartOrEnd(player->GetX(), player->GetY())))
Move(-2, -1);
return;
}

if (player->GetY() >= y && player->GetX() >= x)
{
if (!field->IsWall(x + 1, y + 2))
{
if (!field->IsEnemySet(x + 1, y + 2))
{
Move(1, 2);
return;
}
}
if (!field->IsWall(x + 2, y + 1))
{
if (!field->IsEnemySet(x + 2, y + 1))
{
Move(2, 1);
return;
}
}
if (!field->IsWall(x - 1, y + 2))
{
if (!field->IsEnemySet(x - 1, y + 2))
{
Move(-1, 2);
return;
}
}
if (!field->IsWall(x + 2, y - 1))
{
if (!field->IsEnemySet(x + 2, y - 1))
{
Move(2, -1);
return;
}
}
if (!field->IsWall(x - 2, y + 1))
{
if (!field->IsEnemySet(x - 2, y + 1))
{
Move(-2, 1);
return;
}
}
if (!field->IsWall(x + 1, y - 2))
```

```
{
if (!field->IsEnemySet(x + 1, y - 2))
{
Move(1, -2);
return;
}
}
if (!field->IsWall(x - 2, y - 1))
{
if (!field->IsEnemySet(x - 2, y - 1))
{
Move(-2, -1);
return;
}
}
if (!field->IsWall(x - 1, y - 2))
{
if (!field->IsEnemySet(x - 1, y - 2))
{
Move(-1, -2);
return;
}
}
}
else if (player->GetY() >= y && player->GetX() < x)
{
if (!field->IsWall(x - 1, y + 2))
{
if (!field->IsEnemySet(x - 1, y + 2))
{
Move(-1, 2);
return;
}
}
if (!field->IsWall(x - 2, y + 1))
{
if (!field->IsEnemySet(x - 2, y + 1))
{
Move(-2, 1);
return;
}
}
if (!field->IsWall(x + 1, y + 2))
{
if (!field->IsEnemySet(x + 1, y + 2))
{
Move(1, 2);
return;
}
}
if (!field->IsWall(x - 2, y - 1))
{
if (!field->IsEnemySet(x - 2, y - 1))
{
Move(-2, -1);
return;
}
}
```

```
if (!field->IsWall(x + 2, y + 1))
{
if (!field->IsEnemySet(x + 2, y + 1))
{
Move(2, 1);
return;
}
}
if (!field->IsWall(x - 1, y - 2))
{
if (!field->IsEnemySet(x - 1, y - 2))
{
Move(-1, -2);
return;
}
}
if (!field->IsWall(x + 2, y - 1))
{
if (!field->IsEnemySet(x + 2, y - 1))
{
Move(2, -1);
return;
}
}
if (!field->IsWall(x + 1, y - 2))
{
if (!field->IsEnemySet(x + 1, y - 2))
{
Move(1, -2);
return;
}
}
}
else if (player->GetY() < y && player->GetX() >= x)
{
if (!field->IsWall(x + 1, y - 2))
{
if (!field->IsEnemySet(x + 1, y - 2))
{
Move(1, -2);
return;
}
}
if (!field->IsWall(x + 2, y - 1))
{
if (!field->IsEnemySet(x + 2, y - 1))
{
Move(2, -1);
return;
}
}
if (!field->IsWall(x - 1, y - 2))
{
if (!field->IsEnemySet(x - 1, y - 2))
{
Move(-1, -2);
return;
}
```

```
}
if (!field->IsWall(x + 2, y + 1))
{
if (!field->IsEnemySet(x + 2, y + 1))
{
Move(2, 1);
return;
}
}
if (!field->IsWall(x - 2, y - 1))
{
if (!field->IsEnemySet(x - 2, y - 1))
{
Move(-2, -1);
return;
}
}
if (!field->IsWall(x + 1, y + 2))
{
if (!field->IsEnemySet(x + 1, y + 2))
{
Move(1, 2);
return;
}
}
if (!field->IsWall(x - 2, y + 1))
{
if (!field->IsEnemySet(x - 2, y + 1))
{
Move(-2, 1);
return;
}
}
if (!field->IsWall(x - 1, y + 2))
{
if (!field->IsEnemySet(x - 1, y + 2))
{
Move(-1, 2);
return;
}
}
}
else if (player->GetY() < y && player->GetX() < x)
{
if (!field->IsWall(x - 1, y - 2))
{
if (!field->IsEnemySet(x - 1, y - 2))
{
Move(-1, -2);
return;
}
}
if (!field->IsWall(x - 2, y - 1))
{
if (!field->IsEnemySet(x - 2, y - 1))
{
Move(-2, -1);
return;
```

```
}
}
if (!field->IsWall(x + 1, y - 2))
{
if (!field->IsEnemySet(x + 1, y - 2))
{
Move(1, -2);
return;
}
}
if (!field->IsWall(x - 2, y + 1))
{
if (!field->IsEnemySet(x - 2, y + 1))
{
Move(-2, 1);
return;
}
}
if (!field->IsWall(x + 2, y - 1))
{
if (!field->IsEnemySet(x + 2, y - 1))
{
Move(2, -1);
return;
}
}
if (!field->IsWall(x - 1, y + 2))
{
if (!field->IsEnemySet(x - 1, y + 2))
{
Move(-1, 2);
return;
}
}
if (!field->IsWall(x + 2, y + 1))
{
if (!field->IsEnemySet(x + 2, y + 1))
{
Move(2, 1);
return;
}
}
if (!field->IsWall(x + 1, y + 2))
{
if (!field->IsEnemySet(x + 1, y + 2))
{
Move(1, 2);
return;
}
}
}
}

public:
Enemy(int x, int y)
{
this->x = x;
this->y = y;
```

```
this->policy = new T;
}

T* GetPolicy()
{
return policy;
}

void MakeMove()
{
switch (policy->GetIndex())
{
case 0:
moveChaser();
break;
case 1:
moveAmbusher();
break;
case 2:
moveJumper();
break;
default:
break;
}
}

~Enemy()
{
if (policy)
delete policy;
}
};
```

Название файла: policy.h

```
#pragma once

#include "../player.h"

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

Название файла: policyChaser.h

```
#pragma once

#include "policy.h"

class PolicyChaser : public Policy
{
public:
    int GetIndex();
```

```
    void operator+(Player* player);
};
```

Название файла: policyChaser.cpp

```
#include "policyChaser.h"

int PolicyChaser::GetIndex()
{
return 0;
}
void PolicyChaser::operator+(Player* player)
{
player->TakeDamage(2);
}
```

Название файла: policyAmbusher.h

```
#pragma once

#include "policy.h"

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

Название файла: policyAmbusher.cpp

```
#include "policyAmbusher.h"

int PolicyAmbusher::GetIndex()
{
return 1;
}
void PolicyAmbusher::operator+(Player* player)
{
player->TakeDamage(3);
}
```

Название файла: policyJumper.h

```
#pragma once

#include "policy.h"

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

Название файла: policyJumper.cpp

```
#include "policyJumper.h"

int PolicyJumper::GetIndex()
{
return 2;
}
void PolicyJumper::operator+(Player* player)
{
player->TakeDamage(1);
}
```

Название файла: cell.h

```
#pragma once

#include "../Items/item.h"
#include "../Characters/player.h"
#include "../Characters/Policies/policyChaser.h"
#include "../Characters/Policies/policyAmbusher.h"
#include "../Characters/Policies/policyJumper.h"

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

template <class T>
class Enemy;

class Cell
{
private:
    CellType cellType;
    bool isPlayerSet;
    bool isItemSet;
    bool isEnemySet;
    Player* player = nullptr;
    Item* item = nullptr;
    Enemy<PolicyChaser>* chaser = nullptr;
    Enemy<PolicyAmbusher>* ambusher = nullptr;
    Enemy<PolicyJumper>* jumper = 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();
    bool IsEnemySet();
    void PlaceChaser(Enemy<PolicyChaser>* enemy);
    void RemoveChaser();
    void PlaceAmbusher(Enemy<PolicyAmbusher>* enemy);
    void RemoveAmbusher();
    void PlaceJumper(Enemy<PolicyJumper>* enemy);
    void RemoveJumper();
    int GetEnemyIndex();
    ~Cell();
};
```

Название файла: cell.cpp

```
#include "cell.h"

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

bool Cell::IsEnemySet()
{
    return isEnemySet;
}

void Cell::PlaceChaser(Enemy<PolicyChaser>* enemy)
{
    isEnemySet = true;
    this->chaser = enemy;
}

void Cell::RemoveChaser()
{
    isEnemySet = false;
    this->chaser = nullptr;
}

void Cell::PlaceAmbusher(Enemy<PolicyAmbusher>* enemy)
{
    isEnemySet = true;
    this->ambusher = enemy;
}

void Cell::RemoveAmbusher()
{
    isEnemySet = false;
    this->ambusher = nullptr;
}

void Cell::PlaceJumper(Enemy<PolicyJumper>* enemy)
{
    isEnemySet = true;
    this->jumper = enemy;
}

void Cell::RemoveJumper()
{
    isEnemySet = false;
```

```
    this->jumper = nullptr;
}

int Cell::GetEnemyIndex()
{
    if (chaser)
        return 0;
    if (ambusher)
        return 1;
    return 2;
}

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

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"
#include "../Characters/enemy.h"

class GameManager
{
private:
Field* field;
Player* player;
UserInterface* userInterface;
Enemy<PolicyChaser>* chaser;
Enemy<PolicyAmbusher>* ambusher;
Enemy<PolicyJumper>* jumper;
int numOfItemPoint;
int numOfItemHealth;
int numOfItemEnergy;
int pointsToWin;
std::pair<int, int> randomCell();
void setGameObjects();
void openNew();
void parseMove();
void openHelp(int back);
void openRules(int back);
void openItems(int back);
void openEnemies(int back);
void openControls(int back);
void openWin();
```

```
void openLose(int count);
void openPause();
void nextMove(int dx, int dy);
public:
GameManager();
void OpenMenu();
~GameManager();
};
```

Название файла: gameManager.cpp

```
#include "gameManager.h"

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

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) || field->IsEnemySet(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;
      randPos = randomCell();
      chaser = new Enemy<PolicyChaser>(randPos.first, randPos.second);
      field->GetField()[randPos.second][randPos.first].PlaceChaser(chaser
);
      randPos = randomCell();
      ambusher = new Enemy<PolicyAmbusher>(randPos.first, randPos.second);
      field->GetField()[randPos.second][randPos.first].PlaceAmbusher(ambu
sher);
      randPos = randomCell();
      jumper = new Enemy<PolicyJumper>(randPos.first, randPos.second);
      field->GetField()[randPos.second][randPos.first].PlaceJumper(jumper
);
      }

      void GameManager::OpenMenu()
      {
      userInterface->PrintMenu();
      std::string userCommand = userInterface->ScanCommand();
      if (userCommand == "n" || userCommand == "new")
      openNew();
      else if (userCommand == "h" || userCommand == "help")
      openHelp(0);
      else if (userCommand != "q" && userCommand != "quit")
      {
      std::cout << "Invalid command!\n";
      OpenMenu();
      }
      }

      void GameManager::openNew()
      {
      if (player)
      delete player;
      if (field)
      field->DeleteField();
      if (chaser)
      delete chaser;
      if (ambusher)
      delete ambusher;
      if (jumper)
      delete jumper;
      field = Field::GetInstance();
      player         =         new         Player(field->GetStart().first,
field->GetStart().second);
      player->SetLogPlayer(new LogPlayer(player));
      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();
	player->GetLogPlayer()->GameStarts(pointsToWin);
	parseMove();
	}

	void GameManager::parseMove()
	{
	if (field->IsItemSet(chaser->GetX(), chaser->GetY())) {
	std::cout << "Enemy Chaser(]:) is contesting item ";
	switch (field->GetItem(chaser->GetX(), chaser->GetY())->GetIndex())
{
	case 0:
	std::cout << "Health(<3)\n";
	break;
	case 1:
	std::cout << "Energy(~@)\n";
	break;
	case 2:
	std::cout << "Point({+)\n";
	break;
	default:
	break;
	}
	}
	if (field->IsItemSet(ambusher->GetX(), ambusher->GetY())) {
	std::cout << "Enemy Ambusher(S:) is contesting item ";
	switch                              (field->GetItem(ambusher->GetX(),
ambusher->GetY())->GetIndex()) {
	case 0:
	std::cout << "Health(<3)\n";
	break;
	case 1:
	std::cout << "Energy(~@)\n";
	break;
	case 2:
	std::cout << "Point({+)\n";
	break;
	default:
	break;
	}
	}
	if (field->IsItemSet(jumper->GetX(), jumper->GetY())) {
	std::cout << "Enemy Jumper(D:) is contesting item ";
	switch (field->GetItem(jumper->GetX(), jumper->GetY())->GetIndex())
{
	case 0:
	std::cout << "Health(<3)\n";
	break;
	case 1:
	std::cout << "Energy(~@)\n";
	break;
	case 2:
	std::cout << "Point({+)\n";
	break;
	default:
```

```
      break;
      }
      }
      if (field->IsStartOrEnd(chaser->GetX(), chaser->GetY()))
      std::cout << "Enemy Chaser(]:) is contesting spawn(->) or
exit(>>)\n";
      if (field->IsStartOrEnd(ambusher->GetX(), ambusher->GetY()))
      std::cout << "Enemy Ambusher(S:) is contesting spawn(->) or
exit(>>)\n";
      if (field->IsStartOrEnd(jumper->GetX(), jumper->GetY()))
      std::cout << "Enemy Jumper(D:) is contesting spawn(->) or
exit(>>)\n";
      userInterface->PrintGame(player, pointsToWin);
      std::string userCommand = userInterface->ScanCommand();
      if (userCommand == "p" || userCommand == "pause")
      openPause();
      else 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::openHelp(int back)
      {
      userInterface->PrintHelp();
      std::string userCommand = userInterface->ScanCommand();
      if (userCommand == "r" || userCommand == "rules")
      openRules(back);
      else if (userCommand == "c" || userCommand == "controls")
      openControls(back);
      else if (userCommand == "b" || userCommand == "back") {
      if (back == 0)
      OpenMenu();
      else
      openPause();
      }
      else if (userCommand != "q" && userCommand != "quit")
```

```
{
std::cout << "Invalid command!\n";
openHelp(back);
}
}

void GameManager::openRules(int back)
{
userInterface->PrintRules();
std::string userCommand = userInterface->ScanCommand();
if (userCommand == "b" || userCommand == "back")
openHelp(back);
else if (userCommand == "i" || userCommand == "items")
openItems(back);
else if (userCommand == "e" || userCommand == "enemies")
openEnemies(back);
else if (userCommand != "q" && userCommand != "quit")
{
std::cout << "Invalid command!\n";
openRules(back);
}
}

void GameManager::openItems(int back)
{
userInterface->PrintItems();
std::string userCommand = userInterface->ScanCommand();
if (userCommand == "b" || userCommand == "back")
openRules(back);
else if (userCommand != "q" && userCommand != "quit")
{
std::cout << "Invalid command!\n";
openItems(back);
}
}

void GameManager::openEnemies(int back)
{
userInterface->PrintEnemies();
std::string userCommand = userInterface->ScanCommand();
if (userCommand == "b" || userCommand == "back")
openRules(back);
else if (userCommand != "q" && userCommand != "quit")
{
std::cout << "Invalid command!\n";
openEnemies(back);
}
}

void GameManager::openControls(int back)
{
userInterface->PrintControls();
std::string userCommand = userInterface->ScanCommand();
if (userCommand == "b" || userCommand == "back")
openHelp(back);
else if (userCommand != "q" && userCommand != "quit")
{
std::cout << "Invalid command!\n";
```

```
openControls(back);
}
}

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

void GameManager::openLose(int count)
{
userInterface->PrintLose();
std::string userCommand = userInterface->ScanCommand();
if (userCommand == "m" || userCommand == "menu" || userCommand ==
"<<")
OpenMenu();
else if (userCommand == "n" || userCommand == "new" || userCommand
== ">>")
openNew();
else if (userCommand != "q" && userCommand != "quit")
{
std::cout << "Invalid command!\n";
count++;
if (count == 50)
{
openWin();
return;
}
openLose(count);
}
}

void GameManager::openPause()
{
userInterface->PrintPause();
std::string userCommand = userInterface->ScanCommand();
if (userCommand == "b" || userCommand == "back")
parseMove();
else if (userCommand == "h" || userCommand == "help")
openHelp(1);
else if (userCommand == "m" || userCommand == "menu")
OpenMenu();
else if (userCommand != "q" && userCommand != "quit")
{
std::cout << "Invalid command!\n";
openPause();
```

```
}
}

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();
player->GetLogPlayer()->PlayerJumps();
}
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);
player->GetLogPlayer()->PlayerMoves();
if (field->IsItemSet(player->GetX(), player->GetY()))
{
*(field->GetItem(player->GetX(), player->GetY())) + player;
player->GetLogPlayer()->PlayerCollects(field->GetItem(player->GetX(
), player->GetY()), pointsToWin);
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();
}
bool chaserMove = true;
bool ambusherMove = true;
bool jumperMove = true;
if (player->GetX() == chaser->GetX() && player->GetY() ==
chaser->GetY())
{
chaserMove = false;
*(chaser->GetPolicy()) + player;
```

```
      player->GetLogPlayer()->PlayerTakesDamage();
      std::pair<int, int> randPos = randomCell();
      int newX = randPos.first;
      int newY = randPos.second;
      field->GetField()[chaser->GetY()][chaser->GetX()].RemoveChaser();
      chaser->Move(newX - chaser->GetX(), newY - chaser->GetY());
      field->GetField()[chaser->GetY()][chaser->GetX()].PlaceChaser(chase
r);
      }
      if (player->GetX() == ambusher->GetX() && player->GetY() ==
ambusher->GetY())
      {
      ambusherMove = false;
      *(ambusher->GetPolicy()) + player;
      player->GetLogPlayer()->PlayerTakesDamage();
      std::pair<int, int> randPos = randomCell();
      int newX = randPos.first;
      int newY = randPos.second;
      field->GetField()[ambusher->GetY()][ambusher->GetX()].RemoveAmbushe
r();
      ambusher->Move(newX - ambusher->GetX(), newY - ambusher->GetY());
      field->GetField()[ambusher->GetY()][ambusher->GetX()].PlaceAmbusher
(ambusher);
      }
      if (player->GetX() == jumper->GetX() && player->GetY() ==
jumper->GetY())
      {
      jumperMove = false;
      *(jumper->GetPolicy()) + player;
      player->GetLogPlayer()->PlayerTakesDamage();
      std::pair<int, int> randPos = randomCell();
      int newX = randPos.first;
      int newY = randPos.second;
      field->GetField()[jumper->GetY()][jumper->GetX()].RemoveJumper();
      jumper->Move(newX - jumper->GetX(), newY - jumper->GetY());
      field->GetField()[jumper->GetY()][jumper->GetX()].PlaceJumper(jumpe
r);
      }
      int startX = field->GetStart().first;
      int startY = field->GetStart().second;
      if (ambusherMove)
      {
      field->GetField()[ambusher->GetY()][ambusher->GetX()].RemoveAmbushe
r();
      ambusher->MakeMove();
      field->GetField()[ambusher->GetY()][ambusher->GetX()].PlaceAmbusher
(ambusher);
      }
      if (player->GetX() == ambusher->GetX() && player->GetY() ==
ambusher->GetY())
      {
      *(ambusher->GetPolicy()) + player;
      player->GetLogPlayer()->PlayerTakesDamage();
      field->GetField()[player->GetY()][player->GetX()].RemovePlayer();
      player->Move(startX - player->GetX(), startY - player->GetY());
      field->GetField()[player->GetY()][player->GetX()].PlacePlayer(playe
r);
      player->GetLogPlayer()->PlayerMoves();
```

```
	}
	if (chaserMove)
	{
	field->GetField()[chaser->GetY()][chaser->GetX()].RemoveChaser();
	chaser->MakeMove();
	field->GetField()[chaser->GetY()][chaser->GetX()].PlaceChaser(chase
r);
	}
	if (player->GetX() == chaser->GetX() && player->GetY() ==
chaser->GetY())
	{
	*(chaser->GetPolicy()) + player;
	player->GetLogPlayer()->PlayerTakesDamage();
	field->GetField()[player->GetY()][player->GetX()].RemovePlayer();
	player->Move(startX - player->GetX(), startY - player->GetY());
	field->GetField()[player->GetY()][player->GetX()].PlacePlayer(playe
r);
	player->GetLogPlayer()->PlayerMoves();

	}
	if (jumperMove)
	{
	field->GetField()[jumper->GetY()][jumper->GetX()].RemoveJumper();
	jumper->MakeMove();
	field->GetField()[jumper->GetY()][jumper->GetX()].PlaceJumper(jumpe
r);
	}
	if (player->GetX() == jumper->GetX() && player->GetY() ==
jumper->GetY())
	{
	*(jumper->GetPolicy()) + player;
	player->GetLogPlayer()->PlayerTakesDamage();
	field->GetField()[player->GetY()][player->GetX()].RemovePlayer();
	player->Move(startX - player->GetX(), startY - player->GetY());
	field->GetField()[player->GetY()][player->GetX()].PlacePlayer(playe
r);
	player->GetLogPlayer()->PlayerMoves();
	}
	if (player->GetHealth() <= 0)
	{
	player->GetLogPlayer()->GameEnds();
	openLose(0);
	return;
	}
	if (field->GetField()[player->GetY()][player->GetX()].GetCellType()
== CellType::END)
	{
	if (player->GetPoints() == pointsToWin)
	{
	player->GetLogPlayer()->GameEnds();
	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();
if (userInterface)
delete userInterface;
if (chaser)
delete chaser;
if (ambusher)
delete ambusher;
if (jumper)
delete jumper;
}
```