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

ОТЧЕТ

по лабораторной работе №6

по дисциплине «ООП»

Тема: Сохранение и загрузка / Написание исключений

Студент гр. 9383 Моисейченко К.А.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Изучить паттерн Снимок. Добавить возможность сохранить и загрузить игру из файла. Написать исключения для проверки файлов на корректность.

Задание.

Создать классы, которые позволяют сохранить игру, а потом загрузить ее. Также, написать набор исключений, которые как минимум позволяют контролировать процесс сохранения и загрузки

Обязательные требования:

- Игру можно сохранить в файл
- Игру можно загрузить из файла
- Взаимодействие с файлами по идиоме RAII
- Добавлена проверка файлов на корректность
- Написаны исключения, которые обеспечивают транзакционность

Дополнительные требования:

- Для получения состояния программы используется паттерн Снимок

Выполнение работы.

Класс Exception:

Класс является общим представлением класса ошибки. Хранит в себе сообщение ошибки и имеет виртуальный метод вызова ошибки

Классы ExceptionFile, ExceptionLoad:

ExceptionLoad — исключения, возникающие при работе с файлом: проблемы с его открытием, записью или закрытием

ExceptionLoad — исключения, возникающие при попытке загрузки игры. Возникает при повреждении файла сохранения

Класс SaveFile:

Класс является интерфейсом для работы с файлами по идиоме RAII.

Класс записывает или считывает информацию, хранящуюся в файле сохранений. В исключительных ситуациях срабатывает исключение FileException

Класс Memento:

Класс является реализацией сохранения паттерна Снимок. Метод GetState() возвращает данные о состоянии игрового поля — объект класса GameState

Класс GameState:

Хранит данные о состоянии игрового поля, у класса перегружены потоки ввода и вывода

Поля:

+ int numOfItemPoint
+ int numOfItemHealth
+ int numOfItemEnergy
+ int pointsToWin
+ int playerPoints
+ int playerHealth
+ int playerEnergy
+ pair<int, int> playerPos
+ pair<int, int> chaserPos
+ pair<int, int> ambusherPos
+ pair<int, int> jumperPos
+ pair<int, int>* itemPointPos
+ pair<int, int>* itemHealthPos
+ pair<int, int>* itemEnergyPos

Класс Caretaker:

Класс отвечает за реализацию опекуна в паттерне Снимок — класса, который занимается сохранением и восстановлением состояния игрового поля. Класс хранит в себе указатели на объекты GameManager и SaveFile. Метод Save()

записывает в файл состояние игрового поля, а метод Load() восстанавливает поле по информации из файла сохранения. При этом происходит обработка исключений, связанных, как с работой с файлом — FileException, так и с восстановлением сохранений — LoadException.

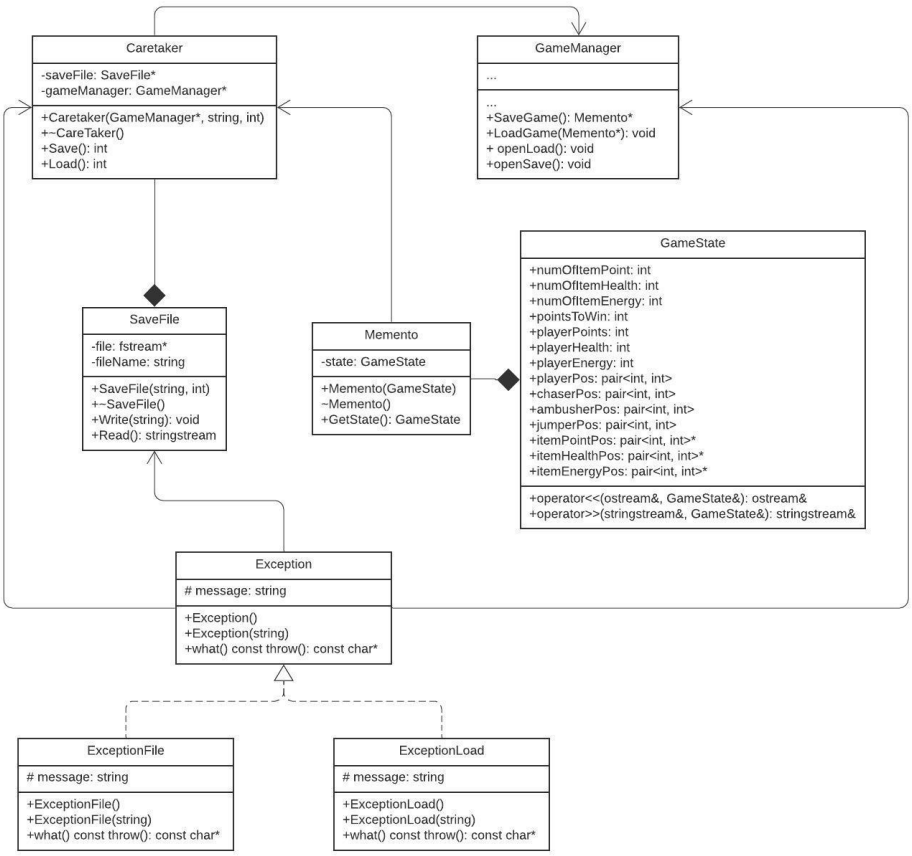
Класс GameManager:

Добавлены методы, позволяющие пользователю сохранить или загрузить игру через интерфейс приложения

Добавлены методы +Memento* SaveGame() и +void LoadGame(Memento*), которые переносят или вносят информацию о поле в/из объекта класса Memento.

Разработанный программный код см. в приложении А.

UML-диаграмма.



Выводы.

Был изучен паттерн Снимок. В игра была добавлена возможность сохранить и загрузить игру из файла и были написаны исключения для

проверки файлов на корректность.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: exception.h

```
#pragma once

#include <string>

class Exception
{
protected:
std::string message = "\n";
public:
Exception() = default;
Exception(const std::string message);
virtual const char* what() const throw() = 0;
};
```

Название файла: exceptionFile.h

```
#pragma once

#include "exception.h"

class ExceptionFile : public Exception
{
public:
ExceptionFile() = default;
ExceptionFile(const std::string message);
const char* what() const throw();
};
```

Название файла: exceptionFile.cpp

```
#include "exceptionFile.h"

ExceptionFile::ExceptionFile(const std::string message)
{
this->message = message;
}

const char* ExceptionFile::what() const throw()
{
std::string answer = "File error; " + message;
return answer.c_str();
}
```

Название файла: exceptionLoad.h

```
#pragma once

#include "exception.h"

class ExceptionLoad : public Exception
{
```

```
public:
ExceptionLoad() = default;
ExceptionLoad(const std::string message);
const char* what() const throw();
};
```

Название файла: exceptionLoad.cpp

```
#include "exceptionLoad.h"

ExceptionLoad::ExceptionLoad(const std::string message)
{
this->message = message;
}

const char* ExceptionLoad::what() const throw()
{
std::string answer = "Load error; " + message;
return answer.c_str();
}
```

Название файла: saveFile.h

```
#pragma once

#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include "Exceptions/ExceptionFile.h"
#include "Exceptions/ExceptionLoad.h"

class SaveFile
{
private:
std::fstream* file;
std::string fileName;
public:
SaveFile(std::string fileName, int mode);
void Write(std::string s);
std::stringstream Read();
~SaveFile();
};
```

Название файла: saveFile.cpp

```
#include "saveFile.h"

SaveFile::SaveFile(std::string fileName, int mode)
{
this->fileName = fileName;
file = new std::fstream;
switch (mode) {
case 0:
file->open(fileName.c_str());
return;
```

```
	case 1:
	file->open(fileName.c_str(), std::fstream::in | std::fstream::out |
std::fstream::trunc);
	return;
	default:
	return;
	}
	if (!file->is_open())
	throw ExceptionFile("Can't open the file.\n");
	}

	void SaveFile::Write(std::string s)
	{
	*file << s;
	}

	std::stringstream SaveFile::Read()
	{
	std::stringstream loadStream;
	loadStream << file->rdbuf();
	return loadStream;
	}

	SaveFile::~SaveFile()
	{
	if (file->is_open())
	file->close();
	if(file)
	delete file;
	}
```

Название файла: memento.h

```
	#pragma once

	#include "gameState.h"

	class Memento
	{
	private:
	GameState state;
	public:
	Memento(GameState state);
	GameState GetState();
	~Memento();
	};
```

Название файла: memento.cpp

```
	#include "memento.h"

	Memento::Memento(GameState state)
	{
	this->state = state;
	}
```

```
GameState Memento::GetState()
{
return state;
}

Memento::~Memento()
{
if (state.itemPointPos)
delete[] state.itemPointPos;
if (state.itemHealthPos)
delete[] state.itemHealthPos;
if (state.itemEnergyPos)
delete[] state.itemEnergyPos;
}
```

Название файла: gameState.h

```
#pragma once

#include <utility>
#include <sstream>
#include "Exceptions/ExceptionLoad.h"

class GameState
{
public:
int numOfItemPoint;
int numOfItemHealth;
int numOfItemEnergy;
int pointsToWin;
int playerPoints;
int playerHealth;
int playerEnergy;
std::pair<int, int> playerPos;
std::pair<int, int> chaserPos;
std::pair<int, int> ambusherPos;
std::pair<int, int> jumperPos;
std::pair<int, int>* itemPointPos = nullptr;
std::pair<int, int>* itemHealthPos = nullptr;
std::pair<int, int>* itemEnergyPos = nullptr;

friend std::ostream& operator<<(std::ostream& out, const GameState&
gameState);
friend   std::stringstream&   operator>>(std::stringstream&   input,
GameState& gameState);
};
```

Название файла: gameState.cpp

```
#include "gameState.h"

std::ostream&  operator<<  (std::ostream&  out,  const  GameState&
gameState)
{
out << gameState.numOfItemPoint << "\n";
out << gameState.numOfItemHealth << "\n";
```

```
    out << gameState.numOfItemEnergy << "\n";
    out << gameState.pointsToWin << "\n";
    out << gameState.playerPoints << "\n";
    out << gameState.playerHealth << "\n";
    out << gameState.playerEnergy << "\n";
    out << gameState.playerPos.first << " " << gameState.playerPos.second
<< "\n";
    out << gameState.chaserPos.first << " " << gameState.chaserPos.second
<< "\n";
    out     <<     gameState.ambusherPos.first     <<     "     "     <<
gameState.ambusherPos.second << "\n";
    out << gameState.jumperPos.first << " " << gameState.jumperPos.second
<< "\n";
    for (int i = 0; i < gameState.numOfItemPoint; i++)
    out    <<    gameState.itemPointPos[i].first    <<    "    "    <<
gameState.itemPointPos[i].second << "\n";
    for (int i = 0; i < gameState.numOfItemHealth; i++)
    out    <<    gameState.itemHealthPos[i].first    <<    "    "    <<
gameState.itemHealthPos[i].second << "\n";
    for (int i = 0; i < gameState.numOfItemEnergy; i++)
    out    <<    gameState.itemEnergyPos[i].first    <<    "    "    <<
gameState.itemEnergyPos[i].second << "\n";
    return out;
    }

    std::stringstream& operator>> (std::stringstream& input, GameState&
gameState)
    {
    input >> gameState.numOfItemPoint;
    if (gameState.numOfItemPoint < 0)
    throw ExceptionLoad("Load file is corrupted.\n");
    input >> gameState.numOfItemHealth;
    if (gameState.numOfItemHealth < 0)
    throw ExceptionLoad("Load file is corrupted.\n");
    input >> gameState.numOfItemEnergy;
    if (gameState.numOfItemEnergy < 0)
    throw ExceptionLoad("Load file is corrupted.\n");
    input >> gameState.pointsToWin;
    input >> gameState.playerPoints;
    input >> gameState.playerHealth;
    input >> gameState.playerEnergy;
    input >> gameState.playerPos.first >> gameState.playerPos.second;
    input >> gameState.chaserPos.first >> gameState.chaserPos.second;
    input          >>          gameState.ambusherPos.first          >>
gameState.ambusherPos.second;
    input >> gameState.jumperPos.first >> gameState.jumperPos.second;
    gameState.itemPointPos         =         new         std::pair<int,
int>[gameState.numOfItemPoint];
    for (int i = 0; i < gameState.numOfItemPoint; i++)
    input         >>         gameState.itemPointPos[i].first         >>
gameState.itemPointPos[i].second;
    gameState.itemHealthPos         =         new         std::pair<int,
int>[gameState.numOfItemHealth];
    for (int i = 0; i < gameState.numOfItemHealth; i++)
    input         >>         gameState.itemHealthPos[i].first         >>
gameState.itemHealthPos[i].second;
    gameState.itemEnergyPos         =         new         std::pair<int,
int>[gameState.numOfItemEnergy];
```

```
	for (int i = 0; i < gameState.numOfItemEnergy; i++)
	input >> gameState.itemEnergyPos[i].first >>
gameState.itemEnergyPos[i].second;
	return input;
	}
```

Название файла: caretaker.h

```
	#pragma once

	#include <string>
	#include <sstream>
	#include "../GameManager/gameManager.h"
	#include "memento.h"
	#include "saveFile.h"

	class GameManager;

	class Caretaker
	{
	private:
	SaveFile* saveFile;
	GameManager* gameManager;
	public:
	Caretaker(GameManager* gameManager, std::string fileName, int mode);
	int Save();
	int Load();
	~Caretaker();
	};
```

Название файла: caretaker.cpp

```
	#include "caretaker.h"

	Caretaker::Caretaker(GameManager* gameManager, std::string fileName,
int mode)
	{
	this->gameManager = gameManager;
	saveFile = new SaveFile(fileName, mode);
	}

	int Caretaker::Save()
	{
	try
	{
	Memento* snapShot = gameManager->SaveGame();
	std::ostringstream tmp;
	tmp << snapShot->GetState();
	saveFile->Write(tmp.str());
	if (snapShot)
	delete snapShot;
	return 0;
	}
	catch (const ExceptionFile& e)
	{
	std::cerr << e.what();
```

```
return 1;
}
}

int Caretaker::Load()
{
try
{
std::stringstream save = saveFile->Read();
GameState state;
save >> state;
Memento* snapShot = new Memento(state);
gameManager->LoadGame(snapShot);
if (snapShot)
delete snapShot;
return 0;
}
catch (const ExceptionFile& e)
{
std::cerr << e.what();
return 1;
}
catch (const ExceptionLoad& e)
{
std::cerr << e.what();
return 1;
}
}

Caretaker::~Caretaker()
{
if (saveFile)
delete saveFile;
}
```

Название файла: gameManager.h

```
#pragma once

#include <stdlib.h>
#include <time.h>
#include <utility>
#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"
#include "../GameSave/caretaker.h"
#include "../GameSave/memento.h"
#include "../GameSave/Exceptions/exceptionLoad.h"

class Caretaker;
```

```
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 openLoad();
void openSave();
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();
Memento* SaveGame();
void LoadGame(Memento* snapShot);
~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 == "l" || userCommand == "load")
```

```
      openLoad();
      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::openLoad()
      {
      std::cout << "LOAD: Enter the file name or go BACK\n";
      std::string userCommand = userInterface->ScanCommand();
      if (userCommand == "b" || userCommand == "back")
      OpenMenu();
      else if (userCommand != "q" && userCommand != "quit")
      {
      userCommand = "./" + userCommand;
      Caretaker* caretaker = new Caretaker(this, userCommand, 0);
      if (!caretaker->Load())
      {
      if (caretaker)
      delete caretaker;
      std::cout << "Game was loaded from file " << userCommand << "\n";
      player->GetLogPlayer()->GameStarts(pointsToWin);
      parseMove();
      }
      else
      {
      if (caretaker)
      delete caretaker;
      std::cout << "An error occured. Load was cancelled\n";
      OpenMenu();
      }
      }
      }

      void GameManager::openSave()
      {
      std::cout << "SAVE: Enter the file name or go BACK\n";
      std::string userCommand = userInterface->ScanCommand();
      if (userCommand == "b" || userCommand == "back")
      openPause();
      else if (userCommand != "q" && userCommand != "quit")
      {
      userCommand = "./" + userCommand;
      Caretaker* caretaker = new Caretaker(this, userCommand, 1);
      if (!caretaker->Save())
      {
      if (caretaker)
      delete caretaker;
```

```
std::cout << "Game was saved in file " << userCommand << "\n";
openPause();
}
else
{
if (caretaker)
delete caretaker;
std::cout << "An error occured. Save was cancelled\n";
openPause();
}
}
}

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 == "s" || userCommand == "save")
openSave();
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();
      }


      Memento* GameManager::SaveGame() {
      GameState state;
      state.numOfItemPoint = numOfItemPoint;
      state.numOfItemHealth = numOfItemHealth;
      state.numOfItemEnergy = numOfItemEnergy;
      state.pointsToWin = pointsToWin;
      state.playerPoints = player->GetPoints();
      state.playerHealth = player->GetHealth();
      state.playerEnergy = player->GetEnergy();
      state.playerPos = std::make_pair(player->GetX(), player->GetY());
      state.chaserPos = std::make_pair(chaser->GetX(), chaser->GetY());
      state.ambusherPos = std::make_pair(ambusher->GetX(),
ambusher->GetY());
      state.jumperPos = std::make_pair(jumper->GetX(), jumper->GetY());
      state.itemPointPos = new std::pair<int, int>[numOfItemPoint];
      state.itemHealthPos = new std::pair<int, int>[numOfItemHealth];
      state.itemEnergyPos = new std::pair<int, int>[numOfItemEnergy];
      int pointCount = 0;
      int healthCount = 0;
      int energyCount = 0;
      for(int i = 0; i < field->GetHeight(); i++)
```

```
      for (int j = 0; j < field->GetWidth(); j++)
      if (field->IsItemSet(j, i))
      switch (field->GetItem(j, i)->GetIndex())
      {
      case 0:
      state.itemHealthPos[healthCount] = std::make_pair(j, i);
      healthCount++;
      break;
      case 1:
      state.itemEnergyPos[energyCount] = std::make_pair(j, i);
      energyCount++;
      break;
      case 2:
      state.itemPointPos[pointCount] = std::make_pair(j, i);
      pointCount++;
      break;
      default:
      break;
      }
      return new Memento(state);
      }

      void GameManager::LoadGame(Memento* snapShot) {
      GameState state = snapShot->GetState();
      if (field)
      field->DeleteField();
      field = Field::GetInstance();

      if (field->IsWall(state.playerPos.first, state.playerPos.second))
      throw ExceptionLoad("Load file is corrupted.\n");
      if (player)
      delete player;
      player = new Player(state.playerPos.first, state.playerPos.second);
      field->SetPlayer(player);
      field->GetField()[state.playerPos.second][state.playerPos.first].Pl
acePlayer(player);

      if(state.playerPoints < 0 || state.playerPoints > 7)
      throw ExceptionLoad("Load file is corrupted.\n");
      player->SetPoints(state.playerPoints);

      if (state.playerHealth < 0 || state.playerHealth > 18)
      throw ExceptionLoad("Load file is corrupted.\n");
      player->SetHealth(state.playerHealth);

      if (state.playerEnergy < 0 || state.playerEnergy > 18)
      throw ExceptionLoad("Load file is corrupted.\n");
      player->SetEnergy(state.playerEnergy);

      if(state.numOfItemPoint < 0 || state.numOfItemPoint > 7)
      throw ExceptionLoad("Load file is corrupted.\n");
      numOfItemPoint = state.numOfItemPoint;

      if(state.numOfItemPoint + state.playerPoints != state.pointsToWin ||
state.pointsToWin > 7)
      throw ExceptionLoad("Load file is corrupted.\n");
      pointsToWin = state.pointsToWin;
```

```
      if(state.numOfItemHealth   >   12   ||   state.numOfItemHealth   +
state.playerHealth > 18)
      throw ExceptionLoad("Load file is corrupted.\n");
      numOfItemHealth = state.numOfItemHealth;

      if   (state.numOfItemEnergy   >   15   ||   state.numOfItemEnergy   +
state.playerEnergy > 18)
      throw ExceptionLoad("Load file is corrupted.\n");
      numOfItemEnergy = state.numOfItemEnergy;

      player->SetLogPlayer(new LogPlayer(player));

      ItemFactory* itemFactory;
      itemFactory = new ItemPointFactory;
      for (int i = 0; i < numOfItemPoint; i++)
      {
      if                       (field->IsWall(state.itemPointPos[i].first,
state.itemPointPos[i].second) ||
      field->IsItemSet(state.itemPointPos[i].first,
state.itemPointPos[i].second) ||
      field->IsPlayerSet(state.itemPointPos[i].first,
state.itemPointPos[i].second))
      {
      delete itemFactory;
      throw ExceptionLoad("Load file is corrupted.\n");
      }
      field->GetField()[state.itemPointPos[i].second][state.itemPointPos[
i].first].SetItem(itemFactory->CreateItem());
      }
      delete itemFactory;

      itemFactory = new ItemHealthFactory;
      for (int i = 0; i < numOfItemHealth; i++)
      {
      if                      (field->IsWall(state.itemHealthPos[i].first,
state.itemHealthPos[i].second) ||
      field->IsItemSet(state.itemHealthPos[i].first,
state.itemHealthPos[i].second) ||
      field->IsPlayerSet(state.itemHealthPos[i].first,
state.itemHealthPos[i].second))
      {
      delete itemFactory;
      throw ExceptionLoad("Load file is corrupted.\n");
      }
      field->GetField()[state.itemHealthPos[i].second][state.itemHealthPo
s[i].first].SetItem(itemFactory->CreateItem());
      }
      delete itemFactory;

      itemFactory = new ItemEnergyFactory;
      for (int i = 0; i < numOfItemEnergy; i++)
      {
      if                      (field->IsWall(state.itemEnergyPos[i].first,
state.itemEnergyPos[i].second) ||
      field->IsItemSet(state.itemEnergyPos[i].first,
state.itemEnergyPos[i].second) ||
      field->IsPlayerSet(state.itemEnergyPos[i].first,
state.itemEnergyPos[i].second))
```

```
      {
      delete itemFactory;
      throw ExceptionLoad("Load file is corrupted.\n");
      }
      field->GetField()[state.itemEnergyPos[i].second][state.itemEnergyPo
s[i].first].SetItem(itemFactory->CreateItem());
      }
      delete itemFactory;

      if (field->IsWall(state.chaserPos.first, state.chaserPos.second) ||
      field->IsEnemySet(state.chaserPos.first, state.chaserPos.second) ||
      field->IsPlayerSet(state.chaserPos.first, state.chaserPos.second))
      throw ExceptionLoad("Load file is corrupted.\n");
      if (chaser)
      delete chaser;
      chaser     =     new     Enemy<PolicyChaser>(state.chaserPos.first,
state.chaserPos.second);
      field->GetField()[state.chaserPos.second][state.chaserPos.first].Pl
aceChaser(chaser);

      if (field->IsWall(state.ambusherPos.first, state.ambusherPos.second)
||
      field->IsEnemySet(state.ambusherPos.first,
state.ambusherPos.second) ||
      field->IsPlayerSet(state.ambusherPos.first,
state.ambusherPos.second))
      throw ExceptionLoad("Load file is corrupted.\n");
      if (ambusher)
      delete ambusher;
      ambusher   =   new   Enemy<PolicyAmbusher>(state.ambusherPos.first,
state.ambusherPos.second);
      field->GetField()[state.ambusherPos.second][state.ambusherPos.first
].PlaceAmbusher(ambusher);

      if (field->IsWall(state.jumperPos.first, state.jumperPos.second) ||
      field->IsEnemySet(state.jumperPos.first, state.jumperPos.second) ||
      field->IsPlayerSet(state.jumperPos.first, state.jumperPos.second))
      throw ExceptionLoad("Load file is corrupted.\n");
      if (jumper)
      delete jumper;
      jumper     =     new     Enemy<PolicyJumper>(state.jumperPos.first,
state.jumperPos.second);
      field->GetField()[state.jumperPos.second][state.jumperPos.first].Pl
aceJumper(jumper);
      }

      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;
}
```