Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Программирование

Отчет по выполнению проекта
Генерация кроссворда

Работу выполнил:
Курякин Д. А.
Группа: 13501/4
Преподаватель:
Вылегжанина К.Д.

Санкт-Петербург
2016

Содержание

1 Генерация кроссворда

1.1 Игровые принадлежности

Кроссворд — игра, состоящая в разгадывании слов по определениям. К каждому слову даётся текстовое определение, в описательной или вопросительной форме указывающее некое слово, являющееся ответом. Ответ вписывается в сетку кроссворда и, благодаря пересечениям с другими словами, облегчает нахождение ответов на другие определения. Загаданные слова представлены в кроссворде в виде цепочки ячеек, в каждую из которых по порядку вписываются буквы ответа — по одной в каждую ячейку. В классическом кроссворде ячейки имеют вид квадратных клеток, собранных в прямую линию. Слова «пересекаются» друг с другом, образуя сетку кроссворда. Сетка должна быть связной, без изолированных участков, «оторванных» от остальной сетки. Классическая сетка кроссворда состоит из слов, написанных по вертикали (сверху вниз) и горизонтали (слева направо). Для привязки ответов к определениям в кроссворде последовательно нумеруются ячейки, содержащие первые буквы ответов. Нумерация идет по правилам чтения: слева направо и сверху вниз. Слова, идущие из одной клетки в разных направлениях, нумеруются одной цифрой. В списке определений уточняется направление каждого слова (чаще всего определения сгруппированы по направлениям).

Вместо текстовых определений могут выступать любые задачи, позволяющие дать ответ в одно слово в случае данного приложения текстовые определения - это сами слова.

1.2 Порядок игры

Игрок добавляет слова в словарь, затем нажимает на кнопку генерация. После того как кроссворд сгенерировался, игрок разгадывает кроссворд. После того как все слова разгаданы игрок получает сообщение, то что кроссворд разгадан.

2 Проектирование приложения

2.1 Концепция приложения

В ходе проектирования было разработана концепция продукта. Созданное приложение должно предполагать возможность добавление и удаление слов из базы данных - словаря, генерация сетки кроссворда и разгадывание.

2.2 Минимально работоспособный продукт

Минимальноым роботоспособным продуктом было признано консольное приложение, позволяющие производить генерацию кроссворда.

2.3 Прецеденты использования

На основе разработанной концепции была составлена UML диаграмма прцедентов использовани (рис.1).

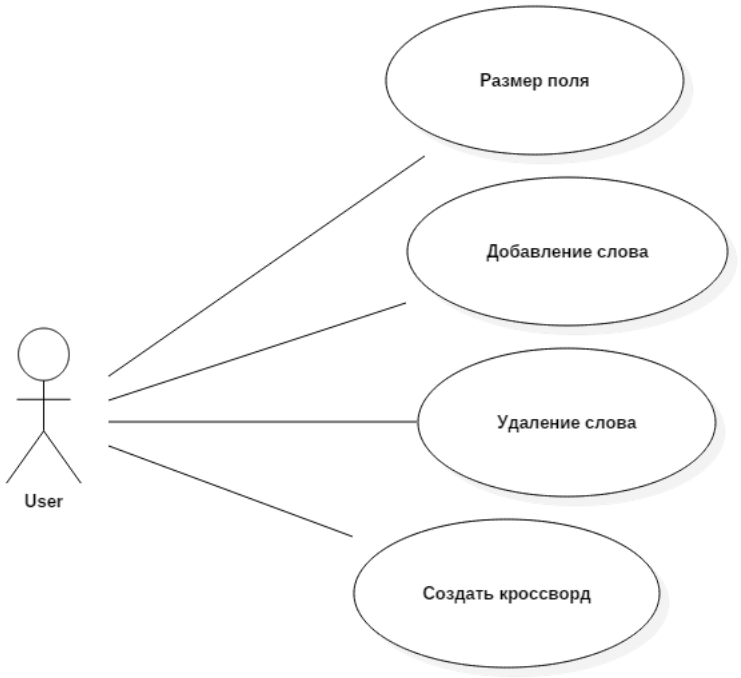

Рис. 1: Диаграмма прцедентов использования

2.4 Основные компоненты приложения

На основе анализа концепции и выделенных прецедентов использования было принято решение выделить три основных компонента, которые будут входить в состав продукта:

1. Библиотека

   Включает в себя игровую модель, а также должна обеспечивать соблюдение правил при ее изменении, сообщать об игровой ситуации и опредлять моменты окончания игры. Кроме того, в библиотеке должны быть реализованы простой исскуственный интелект для игры в Сёги и механизм сохранения и загрузки партий. На основе этого было выделено два интерфейса: первый обеспечивает доступ к игровой модели, взаимодействе с ней, а второй позволяет использовать искуственный интелект для игры.

2. Консольное приложение

   Должно визуализировать с помощью текста игровую модель и позволять пользоватлю взаимодействовать с ней, а также предоставлять возможность использовать остальные функциональности, поддреживаемые библиотекой.

3. Графическое приложение

   Гафически визуализирует игровую модель, предоставляет пользователю графический интерфейс для взаимодействеия с ней и выполнения остальный действий предусмотренных в реализации библиотеки.

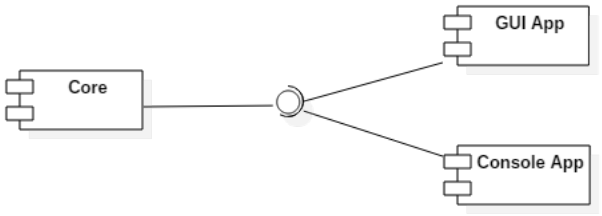

Рис. 2: Диаграмма компонентов

Так, на рис. 2 изображена UML диаграмма компонентов, описывающих взаимодействие компонентов и интерфейсов.

2.5 Используемые инструменты

Было принято решение, для реализации приложения соответсвующего концепции ипользовать сторонии инструменты.

2.5.1 Qt

Qt - кроссплатформенный инструментарий разработки ПО, помимо всего, включающий средства для теститрования (Qt Test), разработки графического интерфеса (Qt Widget) и локализации (Qt Translation), которые необходимы для реализации проекта. Также было принято решение использовать файлы ресурсов, позволяющие хранить данные, необходимые для работы приложения прямо в исполняемом файле. Именно наличие этих инструментов стало основополагающим фактором при выборе Qt. По очевидным причинам использовалась последняя версия Qt 5.6.

3 Реализация приложения

3.1 Среда разработки

- Операционная система: Windows 10
- Компилятор: MinGW 4.9.2 32bit

3.2 Реализация основных компонентов приложения

3.2.1 Библиотека Core

Для реализации всех запланированных функциональстей было принято решение, создать три класс:

1. **Vocabulary**

   Данный класс предстовляет возможность посмотреть, добавить и удалить слова из словаря.

   Метод **AddWord** получает на вход слово и добовляет это слово в словарь.

   Метод **Remove** получает на вход слово и удаляет это слово в из словаря.

   Метод **GenerationFiel** выводит в консоль все слова из словаря.

2. **Field**

   Данный класс предстовляет возможность генерации сетки кроссворда и вписывании самих слов в сетку. Конструктор класса **Field** создаёт 2 поля field и playingField размером sizeField.

   Метод **FirstWordVerification** получает на вход слово и словоря и провряет на наличие слова на поле, если поле пустое то слово добавляется в центор поля, если поле не пустое то, срабатывает метод **PrintNextWords**.

   Метод **PrintNextWords** получает на вход слово и ищит пересечеия с другими словаки которые уже есто на поле.

Метод **InspectionOccupiedLetter** получает на вход координаты пересочния слова, которое ищется методом **PrintNextWords**, и проверяет есто по координатам этого пересечения другие пересечения слов, что обеспечивает то, что по координатам этого пересечения слова не будут добавляться.

Метод **AddOccupiedLetter** получает на вход координаты пересочния слова и добавляет их в базу данных **OccupiedLetter** которая нужна для хранения пересечения на поле.

Метод **AddInformation** получает на вход координаты пересочния слова, размер, направление, и слово. Предназначен для добавления в базу данных **Information** которая нужна для хранения всех перечисленных даных.

Методы **AddCellsPlayingField** и **EnterTheWord** опираясь на базы данных **OccupiedLetter** и **Information** заплняет **playingField**. Метод**PrintPlayingFieldM** выводит игровое поле в консоль пользуясь методами: **MinYInfoField**, **MaxYInfoField**, **MinXInfoField** и **MaxXInfoField**. Эти методы находят максмальные и минимальные координаты букв, что бы выводилось в консоль, та часть поля которая заполнена словами.

3.2.2 Консольное приложение

Консольное приложение было условно поделено на сцены: добавление слова, удаление слова, показать слова и генерация. Был создан классы **ConsoleApp**, **ConsolePlaingField** в которых была, реализвана логика переключения между сценами.

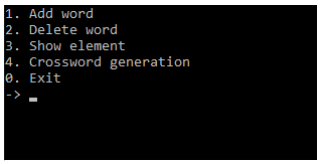

Рис. 3: Основное меню консольного приложения

Рис. 4: Игровой процесс в консольном приложении

3.3 Теститрование

В ходе разработки проекта регулярно проводилось ручное тестирование.
Тестирование позволило обеспечить работоспособность продукта в ходе всего процесса разработки.

3.4 Просмотр кода

Применялась практика так называемого "code review суть которого заключается в том, что достаточно квалифицированные люди просматривали код, в создании которого они не участвовали, и высказывали свои замечения и предложения. Так в ходе разработки было проведено всего лишь один просмотор кода, нацеленный на выявление ошибок и недоработок, связанных непосредственно с кодом и его стилем. В ходе данных проверок было получено беле 50 замечений, большая часть замичаний была исправлена, что позволило значительно повысить качество и надежность кода. Были исправлены ошибки в виде ужасных

char масивов которые сильно усложняли код, в виде плохо написаного конструктора и деструктора иза которого происходии утечки памяти, в виде плохо названних методов которые усложняли написание кода. Но автор не согласился на то чтобы убрать из кода вручную составленные линейные списки, так как когда автор работал с с класом вектор генерация кроссворда происходила непредсказуема.

3.5 Демонстрации

Во время создания приложения было проведено 1 демонстрации, на которых группой людей, представляющих собой потенциальных пользоваталей разработываемого приложения, были сделаны различные замечания и высказаны множество предложений и пожеланий, основанных на внешнем виде продукта и стандартном цикле работы с ним. Анализ полученной информации позволял обнаруживать недочеты присутсвующие в продукте на том, или ином этапе разработки, а также определять дальнейшие направления улучшения и расширения проекта, что, безусловно, положительно сказалось на конечном результате.

4 Выводы

В ходе выполнения данного проекта было получено множество новых знаний. Во-первых удалось улучшить владение языком C++, больше узнать о его строении и принципах работы, глубже познакомиться с STL, исопльзовать многие сильные стороны языка в ходе разработки. Во-вторых был получен опыт, связанный с процессом разработки программного продукта.

Созданный в ходе работы продукт оказался вполне работоспособным. Планируется продолжить работу по совершенствованию библиотеки и создать графическое приложение.

5 Приложение 1

5.1 Листинги

Листинг 1: consoleapp.cpp

```
#include "consoleapp.h"
#include "vocabulary.h"
#include "field.h"
#include "playingfield.h"
#include "consoleplaingfield.h"

void ConsoleApp::Menu()
{
    /*Field *qwe = new Field;

    qwe->FirstWordVerification("qwerbt");
    qwe->FirstWordVerification("wdfmnb");
    qwe->FirstWordVerification("mgfrgysr");
    qwe->FirstWordVerification("utqwf");
    qwe->FirstWordVerification("mhgnjh");
    qwe->FirstWordVerification("yjtlkfjbh");
    qwe->FirstWordVerification("hfjhnnhvc");
    qwe->FirstWordVerification("gcjgnbh");
    qwe->PrintFieldM();
    std::cout<<std::endl;
    qwe->PrintInformation();
    std::cout<<std::endl;
    qwe->AddCellsPlayingField();
    qwe->PrintPlayingFieldM();*/



    Vocabulary *vocabulary = new Vocabulary;
    ConsolePlaingField *consolePlaingField = new ConsolePlaingField;


    std::string word, str;
    int number = 0;

    while(1){
        std::cout << "1. Add word" << std::endl
                  << "2. Delete word" << std::endl
                  << "3. Show element" <<std::endl
                  << "4. Crossword generation" <<std::endl
```

```
                    << "0. Exit" << std::endl
                    << "-> ";

        std::cin>>str;
        try
        {
            number=std::stoi(str);
        }
        catch(std::exception &error)
        {
            number=0;
        }

        switch (number)
        {
        case 0:
            return;
            std::exit(0);
            break;
        case 1:
            std::cout<<"Print word"<<std::endl;
            std::cin>>word;
            vocabulary->AddWord(word);
            break;
        case 2:
            std::cout<<"Print word"<<std::endl;
            std::cin>>word;
            vocabulary->Remove(word);
            break;
        case 3:
            std::cout<<"--------------------"<<std::endl;
            std::cout<<std::endl;
            vocabulary->Output();
            std::cout<<std::endl;
            std::cout<<"--------------------"<<std::endl;
            break;
        case 4:
            consolePlaingField->Console(vocabulary->GenerationField());
            break;
        default:
            std::cout<<"Invalid number! Try again"<<std::endl;
            std::cin.clear();
            getline(std::cin,str);
            break;
        }

    }
}
```

Листинг 2: consoleapp.h

```
#ifndef CONSOLEAPP_H
#define CONSOLEAPP_H

#include "field.h"

#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <cstring>

class ConsoleApp
{
public:

    void Menu();
};

#endif // CONSOLEAPP_H
```

Листинг 3: consoleplaingfield.h

```
#ifndef CONSOLEPLAINGFIELD_H
#define CONSOLEPLAINGFIELD_H
```

```
3
4  #include "field.h"
5
6  #include <iostream>
7  #include <string>
8
9  class ConsolePlaingField
10 {
11 public:
12
13     void Console(Field *field);
14 };
15
16 #endif // CONSOLEPLAINGFIELD_H
```

Листинг 4: consoleplaingfield.cpp

```
1  #include "consoleplaingfield.h"
2  #include "field.h"
3  #include "consoleapp.h"
4
5  void ConsolePlaingField::Console(Field *field)
6  {
7      int number;
8      std::string word;
9
10     while(1)
11     {
12     field->PrintPlayingFieldM();
13     std::cout<<std::endl;
14
15     field->PrintInformation();
16     std::cout<<std::endl;
17
18     std::cout<<"Print number: ";std::cin>>number;
19     std::cout<<std::endl;
20     std::cout<<"Print word: ";std::cin>>word;
21     std::cout<<std::endl;
22
23     if(field->EnterTheWord(number, word) == true)
24         std::cout<<"Word is correct"<<std::endl;
25     else
26         std::cout<<"Word is not correct"<<std::endl;
27     }
28
29 }
```

Листинг 5: crosswordapi.h

```
1  #ifndef CROSSWORDAPI_H
2  #define CROSSWORDAPI_H
3
4  //todo Почему этот файл в APP? API должно быть в бибилиотеке.
5  //todo Почему нигде API не используется?
6  class CrosswordAPI
7  {
8
9  public:
10
11     //todo лучше использовать JAVADOC
12     /*
13      *@brief размер поля
14      */
15     void FieldSize(int x, int y);
16     добавить// слово
17     //todo использовать string, а не сишные строки
18     //todo метод добавляет слово, а получает один символ?
19     bool AddWord(char add_word);
20     удалить// слово
21     //todo использовать string, а не сишные строки
22     //todo метод удаляет слово, а получает один символ?
23     bool DeleteWord(char delete_word);
24     создать// кроссворд
25     //todo может быть, createCrossword?
26     void Create();
```

```
};

#endif // CROSSWORDAPI_H
```

Листинг 6: outputconsole.cpp

```
#include "field.h"
#include "playingfield.h"
#include "vocabulary.h"

void Field::InfoOccupiedLetter()
{
    OccupiedLetter *current = first_OcLet;

    while(current)
    {
        std::cout<<current->x<<" "<<current->y<<std::endl;
        current = current->next;
    }
}

void Field::PrintField()
{
    for(int i = 0; i < sizeField; i++)
    {
        for(int j = 0; j < sizeField; j++)
            std::cout<<field[i][j]<<" ";
        std::cout<<std::endl;
    }
}

void Field::PrintPlayingField()
{
    for(int i = 0; i < sizeField; i++)
    {
        for(int j = 0; j < sizeField; j++)
            std::cout<<playingField[i][j]<<" ";
        std::cout<<std::endl;
    }
}

void Field::PrintFieldM()
{
    for(int i = MinYInfoField(); i < MaxYInfoField(); i++)
    {
        for(int j = MinXInfoField(); j < MaxXInfoField(); j++)
            std::cout<<field[i][j]<<" ";
        std::cout<<std::endl;
    }
}

void Field::PrintPlayingFieldM()
{
    for(int i = MinYInfoField(); i < MaxYInfoField(); i++)
    {
        for(int j = MinXInfoField(); j < MaxXInfoField(); j++)
            std::cout<<playingField[i][j]<<" ";
        std::cout<<std::endl;
    }
}

void Vocabulary::Output()
{
    Vocabulary *current = first;
    while (current)
    {
        std::cout<<current->word<<std::endl;
        current = current->next;
    }

}

void Field::GetAllItemInfo()
{
    Information *current = first;
    while (current)
```

```
71      {
72          std::cout<<current->x<<" ";
73          std::cout<<current->y<<" ";
74          std::cout<<current->length<<" ";
75          std::cout<<current->word<<"     ";
76          std::cout<<current->numberWord<<" ";
77          std::cout<<current->orientation<<std::endl;
78          current = current->next;
79      }
80  }
81
82  void Field::PrintInformation()
83  {
84      Information *current = first;
85      while (current)
86      {
87          std::cout<<current->numberWord<<". "<<current->word<<std::endl;
88          current = current->next;
89      }
90  }
```

Листинг 7: field.cpp

```
1   #include "field.h"
2   #include "vocabulary.h"
3
4   Field::Field()
5   {
6       sizeInfo = 0;
7       first = last = nullptr;
8       first_OcLet = last_OcLet = nullptr;
9
10      field.resize(sizeField);
11      for(int i = 0; i < sizeField; i++)
12      {
13          field[i].resize(sizeField);
14          for(int j = 0; j < sizeField; j++)
15              field[i][j] = "*";
16      }
17
18      playingField.resize(sizeField);
19      for(int i = 0; i < sizeField; i++)
20      {
21          playingField[i].resize(sizeField);
22          for(int j = 0; j < sizeField; j++)
23              playingField[i][j] = "*";
24      }
25  }
26
27  Field::~Field()
28  {
29      Information *currentInf = nullptr;
30      Information *nextInf = first;
31      while(nextInf)
32      {
33          currentInf = nextInf;
34          nextInf =nextInf->next;
35          delete currentInf;
36      }
37
38      OccupiedLetter *currentOcLet = nullptr;
39      OccupiedLetter *nextOcLet = first_OcLet;
40      while(currentOcLet)
41      {
42          currentOcLet = nextOcLet;
43          nextOcLet =nextOcLet->next;
44          delete currentOcLet;
45      }
46
47      field.clear();
48      playingField.clear();
49  }
50
51  void Field::AddInformation(int x, int y, int length, bool orientation, std::string word)
52  {
53      sizeInfo++;
```

```
54      Information *newItem = new Information(x, y, length, word, sizeInfo, orientation);
55      if (!last)
56      {
57          first = newItem;
58      }
59      else
60      {
61          last->next = newItem;
62      }
63      last = newItem;
64  }
65
66  void Field::AddOccupiedLetter(int x, int y)
67  {
68      OccupiedLetter *newItem = new OccupiedLetter(x, y);
69      if(!last_OcLet)
70      {
71          first_OcLet = newItem;
72      }
73      else
74      {
75          last_OcLet->next = newItem;
76      }
77      last_OcLet = newItem;
78  }
79
80  void Field::FirstWordVerification(std::string word)
81  {
82      for(int i = 0; i < sizeField; i++)
83          for(int j = 0; j < sizeField; j++)
84              if(field[i][j] == "*")
85              {
86                  if((i + 1) == sizeField && (j + 1) == sizeField)
87                  {
88                      int sizeWord = word.length();
89                      for(int i = 0; i < sizeWord; i++)
90                      {
91                          field[sizeField / 2][sizeField / 2 + i - (sizeWord / 2)] = word[i];
92                      }
93                      AddInformation(sizeField / 2 - (sizeWord / 2), sizeField / 2, sizeWord,
    ↪ false , word);
94                  }
95              }
96              else
97              {
98                  this->PrintNextWords(word);
99                  return;
100             }
101 }
102
103 bool Field::InspectionOccupiedLetter(int x, int y)
104 {
105     OccupiedLetter *intersection = first_OcLet;
106
107     while(intersection)
108         if((intersection->x != x) && (intersection->y != y)
109                 && (intersection->x + 1 != x) && (intersection->y + 1 != y)
110                     &&(intersection->x - 1 != x) && (intersection->y - 1 != y))
111         { intersection = intersection->next; }
112         else
113         { return false; }
114     return true;
115 }
116
117 void Field::PrintNextWords(std::string word)
118 {
119     Information *current = first;
120     int sizeWord = word.length();
121
122     while (current)
123     {
124         for(int i = 0; i < current->length; i++)
125             for(int j = 0; j < sizeWord; j++)
126             {
127                 if(current->word[i] == word[j] && current->word != word
128                         && this->InspectionOccupiedLetter(current->x + i, current->y + j) ==
```

```
    ↪ true)
                {
                    if(current->orientation == false)
                    {
                        for(int elem = 0; elem < sizeWord; elem++)
                        {
                            field[current->y - j +elem][current->x + i] = word[elem];
                        }

                        AddInformation(current->x + i, current->y - j, sizeWord, true, word);
                        AddOccupiedLetter(current->x + i, current->y);

                    }
                    else
                    {
                        for(int elem = 0; elem < sizeWord; elem++)
                        {
                            field[current->y + i][current->x - j + elem] = word[elem];
                        }

                        AddInformation(current->x - j, current->y + i, sizeWord, false, word);
                        AddOccupiedLetter(current->x, current->y + i);

                    }
                    return;

                }
            }
        current = current->next;
    }
}

void Field::AddCellsPlayingField()
{
    int number = 1;
    for(Information *current = first; current; current = current->next)
    {
        if(current->orientation == false)
            for(int i = 1; i < current->length; i++)
                playingField[current->y][current->x +i] = "-";
        else
            for(int i = 1; i < current->length; i++)
                playingField[current->y + i][current->x] = "|";
    }

    for(OccupiedLetter *current = first_OcLet; current; current = current->next)
        playingField[current->y][current->x] = "+";

    for(Information *current = first; current; current = current->next)
    {
        playingField[current->y][current->x] = std::to_string(number);
        number++;
    }
}

bool Field::EnterTheWord(int number, std::string word)
{
    for(Information *current = first; current; current = current->next)
        if(number == current->numberWord)
            if(word == current->word)
            {
                if(current->orientation == false)
                {
                    for(int i = 0; i < current->length; i++)
                        playingField[current->y][current->x +i] = current->word[i];
                }
                else
                {
                    for(int i = 0; i < current->length; i++)
                        playingField[current->y + i][current->x] = current->word[i];
                }
                return true;
            }
    return false;
}

```

```
204 int Field::MaxXInfoField()
205 {
206     int maxX = 0;
207     for(Information * current = first; current; current = current->next)
208     {
209         if(current->orientation == false)
210             if((current->x + current->length) > maxX)
211                 maxX = (current->x + current->length);
212     }
213     return maxX;
214 }
215
216 int Field::MaxYInfoField()
217 {
218     int maxY = 0;
219     for(Information *current = first; current; current = current->next)
220     {
221         if(current->orientation != false)
222             if((current->y + current->length) > maxY)
223                 maxY = (current->y + current->length);
224     }
225     return maxY;
226 }
227
228 int Field::MinXInfoField()
229 {
230     int minX = sizeField;
231     for(Information *current = first; current; current = current->next)
232     {
233         if(current->orientation == false)
234             if(current->x < minX)
235                 minX = current->x;
236     }
237     return minX;
238 }
239
240 int Field::MinYInfoField()
241 {
242     int minY = sizeField;
243     for(Information *current = first; current; current = current->next)
244     {
245         if(current->orientation != false)
246             if(current->y < minY)
247                 minY = current->y;
248     }
249     return minY;
250 }
```

Листинг 8: field.h

```
1  #ifndef FIELD_H
2  #define FIELD_H
3
4  #include <iostream>
5  #include <string>
6  #include <string>
7  #include <vector>
8
9  class Field
10 {
11 protected:
12     const int sizeField = 60;
13     std::vector<std::vector<std::string>> field;
14     std::vector<std::vector<std::string>> playingField;
15
16     struct Information
17     {
18         int x,y;
19         int length;
20         std::string word;
21         int numberWord;
22         bool orientation;
23
24         Information *next;
25
26         Information(int x_, int y_, int length_, std::string word_, int numerWord_, bool
```

```
    ↪ orientation_ = false, Information *next_ = nullptr):
27          x(x_), y(y_), length(length_),word(word_), numberWord(numerWord_), orientation(
    ↪ orientation_), next(next_){}
28
29      };
30
31      int sizeInfo;
32      Information *first;
33      Information *last;
34
35      struct OccupiedLetter
36      {
37          int x, y;
38          OccupiedLetter *next;
39          OccupiedLetter(int x_, int y_, OccupiedLetter *next_ = nullptr): x(x_), y(y_), next(
    ↪ next_){}
40      };
41
42      OccupiedLetter *first_OcLet;
43      OccupiedLetter *last_OcLet;
44
45  public:
46      Field();
47      ~Field();
48
49      void AddInformation(int x, int y, int length, bool orientation, std::string word);
50      void PrintInformation();
51      void GetAllItemInfo();
52
53      void AddOccupiedLetter(int x, int y);
54      void InfoOccupiedLetter();
55
56      void FirstWordVerification(std::string word);
57      void PrintNextWords(std::string word);
58      bool InspectionOccupiedLetter(int x, int y);
59      bool WrongIntersection(int x, int y, std::string word, bool orientation);
60      void PrintField();
61      void PrintFieldM();
62
63      void AddCellsPlayingField();
64      void PrintPlayingField();
65      void PrintPlayingFieldM();
66      bool EnterTheWord(int number, std::string word);
67
68      int MaxXInfoField();
69      int MaxYInfoField();
70      int MinXInfoField();
71      int MinYInfoField();
72  };
73
74  #endif // FIELD_H
```

Листинг 9: vocabulary.cpp

```
1   #include "vocabulary.h"
2
3   Vocabulary::~Vocabulary()
4   {
5      Vocabulary *current = NULL;
6      Vocabulary *next = first;
7      while(next)
8      {
9          current = next;
10         next =next->next;
11         delete current;
12     }
13  }
14
15  void Vocabulary::AddWord(std::string word)
16  {
17      Vocabulary *newItem = new Vocabulary(word);
18      if (!last)
19      {
20          first = newItem;
21      }
22      else
```

```
23      {
24          last->next = newItem;
25      }
26      last = newItem;
27      size++;
28  }
29
30  bool Vocabulary::Remove(std::string word)
31  {
32      Vocabulary *prev = 0, *current = first;
33      while (current)
34      {
35          if (!current->word.compare(word))
36          {
37              if(prev)
38              {
39                  prev->next = current->next;
40              }
41              else
42              {
43                  first = first->next;
44              }
45              if(current == last)
46              {
47                  last = prev;
48              }
49              delete current;
50              size--;
51              return true;
52          }
53          else
54          {
55              prev =current;
56              current = current->next;
57          }
58      }
59      return false;
60  }
61
62  Field *Vocabulary::GenerationField()
63  {
64      Vocabulary *vocabulary = first;
65      Field *field = new Field;
66      while (vocabulary)
67          {
68              field->FirstWordVerification(vocabulary->word);
69              vocabulary = vocabulary->next;
70          }
71      field->AddCellsPlayingField();
72      return field;
73  }
```

Листинг 10: vocabulary.h

```
1   #ifndef VOCABULARY_H
2   #define VOCABULARY_H
3
4   #include "field.h"
5
6   #include <iostream>
7   #include <string>
8   #include <cstring>
9
10  class Vocabulary:private Field
11  {
12  protected:
13      std::string word;
14      Vocabulary *next;
15      Vocabulary(std::string word_, Vocabulary *next_ = nullptr):word(word_), next(next_){}
16
17      int size;
18      Vocabulary *first;
19      Vocabulary *last;
20  public:
21      Vocabulary()
22      {
```

```
23          size = 0;
24          first = last = NULL;
25      }
26
27      ~Vocabulary();
28
29      void AddWord(std::string word);
30      bool Remove(std::string word);
31      void Output();
32      Field *GenerationField();
33  };
34
35  #endif // VOCABULARY_H
```