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 использования

Пользователь добавляет слова в словарь, затем нажимает на кнопку генерация.

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 и генерация сетки кроссворда.

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).

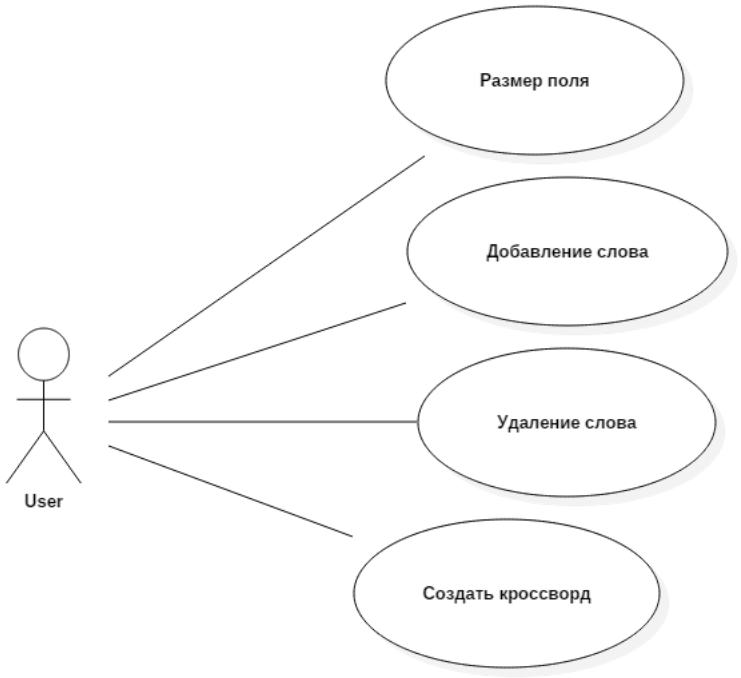

Рис. 1: Диаграмма прцедентов использования

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.

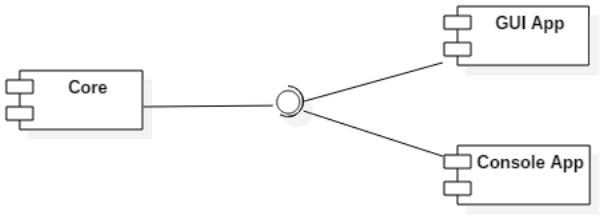

Рис. 2: Диаграмма компонентов

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 два 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 **DelWord** получает на вход слово и удаляет это слово в из словаря.

   Метод **Output** возвращает словарь **vocabulary** для вывода на экран.

2. **Field**

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

   Метод **AddAWordOnTheField** 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, ищит пересечеия с другими словаки которые уже есто на поле.

   Метод **InspectionOccupiedLetter** получает на вход координаты пересечния слова, которое находятся методом **AddAWordOnTheField**, и проверяет есть 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 **Generation** с помощью меода **AddAWordOnTheField** заплняет поле **field**. Метод**OutputField** возвращает поле **field** для вывода на экран.

3.2.2 Консольное приложение

Консольное приложение было условно поделено на сцены: добавление слова, удаление слова, показать слова и генерация. Был создан класс **Console** в котором была, реализвана логика переключения между сценами.

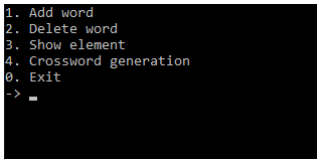

Рис. 3: Основное меню консольного приложения

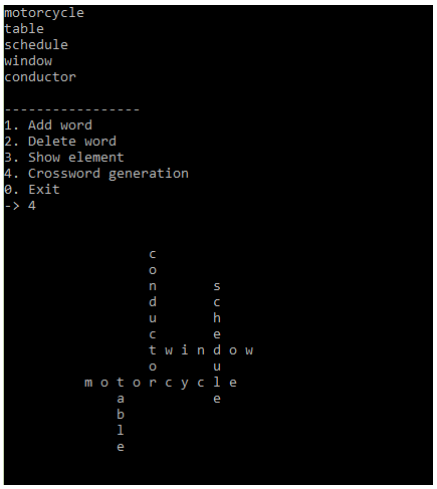

Рис. 4: Вывод сетки кроссворда в консольном приложении

3.2.3 Графическое приложение

Консольное приложение было реализовано с помощью интсрументов Qt. С помощью Qt Widgets было созданы классыд двух основных окон **Interface** и **FieldGUI**, которые отвечали за основное меню и вывод сотки кроссворда.

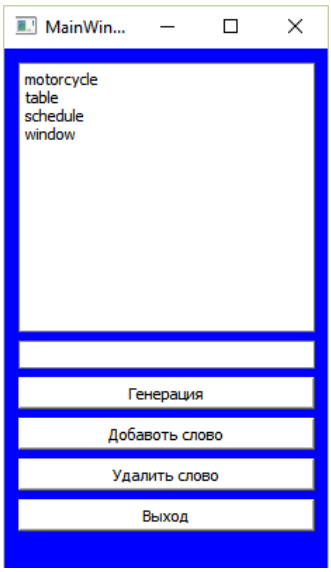

Рис. 5: Основное меню графического приложения

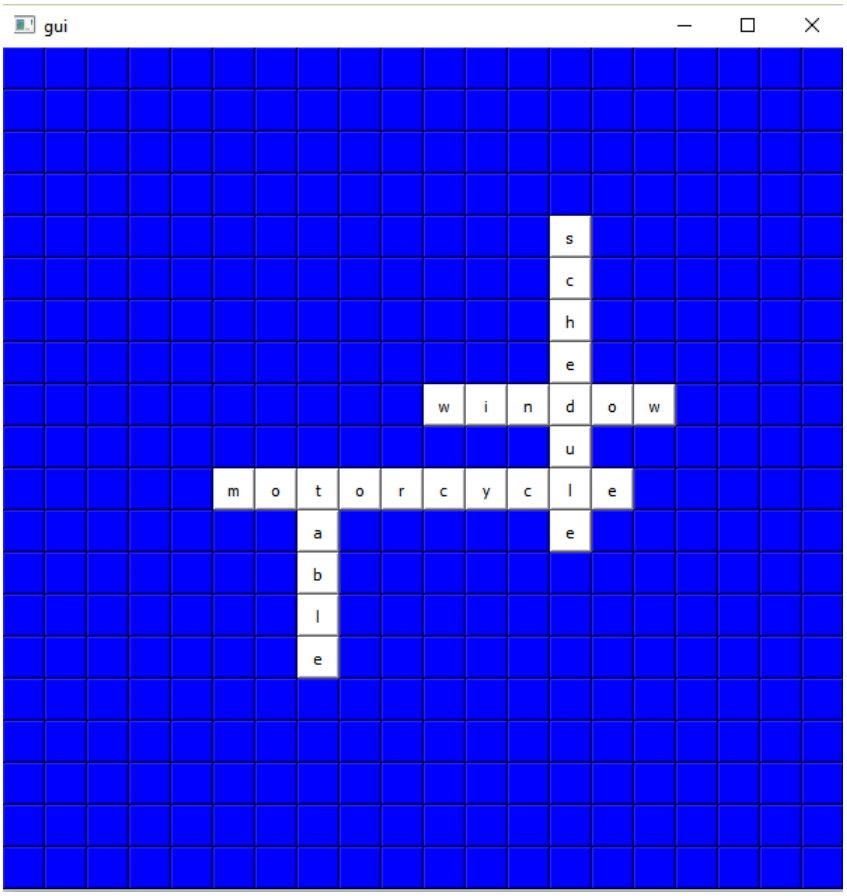

Рис. 6: Вывод сетки кроссворда в графическом приложении

На рисунках 5 и 6 изображены основные окна графического приложения.

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.

5 Приложение 1

5.1 Листинги

Листинг 1: consoleapp.cpp

```
1  #include "console.h"
2
3  Console::Console()
4  {
5
6  }
7
8  void Console::Menu()
9  {
10
11     Vocalbulary *vocabulary = new Vocalbulary();
12     Field *field = new Field();
13
14
15     std::string word, str;
16     int number = 0;
17
18     while(1){
19         std::cout << "1. Add word" << std::endl
20                   << "2. Delete word" << std::endl
21                   << "3. Show element" <<std::endl
22                   << "4. Crossword generation" <<std::endl
23                   << "0. Exit" << std::endl
24                   << "-> ";
25
26         std::cin>>str;
27         try
28         {
29             number=std::stoi(str);
30         }
31         catch(std::exception &error)
32         {
```

```
33              number=100;
34          }
35
36          switch (number)
37          {
38          case 0:
39              return;
40              std::exit(0);
41              break;
42          case 1:
43              std::cout<<"Print word"<<std::endl;
44              std::cin>>word;
45              vocabulary->AddWord(word);
46              break;
47          case 2:
48              std::cout<<"Print word"<<std::endl;
49              std::cin>>word;
50              vocabulary->DelWord(word);
51              break;
52          case 3:
53              std::cout<<"--------------------"<<std::endl;
54              std::cout<<std::endl;
55              PrintWords(vocabulary->Output());
56              std::cout<<std::endl;
57              std::cout<<"--------------------"<<std::endl;
58              break;
59          case 4:
60              PrintField(field->Generation(vocabulary->Output()));
61              break;
62          default:
63              std::cout<<"Invalid number! Try again"<<std::endl;
64              std::cin.clear();
65              getline(std::cin,str);
66              break;
67          }
68
69      }
70  }
71
72  void Console::PrintWords(std::vector<std::string> str)
73  {
74      for(std::vector<std::string>::iterator it = str.begin(); it<str.end(); ++it)
75          std::cout<<*it<<std::endl;
76  }
77
78  void Console::PrintField(std::vector<std::vector<std::string>> field)
79  {
80      int sizeField = field.size();
81      for(int i = 0; i < sizeField; i++)
82      {
83          for(int j = 0; j < sizeField; j++)
84              std::cout<<field[i][j]<<" ";
85          std::cout<<std::endl;
86      }
87  }
```

Листинг 2: consoleapp.h

```
1  #ifndef CONSOLE_H
2  #define CONSOLE_H
3
4  #include "vocalbulary.h"
5  #include "field.h"
6
7  #include <iostream>
8  #include <string>
9  #include <vector>
10 #include <fstream>
11 #include <cstring>
12
13 class Console
14 {
15 public:
16     Console();
17
18     void Menu();
```

```
19      void PrintWords(std::vector<std::string> str);
20      void PrintField(std::vector<std::vector<std::string>> field);
21  };
22
23  #endif // CONSOLE_H
```

Листинг 3: field.cpp

```
1   #include "field.h"
2
3   Field::Field()
4   {
5       field.resize(sizeField);
6       for(int i = 0; i < sizeField; i++)
7       {
8           field[i].resize(sizeField);
9           for(int j = 0; j < sizeField; j++)
10              field[i][j] = "␣";
11      }
12  }
13
14  Field::~Field()
15  {
16      field.clear();
17      InformationAboutWordVector.clear();
18      OccupiedLetterVector.clear();
19  }
20
21  void Field::AddInformationAboutWord(int x, int y, int length, bool orientation, std::string
        ↪ word)
22  {
23      InformationAboutWordVector.push_back(InformationAboutWord(x, y,length, word, orientation))
        ↪ ;
24  }
25
26  void Field::AddOccupiedLetter(int x, int y)
27  {
28      OccupiedLetterVector.push_back(OccupiedLetter(x, y));
29  }
30
31  void Field::AddAWordOnTheField(std::string word)
32  {
33      int sizeWord = word.length();
34      if(InformationAboutWordVector.size() == 0)
35      {
36          for(int i = 0; i < sizeWord; i++)
37          {
38              field[sizeField / 2][sizeField / 2 + i - (sizeWord / 2)] = word[i];
39          }
40          AddInformationAboutWord(sizeField / 2 - (sizeWord / 2), sizeField / 2, sizeWord, false
        ↪ , word);
41      }
42      else
43      {
44          for(std::vector<InformationAboutWord>::iterator infAboutWord =
        ↪ InformationAboutWordVector.begin(); infAboutWord<InformationAboutWordVector.end(); ++
        ↪ infAboutWord)
45          {
46              for(int i = 0; i < infAboutWord->length; i++)
47                  for(int j = 0; j < sizeWord; j++)
48                  {
49                      if(infAboutWord->word[i] == word[j] && infAboutWord->word != word
50                              && this->InspectionOccupiedLetter(infAboutWord->x + i,
        ↪ infAboutWord->y + j) == true)
51                      {
52                          if(infAboutWord->orientation == false)
53                          {
54                              for(int elem = 0; elem < sizeWord; elem++)
55                              {
56                                  field[infAboutWord->y - j +elem][infAboutWord->x + i] = word[
        ↪ elem];
57                              }
58
59                              AddInformationAboutWord(infAboutWord->x + i, infAboutWord->y - j,
        ↪ sizeWord, true, word);
60                              AddOccupiedLetter(infAboutWord->x + i, infAboutWord->y);
```

```
61
62                          }
63                          else
64                          {
65                              for(int elem = 0; elem < sizeWord; elem++)
66                              {
67                                  field[infAboutWord->y + i][infAboutWord->x - j + elem] = word[
    ↪ elem];
68                              }
69
70                              AddInformationAboutWord(infAboutWord->x - j, infAboutWord->y + i,
    ↪ sizeWord, false, word);
71                              AddOccupiedLetter(infAboutWord->x, infAboutWord->y + i);
72
73                          }
74                          return;
75                      }
76                  }
77          }
78          return;
79      }
80  }
81
82  bool Field::InspectionOccupiedLetter(int x, int y)
83  {
84      std::vector<OccupiedLetter>::iterator intersection = OccupiedLetterVector.begin();
85
86      while(intersection < OccupiedLetterVector.end())
87          if((intersection->x != x) && (intersection->y != y)
88                  && (intersection->x + 1 != x) && (intersection->y + 1 != y)
89                      &&(intersection->x - 1 != x) && (intersection->y - 1 != y))
90          { intersection++; }
91          else
92          { return false; }
93      return true;
94  }
95
96  std::vector<std::vector<std::string>> Field::Generation(std::vector<std::string> words)
97  {
98      for(std::vector<std::string>::iterator it = words.begin(); it<words.end(); ++it)
99          this->AddAWordOnTheField(*it);
100     return field;
101 }
102
103 std::vector<std::vector<std::string>> Field::OutputField()
104 {
105     return field;
106 }
107
108 /*int Field::MaxXInfoField()
109 {
110     int maxX = 0;
111     for(std::vector<InformationAboutWord>::iterator infAboutWord = InformationAboutWordVector.
    ↪ begin(); infAboutWord<InformationAboutWordVector.end(); ++infAboutWord)
112     {
113         if(infAboutWord->orientation == false)
114             if((infAboutWord->x + infAboutWord->length) > maxX)
115                 maxX = (infAboutWord->x + infAboutWord->length);
116     }
117     return maxX;
118 }
119
120 int Field::MaxYInfoField()
121 {
122     int maxY = 0;
123     for(std::vector<InformationAboutWord>::iterator infAboutWord = InformationAboutWordVector.
    ↪ begin(); infAboutWord<InformationAboutWordVector.end(); ++infAboutWord)
124     {
125         if(infAboutWord->orientation != false)
126             if((infAboutWord->y + infAboutWord->length) > maxY)
127                 maxY = (infAboutWord->y + infAboutWord->length);
128     }
129     return maxY;
130 }
131
132 int Field::MinXInfoField()
```

```
{
    int minX = sizeField;
    for(std::vector<InformationAboutWord>::iterator infAboutWord = InformationAboutWordVector.
    ↪ begin(); infAboutWord<InformationAboutWordVector.end(); ++infAboutWord)
    {
        if(infAboutWord->orientation == false)
            if(infAboutWord->x < minX)
                minX = infAboutWord->x;
    }
    return minX;
}

int Field::MinYInfoField()
{
    int minY = sizeField;
    for(std::vector<InformationAboutWord>::iterator infAboutWord = InformationAboutWordVector.
    ↪ begin(); infAboutWord<InformationAboutWordVector.end(); ++infAboutWord)
    {
        if(infAboutWord->orientation != false)
            if(infAboutWord->y < minY)
                minY = infAboutWord->y;
    }
    return minY;
}*/
```

Листинг 4: field.h

```
#ifndef FIELD_H
#define FIELD_H

#include <string>
#include <vector>
#include <iostream>

#include "vocalbulary.h"


class Field: public Vocalbulary
{
protected:
    const int sizeField = 20;

    struct InformationAboutWord
    {
        int x,y;
        int length;
        std::string word;
        bool orientation;

        InformationAboutWord(int x_, int y_, int length_, std::string word_, bool orientation_
    ↪  = false):
        x(x_), y(y_), length(length_),word(word_), orientation(orientation_){}
    };

    struct OccupiedLetter
    {
        int x, y;
        OccupiedLetter(int x_, int y_): x(x_), y(y_){}
    };

    std::vector<std::vector<std::string>> field;
    std::vector<InformationAboutWord> InformationAboutWordVector;
    std::vector<OccupiedLetter> OccupiedLetterVector;
public:
    Field();
    ~Field();

    void AddAWordOnTheField(std::string word);

    void AddInformationAboutWord(int x, int y, int length, bool orientation, std::string word)
    ↪ ;
    void AddOccupiedLetter(int x, int y);
    bool InspectionOccupiedLetter(int x, int y);
    std::vector<std::vector<std::string>> Generation(std::vector<std::string> words);
    std::vector<std::vector<std::string>> OutputField();

```

```
48 //     int MaxXInfoField();
49 //     int MaxYInfoField();
50 //     int MinXInfoField();
51 //     int MinYInfoField();
52
53 };
54
55 #endif // FIELD_H
```

Листинг 5: vocabulary.cpp

```
1  #include "vocalbulary.h"
2
3
4  Vocalbulary::Vocalbulary()
5  {
6  }
7
8  Vocalbulary::~Vocalbulary()
9  {
10     words.clear();
11 }
12
13 void Vocalbulary::AddWord(std::string word)
14 {
15     words.push_back(word);
16 }
17
18 void Vocalbulary::DelWord(std::string word)
19 {
20     for(std::vector<std::string>::iterator it = words.begin(); it<words.end(); ++it)
21     {
22         if(*it==word)
23         {
24             words.erase(it);
25         }
26     }
27 }
28
29 std::vector<std::string> Vocalbulary::Output()
30 {
31     return words;
32 }
```

Листинг 6: vocabulary.h

```
1  #ifndef VOCALBULARY_H
2  #define VOCALBULARY_H
3
4  #include <string>
5  #include <sstream>
6  #include <vector>
7  #include <iostream>
8
9  class Vocalbulary
10 {
11 protected:
12     std::vector<std::string>words;
13 public:
14     Vocalbulary();
15     ~Vocalbulary();
16     void AddWord(std::string word);
17     void DelWord(std::string word);
18     std::vector<std::string> Output();
19 };
20
21 #endif // VOCALBULARY_H
```

Листинг 7: interface.cpp

```
1  #include "interface.h"
2  #include "ui_interface.h"
3
4  Interface::Interface(QWidget *parent) :
```

```
5       QMainWindow(parent),
6       ui(new Ui::Interface)
7   {
8       ui->setupUi(this);
9
10      QRegExp reg("[a-z]{1,20}");
11      ui->lineEdit->setValidator(new QRegExpValidator(reg, this));
12  }
13
14  Interface::~Interface()
15  {
16      delete ui;
17  }
18
19  void Interface::Update()
20  {
21      QString outputWords;
22      for(std::vector<std::string>::iterator i = words.begin(); i < words.end(); i++)
23          outputWords = outputWords + QString::fromStdString(*i) + "\n";
24
25      ui->textEdit->setText(outputWords);
26  }
27
28  void Interface::on_generation_clicked()
29  {
30      Field *field = new Field();
31      FieldGUI *fieldGui = new FieldGUI(field->Generation(Output()));
32      fieldGui->show();
33  }
34
35  void Interface::on_addButton_clicked()
36  {
37      AddWord(ui->lineEdit->text().toStdString());
38      ui->lineEdit->clear();
39      this->Update();
40  }
41
42  void Interface::on_delButton_clicked()
43  {
44      DelWord(ui->lineEdit->text().toStdString());
45      ui->lineEdit->clear();
46      this->Update();
47  }
```

Листинг 8: interface.h

```
1   #ifndef INTERFACE_H
2   #define INTERFACE_H
3
4
5   #include <QMainWindow>
6   #include <QMessageBox>
7   #include <QTextEdit>
8   #include <QLineEdit>
9   #include <QPushButton>
10  #include <QVector>
11  #include <QString>
12  #include <string>
13
14  #include "vocalbulary.h"
15  #include "field.h"
16  #include "fieldgui.h"
17
18  namespace Ui {
19  class Interface;
20  }
21
22  class Interface : public QMainWindow, public Vocalbulary
23  {
24      Q_OBJECT
25
26  public:
27      explicit Interface(QWidget *parent = 0);
28      ~Interface();
29
30  private slots:
```

```
31      void Update();
32
33      void on_generation_clicked();
34
35      void on_addButton_clicked();
36
37      void on_delButton_clicked();
38
39  private:
40      Ui::Interface *ui;
41  };
42
43  #endif // INTERFACE_H
```

Листинг 9: fieldgui.cpp

```
1   #include "fieldgui.h"
2
3   FieldGUI::FieldGUI(std::vector<std::vector<std::string>> field, QWidget *parent) :
4       QMainWindow(parent)
5   {
6       for(int i = 0; i < sizeField; i++)
7       {
8           for(int j = 0; j < sizeField; j++)
9           {
10              if(field[i][j] == "␣")
11              {
12                  QPushButton *b = new QPushButton(QString::fromStdString(field[i][j]),this);
13                  b->setStyleSheet("background-color:␣blue");
14                  b->setGeometry(QRect(QPoint((j*sizeButton), (i*sizeButton)), QSize(sizeButton,
    ↪   sizeButton)));
15              }
16              else
17              {
18                  QPushButton *b = new QPushButton(QString::fromStdString(field[i][j]),this);
19                  b->setStyleSheet("background-color:␣white");
20                  b->setGeometry(QRect(QPoint((j*sizeButton), (i*sizeButton)), QSize(sizeButton,
    ↪   sizeButton)));
21              }
22          }
23      }
24      this->setMinimumSize(sizeButton*sizeField, sizeButton*sizeField);
25  }
26
27  FieldGUI::~FieldGUI()
28  {
29
30  }
```

Листинг 10: fieldgui.h

```
1   #ifndef FIELDGUI_H
2   #define FIELDGUI_H
3
4   #include <QMainWindow>
5   #include <QPushButton>
6   #include <QVector>
7   #include <QString>
8   #include <string>
9   #include <vector>
10
11  #include "field.h"
12
13  namespace Ui {
14  class FieldGUI;
15  }
16
17  class FieldGUI : public QMainWindow, public Field
18  {
19      Q_OBJECT
20
21  public:
22      explicit FieldGUI(std::vector<std::vector<std::string>> field, QWidget *parent = 0);
23      ~FieldGUI();
24  private:
```

```
25      int sizeButton = 30;
26
27
28  };
29
30  #endif // FIELDGUI_H
```