ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра вычислительных систем

КУРСОВАЯ РАБОТА

по дисциплине «Технологии разработки программного обеспечения»

на тему «Hangman. Виселица.»

Выполнил:
ст. гр. ИВ-921
Ярошев Р.А.

Проверил:
доц., к.ф.-м.н. Пудов С.Г.

Новосибирск 2020

Оглавление

Введение и постановка задачи

Цель работы: разработка приложения «Hangman. Виселица».

Выбор данной темы проекта основывается на популярности игры. Помимо этого, целью продукта, является пополнение собственного словарного запаса словами английского языка. Программистам приходится много работать с иностранным текстом, поэтому есть необходимость в изучении базовых слов-профессионализмов.

Задачи:

1) разработка алгоритма игры «Hangman. Виселица»;

2) создание игрового интерфейса при помощи конструктора интерфейса Glade;

Постановка задач:

1) Составить базу слов;

2) Создать игровой интерфейс;

3) Реализация функции случайного выбора слова;

4) Реализация алгоритма хода игры;

5) Подключение интерфейса к игровому алгоритму.

Техническое задание

Тема: Hangman. Виселица.

Состав команды (группа ИВ-921): Ярошев Роман, Воронова Ангелина.

Функционал проекта: "Hangman" (Виселица) – продукт, направленный на развитие мышления в легкой игровой форме, посредством разгадывания слова, которое выбирается случайным образом. В начале игры предлагается выбрать категорию сложности слов. На разгадку дается определенное количество попыток, до тех пор пока слово не будет угадано или виселица не построится полностью. Размер слова известен заранее.

Формат входных данных: на вход со стороны игрока путем клика по виртуальной клавиатуре приложения поступают символы (переменная типа char), буквы латинского алфавита .

Интерфейс приложения: интерфейс разработан в конструкторе Glade. Приложение работает в интерактивном режиме и отображается в графическом окне. При вводе существующего символа, он исчезает, чтобы избежать повторений со стороны игрока.

При запуске пользователю будет предложено меню, которое включает:

-Уровни сложности;

-Начать игру;

-Выйти из игры.

Описание выполненного проекта

Этапы разработки приложения

1) Составили базу слов.

Все слова разделяются на четыре группы (четыре уровня сложности) в зависимости от количества букв в слове:

- 1 группа (4 буквы)
- 2 группа (5 буквы)
- 3 группа (6 букв)
- 4 группа (7 букв)

Каждая группа хранится в отдельном текстовом файле, который содержит по 15 слов на английском языке. Слова являются существительными в единственном числе, относятся к IT - терминам.

2) Создали макет игрового интерфейса в редакторе Glade.

Интерфейс состоит из трех блоков:

1. начало игры (выбор сложности);
2. непосредственно игровой процесс (динамическое изображение, буквы);
3. итог (изображение и надпись о победе/проигрыше);

3) Разработали функцию случайного выбора слова.

Функция WordSelect() выбирает одно слово из группы слов, которые хранятся в текстовых файлах, в соответствии с выбранным уровнем сложности. Если игрок захочет продолжить игру, то временно исключается ранее загаданное слово, что позволит избежать повторений.

4) Разработали алгоритм игрового процесса.

Если буква содержится в загаданном слове, то, при нажатии на клавиатуру, осуществляется ее поиск. Она отобразится в слове и будет убрана из предлагаемого алфавита , для дальнейшего угадывания слова.

Если буква не содержится в загаданном слове, то, при нажатии на клавиатуру, осуществляется ее поиск. При отсутствии буквы в слове, она исчезает из предлагаемого алфавита и количество попыток уменьшается на единицу.

Выигрыш/проигрыш:

- при отгадывании слова, выводится сообщение о выигрыше;

- при достижении предельного количества ошибок, выводится сообщение о проигрыше.

5) Подключили интерфейс к игровому алгоритму.

Созданный интерфейс цепляем к игровым алгоритмам.

Вначале игроку представляется окно, в котором он выбирает сложность ("Just", "Easy", "Medium", "Hard") и запускает игру, путем нажатия кнопки "Start the game".

Также игрок может покинуть игру, нажав кнопку "Quit the game".

Игровой процесс:

перед игроком открывается окно, в нижней части которого находится кликабельная экранная клавиатура. Раскладка клавиатуры соответствует техническому стандарту (не по алфавиту).

Над клавиатурой расположена область загаданного слова.

В верхней части окна будет строиться виселица.

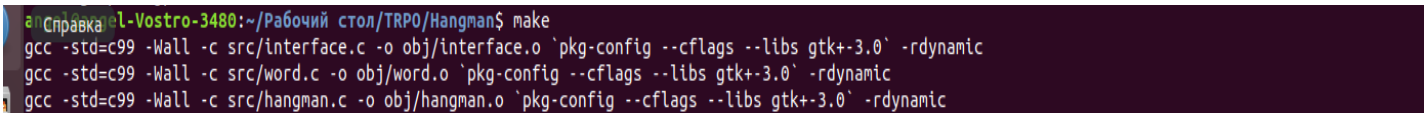
Завершение игры:

будет представлено окно с выводом сообщения о победе или проигрыше.

Также игроку будет предложено начать игру заново ("Replay") или покинуть игру ("Quit").

Компиляция и запуск программы

1) Компиляцию программы запускаем командой «make»:



2) Далее командой «./gam/hangman» запускаем приложение. Открывается стартовое окно интерфейса. Здесь представлено название игры, а так же предлагается выбрать один из четырех уровней сложности. Далее, мы можем начать игру либо покинуть ее.

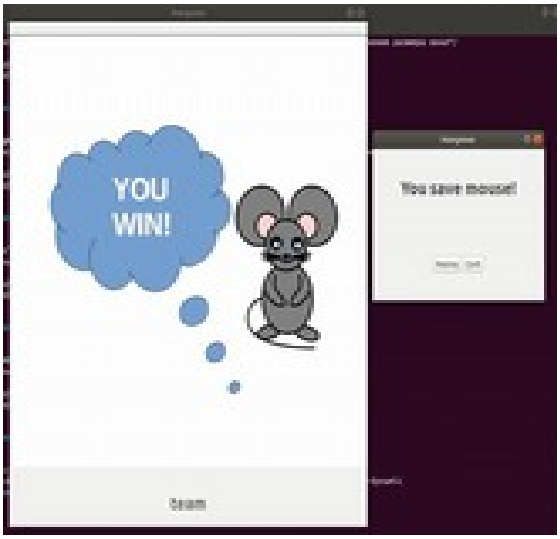
Выбираем уровень сложности (например Just) и начинаем игру (Start the game):

Открывается окно, в котором будет происходить игровой процесс. В нем представленна кликабельная клавиатура, количество прочерков, соответствующие количеству букв в загаданном слове, а так же картинка, на которой изображенна мышонок. Если игрок нажмет на букву, которой нет в загаданном слове, то она исчезнет из клавиатуры и будет строиться виселица. Если буква будет присутствовать в загаданном слове, то она исчезнет из клавиатуры и отобразится на своем месте вместо прочерка.

Итог игры

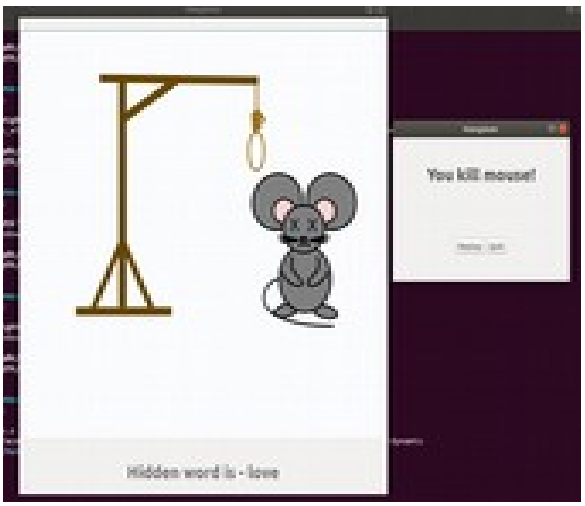
Победа

Если игрок отгадает загаданное слово, то появится окно с сообщением о спасении мышонка и ему будет предложено начать игру заново «Replay» либо покинуть игру «Quit»:



Поражение

Если игрок не отгадает загаданное слово, то появится окно с сообщением о том, что он убил мышонка и ему будет предложено начать игру заново «Replay» либо покинуть игру «Quit». В окне, где происходил игровой процесс под картинкой будет отображаться не угаданное слово.



Личный вклад в проект

Мой вклад в работу - это написание алгоритма интерфейса игры, его подключение к игровому алгоритму, написание тестов для алгоритма интерфейса, а также создание и визуальная начинка интерфейса.

Алгоритм интерфейса реализован библиотекой GTK (виджеты). GDK - вывод информации.

Сразу после компиляции запускается GTK (general_windows) и файл с интерфейсом.

Точкой входа для интерфейса будет функция gtk_init с параметрами &argc и &argv.

General_windows представлен областью difficultGrid, которая в свою очередь содержит контейнеры:

- gameLabel (название игры - «HANGMAN»);
- selDiffLabel («Choose the difficulty of the game»);
- lv_just , lv_easy, lv_medium, lv_hard (уровени сложности «Just», «Easy», «Medium» и «Hard» соответсвенно. Функция GTKRadioButton);
- Start_but, Exit_but (кнопки начала игры «Start the game» и выхода из нее «Quit the game». Функция GTKButton);

Далее, выбрав уровень сложности (параметр функции GTKRadioButton), игрок кликает «Start the game», запускается функция void StartGame(), скрывается general_windows функцией gtk_widget_hide.

Функцией gtk_widget_show с параметром (level_windows) обращаемся к нашему контейнеру. В данном контейнере будет идти основной игровой процесс. Содержание контейнера таково:

- Виджеты контейнера размещены в боксе — GtkBox
- Функцией gtk_image_set_from_file с параметром (Image, "./iwi/image/0.jpg") подключаем директорию, в которой хранятся изображения. Данные изображения, интегрируясь с игровыми событиями, будут сменяться.
- Под изображением мы выделяем область под загаданное слово. Строкой «hiddenWord = (char*)malloc(word_size*sizeof(char))» динамически резервируем память: размер слова в бит умножаем на размер типа данного char - 8 бит на символ.
- Ниже отображаем область клавиатуры GTK функцией gtk_builder_get_object. Параметр keyboard_Gr.
- Получение символа из кнопки на виртуальной клавиатуре осуществляется в функции (или обработчике для Glade) int getSymbol с параматром button.
- После того как текст кнопки был извлечен кнопка скрывается. Функция gtk_widget_hide параметр button.

- Далее, в случае, если сумма жизней: symExists = 1, открывается mess_windows функцией gtk_window_move. Размеры этого окна задаются параметрами вывода информации: gdk_screen_width() / 1.35 (ширина в столбцах), gdk_screen_height() / 2 (высота в строках). Экранная клавиатура скрывается: gtk_widget_hide(GTK_WIDGET(keyboard_Gr).

- Функция gtk_label_set_text устанавливает текст в виджете GtkLabel (в случае выигрыша: «"You save mouse!"»).

- Далее устанавливаем путь к файлу, содержащему изображение о выигрыше (gtk_image_set_from_file(Image, "./iwi/image/win.jpg")), извлекаем win.jpg.

- В противном случае , если символ, введенный игроком не совпадает с символом загаданного слова: else if (!symExists), то счетчик ошибок увеличивается на 1: err_amount++ и подгружается изображение с порядковым номером, соответствующим количеству ошибок. Например, «5.jpg» свидетельствует об исчерпании попыток и проигрыше; хранит в себе изображение с «убитой» мышкой.

  Когда количество ошибок достигнет 5 штук, открываем mess_windows с шириной 1.35 столбцов и высотой 2 строки. Вновь скрываем клавиатуру: gtk_widget_hide(GTK_WIDGET(keyboard_Gr)); вместо строки word_game — строка где отображалось угадываемое слово, записываем: "Hidden word is - %s". Из переменной word_game отображаем текущее не угаданное слово.

- В mess_Lab в виджете GtkLabel вписываем текст: "You kill mouse!" Тут же имеем кнопки Replay и Quit.

- При нажатии «Replay» вызываем функцию void replayGame(), которая закрывает окн с игровым процессом (level_windows) и окно с результатом игры (mess_windows), и открывает стартовое окно с выбором сложности: gtk_widget_show(general_windows).

Приложение

word.c

```
1   #include "game.h"
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include <string.h>
5   #include <time.h>
6
7   /*
8    * Function:  get_filepath_by_difficult
9    * ------------------------------------
10   * получает путь к файлу по сложности
11   *
12   * difficult: сложность игры
13   *
14   * return: путь к файлу либо NULL
15  */
16
17  char* get_filepath_by_difficult(int difficult)
18  {
19      switch (difficult)
20      {
21      case 0:
22          return "./iwi/words/just.txt";
23          break;
24      case 1:
25          return "./iwi/words/easy.txt";
26          break;
27      case 2:
28          return "./iwi/words/medium.txt";
29          break;
```

```
30      case 3:
31          return "./iwi/words/hard.txt";
32          break;
33      default:
34          return NULL;
35      }
36  }
37
38  /*
39   * Function:  read_file
40   * --------------------
41   * считывает файл
42   *
43   * difficult: сложность игры
44   *
45   * return: читает файл
46  */
47  FILE* read_file(int difficult)
48  {
49      char* filepath = get_filepath_by_difficult(difficult);
50      if (filepath == NULL) {
51          return NULL;
52      }
53
54      return fopen(filepath, "r");
55  }
56
57  /*
58   * Function: get_words_amount
59   * --------------------------
```

```
 * подсчитывает количество слов в файле
 *
 * words: указатель на текстовый файл
 *
 * return: количество слов в файле
*/
int get_words_amount(FILE* words)
{
    int ch, str_length = 0, words_amount = 0;
    while ((ch = fgetc(words)) != EOF) {
        if (ch == '\n') {
            if (str_length > 0) {
                words_amount += 1;
            }
            str_length = 0;
        }

        str_length += 1;
    }
    return words_amount;
}
/*
 * Function: get_word_length
 * -------------------------
 * подсчитывает количество букв в слове
 *
 * str: указатель на массив, куда будет записываться загаданное слово
 *
 * return: количество букв в слове
*/
```

```
90  int get_word_length(char* str)
91  {
92      int word_length = 0;
93      word_length = strlen(str) - 1;
94      return word_length;
95  }
96
97  /*
98   * words_amount: количество слов в текстовом файле
99   *
100  * (*rand_fn)(): указатель на функцию rand()
101  *
102  * return: случайно выбранный номер слова
103 */
104 int get_random_word_num(int words_amount, int (*rand_fn)())
105 {
106     return rand_fn() % words_amount;
107 }
108
109 /*
110  * words: указатель на текстовый файл
111  *
112  * (*rand_fn)(): указатель на функцию rand()
113  *
114  * return: новое слово из текстового файла
115 */
116 char* word_select_with_rand(FILE* words, int (*rand_fn)())
117 {
118     int words_amount = 0, word_length = 0;
119     char str[10];
```

```
120     words_amount = get_words_amount(words);
121     int new_word_num = get_random_word_num(words_amount, rand_fn);
122     fseek(words, 0, SEEK_SET);
123     int i = 0;
124     while (i <= new_word_num) {
125         fgets(str, 10, words);
126         i++;
127     }
128 
129     word_length = get_word_length(str);
130     char* new_word;
131     new_word = (char*)malloc(word_length * sizeof(char));
132     for (i = 0; i < word_length; i++) {
133         new_word[i] = str[i];
134     }
135 
136     new_word[i] = '\0';
137 
138     return new_word;
139 }
140 
141 /*words: указатель на текстовый файл
142  *
143  * returns: слово
144  */
145 char* word_select(FILE* words)
146 {
147     srand(time(NULL));
148     return word_select_with_rand(words, rand);
149 }
```

word.h

```
1   #ifndef WORD_H
2   #define WORD_H
3
4   #include <stdio.h>
5
6   char* get_filepath_by_difficult(int difficult);
7   FILE* read_file(int difficult);
8   int get_words_amount(FILE* words);
9   int get_word_length(char* str);
10  int get_random_word_num(int words_amount, int (*rand_fn)());
11  char* word_select(FILE* words);
12  char* word_select_with_rand(FILE* words, int (*rand_fn)());
13
14  #endif
```

game.c

```
1   #include "game.h"
2   #include "hangman.h"
3   #include "interface.h"
4   #include "word.h"
5   #include <gtk/gtk.h>
6   #include <stdlib.h>
7   #include <string.h>
8
9   int difficult = 0;
10
11  /*
```

```
 * Function: difficult_set
 * ----------------------
 * выбор сложности
 *
 * GtkToggleButton *button:  GtkToggleButton запоминает состояние кнопки при нажатии
 *
 * returns: void
*/
void difficult_set(GtkToggleButton* button)
{
    if (gtk_toggle_button_get_active(button))
    {
        const char* difficult_game = gtk_button_get_label(GTK_BUTTON(button));
        if (!strcmp(difficult_game, "Just")) {
            difficult = 0;
        } else if (!strcmp(difficult_game, "Easy")) {
            difficult = 1;
        } else if (!strcmp(difficult_game, "Medium")) {
            difficult = 2;
        } else if (!strcmp(difficult_game, "Hard")) {
            difficult = 3;
        }
    }
}


/*
 * Function: start_game
 * --------------------
 * начало игры
```

```
 *
 * returns: void
*/
void start_game()
{
    int i;
    err_amount = 0;
    char path_word[60];
    gtk_image_set_from_file(Image, "./iwi/image/0.jpg");
    gtk_widget_hide(general_windows);
    gtk_widget_hide(mess_windows);
    gtk_widget_show_all(GTK_WIDGET(keyboard_Gr));

    FILE* words = read_file(difficult);

    if (words == NULL) {
        printf("Words database \"%s\" not found.\n", path_word);
        GtkWidget* err_words_msg = gtk_message_dialog_new(
                NULL,
                GTK_DIALOG_MODAL,
                GTK_MESSAGE_ERROR,
                GTK_BUTTONS_CLOSE,
                "Error");
        gtk_message_dialog_format_secondary_text(
                GTK_MESSAGE_DIALOG(err_words_msg),
                "Words database \"%s\" not found.\n",
                path_word);
        gtk_dialog_run(GTK_DIALOG(err_words_msg));
        gtk_main_quit();
        return;
```

```
72      }
73
74      if (!(word_game = word_select(words))) {
75          return;
76      }
77      printf("Word: %s\n", word_game);
78
79      word_size = strlen(word_game);
80
81      hidden_word = (char*)malloc(word_size * sizeof(char));
82      for (i = 0; i < word_size; i++) {
83          hidden_word[i] = '-';
84      }
85      hidden_word[i] = '\0';
86      gtk_label_set_text(hidden_wor_lab, hidden_word);
87      gtk_widget_show(level_windows);
88  }
89  void replay_game()
90  {
91      gtk_widget_hide(level_windows);
92      gtk_widget_hide(mess_windows);
93      gtk_widget_show(general_windows);
94  }
```

game.h

```
1   #ifndef GAME_H
2   #define GAME_H
3
4   int word_size, err_amount;
```

```
5    char *word_game, *hidden_word;
6
7    #endif
```

hangman.c

```
1    #include "game.h"
2    #include "interface.h"
3    #include <stdio.h>
4    #include <stdlib.h>
5    #include <string.h>
6
7    int get_symbol(GtkButton* button)
8    {
10       char err_path[20], word_mess[20];
11       int width, height;
12       gtk_window_get_size(GTK_WINDOW(mess_windows), &width, &height);
13       const char* symbol = gtk_button_get_label(button);
14       gtk_widget_hide(GTK_WIDGET(button));
15       int sym_exists = 0;
16
17       for (int i = 0; i < word_size; i++) {
18           if (word_game[i] == *symbol) {
19               sym_exists = 1;
20               break;
21           }
22       }
23
24       if (sym_exists) {
```

```
25          for (int i = 0; i < word_size; i++) {
26              if (word_game[i] == *symbol) {
27                  hidden_word[i] = *symbol;
28              }
29          }
30          gtk_label_set_text(hidden_wor_lab, hidden_word);
31          if (!strcmp(word_game, hidden_word)) {
32             gtk_window_move(
33                      GTK_WINDOW(mess_windows),
34                      gdk_screen_width() / 1.35,
35                      gdk_screen_height() / 2
36                              - height / 2);
37              gtk_label_set_text(mess_Lab, "You save mouse!");
38              gtk_image_set_from_file(Image, "./iwi/image/win.jpg");
39              gtk_widget_show(mess_windows);
40          }
41      } else if (!sym_exists) {
42          err_amount++;
43          sprintf(err_path, "./iwi/image/%d.jpg", err_amount);
44          gtk_image_set_from_file(Image, err_path);
45      }
46
47      if (err_amount == 5)
48      {
49          gtk_window_move(
50                  GTK_WINDOW(mess_windows),
51                  gdk_screen_width() / 1.35,
52                  gdk_screen_height() / 2 - height / 2);
53          gtk_widget_hide(GTK_WIDGET(keyboard_Gr));
54          sprintf(word_mess, "Hidden word is - %s", word_game);
```

```
55          gtk_label_set_text(mess_Lab, "You kill mouse!");
56          gtk_label_set_text(hidden_wor_lab, word_mess);
57          gtk_widget_show(mess_windows);
58      }
59      return 0;
60  }
```

hangman.h

```
1   #ifndef HANGMAN_H
2   #define HANGMAN_H
3   #include <gtk/gtk.h>
4
5   int sym_used(char sym, char* str);
6   int get_symbol(GtkButton* button);
7   #endif
```

interface.c

```
1   #include "interface.h"
2   #include <gtk/gtk.h>
3
4   int main(int argc, char* argv[])
5   {
6       GError* error = NULL;
7       gtk_init(&argc, &argv);
8       builder = gtk_builder_new();
9       if (!gtk_builder_add_from_file(builder, UI_FILE, &error)) {
10          printf("Interface file \"%s\" not found.", UI_FILE);
11          GtkWidget* errMsg = gtk_message_dialog_new(
12                  NULL,
```

```
13                  GTK_DIALOG_MODAL,
14                  GTK_MESSAGE_ERROR,
15                  GTK_BUTTONS_CLOSE,
16                  "Error");
17          gtk_message_dialog_format_secondary_text(
18                  GTK_MESSAGE_DIALOG(errMsg),
19                  "Interface file \"%s\" not found.",
20                  UI_FILE);
21          gtk_dialog_run(GTK_DIALOG(errMsg));
22          return 1;
23      }
24      general_windows
25              = GTK_WIDGET(gtk_builder_get_object(builder,
    "general_windows"));
26
        level_windows
27
              = GTK_WIDGET(gtk_builder_get_object(builder,
28  "level_windows"));
29      mess_windows = GTK_WIDGET(gtk_builder_get_object(builder,
    "mess_windows"));
30
        lv_just = GTK_RADIO_BUTTON(gtk_builder_get_object(builder,
31  "lv_just"));
32      lv_easy = GTK_RADIO_BUTTON(gtk_builder_get_object(builder,
    "lv_easy"));
33
        lv_medium = GTK_RADIO_BUTTON(gtk_builder_get_object(builder,
34  "lv_medium"));
35      lv_hard = GTK_RADIO_BUTTON(gtk_builder_get_object(builder,
    "lv_hard"));
36
37      Start_but = GTK_BUTTON(gtk_builder_get_object(builder, "Start_but"));
38      Exit_but = GTK_BUTTON(gtk_builder_get_object(builder, "Exit_but"));
39      hidden_wor_lab
40              = GTK_LABEL(gtk_builder_get_object(builder,
    "hidden_wor_lab"));
41
        mess_Lab = GTK_LABEL(gtk_builder_get_object(builder, "mess_Lab"));
42
        keyboard_Gr = GTK_GRID(gtk_builder_get_object(builder,
```

```
43  "keyboard_Gr"));
44      Image = GTK_IMAGE(gtk_builder_get_object(builder, "Image"));
45
46      gtk_builder_connect_signals(builder, NULL);
47      g_object_unref(G_OBJECT(builder));
48      gtk_widget_show(general_windows);
49      gtk_main();
50
51      return 0;
52  }
```

interface.h

```
1   #ifndef INTERFACE_H
2   #define INTERFACE_H
3   #include <gtk/gtk.h>
4   #define UI_FILE "./iwi/interface.glade"
5
6   GtkBuilder* builder;
7   GtkWidget *general_windows, *level_windows, *mess_windows;
8   GtkRadioButton *lv_just, *lv_easy, *lv_medium,*lv_hard;
9   GtkButton *Start_but, *Exit_but;
10  GtkLabel *hidden_wor_lab, *mess_Lab;
11  GtkGrid* keyboard_Gr;
12  GtkImage* Image;
13
14  #endif
```

hangman_cli.c

```
1   #include "conio.h"
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define alphabet_size 26

int sym_used(char sym, char* str)
{
    for (int i = 0; i < strlen(str); i++) {
        if (str[i] == sym) {
            return 1;
        }
    }
    return 0;
}
char read_symbol(char symbol,char used_symbols[alphabet_size],
int err_amount, char* hidden_word)
{

    int word_size = strlen(hidden_word);

    printf("Used symbols:\n");
    for (int k = 0; k < alphabet_size; k++) {
        printf("%c", used_symbols[k]);
    }
    printf("\n\n");
    printf("Errors: %d\n\n", err_amount);
    for (int i = 0; i < word_size; i++) {
        printf("%c", hidden_word[i]);
    }
```

```
32      printf("\n\nEnter symbol: ");
33      symbol = getche();
34      return symbol;
35  }
36  int check_symbol(char symbol, char* word)
37  {
38
39      int word_size, sym_exists = 0;
40      for (word_size = 0; word[word_size] != '\0'; word_size++) {
41      }
42      for (int i = 0; i < word_size; i++) {
43          if (word[i] == symbol) {
44              sym_exists = 1;
45              break;
46          }
47      }
48      return sym_exists;
49  }
50  void add_symbol_to_hidden_word(char* word, char symbol, char* hidden_word)
51  {
52
53      int word_size = strlen(word);
54      for (int i = 0; i < word_size; i++) {
55          if (word[i] == symbol) {
56              hidden_word[i] = symbol;
57          }
58      }
59  }
60
61  int win_or_lose(int err_amount)
```

```
{
    if (err_amount != 5)
    {
        printf("You win!\n");
        return 1;
    } else {
        printf("You lose!\n");
        return 0;
    }
}

int Hangman(char* word)
{
    int i, count = 0, word_size,
           err_amount = 0;
    char symbol, used_symbols[alphabet_size];

    for (int k = 0; k < alphabet_size; k++) {
        used_symbols[k] = '-';
    }

    for (word_size = 0; word[word_size] != '\0'; word_size++) {
        ;
    }

    char* hidden_word;
    hidden_word = (char*)malloc(word_size * sizeof(char));
    for (i = 0; i < word_size; i++) {
        hidden_word[i] = '-';
    }
```

```
92      printf("Press any key to start...\n");
93      getch();
94      system("clear");
95      while (err_amount != 5)
96      {
97          symbol = read_symbol(symbol, used_symbols, err_amount,
98  hidden_word);
99          if (symbol < 'a' || symbol > 'z') {
100             printf("\nYou should enter [a..z] symbols only.");
101             getch();
102             system("clear");
103             continue;
104         }
105
106         int sym_exists = check_symbol(symbol, word);
107
108         if (sym_exists == 1) {
109             add_symbol_to_hidden_word(word, symbol, hidden_word);
110             if (!strcmp(word, hidden_word)) {
111                 system("clear");
112                 break;
113             }
114         } else if (!sym_exists && !sym_used(symbol, used_symbols)) {
115
116             err_amount++;
117         }
118
119         if (!sym_used(symbol, used_symbols)) {
120             used_symbols[count] = symbol;
121             count++;
```

```
122         }
123
124         system("clear");
125     }
126     return win_or_lose(err_amount);
127 }
```

hangman_cli.h

```
1   #ifndef HAGMAN_H
2   #define HAGMAN_H
3
4   int sym_used(char sym, char* str);
5   char read_symbol(
6           char symbol, char used_symbols, int err_amount, char*
    hidden_word);
7
    int check_symbol(char symbol, char* word);
8
    void add_symbol_to_hidden_word(char* word, char symbol, char*
9
    hidden_word);
10
    int win_or_lose(int err_amount);
11
    int Hangman(char* word);
12
13
    #endif
14
```

conio.c

```
1   #include <stdio.h>
2   #include <termios.h>
```

```
3   #include <unistd.h>
4
5   int getch(void)
6   {
7       struct termios oldattr, newattr;
8       int ch;
9       tcgetattr(STDIN_FILENO, &oldattr);
10      newattr = oldattr;
11      newattr.c_lflag
12              &= ~(ICANON | ECHO);
13      tcsetattr(STDIN_FILENO, TCSANOW, &newattr);
14      ch = getchar();
15      tcsetattr(STDIN_FILENO, TCSANOW, &oldattr);
16      return ch;
17  }
18
19  /*
20   * Function: getche
21   * ----------------
22   * читает символ с клавиатуры и отображает экран
23   *
24   * return: переменная, куда будет записана нажатая клавиша
25  */
26  int getche(void)
27  {
28      struct termios oldattr, newattr;
29      int ch;
30      tcgetattr(STDIN_FILENO, &oldattr);
31      newattr = oldattr;
32      newattr.c_lflag &= ~(ICANON);
```

```
33      tcsetattr(STDIN_FILENO, TCSANOW, &newattr);
34      ch = getchar();
35      tcsetattr(STDIN_FILENO, TCSANOW, &oldattr);
36      return ch;
37  }
```

conio.h

```
1   #ifndef CONIO_H
2   #define CONIO_H
3
4   int getch(void);
5   int getche(void);
6
7   #endif
```

cli.c

```
1   #include "conio.h"
2   #include "hangman_cli.h"
3   #include "word.h"
4   #include <stdio.h>
5   #include <stdlib.h>
6
7   int main()
8   {
9       char *word_of_the_game, prompt = 'y';
10      int difficult = 0;
11      FILE* words = read_file(difficult);
```

```
12      while (prompt != 'n') {
13          system("clear");
14          word_of_the_game = word_select(words);
15          printf("Word of the game: %s.\n", word_of_the_game);
16          Hangman(word_of_the_game);
17
18          printf("Another try? [y/n] ");
19          prompt = getche();
20      }
21      printf("\n");
22      return 0;
23  }
```

word_test.c

```
1   #include "assert.h"
2   #include "word.h"
3   #include <stdio.h>
4   #include <string.h>
5
6   int fake_random()
7   {
8       return 1;
9   }
10
11  /* тесты на функцию get_random_word_num */
12  void test_get_random_word_num()
13  {
14      assert(get_random_word_num(2, fake_random) == 1);
```

```
15  }
16
17  /* тесты на функцию get_filepath_by_difficult */
18  void test_get_filepath_by_difficult()
19  {
20      assert(strcmp(get_filepath_by_difficult(0), "./iwi/words/just.txt") ==
    0);
21
22      assert(strcmp(get_filepath_by_difficult(1), "./iwi/words/easy.txt") ==
    0);
23      assert(strcmp(get_filepath_by_difficult(2), "./iwi/words/medium.txt")
    == 0);
24
25      assert(strcmp(get_filepath_by_difficult(3), "./iwi/words/hard.txt") ==
    0);
26      assert(get_filepath_by_difficult(-1) == NULL);
27      assert(get_filepath_by_difficult(100) == NULL);
28  }
29
30  /* тесты на функцию get_word_amount */
31  void test_get_word_amount()
32  {
33      FILE* words;
34
35      words = fopen("./src/word_test_data/words/1.txt", "r");
36      assert(get_words_amount(words) == 1);
37      fclose(words);
38
39      words = fopen("./src/word_test_data/words/2.txt", "r");
40      assert(get_words_amount(words) == 2);
41      fclose(words);
42
43      words = fopen("./src/word_test_data/words/3.txt", "r");
44      assert(get_words_amount(words) == 3);
```

```
45      fclose(words);
46  }
47
48  /* тесты на функцию get_word_length */
49  void test_get_word_length()
50  {
51      assert(get_word_length("1\n") == 1);
52      assert(get_word_length("123\n") == 3);
53  }
54
55  int fake_random_return_0()
56  {
57      return 0;
58  }
59
60  int fake_random_return_2()
61  {
62      return 2;
63  }
64
65  int fake_random_return_4()
66  {
67      return 4;
68  }
69
70  /* тесты на функцию word_select_with_rand */
71  void test_word_select_with_rand()
72  {
73      FILE* words;
74      char* word;
```

```
75
76      words = fopen("./src/word_test_data/words/5.txt", "r");
77      word = word_select_with_rand(words, fake_random_return_0);
78      assert(strcmp(word, "one") == 0);
79      fclose(words);
80
81      words = fopen("./src/word_test_data/words/5.txt", "r");
82      word = word_select_with_rand(words, fake_random_return_2);
83      assert(strcmp(word, "three") == 0);
84      fclose(words);
85
86      words = fopen("./src/word_test_data/words/5.txt", "r");
87      word = word_select_with_rand(words, fake_random_return_4);
88      assert(strcmp(word, "five") == 0);
89      fclose(words);
90  }
91
92  int main()
93  {
94      test_get_random_word_num();
95      test_get_filepath_by_difficult();
96      test_get_word_amount();
97      test_get_word_length();
98      test_word_select_with_rand();
99      printf("all tests passed\n");
100     return 0;
101 }
```

hangman_cli_test.c

```
1   #include "assert.h"
```

```
2
3   #include "hangman_cli.h"
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <string.h>
7
8   /*тест на функцию sym_used*/
9   void test_sym_used()
10  {
11      char sym = 'a', *str = "abcd";
12      assert(sym_used(sym, str) == 1);
13
14      sym = 'f';
15      assert(sym_used(sym, str) == 0);
16  }
18
19  /*тест на функцию check_symbol*/
20  void test_check_symbol()
21  {
22      char symbol = 'a', *word = "cats";
23      assert(check_symbol(symbol, word) == 1);
24
25      symbol = 'c';
26      assert(check_symbol(symbol, word) == 1);
27
28      symbol = 'd';
29      assert(check_symbol(symbol, word) == 0);
30  }
31
32  /*тест на функцию add_symbol_to_hidden_word*/
```

```
33  void test_add_symbol_to_hidden_word()
34  {
35      char symbol = 'a', *word = "cats";
36      char* hidden_word = (char*)malloc(4 * sizeof(char));
37      strcat(hidden_word, "----");
38      add_symbol_to_hidden_word(word, symbol, hidden_word);
39      assert(strcmp(hidden_word, "-a--") == 0);
40  }
41
42  /*тест на функцию add_symbol_to_hidden_word 2*/
43  void test_add_symbol_to_hidden_word_2()
44  {
45      char symbol = 'a', *word = "cats";
46      char* hidden_word = (char*)malloc(4 * sizeof(char));
47      strcat(hidden_word, "----");
48      add_symbol_to_hidden_word(word, symbol, hidden_word);
49      assert(strcmp(hidden_word, "-a--") == 0);
50      symbol = 'c';
51      add_symbol_to_hidden_word(word, symbol, hidden_word);
52      assert(strcmp(hidden_word, "ca--") == 0);
53  }
54
55  /* тест на функцию win_or_lose */
56  void test_win_or_lose()
57  {
58      int err_amount = 0;
59      assert(win_or_lose(err_amount) == 1);
60
61      err_amount = 5;
62      assert(win_or_lose(err_amount) == 0);
```

```
}

int main()
{
    test_sym_used();
    test_check_symbol();
    test_add_symbol_to_hidden_word();
    test_add_symbol_to_hidden_word_2();
    test_win_or_lose();
    printf("all tests passed\n");
    return 0;
}
```