DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KATHMANDU UNIVERSITY

COMP 102: STRUCTURED PROGRAMMING
PROJECT REPORT : BINGO

Submitted by

Group K
Nisham Ghimire
Ashwin Sapkota
Sashank Baral
Aayush Bhatta

Department of Computer Science and Engineering
School of Engineering, Kathmandu University

Submitted to

Manish Joshi / Pratit Raj Giri

Lecturer
Department of Computer Science and Engineering
School of Engineering, Kathmandu University

Date: August 17, 2021

Contents

1. Introduction…………………………………………………………..... 3
2. Case Study of similar applications………………………………. 4
3. Libraries used……………………………………………………..... 8
4. Algorithms and flowcharts of the program……………………..… 9
5. Features…………………………………………………………….. 17
6. Conclusion with future enhancement …………………………..... 19
7. Code with some snippets of the output screen ………………… 21
8. References………………………………………………………….. 47

Introduction

The C Programming Language

The C programming language is a procedural computer programming language developed in the early 1970s in the Bell Laboratories by Dennis Ritchie. It was designed to be a successor to the programming language B. Over the years, C has evolved into a language used to design system software. Due to its closeness with the hardware, the C programming language is well-suited for low level development. C also provides dynamic memory management features that allow developers to design their programs to be more efficient and fast. It is also used in scientific computing where a good management of memory and speed is crucial.

The Bingo Game

Bingo is a board game played using a 5x5 grid. Players numbering anywhere from 2 to 5 can play the game at the same time. Each player gets a grid, all of which are initialized with the same set of 25 numbers, although in different order in each grid. Players then take turns to call out numbers which are crossed out on each player's grid. Once the numbers in a row, column, or a diagonal are all crossed out, a letter in the word "BINGO" at the top of the grid is eliminated. The player who succeeds in eliminating all five letters of the word
first wins the game.

Study of Similar Applications

Tic tac toe

Tic-tac-toe (America), noughts and crosses (English and British English), or Aalu-cross(Nepali) is a paper-and-pencil game for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a diagonal, horizontal, or vertical row is the winner. It is a solved game with a forced draw assuming best play from both players.

Rules of the Game

- The game is to be played between two people.
- One of the players chooses 'O' and the other 'X' to mark their respective cells.
- The game starts with one of the players and the game ends when one of the players has one whole row/ column/ diagonal filled with his/her respective character ('O' or 'X').
- If no one wins, then the game is said to be draw

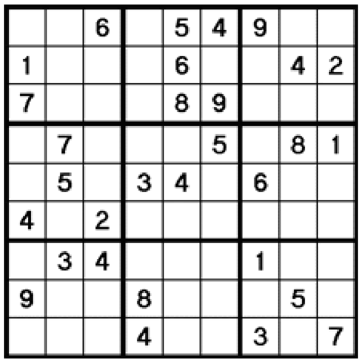
Similarity with bingo game

- Both of the games use a grid to separate spaces between the objects like "X", "O", or numbers.
- The user is required to fill in the grid to play.

- The winning conditions of both games are similar. In both the games the player has to fill the row, column or diagonal to win.

<u>Sudoku</u>

Sudoku is played on a grid of 9 x 9 spaces. Within the rows and columns are 9 “squares” (made up of 3 x 3 spaces). Each row, column and square (9 spaces each) needs to be filled out with the numbers 1-9, without repeating any numbers within the row, column or square. Each Sudoku grid comes with a few spaces already filled in; the more spaces filled in, the easier the game – the more difficult Sudoku puzzles have very few spaces that are already filled in. They are stacked nine vertically and nine horizontally, making 81 cells total. The puzzle comes with some of the cells (usually less than half of them) already filled in, like this:



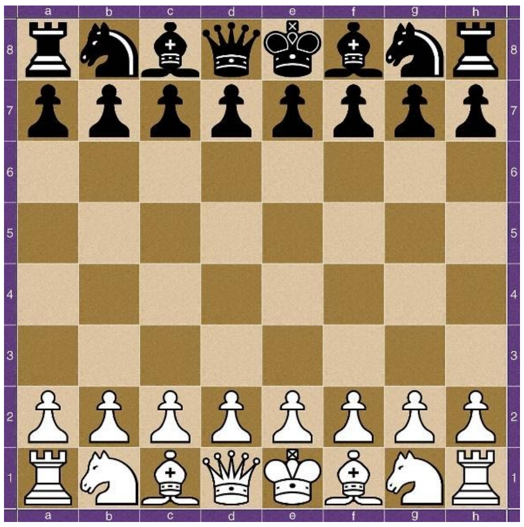
		6		5	4	9		
1				6			4	2
7				8	9			
	7				5		8	1
	5		3	4		6		
4		2						
	3	4				1		
9			8				5	
			4			3		7

The rule of the game is simple, the player has to fill in all the vacant spaces with the numbers 1-9 in such a way that the numbers do not repeat in any row, column or squares. But it can be very hard to solve due to the condition to not repeat any numbers.
This game can be created similarly to the bingo game we created.
Similarity with bingo game:

- Both of the games use a grid and require the user to play or input the numbers.

Chess

Chess is a recreational and competitive board game played between two players. Chess is an abstract strategy game and involves no hidden information. It is played on a square chessboard with 64 squares arranged in an eight-by-eight grid. At the start, each player (one controlling the white pieces, the other controlling the black pieces) controls sixteen pieces: one king, one queen, two rooks, two knights, two bishops, and eight pawns. The object of the game is to checkmate the opponent's king, whereby the king is under immediate attack (in "check") and there is no way for it to escape. There are also several ways a game can end in a draw.
One of the goals of early computer scientists was to create a chess-playing machine. In 1997, Deep Blue became the first computer to beat the reigning World Champion in a match when it defeated Garry Kasparov. Though not flawless, today's chess engines are significantly stronger than even the best human players, and have deeply influenced the development of chess theory.



Rules:

Chess pieces are divided into two different colored sets. While the sets may not be literally white and black (e.g. the light set may be a yellowish or off-white color, the dark set may be brown or red), they are always referred

to as "white" and "black". The players of the sets are referred to
as White and Black, respectively. Each set consists of 16 pieces: one king, one queen, two rooks, two bishops, two knights, and eight pawns.
The game is played on a square board of eight rows (called ranks) and eight columns (called files). By convention, the 64 squares alternate in color and are referred to as light and dark squares; common colors for chess boards are white and brown, or white and dark green.
The pieces are set out as shown in the diagram and photo. Thus, on White's first rank, from left to right, the pieces are placed in the following order: rook, knight, bishop, queen, king, bishop, knight, rook. On the second rank is placed a row of eight pawns. Black's position mirrors White's, with an equivalent piece on the same file. The board is placed with a light square at the right-hand corner nearest to each player. The correct positions of the king and queen may be remembered by the phrase "queen on her own color" ─ i.e. the white queen begins on a light square; the black queen on a dark square.

Similarity with bingo:

- Both games are played on a grid.
- Both the games can be created using C programming

Libraries used

GTK (Gimp Tool Kit)

GTK is a library used for graphical user interfaces (GUIs). It provides a number of functions which can be used to create graphical widgets such as buttons, text entry fields, labels, etc. It can be included in a program using the gtk.h file that comes with the GTK package. GTK is an event-driven system, which means that applications built with GTK continuously look for events in a main loop. Widgets may be programmed to react to certain events by emitting *signals*. These signals can then be connected to *callback* functions (event handlers).

In this project, GTK has been used to design the user interface (UI) of the Bingo game. It uses GTK widgets such as text entry fields to ask the user for input, buttons to create the clickable grid (the game board), and labels to let players know when it’s their turn.

C standard libraries

The GTK library provides access to C standard library functions, which alleviates the need to explicitly include them in a GTK program. This project uses the stdlib.h header through GTK which provides access to the Standard Library. The standard library provides access to memory management functions such as malloc, calloc, realloc, and free. malloc has been used to allocate memory for structures in this project.

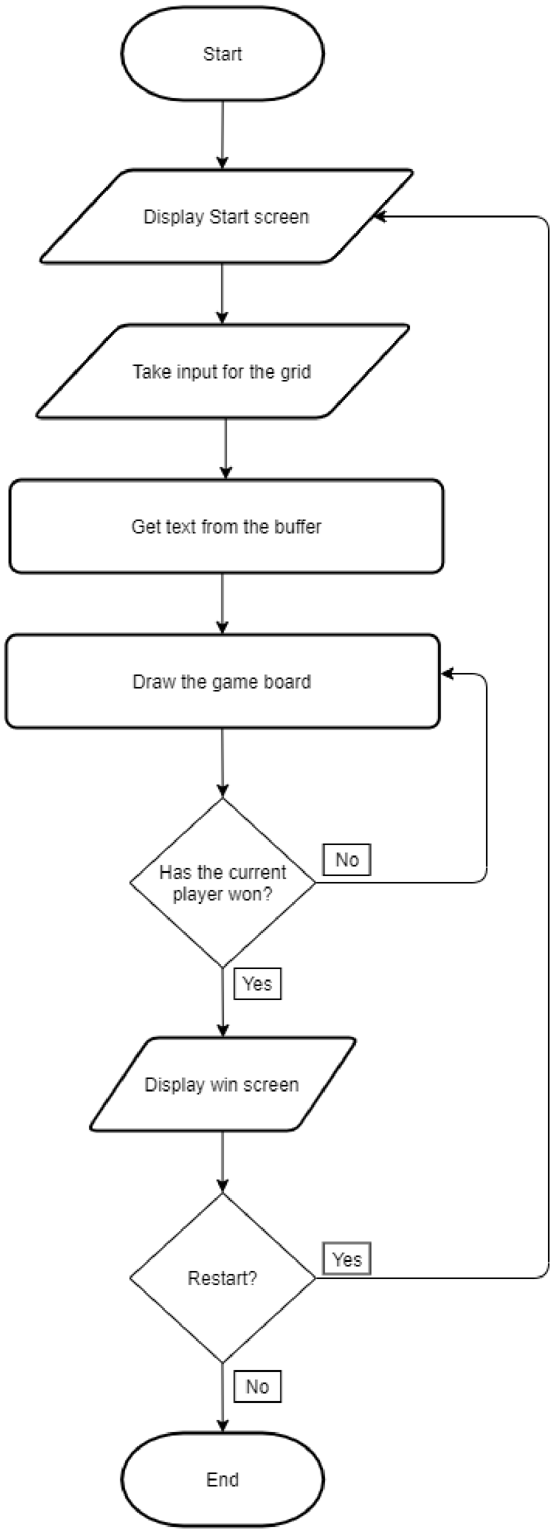
Flowcharts and algorithms

General algorithm and flowchart

<u>Algorithm:</u>

Step 1: Start
Step 2: Display Start screen
Step 3: Take input for the grid
Step 4: Get text from the buffer
Step 5: Draw the game board
Step 6: Has the current player won? If yes, go to step 7, else go to step 5.
Step 7: Display win screen
Step 8: Restart? If yes, go to step 2, else go to step 9.
Step 9: End

Flowchart:


Start
Display Start screen
Take input for the grid
Get text from the buffer
Draw the game board
Has the current player won?
No
Yes
Display win screen
Restart?
Yes
No
End

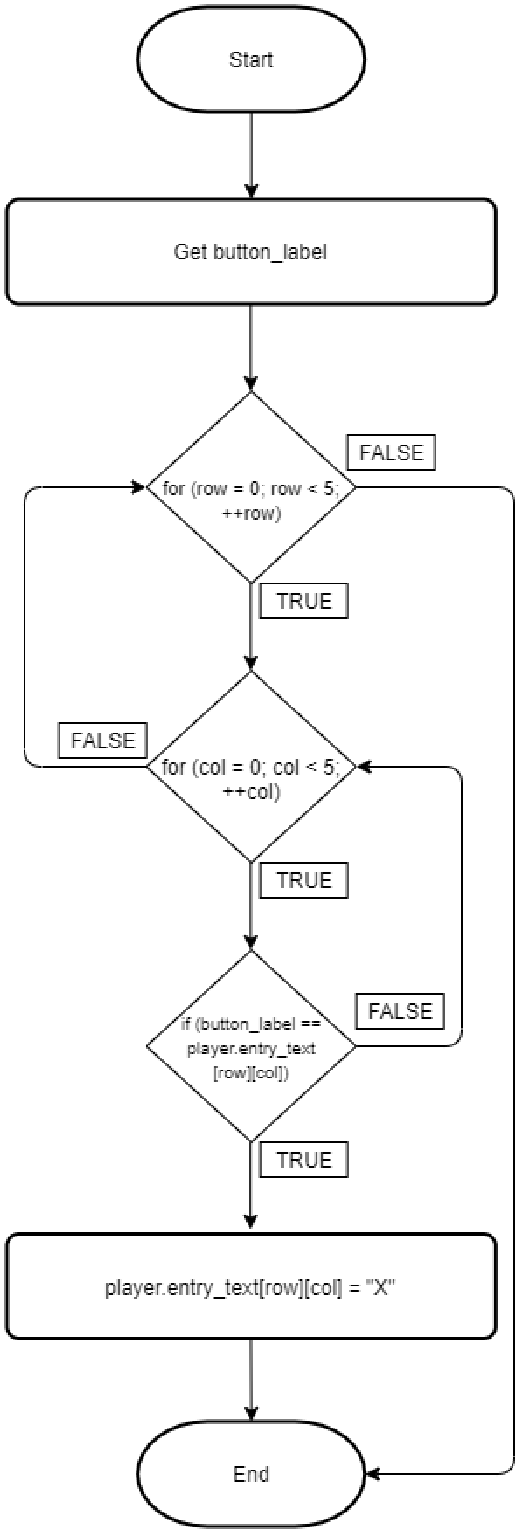
Algorithm and flowchart for replacing the label on the button

The same algorithm is used to replace the button labels of both players.

<u>Algorithm:</u>

```
Step 1: Start
Step 2: get button_label
Step 3: If for (row = 0; row < 5; ++row) is true, go to Step 4, else go to Step 6
Step 4: If for (col = 0; col < 5; ++col) is true, go to Step 5, else continue
Step 5: If (button_label == player.entry_text[row][col]) set player.entry_text[row][col] = "X", else continue
Step 6: End
```

Flowchart:


Start
Get button_label
for (row = 0; row < 5; ++row)
FALSE
TRUE
for (col = 0; col < 5; ++col)
FALSE
TRUE
if (button_label == player.entry_text [row][col])
FALSE
TRUE
player.entry_text[row][col] = "X"
End

Algorithm and flowchart to check if the current player has won

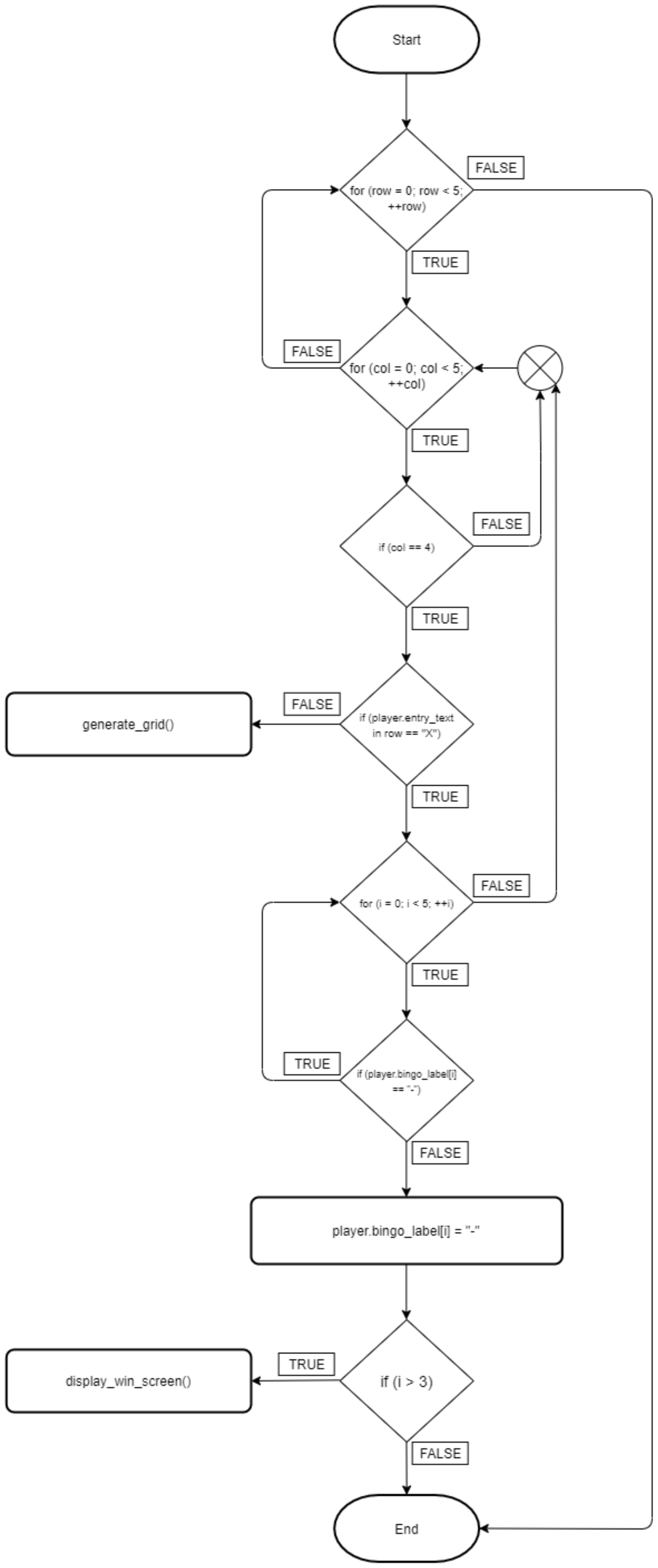
<u>For rows and columns:</u>

The same algorithm is used to check the winning criterion in rows and columns.

<u>Algorithm:</u>

```
Step 1: Start
Step 2: If for (row = 0; row < 5; ++row) is true, go to Step 3, else go to Step 9
Step 3: If for (col = 0; col < 5; ++col) is true, go to Step 4, else continue
Step 4: If (col == 4) go to Step 5, else continue
Step 5: If (player.entry_text in row == "X") go to Step 6, else generate_grid()
Step 6: If for (i = 0; i < 5; ++i) is true, go to Step 7, else continue
Step 7: If (player.bingo_label[i] == "-") continue, else set player.bingo_label[i] = "-"
Step 8: If (i > 3) display_win_screen(), else go to Step 9
Step 9: End
```

Flowchart:


Start
for (row = 0; row < 5; ++row)
FALSE
TRUE
for (col = 0; col < 5; ++col)
FALSE
TRUE
if (col == 4)
FALSE
TRUE
if (player.entry_text in row == "X")
FALSE
generate_grid()
TRUE
for (i = 0; i < 5; ++i)
FALSE
TRUE
if (player.bingo_label[i] == "-")
TRUE
FALSE
player.bingo_label[i] = "-"
if (i > 3)
TRUE
display_win_screen()
FALSE
End

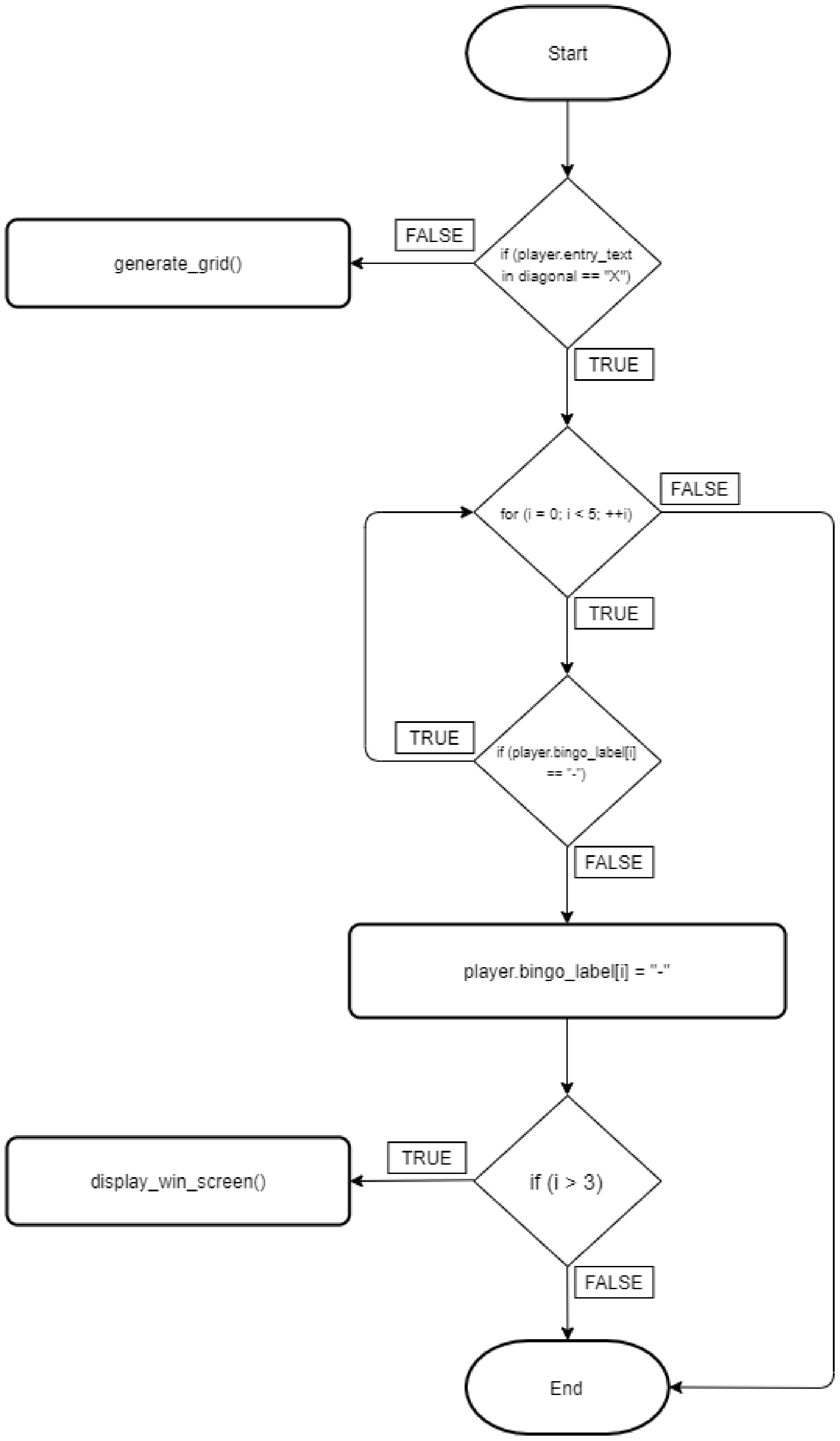
For diagonals:

The same algorithm is used to check both diagonals of the grid.

Algorithm:

```
Step 1: Start
Step 2: If (player.entry_text in diagonal == "X") go to Step 3, else generate_grid()
Step 3: If for (i = 0; i < 5; ++i) is true, go to Step 4, else go to Step 6
Step 4: If (player.bingo_label[i] == "-") continue, else set player.bingo_label[i] = "-"
Step 5: If (i > 3) display_win_screen(), else go to Step 6
Step 6: End
```

Flowchart:


Start
if (player.entry_text in diagonal == "X")
FALSE
generate_grid()
TRUE
for (i = 0; i < 5; ++i)
FALSE
TRUE
if (player.bingo_label[i] == "-")
TRUE
FALSE
player.bingo_label[i] = "-"
if (i > 3)
TRUE
display_win_screen()
FALSE
End

Features

Graphical user interface (GUI)

- The program is built with a graphical user interface. This program uses GTK widgets to create and insert different types of buttons, pictures, and grids so that the user can easily interact with the program and use it without getting disturbed by its environment. Any user can easily familiarize with the program and use it.

Multiplayer support

- This program can be used by two players at a time.

User Friendly

- Users can make their own grid. The grid can be made directly by input given by the users. The input can be numbers or alphabets (whatever the user prefers). Anyone playing this game can get the same experience as the paper-based one.

Faster Execution

- As the program requires less memory and can be executed within no time. It does not take a lot of time for the declaration of the winner. It declares the winner at the time the game is finished. Also it won’t take time to switch from one screen to another screen and anyone using the program won’t feel lagging.

Easy to play

- To use this program won't require special knowledge. So, anyone who can operate a computer can easily use this program and play this game. This game is easy to play and any users can play this game easily.

<u>No Interruption</u>

- There is no interruption in this game. Users can play this game without getting interruptions like: warnings,ads or errors.

<u>Easy to Understand</u>

- The source code of the game can be understood easily by the users and can be modified as the user wants. To create any change in the source code of the program you won’t require any special knowledge, if you have the knowledge of structured programming. So, anyone with the knowledge of structured programming can modify or understand the source code of the program with ease.

PROJECT LIMITATIONS AND FUTURE ENHANCEMENTS

There are a lot of features and functionalities that can be integrated in the proposed app, but the project scope has been limited to diligently resolve the problems. The project objective must be achieved pertaining to the Time Constraint applied in accordance with the defined functionality of the system. However, features that are not included in the system can be considered as future enhancements. The limiting areas of the project contributing for enhancement thus are as follows:

There's a tiny flaw in the application. When the user does not put an entry in the grid, the program is still valid and the player one wins the game. This can then be changed or updated in the future, requiring the user to provide an entry to complete the grid as a requirement needed to move to the next page.

Currently, the system is geared towards incorporating functionality that allows users to create grids for the game but does not implement the idea of automatically generating grids for players to make the game fun, fair, and fast. It can help reduce the game start time because you don't need to waste your efforts creating the other person's grid.

Until now the game is implemented to run offline in the same system but can be further enhanced to run on the internet using the net. The idea of a multiplayer online game allows the user to play the game from anywhere in the world.

Due to time and resource constraints, the concept of more than two players is not currently available in the game. We may add a feature that allows more than two users in a game so that a group of people can play together as a means of entertainment.

The current application is a simple demonstration of how to implement the bingo concept in C programming to create a game. Later, we can make the application more interactive by adding different effects. We can explode the confetti when the user wins with the sound of the cheering crowd saying Bingo. To make it more visually appealing, the finished line can be switched to different colors. In addition, some background sounds can be added to the game, which makes the game exciting.

Since then, social media has become extremely popular these days. Bingo apps can be connected to Facebook like Candy Crush, the hugely successful app on Facebook. With the Facebook login function, any user can invite his or her friends to play. In addition, attractive bonuses can be added to the game, which makes the game interesting among gamers. Since everyone has different preferences, having one color defined may not be right for everyone, so we can add a color toggle button that allows the user to change the color of the interface based on his preferences.

Code with some snippets of the output screen

main.c

```c
#include "functions.h"
#include <gtk/gtk.h>

int main(int argc, char **argv) {

    GtkApplication *app;
    int status;

    app = gtk_application_new("com.attempt.one", G_APPLICATION_FLAGS_NONE);
    g_signal_connect(app, "activate", G_CALLBACK(activate), NULL);
    status = g_application_run(G_APPLICATION(app), argc, argv);
    g_object_unref(app);

    return status;
}
```

activate.c

```c
#include "functions.h"
#include <gtk/gtk.h>
//#include "main.h"

void activate(GApplication *app, gpointer user_data) {
    // definitons
    GtkWidget *win;

    // drawing window
    win = gtk_application_window_new(GTK_APPLICATION(app));
    gtk_window_set_title(GTK_WINDOW(win), "BINGO");
    gtk_window_set_default_size(GTK_WINDOW(win), 800, 700);
    gtk_window_set_resizable(GTK_WINDOW(win), FALSE);

    first_screen(win);

    // rendering
    gtk_widget_show(win);
}
```

button_callbacks.c

```
#include "functions.h"
#include "main.h"
#include <gtk/gtk.h>

void hide_first(GtkButton *button, gpointer user_data) {
    struct udata *data = user_data;
    gtk_widget_hide(data->mainBox);
    player_one(data);
}
void replace(GtkWidget *widget, gpointer user_data) {
    struct udata *data = user_data;
    const gchar *btn_label = gtk_button_get_label(GTK_BUTTON(widget));
    for (int row = 0; row < 5; ++row) {
      for (int col = 0; col < 5; ++col) {
        if (g_strcmp0(global_player_one_data->entry_text[row][col], btn_label) ==
0) {
          global_player_one_data->entry_text[row][col] = "X";
          break;
        }
      }
    }
    for (int row = 0; row < 5; ++row) {
      for (int col = 0; col < 5; ++col) {
        if (g_strcmp0(global_player_two_data->entry_text[row][col], btn_label) ==
0) {
          global_player_two_data->entry_text[row][col] = "X";
          break;
        }
      }
    }
    //gtk_button_set_label(GTK_BUTTON(widget), "X");
    if (data->player_id == 1) {
      check_win(data);
    } else { check_win_two(data);
    }
}
```

player_one.c

```c
#include "functions.h"
#include "main.h"
#include <gtk/gtk.h>

void player_one(gpointer user_data) {
    struct udata *data = user_data;
    data->player_id = 1;
    draw_input_grid(data);
}

void player_one_new(gpointer user_data) {
  struct udata *data = user_data;
  gtk_widget_hide(data->box);
  player_two(data);
}

void player_one_generate(GtkWidget *widget, gpointer user_data) {
  struct udata *data = user_data;
  get_entry_text(data);
  player_one_new(data);
}
```

check_win.c

```c
#include "functions.h"
#include "main.h"
#include <gtk/gtk.h>

void new_game(GtkWidget *widget, gpointer user_data) {
    struct udata *data = user_data;

    GtkWidget *win = data->window;

    first_screen(win);
}

void display_win_screen(gpointer user_data) {
    run_count_one = 0;
    run_count_two = 0;
    struct udata *data = user_data;
    gtk_widget_unrealize(data->box);
    GtkWidget *box;

    box = gtk_box_new(GTK_ORIENTATION_VERTICAL, 5);
    gtk_window_set_child(GTK_WINDOW(data->window), box);

    gtk_widget_set_halign(box, GTK_ALIGN_CENTER);
    gtk_widget_set_valign(box, GTK_ALIGN_CENTER);

    GtkWidget *label;

    if (data->player_id == 1) {
      label = gtk_label_new("Player 1 wins!");
      gtk_widget_set_name(label, "winLabel");
    } else {
      label = gtk_label_new("Player 2 wins!");
      gtk_widget_set_name(label, "winLabel");
    }

    GtkWidget *btn;
    btn = gtk_button_new_with_label("NEW GAME");
    g_signal_connect(btn, "clicked", G_CALLBACK(new_game), data);
    gtk_button_set_has_frame(GTK_BUTTON(btn), FALSE);
    gtk_widget_set_name(btn, "button");

    gtk_box_append(GTK_BOX(box), label);
    gtk_box_append(GTK_BOX(box), btn);
}

void check_win(gpointer user_data) {
  struct udata *data = user_data;
  if (!((g_strcmp0(data->label[0], "-") == 0) && (g_strcmp0(data->label[1], "-")
== 0) && (g_strcmp0(data->label[2], "-") == 0) && (g_strcmp0(data->label[3], "-")
== 0) && (g_strcmp0(data->label[4], "-") == 0))) {
    const gchar* s[5][5];
    static int row_status[5] = {0, 0, 0, 0, 0};
    static int col_status[5] = {0, 0, 0, 0, 0};
    static int diag_status[2] = {0, 0};

    if (run_count_one == 0) {
      for (int i = 0; i < 5; ++i) {
        row_status[i] = 0;
        col_status[i] = 0;
      }
      diag_status[0] = 0;
      diag_status[1] = 0;
    }

    for (int row = 0; row < 5; ++row) {
      for (int col = 0; col < 5; ++col) {
        s[row][col] = data->entry_text[row][col];
      }
    }

    for (int row = 0; row < 5; ++row) {
      for (int col = 0; col < 5; ++col) {
        if (col == 4) {
          if ((g_strcmp0(s[row][0], "X") == 0) && (g_strcmp0(s[row][1], "X") ==
0) && (g_strcmp0(s[row][2], "X") == 0) && (g_strcmp0(s[row][3], "X") == 0) &&
(g_strcmp0(s[row][4], "X") == 0)) {
            if (row_status[row] != 1) {
              for (int i = 0; i < 5; ++i) {
                if ((g_strcmp0(data->label[i], "-") == 0)) {
                  continue;
                } else {
                  data->label[i] = "-";
                  run_count_one++;
                  if (i > 3) display_win_screen(data);
                  row_status[row] = 1;
                  break;
                }
              }
            }
          } else {
            generate_grid(global_player_two_data);
          }
        } else {
          continue;
        }

        if (col == 4) {
          if ((g_strcmp0(s[0][row], "X") == 0) && (g_strcmp0(s[1][row], "X") ==
0) && (g_strcmp0(s[2][row], "X") == 0) && (g_strcmp0(s[3][row], "X") == 0) &&
(g_strcmp0(s[4][row], "X") == 0)) {
            if (col_status[row] != 1) {
              for (int i = 0; i < 5; ++i) {
                if ((g_strcmp0(data->label[i], "-") == 0)) {
                  continue;
                } else {
                  data->label[i] = "-";
                  run_count_one++;
                  if (i > 3) display_win_screen(data);
                  col_status[row] = 1;
                  break;
                }
              }
            }
          } else {
            generate_grid(global_player_two_data);
          }
        } else {
          continue;
        }
      }
    }

    if ((g_strcmp0(s[0][0], "X") == 0) && (g_strcmp0(s[1][1], "X") == 0) &&
(g_strcmp0(s[2][2], "X") == 0) && (g_strcmp0(s[3][3], "X") == 0) &&
(g_strcmp0(s[4][4], "X") == 0)) {
          if (diag_status[0] != 1) {
              for (int i = 0; i < 5; ++i) {
                  if ((g_strcmp0(data->label[i], "-") == 0)) {
                      continue;
                  } else {
                      data->label[i] = "-";
                      run_count_one++;
                      if (i > 3) display_win_screen(data);
                      diag_status[0] = 1;
                      break;
                  }
              }
          }
      } else {
        generate_grid(global_player_two_data);
      }
      if ((g_strcmp0(s[0][4], "X") == 0) && (g_strcmp0(s[1][3], "X") == 0) &&
(g_strcmp0(s[2][2], "X") == 0) && (g_strcmp0(s[3][1], "X") == 0) &&
(g_strcmp0(s[4][0], "X") == 0)) {
          if (diag_status[1] != 1) {
              for (int i = 0; i < 5; ++i) {
                  if ((g_strcmp0(data->label[i], "-") == 0)) {
                      continue;
                  } else {
                      data->label[i] = "-";
                      run_count_one++;
                      if (i > 3) display_win_screen(data);
                      diag_status[1] = 1;
                      break;
                  }
              }
          }
      } else {
        generate_grid(global_player_two_data);
      }
  } else {
    display_win_screen(data);
  }
}

void check_win_two(gpointer user_data) {
  struct udata *data = user_data;
  if (!((g_strcmp0(data->label[0], "-") == 0) && (g_strcmp0(data->label[1], "-")
== 0) && (g_strcmp0(data->label[2], "-") == 0) && (g_strcmp0(data->label[3], "-")
== 0) && (g_strcmp0(data->label[4], "-") == 0))) {
    const gchar* s[5][5];
    static int row_status[5] = {0, 0, 0, 0, 0};
    static int col_status[5] = {0, 0, 0, 0, 0};
    static int diag_status[2] = {0, 0};

    if (run_count_two == 0) {
      for (int i = 0; i < 5; ++i) {
        row_status[i] = 0;
        col_status[i] = 0;
      }
      diag_status[0] = 0;
      diag_status[1] = 0;
    }

    for (int row = 0; row < 5; ++row) {
      for (int col = 0; col < 5; ++col) {
        s[row][col] = data->entry_text[row][col];
      }
    }
    for (int row = 0; row < 5; ++row) {
      for (int col = 0; col < 5; ++col) {
        if (col == 4) {
          if ((g_strcmp0(s[row][0], "X") == 0) && (g_strcmp0(s[row][1], "X") ==
0) && (g_strcmp0(s[row][2], "X") == 0) && (g_strcmp0(s[row][3], "X") == 0) &&
(g_strcmp0(s[row][4], "X") == 0)) {
            if (row_status[row] != 1) {
              for (int i = 0; i < 5; ++i) {
                if ((g_strcmp0(data->label[i], "-") == 0)) {
                  continue;
                } else {
                  data->label[i] = "-";
                  run_count_two++;
                  if (i > 3) display_win_screen(data);
                  row_status[row] = 1;
                  break;
                }
              }
            }
          } else {
            generate_grid(global_player_one_data);
          }
        } else {
          continue;
        }

        if (col == 4) {
          if ((g_strcmp0(s[0][row], "X") == 0) && (g_strcmp0(s[1][row], "X") ==
0) && (g_strcmp0(s[2][row], "X") == 0) && (g_strcmp0(s[3][row], "X") == 0) &&
(g_strcmp0(s[4][row], "X") == 0)) {
            if (col_status[row] != 1) {
              for (int i = 0; i < 5; ++i) {
                if ((g_strcmp0(data->label[i], "-") == 0)) {
                  continue;
                } else {
                  data->label[i] = "-";
                  run_count_two++;
                  if (i > 3) display_win_screen(data);
                  col_status[row] = 1;
```

```
         break;
        }
       }
      }
     } else {
      generate_grid(global_player_one_data);
     }
    } else {
     continue;
    }
   }
  }

  if ((g_strcmp0(s[0][0], "X") == 0) && (g_strcmp0(s[1][1], "X") == 0) &&
(g_strcmp0(s[2][2], "X") == 0) && (g_strcmp0(s[3][3], "X") == 0) &&
(g_strcmp0(s[4][4], "X") == 0)) {
     if (diag_status[0] != 1) {
       for (int i = 0; i < 5; ++i) {
         if ((g_strcmp0(data->label[i], "-") == 0)) {
           continue;
         } else {
           data->label[i] = "-";
           run_count_two++;
           if (i > 3) display_win_screen(data);
           diag_status[0] = 1;
           break;
         }
       }
     }
   } else {
    generate_grid(global_player_one_data);
   }
   if ((g_strcmp0(s[0][4], "X") == 0) && (g_strcmp0(s[1][3], "X") == 0) &&
(g_strcmp0(s[2][2], "X") == 0) && (g_strcmp0(s[3][1], "X") == 0) &&
(g_strcmp0(s[4][0], "X") == 0)) {
     if (diag_status[1] != 1) {
       for (int i = 0; i < 5; ++i) {
         if ((g_strcmp0(data->label[i], "-") == 0)) {
           continue;
```

```
                  } else {
                      data->label[i] = "-";
                      run_count_two++;
                      if (i > 3) display_win_screen(data);
                      diag_status[1] = 1;
                      break;
                  }
              }
          }
      } else {
        generate_grid(global_player_one_data);
      }
  } else {
    display_win_screen(data);
  }
}
```

player_two.c

```c
#include "functions.h"
#include "main.h"
#include <gtk/gtk.h>

void player_two(gpointer user_data) {
    struct udata *data = user_data;

    global_player_two_data = malloc(sizeof(*global_player_two_data));
    global_player_two_data->window = data->window;
    global_player_two_data->mainBox = data->mainBox;
    global_player_two_data->player_id = 2;
    draw_input_grid(global_player_two_data);
}

void player_two_new(gpointer user_data) {
  struct udata *data = user_data;
  gtk_widget_hide(data->box);
  generate_grid(global_player_one_data);
}

void player_two_generate(GtkWidget *widget, gpointer user_data) {
  struct udata *data = user_data;
  get_entry_text(data);
  player_two_new(data);
}
```

globals.c

```c
#include <gtk/gtk.h>
#include "functions.h"
#include "main.h"

struct udata *global_player_one_data;
struct udata *global_player_two_data;

int run_count_one;
int run_count_two;

```

drawing.c

```c
#include "functions.h"
#include "main.h"
#include <gtk/gtk.h>

void generate_grid(gpointer user_data) {
    struct udata *data = user_data;
    static int count = 0;

    GtkWidget *box, *player_id, *grid, *label[5], *btn[5][5];

    box = gtk_box_new(GTK_ORIENTATION_VERTICAL, 5);
    gtk_window_set_child(GTK_WINDOW(data->window), box);

    if (data->player_id == 1) {
      player_id = gtk_label_new("Player 1");
      gtk_widget_set_name(player_id, "playerLabel");
    } else {
      player_id = gtk_label_new("Player 2");
      gtk_widget_set_name(player_id, "playerLabel");
    }
    gtk_box_append(GTK_BOX(box), player_id);

    grid = gtk_grid_new();
    gtk_grid_set_row_spacing(GTK_GRID(grid), 5);
    gtk_grid_set_column_spacing(GTK_GRID(grid), 5);
    gtk_box_append(GTK_BOX(box), grid);
    data->box = box;

    label[0] = gtk_label_new(data->label[0]);
    gtk_widget_set_name(label[0], "gridlabel");
    label[1] = gtk_label_new(data->label[1]);
    gtk_widget_set_name(label[1], "gridlabel");
    label[2] = gtk_label_new(data->label[2]);
    gtk_widget_set_name(label[2], "gridlabel");
    label[3] = gtk_label_new(data->label[3]);
    gtk_widget_set_name(label[3], "gridlabel");
    label[4] = gtk_label_new(data->label[4]);
    gtk_widget_set_name(label[4], "gridlabel");

    for (int i = 0; i < 5; ++i) {
        if (i == 0) gtk_grid_attach(GTK_GRID(grid), label[i], 1, 1, 2, 2);
        else gtk_grid_attach_next_to(GTK_GRID(grid), label[i], label[i-1],
GTK_POS_RIGHT, 2, 2);
    }

    for (int row = 0; row < 5; ++row) {
        for (int col = 0; col < 5; ++col) {
            btn[row][col] =
gtk_button_new_with_label(data->entry_text[row][col]);
            data->button[row][col] = btn[row][col];
            gtk_widget_set_name(btn[row][col], "gridbtn");
            if (row == 0 && col == 0) {
                gtk_grid_attach(GTK_GRID(grid), btn[row][col], 1, 4, 2, 2);
            } else if (row > 0 && col == 0) {
                gtk_grid_attach(GTK_GRID(grid), btn[row][col], col + 1, (row * 4)
+ 2, 2, 2);
            } else if (row > 0 || col > 0) {
                gtk_grid_attach_next_to(GTK_GRID(grid), btn[row][col],
btn[row][col-1], GTK_POS_RIGHT, 2, 2);
            }
            gtk_button_set_has_frame(GTK_BUTTON(btn[row][col]), FALSE);
            g_signal_connect(btn[row][col], "clicked", G_CALLBACK(replace),
data);
        }
    }

    count++;

    gtk_widget_set_halign(box, GTK_ALIGN_CENTER);
    gtk_widget_set_valign(box, GTK_ALIGN_CENTER);
}

void get_entry_text(gpointer user_data) {
  GtkEntryBuffer *buffer_text[5][5];
  const gchar *text[5][5];
  struct udata *data = user_data;

  data->label[0] = "B";
  data->label[1] = "I";
  data->label[2] = "N";
  data->label[3] = "G";
  data->label[4] = "O";

  for (int row = 0; row < 5; ++row) {
      for (int col = 0; col < 5; ++col) {
          buffer_text[row][col] =
gtk_entry_get_buffer(GTK_ENTRY(data->entry[row][col]));
          text[row][col] = gtk_entry_buffer_get_text(buffer_text[row][col]);
          data->entry_text[row][col] = (gchar*)text[row][col];
      }
  }
}

void draw_input_grid(gpointer user_data) {
    GtkEntryBuffer *entry_buffer[5][5];
    const gchar *entry_text[5][5];
    GtkWidget *box, *grid, *label, *entry[5][5], *btn;
    struct udata *secondSc_data = user_data;

    box = gtk_box_new(GTK_ORIENTATION_VERTICAL, 5);
    gtk_window_set_child(GTK_WINDOW(secondSc_data->window), box);
    gtk_widget_set_halign(box, GTK_ALIGN_CENTER);
    gtk_widget_set_valign(box, GTK_ALIGN_CENTER);

    if (secondSc_data->player_id == 1) {
      label = gtk_label_new("Player 1");
      gtk_widget_set_name(label, "playerLabel");
    } else {
      label = gtk_label_new("Player 2");
      gtk_widget_set_name(label, "playerLabel");
    }

    gtk_box_append(GTK_BOX(box), label);

    grid = gtk_grid_new();
    gtk_grid_set_row_spacing(GTK_GRID(grid), 5);
    gtk_grid_set_column_spacing(GTK_GRID(grid), 5);
    gtk_box_append(GTK_BOX(box), grid);

    secondSc_data->box = box;

    for (int row = 0; row < 5; ++row) {
        for (int col = 0; col < 5; ++col) {
            entry[row][col] = gtk_entry_new();
            entry_buffer[row][col] = gtk_entry_buffer_new(NULL, -1);
            gtk_entry_set_buffer(GTK_ENTRY(entry[row][col]),
entry_buffer[row][col]);
            gtk_grid_attach(GTK_GRID(grid), entry[row][col], col + 1, row + 1, 1,
1);
        }
    }

    for (int row = 0; row < 5; ++row) {
        for (int col = 0; col < 5; ++col) {
            secondSc_data->entry[row][col] = entry[row][col];
        }
    }

    btn = gtk_button_new_with_label("OK");
    gtk_grid_attach(GTK_GRID(grid), btn, 3, 6, 1, 1);
    if (secondSc_data->player_id == 1) {
      g_signal_connect(btn, "clicked", G_CALLBACK(player_one_generate),
secondSc_data);
    } else if (secondSc_data->player_id == 2){
      g_signal_connect(btn, "clicked", G_CALLBACK(player_two_generate),
secondSc_data);
    }

}
```

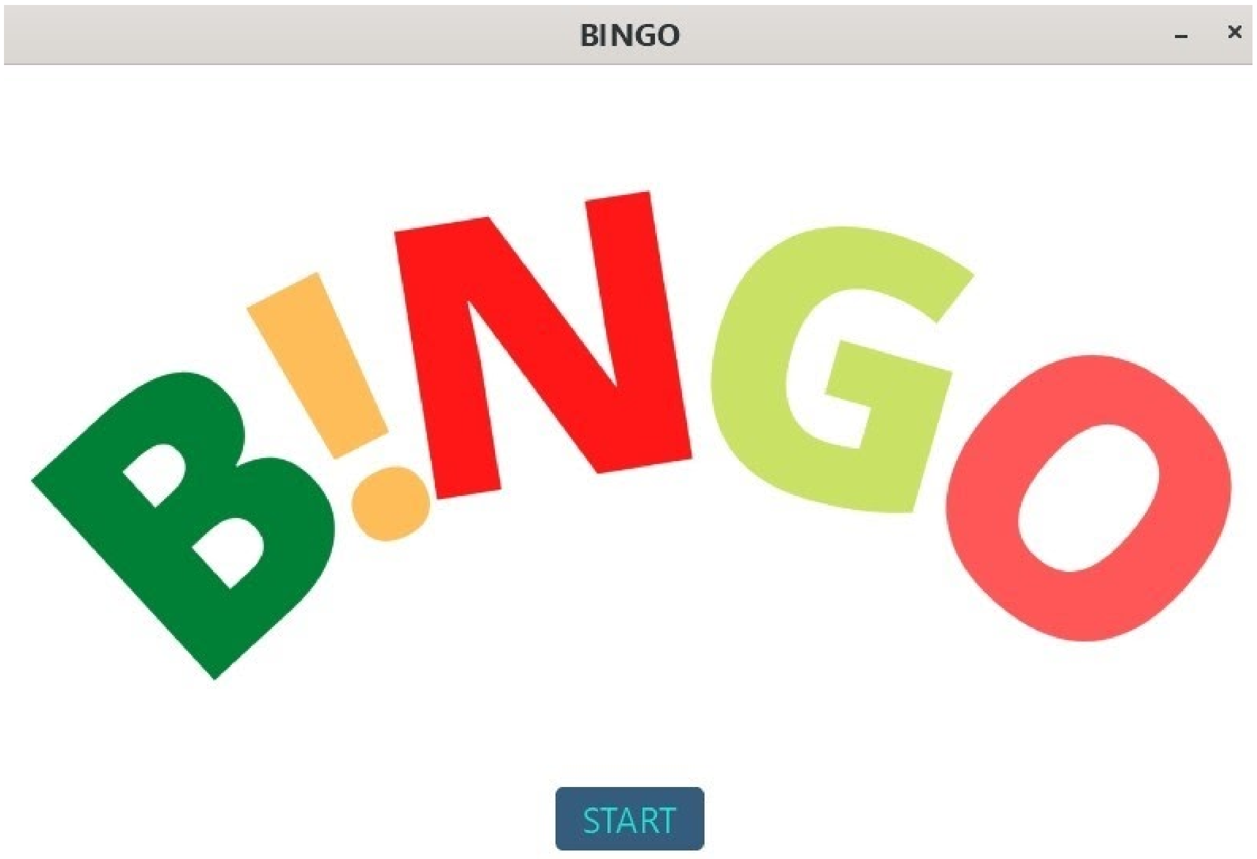
first_screen.c

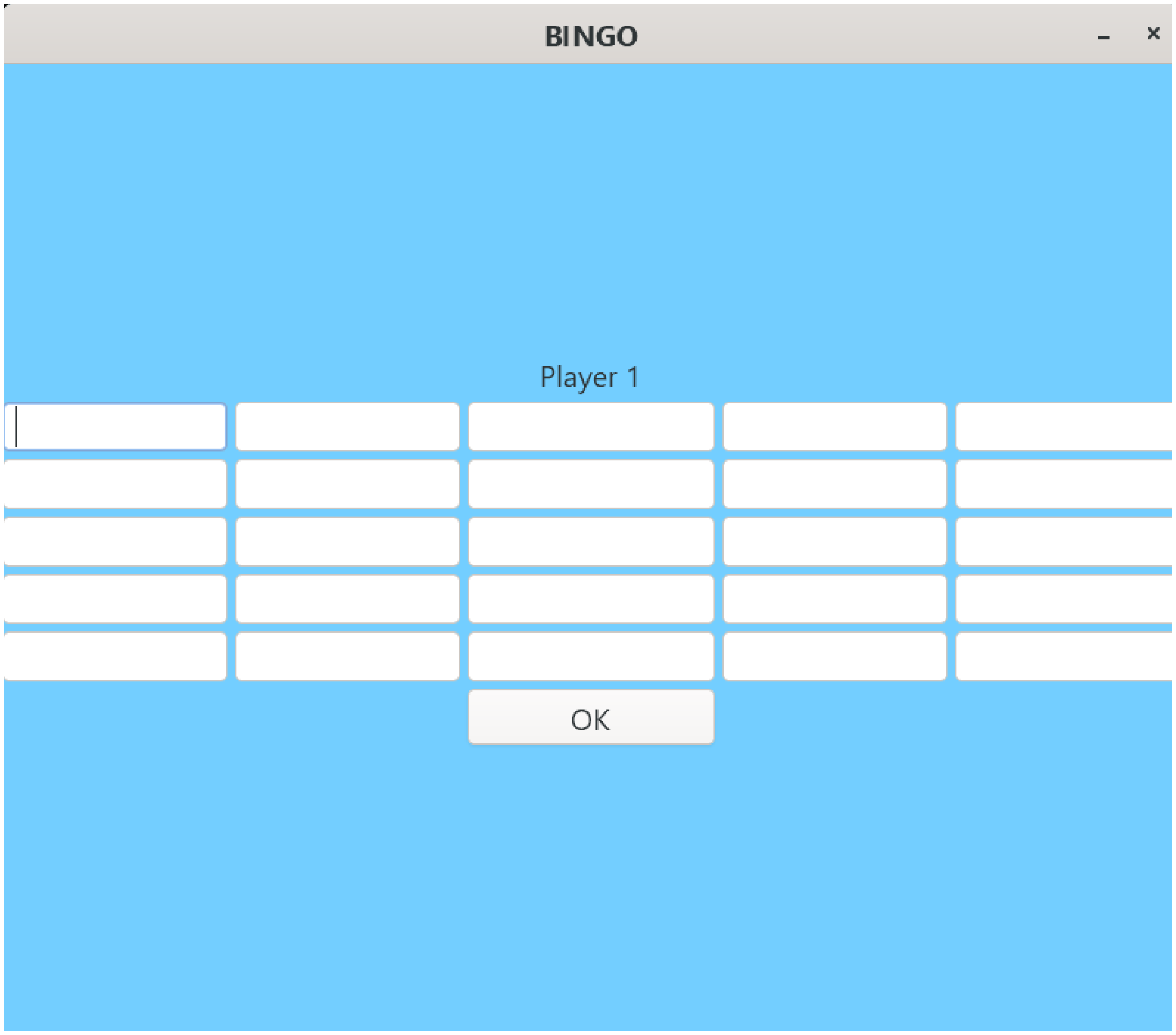
```c
#include "functions.h"
#include <gtk/gtk.h>
#include "main.h"

void first_screen(GtkWidget *win) {
    GtkWidget *mainBox, *buttonBox, *image, *btn, *label;
    GtkCssProvider *cssProvider;

    // intializing widgets
    mainBox = gtk_box_new(GTK_ORIENTATION_VERTICAL, 5);
    gtk_window_set_child(GTK_WINDOW(win), mainBox);
    gtk_widget_set_name(mainBox, "mainBox");

    image = gtk_picture_new_for_filename("img/bingo.png");
    gtk_box_append(GTK_BOX(mainBox), image);

    buttonBox = gtk_box_new(GTK_ORIENTATION_VERTICAL, 5);
    gtk_box_append(GTK_BOX(mainBox), buttonBox);
    gtk_widget_set_halign(buttonBox, GTK_ALIGN_CENTER);

    btn = gtk_button_new_with_label("START");
    gtk_box_append(GTK_BOX(buttonBox), btn);
    gtk_button_set_has_frame(GTK_BUTTON(btn), FALSE);


    // initializing the struct data
    global_player_one_data = malloc(sizeof(*global_player_one_data));
    global_player_one_data->window = win;
    global_player_one_data->mainBox = mainBox;

    // setting names
    gtk_widget_set_name(win, "win");
    gtk_widget_set_name(btn, "button");

    // listening to events
    g_signal_connect(btn, "clicked", G_CALLBACK(hide_first),
global_player_one_data);
```

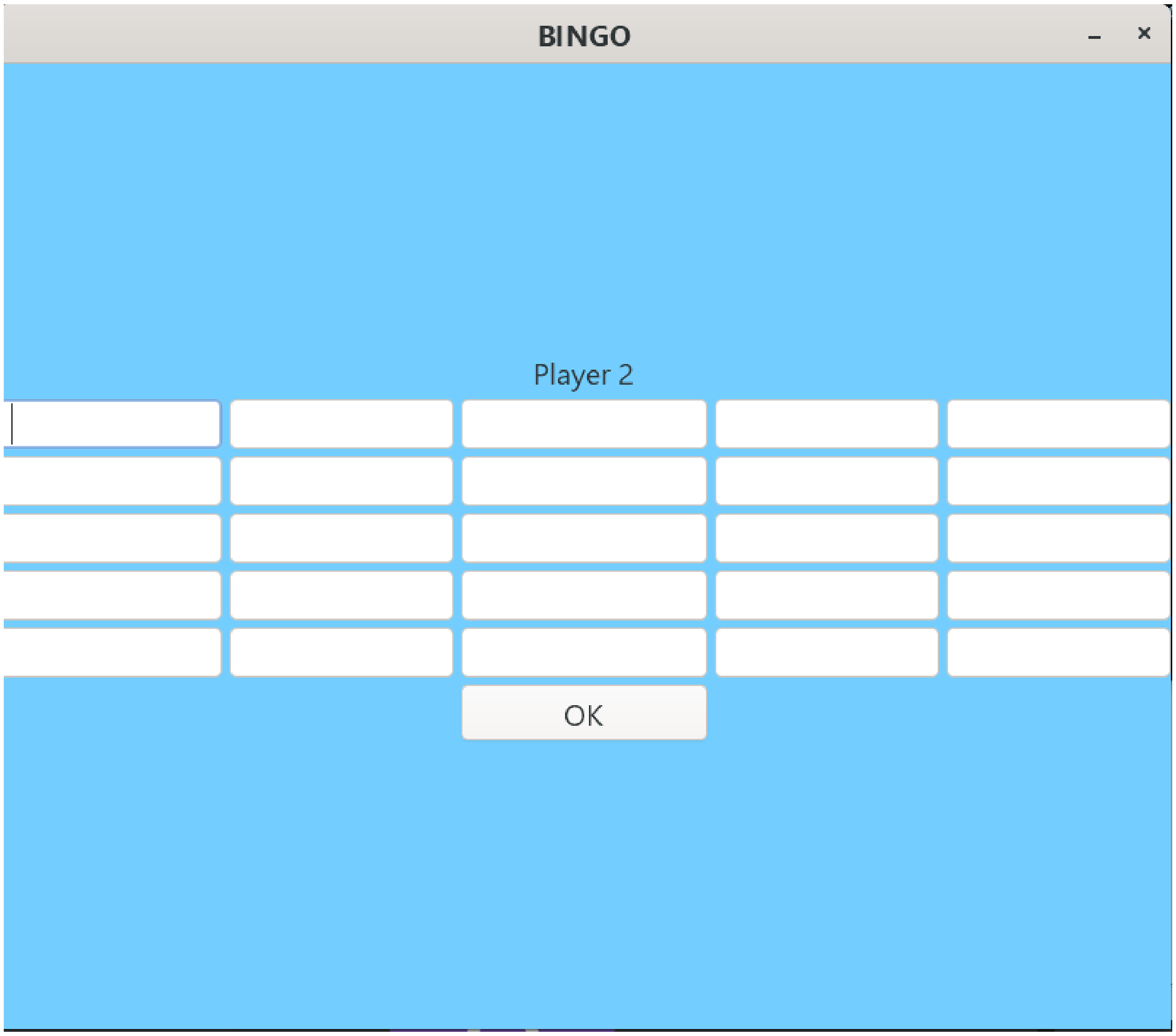
```
    // loading css
    cssProvider = gtk_css_provider_new();
    gtk_css_provider_load_from_path(cssProvider, "css/themes.css");
    gtk_style_context_add_provider_for_display(gdk_display_get_default(),
GTK_STYLE_PROVIDER(cssProvider), GTK_STYLE_PROVIDER_PRIORITY_USER);
}
```

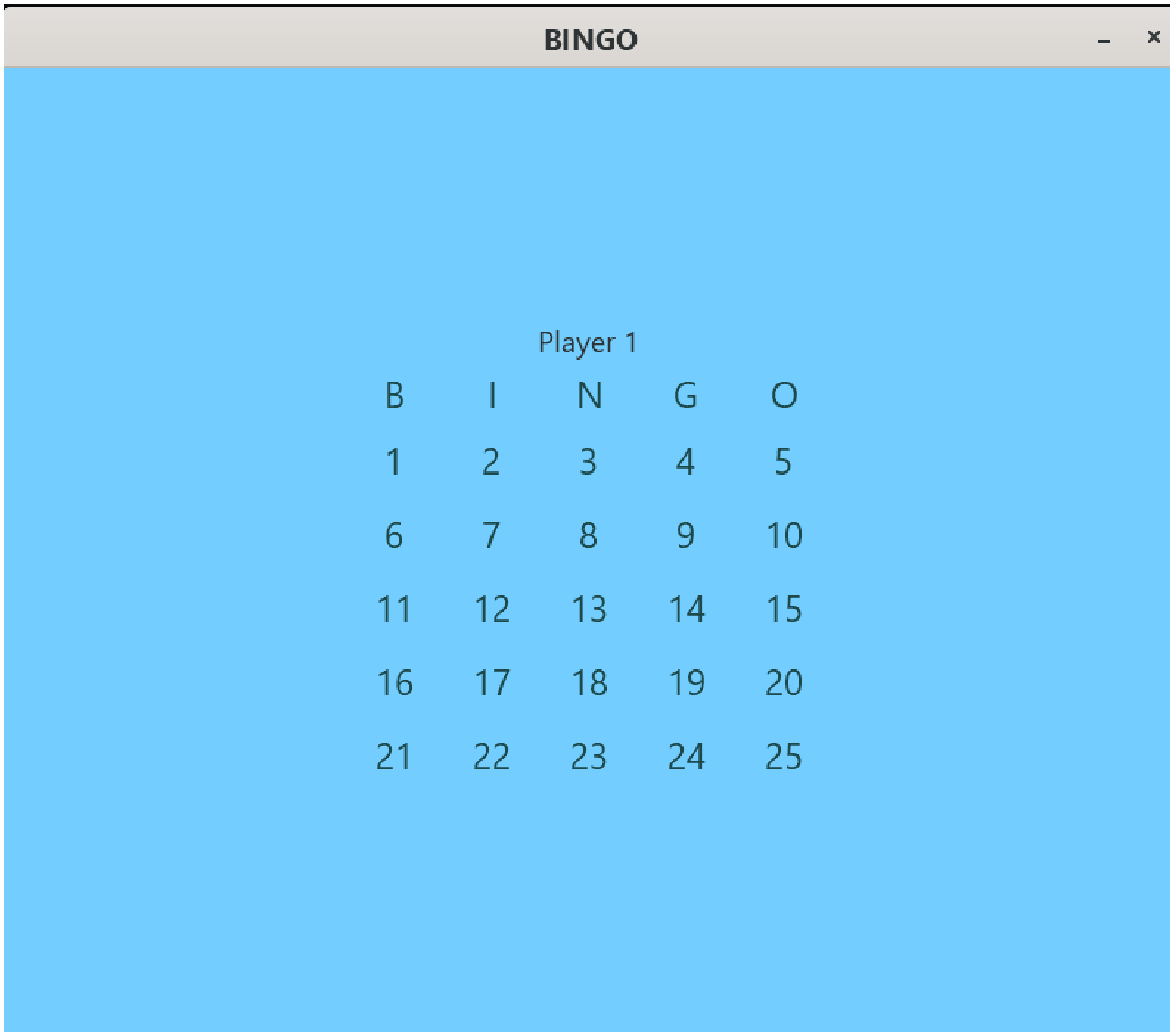
Output

First Screen




BINGO
Player 1
OK

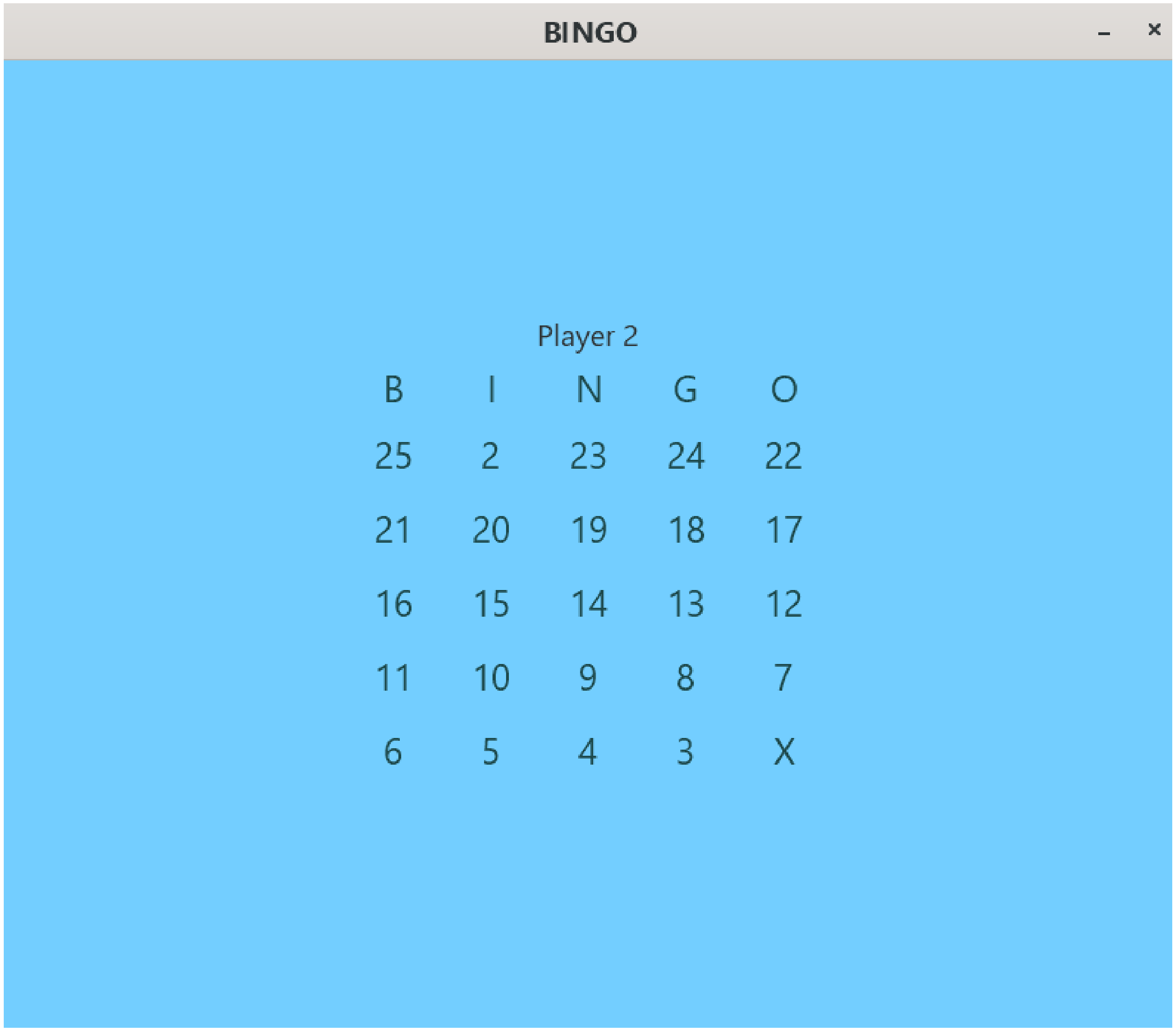

BINGO
Player 2
OK



BINGO

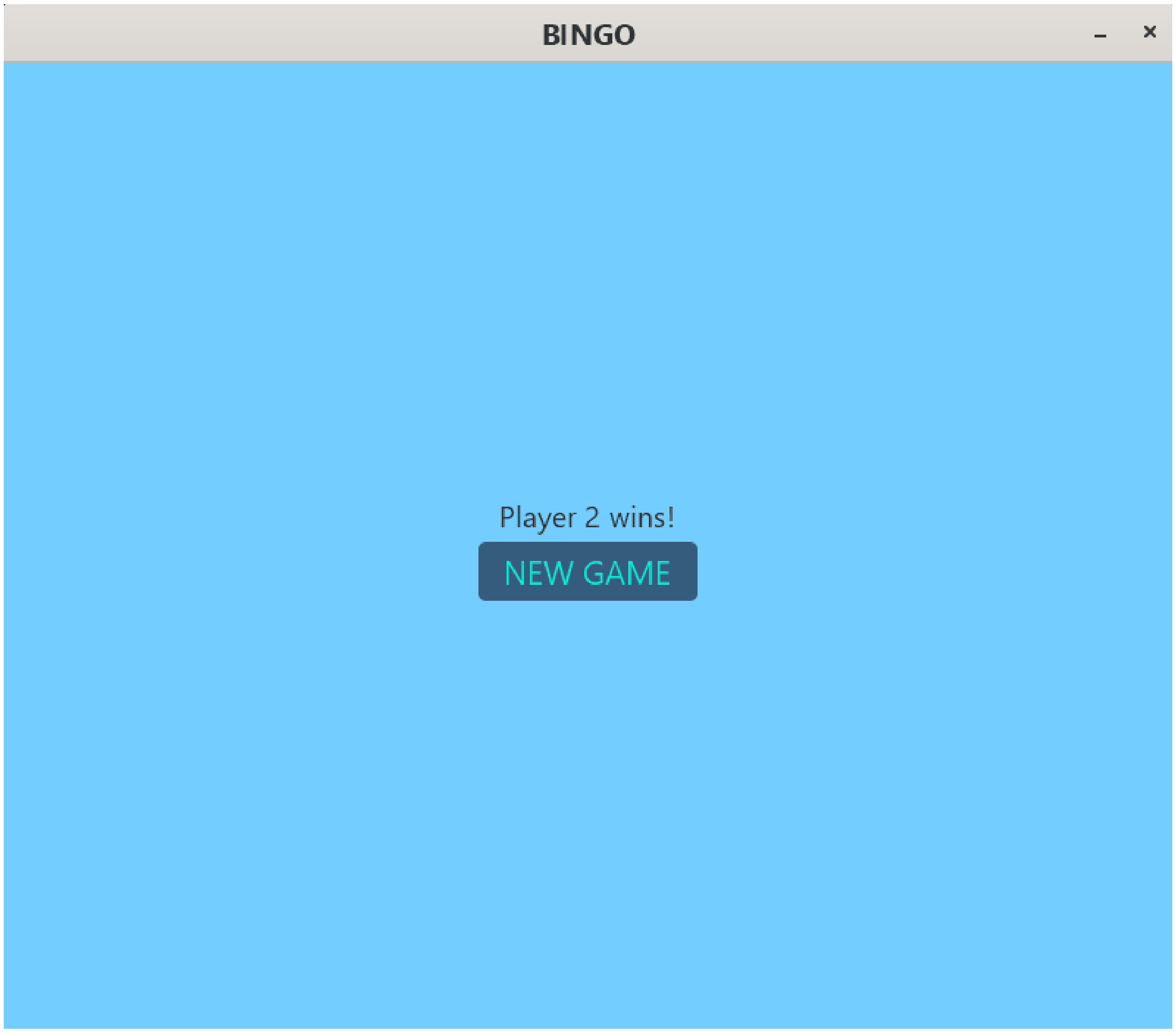
Player 1

B	I	N	G	O
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25



Player 2

B	I	N	G	O
25	2	23	24	22
21	20	19	18	17
16	15	14	13	12
11	10	9	8	7
6	5	4	3	X


BINGO
Player 2 wins!
NEW GAME

REFERENCES

gtk.org
Devdocs.io