Dots&Boxes v2.0

Description

Dots&Boxes is a game in which two players take turns to connect between dots by vertical or horizontal lines. The goal to complete a box to take a point. If a player completes the 4th side of a box, he takes an extra turn to play.

When all boxes are completed the winner is the player who has completed the most boxes (has more points).

The game has two levels, beginner level which is 2x2 grid and expert level which is 5x5 grid where the game becomes more complex.

The user chooses between two players mode and single player mode at which the player play against the computer.

When the game begins the main menu appears, which contains:

1. Start Game
2. User manual
3. Load Game
4. Top 10 players
5. Exit

To start a new game you'll be asked to choose the game level (beginner, expert), game mode (two players, single player) and to enter your name before starting.

Features of v2.0

- Graphical User Interface: the huge deference between v1.0 and v2.0 is the user interface used as in v2.0 the GUI used is way much better for the user to interact with the game.
- Save and load: the user is able to save up to three different games to be able to continue playing at any time.
- Undo and Redo option in v2.0 makes the user able to undo all moves till the beginning of the game and to redo all those moves again, so don't be afraid if you've done any mistakes while playing you can always undo it ;)
- One cool feature of v2.0 is the dynamic timer, which shows the time elapsed in hr:min:sec format.
- Colors make things a life, so we chose to make the game colorful and bright using a deferent color for each player, and colorful photos in the gameplay ☺.
- We also chose joyful music in the background of the software and a special sound effects for events like player move and completing a box.
- The winner is shown at the end of the game and highest 10 scores are recorded in the top ten which can always be viewed from the main menu, we are waiting to see your names there ;).

Design Overview

User Interface: in Dots&Boxes v2.0 we used GTK+2 library to create a Graphical User Interface, which makes it easier to understand and interact with the game. Adobe Photoshop was also used to create and edit photos to fit in our Design.

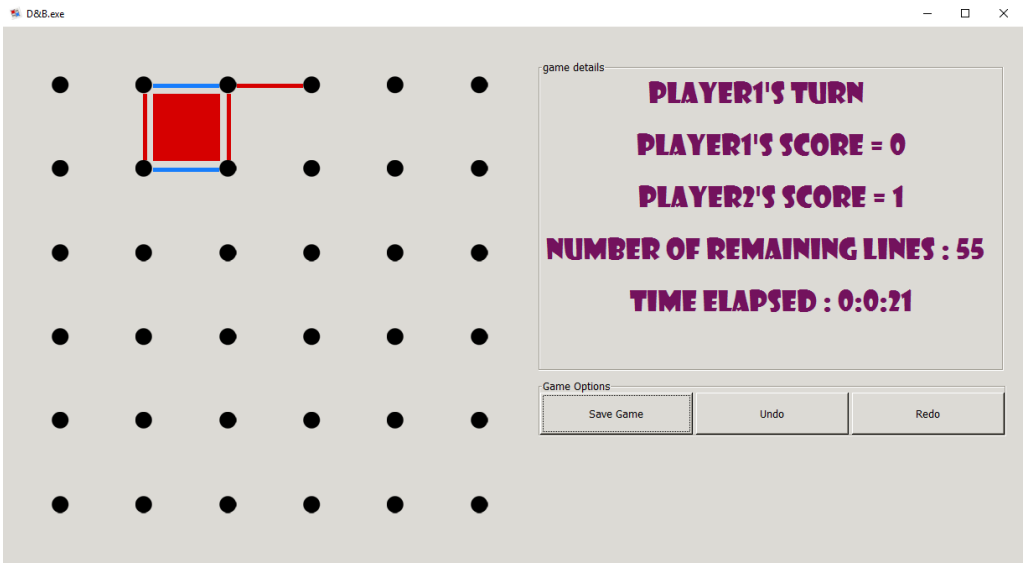


Figure 1

The Software consists of several windows, each window has a specific purpose:

Main-Menu window: TOP_LEVEL window which contains the before-game options and the game logo.

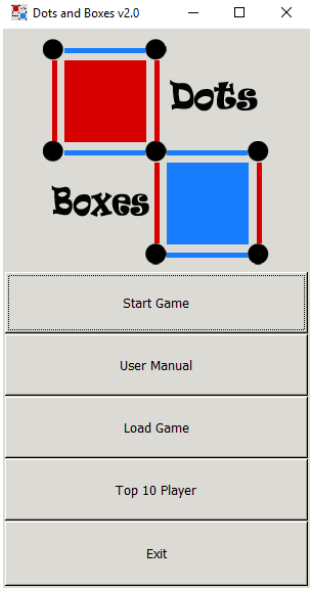


Figure 2

Main-Menu widgets:

The logo: to give the user an impression of what he's playing.

Start Game button: which shows a new vertical box containing game config (next section) and hides the rest of the menu.

User Manual button: opens a new POPUP window containing a description of the whole game and each button on the menu.

Load Game button: opens a new POPUP window containing up to three saved games that can be loaded.

Top 10 Players button: opens a new POPUP window showing players rank from 1 to 10.

Exit button: terminates the program.

Figure 3

Start Game Menu: it's a vertical box that shows when Start Game button is clicked, it contains the pregame option (single or multi, expert or beginner, .. etc.)

Start button: starts the game(in a new TOP_LEVEL window) with the settings set in Start Game window. (next section)

Game window: it's a TOP_LEVEL window that contains the game grid, the game details (labels printing the score of each player, player turn and time elapsed) and the game options (undo, redo and save).

The game grid is a 600x600 (px) image (300x300 for the beginner mode) with a matrix of event boxes each positioned on a vertical or a horizontal line (space between dots) and each event box contains a blank image which changes to a line image with the player color when the event box is clicked.(figure 1)

Design benefits:

- Graphical user interface in v2.0.
- Dividing the software to several windows makes it easy to upgrade the program at any time, it also limits the cases at which the software may crush due to the conflict in the same window, and makes the sources code more readable and understandable.
- Dynamic timer, which gives the accurate time elapsed from the beginning of the game.
- Using images for the lines instead of printing in a label gives the game a better view.
- The music in the background and the line change music makes the game more exciting.

Design risks:

- The rank (top 10) is saved in a txt file configured from the developer, if the file is deleted or edited in an improper way this will make the game crush at some cases. (under developing).

Assumptions and Hypotheses

At the first steps of developing the software we took some technical hypotheses, and some in game assumptions.

Technical hypotheses:

- Assumed to use win32 API library to build the GUI but in the end we used GTK+2 library.
- Assumed to use windows.h library to play sound but the functions only play wav format which will make the program large, so we used bass.h library which provided us with powerful sound functions that can play MP3 sounds.
- We assumed to make a dynamic timer using manual interruption methods or using multithreading but fortunately we found a gtk function to interrupt and call another function every specific time period in milliseconds (1000 milliseconds in our case).

In game assumptions:

- We assumed that the user may try to play the same move again so the program always check if the move is valid or not.
- We assumed that the user may enter a name bigger than 20 character so we set a limit for the user input to be under 20.

Data Structures:

Structs and DatatTypes used in the project:

```c
struct boxes{
    GtkWidget *eb1;
    GtkWidget *eb2;
    GtkWidget *h;
    GtkWidget *v;
    GtkWidget *b;
    int hl,vl;
    int n;
    int m;
    int semif;
    int f;
};

struct player{
    char *name;
    int score;
    int len;
};
```

The boxes structure

(eb1, eb1) type GtkWidget* -> event boxes.

(h, v) type GtkWidget* -> images (horizontal and vertical lines).

b type GtkWidget -> image (box image).

(hl, vl) type integers to indicate the line exists or not.

(n, m) type integers -> box index.

(semif, f) type integers to indicate if the box is closed or not.

The player structure

name type char* (string) -> player name.

score type integer -> player score.

```c
struct box_data{
    int hl,vl;
    int n;
    int m;
    int semif;
    int f;
};

struct save{
    struct box_data b[7][7];
    struct player p1;
    struct player p2;
    int n, turn;
    int s;
    int l1,l2;
    int r_dots;
    int available;
};
```

box_data structure

Contains all the integers that describe the box in **boxes** structure.

Save structure

b[7][7] array of structure **box_data** to save the grid in a file.

p1, p2 type **struct player** to save the players in a file

n, turn, s, l1, l2, r_dots, available type integers to describe the game state.

```c
struct move_details {
    int n;
    int m;
    int vl;
    int hl;
    int f;
    int p1score;
    int p2score;
    int turn;
    int semif;
    int r_dots;
};
```

move_details structure

Contains only integers to describe the move that the player made to be stored in the moves stack for undo and redo

Arrays Defined in the project:

```c
extern struct boxes box[7][7];
```

A 2D array of type struct boxes and size 7x7, to display and configure the grid

```c
extern struct move_details moves[100];
extern struct move_details mredo[100];
```

The Undo and Redo Stacks (arrays of type struct move_details and size 100

We also created a resources file (.res) and an icon for the .exe file of the game.

Description of important functions and Header files

The following contains a small description of each header file and the important functions defined in the file.

Main_menu.h

The header file used to show the main menu and start the software it contains all the main menu events and widgets.

Important functions:

- **main_menu_win():** initializes the main menu window and widgets on it.
- **start_menu():** initialize the options and widgets in the start game menu.
- **menu_signals():** sets signals for different events (button clicked, window destroy, etc)
- **init_top_ten():** initialize top ten window and it's widgets
- **open_top10():** loads saved ranking from file and printing it in the top ten window.

Game_play.h

the header file contains all the functions used during the game and runs the game events.

Important functions:

- **init_grid():** initializes the game window and the grid photos.
- **init_boxes():** initializes each box and event box, initializes each line photo and position on the grid.
- **init_eb():** sets the event of each event box to be called when the eb is clicked.
- **init_labels():** sets the game details labels and fix their position.
- **v_line_change(GtkWidget *w, GdkEvent *e, gpointer u):** an event to be called when a vertical eb is clicked, this event checks if the line valid to be clicked and changes the line for the player's color then check for score update and turn change it also calls save function to save the move in the undo stack.
- **h_line_change(GtkWidget *w, GdkEvent *e, gpointer u):** an event similar to v_line_change, but for the horizontal event boxes.
- **timeout_callback():** a function to be called every 1000 milliseconds which updates the timer.

Undo_redo.h

The header file contains all the functions used in the undo and redo events.

Important functions:

- **store(int n, int m):** stores the move done in the undo stack.
- **undo_v(), undo_h():** two functions to do the undo procedures in a vertical or a horizontal line.
- **redo_v(), redo_h():** two functions to do the redo procedures in a vertical or a horizontal line.

Save_game.h

The header contains the save functions and the initialization for the save window and save events.

save(const char *file): the function stores the game state, players' name, players' score, and time elapsed, in the file passed to it.

Load_game.h

The header contains functions used in the load game procedure.

Important functions:

- **struct save load_from_file(const char *file):** reads the data from the file passed to it and returns a save structure containing the game state, players' name, players' score and time elapsed.
- **set_game(const char* file):** loads data from the file and sets the grid with this data.

Computer.h

The header contains the functions used in the single player mode. All the algorithms are explained in the next section.

Important functions:

- **computer():** calculates the score of each available move on the gride by calling minimax() and chooses the max value.
- **minimax(int player, int depth, int alpha, int beta):** calculates the score of each available move using a recursive search algorithm combined with alpha and beta algorithm to shorten the search paths.
- **check_for_winner():** returns the score of the computer with a regret algorithm(-1 if the box closed by the player, 1 if the box closed by the computer).

Game_options.h

The header contains functions that handle the in game options and events(undo, redo, save).

Game_over.h

The header contains functions that handle the game over situation(the winner, game over image, game over options, .. etc).

Top_ten.h

The header contains only one function that ranks the players between the top 10 saved players and saves them to the file.

User_manual.h

This is only a header without a source file (.c) it only contains a description of the game defined to be used in the user manual POPUP window.

Sound_threads.h

The header contains all the music related functions and threads

Line_change_threads() the function creates 3 threads running in the background of the program for the background music, line change sound effect and box closing sound effect.

Important algorithms in the project

Computer AI algorithm (Minimax, AlphaBeta)

```
Computer AI algorithm (Minmax, AlphaBeta)

function computer():
    set move(integer array of size 3) = {-1,-1,-1} //{row index, column index, vertical or horizontal line(1 or 2)}
    set score integer = -INFINITY //-100
    for each available move on the grid:
        set integer s = minimax(do_the_move_return_next_player_turn,4,-100,100) //minimax(player_turn, depth, alpha, beta)

        undo_the_move

        if s>score:
            score =s;
            move = {move_row, move_column, vertical_or_horizontal}


    apply move
    return
```

```
function minimax(player turn, depth, alpha, beta):
    if depth = 0 or remaining moves = 0 :
        current score = calculate_score()
        return current score

    set move(integer array of size 3) = {-1,-1,-1}
    set score = 100
    if player turn = real player:
        for each available move on the grid:
            set integer s = minimax(do_the_move_return_next_player_turn, depth-1, alpha, beta)
            undo the move
            if s<score :
                score=s
                move = {move_row, move_column, vertical_or_horizontal}
            if beta>score :
                beta = score
            if beta<=alpha :
                return score

    else if player turn = computer :
        set score = -100
        for each available move on the grid:
            set integer s = minimax(do_the_move_return_next_player_turn, depth-1, alpha, beta)
            undo_the_move
            if s>score :
                score = score
                move = {move_row, move_column, vertical_or_horizontal}
            if alpha < score :
                alpha = score
            if beta <= alpha :
                return score

    return score
```

```
function calculate_score():
    set computer score =0
    for each box on the grig:
        if box is closed:
            if closed by computer:
                computer score = computer score + 1
            else if closed by real player:
                computer score = computer score - 1

    return computer score
```

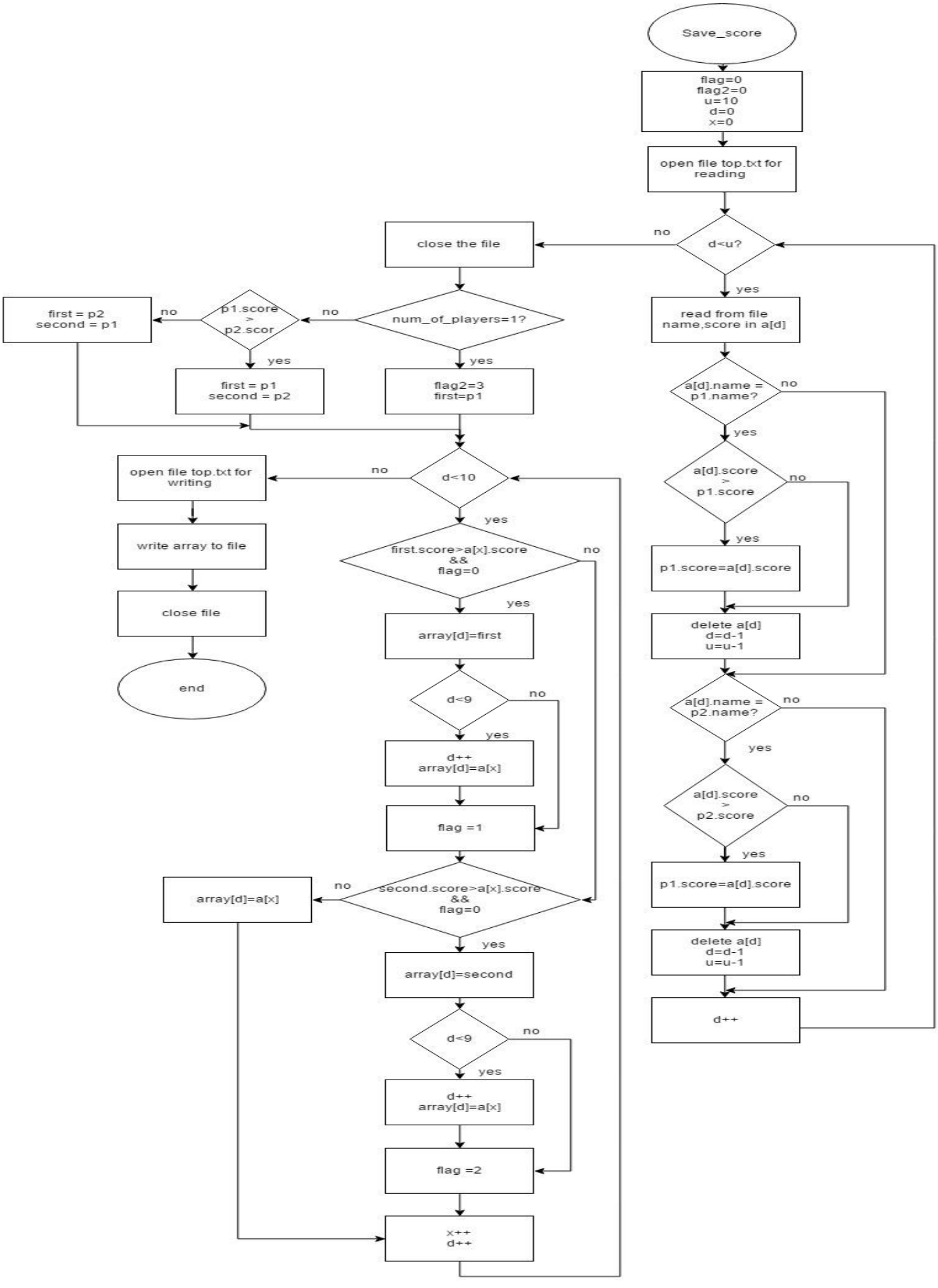
Undo pseudo code

```
function undo_game():
    if (remaining dots=0)
        return
    if (the stored move is vertical)
        call undo_v()
    if (the stored move is horizontal)
        call undo_h
    update_labels()//labels as (player turn, player1 score,player2 score,remaining dots)
    return
```

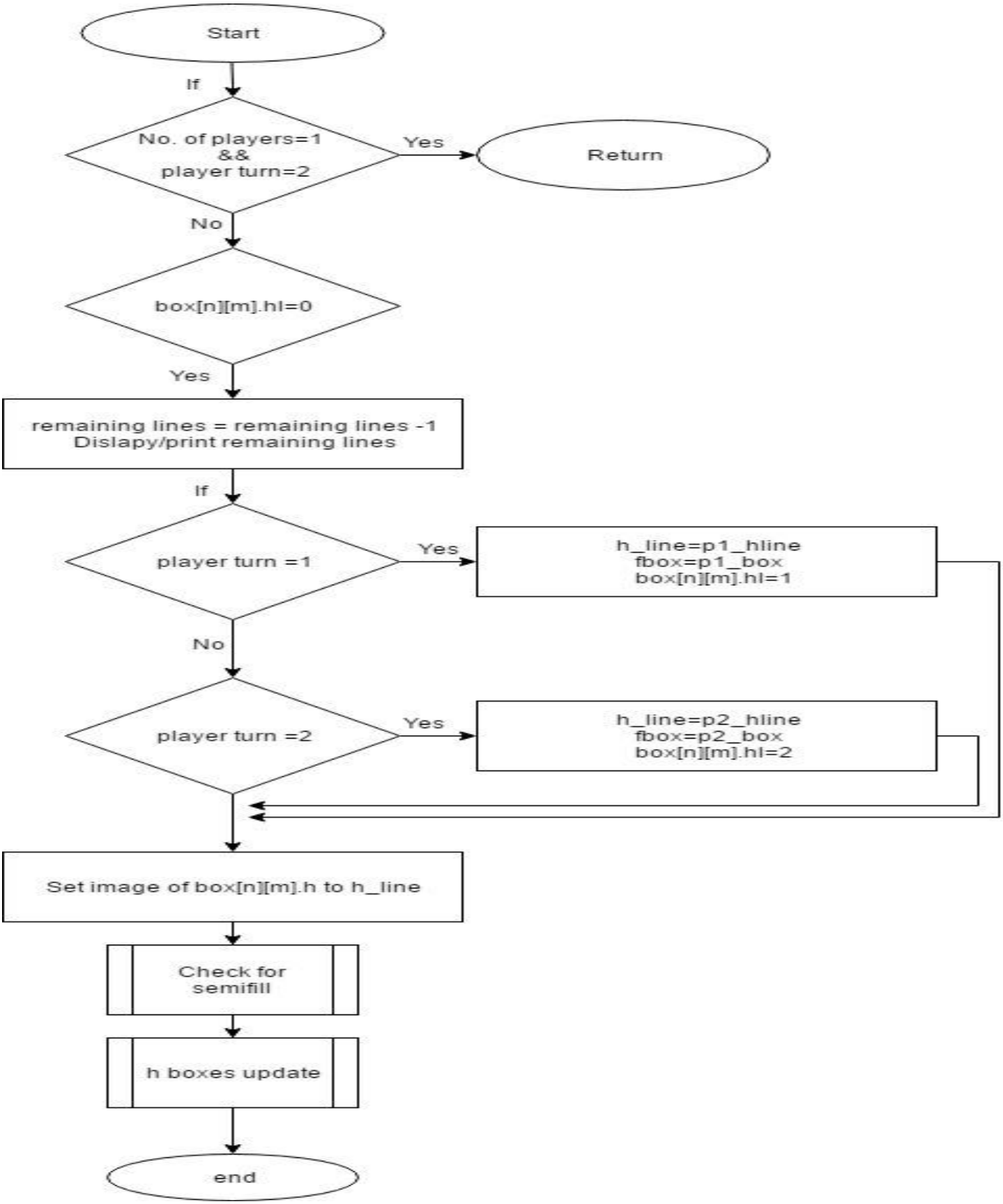
```
function store (no. of row,no. of coloumn):
    i=i+1//increase index of store array
    store move details in array of structure
    //(remaining dots,no.row,no.coloumn,semif,player turn,fill,player1 score, player2 score)
    if (move is vertical) :
        store value of box.vl
    else if (move is horizontal):
        store value of box.hl

    return

function undo_v():
    set remaining dots to its value in the previous move
    store the store array in redo array
    k=k+1//k is the index of redo array
    box[moves[i].n][moves[i].m].vl=0
    set image of vertical move to blanck
    if (current box is semifill)//stored semif value
        box[moves[i].n][moves[i].m].semif=0
    if (box[moves[i].n][moves[i].m] is fill)
        box[moves[i].n][moves[i].m].f=0
        set image of box to blanck
    if (box[moves[i].n][moves[i].m-1] if fill)
        box[moves[i].n][moves[i].m-1].f=0
        set image of box to blanck
    set player1 score to its value in the previous move
    set player2 score to its value in the previous move
    set player turn to its value in the previous move
    Set move_zero //delete the current move from the store array
    i=1-1
```

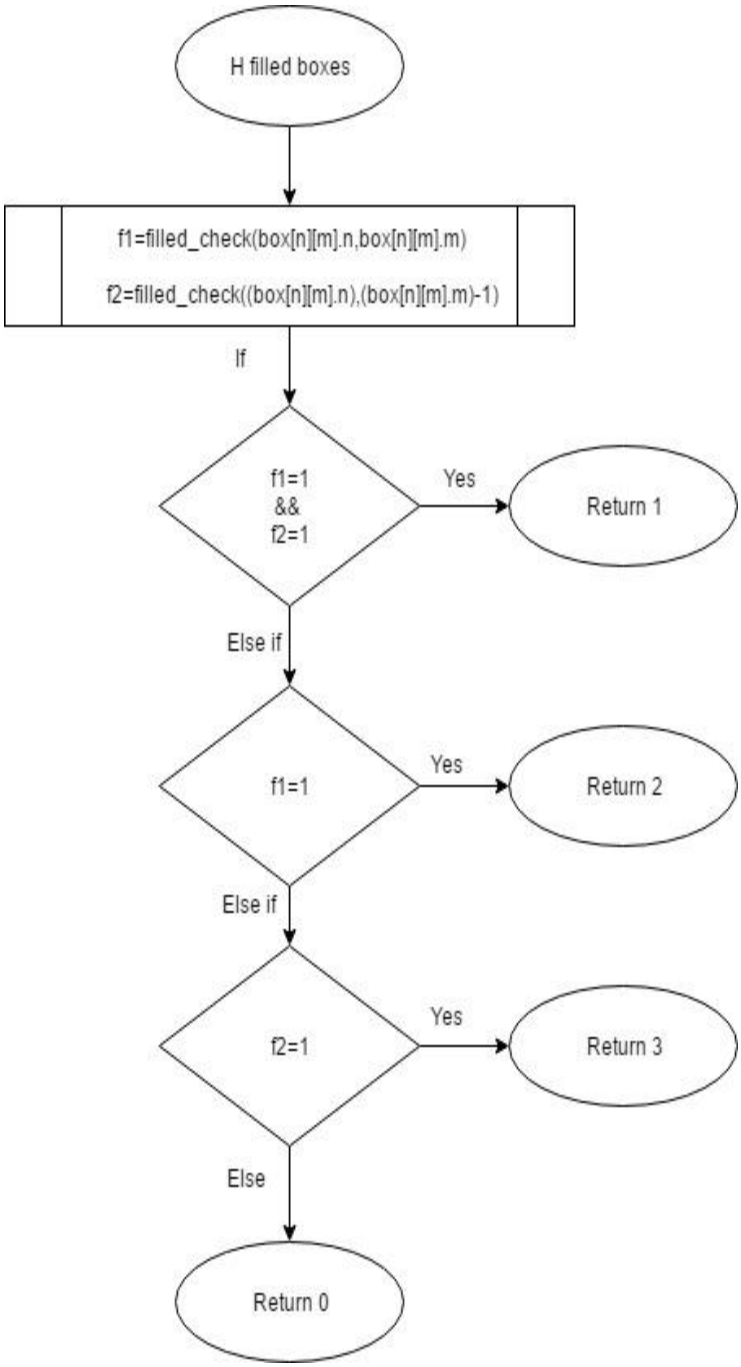
```
function undo_h():
    set remaining dots to its value in the previous move
    store the store array in redo array
    k=k+1//k is the index of redo array
    box[moves[i].n][moves[i].m].hl=0
    set image of horizontal move to blanck
    if (current box is semifill)//stored semif value
        box[moves[i].n][moves[i].m].semif=0
    if (box[moves[i].n][moves[i].m] is fill)
        box[moves[i].n][moves[i].m].f=0
        set image of box to blanck
    if (box[moves[i].n][moves[i].m-1] if fill)
        box[moves[i].n][moves[i].m-1].f=0
        set image of box to blanck
    set player1 score to its value in the previous move
    set player2 score to its value in the previous move
    set player turn to its value in the previous move
    Set move_zero //delete the current move from the store array
    i=i-1
    return
```

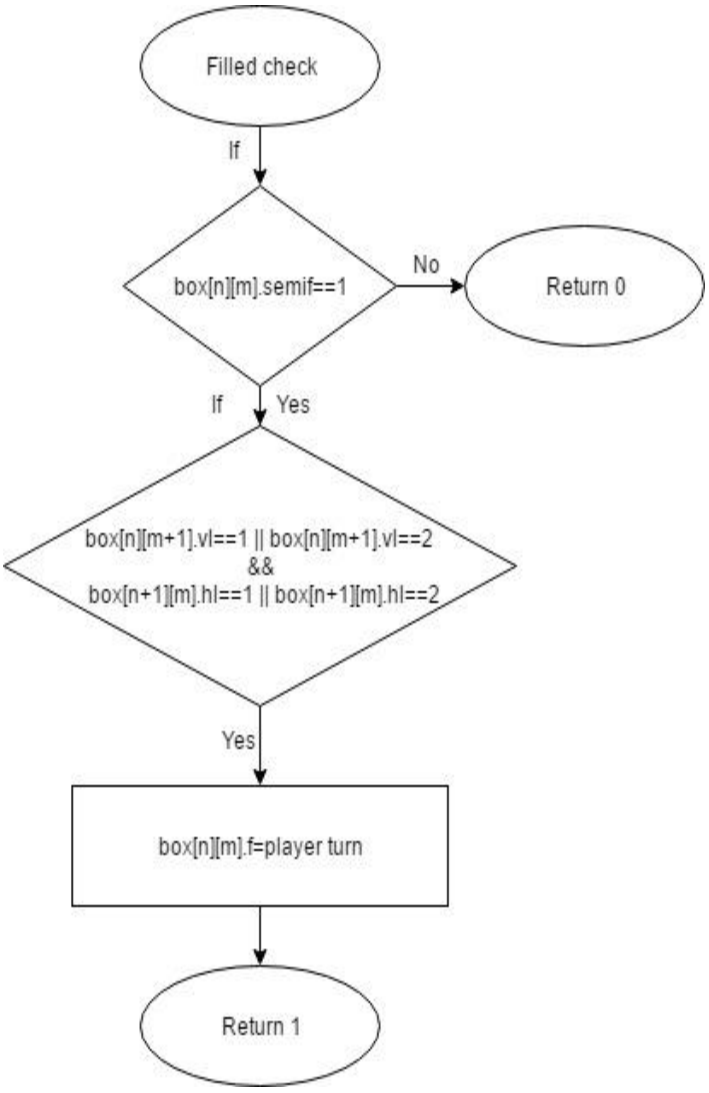
Ranking Flowchart

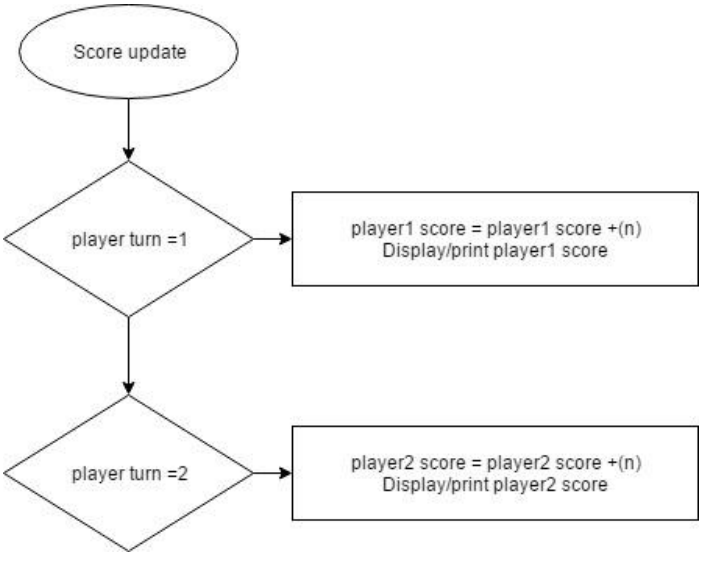


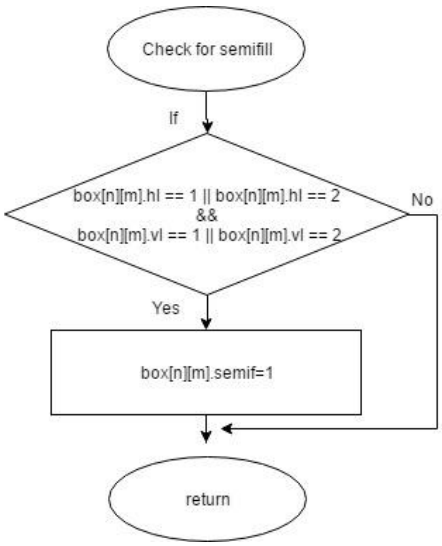
What happens when a horizontal event box is clicked? (Flowchart) and the same happens with a vertical event box.

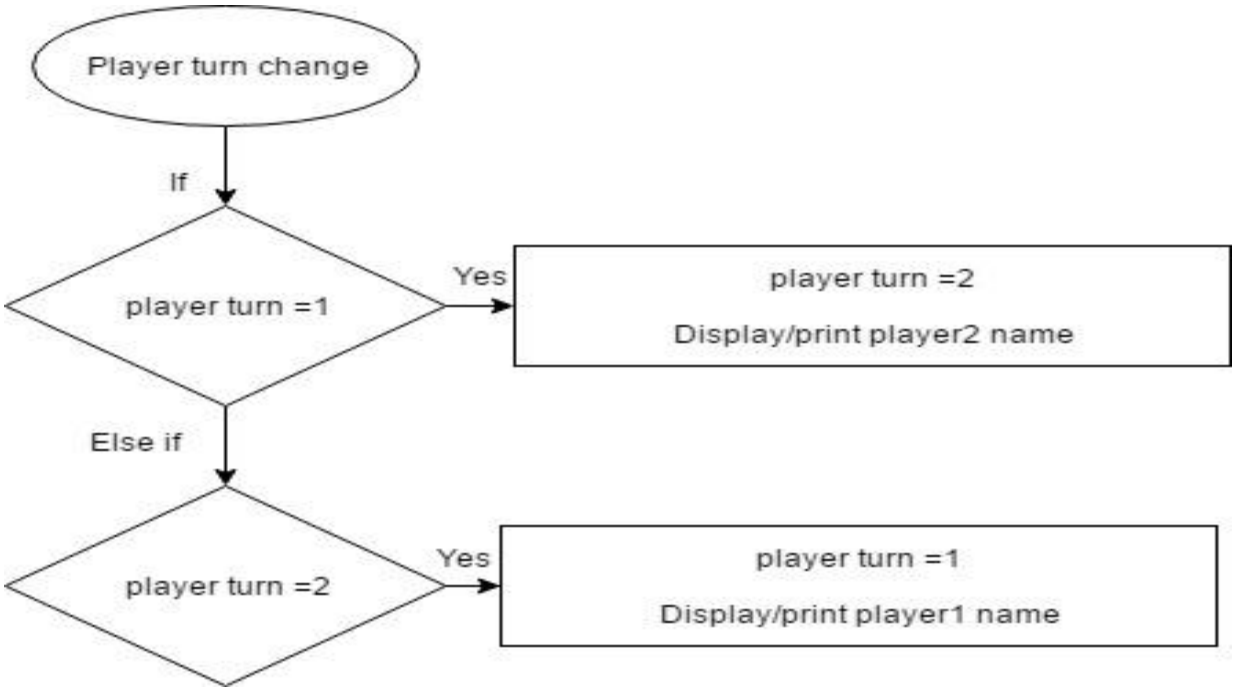



H filled boxes
f1=filled_check(box[n][m].n,box[n][m].m)
f2=filled_check((box[n][m].n),(box[n][m].m)-1)
If
f1=1
&&
f2=1
Yes
Return 1
Else if
f1=1
Yes
Return 2
Else if
f2=1
Yes
Return 3
Else
Return 0


Filled check
If
box[n][m].semif==1
No
Return 0
If
Yes
box[n][m+1].vl==1 || box[n][m+1].vl==2
&&
box[n+1][m].hl==1 || box[n+1][m].hl==2
Yes
box[n][m].f=player turn
Return 1


Score update
player turn =1
player1 score = player1 score +(n)
Display/print player1 score
player turn =2
player2 score = player2 score +(n)
Display/print player2 score


Check for semifill
If
box[n][m].hl == 1 || box[n][m].hl == 2
&&
box[n][m].vl == 1 || box[n][m].vl == 2
No
Yes
box[n][m].semif=1
return



Sample runs of the application

The player can choose the game level beginner (fig. 4) or expert (fig. 5)

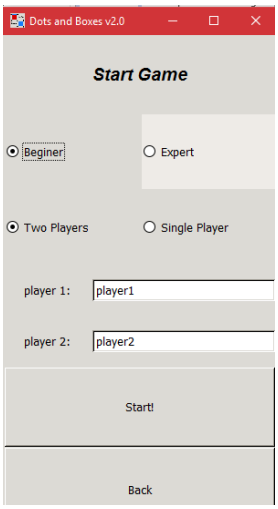


Figure 5 game options

Figure 4 Beginner level

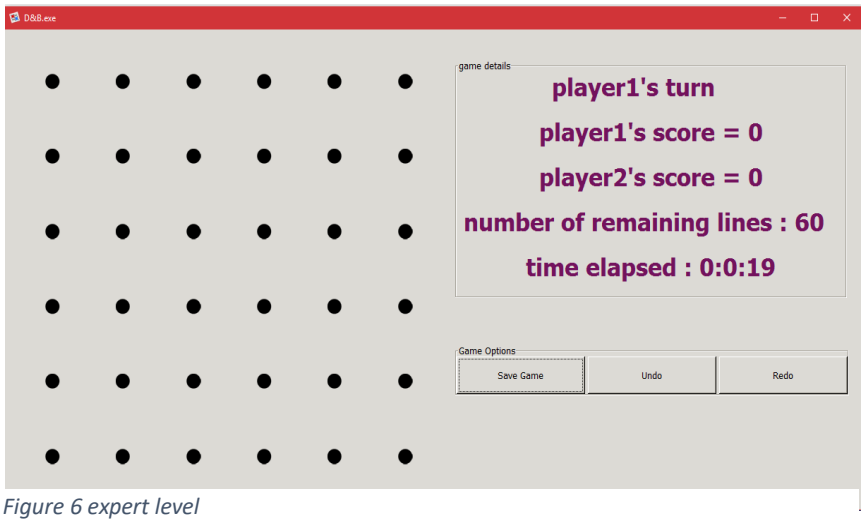


Figure 6 expert level

Players takes turns to connect dots and complete boxes:

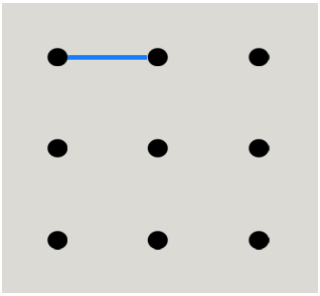


Figure 9

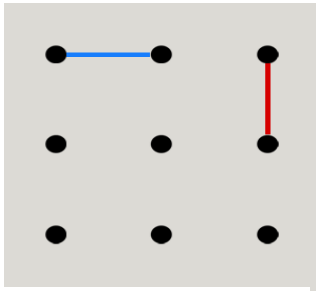


Figure 8

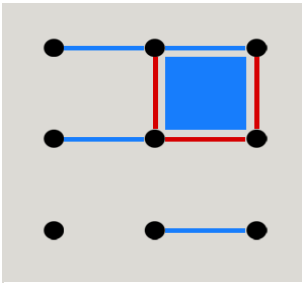
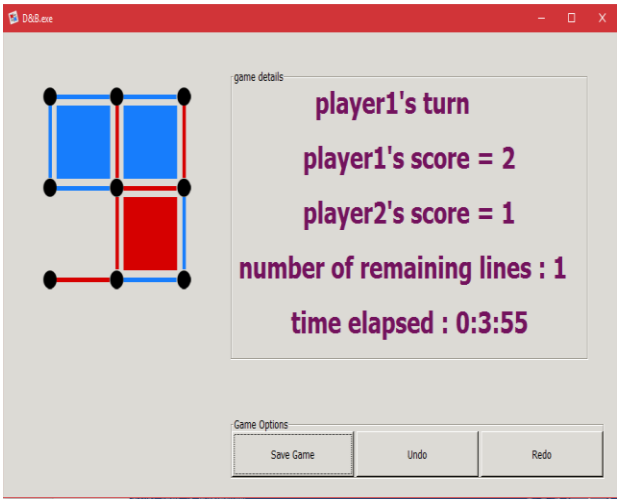
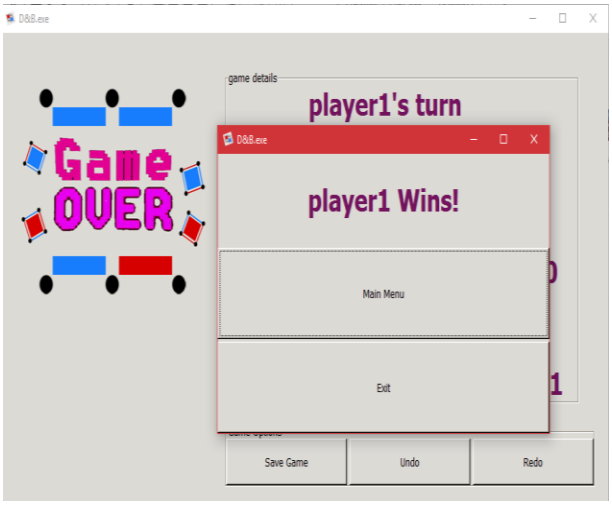


Figure 7 player1 takes extra turn

The player who complete more boxes wins





Undo and redo

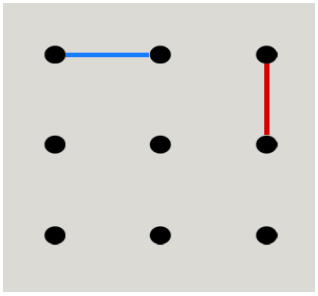


Figure 10

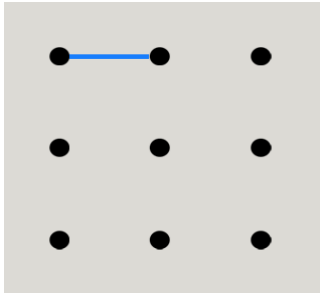


Figure 11 move in fig 10 undone

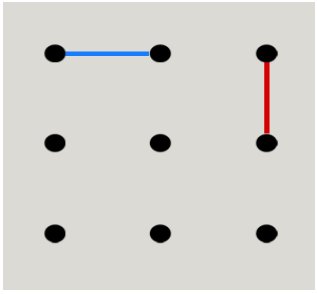


Figure 12 undo in fig 11 redone

Single Player mode

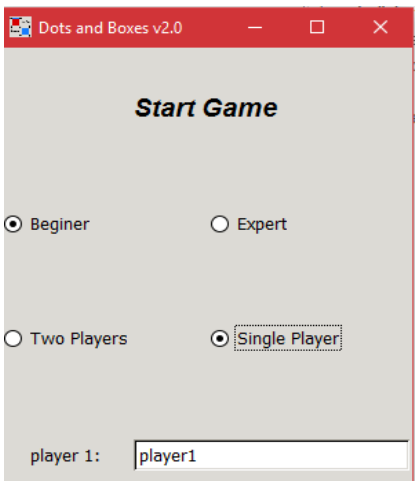


Figure 10

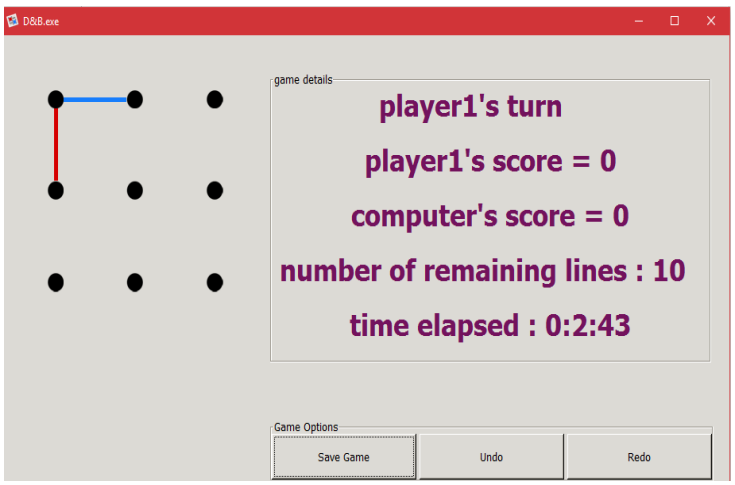


Figure 14

Save and Load Game

Figure 15

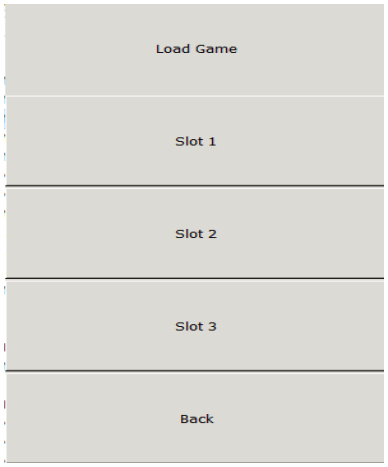
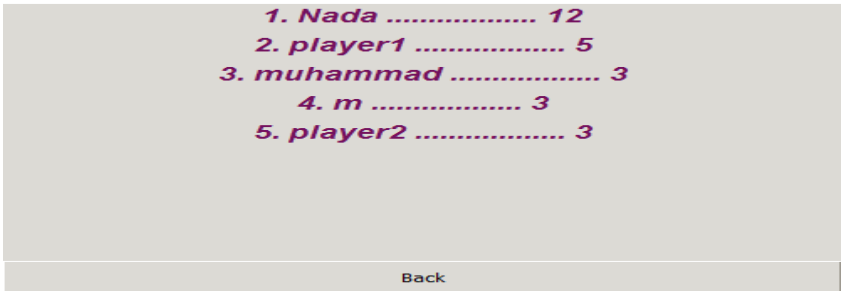


Figure 16

Top Ten Players



References

1. For the GUI we used GTK+2 library from: https://www.gtk.org/
2. For the multithreading we used pthreads library: http://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html
3. For the music handling we used bass.h library from: http://www.un4seen.com/
4. To understand the minimax algorithm on a simple tic tac teo game: http://www.flyingmachinestudios.com/programming/minimax/
5. Minimax algorithm in game theory: http://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/
6. Minimax and alpha beta as a search algorithm: https://www.cs.cornell.edu/courses/cs312/2002sp/lectures/rec21.htm
7. The music used from a game called sims 2.
8. We must also refer to the great problem solver stackoverflow: https://stackoverflow.com/

Any other used file or image is our own creation, All the source files are included with the project and are available on our repository on bitbucket and soon on github.