Semanur Aydınlık- 2018555005

Özden Kırıcı- 2018555041

Amine Ceyda Tandoğan- 2018555061

**GAME RULES**

**1. Game description**

- The game Black & White is a game that is played with human vs computer players.

- First plays with the camp of black pawns, the other with the camp of white pawns.

- The pawns are all double-sided, one white and one black.

- The goal of the game is to count, at the end of the game, more pawns of its color than the opponent

by returning them according to precise rules.

**2. Placing of pawns and capturing**

- In his turn of play, the player must place a pawn of his color on an empty box

adjacent to an opposing pawn.

- He must also, by placing his pawn, enclose one or several opposing pawns between the pawn he places and

a pawn of his color, already placed on the game board.

- Then, the enclosed opposing pawns are returned to have its color.

- The pawns are neither removed from the game board nor moved from one box to another.

- The enclosure is done according to the lines, the columns and the diagonals.

- If a player cannot enclose at least one opposing pawn, he must skip his turn.

- If a player can place a pawn in at least one box, he has to do so.

- At the beginning of the game, two black pawns and two white pawns are already placed

**3. Game Over**

- If a player has no pawns on the game board, he cannot capture anymore opposing pawn,

so he has lost the game.

- If none of the players can play, then the game ends with a draw.

- If all the board is filled with pawns, a count of each players' pawns is made

The player who obtains the highest total won the game.

There may be a draw.

**HOW DID WE MAKE “DR. REVERSI”**

First of all, we added the Allegro library to visual studio, activated image addon to create bacground bitmap, and activated primitive addon to draw black&white pawns. In addition writing to the display we activated font addon and ttf, defined board and parcel size, board width and board height. In main function, we constituted a char array to represent the game board. We needed mouse and keyboard to play. So installed mouse and keyboard function, added images as bitmap and font.

If the player want to start again the game before, we initialize integer value as kimde and was equeled to 1, then used in while loop like while(1).In order to have no paw in the board we equaled to ‘n’, and then defined 4 array as 2 black ( ‘s’ ) and 2 white ( ‘ b’ ) pawns to center of board, added menu as bitmap , in order to appear in display. We constituted if loop, because of keyboard motions. When the game start, menu will come to display, and if player press to space(to start) , game will be start. Then board will appear. We drawed 4 center of board pawns with al\_draw\_filled\_circle() function. If player dont want to start game again, we constitute gotoagain value, and equal to 0.

We defined int counter\_black in order to learn who win or lose, and equaled to 0. Then black or White is increasing in for loop, if player or computer have a pawn. To correct motion, we added if, else if loops.

First, if we created our cycle to indicate who won and lost after the game was over, and we sent the ctrl int value as a dwelling when the ctrl value was not available on the board. We used the al\_draw\_text function in the "GAIN" and "Game OVER" game (3000) to appear for 3 seconds and then disappear; we made the text color we wrote and made it black. To make it come back to the next stage, gotoagain = 1 and break;

In cases where the ctrl value is 0, that is, we have defined the integer in order to determine the order of movement of the computer and the user. We wanted the user to be able to move in situations where the game cycle is inside, otherwise the computer would move.

In the if loop where else (if) who is, that is, the user's movements, we first checked that the player has a valid place to move, we did this with the player\_pass function. If there is no place to make a valid move, we wanted the screen to display a “pass” and we used the al\_draw\_text () function in the same way to make it disappear for 3 seconds. We wrote the line. In cases where it has a valid move, ALLEGRO\_event ev; we defined and used the function al\_wait\_for\_event (queue, & home). We used the self-defined make\_move function to draw a black stone when the user clicks the mouse once and turn the places between the two black stones to white stone. We sent. Then we put this function in the if loop to prevent the user from drawing stones in invalid places, and when the current move was made, who has the =! We used the line.

When the turn went to the computer, we first waited 1 second for the user to see the move and run the computer\_move function we wrote. In this function, we sent the character as 'b' in order to indicate that the board and the move are white as a parameter. After making the current move on the computer, we used the who = = who line so that the sequence passes to the user in the same way.This way, the game will close smoothly after all the cycles are overal\_destroy\_bitmap (background);al\_destroy\_bitmap (menu);al\_destroy\_display (display);We finished the game by adding int commands and finished int main.

The functions we use in int main and how they work;

**1-int make\_move (char board [BOARD\_SIZE] [BOARD\_SIZE], int row, int col, char player);**

The purpose of writing this function is to change the color of the opposing stones between the stones after the move is made to the current location. In order to run this function, we asked the final status of the board as a parameter, the position of the user, as the value of the row and milk, and the character value of the move based on who moves. In order to determine the direction of movement within itself, we use the retVal int value to send int int as a secondary line value to use int coldelta values ​​when searching with rows and columns. we defined the int t\_row and int t\_col values ​​we assigned, and finally the char opponent value to determine the opposite side.Then we used rowdelta and coldelta to find the direction of movement, using nested for loops between 1 and +1. In this for loop, we created an if loop that controls them so that the direction of motion is not constant and does not go off the board. ; We used the command. In cases where the opposite side finds the stone in the direction of movement, we recorded that direction to the values ​​of xm and ym, and after finding the opposite stone, we break it to move in that direction and change the stones until we find our move; We have created a for loop that allows it to run as much as we say. In this cycle, we increased the xm and ym values ​​in the direction direction, and we added the if loops so that the situations don't get confused. Again, when we meet with an empty square in the direction of the direction in which we are moving, another invalid move, we have drawn the move to change the remaining pieces according to the character of the move situation in order to exit the loop when we made a valid move in the last if cycle. We used the while loop to reduce it one by one until we return to our move again and we used the function al\_draw\_filled\_circle (); to record it in the wooden string and display it on the display, depending on whether the move was black or white. RetVal = EXIT\_SUCCESS because the current move was made; add the line and break; We exit the loop with and return retVal so that the most recent function is completed and returns to int maine; We wrote the command and finished this function.

**2-int player\_pass (char board [BOARD\_SIZE] [BOARD\_SIZE]);**

We created this function to determine whether the user has a valid move. As a parameter, we only used the char array that indicates the status of the current board. First, we defined the char temp\_board [BOARD\_SIZE] [BOARD\_SIZE] that we will use to transfer the char board to avoid confusion while doing research in the function, then we defined the int retVal value that we will return for control, and finally we defined int values to use the row and column values in the for loop. If (board [row] [col]! = 'N') continue to continue searching until we find the empty box in our first loop; We then used the line to send the board to the make\_move2 function in the for loop and pass it to the temp\_board array to investigate whether it is valid move and send this board to the make\_move2 function as 's' as a character for the current move control as the character's row and column value and the user's move black. In order to determine whether there is a valid move and send it to int main, we put this function in the if loop and determined the states of the retval. Finally, we made return retVal and finished the function.

**3-int make\_move2 (char board [BOARD\_SIZE] [BOARD\_SIZE], int row, int col, char player);**

We created this function to check computer movements and whether there are valid moves. Basically, make\_move works like the same function, the only difference is that after using this representation board and lines, it only moves the move stones to this board, and does not make stone plotting movements on the display just like it does in the make\_move function.

**4-void computer\_move (char board [BOARD\_SIZE] [BOARD\_SIZE], char player);**

We created this function for the computer to move when it is time for movement. With this function, the computer finds its best move and ends after drawing the white stone there and turning the black stones in between. We wanted the current position of the board as the parameter for the function to work and the white of its stone color as the character. Then we defined the int and char values for the row and column values and for use in the function. The function first explores the elements of the char board array with the nested for loop. In this for loop, we first searched for the empty square and found the empty square, then copied the board to the temp\_board array to use in the make\_move2 function, then temp\_board 'with the current row column values to check for valid moves. We sent it to make\_move2 inside the if loop. If there is no valid move, we added the continue line to search for other boxes. Then, to compare the new score with the current score, we have set the new\_score value that we define as int to the score value returned by the best\_move function. This function is the best among all the moves the computer can do. We found the best row and column position by comparing the if loop with the current score and the best score. We have created pass states with an if loop in case there are situations where the computer cannot move in any way. If there is no pass condition, we used the make\_move function to wait for 3 seconds after turning the stone to white on the display after using the best\_row, best\_col using al\_draw\_filled\_circle () function to draw a white stone on the display, which is the movement of the computer. And we have completed the computer\_move function.

**5-int best\_move (char board [BOARD\_SIZE] [BOARD\_SIZE], char moves [BOARD\_SIZE] [BOARD\_SIZE], char player);**

We used this function to find the best score value in the movements performed by the computer. As a parameter, we wanted the current position of the board, the sequence in which we record the movements and the movement of the computer as characters. Then we returned the score value we calculated after the operations were completed. We also defined char opponent to determine the opposite side, char array, a new board that we will use in comparisons, new\_board and int and score and new\_score. First, we used nested for loops to check the values of the board we sent, and in these for loops, we first checked whether the box was empty and whether the current position of the char array in which we recorded the motion values was empty. Then we saved the new board to new\_board and used make\_move2 function to use it in make\_move2. After this function worked, we synchronized the value of new\_score int to get\_score. In this function, we created to calculate the current score of the board. Then, to find the best score, we compared these scores with the if loop and recorded the best score in the score int value and returned this value.

**6-int get\_score (char board [BOARD\_SIZE] [BOARD\_SIZE], char player);**

In this function, we asked the current move character to determine the current board as a parameter and which side scored. Then, with the nested for loop, we checked the elements of the current board, calculated the score and returned the score.

**OUR MİSTAKES AND DIFFICULTIES**

\*\* int valid\_moves (char board [BOARD\_SIZE] [BOARD\_SIZE], char moves [BOARD\_SIZE] [BOARD\_SIZE], char player); With this function, we wanted to use it to check if there is a valid move on the board, and we thought that we could use this function to see if the last game was over, but the value we returned was sometimes returned 0 in the current move situation, and this caused the game to work incorrectly so we could not use it actively at int mai () and instead we produced alternative solutions.

\*\* int counter (char board [BOARD\_SIZE] [BOARD\_SIZE]);We wanted to use this function to calculate and print the score when the game is over. First, we wanted to use the score in black: and white: using the al\_draw\_textf () function, but we wanted to use the al\_draw\_textf () error in the line we used it, then we wanted to define the score values globally and return them to int main in the function. But we gave up running this function since we received an error again in the main line in the main\_function of the al\_draw\_textf () which we added on the 138th line and could not find out what it was due to.

\*The problem of mouse movements: To solve it, we installed install\_mouse () and used ALLEGRO\_MOUSE\_AXES and ALLEGRO\_EVENT\_DISPLAY\_SOURCE functions. We wanted the mouse to place a circle where we clicked, so we used the ALLEGRO\_MOUSE\_BUTTON\_DOWN function.

\* We decided to make a menu. We set the cover photo as bitmap. When we entered the press to start coordinates, we would set the mouse to start the game. Then we gave up this and installed install\_keyboard () and decided to start the game by pressing the space key on the keyboard using ALLEGRO\_KEY\_SPACE.

\* When the game is over, we will prepare 3 similar menu covers and make sure that the cover that says game over, deluce or win is whatever the situation is equal to. Instead, we found it more logical to print as fonts.

**TASK DISTRIBUTION**

**\***We went step by step. Three of us tried to solve the stages. Person who solved and the added this to our code.