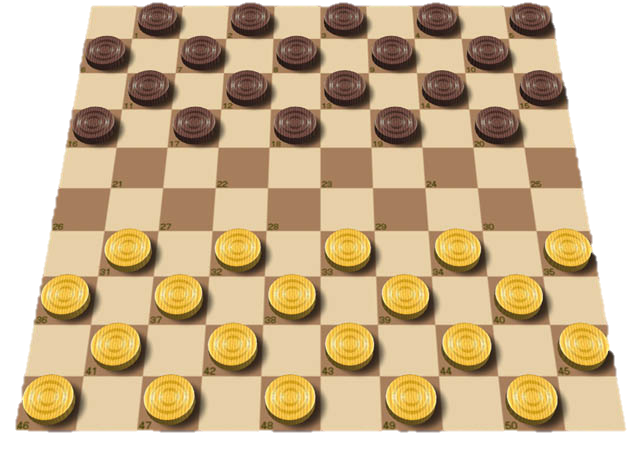
**CHECKERS**

The game



**Project prepared by:**

**Name:** Viren Mohindra

**Class:** XII-C

**Session:** 2014-15

**School:** Vasant Valley School, New Delhi

**Table of Contents**

1. Acknowledgement
2. Certificate
3. History
4. General Rules
5. My Program - CHECKERS
6. Code
7. Output Screens

**Acknowledgement**

I would like to thank my Computer Science teacher, Mrs. Sonya Bahri, who helped me in analyzing errors in my code while also teaching me how to program in C++. She made concepts clear, enhanced my knowledge and helped me realise the importance of good, clear code.

I am also thankful to the Central Board of Secondary Education (CBSE) of India for giving me the opportunity of making this project, and giving us suitable guidelines for the project.

**Name:** Viren Mohindra

**Class:** XII-C

**Session:** 2014-15

**School:** Vasant Valley School, New Delhi

**Certificate of Authenticity**

This is to certify that Viren Mohindra, of class XII-C, has created a project called “Checkers” and the project is original and genuine. This project may be considered as the part of the practical exam of CBSE 2015 conducted by CBSE.

Signature

Date

**Mrs. Sonya Bahri**

**HOD, Computer Science**

**Vasant Valley School**

**History**

**Draughts** or **checkers**is a group of strategy board games for two players which involve diagonal moves of uniform game pieces and mandatory captures by jumping over opponent pieces. The name derives from the verb *to draw* or to move.

English draughts, also called *American checkers*, are played on an 8×8 checkerboard. I used this instead of the 10x10 and other variants.

**General Rules**

* Checkers is played by two opponents, on opposite sides of the gameboard.
* One player has the dark pieces; the other has the light pieces.
* Players alternate turns. A player may not move an opponent's piece.
* The player with the light pieces moves first unless stated otherwise.
* A move consists of moving a piece diagonally to an adjacent unoccupied square. If the adjacent square contains an opponent's piece, and the square immediately beyond it is vacant, the piece may be captured (and removed from the game) by jumping over it.
* Only the dark squares of the checkered board are used. A piece may move only diagonally into an unoccupied square.
* In almost all variants, the player without pieces remaining, or who cannot move due to being blocked, loses the game.

All of these rules/ constraints have been implemented in the program in the **check\_move()** function.

The two types of pieces in checkers:

1. **Men**

* Uncrowned pieces (*men*) move one step diagonally forward, and capture an opponent's piece by moving two consecutive steps in the same line, jumping over the piece on the first step.
* Multiple opposing pieces may be captured in a single turn provided this is done by successive jumps made by a single piece; the jumps do not need to be in the same line but may "zigzag" (change diagonal direction).

1. **Kings**

* When a man reaches the *crownhead* or *kings row* (the farthest row forward), it becomes a *king*, and is marked by placing an additional piece on top of the first man, and acquires additional powers including the ability to move backwards (and capture backwards, in variants in which they cannot already do so).
* Since captured pieces remain on the board until the turn is complete, it is possible to reach a position in a multi-capture move where the flying king is blocked from capturing further by a piece already captured.

This unique aspect of Checkers has also been implemented in my program. Function king() changes a man into a king.

**My Program - CHECKERS**

Not wanting to complicate the program too much, I used a 2D array to make up the board and printed the lines of the board for simple error handling later on instead of using loops to print them out. I used the iostream header file for this.

I started with the hardest part of the program, implementing the constraints in the program. Using boolean was easy because it checks for only two values.

I progressed on, creating the king() function and added support for the multiple leaps: which are an essential part of checkers to win a game.

Having to implement a class in this program was especially difficult so I stuck with adding that in the highscore() function.

The highscore() function was an important and interesting part of the program; It used data file handling and a class injunction to parse time so the player could know when he won the game. I used the fstream and ctime header files for this.

My last part of the program was user experience and testing. I knew many people would have heard about Checkers but not actually know how to play it. I wanted to make it as easy as possible so I started printing instructions to aid with gameplay. I hope to optimise it much more in the near future. Adding comments throughout the program not only helped me keep track of what I needed to implent in the program but aided in reading through all the functions quickly.

I could have made a header file for all the functions and make my main() function much more easier to understand but I wanted to keep the program in one file and centralised.

Overall time to complete this project was two weeks, including basic mockups and error handling.

**Some statistics:**

* 449 lines of code
* Written in C++ Standard C++11 (hence usage of namespace)
* Program used to compile is Xcode Version 6.1

Code along with output is given in the following pages.

**Code**

#include <iostream> //basic input-output

#include <ctime> //header for current time when game is won

#include <fstream> //highscore file

using namespace std;

char board[8][8] =

{

{' ', 'b', ' ', 'b', ' ', 'b', ' ', 'b'},

{'b', ' ', 'b', ' ', 'b', ' ', 'b', ' '},

{' ', 'b', ' ', 'b', ' ', 'b', ' ', 'b'},

{' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},

{' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},

{'r', ' ', 'r', ' ', 'r', ' ', 'r', ' '},

{' ', 'r', ' ', 'r', ' ', 'r', ' ', 'r'},

{'r', ' ', 'r', ' ', 'r', ' ', 'r', ' '},

};

//variables

char turn = 'B';

bool leap;

bool game\_running = true;

int row1, row2, column1, column2;

//functions

void display\_board();

void input();

bool check\_move();

void move();

void do\_leap();

void king();

void game\_over();

void highscore();

int main()

{

cout << " \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \n";

cout << "| | | | \_\_| | | | \_\_| \_\_ | \_\_| \n";

cout << "| --| | \_\_| --| -| \_\_| -|\_\_ | \n";

cout << "|\_\_\_\_\_|\_\_|\_\_|\_\_\_\_\_|\_\_\_\_\_|\_\_|\_\_|\_\_\_\_\_|\_\_|\_\_|\_\_\_\_\_| \n";

cout <<" The Game \n" << endl;

cout << "Instructions:\n";

cout << "Player 1 is [b] on the checker board\n";

cout << "Player 2 is [r] on the checker board\n\n";

cout << "Multiple leaps are supported.\n";

cout << "A capital letter represents a king piece.\n";

cout << "King Pieces can be utilised when a piece reaches the end of the other side of the board.\n";

cout << "NOTE: Rows and columns are counted starting from 0, not 1.\n" <<endl;

cout << "<------COLUMNS------>\n";

cout << "^\n";

cout << "|\n";

cout << "|\n";

cout << "|\n";

cout << "ROWS\n";

cout << "|\n";

cout << "|\n";

cout << "|\n";

cout << "v\n\n";

cout << "NOTE: Resize your command line console so that this line fits on the screen:\n";

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n";

cout <<"Remember to input position with rows first then columns.\n";

cout <<"Good luck and may the best player win!\n\n";

cout << "Press enter to begin...";

cin.get(); //This waits for the user to press enter before continuing

while (game\_running)

{

display\_board();

if (turn == 'B')

{

cout << "--Player 1 [B]--\n";

}

else if (turn == 'R')

{

cout << "--Player 2 [R]--\n";

}

input();

move();

king();

game\_over();

}

if (turn == 'B')

{

cout << endl << endl << "Player 2 [Red] wins!!!\n";

highscore();

}

else if (turn == 'R')

{

cout << endl << endl << "Player 1 [Black] wins!!!\n";

highscore();

}

cout << "GAME OVER!\n";

}

void display\_board()

{

cout << "==================================================================================\n\n\n\n";

cout << " 0 1 2 3 4 5 6 7 \n";

cout << " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << " | | | | | | | | |\n";

cout << " | | | | | | | | |\n";

cout << "0 | " << board[0][0] << " | " << board[0][1] << " | " << board[0][2] << " | " << board[0][3] << " | " << board[0][4] << " | " << board[0][5] << " | " << board[0][6] << " | " << board[0][7] << " |\n";

cout << " | | | | | | | | |\n";

cout << " |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\n";

cout << " | | | | | | | | |\n";

cout << " | | | | | | | | |\n";

cout << "1 | " << board[1][0] << " | " << board[1][1] << " | " << board[1][2] << " | " << board[1][3] << " | " << board[1][4] << " | " << board[1][5] << " | " << board[1][6] << " | " << board[1][7] << " |\n";

cout << " | | | | | | | | |\n";

cout << " |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\n";

cout << " | | | | | | | | |\n";

cout << " | | | | | | | | |\n";

cout << "2 | " << board[2][0] << " | " << board[2][1] << " | " << board[2][2] << " | " << board[2][3] << " | " << board[2][4] << " | " << board[2][5] << " | " << board[2][6] << " | " << board[2][7] << " |\n";

cout << " | | | | | | | | |\n";

cout << " |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\n";

cout << " | | | | | | | | |\n";

cout << " | | | | | | | | |\n";

cout << "3 | " << board[3][0] << " | " << board[3][1] << " | " << board[3][2] << " | " << board[3][3] << " | " << board[3][4] << " | " << board[3][5] << " | " << board[3][6] << " | " << board[3][7] << " |\n";

cout << " | | | | | | | | |\n";

cout << " |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\n";

cout << " | | | | | | | | |\n";

cout << " | | | | | | | | |\n";

cout << "4 | " << board[4][0] << " | " << board[4][1] << " | " << board[4][2] << " | " << board[4][3] << " | " << board[4][4] << " | " << board[4][5] << " | " << board[4][6] << " | " << board[4][7] << " |\n";

cout << " | | | | | | | | |\n";

cout << " |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\n";

cout << " | | | | | | | | |\n";

cout << " | | | | | | | | |\n";

cout << "5 | " << board[5][0] << " | " << board[5][1] << " | " << board[5][2] << " | " << board[5][3] << " | " << board[5][4] << " | " << board[5][5] << " | " << board[5][6] << " | " << board[5][7] << " |\n";

cout << " | | | | | | | | |\n";

cout << " |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\n";

cout << " | | | | | | | | |\n";

cout << " | | | | | | | | |\n";

cout << "6 | " << board[6][0] << " | " << board[6][1] << " | " << board[6][2] << " | " << board[6][3] << " | " << board[6][4] << " | " << board[6][5] << " | " << board[6][6] << " | " << board[6][7] << " |\n";

cout << " | | | | | | | | |\n";

cout << " |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\n";

cout << " | | | | | | | | |\n";

cout << " | | | | | | | | |\n";

cout << "7 | " << board[7][0] << " | " << board[7][1] << " | " << board[7][2] << " | " << board[7][3] << " | " << board[7][4] << " | " << board[7][5] << " | " << board[7][6] << " | " << board[7][7] << " |\n";

cout << " | | | | | | | | |\n";

cout << " |\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\n";

}

void input()

{

cout << "Move piece\n";

cout << "Row: ";

cin >> row1;

cout << "Column: ";

cin >> column1;

while (row1 < 0 || row1 > 7 || column1 < 0 || column1 > 7) //handles errors in program while users enter data

{

cout << "Incorrect input. Enter numbers between 0 and 7.\n";

cout << "Move piece\n";

cout << "Row: ";

cin >> row1;

cout << "Column: ";

cin >> column1;

}

cout << "To box\n";

cout << "Row: ";

cin >> row2;

cout << "Column: ";

cin >> column2;

while (row2 < 0 || row2 > 7 || column2 < 0 || column2 > 7)

{

cout << "Incorrect input. Enter numbers between 0 and 7.\n";

cout << "To box\n";

cout << "Row: ";

cin >> row2;

cout << "Column: ";

cin >> column2;

}

while (check\_move() == false)

{

cout << "Illegal Move. Please try again. Remember, rows first then columns (left hand side then the top)\n";

input();

}

}

bool check\_move() //constructs restrictions based on rules of checkers and handles errors associated with it.

{

//checks if a non-king piece is moving backwards.

if (board[row1][column1] != 'B' && board[row1][column1] != 'R')

{

if ((turn == 'B' && row2 < row1) || (turn == 'R' && row2 > row1))

{

leap = false;

return false;

}

}

//checks if the location the piece is moving to is already taken.

if (board[row2][column2] != ' '){leap = false; return false;}

//checks if location entered by the user contains a piece to be moved.

if (board[row1][column1] == ' '){leap = false; return false;}

//checks if the piece isn't moving diagonally.

if (column1 == column2 || row1 == row2){leap = false; return false;}

//checks if the piece is moving by more than 1 column and only 1 row

if ((column2 > column1 + 1 || column2 < column1 - 1) && (row2 == row1 +1 || row2 == row1 - 1)) {leap = false; return false;}

//checks if the piece is leaping.

if (row2 > row1 + 1 || row2 < row1 - 1)

{

//checks if the piece is leaping too far.

if (row2 > row1 + 2 || row2 < row1 - 2){leap = false; return false;}

//checks if the piece isn't moving by exactly 2 columns

if (column2 != column1 + 2 && column2 != column1 - 2){leap = false; return false;}

//checks if the piece is leaping over another piece.

if (row2 > row1 && column2 > column1)

{if (board[row2 - 1][column2 - 1] == ' '){leap = false; return false;}}

else if (row2 > row1 && column2 < column1)

{if (board[row2 - 1][column2 + 1] == ' '){leap = false; return false;}}

else if (row2 < row1 && column2 > column1)

{if (board[row2 + 1][column2 - 1] == ' '){leap = false; return false;}}

else if (row2 < row1 && column2 < column1)

{if (board[row2 + 1][column2 + 1] == ' '){leap = false; return false;}}

leap = true; return true;}

leap = false; return true;

}

void move()

{

bool king\_piece = false;

if (board[row1][column1] == 'B' || board[row1][column1] == 'R')

{

king\_piece = true;

}

board[row1][column1] = ' ';

if (turn == 'B')

{

if (king\_piece == false)

{

board[row2][column2] = 'b';

}

else if (king\_piece == true)

{

board[row2][column2] = 'B';

}

turn = 'R';

}

else if (turn == 'R')

{

if (king\_piece == false)

{

board[row2][column2] = 'r';

}

else if (king\_piece == true)

{

board[row2][column2] = 'R';

}

turn = 'B';

}

if (leap == true)

{

do\_leap();

}

}

void do\_leap()

{

char answer;

//removes the checker piece after leap.

if (row2 > row1 && column2 > column1)

{

board[row2 - 1][column2 - 1] = ' ';

}

else if (row2 > row1 && column2 < column1)

{

board[row2 - 1][column2 + 1] = ' ';

}

else if (row2 < row1 && column2 > column1)

{

board[row2 + 1][column2 - 1] = ' ';

}

else if (row2 < row1 && column2 < column1)

{

board[row2 + 1][column2 + 1] = ' ';

}

display\_board();//displays/ refreshes the board after the changes

//asks if the user wants to leap again.

do

{

cout << "You just leaped once. Do you want to do a second leap IF YOU CAN? (y/n): ";

cin >> answer;

}

while (answer != 'Y' && answer != 'y' && answer != 'N' && answer != 'n');

if (answer == 'y' || answer == 'Y')

{

row1 = row2;

column1 = column2;

cout << "Leap piece: row: " << row1 << ", column: " << column1 << endl;

cout << "To box\n";

cout << "row: ";

cin >> row2;

cout << "column: ";

cin >> column2;

while (row2 < 0 || row2 > 7 || column2 < 0 || column2 > 7)

{

cout << "Incorrect input. Enter numbers between 0 and 7.\n";

cout << "To box\n";

cout << "Row: ";

cin >> row2;

cout << "Column: ";

cin >> column2;

}

if (turn == 'B')

{

turn = 'R';

}

else if (turn == 'R')

{

turn = 'B';

}

check\_move();

if (leap == false)

{

cout << "INVALID LEAP!!\n";

if (turn == 'B')

{

turn = 'R';

}

else if (turn == 'R')

{

turn = 'B';

}

}

else if (leap == true)

{

move();

}

}

}

void king() //capitalises the checker letter so it becomes a KING piece

{

for (int i = 0; i < 8; i++)

{

if (board[0][i] == 'r')

{

board[0][i] = 'R';

}

if (board[7][i] == 'b')

{

board[7][i] = 'B';

}

}

}

void game\_over()

{

int counter = 0;

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 8; j++)

{

if (board[i][j] != ' ')

{

counter++;

}

}

}

if (counter > 1)

{

game\_running = true;

}

else if (counter == 1)

{

game\_running = false;

}

}

void highscore()

{

{

char data[100];

time\_t currentTime;

class tm \*localTime;

time( &currentTime ); // Get the current time

localTime = localtime( &currentTime ); // Convert the current time to the local time

int Hour = localTime->tm\_hour;

int Min = localTime->tm\_min;

int Sec = localTime->tm\_sec;

// open the highscore file in write mode.

ofstream outfile;

outfile.open("highscore.dat");

cout << "\nWriting to the highscore file\n" << endl;

cout << "\nEnter your name: ";

cin.getline(data,100);

outfile << data << endl;

cout << "\nEnter your age: ";

cin >> data;

cin.ignore();

outfile << data << endl;

outfile << Hour << Min << Sec << endl;

outfile.close();

ifstream infile;

infile.open("highscore.dat");

cout << "Reading from the highscore file" << endl;

infile >> data;

cout << "\nName: " << data << endl;

infile >> data;

cout << "\nAge: " << data << endl;

infile >> data;

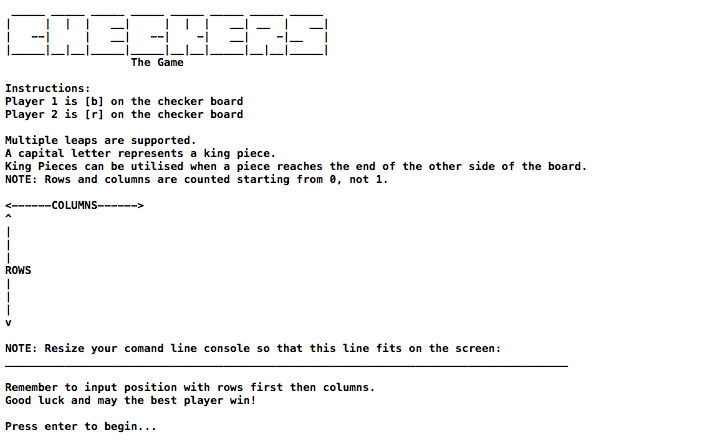
cout<< "\nTime game was won:" << Hour << ":" << Min << ":" << Sec;

infile.close();

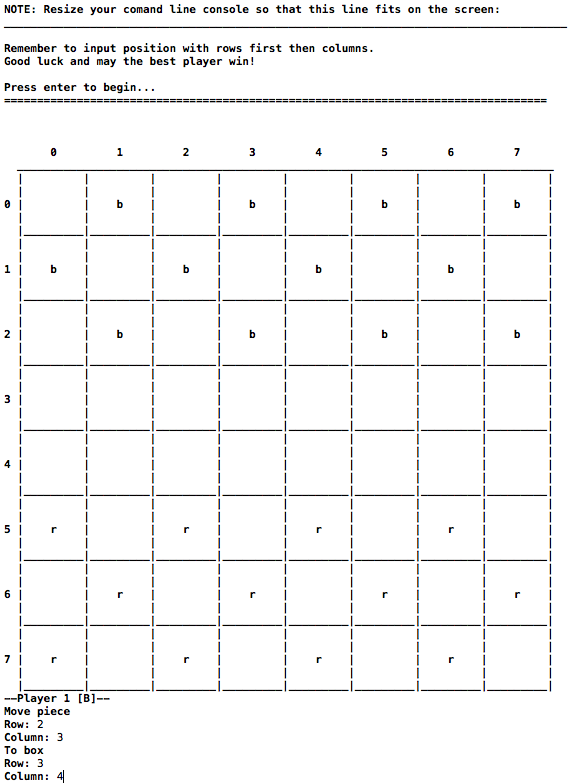
}

}

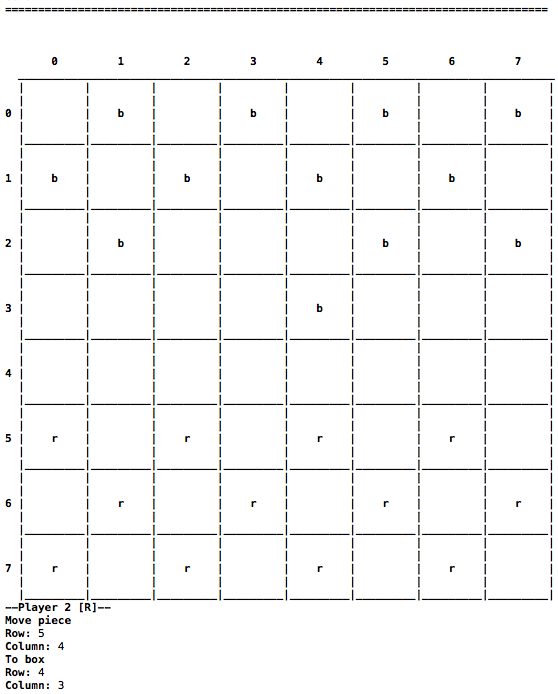
**Output Screens**



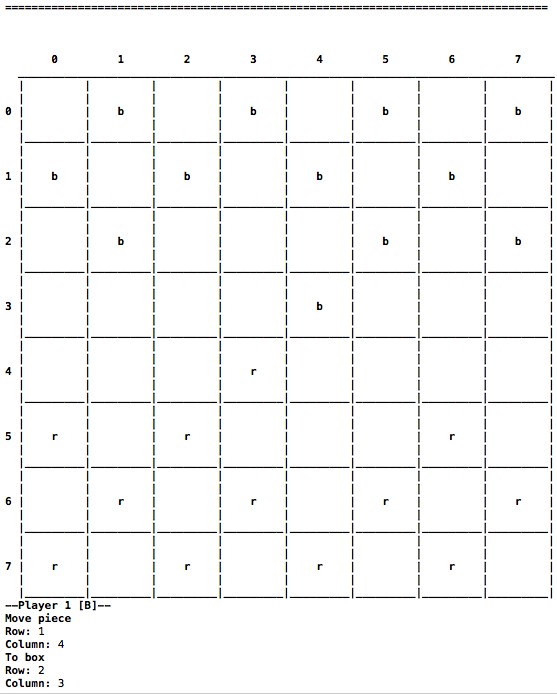
**Instruction Screen with ASCII text as a header**



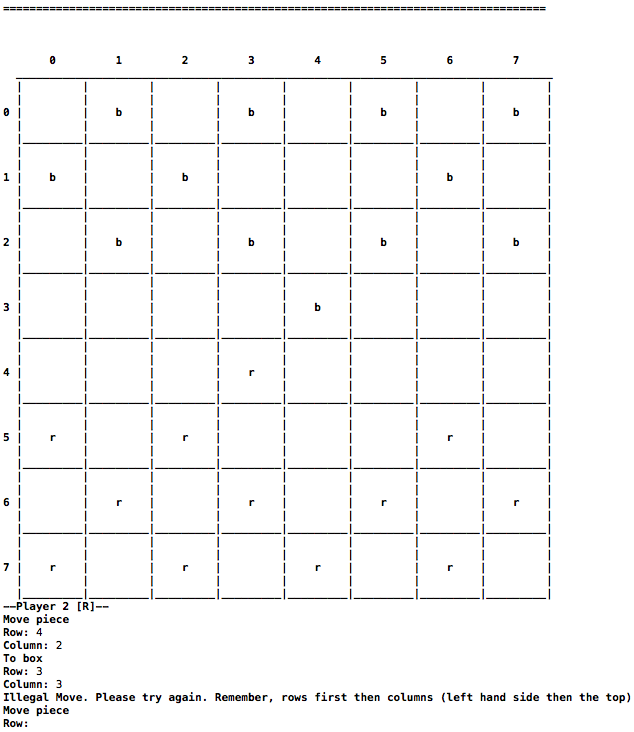
**First move; Board is generated and pieces are placed. Data is added first by rows then by columns (left then right)**



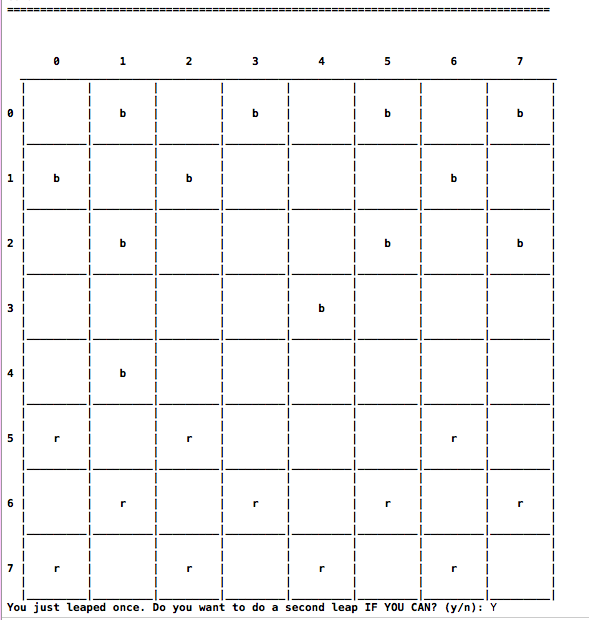
**Second move; ‘men’ are moved across the board.**



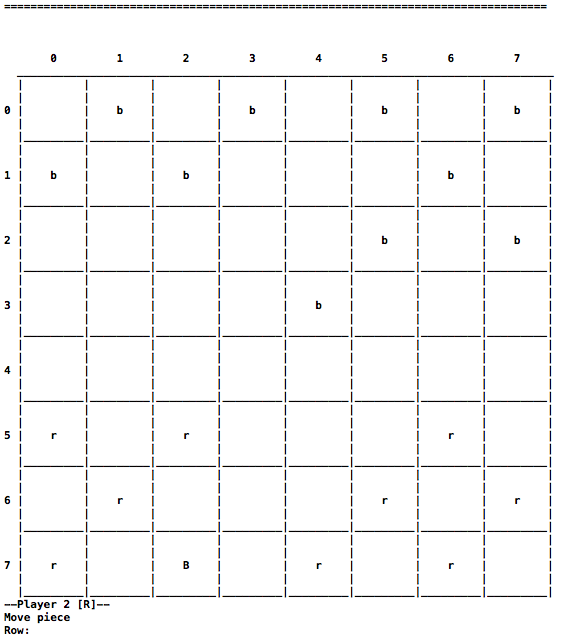
**Third move; game progresses.**



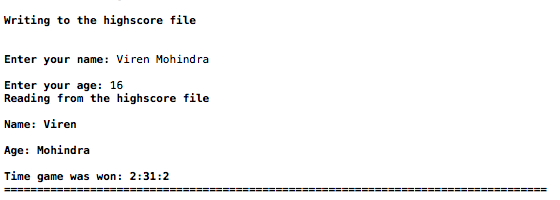
**Fourth Move; Error handling is done if user makes a mistake. Small snippet of how to play is printed along with the same move repeated.**



**Fifth move; Leaping in action. R Piece has taken command of B and has another chance. User can either choose Yes/ No to leap more.**



**b piece has reached the end of the board; King() function is called and the piece immediately turns into a capital letter (B), signifying it has become a king. Now there are no constraints on that specific piece; it can move forwards as well as backwards.**



**End of game. Highscore() function is called; Data file handling is used for the input and saving of data while classes are used for the displaying of current time (when the game was won). Program ends.**