

# BATTLE REVERSI

CS101  
IIT BOMBAY  
AUTUMN 2014



## TEAM MEMBERS:

HIMANSHU DENGRE  
ARVIND SHANKARA PS  
ZULFIQAR ALI  
AAKASH KUMAR

## Introduction to Project

The project is based on the well known classic game of Reversi. Reversi is a strategy board game for two players, played on an 8×8 uncheckered board. There are sixty-four identical game pieces called disks, which are light on one side and dark on the other. Players take turns placing disks on the board with their assigned color facing up. During a play, any disks of the opponent's color that are in a straight line and bounded by the disk just placed and another disk of the current player's color are turned over to the current player's color.

The object of the game is to have the majority of disks turned to display your color when the last playable empty square is filled.

## Specific Requirements

Minimum Requirements-

The system must have

- \*at least 256MB of RAM
- \*at least 2 MB of free space on hdd
- \*pentium 4 or later
- \*a keyboard
- \*a simple cpp c++ compiler preferably code blocks

## General Description

When the player opens and starts playing the game, the game automatically keeps a track record of the score of the player. It also keeps the record of the high scores of the players. The player has the option of selecting the difficulty level of the game.

## Software Interface

The game's start-up screen has multiple options like GAME MODE, , INSTRUCTIONS, ABOUT, and EXIT.

The gameplay screen of the game is a simple screen with an 8x8 checkerboard.



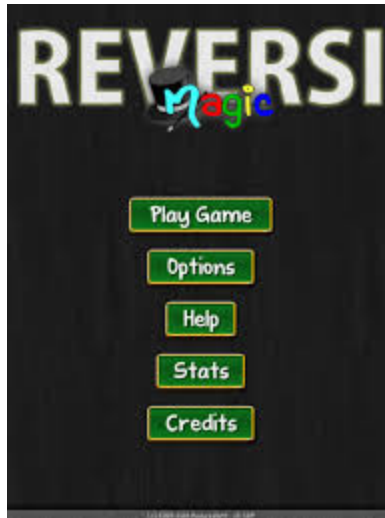
The screen also has a points counter at the top to keep track of the number of discs of each colour.

## Hardware Interface

The pointer is controlled by the four arrow keys to move the pointer and the enter key to place his piece. The startup screen has different options which can be selected by pressing the key corresponding to the choice.

## User Interface

The first window displays the name of the game and asks the user to press any key to begin. This takes the user to the main menu which includes the following tabs:



- a) game mode – Includes two options: “single player” and “two players”.
- b) instructions – Provides the controls and the objective of the game.
- d) credit – Displays the group members' names and their contribution.
- f) Exit – Quits the game.

## Features:

The game intends to develop the following features:

Suggesting the best move.

Storage of user information

Maintenance of top five scores..

Different levels of difficulty.

## Functions Used

```
main menu();  
new game();  
check(); //to check the input(also to validate it)  
bestmove(); //to suggest best move  
game_start(); //start of game  
game_over(); //end of game  
clscreen(); //clear screen  
crd(); //credit  
instructions(); //help  
gameplay();  
get cord();
```

## Libraries Used:

```
iostream  
simplecpp  
screenprnt
```

## Algorithm

Our project will be following the the minimax algorithm .a minimax strategy is a mixed strategy which is part of the solution to a zero-sum game.

For every two-person, zero sum game with finitely many strategies, there exists a value  $V$  and a mixed strategy for each player, such that

- (a) Given player 2's strategy, the best payoff possible for player 1 is  $V$ , and
- (b) Given player 1's strategy, the best payoff possible for player 2 is  $-V$ .

Equivalently, Player 1's strategy guarantees him a payoff of  $V$  regardless of Player 2's strategy, and similarly Player 2 can guarantee himself a payoff of  $-V$ . The name minimax arises because each player minimizes the maximum payoff possible for the other—since the

game is zero-sum, he also minimizes his own maximum loss (i.e. maximize his minimum payoff).

A **minimax algorithm** is actually a recursive algorithm for choosing the next move in an n-player game, usually a two-player game. A value is associated with each position or state of the game. This value is computed by means of a positive evaluation function and it indicates how good it would be for a player to reach that position. The player then makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves. If it is **A**'s turn to move, **A** gives a value to each of his legal moves.