Miguel Porcar Querol
Erasmus student

REVERSI

Reversi.c

The functionality of the main function is to parse the arguments and prepare the configuration for the game. A problem I found when parsing the 'size' parameter was that optarg had sometimes the value and others it was in argv[optind]. When I finish parsing the arguments, I call game(filename).

If filename is null I load a normal board, if not I charge the one saved. Then I start an infinite loop with all the logic. First I check if there is any movement available for the current player, if not turn moves and if again there is no possible move, game is over. When the game is set without any AI player it executes human_player(), explained in the next paragraph. When there are automatic players ai_player, which returns a move and then it is applied against the board. Finally it changes the turn.

Human_player reads, parses and tries to apply a move with the current state. When reading the move I just use a pointer which continues moving if there are spaces and subtracts 'a' to the letter in order to know the row and subtracts 1 to the column. To check whether the move is correct or not, we use board_play, which had a previous implementation when the board was char ** but now it only calls bb_move from bitboard.c.

The load of the board has been done with the previous implementation as with char** board but at the end it moves it to a bitboard_t structure.

Bitboard.c

First of all, to manipulate the bits of the uint64_t boards I made two functions which saved me much time: get_bit() and set_bit(). There is also another function *one_dimension(x, y, size)* which lets me use the boards as bi-dimensional arrays.

The functions *bb_new* and *bb_set* have a trivial implementation very easily readable.

bb_init() returns the board with the four central boxes already filled.

bb_print() goes over the board and prints whether the box is a 'O', 'X' or is empty.

bb_*score()* counts the number of active bits in every board and returns it as an score.

bb_move() is one of the trickiest functions. It has to check if a move is possible and if it is apply it to the board. It tries to every direction passing to the function on the direction 'false' to indicate it

has not to be changed the board and later true to change.

bb_moves() executes *bb_move* with every box and returns the corresponding bitboard.

The AI algorithms share a common structure. The given header of the function executes an auxiliary one with the parameters accommodated to recursion. The recursive ones return an *algo_t* structure which is composed by a value (for the algorithm) and a move (the one returned). Their implementations have been followed by the pseudocodes given in the wikipedia sites provided.

I have also implemented the coin parity heuristic which captures the difference in coins between the max player and the min player. After several simulations of different programs, I found minimax with alpha beta pruning is the best.