```
.
+-- reversi/
    +-- check_parser_results.py
    +-- include/
    |   +-- board.h
    |   +-- player.h
    +-- Makefile
    +-- src/
    |   +-- board.c
    |   +-- Makefile
    |   +-- player.c
    |   +-- reversi.c
    |   +-- reversi.h
    +-- test/
    |   +-- board_tests.c
    |   +-- Makefile
    +-- test_player/
        +-- Makefile
        +-- sample_test.sh
        +-- test_player*
        +-- test_player.c

5 directories, 15 files
```



```
# Variables
EXE = my_tests_player

# Usual compilation flags
CFLAGS = -std=c11 -Wall -Wextra -g -O2
CPPFLAGS = -I../include -DDEBUG

# Special rules and targets
.PHONY: all clean help

# Rules and targets
all: test_player

board_tests: test_player.o
	$(CC) $(CFLAGS) -o $@ $^

test_player.o: test_player.c
	$(CC) $(CFLAGS) $(CPPFLAGS) -c $^

clean:
	@rm -f *~ *.o $(EXE)

help:
	@echo "Usage:"
	@echo " make [all]\t\tBuild the software"
	@echo " make clean\t\tRemove all files generated by make"
	@echo " make help\t\tDisplay this help"
```

```
# Variables
EXE = my_tests

# Usual compilation flags
CFLAGS = -std=c11 -Wall -Wextra -g -O2
CPPFLAGS = -I../include -DDEBUG

# Special rules and targets
.PHONY: all clean help

# Rules and targets
all: board_tests

board_tests: board_tests.o board.o
	$(CC) $(CFLAGS) -o $@ $^

board.o: ../src/board.c ../include/board.h
	$(CC) $(CFLAGS) $(CPPFLAGS) -c $^

board_test.o: board_tests.c ../include/board.h
	$(CC) $(CFLAGS) $(CPPFLAGS) -c $^

clean:
	@rm -f *~ *.o $(EXE)

help:
	@echo "Usage:"
	@echo " make [all]\t\tBuild the software"
	@echo " make clean\t\tRemove all files generated by make"
	@echo " make help\t\tDisplay this help"
```

COUMAU_Louis (Group1) **Makefile** Page 1/1

```
# Variables
EXE = reversi

# Special rules and targets
.PHONY: all build clean help

# Rules and targets
all: build

build:
	@cd src && $(MAKE)
	@cd test && $(MAKE)
	@cd test_player && $(MAKE)
	@cp -f src/$(EXE) ./

clean:
	@cd src && $(MAKE) clean
	@cd test && $(MAKE) clean
	@cd test_player && $(MAKE) clean
	@rm -f $(EXE)

help:
	@echo "Usage:"
	@echo " make [all]\t\tBuild"
	@echo " make build\t\tBuild the software"
	@echo " make clean\t\tRemove all files generated by make"
	@echo " make help\t\tDisplay this help"
```

```
# Variables
EXE = reversi

# Usual compilation flags
CFLAGS = -std=c11 -Wall -Wextra -g
CPPFLAGS = -I../include -DDEBUG
LDFLAGS =

# Special rules and targets
.PHONY: all clean help

# Rules and targets
all: reversi

reversi: reversi.o board.o player.o
	$(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)

reversi.o: reversi.c reversi.h ../include/board.h ../include/player.h
	$(CC) $(CFLAGS) $(CPPFLAGS) -c $<

board.o: board.c ../include/board.h
	$(CC) $(CFLAGS) $(CPPFLAGS) -c $<

player.o: player.c ../include/player.h ../include/board.h
	$(CC) $(CFLAGS) $(CPPFLAGS) -c $<

clean:
	@rm -f *~ *.o $(EXE)

help:
	@echo "Usage:"
	@echo " make [all]\t\tBuild the software"
	@echo " make clean\t\tRemove all files generated by make"
	@echo " make help\t\tDisplay this help"
```



```
#include "board.h"

#include <err.h>
#include <stdbool.h>
#include <string.h>
#include <time.h>

/* Internal board_t structure (hidden from the outside) */
struct board_t {
  size_t size;
  disc_t player;
  bitboard_t black;
  bitboard_t white;
  bitboard_t moves;
  bitboard_t next_move;
};

/* set the bit to 1 at row, column on the bitboard */
/* row=[0,size-1] column=[0,size-1] */
static bitboard_t set_bitboard(const size_t size, const size_t row,
                               const size_t column) {
  bitboard_t new_bitboard = 1;
  int power = (size * row + column);
  return new_bitboard << power;
}

static bitboard_t shift_north(const size_t size, const bitboard_t bitboard) {
  return (bitboard >> size);
}

static bitboard_t shift_south(const size_t size, const bitboard_t bitboard) {
  switch (size) {
  case 2:
    return (bitboard << size) & ~0x30;
  case 4:
    return (bitboard << size) & ~0xF0000;
  case 6:
    return (bitboard << size) & ~MASK_BITB_BOTTOM_6;
  case 8:
    return (bitboard << size) & ~MASK_BITB_BOTTOM_8;
  case 10:
    return (bitboard << size) & ~MASK_BITB_BOTTOM_10;
  }
  return -1;
}

static bitboard_t shift_west(const size_t size, const bitboard_t bitboard) {
  switch (size) {
  case 2:
    return (bitboard >> 1) & ~0xA;
  case 4:
    return (bitboard >> 1) & ~0x8888;
  case 6:
    return (bitboard >> 1) & ~0x820820820;
  case 8:
    return (bitboard >> 1) & ~0x8080808080808080;
  case 10:
    return (bitboard >> 1) & ~MASK_BITB_RIGHT_10;
  }
  return -1;
}

static bitboard_t shift_est(const size_t size, const bitboard_t bitboard) {
  switch (size) {
```

```
  case 2:
    return (bitboard << 1) & ~0x5 & ~0x30;
  case 4:
    return (bitboard << 1) & ~0x1111 & ~0xF0000;
  case 6:
    return (bitboard << 1) & ~0x41041041 & ~MASK_BITB_BOTTOM_6;
  case 8:
    return (bitboard << 1) & ~0x1010101010101010 & ~MASK_BITB_BOTTOM_8;
  case 10:
    return (bitboard << 1) & ~MASK_BITB_LEFT_10 & ~MASK_BITB_BOTTOM_10;
  }
  return -1;
}

static bitboard_t shift_nw(const size_t size, const bitboard_t bitboard) {
  return shift_west(size, shift_north(size, bitboard));
}

static bitboard_t shift_ne(const size_t size, const bitboard_t bitboard) {
  return shift_est(size, shift_north(size, bitboard));
}

static bitboard_t shift_sw(const size_t size, const bitboard_t bitboard) {
  return shift_west(size, shift_south(size, bitboard));
}

static bitboard_t shift_se(const size_t size, const bitboard_t bitboard) {
  return shift_est(size, shift_south(size, bitboard));
}

/* array of shift functions */
static bitboard_t (*shift_func[SIZE_SHIFT_ARRAY])(const size_t size,
                                                  const bitboard_t bitboard) = {
    shift_north, shift_south, shift_west, shift_est,
    shift_nw,    shift_ne,    shift_sw,   shift_se};

/* compute all the possible moves */
static bitboard_t compute_moves(const size_t size, const bitboard_t player,
                                const bitboard_t opponent) {
  bitboard_t moves = 0;
  bitboard_t candidates;
  for (size_t i = 0; i < SIZE_SHIFT_ARRAY; i++) { /* for each directions */
    candidates = opponent & (shift_func[i](size, player));
    while (candidates != 0) {
      moves |= shift_func[i](size, candidates);
      candidates = opponent & shift_func[i](size, candidates);
    }
  }
  return moves & ~opponent & ~player;
}

board_t *board_alloc(const size_t size, const disc_t player) {
  board_t *board = malloc(sizeof(board_t));
  if (board == NULL) {
    fprintf(stderr, "board.c:board_alloc(): error: error of malloc board\n");
    return NULL;
  }
  board->size = size;
  board->player = player;
  board->black = 0;
  board->white = 0;
  board->moves = 0;
  board->next_move = 0;
  return board;
```



```
}

void board_free(board_t *board) {
  if (board != NULL) {
    free(board);
  }
}

board_t *board_init(const size_t size) {
  /* verification of size */
  if (size % 2 != 0 || size < MIN_BOARD_SIZE || size > MAX_BOARD_SIZE) {
    fprintf(stderr, "board.c:board_init(): error: error of size (%zu)\n", size);
    return NULL;
  }
  board_t *board = board_alloc(size, BLACK_DISC); /* creat a new void board */
  /* set white disc on the board */
  board->white = set_bitboard(size, (size / 2) - 1, (size / 2) - 1) |
                 set_bitboard(size, size / 2, size / 2);
  /* set black disc on the board */
  board->black = set_bitboard(size, (size / 2) - 1, size / 2) |
                 set_bitboard(size, size / 2, (size / 2) - 1);
  /* init possibles moves */
  board->moves = compute_moves(size, board->black, board->white);
  if (!board->moves) { /* if move is not possible */
    board->player = EMPTY_DISC;
  }
  return board;
}

board_t *board_copy(const board_t *board) {
  if (board == NULL) {
    return NULL;
  }
  size_t size_copy = board->size;     /* get size */
  disc_t player_copy = board->player; /* get player */
  board_t *board_copy = board_alloc(size_copy, player_copy);
  board_copy->black = board->black;
  board_copy->white = board->white;
  board_copy->moves = board->moves;
  board_copy->next_move = board->next_move;
  return board_copy;
}

size_t board_size(const board_t *board) { return board->size; }

disc_t board_player(const board_t *board) { return board->player; }

void board_set_player(board_t *board, disc_t new_player) {
  if (new_player != HINT_DISC) {
    board->player = new_player;
  } else {
    fprintf(stderr, "board.c:board_set_player(): error: you try to set player "
                    "with EMPTY_DISC\n");
  }
}

disc_t board_get(const board_t *board, const size_t row, const size_t column) {
  if (board != NULL) {
    size_t size = board->size;
    bitboard_t current_bitboard = set_bitboard(size, row, column);
    if ((board->moves) & current_bitboard) {
      return HINT_DISC;
    }
    if ((board->black) & current_bitboard) {
```

```
      return BLACK_DISC;
    }
    if ((board->white) & current_bitboard) {
      return WHITE_DISC;
    }
  }
  return EMPTY_DISC;
}

/* set the given disc at the given position */
void board_set(board_t *board, const disc_t disc, const size_t row,
               const size_t column) {
  if (board != NULL) {
    size_t size = board->size;
    if (row < size && column < size) {
      bitboard_t current_bitboard = set_bitboard(size, row, column);
      switch (disc) {
      case BLACK_DISC:
        board->black = (board->black) | current_bitboard;
        board->white = (board->white) & ~current_bitboard;
        break;
      case WHITE_DISC:
        board->white = (board->white) | current_bitboard;
        board->black = (board->black) & ~current_bitboard;
        break;
      case HINT_DISC:
        board->moves = (board->moves) | current_bitboard;
        break;
      case EMPTY_DISC:
        board->black = (board->black) & ~current_bitboard;
        board->white = (board->white) & ~current_bitboard;
        break;
      }
    }
    if (board->player == BLACK_DISC) {
      board->moves = compute_moves(size, board->black, board->white);
    } else {
      board->moves = compute_moves(size, board->white, board->black);
    }
  }
}

/* count the number of bits set to 1 */
static size_t bitboard_popcount(const bitboard_t bitboard) {
  bitboard_t i = bitboard;
  i = i - ((i >> 1) & MASK1_POPCOUNT);
  i = (i & MASK2_POPCOUNT) + ((i >> 2) & MASK2_POPCOUNT);
  return (((i + (i >> 4)) & MASK3_POPCOUNT) * MASK4_POPCOUNT) >> 120;
}

score_t board_score(const board_t *board) {
  score_t score;
  score.black = bitboard_popcount(board->black);
  score.white = bitboard_popcount(board->white);
  return score;
}

size_t board_count_player_moves(board_t *board) {
  return bitboard_popcount(board->moves);
}

size_t board_count_opponent_moves(board_t *board) {
  if (board->player == BLACK_DISC) {
    return bitboard_popcount(
```



```
        compute_moves(board->size, board->white, board->black));
  } else {
    return bitboard_popcount(
        compute_moves(board->size, board->black, board->white));
  }
}

bool board_is_move_valid(const board_t *board, const move_t move) {
  bitboard_t bitboard_move = set_bitboard(board->size, move.row, move.column);
  return (board->moves) & bitboard_move;
}

/* find all trace until the current player */
static bitboard_t trace_move(board_t *board, const move_t move) {
  disc_t current_player = board->player;
  size_t size = board->size;
  bitboard_t player = board->black; /* init with BLACK_DISC player */
  bitboard_t opponent = board->white;
  if (current_player == WHITE_DISC) {
    player = board->white;
    opponent = board->black;
  }

  bitboard_t start = set_bitboard(size, move.row, move.column);
  bitboard_t trace;
  bitboard_t final_trace = 0;
  bitboard_t shift;

  for (size_t i = 0; i < SIZE_SHIFT_ARRAY; i++) {
    trace = start;
    shift = shift_func[i](size, start);
    while ((shift & opponent) != 0) {
      trace |= shift;
      shift = shift_func[i](size, shift);
      if (shift & player) {
        final_trace |= trace;
      }
    }
  }
  return final_trace;
}

bool board_play(board_t *board, const move_t move) {
  size_t size = board->size;  /* get size */
  bool returned_value = true; /* false if invalid move is given */
  disc_t current_player = board->player;

  if (current_player == BLACK_DISC) { /* BLACK_DISC player plays */
    if (board_count_player_moves(board) != 0 &&
        board_is_move_valid(board, move)) {
      bitboard_t trace =
          trace_move(board, move); /* get the trace of direction */
      board->white &= ~trace;
      board->black |= trace;
    } else { /* invalid move */
      returned_value = false;
    }
    board->moves = compute_moves(size, board->white, board->black);
    board->player = WHITE_DISC; /* alternate in all cases */
  } else {                      /* WHITE_DISC player plays */
    /* if move is valid an player can plays */
    if (board_count_player_moves(board) != 0 &&
        board_is_move_valid(board, move)) {
      bitboard_t trace =
```

```
          trace_move(board, move); /* get the trace of direction */
      board->black &= ~trace;
      board->white |= trace;
    } else { /* invalid move */
      returned_value = false;
    }
    board->moves = compute_moves(size, board->black, board->white);
    board->player = BLACK_DISC; /* alternate in all cases */
  }
  board->next_move = board->moves;
  /* if black and white players can't play after playing */
  if (board_count_opponent_moves(board) == 0 &&
      board_count_player_moves(board) == 0) {
    board->player = EMPTY_DISC;
    return true;
  }

  return returned_value;
}

move_t board_next_move(board_t *board) {
  move_t move;
  move.row = 0;
  move.column = 0;
  int nbr_tz = 0; /* no of zeros from last to first occurence of one */

  if (board == NULL) {
    move.row = -1;
    move.column = -1;
    return move;
  }
  if (board->next_move == 0) {
    board->next_move = board->moves;
  }
  size_t size = board->size;                    /* get the size of board */
  bitboard_t possibles_moves = board->next_move; /* get possibles moves */
  if (size < 10) {
    /* if possible_moves is a 64bit number */
    nbr_tz = __builtin_ctzll(possibles_moves);
  } else {
    nbr_tz = bitboard_popcount((possibles_moves - 1) ^ possibles_moves) - 1;
  }
  board->next_move &= possibles_moves - 1; /* remove the position */
  move.row = nbr_tz / size;                /* compute row */
  move.column = nbr_tz % size;             /* compute column */
  return move;
}

int board_print(const board_t *board, FILE *fd) {
  if (fd == NULL || board == NULL) {
    return -1;
  }
  int nbr_char = 0;                      /* number of printed characters */
  size_t size = board->size;             /* get size */
  disc_t current_player = board->player; /* get player */
  score_t score = board_score(board);    /* set the score */
  char *space_tampon = " ";             /* set the initial space */
  char current_char = 'A';               /* set the initial character */
  if (current_player == EMPTY_DISC) {
    nbr_char += fprintf(fd, "\nGame over.\n");
  } else {
    nbr_char += fprintf(fd, "\n%c player's turn.\n", current_player);
  }
  nbr_char += fprintf(fd, "\n  ");
```



```
  for (size_t i = 0; i < size; i++) { /* write A B C D... */
    nbr_char += fprintf(fd, "%c ", (char)(current_char + i));
  }
  nbr_char += fprintf(fd, "\n");

  for (size_t i = 0; i < size; i++) /* write board */
  {
    if (i + 1 == 10) {
      space_tampon = " "; /* reduce two space to one space */
    }
    nbr_char += fprintf(fd, "%s%lu ", space_tampon, i + 1);
    for (size_t j = 0; j < size; j++) {
      nbr_char += fprintf(fd, "%c ", board_get(board, i, j));
    }
    nbr_char += fprintf(fd, "\n");
  }
  nbr_char += fprintf(fd, "\n");
  nbr_char +=
      fprintf(fd, "Score: 'X' = %d, 'O' = %d\n\n", score.black, score.white);
  return nbr_char;
}
```

```c
#ifndef REVERSI_H
#define REVERSI_H

#define VERSION 1
#define SUBVERSION 0
#define REVISION 0

#endif /* REVERSI_H */
```



```c
#include "reversi.h"
#include "board.h"
#include "player.h"

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#include <err.h>
#include <errno.h>
#include <getopt.h>
#include <string.h>

#define MAX_LENGTH 512
/* define number of player functions and change array */
#define NBR_PLAY_FUNC 5
static move_t (*play_func[NBR_PLAY_FUNC])(board_t *board) = {
    human_player, random_player, simul_minimax_player, simul_alpha_beta_player,
    simul_alpha_beta_bis_player};
static char *name_play_func[NBR_PLAY_FUNC] = {
    "human", "random", "minimax_player", "alpha_beta_player",
    "alpha_beta_bis_player"};

static bool VERBOSE = false; /* verbose variable */

/* use for keep disc player in file parser */
typedef struct {
  disc_t player;
  int x;
  int y;
} coor_disc;

static char get_alpha_column(size_t row) {
  int alpha_maj[10] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'};
  return alpha_maj[row];
}

static int game(move_t (*black)(board_t *), move_t (*white)(board_t *),
                board_t *board) {
  /**/
  int possible_moves_sum = 0;
  int nbr_of_turn = 0;
  /**/
  disc_t current_player = board_player(board); /* get current player */
  score_t score;
  size_t black_score;
  size_t white_score;
  size_t size = board_size(board);
  move_t move;
  /*welcome print part */
  char *black_player_type;
  char *white_player_type;
  char *first_player = "Black";
  if (current_player == WHITE_DISC) {
    first_player = "White";
  }
  for (size_t i = 0; i < NBR_PLAY_FUNC; i++) {
    if (black == play_func[i]) {
      black_player_type = name_play_func[i];
    }
    if (white == play_func[i]) {
      white_player_type = name_play_func[i];
    }
  }
```

```
  fprintf(stdout,
          "Welcome to this reversi game !\n"
          "Black player (X) is %s and white player (O) is %s.\n"
          "%s player start !\n",
          black_player_type, white_player_type, first_player);

  if (!VERBOSE) {
    fprintf(stdout, "\nPlaying...\n");
  }
  /* main loop */
  while (current_player != EMPTY_DISC) {
    /* Analyse part */
    nbr_of_turn++;
    possible_moves_sum += board_count_player_moves(board);
    /****************/
    if (current_player == BLACK_DISC) {          /* turn of black player */
      if (board_count_player_moves(board) > 0) { /* player can play */
        move = black(board);                     /* get the move */
        if (black == human_player) {
          if (move.row == size && move.column == size) {
            fprintf(stdout, "Player 'X' resigned. player 'O' win the game.\n");
            return -1;
          }
        } else {          /* if random player or others */
          if (VERBOSE) { /* if verbose option is activated */
            board_print(board, stdout);
            fprintf(stdout, "'X' plays %c%zu\n", get_alpha_column(move.column),
                    (move.row + 1));
          }
        }
      }
      board_play(board, move); /* play in all cases */
    }
    if (current_player == WHITE_DISC) { /* turn of white player */
      if (board_count_player_moves(board) > 0) {
        move = white(board); /* get the move */
        if (white == human_player) {
          if (move.row == size && move.column == size) {
            fprintf(stdout, "Player 'O' resigned. player 'X' win the game.\n");
            return -2;
          }
        } else {          /* if random player or others */
          if (VERBOSE) { /* if verbose option is activated */
            board_print(board, stdout);
            fprintf(stdout, "'O' plays %c%zu\n", get_alpha_column(move.column),
                    move.row + 1);
          }
        }
      }
      board_play(board, move); /* play in all cases */
    }
    if (VERBOSE) {
      fprintf(stdout, "========================================\n");
    }
    current_player = board_player(board);
  }
  board_print(board, stdout);
  score = board_score(board);
  black_score = score.black;
  white_score = score.white;
  /* result of analyse */
  fprintf(stdout, "the average of possibles moves %d, sum : %d, turn : %d\n",
          possible_moves_sum / nbr_of_turn, possible_moves_sum, nbr_of_turn);
  /**/
```



```
  if (black_score > white_score) {
    fprintf(stdout, "Player 'X' win the game.\n");
    return 1;
  }
  if (black_score < white_score) {
    fprintf(stdout, "Player 'O' win the game.\n");
    return 2;
  }
  /* if black_score == white_score */
  fprintf(stdout, "Draw game, no winner.\n");
  return 0;
}

static board_t *file_parser(const char *filename) {
  FILE *file;
  file = fopen(filename, "r+");
  if (!file) {
    fprintf(stderr,
            "reversi.c:file_parser(): error: failed to read file : %s\n",
            strerror(errno));
    exit(EXIT_FAILURE);
  }

  /* boolean type */
  bool begin_board = false;
  bool comment_status = false;
  bool content = false;
  bool player_is_saved = false;

  /* int type */
  int current_column_count = 0;
  int line_count = 0;
  int size_column = 0;
  int size_raw = 0;

  /* other type */
  int current_char = '\0';
  size_t nbr_of_disc = 0;
  disc_t player;
  coor_disc tab_disc[100];

  while (current_char != EOF) {

    current_char = fgetc(file);

    switch (current_char) {
    case '#':
      comment_status = true;
      break;
    case 'O':
    case 'X':
      if (!comment_status) {    /* not comment line */
        if (!player_is_saved) { /* player is not saved */
          player = current_char;
          player_is_saved = true;
        } else {
          begin_board = true;
          content = true;
          /* keep coordinate in tab_disc */
          tab_disc[nbr_of_disc].player = current_char;
          tab_disc[nbr_of_disc].x = current_column_count;
          tab_disc[nbr_of_disc].y = size_raw;
          nbr_of_disc++;
          current_column_count++;
```

```
        }
      }
      break;
    case '_': /* beginning of board */
      if (!comment_status) {
        begin_board = true;
        if (!player_is_saved) {
          errx(1, "player is missing");
        }
        content = true;
        current_column_count++;
      }
      break;
    case EOF:
    case '\n':
      line_count++;
      if (begin_board) {
        if (content) {
          size_raw++;
        }
        if (size_column == 0) {
          size_column = current_column_count;
        }
      }
      if (content) {
        if (current_column_count != size_column) {
          if (size_column > current_column_count) {
            fprintf(stderr, "reversi: character is missed in line %d\n",
                    line_count);
          } else {
            fprintf(stderr, "reversi: too much character in line %d\n",
                    line_count);
          }
          exit(EXIT_FAILURE);
        }
      }
      current_column_count = 0;
      comment_status = false;
      content = false;
      break;
    default:
      if (current_char != ' ' && current_char != '\t' && !comment_status) {
        if (!player_is_saved) {
          fprintf(stderr, "reversi: player is incorrect\n");
        } else {
          fprintf(stderr, "reversi: wrong character %c at line %d\n",
                  current_char, line_count + 1);
        }
        exit(EXIT_FAILURE);
      }
      break;
    }
  } /* end of while */
  if (size_raw != size_column) {
    if (size_raw > size_column) {
      fprintf(stderr, "reversi: board has %d extra raw\n",
              size_raw - size_column);
    } else {
      fprintf(stderr, "reversi: board has %d missing raw\n",
              size_column - size_raw);
    }
    exit(EXIT_FAILURE);
  }
  if (size_raw % 2 != 0) {
```



```
    errx(1, "reversi: size of board need to be pair");
  }
  if (size_raw < MIN_BOARD_SIZE || size_raw > MAX_BOARD_SIZE) {
    if (size_raw == 0) {
      fprintf(stderr, "reversi: error: missing board\n");
    } else {
      fprintf(stderr,
              "reversi: size of square need to be 2, 4, 6, 8 or 10, not %d\n",
              size_raw);
    }
    exit(EXIT_FAILURE);
  }
  fclose(file);
  size_t size = size_raw;
  board_t *newboard = board_alloc(size, player);
  /* set value of board */
  for (size_t i = 0; i < nbr_of_disc; i++) {
    board_set(newboard, tab_disc[i].player, tab_disc[i].y, tab_disc[i].x);
  }
  return newboard;
}

static void print_usage() {
  printf("Usage: reversi [-s SIZE|-b [N]|-w [N]|-c|-v|-V|-h] [FILE]\n"
         "Play a reversi game with human or program players.\n"
         "  -s, --size SIZE       board size (min=1, max=5 (default=4))\n"
         "  -b, --black-ai [N]    set tactic of black player (default: 0)\n"
         "  -w, --white-ai [N]    set tactic of black player (default: 0)\n"
         "  -c, --contest         enable 'contest' mode\n"
         "  -V, --version         display version and exit\n"
         "  -h, --help            display this help and exit\n"
         "\n"
         "Tactic list: human (0), random (1)\n");
}

int main(int argc, char *argv[]) {
  const struct option longopts[] = {{"size", required_argument, NULL, 's'},
                                    {"black-ai", optional_argument, NULL, 'b'},
                                    {"white-ai", optional_argument, NULL, 'w'},
                                    {"contest", no_argument, NULL, 'c'},
                                    {"verbose", no_argument, NULL, 'v'},
                                    {"version", no_argument, NULL, 'V'},
                                    {"help", no_argument, NULL, 'h'},
                                    {NULL, 0, NULL, 0}};

  /* boolean variables */
  bool contest_mode = false;  /* true if -c option is given */
  bool file_argument = false; /* true if file argument is given */
  int tactic_b_player =
      0; /* 0 if humain plays, 1 if it is random, 2 if minimax player */
  int tactic_w_player =
      0; /* 0 if humain plays, 1 if it is random, 2 if minimax player */

  /* integer variables */
  int cpt_of_file = 0; /* number of file argument */
  int int_optarg;      /* argument of option */
  int optc;            /* current value of option */

  /* other type */
  size_t board_size_num = 4;
  FILE *file = NULL;
  char *file_name;
  static const char *opts = ":s:b::w::cvVh"; /*valid options*/
```

```
while ((optc = getopt_long(argc, argv, opts, longopts, NULL)) != -1) {
  switch (optc) {

  case 's': /* 'size' option */
    int_optarg = atoi(optarg);
    if (strlen(optarg) != 1) {
      errx(1, "the argument of -s option is too long");
    }
    if ((int_optarg < 1 || int_optarg > 5)) {
      errx(1, "the argument of -s option should be between 1 and 5");
    }
    board_size_num = int_optarg;
    break;

  case 'b':               /* 'black-ai' option */
    if (optarg != NULL) { /* if argument is given */
      int_optarg = atoi(optarg);
      if (strlen(optarg) != 1) {
        errx(1, "the argument of -b option is too long");
      }
      if (int_optarg > NBR_PLAY_FUNC - 1) {
        errx(1, "the argument of -w option should be 0 to %d",
             NBR_PLAY_FUNC - 1);
      }
      tactic_b_player = int_optarg;
    }
    break;

  case 'w':               /* 'white-ai' option */
    if (optarg != NULL) { /* if argument is given */
      int_optarg = atoi(optarg);
      if (strlen(optarg) != 1) {
        errx(1, "the argument of -w option is too long");
      }
      if (int_optarg > NBR_PLAY_FUNC - 1) {
        errx(1, "the argument of -w option should be 0 to %d",
             NBR_PLAY_FUNC - 1);
      }
      tactic_w_player = int_optarg;
    }
    break;

  case 'c': /* 'contest mode' option */
    contest_mode = true;
    break;

  case 'v': /* 'verbose' option */
    VERBOSE = true;
    break;

  case 'V':               /* 'version' option */
    printf("\033[0;32m"); /* green color text */
    printf("reversi %d.%d.%d\n", VERSION, SUBVERSION, REVISION);
    printf("\033[0;0m"); /* end green color */
    printf("This software allows to play to reversi game.\n");
    exit(EXIT_SUCCESS);
    break;

  case 'h': /* 'help' option */
    print_usage();
    exit(EXIT_SUCCESS);
    break;

  case ':': /* if argument is not given */
```



```
      errx(1, "-%c commande must have an argument", optopt);
      break;

    default: /* if an unknown option is given */
      errx(1, "Illegal option: Try 'reversi --help' for more information");
      break;
    }
  }

  /* get the argument of the program */
  for (; optind < argc; optind++) {
    file_name = argv[optind];
    file_argument = true;
    cpt_of_file++;
  }
  if (cpt_of_file > 1) { /* more than 1 file is given */
    errx(1, "You must to give 1 file for the contest mode");
  } else {
    /*********************************************/
    move_t (*blackfunc)(board_t *) = play_func[tactic_b_player];
    move_t (*whitefunc)(board_t *) = play_func[tactic_w_player];
    /*********************************************/
    if (file_argument) { /* file argument is given */
      file = fopen(file_name, "r+");
      if (file == NULL) { /* The given argument is not a readable file */
        err(errno, "%s", file_name);
        exit(EXIT_FAILURE);
      }
      if (contest_mode) { /* contest mode is enable */
        /****************** contest mode *********************/
        board_t *board = file_parser(file_name); /* read in contest file */
        game(blackfunc, whitefunc, board);
        board_free(board);
      } else { /* normal mode */
        /******************* normal mode with file ***********************/
        board_t *board = file_parser(file_name);
        game(blackfunc, whitefunc, board);
        board_free(board);
      }
      fclose(file);
    } else {              /* no file is given */
      if (contest_mode) { /* contest mode is enable */
        errx(1, "Contest mode is activated but no file is given");
      }
      /****************** normal mode without file *********************/
      board_t *board = board_init(board_size_num * 2);
      game(blackfunc, whitefunc, board);
      board_free(board);
    }
  }

  exit(EXIT_SUCCESS);
}
```

```c
#include "player.h"

/* define size of file_name string but it can be longer than 64 */
#define MAX_LENGTH_FILE_NAME 64
#define INFINITY 32767

/*****************************************************************************/
/***************************** HEURISTIC PART ********************************/
/*****************************************************************************/

/* return the difference of score between the current player and its opponent */
static int score_heuristic(board_t *board, disc_t player) {
  score_t score = board_score(board);
  if (player == BLACK_DISC) {
    return score.black - score.white;
  }
  return score.white - score.black;
}

static int score_heuristic_bis(board_t *board, disc_t player) {
  score_t score = board_score(board);
  size_t size = board_size(board);
  /* get score */
  int int_score = 0;
  if (player == BLACK_DISC) {
    int_score = score.black - score.white;
  } else {
    int_score = score.white - score.black;
  }
  /* put forward corner */
  move_t tmp_move;
  size_t tab_move[2] = {0, size - 1};
  for (size_t i = 0; i < 2; i++) {
    for (size_t j = 0; j < 2; j++) {
      tmp_move.row = tab_move[i];
      tmp_move.column = tab_move[j];
      if (board_get(board, tab_move[i], tab_move[i]) == player) {
        int_score = int_score + 5;
      }
    }
  }
  /*size_t possible_moves = board_count_player_moves(board);
  size_t possible_moves_opponent = board_count_opponent_moves(board);
  if (possible_moves < 5) {
    int_score = int_score + 5;
  }
  if (possible_moves_opponent > 5) {
    int_score = int_score + 10;
  }*/
  return int_score;
}

/*****************************************************************************/
/************************** SIMUL BEST PLAYER ********************************/
/*****************************************************************************/

move_t simul_best_player(board_t *board) {
  return simul_alpha_beta_player(board);
}

/*****************************************************************************/
/******************** MINIMAX NEGAMAX ALPHA BETA PLAYER **********************/
/*****************************************************************************/
```



```c
static int alpha_beta_bis_machine(board_t *board, size_t depth, int alpha,
                                  int beta, disc_t player) {
  disc_t current_player = board_player(board);
  /* end of game or depth */
  if (current_player == EMPTY_DISC || depth == 0) {
    return score_heuristic_bis(board, player);
  }

  if (current_player == player) {
    size_t nbr_poss_moves = board_count_player_moves(board);
    for (size_t i = 0; i < nbr_poss_moves; i++) {
      board_t *tmp_board = board_copy(board);
      move_t current_move = board_next_move(board);
      board_play(tmp_board, current_move);
      int score =
          alpha_beta_bis_machine(tmp_board, depth - 1, alpha, beta, player);
      board_free(tmp_board);
      if (score > alpha) {
        alpha = score;
        if (alpha >= beta) {
          break;
        }
      }
    }
    return alpha;
  } else {
    size_t nbr_poss_moves = board_count_player_moves(board);
    for (size_t i = 0; i < nbr_poss_moves; i++) {
      board_t *tmp_board = board_copy(board);
      move_t current_move = board_next_move(board);
      board_play(tmp_board, current_move);
      int score =
          alpha_beta_bis_machine(tmp_board, depth - 1, alpha, beta, player);
      board_free(tmp_board);
      if (score < beta) {
        beta = score;
        if (alpha >= beta) {
          break;
        }
      }
    }
    return beta;
  }
}

static move_t alpha_beta_bis_player(board_t *board, size_t depth) {
  disc_t current_player = board_player(board);
  int best_score = -INFINITY;
  move_t best_move;
  size_t nbr_poss_moves = board_count_player_moves(board);
  for (size_t i = 0; i < nbr_poss_moves; i++) {

    board_t *tmp_board = board_copy(board);
    move_t current_move = board_next_move(board);
    if (i == 0) {
      best_move = current_move;
    }
    board_play(tmp_board, current_move);
    int score = alpha_beta_bis_machine(tmp_board, depth - 1, -INFINITY,
                                       INFINITY, current_player);
    if (score >= best_score) {
      best_score = score;
      best_move = current_move;
    }
```

```
    board_free(tmp_board);
  }
  return best_move;
}

move_t simul_alpha_beta_bis_player(board_t *board) {
  return alpha_beta_bis_player(board, DEPTH_ALPHABETA_BIS);
}

/*****************************************************************************/
/************************ MINIMAX ALPHA BETA PLAYER **************************/
/*****************************************************************************/

static int alpha_beta_machine(board_t *board, size_t depth, int alpha, int beta,
                              disc_t player) {
  disc_t current_player = board_player(board);
  if (current_player == EMPTY_DISC || depth == 0) {
    return score_heuristic(board, player);
  }
  int best_score;
  if (current_player == player) {
    size_t nbr_poss_moves = board_count_player_moves(board);
    for (size_t i = 0; i < nbr_poss_moves; i++) {
      board_t *tmp_board = board_copy(board);
      move_t current_move = board_next_move(board);
      board_play(tmp_board, current_move);
      int score = alpha_beta_machine(tmp_board, depth - 1, alpha, beta, player);
      board_free(tmp_board);
      if (score > alpha) {
        alpha = score;
        if (alpha >= beta) {
          break;
        }
      }
    }
    return alpha;
  } else {
    size_t nbr_poss_moves = board_count_player_moves(board);
    for (size_t i = 0; i < nbr_poss_moves; i++) {
      board_t *tmp_board = board_copy(board);
      move_t current_move = board_next_move(board);
      board_play(tmp_board, current_move);
      int score = alpha_beta_machine(tmp_board, depth - 1, alpha, beta, player);
      board_free(tmp_board);
      if (score < beta) {
        beta = score;
        if (alpha >= beta) {
          break;
        }
      }
    }
    return beta;
  }
}

static move_t alpha_beta_player(board_t *board, size_t depth) {
  disc_t current_player = board_player(board);
  int best_score = -INFINITY;
  move_t best_move;
  size_t nbr_poss_moves = board_count_player_moves(board);
  for (size_t i = 0; i < nbr_poss_moves; i++) {

    board_t *tmp_board = board_copy(board);
    move_t current_move = board_next_move(board);
```



```
    if (i == 0) {
      best_move = current_move;
    }
    board_play(tmp_board, current_move);
    int score = alpha_beta_machine(tmp_board, depth - 1, -INFINITY, INFINITY,
                                   current_player);
    board_free(tmp_board);
    if (score > best_score) {
      best_score = score;
      best_move = current_move;
    }
  }
  return best_move;
}

move_t simul_alpha_beta_player(board_t *board) {
  return alpha_beta_player(board, DEPTH_ALPHABETA);
}

/*****************************************************************************/
/************************** MINIMAX PLAYER PART ******************************/
/*****************************************************************************/

static int minimax_machine(board_t *board, size_t depth, disc_t player) {
  disc_t current_player = board_player(board);
  if (current_player == EMPTY_DISC || depth == 0) {
    return score_heuristic(board, player);
  }
  int best_score;
  if (current_player == player) {
    best_score = -INFINITY;
    size_t nbr_poss_moves = board_count_player_moves(board);
    for (size_t i = 0; i < nbr_poss_moves; i++) {
      board_t *tmp_board = board_copy(board);
      move_t current_move = board_next_move(board);
      board_play(tmp_board, current_move);
      int score = minimax_machine(tmp_board, depth - 1, player);
      if (score > best_score) {
        best_score = score;
      }
      board_free(tmp_board);
    }
  } else {
    best_score = INFINITY;
    size_t nbr_poss_moves = board_count_player_moves(board);
    for (size_t i = 0; i < nbr_poss_moves; i++) {
      board_t *tmp_board = board_copy(board);
      move_t current_move = board_next_move(board);
      board_play(tmp_board, current_move);
      int score = minimax_machine(tmp_board, depth - 1, player);
      if (score < best_score) {
        best_score = score;
      }
      board_free(tmp_board);
    }
  }
  return best_score;
}

static move_t minimax_player(board_t *board, size_t depth) {
  disc_t current_player = board_player(board);
  int best_score = -INFINITY;
  move_t best_move;
```

```
  size_t nbr_poss_moves = board_count_player_moves(board);
  for (size_t i = 0; i < nbr_poss_moves; i++) {

    board_t *tmp_board = board_copy(board);
    move_t current_move = board_next_move(board);
    if (i == 0) {
      best_move = current_move;
    }
    board_play(tmp_board, current_move);
    int score = minimax_machine(tmp_board, depth - 1, current_player);
    if (score > best_score) {
      best_score = score;
      best_move = current_move;
    }
    board_free(tmp_board);
  }
  return best_move;
}

move_t simul_minimax_player(board_t *board) {
  return minimax_player(board, DEPTH_MINIMAX);
}

/*****************************************************************************/
/*************************** RANDOM PLAYER PART ******************************/
/*****************************************************************************/

static void rand_init(void) {
  static bool isinitialized = false;
  if (!isinitialized) {
    srandom(time(NULL) - getpid());
    isinitialized = true;
  }
}

move_t random_player(board_t *board) {
  /* nbr_poss_moves > 0 if random_player is run */
  size_t nbr_poss_moves = board_count_player_moves(board);
  rand_init();
  size_t r = (size_t)(random() % nbr_poss_moves);
  for (size_t i = 0; i < r; i++) {
    board_next_move(board);
  }
  return board_next_move(board);
}

/*****************************************************************************/
/*************************** GAME SAVE PART **********************************/
/*****************************************************************************/

static void clean_buffer() {
  int c = 0;
  while (c != '\n') {
    c = getchar();
  }
}

/* save the content of the board in a file given by the user */
static void game_save(board_t *board) {
  char *file_name = calloc(MAX_LENGTH_FILE_NAME, sizeof(char));
  if (file_name == NULL) {
    return;
  }
  fprintf(stdout, "Give a filename to save the game (default: 'board.txt'):");
```



```
  /** get file name section **/
  char current_char = getchar();
  if (current_char == '\n') { /* file name is not given */
    char *init_file_name = "board.txt";
    for (size_t i = 0; i < 9; i++) {
      file_name[i] = init_file_name[i];
    }
  } else {
    size_t i = 0;
    size_t nbr_out = 0;
    while (current_char != '\n') {
      if (current_char != ' ') {
        file_name[i] = current_char;
        i++;
        if (i == MAX_LENGTH_FILE_NAME) {
          nbr_out++;
          char *new_file_name = calloc(MAX_LENGTH_FILE_NAME, sizeof(char));
          if (new_file_name == NULL) {
            return;
          }
          new_file_name = file_name;
          free(new_file_name);
          file_name = calloc(MAX_LENGTH_FILE_NAME + nbr_out, sizeof(char));
          if (file_name == NULL) {
            return;
          }
          file_name = new_file_name;
        }
      }
      current_char = getchar();
    }
  }

  /** writing section **/
  fprintf(stdout, "creating file with name %s\n", file_name);
  FILE *file =
      fopen(file_name, "w+"); /* create this file if no already exist */
  if (file == NULL) {
    return;
  }
  fprintf(file, "%c\n", board_player(board));
  size_t size = board_size(board);
  disc_t current_disc;
  for (size_t i = 0; i < size; i++) {
    for (size_t j = 0; j < size; j++) {
      current_disc = board_get(board, i, j);
      if (current_disc == HINT_DISC) {
        current_disc = EMPTY_DISC;
      }
      fprintf(file, "%c ", current_disc);
    }
    fprintf(file, "\n");
  }
  fclose(file);
  free(file_name);
}

/*****************************************************************************/
/************************** HUMAN PLAYER PART ********************************/
/*****************************************************************************/

/* return size if char of column is incorrect */
static size_t get_column(char current_char, size_t size) {
```

```
  int alpha_maj[10] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'};
  int alpha_min[10] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'};
  for (size_t i = 0; i < 10; i++) {
    if (current_char == alpha_maj[i] || current_char == alpha_min[i]) {
      return i;
    }
  }
  return size; /* incorrect column */
}

/* return size,x or x,size if incorrect move is given and size+1,size+1 if
 * quit
 */
static move_t get_move(size_t size) {
  move_t move;
  char current_char = getchar(); /* get the first char */
  if (current_char == 'q' || current_char == 'Q') {
    clean_buffer();
    move.row = size + 1;
    move.column = size + 1;
    return move;
  }
  move.column = get_column(current_char, size);
  char char_number[2]; /* column number */
  for (size_t i = 0; current_char != '\n'; i++) {
    current_char = getchar();
    if (i < 2) {
      char_number[i] = current_char;
    }
  }
  move.row = atoi(char_number);
  if (move.row == 0 ||
      move.row > size) { /* if a10 is given, 10!=0 so move.row=9 error ! */
    move.row = size;
  } else {
    move.row--;
  }
  return move;
}

move_t human_player(board_t *board) {
  board_print(board, stdout);      /* print the board */
  size_t size = board_size(board); /* get size of board */
  bool error_case = true;
  move_t move;
  move.row = size;
  move.column = size;
  while (error_case) {
    fprintf(stdout,
            "Give your move (e. g. 'A5' or 'a5'), press 'q' or 'Q' to quit:");
    move = get_move(size);
    if (move.row == size + 1) { /* if q or Q is given */
      fprintf(stdout, "Quitting, do you want to save this game ? (y/N):");
      char current_char = getchar();
      if (current_char == 'y' || current_char == 'Y') {
        clean_buffer();
        game_save(board);
      }
      board_set_player(board, EMPTY_DISC);
      move.row = size;
      move.column = size;
      return move;
    }
    if (move.row == size) { /* row or colum is invalid */
```



```
      error_case = true;
    } else {
      if (move.row >= size) {
        error_case = true;
        fprintf(stdout, "Row out of bounds. ");
      } else {
        error_case = false;
      }
      if (move.column >= size) {
        error_case = true;
        fprintf(stdout, "Column out of bounds. ");
      } else {
        error_case = false;
      }
      if (!error_case && !board_is_move_valid(board, move)) {
        error_case = true;
        fprintf(stdout, "This move is invalid. ");
      } else {
        error_case = false;
      }
    }
    if (error_case) {
      fprintf(stdout, "Wrong input, try again !\n\n");
    }
  }
  return move;
}
```

```c
#ifndef PLAYER_H
#define PLAYER_H

#include <board.h>

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>

/* A player function 'move_t (*player_func) (board_t *)' returns a
 * chosen move depending on the given board.
 * return value between [0,size-1]
 * return (size,size) if no possible move exist */

/* return a move given by the user through stdin */
move_t human_player(board_t *board);

/* return a random move among the possible ones */
move_t random_player(board_t *board);

move_t simul_best_player(board_t *board);

move_t simul_minimax_player(board_t *board);
#define DEPTH_MINIMAX 1

move_t simul_alpha_beta_player(board_t *board);
#define DEPTH_ALPHABETA 4

move_t simul_alpha_beta_bis_player(board_t *board);
#define DEPTH_ALPHABETA_BIS 3

move_t priority_borders(board_t *board);

#endif /* PLAYER_H */
```



```c
#ifndef BOARD_H
#define BOARD_H

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

/* Min/Max with board */
#define MIN_BOARD_SIZE 2
#define MAX_BOARD_SIZE 10

/* Size of shift array */
#define SIZE_SHIFT_ARRAY 8

/* Popcount masks */
#define MASK1_POPCOUNT                                                          \
  (bitboard_t)((bitboard_t)0x5555555555555555 << 64) | 0x5555555555555555)
#define MASK2_POPCOUNT                                                          \
  (bitboard_t)((bitboard_t)0x3333333333333333 << 64) | 0x3333333333333333)
#define MASK3_POPCOUNT                                                          \
  (bitboard_t)((bitboard_t)0x0F0F0F0F0F0F0F0F << 64) | 0x0F0F0F0F0F0F0F0F)
#define MASK4_POPCOUNT                                                          \
  (bitboard_t)((bitboard_t)0x0101010101010101 << 64) | 0x0101010101010101)

/* Shifts masks */
#define MASK_BITB_BOTTOM_6 (bitboard_t)((bitboard_t)0x3F << 36))
#define MASK_BITB_BOTTOM_8 (bitboard_t)((bitboard_t)0xFF << 64))
#define MASK_BITB_BOTTOM_10 (bitboard_t)((bitboard_t)0x3FF << 100))
#define MASK_BITB_RIGHT_10                                                      \
  (bitboard_t)((bitboard_t)0x2008020080200 << 50) | 0x2008020080200)
#define MASK_BITB_LEFT_10                                                       \
  (bitboard_t)((bitboard_t)0x10040100401 << 50) | 0x10040100401)

/* Board discs */
typedef enum {
  BLACK_DISC = 'X',
  WHITE_DISC = 'O',
  EMPTY_DISC = '_',
  HINT_DISC = '*'
} disc_t;

/* A move in the reversi game */
typedef struct {
  size_t row;
  size_t column;
} move_t;

/* Store the score of game */
typedef struct {
  unsigned short black;
  unsigned short white;
} score_t;

/* Reversi board (forwards declaration to hide the implementation) */
typedef struct board_t board_t;
/* Base bitboard type */
typedef unsigned __int128 bitboard_t;

/** board_t type **/
/* allocate memory needed to creat a board of size 'size'
 * EXIT_FAILURE if something wrong happened */
board_t *board_alloc(const size_t size, const disc_t player);
/* init all the squares of the board as a starting game */
board_t *board_init(const size_t size);
```

```c
/* perform a deep copy of the board structure */
board_t *board_copy(const board_t *board);

/** boolean type **/
/* check if a move is valid */
bool board_is_move_valid(const board_t *board, const move_t move);
/* apply a move according to rules and set the board for next move */
bool board_play(board_t *board, const move_t move);

/** disc_t type **/
/* get current player */
disc_t board_player(const board_t *board);
/* get the content of the square */
disc_t board_get(const board_t *board, const size_t row, const size_t column);

/** int type **/
/* write on the file 'fd' the content of the given board */
int board_print(const board_t *board, FILE *fd);

/** move_t type **/
/* store the next possible move
 * fail : return -1,-1 */
move_t board_next_move(board_t *board);

/** score_t type **/
/* return score of the given board */
score_t board_score(const board_t *board);

/** size_t type **/
/* count the number of possible moves */
size_t board_count_player_moves(board_t *board);
/* count the number of opponent's possible moves */
size_t board_count_opponent_moves(board_t *board);
/* return size of the board */
size_t board_size(const board_t *board);

/** void type **/
/* set the current player */
void board_set_player(board_t *board, disc_t new_player);
/* free memory allocated to hold the board */
void board_free(board_t *board);
/* set the given disc at the given position */
void board_set(board_t *board, const disc_t disc, const size_t row,
               const size_t column);

#endif /* BOARD_H */
```



```
===============[ reversi ]===============

----------( File Hierarchy )----------

* Check requested files
Expected paths: ['Makefile', 'include/board.h', 'include/player.h', 'src/Makefil
e', 'src/board.c', 'src/player.c', 'src/reversi.c', 'src/reversi.h']
Result: *Passed*


----------( Build System )----------

* Check 'make'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check requested files
Expected paths: ['reversi', 'src/reversi']
Result: *Passed*

* Check Check if re-'make' avoid rebuild
Expected result: C compiler must not be called again
Result: *Passed*

* Check 'make clean'
Expected result: clean target must remove all built files
Result: *Passed*

* Check 'make help'
Expected result: EXIT_SUCCESS
Result: *Passed*

----------( Option Parser )----------

* Check './reversi' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -x' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi -v' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --verbose' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -V' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --version' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -h' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --help' with stdin='Q\n\n'
```

```
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --he' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -b' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --black-ai' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -b0' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --black-ai=0' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -b1' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --black-ai=1' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -w' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --white-ai' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -w0' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --white-ai=0' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -w1' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --white-ai=1' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -b -w' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --black-ai --white-ai' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -b0 -w1' with stdin='Q\n\n'
```



```
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --black-ai=0 --white-ai=1' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -s' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi --size' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi -s 1' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *****Failed!*****
Output:
+ Return code: -8 (Floating point exception (SIGFPE))

* Check './reversi --size 1' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *****Failed!*****
Output:
+ Return code: -8 (Floating point exception (SIGFPE))

* Check './reversi -s 2' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --size 2' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -s 3' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --size 3' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -s 4' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --size 4' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -s 5' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --size 5' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -s 0' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi --size 0' with stdin='Q\n\n'
```

```
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi -s 6' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi --size 6' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi -b -s 1' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *****Failed!*****
Output:
+ Return code: -8 (Floating point exception (SIGFPE))

* Check './reversi --black-ai --size 1' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *****Failed!*****
Output:
+ Return code: -8 (Floating point exception (SIGFPE))

* Check './reversi -w -s 1' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *****Failed!*****
Output:
+ Return code: -8 (Floating point exception (SIGFPE))

* Check './reversi --white-ai --size 1' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *****Failed!*****
Output:
+ Return code: -8 (Floating point exception (SIGFPE))

----------( Argument Parser )----------

* Check './reversi board.txt' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -v board.txt' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --verbose board.txt' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -c' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi --contest' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

Expected result: EXIT_FAILURE or return code == 2
* Check './reversi -c missing.txt' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

Expected result: EXIT_FAILURE or return code == 2
* Check './reversi --contest missing.txt' with stdin='Q\n\n'
```



```
Expected result: EXIT_FAILURE
Result: *Passed*

Expected result: EXIT_FAILURE or return code == 13
* Check './reversi -c /etc/shadow' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

Expected result: EXIT_FAILURE or return code == 13
* Check './reversi --contest /etc/shadow' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi -c board.txt' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --contest board.txt' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -c -v board.txt' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi --contest --verbose board.txt' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*


----------( Board Parser )----------

* Check './reversi parser/board-0x0.fail' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-1x1.fail' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-2x2.pass' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi parser/board-3x3.fail' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-4x4.pass' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi parser/board-5x5.fail' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-6x6.pass' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi parser/board-7x7.fail' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*
```

```
* Check './reversi parser/board-8x8.pass' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi parser/board-9x9.fail' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-10x10.pass' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi parser/board-11x11.fail' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-12x12.fail' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-almost_full_board.pass' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi parser/board-current_player_get_two_chars.fail' with stdin='Q
\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-empty_board.pass' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi parser/board-empty_stone_as_current_player.fail' with stdin='
Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-eof_after_current_player.fail' with stdin='Q\n\n
'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-eof_before_end_of_the_board.fail' with stdin='Q\
n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-extra_empty_lines.pass' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi parser/board-extra_spaces_around_chars.pass' with stdin='Q\n\
n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi parser/board-first_line_overflow.fail' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-full_board.pass' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*
```



```
* Check './reversi parser/board-impossible_board-01.pass' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi parser/board-impossible_board-02.pass' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi parser/board-line_too_long.fail' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-line_too_short.fail' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-line_too_short_with_comment.fail' with stdin='Q\
n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-line_too_short_with_no_newline.fail' with stdin=
'Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-long_line_filled_with_spaces.pass' with stdin='Q
\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi parser/board-missing_board.fail' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-missing_current_player.fail' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-missing_newline_after_current_player.pass' with
stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi parser/board-no_final_newline.pass' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi parser/board-stop_at_first_line_without_newline.fail' with st
din='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-too_few_lines.fail' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-too_few_lines_with_comment.fail' with stdin='Q\n
\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-too_many_lines.fail' with stdin='Q\n\n'
```

```
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-with_comments.pass' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi parser/board-wrong_character.fail' with stdin='Q\n\n'
Expected result: EXIT_FAILURE
Result: *Passed*

* Check './reversi parser/board-wrong_current_player_char.fail' with stdin='Q\n\
n'
Expected result: EXIT_FAILURE
Result: *Passed*


----------( Checking Memory Usage )----------

* Check 'valgrind ./reversi -h'
Result: *Passed*: No memory leak detected
Result: *Passed*: No memory corruption detected

 Memcheck, a memory error detector
 Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
 Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
 Command: ./reversi -h
 Parent PID: 20668


 HEAP SUMMARY:
     in use at exit: 0 bytes in 0 blocks
   total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated

 All heap blocks were freed -- no leaks are possible

 For lists of detected and suppressed errors, rerun with: -s
 ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

* Check 'valgrind ./reversi parser/board-with_comments.pass' with stdin='q\n\n'
Result: *Passed*: No memory leak detected
Result: *Passed*: No memory corruption detected

 Memcheck, a memory error detector
 Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
 Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
 Command: ./reversi parser/board-with_comments.pass
 Parent PID: 20668


 HEAP SUMMARY:
     in use at exit: 0 bytes in 0 blocks
   total heap usage: 6 allocs, 6 frees, 13,472 bytes allocated

 All heap blocks were freed -- no leaks are possible

 For lists of detected and suppressed errors, rerun with: -s
 ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

* Check 'valgrind ./reversi' with stdin='c4\nc3\nd3\nc5\nb3\ne3\nf3\na2\nf5\ng3\
ne2\ne1\nb5\na6\na5\na4\nd2\ng5\nf1\nd1\nc1\nq\ny\n\n'
Result: *Passed*: No memory leak detected
Result: *Passed*: No memory corruption detected
```



```
 Memcheck, a memory error detector
 Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
 Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
 Command: ./reversi
 Parent PID: 20668


 HEAP SUMMARY:
     in use at exit: 0 bytes in 0 blocks
   total heap usage: 6 allocs, 6 frees, 12,984 bytes allocated

 All heap blocks were freed -- no leaks are possible

 For lists of detected and suppressed errors, rerun with: -s
 ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)


----------( Board Module )----------

* Check './board_tests' unit test module
Result: *Passed*: All unit tests passed!

* Check 'valgrind ./board_tests'
Result: *Passed*: No memory leak detected
Result: *Passed*: No memory corruption detected

 Memcheck, a memory error detector
 Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
 Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
 Command: ./board_tests
 Parent PID: 20668


 HEAP SUMMARY:
     in use at exit: 0 bytes in 0 blocks
   total heap usage: 20 allocs, 20 frees, 10,104 bytes allocated

 All heap blocks were freed -- no leaks are possible

 For lists of detected and suppressed errors, rerun with: -s
 ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)



----------( Program User Interface )----------

* Check './reversi' with stdin='q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi' with stdin='Q\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi' with stdin='q\ny\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi' with stdin='q\ny\nmy_board.txt\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi' with stdin='q\nY\n\n'
Expected result: EXIT_SUCCESS
```

```
Result: *Passed*

* Check './reversi -s 1' with stdin='q\nY\n\n'
Expected result: EXIT_SUCCESS
Result: *****Failed!*****
Output:
+ Return code: -8 (Floating point exception (SIGFPE))

* Check './reversi -s 2' with stdin='q\nY\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -s 3' with stdin='q\nY\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -s 4' with stdin='q\nY\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi -s 5' with stdin='q\nY\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi' with stdin='d0\nq\ny\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi' with stdin='aaaaaaaaaaaaaaaaaaaaa\nq\ny\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi' with stdin='   a    a  \nq\ny\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi' with stdin='0d\nq\ny\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi' with stdin='00\nq\ny\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi' with stdin=' 0    0  \nq\ny\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi' with stdin='   d    2\ne   2   \nq\ny\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi' with stdin='
```



```



                                d2\nq\ny\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi' with stdin='aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\nq\ny\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi ../Utils/game_engine/boards/board-only_one_possible-input' wi
th stdin='d5\nq\ny\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi' with stdin='c4\nq\ny\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi' with stdin='c4\nc3\nd3\nc5\nb3\ne3\nf3\na2\nf5\ng3\ne2\ne1\n
b5\na6\na5\na4\nd2\ng5\nf1\nd1\nc1\nq\ny\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*

* Check './reversi ../Utils/game_engine/boards/board-skip_player-input' with std
in='d6\nq\ny\n\n'
Expected result: EXIT_SUCCESS
Result: *Passed*


----------( Checking Contest Mode )----------

* Check './reversi -c ../Utils/game_engine/boards/board-only_one_possible-input'
Expected result: EXIT_SUCCESS
Result: *****Failed!*****
Output:
+ Return code: 124 (Timeout (ETIME))

* Check 'valgrind ./reversi -c ../Utils/game_engine/boards/board-8x8'
Result: ***Failed!***: Memory leak(s) detected!
```

```
Result: *Passed*: No memory corruption detected

 Memcheck, a memory error detector
 Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
 Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
 Command: ./reversi -c ../Utils/game_engine/boards/board-8x8
 Parent PID: 20668


* Check 'valgrind ./reversi -c ../Utils/game_engine/boards/board-only_one_possib
le-input'
Result: ***Failed!***: Memory leak(s) detected!
Result: *Passed*: No memory corruption detected

 Memcheck, a memory error detector
 Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
 Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
 Command: ./reversi -c ../Utils/game_engine/boards/board-only_one_possible-input
 Parent PID: 20668



----------( Final result )----------
Passed: 247; Failed: 10
Time elapsed: 633.46s
```