AGH – UNIVERSITY OF SCIENCE AND TECHNOLOGY

Project documentation for

Othello - game

Object-oriented programming languages

Electronics and Telecommunication EN, III year

Gregório da Luz

lecturer: Rafał Frączek

16/02/2021

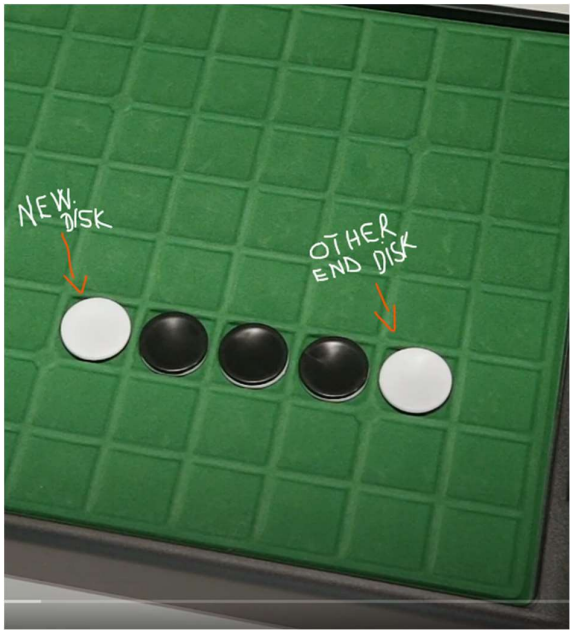
1. Project description

My project is a game called Othello. It was designed to be played using command prompt/terminal interface. The board is printed and the players should pick the position they would like to place their disks. The purpose of the project is to give the chance for 2 people to have fun playing this game. Othello can be very challenging at a times, so it is not just about the fun but also about logic training.

2. User's manual

Once the program is running, you should not have any difficulties about any special start-up action. One person should be the black disks and the other one the white ones. It is not a very widely known game, so it is important to lay out the rules. Here they are:

- The game starts with 2 white and 2 black disks diagonally placed on the centre of the table;
- The board is composed of 64 squares (8x8) where it is possible to place the disks;
- Each player starts with 28 disks to place on the board during the game;
- Black disks always have the first move;
- At each turn, a player can place a disk on the board. The placed disk must be on the side of one of the opponent's disk and there must always be a disk of the player that is playing at the end of that direction. This way, all the opponent's disks between the new placed disk and the one already on the board must be flipped. An Image can illustrate such a situation:



In this case, the black disks must be flipped to the white colour;

- If a player has more than one direction (diagonal, horizontally or vertically) with a similar situation to the one described on the image above, those disks are flipped as well. However, you cannot place more than one disk on the board at a turn;
- If a player has no place on the board that would make possible for him to flip at least one disk, he loses his turn;
- If a player has no more disks but still could place a disk and flip at least one disk, the opponent must give him one of his disks to the other player;

- If both players cannot flip any disks anymore, the white and black disks on the board are counted and the one that has more disks on the board is the winner;
- When all the disks are placed on the board the white and black disks are counted and the one with more disks is the winner;
- When the number of disks are equal, it is a draw;

3. Compilation

Opening the project using an IDE like Visual Studios or CodeBlocks, you either have to install an extension to develop in C++ either things are already set. Build and running the program is very straight forward using IDE. In case you would like to use command prompt or Linux terminal, it's a bit different. For Windows you could install MinGW which will allow to compile the C++ code using GCC in the command prompt. For Linux GCC is normally already present. Standard Building will be enough to run the program.

4. Source files

The project consists of the following source files:

- *Game.h*, *Game.cpp* – declaration and definition for `Game` class member functions and variables,
- *Player.h*, *Player.cpp* – declaration and definition of variables and functions each player needs;
- *main.cpp* – in this file, using the 2 classes previously mentioned, we have our game set;

5. Dependencies

No external library was used.

6. Class description

In the project, the following classes were created:

- `Game` – this class has all the functions necessary for the game logic to work.
  - `void InitGame()` – function used inside the constructor as an initializer and also called when the players want to play again after the game ends;
  - `void Info() const` – It prints out whose turn is;
  - `void printBoard()const` – It prints the current state of the board;
  - `bool goodInput(int x_value, int y_value)const` – It checks if the player has entered a valid input;
  - `bool freeSlot(int x, int y)const` – It checks if the slot is free on the board;
  - `void allowedSlotLoop(int x_, int y_, int i_c, int j_c, int lim_i, int lim_j)` – It contains the loop that is used many times inside the allowedSlot();
  - `bool allowedSlot(int x, int y)` – It checks if the slot chosen has an opponent's disk close to it;
  - `int flipAllDirections(int x, int y, int i_c, int j_c)` – It is called by the allowedSlotLoop() and it checks if the opponent's disk can be flipped. If yes, it flips it. It is important to mention that this function DOES NOT flip when it is used by the allowedTurn(). More information about allowedTurn() is presented further in the documentation;
  - `void flipWhoseTurnAndOpponent()` – It flips the variables m_whose_turn and m_opponent preparing for the new turn of the game;

- `void settingNewDisk(int x, int y)` – Once all the checks are done, it places the new disk on the board;
- `bool allowedTurn()` – It makes use of the allowedSlot() to check if the next player has any possible that move he can execute;
- `bool noPossibleMove()` – It makes use of the allowedTurn() to see if the game ends because of lack of possible moves from both players;
- `void theWinnerIs()` – It counts the disks present on the board, checks who is the winner and print it to the screen;
- `void playAgain()` – It asks the players if they would like to play again. If yes, it calls the InitGame() to reset the board and the variables;
- `void setEnd(bool end);`
- `char getWhoseTurn() const;`
- `int getDiskCounter() const;`
- `bool getEnd() const;`

- `Player` – a class that contains functions and variables concerning a player.
  - `void oneLessDisk()` – It decreases by one the number of disks of a player has after he has played;
  - `void giveMeOneDisk(Player& giving)` – It gives one disk to the player that calls the function and decreases one disk from the "Player giving";
  - `void resetDisks()` – It resets the number of disks to 28 when a new game starts;
  - `int getDisksAvailable() const;`
  - `int getColor() const;`

7. Resources

None.

8. Future development

The most important points regarding a future development are:

1 – "Translate" the game to a GUI event-based program;

2 – Keep track of how many times a player has won;

3 – Build an AI for a single player version;

4 – Keep a small database with players/users with the number of wins, losses and draws. Possibly, also keeping track of against whom they played and the score of the matches. To have such an option, a user and a password should be demanded from the players upon starting the game;

9. Other

The codes are available on the GitHub repository bellow:

gregorio1212/Othello_Cpp: Othello game in C++ (github.com)

In the next days, I will be developing a GUI version of the game that will be more appealing to the eyes. I would also like to include a tracker of wins and losses from each player.