Project Caro Report : Nguyễn Lê Minh (18125040) class APCS1

It is recommended to read this report **backward** (from III, then II, to I).

Features:

1. A splash screen and about us screen, with animation
2. Variable board size: 10x10, 15x15 and 20x20.
3. Can undo the mistake move.
4. Have four slots data to save / load game.
5. About the rule sets, it could be Gomoku (five-in-a-row) or Caro (five-in-a-row which is not blocked both sides). It haves "place-stone mode", where each players have 3 initial stones to place on the board (placing a stone don't lose a turn). It can predetermine whether you can undo mistakes in this game or not. So, there are 8 combinations of rule.
6. Statistic of PvP and PvC modes .
7. Poor computer with 3 difficulties: Easy, Normal, Hard

I. Project structure: the source files and what each function do in each file

1. Header: "*menuScreen.h*", implemented in "*MenuScreen.cpp*" :

Contains every general method to draw the needed screen.

```
void SplashScreen ();
```

Purpose: Display the splash screen and about screen.

```
int MainMenuScreen (int optionCursor);
```

Purpose: Display the main menu screen and handle the keyboard input until user choose one of the items. User will use arrow keys and number keys to choose an item and Enter to confirm.

The function returns the id of the selected item, i.e the item which the cursor is choosing at the time the user presses Enter.

The optionCursor will change the initial selected item upon drawing. This will help user to improve their expericence, i.e they want to come back to previous choice quickly.

```
void StatisticScreen ();
```

Purpose: Display the statistic screen.

```
bool DataScreen (bool EnableSave);
```

Purpose: Display the data screen for saving current game (only if calls from a playing game) and loading previous saved game. It handles the input and do saving and loading until the program go out of the scope of data screen.

It returns true if a new game is loaded and return false otherwise. It notices the caller to reload the necessary game graphics.

Sole parameter meaning: whether the screen in the save/load mode or it is just loading (no saving).

```
void OptionScreen ();
```

Purpose: Display & handle input in the scope of option screen.

```
void PlayScreen (bool IsANewGame);
```

Purpose: Display the usual interactive playing gomoku / caro between two players. Not much info is passed because of the design (see III) relies heavily on global variables.

It takes one parameter, to know whether the user needs to play a new game or load saved one.

```
void Stat_Load ();
```

Purpose: Read the statistic file saved from disk. This should be called before the main loop of the program.

```
void Stat_Save ();
```

Purpose: Write the current statistic into disk, for future loading. This should be called after the main loop of the program, i.e after the user wants to exit the program.

```
void Data_InitializeSlots ();
```

Purpose: Read all the saved games from disk. This should be called before the main loop of the program.

```
void Data_FinalizeSlots ();
```

Purpose: Write all saved games to disk. This should be called after main loop of the program.

```
void Option_InitializeOption (const char OptionFile[]);
```

Purpose: Read previous configuration from disk. This configuration could be changed in Option screen.

```
void Option_FinalizeOption (const char OptionFile[]);
```

Purpose: Write current configuration to disk.

2. Header: "`basicdraw.h`", implements in `basicdraw.cpp`

Purpose: Define the value of colors (for calling purpose) and some basic drawing function.

```
void GotoXY (short x, short y);
```

Move the console cursor to (x,y)

```
void SetColor (short BackgroundColor, short ForegroundColor);
```

Set the current drawing color.

```
void PutChar (char c);
```
Put one character to the screen at the console cursor.

```
void PutString (const char* s);
```
Put one string to the screen.

```
void PutFilledRectangle (short x1, short y1, short x2, short y2, char
c);
```
Put a filled rectangle with character c at top left of (x1, y1) and bottom right of (x2, y2).

```
void PutBorderedRectangle (short x1, short y1, short x2, short y2,
                           char TopLeftCorner='/',
                           char TopRightCorner='\\',
                           char BottomLeftCorner='\\',
                           char BottomRightCorner='/',
                           char TopSide='-',
                           char LeftSide='|',
                           char RightSide='|',
                           char BottomSide='-');
```
Put a bordered rectangle with the characters made the four sides and four corners in parameters at top left of (x1, y1) and bottom right of (x2, y2).

3. Header: "`dataStruct.h`", no implementation

Define any data structure that is not primitives (int, char, float...), such as structs and vectors.

4. Header: "`globalVars.h`", real definition in "`globalVars.cpp`"

Declares every global variables. As in (III), the design relies heavily on global variables.

5. Header: "`logic.h`", implements in "`logic.cpp`"

Any functions related to the logic of Gomoku/ Caro is put here. These includes:

```
bool isLegalMove (AMove Move);
```
Check if a move is legal.

```
bool isWinningMove (AMove Move);
```
Check if a move is a winning move.

```
void NextMove (int ScreenMode, short &x, short &y);
```
Tells the computer to give their next move, in PvC mode.
First parameter is the difficulty, 1 for Easy, 2 for Normal and 3 for Hard.
Second and third parameters to store the answer of computer.

```
void Expand (short x, short y, CoordList &The_List, short radius);
```

Tells the computer that "Look! I just make a move at (x,y), and you should consider the coorinates around it by a certain radius, add them to a list for future thinking."

6. Header "`screenutil.h`", implements in "`screenutil.cpp`"

Utility to work with screen: adjust the screen size and clear the screen with selected drawing color (which could be set in SetColor() of "`basicdraw.h`")

```
void AdjustScreenSize (short x, short y);
```

Adjust screen size, such that the new screen have width of x and height of y.

```
void ClearScreen ();
```

Clear the screen.

7. Header "`csDes_sup.h`", implements in "`csDes_sup.cpp`"

Supports reading and drawing images created "Console Designer", our another project. See (IV [1])

```
void ReadCsDesImage (const char * fileName, int & width, int & height,
                short background[], short foreground[], char ascii[]);
```

Read an "ConsoleDesigner" image from the disk. Its file name is given in first parameter. The rest is for storing the image. The size of three last parameters should be at least width*height, enough to store the image.

```
void PutCsDesImage (short x, short y, int width, int height, short
                background[], short foreground[], char ascii[]);
```

Draw an "ConsoleDesigner" image to the disk at the coordinate (x,y) given by first two parameters.

8. Remarks: Only "public" functions are listed, i.e those functions that you should "see" when including the respective header.

Finally, a "`main.cpp`" contains the main loop of program.

II. Data Structure

1. How was the board represented

The column (x coordinate) numbered from 0 to `Board_Logical_Size-1`

The row (y coordinate) numbered from 0 to `Board_Logical_Size-1`

Each cell (x,y) is stored as a character, '1' means player 1 moves there, '2' means player 2 moves there, 'S' mean a stone placed there, and '.' means it is a blank cell.

The board is represented as a global array of char named Board, the (x,y) cell is stored at `Board[y*20+x]`

2. How was a move represented

There is a struct "`AMove`" in "`dataStruct.h`", with their members' meaning:

`bool` `TurnLost`: is it really a move or just "placing a stone" (don't lose turn)

`short` `maker`: equals 1 or 2, indicates who makes that move.

`short` `x,y` : the two column and row coordinates of the move, respectively.

A "`MoveList`" is a `std::vector` of struct "`AMove`".

3. How was a game represented

Clarification: A "game" means every information related to the game, such as the game mode, what are the moves, size of the board, ...

Represented by a struct "`GameData`" in "`dataStruct.h`". Its members meaning as follow:

`int` `Screen_Mode`: is zero if this game is PvP mode, 1 if this game is PvC Easy, 2 if this game is PvC Normal, and 3 if this game is PvC Hard

`int` `Board_Logical_Size`: The logical size of the gomoku/caro board.

`int` `Player_1_Stone, Player_2_Stone`: Number of stones each player currently have (reserved for placing stone mode)

`int` `Current_Game_Mode`: 0 if Gomoku, CARO_MODE if Caro

`MoveList` `Move_List`: List of moves played

`bool` `Enable_Undo`: Does this game allowed undo or not.

`int` `Current_Turn`: who is currently has the turn? (1 or 2)

4. How we saved game

Basically, saving the data to disk is just the problem of writing a struct of "`GameData`" (See II.3) to disk.

We write to a text file named "`data0x.dat`", with x is either 0,1,2 or 3. The process of saving, as well as how the data arranged in the text file, can be found at function "`Data_WriteToFile`" in "`MenuScreen.cpp`".

5. How, the statistic stored.

As six global integers: `Number_of_PvP_P1_Wins, Number_of_PvP_P2_Wins, Number_of_PvP_Draws, Number_of_PvP_P1_Wins, Number_of_PvP_P2_Wins, and Number_of_PvP_Draws`.

As the program starts running, it will read the statistic data from an external file named "`stat.dat`" or reset the statistic data if it is not found (i.e first time running)

As the program terminates, it will write the current statistic data to the same external file, as a progress of finalizing data.

6. How, the options (settings) stored.

It is just the problem to arrange variables, to fit those features presented by the option.

7. How the computer plays.

First, computer performs a simple check for its own four-in-a-row patterns, (it would move and win of one of such is found), then opened three and four patterns of the player. If such patterns are found, it tries to block the user from winning.

If no obvious pattern is found, it uses the minimax algorithm with alpha-beta pruning with a given depth to find an optimal moves. At the leaf (at their depth limit), computer tries to evaluate the score of the board to know how good current state for him and for the player.

III. Remarks

1. Why we choose our approach: global variables although some disadvantages:

+ Less clarification: if a function just use a variable out of nowhere, it makes those who trying to learn the code (or even the coder himself) feel dizzy.

+ No Access Control: a function cannot know if any other functions change the global variable.

+ No Constraint Checking: it is unsafe for careless programmer.

But:

+ If you can control how the variables are read and write, the constraint checking should not be a problem. Because this is an isolated project (what we really mean, is, no 3rd party plugins except standard library, could jump in and change the variable out of nowhere), so we have total access control, as long as the name don’t overlap with declared one in the library.

+ A good reference provides clear meaning for each (global) variable could solve the clarification problem.

+ A variable is global means it is singleton. So if you change the variable itself, every functions involve that variable automatically change themselves. We take advantage of this property without using a complicated design pattern.

It is singleton means you don’t have to check what version it is, which should be avoid when extensively passing by value for multiple layers.

+ It makes code shorter. We see that this is a small project, so no need to struct this, struct that, pass this, pass that. Just a clear reference list of about 100 global variables or so can easily simplify the data flow.

+ It is debug-friendly.

2. About the name of variables

Name should be obvious about its meaning, so it should be less comments on the code.

IV. References

[1] https://github.com/gigajet/ConsoleDesigner