VIETNAM NATIONAL UNIVERSITY OF HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF SCIENCE
DEPARTMENT OF INFORMATION TECHNOLOGY

END-TERM PROJECT

OBJECT-ORIENTED PROGRAMMING

LECTURES

- **PhD. TRUONG TOAN THINH (Theory Lecturer)**
- **PhD. NGUYEN THANH AN (Teaching Assistant)**
- **M. TRAN NGOC DAT THANH (LAB Lecturer)**

CARO GAME VERSION 1.0

N H Ó M 2

19CLC3 - HCMUS

MEMBERS OF GROUP 2

HO HUU VIEN	18127251
PHAM ANH TUAN	19127084
NGUYEN THANH TINH	19127292
TRAN XUAN SON	19127321

05th September, 2020

Ho Chi Minh City

fit@hcmus

ACKNOWLEDGEMENTS

During the period of this project, we received a plenty of enthusiastic help and support that guide and encourage us to overcome all difficulties and finish this hard but meaningful project.

We would like to express our special thanks to all lecturers of this subject, for their guildances and support in knowledge and skills you taught us:

- PhD. Truong Toan Thinh (Theory Lecturer)
- PhD. Nguyen Thanh An (Teaching Assistant)
- M. Tran Ngoc Dat Thanh (LAB Lecturer)

Last but not least, me – Nguyen Thanh Tinh – the leader – really grateful to all of members' efforts (Tran Xuan Son, Pham Anh Tuan, Ho Huu Vien), who contributed to finish this project perfectly.

Ho Chi Minh City, 05th September 2020
On behalf of the group
Leader

Nguyen Thanh Tinh.

TABLE OF CONTENTS

I. ABOUT US

We now are students of Ho Chi City University of Science, studying Information Technology in faculty of Information Technology, and this group was created since we joined in the same Object-Oriented Programming class. Not only we knew each other before, but also we figured out the individual strengths that are really suitable for effective project working.

1. Nguyen Thanh Tinh

- Student ID: 19127292
- Hometown: Quy Nhon, Binh Dinh
- Role: Leader, Backend Developer

2. Tran Xuan Son

- Student ID: 19127321
- Hometown: Ly Nhan, Ha Nam
- Role: Frontend Developer

3. Pham Anh Tuan

- Student ID: 19127084
- Hometown: Ho Chi Minh City
- Role: Tester

4. Ho Huu Vien

- Student ID: 18127251
- Hometown: Ho Chi Minh City
- Role: Project Assistant, Tester

II. GENERAL INTRODUTION

1. Topic

- Name: CARO GAME
- Technique:

  In this project, we use class techniques and basic data structure to build a simple caro game. To complete this project, we also have to know some basic knowledge: object-oriented design, file processing, two-dimensional array, graphic library, UX/UI design, …
- Requirements we followed:
  - **Save/Load (2 points) : Done**

    When the players hit 'L', we show a screen requesting the players provide the filename to save. When the players hit 'T', we show a screen requesting the players provide the filename to load.
  - **Recognize win/lose/draw (2 points) : Done**

    We wrote a function to recognize win/lose/draw. Furthermore, that function have more ways to check, more results to show, so that we could re-use that function for more purposes (ex: AI Moving,…).
  - **Provide animation of win/lose/draw (2 points) : Done**

    Of course we made a vivid animation for this important event!
  - **Creating playing interface (1.5 points): Done**

    We organized the interface for the game, and it's really clear and pretty – all we want is the simpliest experiences for players.
  - **Provide the main menu (1.5 points) : Done**

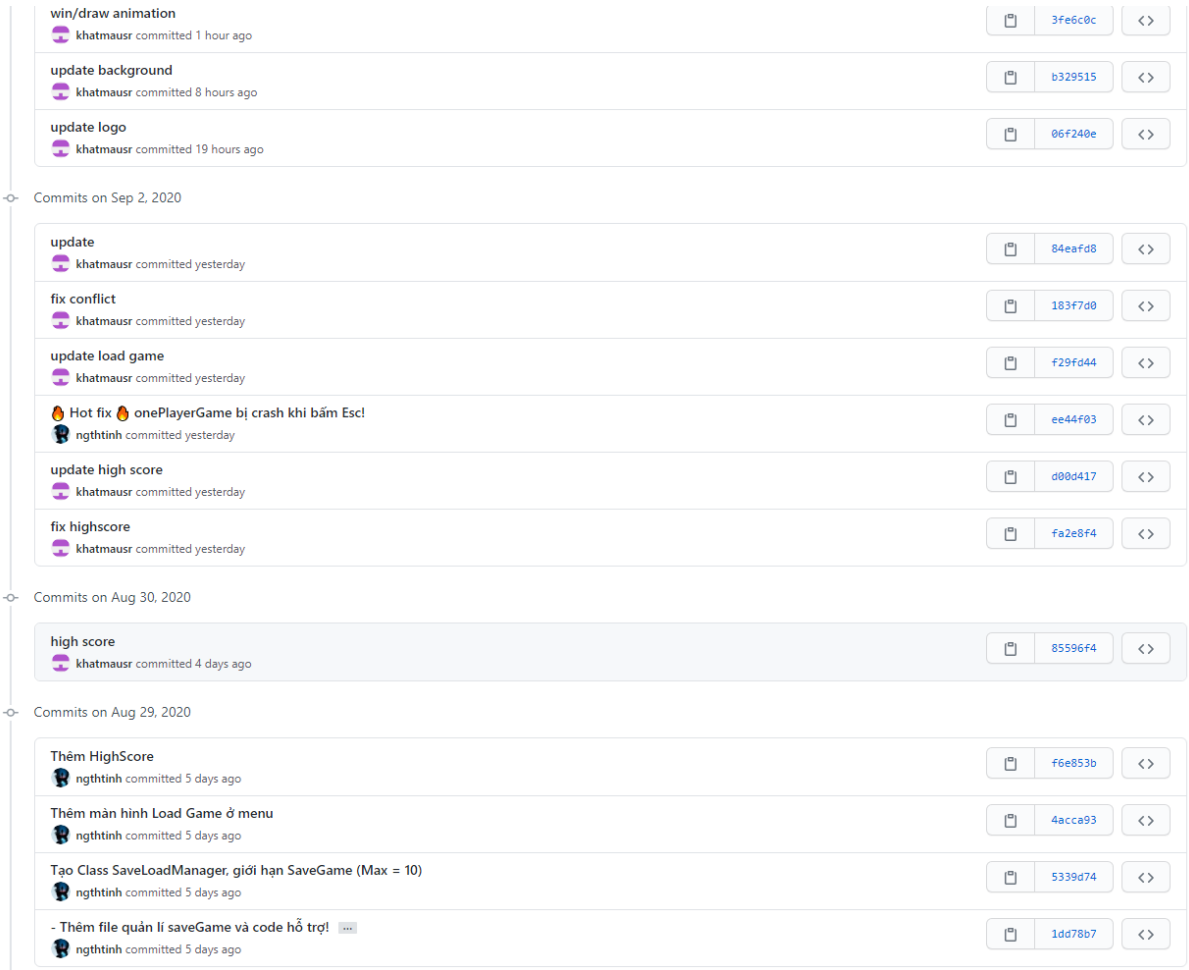
    When the game starts, we print a menu game with many options ("Continue Game", "New Game", "Load Game", "High Score", "About" and "Quit"). It helps players to easily choose actions they want.
  - **Playing with machine (Alpha – Beta prunning) (1 point): Done**

    We allow players to choose "Play with machine", and players can choose the level, for example : Easy – randomize position, Medium – Greedy Algorithm, and Hard – Alpha Beta Prunning.

2. Supporting tools

- Operating System: Windows 10 Version 2004 (OS Build 19041.264)
- IDE: Microsoft Visual Studio Community 2019 Version 16.7.2
- Programming language: C++
- Graphics Library: SFML 2.5
  https://www.sfml-dev.org/index.php
- Supporting software: Adobe Xd 2020, Adode Illustrator 2020,…
- Collaborative version control: GitHub

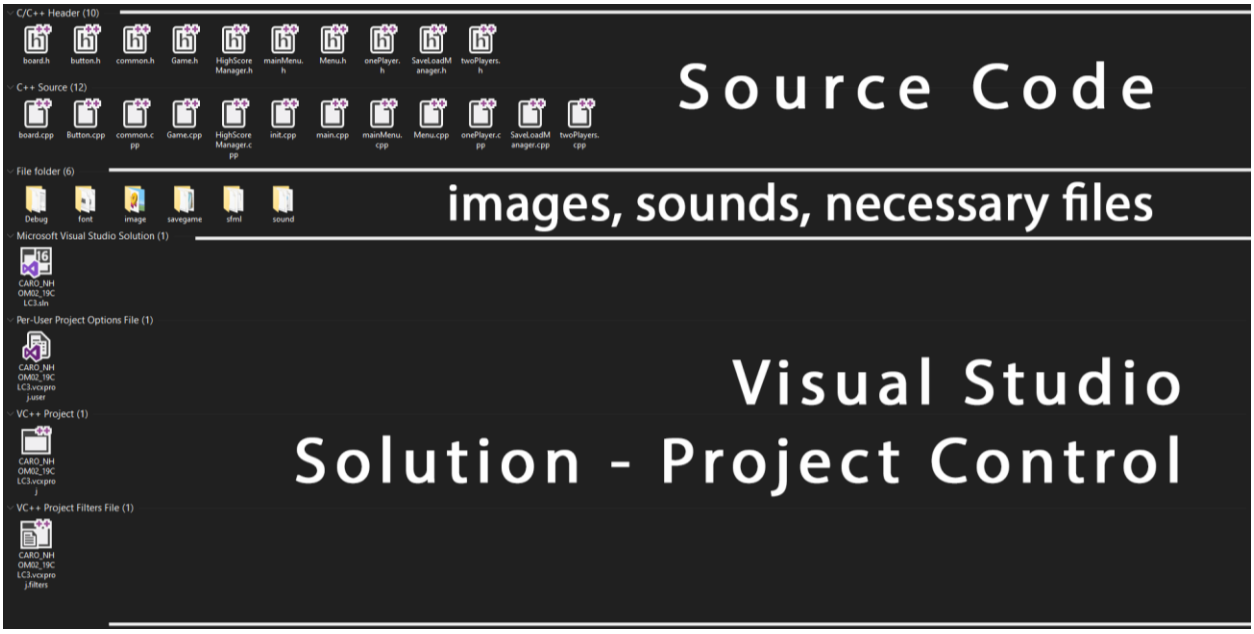
Overview about proof of work



GitHub is a code hosting platform for collaboration and version control. GitHub lets us (and others) work together on this projects.

III. PRODUCT INTRODUTION

1. Folders and Files

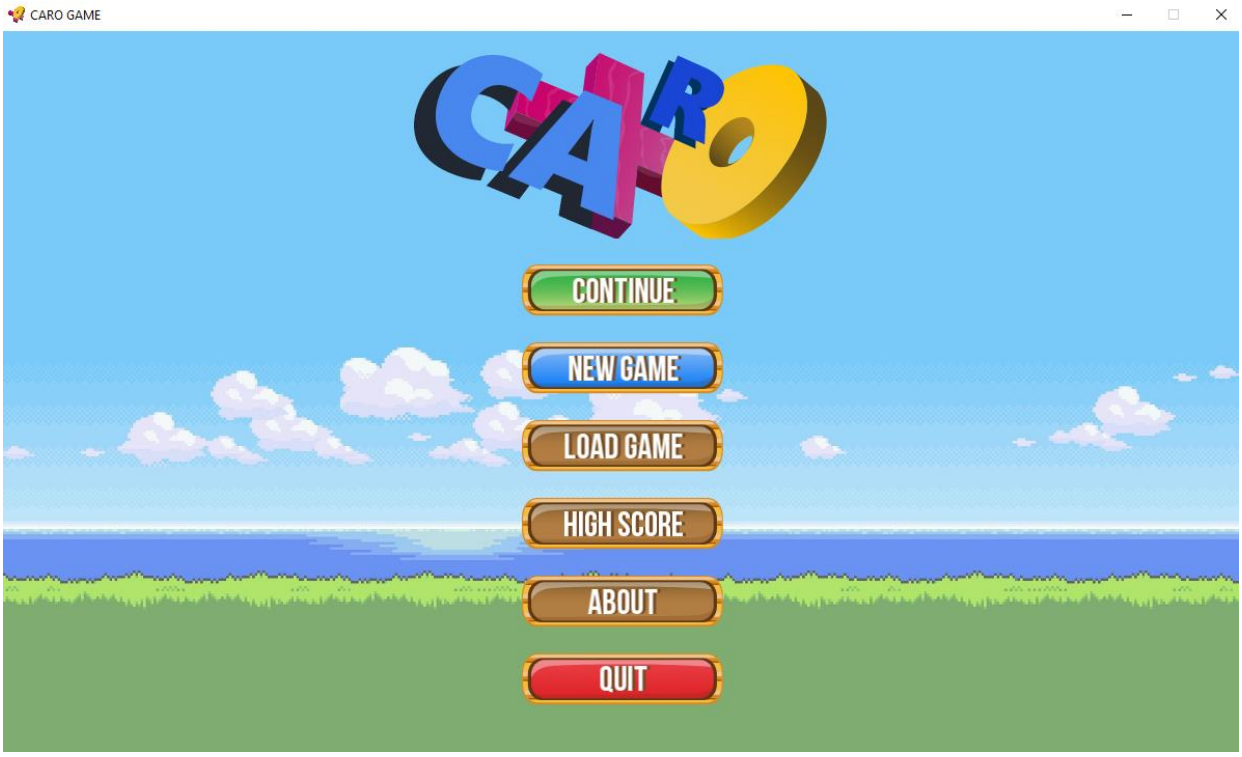


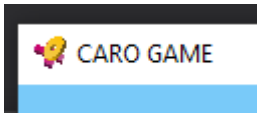
These files on the picture above may be a bit of different from the last submit .zip file

As you can see in the picture, there're many file in our project. Specifically:

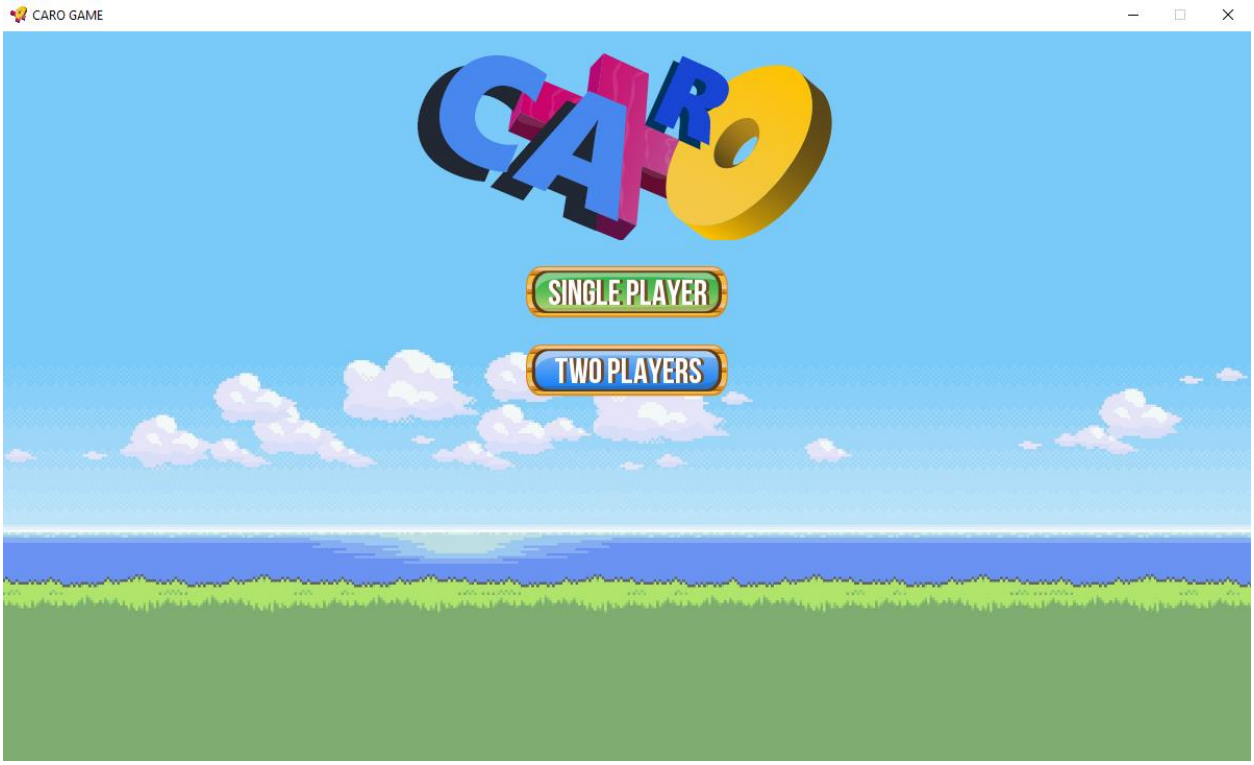
- Folders:
  - Debug: using for building a new program from source codes openning in Visual Studio.
  - font: include some font to display text objects.
  - image: all images are here, the game's interface load texture from this folder.
  - savegame: include savegame file, savegame manager file and highscore manager file.
  - sfml: graphic library
  - sound: menu music, game music, button sound, etc…
- C/C++ Header, C/C++ Sources
- Visual Studio Solution – Project Control file: All necessary file for a normal compiling task of Visual Studio. You can open the file name that `CARO_NHOM02_19CLC3.sln` to view detail of our sources.

2. Game's Interface

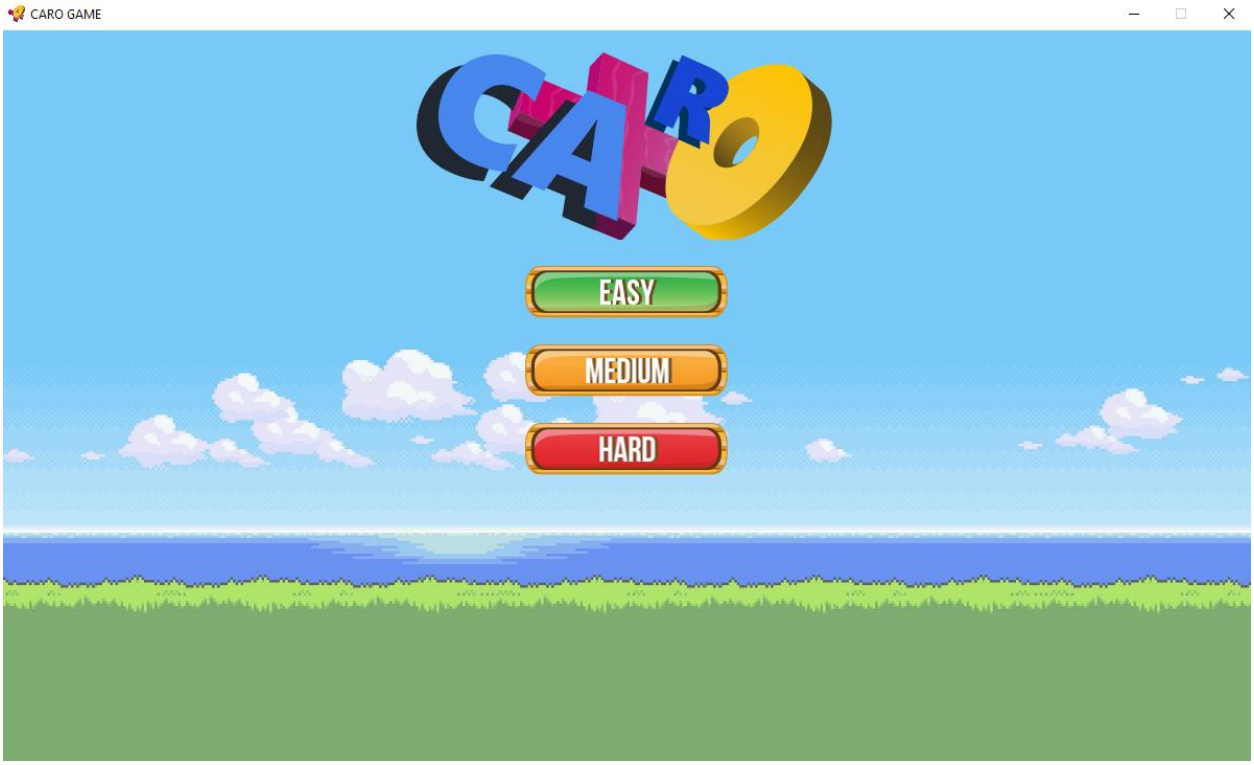


	Game's icon and name.
	Windows' control buttons. We disabled the maximum/minimum button so that everything displaying must be correctly.
	Game's logo.
	Option buttons. You can use the mouse to click the actions you would like to run.

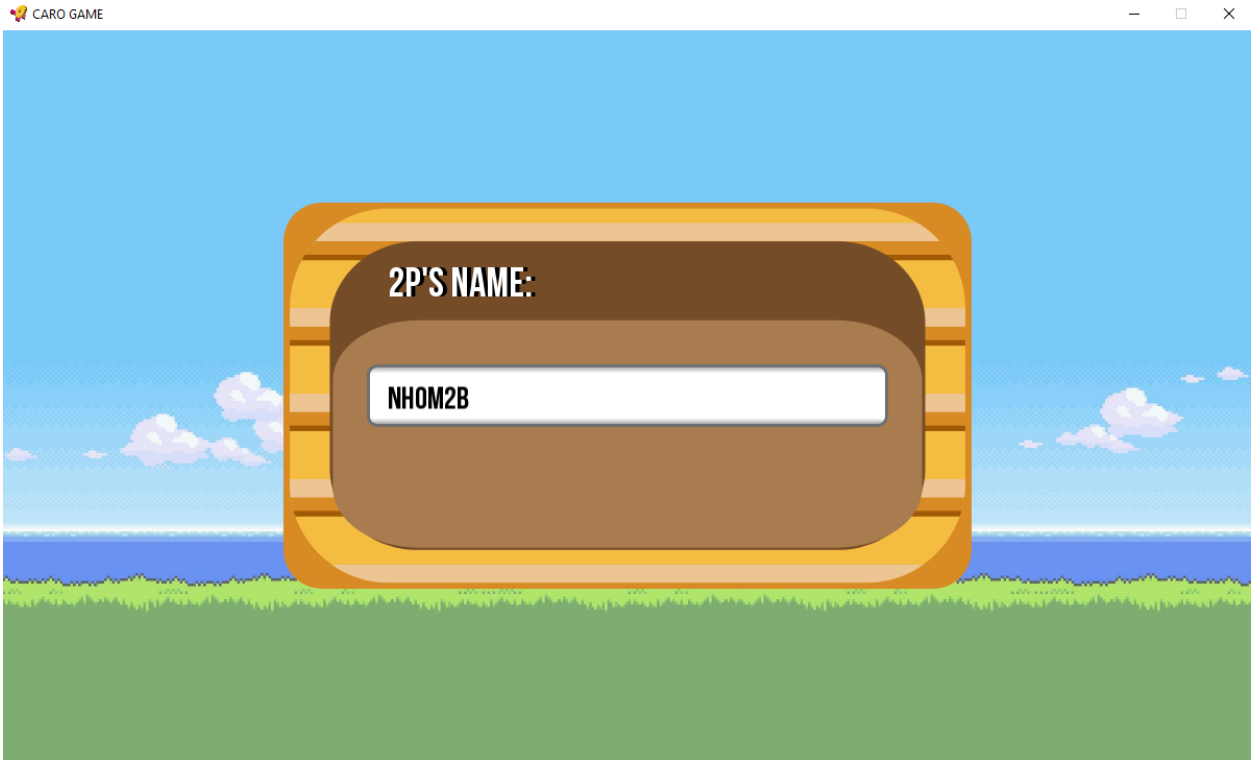
Here is “New Game” option, we provide one player game and two players game.



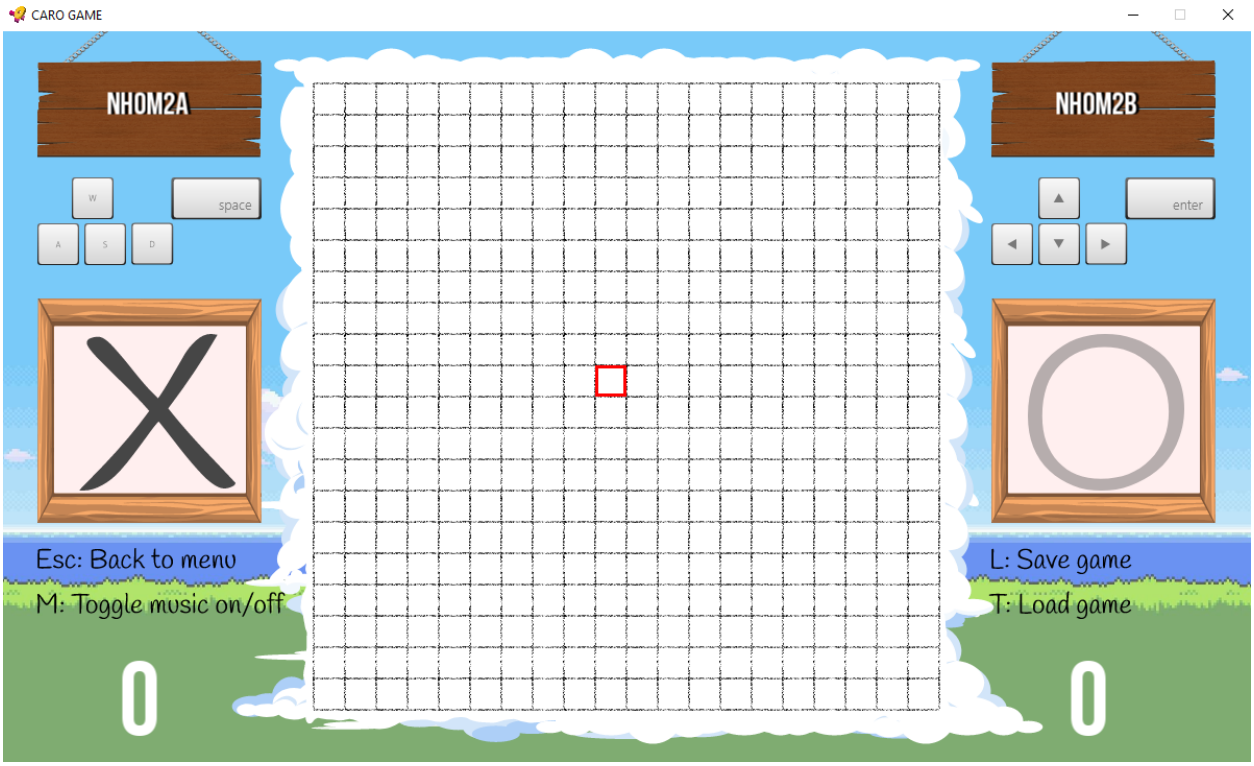
Espeacially, when players want to try “Single player”, we provide 3 mode: Easy, Medium and Hard.

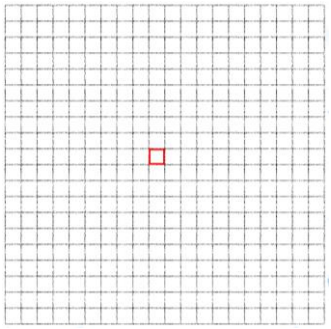


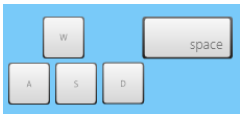
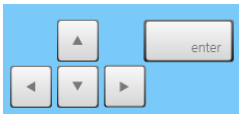
Before starting a game, players need to provide their name. This information is really important for saving games and highscores.



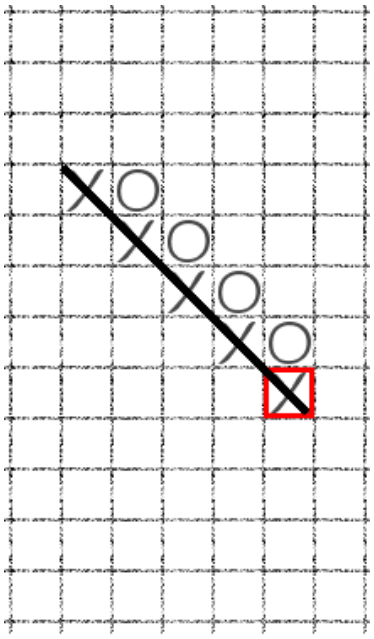
And this is in-game interface:

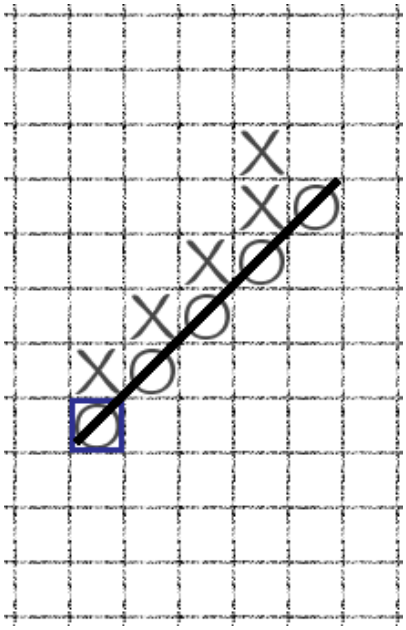


		The board: Display all moves of players.
		X and O.
		Cursor of X and cursor of O. Let players know where are they.

		Players Name.
		Cusor control buttons introductions.
		Game control buttons introductions.
		Score of players.

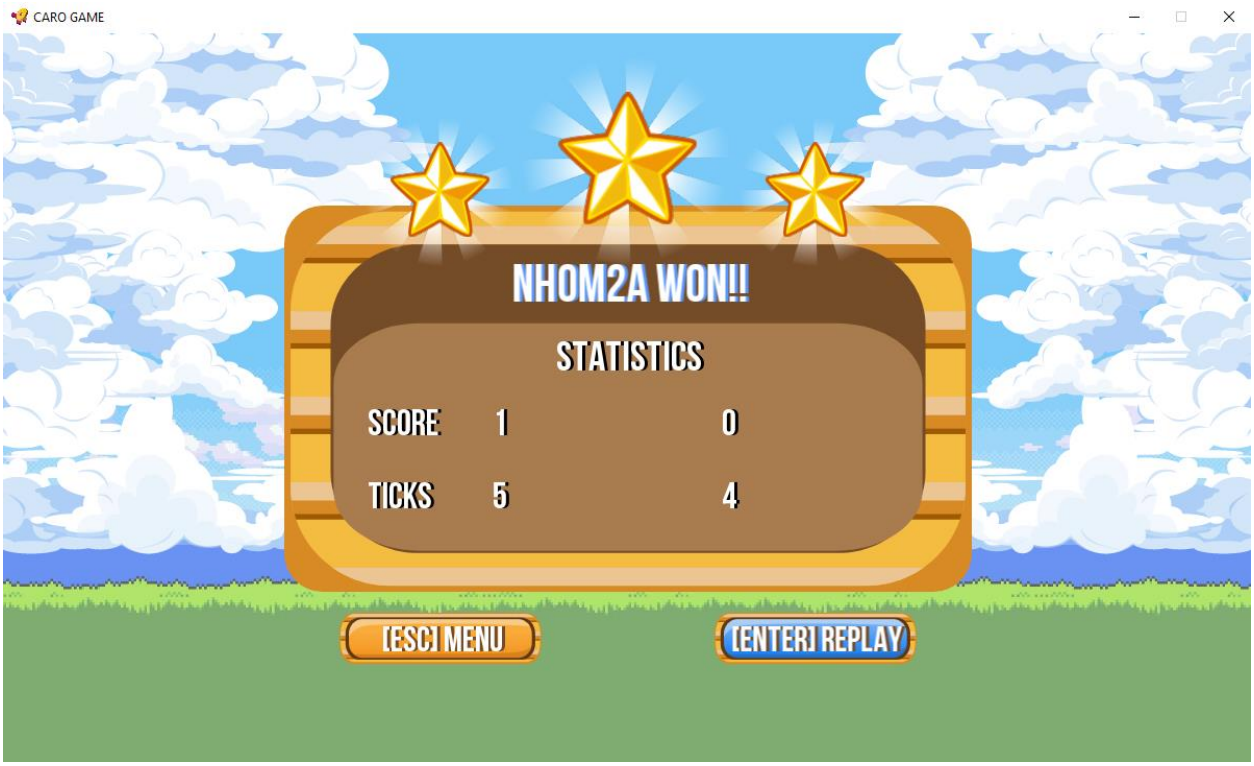
The game usually check win/lose/draw every time players hit on board. Whenever a player win, the game draw a line to show winning moves.

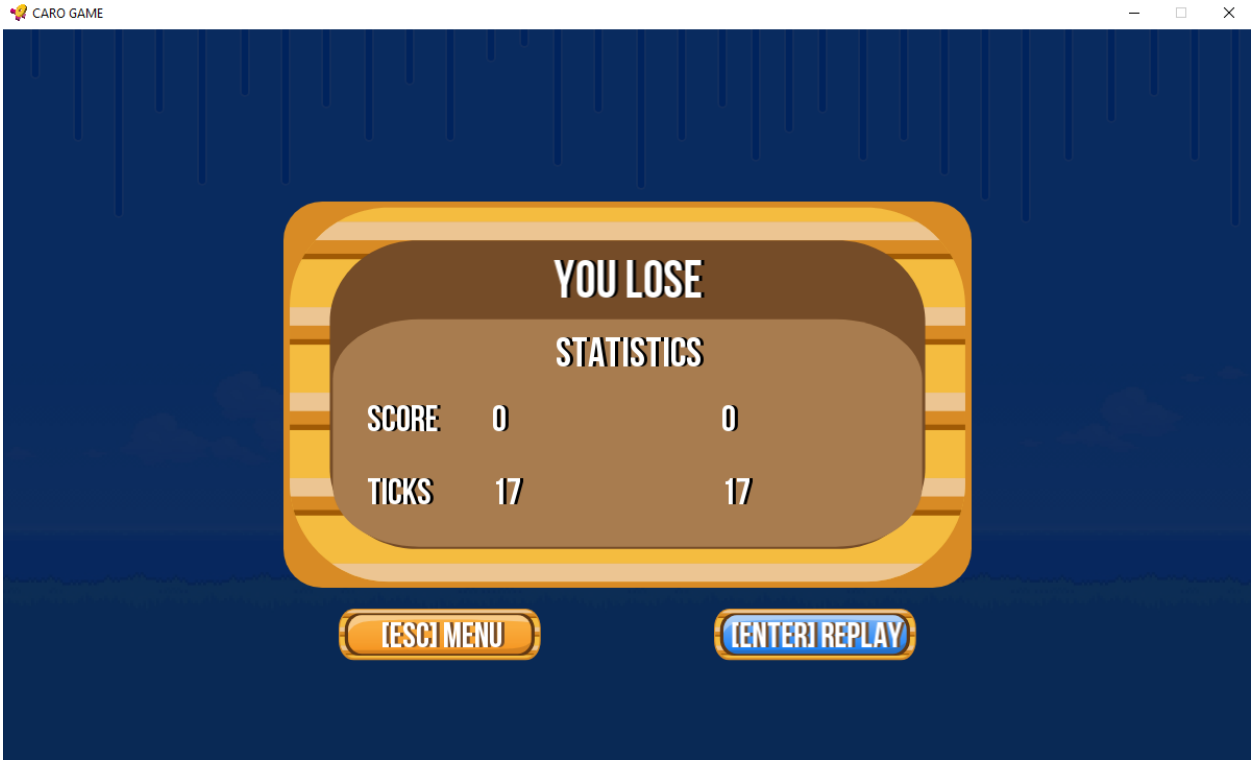




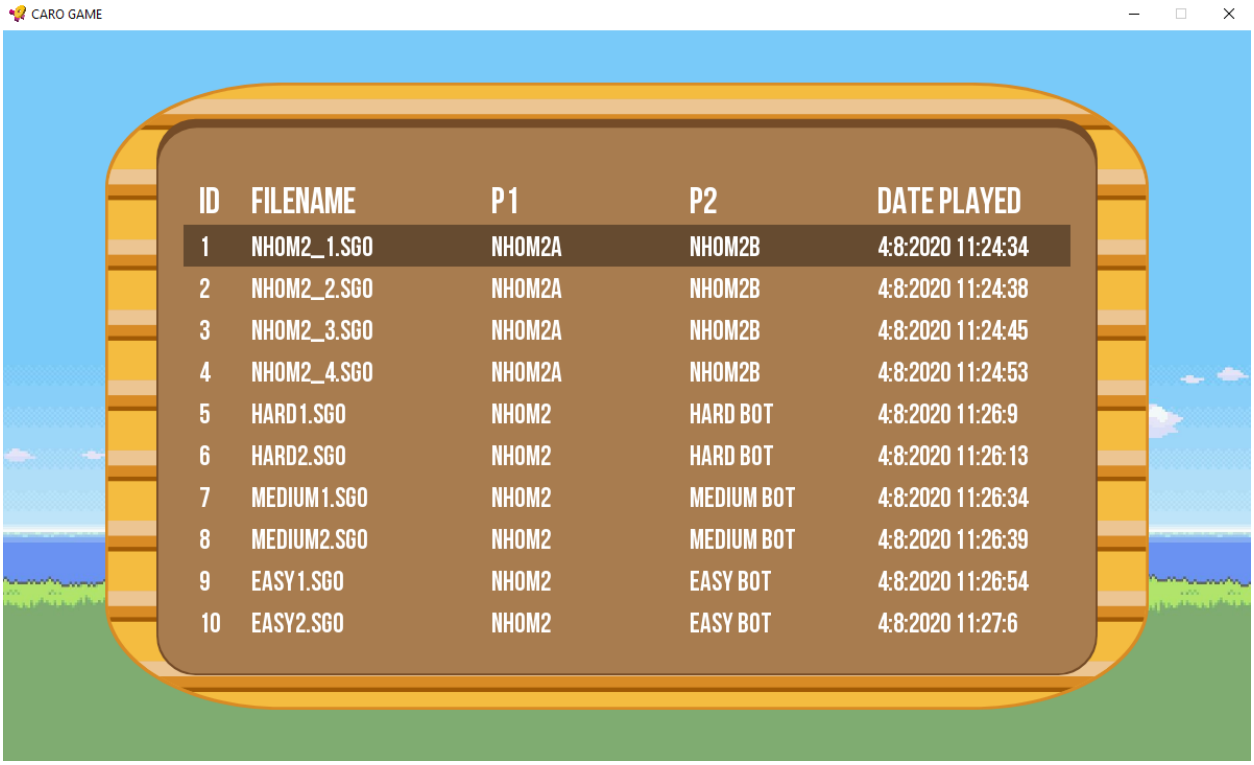
When the players hit ‘L’, we show a screen requesting the players provide the filename to save. When the players hit ‘T’, we show a screen requesting the players provide the filename to load.

We also provide a vivid animation for Win/Lose/Draw event.

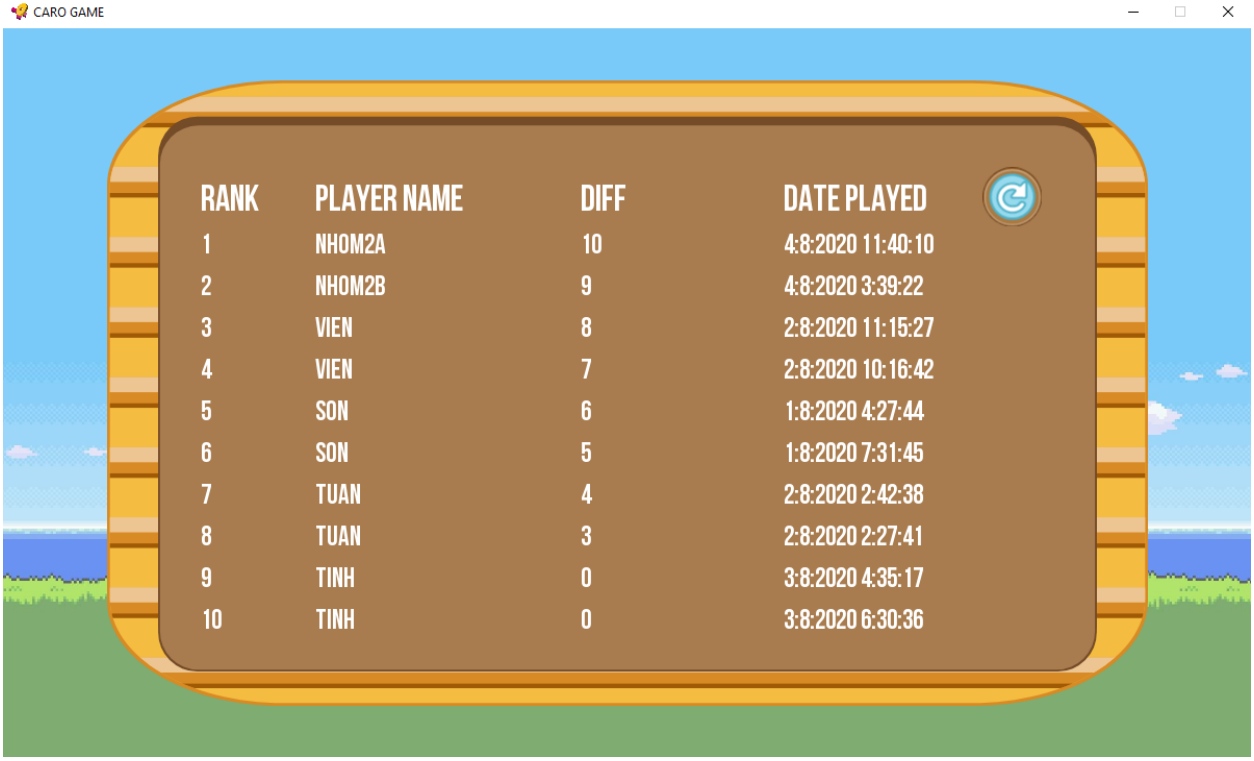




Load Game Screen:



High Scores Screen:



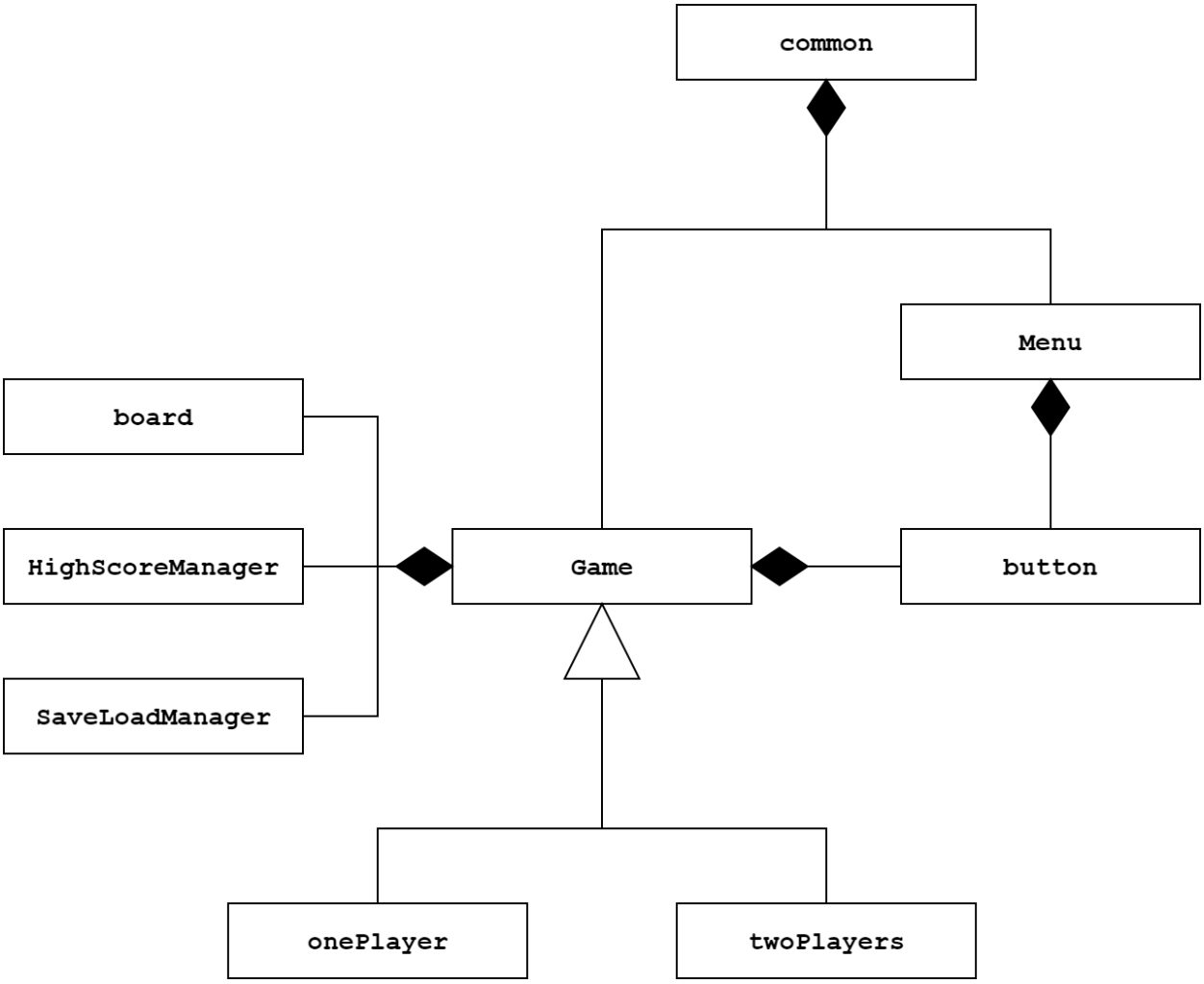
"About" Screen:

To create a wonderful graphics game interface like these, we use a lot of UX/UI designing skills and knowledge about graphics library (in this project, it's SFML, and we learned it on tutorial page).

Graphics library	SFML
Support sofware	Xd Ai Au and so on…

IV. ABOUT SOURCE CODE

1. UML



- Class: Common (HAS-A class Game and class Menu)

4 statics methods: initializing game textures, musics, font,…; running the game; displaying opening screeen and "About menu".

common
+ initGame() : void + runGame() : void + displayOpeningScreen() : void + aboutMenu() : void

- Class: Menu (HAS-A class button)

(sf:: is a namespace in SFML Graphics Library, same as std::)

Displaying list of menu buttons and allowing players to choose actions they want by mouse click.

```
Menu
- background    : sf::Sprite
- btnList       : vector<Button*>
- buttonSpacing : float
- position      : sf::Vector2f
+ isActive          : bool
+ selectedItemIndex : int
+ Menu(backGroundTexture: sf::Texture*, position: sf::Vector2f, buttonSpacing: float)
+ ~Menu()
+ pushButton (btnCode: unsigned int, title: string): void
+ draw() : void
+ update(mousePosz: sf::Vector2f, isClicked: bool)    : void
+ setActiveBtn(idx: unsigned int, isActive: bool)  : void
```

- Class: button (PART-OF class Menu and class Game)

Creating buttons to click, be used in class Menu and class Game.

```
button
- soundEffect          : sf::Sound
- defaultPlaceHolder   : sf::Sprite
- mouseOverPlaceHolder : sf::Sprite
- title                : sf::Text
- titleShadow          : sf::Text
- currentSprite        : sf::Sprite*
- currentState         : unsigned int
+ Button(defaultTexture: sf::Texture, mouseOverTexture: sf::Texture*,
soundBuffer: sf::SoundBuffer*, title string, position sf::Vector2f);

+ Button(btn: const Button&, position: sf::Vector2f);
+ ~Button()
+ update(mousePos: sf::Vector2f, isClicked: bool): void
+ setText(title: string)                          : void
+ setState(state: unsigned int)                   : void
+ getState()                                      : unsigned int
+ setPosition(position: sf::Vector2f)             : void
+ getPosition()                                   : sf::Vector2f
+ getScale()                                      : sf::Vector2f
+ setScale(scaleX: float, scaleY: float)          : void
+ draw()                                          : void
```

- Class: board (PART-OF class Game)

Saving players' moves, displaying them to screen and check for Win/Lose/Draw.

```
board
-------------------------------------------------------------------------
- arr: int**
- countX: int
- countO: int
-------------------------------------------------------------------------
+ board()
+ ~board()

+ exportBoard()                        : vector <int>
+ importBoard(dataBoard: vector <int>): bool

+ resetBoard()                        : void
+ displayBoard()                      : void

+ markCell(x: int, y: int, val: int) : bool
+ redoMarkCell(x: int, y: int)       : void
+ checkBoard(x: int, y: int, x_begin: int&, y_begin: int&, k: int): int

+ getCell(x: int, y: int) : int
+ getCountX()             : int
+ getCountO()             : int
```

- Class: HighScoreManager and SaveLoadManager (PART-OF class Game)

Managing SaveGame files and HighScore record.

```
HighScoreManager
-------------------------------------------------------------------------
- highScoreList         : vector <HighScoreInfo>
- updateHighScoreList() : void
- sortHighScoreList()   : void
- updateFileManager()   : void
-------------------------------------------------------------------------
+ pushHighScore(temp: HighScoreInfo&) : void
+ showHighScore()                     : void
```

```
SaveLoadManager
-------------------------------------------------------------------------
- saveList            : vector <SaveGameInfo>
- updateSaveList()    : void
- updateFileManager() : void
-------------------------------------------------------------------------
+ checkFile(filename: string, typeGame: int): bool
+ pushSaveGame(temp: SaveGameInfo&)         : void
+ loadForGame(filename: string&)            : int
```

- Class: Game(HAS-A button, board, HighScoreManager, SaveLoadManager)

This class is the core of the game:

- Specify what type of game is (Single player or two players? Easy or Hard?)
- Save names, scores of two players
- Display game board to the screen
- Run Game (New Game or Continue Game) and Exit Game.
- Save all data game into a BINARY FILE (.SGO)
- Polymorphism (class Game is base class of onePlayer and twoPlayer, and it's polymorphism object in a function of common.cpp).

```
                              Game
# typeGame       : int
# b              : board
# turn           : int
# cursorP        : sf::Vector2u
# scoreX         : int
# scoreO         : int
# isExit         : bool
# playerName[2]  : string
# background     : sf::Sprite

# runGame()      : void
# resetData()    : void
# exitGame()     : void
# displayGame()  : void

# changeTurn()   : void
# markWin(x_begin: int, y_begin: int, direction: int) : void
# displayWin(isDraw: bool) : bool

# askForName()                     : void
# processKeyPressed(keyCode: int)  : void
+ ~Game()
```

```
+ Game(bgTexture: sf::Texture*)
+ isContinue()                : bool
+ continueGame()              : void
+ newGame()                   : void
+ askForLoad()                : void
+ askForSave()                : void
+ saveGame(fileName: string)  : void
+ loadGame(fileName: string)  : void
+ getType()                   : int
```

- Class: onePlayer and twoPlayers (IS-A class Game)

```
onePlayer
- askForName()                      : void
- processKeyPressed(keyCode: int)   : void

- distance(pA: sf::Vector2u, pB: sf::Vector2u) : double
- botMove()      : sf::Vector2u
- easyBotMove()  : sf::Vector2u
- medBotMove()   : sf::Vector2u
- hardBotMove()  : sf::Vector2u
- alphaBetaPrunning(isMaximize: bool, depth: int, maxDepth: int) : sf::Vector3i

- displayLose(isDraw: bool) : bool
+ onePlayer(bgTexture: sf::Texture*, level: int);
+ ~onePlayer();
```

```
twoPlayers
- askForName()                      : void
- processKeyPressed(keyCode: int)   : void
+ twoPlayers(bgTexture: sf::Texture*);
+ ~twoPlayers();
```

onePlayer and twoPlayers class are inherit from Game class. Each class has a own way to process key press, so we used inheritance technique on these class, so the code is short and clean.

Different from twoPlayers class, onePlayer class has more methods inside.

Methods for AI Move:

- `Distance(pA: Vector2u, pB: Vector2u) : double`
- `botMove()  : sf::Vector2u`
- `easyBotMove()  : sf::Vector2u`
- `medBotMove()  : sf::Vector2u`
- `hardBotMove()  : sf::Vector2u`
- `alphaBetaPrunning (…) : sf::Vector3i`

Methods to show that you lose whenever you play with AI.

- `displayLose(isDraw: bool) : bool`

2. Source-Code Spotlight

a. Save/Load (2 points)

```
void Game::Process Key Pressed (Key Code)
{
…
If Key Code is 'L': Call Ask For Save;
If Key Code is 'T': Call Ask For Load;
…
}

void Game::Ask For Save ()
{
Step 1: Get players' file name string
Step 2: Check for Validity of the string
            If Invalid: Back to Step 1;
            Else Head to Step 3;
Step 3: Create a struct of all game's data
Step 4: Write the struct into a file with BINARY format
Step 5: Save some info into SaveLoadManager class, so that the game could manage and keep save files original.
Step 6: Announce "Sucessful"
}
void Game::Ask For Load ()
{
Step 1: Get players' file name string
Step 2: Check for Validity of the string
            If Invalid: Back to Step 1;
            Else Head to Step 3;
```

Step 3: Create a struct to store all game's data

Step 4: Read the file with filename entered by players into the struct in BINARY format

Step 5: Coping data from the struct to game's data

Step 6: Announce "Sucessful"

}

b. Recognize win/lose/draw (2 points)

```
int board::Check Board (int x and y coordinate)
{
Step 1: If arr[x][y] equals zero, it means no move there, return 0 (No result)
Step 2: Check vertical
  If there are 5 consecutive moves have the same value, return 1 (Vertical winning moves)
  Else go to Step 3.
Step 3: Check horizon
  If there are 5 consecutive moves have the same value, return 1 (Vertical winning moves)
  Else go to Step 4.
Step 4: Check diagonal
  If there are 5 consecutive moves have the same value, return 1 (Vertical winning moves)
  Else go to Step 5.
Step 5: Check anti-diagonal
  If there are 5 consecutive moves have the same value, return 1 (Vertical winning moves)
  Else go to Step 6.
Step 6: return 0 because there are no winning moves.
}
```

c. Provide animation of win/lose/draw (2 points)

```
void Game::Process Key Pressed (Key Code)
{
…
If Key Code is 'Enter':
{
        Step 1: Mark the board
        Step 2: If Check Board equals TRUE, display
        Win/Lose Animation!
        Else head to Step 3;
        Step 3: If Board is full of move, display DRAW!
        Else head to Step 4;
        Step 4: Change Turn (in twoPlayers)
        Step 5: Back to the game
}
…
}
```

d. Creating playing interface (1.5 points): Done

class board	class Game
{	{
…	…
int countX;	int scoreX;
// Count moves of X	// Score of X
int countO;	int scoreO;
// Count moves of O	// Score of O
…	…
}	}

e. Provide the main menu (1.5 points) : Done

```
mainMenu.pushButton(2, "CONTINUE");
mainMenu.pushButton(0, "NEW GAME");
mainMenu.pushButton(1, "LOAD GAME");
mainMenu.pushButton(1, "HIGH SCORE");
mainMenu.pushButton(1, "ABOUT");
mainMenu.pushButton(3, "QUIT");
newGameMenu.pushButton(2, "SINGLE PLAYER");
newGameMenu.pushButton(0, "TWO PLAYERS");
botMenu.pushButton(2, "EASY");
botMenu.pushButton(4, "MEDIUM");
botMenu.pushButton(3, "HARD");
```

f. Playing with machine (Alpha – Beta prunning) (1 point): Done

```
Vector2u onePlayer::botMove()
// Return a Vector2u (Vector has 2 unsigned integers)
with x and y, to allocate where the BOT needs to hit.
{
     If Game's mode is Easy
          Return easyBotMove();
     If Game's mode is Medium
          Return medBotMove();
     Return hardBotMove();
}

Vector2u onePlayer::easyBotMove()
// Random somewhere nearly player's cell
{
     Step 1: Init vector of answer
     Step 2: Browse all the Board
          If that cell is not hitted
          {
               If it's closer, delete old vector and
               push the new one into vector answer.
               If the distances are equal,
               Push that cell into vector answer.
          }
     Step 3: return any value in vector answer

}
```

```
Vector2u onePlayer::mediumBotMove()
// Greedy Algorithm
{
     Step 1: k = 4 (k is priority, equals how many
     consecutive moves has the same value)
     Step 2: If k >= 2, head to Step 3;
     Else go to Step 5;
     Step 3: Browse all cell that are hitted
          If that cell has k - consecutive moves and
     not to be blocked at both head and tail, Return
     that cell!
          If that cell has k - consecutive moves and
     not to be blocked at head or tail, Return that
     cell!
     Step 4: Decrease k. Back to Step 2;
     Step 5: No cell is return? Return easyBotMove()
}

Vector2u onePlayer::hardBotMove()
// Alpha - Beta Prunning
{
     Step 1: Get answer from alphaBetaPrunning(…)
     Step 2: Take x and y coordinate out, ignore z
     (z is score of move found by alphaBetaPrunning())
     Step 3: Return answer
}

Vector3i onePlayer::alphaBetaPrunning

(bool isMaximize, int depth, int maxDepth)

// isMaximize show that step is Find Max, or Find Min

// depth : depth of DFS, maxDepth : maximum of DFS

{

Step 1: Browse all cells that are NOT hitted

Step 2: Try to mark that cell

Step 3: Find point of that cell

(point of cell equals how many consecutive moves has the
same value, and it's stored in variable name z)

Step 4: //Prunning

If that cell has no point, ignore it and go to Step 7;

Else head to Step 5;
```

```
Step 5: Find curseMove by call a cursed function

alphaBetaPruning(!isMaximize, depth + 1, maxDepth);

Step 6: //Minimax

If we are maximizing, choose minimum value between
currentMove.z and cursedMove.z, or opposite.

Store suitable value into currentMove (tempMove).

Step 7: Redo marking that cell. If there're unchecked
cells left, back to Step 1 to continue browsing

Step 8: //Minimax

If the moveVector is not empty

Return the cell with Maximum z point if we are Maximize,

Or Return the cell with Minimum z point instead!

Step 9: If there's no move found by AlphaBetaPruning()

Return medBotMove().

}
```

V. REFERENCE

- CaroProject_OOP.pdf: PhD.Truong Toan Thinh, CARO Game GUILDLINE
- Main Menu Music:
  https://youtu.be/3XH-FOSseek
- Game Music:
  https://youtu.be/MiUjLJJligs
- SFML 2.5 Tutorial:
  https://www.sfml-dev.org/tutorials/2.5/
- Giải thuật cắt tỉa Alpha-Beta
  www.stdio.vn/giai-thuat-lap-trinh/giai-thuat-cat-tia-alpha-beta-Wu7F1
- Giải thuật tìm kiếm MiniMax
  www.stdio.vn/giai-thuat-lap-trinh/giai-thuat-tim-kiem-minimax-s1EVnH
- YOUTUBE VIDEO LINK of our group:
  https://youtu.be/O5LK14R0APQ