Design Notes

1. Files and their functionality (Task 2)

   Implementation includes 2 C++ files, 1 header file.

   The assignextra_main.cpp contains `main()` program to control the workflow;

   The tictactoe.h is the header file that declares a Game class and two functions to implement the whole Tic Tac Toe game. pve() is to control task flow in PVE mode, `pvp()` is to handle that in PVP mode;

   Algorithms and judgements involved in the Tic-Tac-Toe game are in a class-type called `Game`, its details are listed in section 4.a. It handles both gameboard situation storing, gameboard interpreting, reading input and bot simulation.

2. Bot Performance Review (Task 1)

   Generally the bot is designed to be passive. It will attack only when there are two bot's pieces and one vacancy in one single line. In a no-danger-situation (meaning the player cannot win in the next step), it will randomly place a piece, while in a danger-situation, it will block the player's winning move.

   There is also an Easy Mode designed for kids aged 1-3. In the Easy Mode, Bot will follow the above algorithm before the first 5 piece-moves (2 by player 3 by computer; or 3 by player 2 by computer). For the rest moves, the computer will randomly choose a cell to place its piece. Therefore, the probability for player's winning is much higher, which is the suitable difficulty for the low-aged target client.

3. Notable Aspects of Implementation

a. A <class> type is used in the implementation, which includes all the features involved in the Tic-Tac-Toe game. This makes the program portable and easy-to-modify. The functions and variables within are explained in the following chart.

Function / Variable	Type	Usage and Specification
`gameboard`	variable, array of 9 ints	Current game board situation; -1: player2’s piece, 0: empty cell, 1: player1’s piece
`status`	variable, vector of 8 ints	The sum of each line; Range from -3 to 3; Represent the condition to win
`piece`	variable, array of 3 chars	`‘O’` or `‘ ’` or `‘X’`; Can be empty, player1’s piece or player2’s piece
`result`	variable, 1 int	Range from -1 to 1; -1: player2 wins, 0: draw, 1: player1 wins
`process`	variable, 1 int	The number of steps taken; range from 0 (empty) ~ 9 (filled)
`init()`	function	Initialize the variables to default
`cal_status()`	function	Calculate the status by gameboard
`print_board()`	function	Print formatted game board
`get_input()`	function	Get user input of the desired placing-piece cell position index
`bot_play()`	function	Computer in PVE mode
`show_win()`	function	Show the winning side (or tie)

b. The gameboard is represented in 9 integers of -1, 0 or 1. This way the situation and the possibility can be directly reflected by the sum of the lines. For example: one line is `['O', 'X', 'O']`, sum is -1 or 1 which indicates safe; `['O', ' ', 'O']`'s sum is 2 or -2 which indicates danger; `['O', 'O', 'O']`'s sum is 3 or -3 which indicates winning.