Nathan Roe
OOP with C++
EN.605.604 Section 81
2/14/2022

Module 3 Assignment: Tic Tac Toe

Design

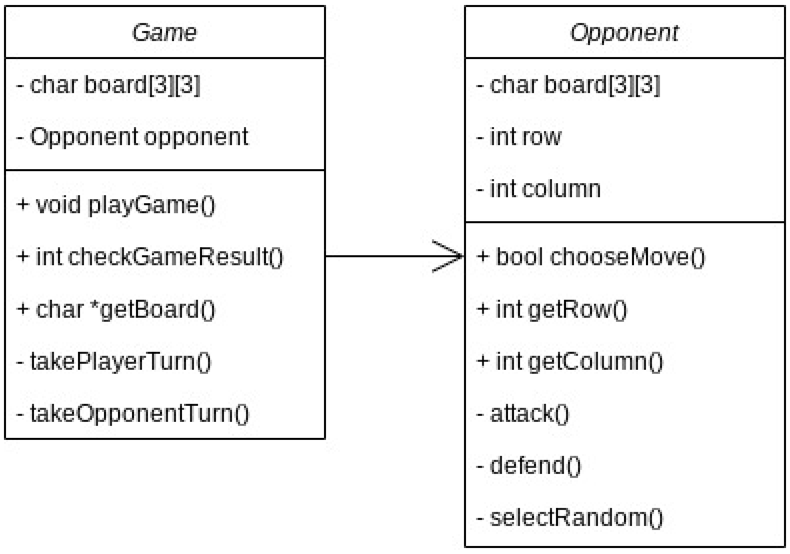
The assignment requirements for the TicTacToe project require building of a single-player game of TicTacToe, where the player plays as X and the computer plays as O. To do this, I will make two classes; the first will include all of the Game rules, and the second will be the Opponent class. The first is needed to handle all of the game rules; the second I will break out into a second object in order to have a replaceable Opponent type, that can ideally be switched out with similar objects using a standard interface to provide different levels of competence in the opponent.

The game object will need a few things:

- Run turn-by-turn behavior of the game (playGame)
- Handle User turns to parse and accept user input (takePlayerTurn)
- Handle calling and accepting Opponent turn choices (takePlayerTurn)
- Evaluate game status to determine wins/loses/ties (checkGameResult)
    - Make this static so that Opponents can leverage for testing moves

The opponent will need a few things for a basic level of gamesmanship:

- Function to call to make choices
- Block user turns (defend)
- Attempt winning moves (attack)
- Otherwise, select random location (selectRandom)



```cpp
/// @file
/// @author Nathan Roe
/// Single-Player Tic-Tac-Toe Game
///
/// Player plays as 'X' and enters moves in the form "# #"
/// for the row, column of their move, and the software
/// plays as 'O'.

#include "Game/game.hpp"
#include <iostream>

using namespace std;

/// Main function; runs gameplay loop
int main()
{
    Game *newGame = new Game();

    cout << "Welcome to Tic Tac Toe" << endl;
    cout << "You will play as x and you're up First" << endl;
    cout << "Enter your Move as <row column>, ie '1 2'" << endl;
    // Kick-off new game
    newGame->playGame();

    cout << "Game Over!" << endl;

    return 0;
} // End function main
```

```cpp
#pragma once
#include "../Opponent/opponent.hpp"

/// @file
/// @author Nathan Roe
/// Main Game Object for Tic-Tac-Toe
///
/// Player plays as 'X' and enters moves in the form "# #"
/// for the row, column of their move, and the software
/// plays as 'O'.
class Game
{
public:
    /// Enumeration of Game States
    ///
    /// Player - Player Win
    /// Opponent - Opponent Win
    /// Tie - Board filled in with no win
    /// Ongoing - Incomplete Game
    enum class GameResultType
    {
        Player,
        Opponent,
        Tie,
        Ongoing
    };

    /// Run Main Gameplay Loop
    ///
    /// Runs a set of moves from player/opponent, and evaluates
    /// gamestate. Print results when game is complete
    void playGame();

    /// Check the status of a Game Board
    ///
    /// Checks to see the current status of a 3x3 matrix, to
    /// Evaluate if there is a winner (Player or Opponent), a tie,
    /// or an incomplete game.
    ///
    /// @param *board - pointer to 3x3 char matrix
    /// @param rows - number of rows in the board
    /// @param col - number of columns in the board
    /// @return the game result of type Game::GameResultType
    static GameResultType checkGameResult(char *board, int rows, int cols);

    /// Getter for the current 3x3 Game Board matrix
    char *getBoard();

    /// Print out current Game Board with Xs and Os
    void printCurrentGameBoard();

private:
    const int BOARD_SIZE = 3;
    const char EMPTY = ' ';
    char board[3][3] = {
        {' ', ' ', ' '},
        {' ', ' ', ' '},
        {' ', ' ', ' '}};
    Opponent *opponent = new Opponent();
    GameResultType gameResult = GameResultType::Ongoing;

    /// Run a round of turns.
    ///
    /// Runs Player turn, and if game is not finished, runs the
    /// Opponent turn.
    void playRound();

    /// Runs Player Turn
    ///
    /// Accepts input for row and column for player moves in the
    /// form "# #" for <row column>. Input parsing is based on
    /// https://www.geeksforgeeks.org/extract-integers-string-c/
    void takePlayerTurn();

    /// Runs Opponent Turn
    ///
    /// Opponent evaluates current board, and the game grabs
    /// results and places opponent 'O'
    void takeOpponentTurn();
};
```

```cpp
// Main Game Object for TicTacToe

#include "game.hpp"
#include <iostream>
#include <sstream>

using namespace std;

// Run Main Gameplay Loop
void Game::playGame()
{
    // Main Gameplay Loop
    do
    {
        printCurrentGameBoard();
        playRound();
        this->gameResult = checkGameResult(*this->board, BOARD_SIZE, BOARD_SIZE);
    } while (this->gameResult == Game::GameResultType::Ongoing);

    printCurrentGameBoard();

    // Print game results
    switch (this->gameResult)
    {
    case Game::GameResultType::Player:
        cout << "You Win!" << endl;
        break;
    case Game::GameResultType::Opponent:
        cout << "You Lose!" << endl;
        break;
    default:
        cout << "It's a Tie." << endl;
        break;
    }
} // End function playGame

// Print out current Game Board with Xs and Os
void Game::printCurrentGameBoard()
{
    cout << "\nCurrent Board:" << endl;
    cout << this->board[0][0] << "|" << this->board[0][1] << "|" << this->board[0][2]
<< endl;
    cout << "_ _ _" << endl;
    cout << this->board[1][0] << "|" << this->board[1][1] << "|" << this->board[1][2]
<< endl;
    cout << "_ _ _" << endl;
    cout << this->board[2][0] << "|" << this->board[2][1] << "|" << this->board[2][2]
<< "\n"
         << endl;
} // End function prinCurrentGameBoard

// Check the status of a Game Board
Game::GameResultType Game::checkGameResult(char *board, int rows, int cols)
{
    // Check Rows
    for (int row = 0; row < rows; ++row)
    {
        char *rowLoc = board + (row * cols);
        if (*(rowLoc + 0) == *(rowLoc + 1) &&
            *(rowLoc + 1) == *(rowLoc + 2))
        {
            if (*(board + (row * cols) + 0) == 'X')
            {
                return Game::GameResultType::Player;
            }
            else if (*(board + (row * cols) + 0) == 'O')
            {
                return Game::GameResultType::Opponent;
            }
        }
    }

    // Check Columns
    for (int col = 0; col < cols; ++col)
    {
        char *colLoc = board + col;
        if (*(colLoc + (0 * cols)) == *(colLoc + (1 * cols)) &&
            *(colLoc + (1 * cols)) == *(colLoc + (2 * cols)))
        {
            if (*(board + (0 * cols) + col) == 'X')
            {
                return Game::GameResultType::Player;
            }
            else if (*(board + (0 * cols) + col) == 'O')
            {
                return Game::GameResultType::Opponent;
            }
        }
    }

    // Check Diagonals
    if ((*(board + (0 * cols) + 0) == *(board + (1 * cols) + 1) &&
         *(board + (1 * cols) + 1) == *(board + (2 * cols) + 2)) ||
        (*(board + (0 * cols) + 2) == *(board + (1 * cols) + 1) &&
         *(board + (1 * cols) + 1) == *(board + (2 * cols) + 0)))
    {
        if (*(board + (1 * cols) + 1) == 'X')
        {
            return Game::GameResultType::Player;
        }
        else if (*(board + (1 * cols) + 1) == 'O')
        {
            return Game::GameResultType::Opponent;
        }
    }

    // Check For Tie
    for (int row = 0; row < rows; ++row)
    {
        for (int col = 0; col < cols; ++col)
        {
            if (' ' == *(board + (row * cols) + col))
            {
                return Game::GameResultType::Ongoing;
            }
        }
    }
    return Game::GameResultType::Tie;
} // End function checkGameResult

// Getter for the current 3x3 Game Board matrix
char *Game::getBoard()
{
    return *this->board;
}

// Run a round of turns.
void Game::playRound()
{
    takePlayerTurn();
    this->gameResult = checkGameResult(*this->board, BOARD_SIZE, BOARD_SIZE);
    if (Game::GameResultType::Ongoing == this->gameResult)
    {
        takeOpponentTurn();
    }
} // End function playRound

// Runs Player Turn
void Game::takePlayerTurn()
{
    // Run loop until a valid move is made
    while (true)
    {
        int row, column;
        cout << "Your Move: ";
        string result;
        getline(cin, result);
        // Verify the form is <char char>
        if (!(result.length() == 3))
        {
            cout << "Please Submit Your move in the form <row column>, i.e.:" << endl;
            cout << "1 2" << endl;
            continue;
        }
        else
        {
            // Iterate through input string, and find integers
            stringstream ss;
            ss << result;
            string temp;
            int found[2] = {0, 0};
            int loc = 0;
            while (!ss.eof())
            {
                ss >> temp;
                if (stringstream(temp) >> found[loc])
                {
                    ++loc;
                }
            }
            row = found[0];
            column = found[1];
        }

        // Validate integers are valid spaces
        if (row <= 0 || row > BOARD_SIZE ||
            column <= 0 || column > BOARD_SIZE)
        {
            cout << "Make sure your values are between 1 and 3." << endl;
        }
        else
        {
            // Validate that move location is empty
            if (!(this->board[row - 1][column - 1] == EMPTY))
            {
                cout << "That Space is already Played." << endl;
            }
            else
            {
                this->board[row - 1][column - 1] = 'X';
                break;
            }
        }
    }
} // End function takePlayerTurn

// Runs Opponent Turn
void Game::takeOpponentTurn()
{
    cout << "Opponent Turn" << endl;
    bool result = false;
    result = this->opponent->chooseMove(*this->board, BOARD_SIZE, BOARD_SIZE);

    if (result)
    {
        int row, col = 0;
        row = this->opponent->getRow();
        col = this->opponent->getColumn();
        this->board[row][col] = 'O';
    }
} // End function takeOpponentTurn
```

```cpp
#pragma once

/// @file
/// @author Nathan Roe
/// Opponent for Single-Player TicTacToe Game
///
/// Evaluates a 3x3 matrix to find a move, then
/// saves selected location. Chosen location can
/// be accessed with getRow and getColumn
class Opponent
{
public:
    /// Selects move based on a Game Board
    ///
    /// Evaluates the current status of a 3x3 matrix to
    /// select the location of the Opponent's next move
    ///
    /// @param *board - pointer to 3x3 char matrix
    /// @param rows - number of rows in the board
    /// @param col - number of columns in the board
    /// @return a boolean of whether a valid move was found
    bool chooseMove(char *board, int rows, int cols);

    /// Gets the selected row for the Opponent's next move
    ///
    /// @return an int of the Opponent's chosen row
    int getRow();

    /// Gets the selected column for the Opponent's next move
    ///
    /// @return an int of the Opponent's chosen column
    int getColumn();

private:
    int row = 0;
    int column = 0;
    const int BOARD_SIZE = 3;
    const char EMPTY = ' ';
    char board[3][3] = {
        {' ', ' ', ' '},
        {' ', ' ', ' '},
        {' ', ' ', ' '}};

    /// Search for possible winning moves
    ///
    /// Tests move locations to see if any will win game
    bool attack();

    /// Search for possible Player winning moves
    ///
    /// Tests move locations to see if any spaces will
    /// win for the player, and block that move
    bool defend();

    /// Seaches for first open space
    bool selectRandom();
};
```

```cpp
#include "opponent.hpp"
#include "../Game/game.hpp"
#include <iostream>

using namespace std;

// Gets the selected row for the Opponent's next move
int Opponent::getRow()
{
    return this->row;
} // End function getRow

// Gets the selected column for the Opponent's next move
int Opponent::getColumn()
{
    return this->column;
} // End function getColumn

// Selects move based on a given Game Board
bool Opponent::chooseMove(char *board, int rows, int cols)
{
    // Verify the board is the correct size
    if (!(rows == BOARD_SIZE) or !(cols == BOARD_SIZE))
    {
        return false;
    }

    // Make a copy of the board
    for (int row = 0; row < rows; ++row)
    {
        for (int col = 0; col < cols; ++col)
        {
            this->board[row][col] = *(board + (row * cols) + col);
        }
    }

    bool moveChosen = false;

    // Try to find a winning move
    moveChosen = attack();

    // If no winning moves, protect against a Player win
    if (!moveChosen)
    {
        moveChosen = defend();
    }

    // If player has no winning moves, select first open space
    if (!moveChosen)
    {
        moveChosen = selectRandom();
    }

    return moveChosen;
}

// Search for possible winning moves
```

```cpp
bool Opponent::attack()
{
    // Iterate over board
    for (int row = 0; row < BOARD_SIZE; ++row)
    {
        for (int col = 0; col < BOARD_SIZE; ++col)
        {
            if (this->board[row][col] == EMPTY)
            {
                // Test open spaces to see if they are winning placements
                this->board[row][col] = 'O';
                Game::GameResultType result;
                result = Game::checkGameResult(*this->board, BOARD_SIZE, BOARD_SIZE);
                if (result == Game::GameResultType::Opponent)
                {
                    // If space wins, set selection
                    this->row = row;
                    this->column = col;
                    cout << "Attacking" << endl;
                    return true;
                }
                else
                {
                    // Return board to starting configuration
                    this->board[row][col] = EMPTY;
                }
            }
        }
    }
    return false;
} // End function attack

bool Opponent::defend()
{
    // Iterate over board
    for (int row = 0; row < BOARD_SIZE; ++row)
    {
        for (int col = 0; col < BOARD_SIZE; ++col)
        {
            if (this->board[row][col] == EMPTY)
            {
                // Test open spaces to see if they are winning placements
                // for the player
                this->board[row][col] = 'X';
                Game::GameResultType result;
                result = Game::checkGameResult(*this->board, BOARD_SIZE, BOARD_SIZE);
                if (result == Game::GameResultType::Player)
                {
                    // If space wins for player, set selection to block
                    this->board[row][col] = 'O';
                    this->row = row;
                    this->column = col;
                    cout << "Defending" << endl;
                    return true;
                }
                else
                {
                    // Return board to starting configuration
                    this->board[row][col] = EMPTY;
                }
            }
        }
    }
    return false;
} // End function defend

bool Opponent::selectRandom()
{
    // Iterate over board
    for (int row = 0; row < BOARD_SIZE; ++row)
    {
        for (int col = 0; col < BOARD_SIZE; ++col)
        {
            if (this->board[row][col] == EMPTY)
            {
                // If empty space is found, select space
                this->row = row;
                this->column = col;
                this->board[row][col] = 'O';
                cout << "Choosing Randomly" << endl;
                return true;
            }
        }
    }
    return false;
} // End function selectRandom
```

```
Welcome to Tic Tac Toe
You will play as x and you're up First
Enter your Move as <row column>, ie '1 2'

Current Board:
 | |
_ _ _
 | |
_ _ _
 | |

Your Move: 1 1
Opponent Turn
Choosing Randomly

Current Board:
X|O|
_ _ _
 | |
_ _ _
 | |

Your Move: 2 2
Opponent Turn
Defending

Current Board:
X|O|
_ _ _
 |X|
_ _ _
 | |O

Your Move: 3 1
Opponent Turn
Defending

Current Board:
X|O|O
_ _ _
 |X|
_ _ _
X| |O

Your Move: 2 1
Current Board:
X|O|O
_ _ _
X|X|
_ _ _
X| |O

You Win!
Game Over!
```

```
Welcome to Tic Tac Toe
You will play as x and you're up First
Enter your Move as <row column>, ie '1 2'

Current Board:
 | |
_ _ _
 | |
_ _ _
 | |

Your Move: 1 2
Opponent Turn
Choosing Randomly

Current Board:
O|X|
_ _ _
 | |
_ _ _
 | |

Your Move: 1 3
Opponent Turn
Choosing Randomly

Current Board:
O|X|X
_ _ _
O| |
_ _ _
 | |

Your Move: 2 3
Opponent Turn
Attacking

Current Board:
O|X|X
_ _ _
O| |X
_ _ _
O| |

You Lose!
Game Over!
```

```
Welcome to Tic Tac Toe
You will play as x and you're up First
Enter your Move as <row column>, ie '1 2'

Current Board:
 | |
_ _ _
 | |
_ _ _
 | |

Your Move: 2 2
Opponent Turn
Choosing Randomly

Current Board:
O| |
_ _ _
 |X|
_ _ _
 | |

Your Move: 2 1
Opponent Turn
Defending

Current Board:
O| |
_ _ _
X|X|O
_ _ _
 | |

Your Move: 1 2
Opponent Turn
Defending

Current Board:
O|X|
_ _ _
X|X|O
_ _ _
 |O|

Your Move: 3 1
Opponent Turn
Defending

Current Board:
O|X|O
_ _ _
X|X|O
_ _ _
X|O|

Your Move: 3 3

```

```
Current Board:
O|X|O
_ _ _
X|X|O
_ _ _
X|O|X

It's a Tie.
Game Over!
```

```
Welcome to Tic Tac Toe
You will play as x and you're up First
Enter your Move as <row column>, ie '1 2'

Current Board:
 | |
_ _ _
 | |
_ _ _
 | |

Your Move: 1 a
Make sure your values are between 1 and 3.
Your Move: 4 4
Make sure your values are between 1 and 3.
Your Move: asdf
Please Submit Your move in the form <row column>, i.e.:
1 2
Your Move: 12 1
Please Submit Your move in the form <row column>, i.e.:
1 2
Your Move: 01 1
Please Submit Your move in the form <row column>, i.e.:
1 2
Your Move: 1 1
Opponent Turn
Choosing Randomly

Current Board:
X|O|
_ _ _
 | |
_ _ _
 | |

Your Move: 1 2
That Space is already Played.
Your Move: 1 3
Opponent Turn
Choosing Randomly

Current Board:
X|O|X
_ _ _
O| |
_ _ _
 | |

Your Move: 2 2
Opponent Turn
Defending

Current Board:
X|O|X
_ _ _
O|X|
_ _ _
```

```
O| |

Your Move: 2 2
That Space is already Played.
Your Move: 3 3

Current Board:
X|O|X
_ _ _
O|X|
_ _ _
O| |X

You Win!
Game Over!
```