

Design

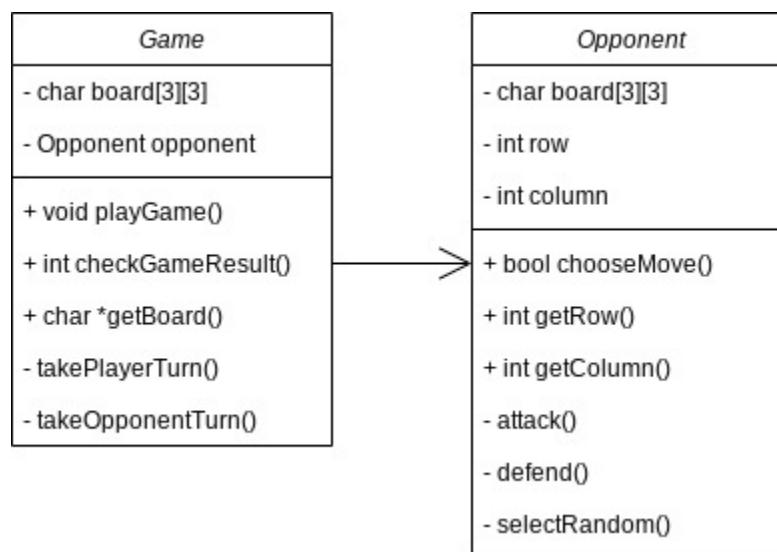
The assignment requirements for the TicTacToe project require building of a single-player game of TicTacToe, where the player plays as X and the computer plays as O. To do this, I will make two classes; the first will include all of the Game rules, and the second will be the Opponent class. The first 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 (takeOpponentTurn)
- 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 (chooseMove)
- Block user turns (defend)
- Attempt winning moves (attack)
- Otherwise, select random location (selectRandom)



```
1 /// @file
2 /// @author Nathan Roe
3 /// Single-Player Tic-Tac-Toe Game
4 ///
5 /// Player plays as 'X' and enters moves in the form "# #"
6 /// for the row, column of their move, and the software
7 /// plays as 'O'.
8
9 #include "Game/game.hpp"
10 #include <iostream>
11
12 using namespace std;
13
14 /// Main function; runs gameplay loop
15 int main()
16 {
17     Game *newGame = new Game();
18
19     cout << "Welcome to Tic Tac Toe" << endl;
20     cout << "You will play as x and you're up First" << endl;
21     cout << "Enter your Move as <row column>, ie '1 2'" << endl;
22     // Kick-off new game
23     newGame->playGame();
24
25     cout << "Game Over!" << endl;
26
27     return 0;
28 } // End function main
```

```
1 #pragma once
2 #include "../Opponent/opponent.hpp"
3
4 /// @file
5 /// @author Nathan Roe
6 /// Main Game Object for Tic-Tac-Toe
7 ///
8 /// Player plays as 'X' and enters moves in the form "# #"
9 /// for the row, column of their move, and the software
10 /// plays as 'O'.
11 class Game
12 {
13 public:
14     /// Enumeration of Game States
15     ///
16     /// Player - Player Win
17     /// Opponent - Opponent Win
18     /// Tie - Board filled in with no win
19     /// Ongoing - Incomplete Game
20     enum class GameResultType
21     {
22         Player,
23         Opponent,
24         Tie,
25         Ongoing
26     };
27
28     /// Run Main Gameplay Loop
29     ///
30     /// Runs a set of moves from player/opponent, and evaluates
31     /// gamestate. Print results when game is complete
32     void playGame();
33
34     /// Check the status of a Game Board
35     ///
36     /// Checks to see the current status of a 3x3 matrix, to
37     /// Evaluate if there is a winner (Player or Opponent), a tie,
38     /// or an incomplete game.
39     ///
40     /// @param *board - pointer to 3x3 char matrix
41     /// @param rows - number of rows in the board
42     /// @param col - number of columns in the board
43     /// @return the game result of type Game::GameResultType
44     static GameResultType checkGameResult(char *board, int rows, int cols);
45
46     /// Getter for the current 3x3 Game Board matrix
47     char *getBoard();
48
49     /// Print out current Game Board with Xs and Os
50     void printCurrentGameBoard();
51
52 private:
53     const int BOARD_SIZE = 3;
54     const char EMPTY = ' ';
55     char board[3][3] = {
56         {' ', ' ', ' '},
57         {' ', ' ', ' '},
58         {' ', ' ', ' '},
59     };
```

```
58     {' ', ' ', ' ', ' '}}};
59     Opponent *opponent = new Opponent();
60     GameResultType gameResult = GameResultType::Ongoing;
61
62     /// Run a round of turns.
63     ///
64     /// Runs Player turn, and if game is not finished, runs the
65     /// Opponent turn.
66     void playRound();
67
68     /// Runs Player Turn
69     ///
70     /// Accepts input for row and column for player moves in the
71     /// form "# #" for <row column>. Input parsing is based on
72     /// https://www.geeksforgeeks.org/extract-integers-string-c/
73     void takePlayerTurn();
74
75     /// Runs Opponent Turn
76     ///
77     /// Opponent evaluates current board, and the game grabs
78     /// results and places opponent 'O'
79     void takeOpponentTurn();
80 };
```

```
1 // Main Game Object for TicTacToe
2
3 #include "game.hpp"
4 #include <iostream>
5 #include <sstream>
6
7 using namespace std;
8
9 // Run Main Gameplay Loop
10 void Game::playGame()
11 {
12     // Main Gameplay Loop
13     do
14     {
15         printCurrentGameBoard();
16         playRound();
17         this->gameResult = checkGameResult(*this->board, BOARD_SIZE, BOARD_SIZE);
18     } while (this->gameResult == Game::GameResultType::Ongoing);
19
20     printCurrentGameBoard();
21
22     // Print game results
23     switch (this->gameResult)
24     {
25     case Game::GameResultType::Player:
26         cout << "You Win!" << endl;
27         break;
28     case Game::GameResultType::Opponent:
29         cout << "You Lose!" << endl;
30         break;
31     default:
32         cout << "It's a Tie." << endl;
33         break;
34     }
35 } // End function playGame
36
37 // Print out current Game Board with Xs and Os
38 void Game::printCurrentGameBoard()
39 {
40     cout << "\nCurrent Board:" << endl;
41     cout << this->board[0][0] << "|" << this->board[0][1] << "|" << this->board[0][2]
42 << endl;
43     cout << "_ _ _" << endl;
44     cout << this->board[1][0] << "|" << this->board[1][1] << "|" << this->board[1][2]
45 << endl;
46     cout << "_ _ _" << endl;
47     cout << this->board[2][0] << "|" << this->board[2][1] << "|" << this->board[2][2]
48 << "\n"
49 << endl;
50 } // End function printCurrentGameBoard
51
52 // Check the status of a Game Board
53 Game::GameResultType Game::checkGameResult(char *board, int rows, int cols)
54 {
55     // Check Rows
56     for (int row = 0; row < rows; ++row)
```

```
55     char *rowLoc = board + (row * cols);
56     if (*(rowLoc + 0) == *(rowLoc + 1) &&
57         *(rowLoc + 1) == *(rowLoc + 2))
58     {
59         if (*(board + (row * cols) + 0) == 'X')
60         {
61             return Game::GameResultType::Player;
62         }
63         else if (*(board + (row * cols) + 0) == 'O')
64         {
65             return Game::GameResultType::Opponent;
66         }
67     }
68 }
69
70 // Check Columns
71 for (int col = 0; col < cols; ++col)
72 {
73     char *colLoc = board + col;
74     if (*(colLoc + (0 * cols)) == *(colLoc + (1 * cols)) &&
75         *(colLoc + (1 * cols)) == *(colLoc + (2 * cols)))
76     {
77         if (*(board + (0 * cols) + col) == 'X')
78         {
79             return Game::GameResultType::Player;
80         }
81         else if (*(board + (0 * cols) + col) == 'O')
82         {
83             return Game::GameResultType::Opponent;
84         }
85     }
86 }
87
88 // Check Diagonals
89 if ((*(board + (0 * cols) + 0) == *(board + (1 * cols) + 1) &&
90     *(board + (1 * cols) + 1) == *(board + (2 * cols) + 2)) ||
91     (*(board + (0 * cols) + 2) == *(board + (1 * cols) + 1) &&
92     *(board + (1 * cols) + 1) == *(board + (2 * cols) + 0)))
93 {
94     if (*(board + (1 * cols) + 1) == 'X')
95     {
96         return Game::GameResultType::Player;
97     }
98     else if (*(board + (1 * cols) + 1) == 'O')
99     {
100         return Game::GameResultType::Opponent;
101     }
102 }
103
104 // Check For Tie
105 for (int row = 0; row < rows; ++row)
106 {
107     for (int col = 0; col < cols; ++col)
108     {
109         if (' ' == *(board + (row * cols) + col))
110         {
111             return Game::GameResultType::Ongoing;
```

```
112     }
113 }
114 }
115     return Game::GameResultType::Tie;
116 } // End function checkGameResult
117
118 // Getter for the current 3x3 Game Board matrix
119 char *Game::getBoard()
120 {
121     return *this->board;
122 }
123
124 // Run a round of turns.
125 void Game::playRound()
126 {
127     takePlayerTurn();
128     this->gameResult = checkGameResult(*this->board, BOARD_SIZE, BOARD_SIZE);
129     if (Game::GameResultType::Ongoing == this->gameResult)
130     {
131         takeOpponentTurn();
132     }
133 } // End function playRound
134
135 // Runs Player Turn
136 void Game::takePlayerTurn()
137 {
138     // Run loop until a valid move is made
139     while (true)
140     {
141         int row, column;
142         cout << "Your Move: ";
143         string result;
144         getline(cin, result);
145         // Verify the form is <char char>
146         if (!(result.length() == 3))
147         {
148             cout << "Please Submit Your move in the form <row column>, i.e.:" << endl;
149             cout << "1 2" << endl;
150             continue;
151         }
152         else
153         {
154             // Iterate through input string, and find integers
155             stringstream ss;
156             ss << result;
157             string temp;
158             int found[2] = {0, 0};
159             int loc = 0;
160             while (!ss.eof())
161             {
162                 ss >> temp;
163                 if (stringstream(temp) >> found[loc])
164                 {
165                     ++loc;
166                 }
167             }
168             row = found[0];
```

```
169         column = found[1];
170     }
171
172     // Validate integers are valid spaces
173     if (row <= 0 || row > BOARD_SIZE ||
174         column <= 0 || column > BOARD_SIZE)
175     {
176         cout << "Make sure your values are between 1 and 3." << endl;
177     }
178     else
179     {
180         // Validate that move location is empty
181         if (!(this->board[row - 1][column - 1] == EMPTY))
182         {
183             cout << "That Space is already Played." << endl;
184         }
185         else
186         {
187             this->board[row - 1][column - 1] = 'X';
188             break;
189         }
190     }
191 }
192 } // End function takePlayerTurn
193
194 // Runs Opponent Turn
195 void Game::takeOpponentTurn()
196 {
197     cout << "Opponent Turn" << endl;
198     bool result = false;
199     result = this->opponent->chooseMove(*this->board, BOARD_SIZE, BOARD_SIZE);
200
201     if (result)
202     {
203         int row, col = 0;
204         row = this->opponent->getRow();
205         col = this->opponent->getColumn();
206         this->board[row][col] = 'O';
207     }
208 } // End function takeOpponentTurn
```



```
1 #pragma once
2
3 /// @file
4 /// @author Nathan Roe
5 /// Opponent for Single-Player TicTacToe Game
6 ///
7 /// Evaluates a 3x3 matrix to find a move, then
8 /// saves selected location. Chosen location can
9 /// be accessed with getRow and getColumn
10 class Opponent
11 {
12 public:
13     /// Selects move based on a Game Board
14     ///
15     /// Evaluates the current status of a 3x3 matrix to
16     /// select the location of the Opponent's next move
17     ///
18     /// @param *board - pointer to 3x3 char matrix
19     /// @param rows - number of rows in the board
20     /// @param col - number of columns in the board
21     /// @return a boolean of whether a valid move was found
22     bool chooseMove(char *board, int rows, int cols);
23
24     /// Gets the selected row for the Opponent's next move
25     ///
26     /// @return an int of the Opponent's chosen row
27     int getRow();
28
29     /// Gets the selected column for the Opponent's next move
30     ///
31     /// @return an int of the Opponent's chosen column
32     int getColumn();
33
34 private:
35     int row = 0;
36     int column = 0;
37     const int BOARD_SIZE = 3;
38     const char EMPTY = ' ';
39     char board[3][3] = {
40         {' ', ' ', ' '},
41         {' ', ' ', ' '},
42         {' ', ' ', ' '}};
43
44     /// Search for possible winning moves
45     ///
46     /// Tests move locations to see if any will win game
47     bool attack();
48
49     /// Search for possible Player winning moves
50     ///
51     /// Tests move locations to see if any spaces will
52     /// win for the player, and block that move
53     bool defend();
54
55     /// Seaches for first open space
56     bool selectRandom();
57 };
```

```
1 #include "opponent.hpp"
2 #include "../Game/game.hpp"
3 #include <iostream>
4
5 using namespace std;
6
7 // Gets the selected row for the Opponent's next move
8 int Opponent::getRow()
9 {
10     return this->row;
11 } // End function getRow
12
13 // Gets the selected column for the Opponent's next move
14 int Opponent::getColumn()
15 {
16     return this->column;
17 } // End function getColumn
18
19 // Selects move based on a given Game Board
20 bool Opponent::chooseMove(char *board, int rows, int cols)
21 {
22     // Verify the board is the correct size
23     if (!(rows == BOARD_SIZE) or !(cols == BOARD_SIZE))
24     {
25         return false;
26     }
27
28     // Make a copy of the board
29     for (int row = 0; row < rows; ++row)
30     {
31         for (int col = 0; col < cols; ++col)
32         {
33             this->board[row][col] = *(board + (row * cols) + col);
34         }
35     }
36
37     bool moveChosen = false;
38
39     // Try to find a winning move
40     moveChosen = attack();
41
42     // If no winning moves, protect against a Player win
43     if (!moveChosen)
44     {
45         moveChosen = defend();
46     }
47
48     // If player has no winning moves, select first open space
49     if (!moveChosen)
50     {
51         moveChosen = selectRandom();
52     }
53
54     return moveChosen;
55 }
56
57 // Search for possible winning moves
```

```
58 bool Opponent::attack()
59 {
60     // Iterate over board
61     for (int row = 0; row < BOARD_SIZE; ++row)
62     {
63         for (int col = 0; col < BOARD_SIZE; ++col)
64         {
65             if (this->board[row][col] == EMPTY)
66             {
67                 // Test open spaces to see if they are winning placements
68                 this->board[row][col] = 'O';
69                 Game::GameResultType result;
70                 result = Game::checkGameResult(*this->board, BOARD_SIZE, BOARD_SIZE);
71                 if (result == Game::GameResultType::Opponent)
72                 {
73                     // If space wins, set selection
74                     this->row = row;
75                     this->column = col;
76                     cout << "Attacking" << endl;
77                     return true;
78                 }
79                 else
80                 {
81                     // Return board to starting configuration
82                     this->board[row][col] = EMPTY;
83                 }
84             }
85         }
86     }
87     return false;
88 } // End function attack
89
90 bool Opponent::defend()
91 {
92     // Iterate over board
93     for (int row = 0; row < BOARD_SIZE; ++row)
94     {
95         for (int col = 0; col < BOARD_SIZE; ++col)
96         {
97             if (this->board[row][col] == EMPTY)
98             {
99                 // Test open spaces to see if they are winning placements
100                 // for the player
101                 this->board[row][col] = 'X';
102                 Game::GameResultType result;
103                 result = Game::checkGameResult(*this->board, BOARD_SIZE, BOARD_SIZE);
104                 if (result == Game::GameResultType::Player)
105                 {
106                     // If space wins for player, set selection to block
107                     this->board[row][col] = 'O';
108                     this->row = row;
109                     this->column = col;
110                     cout << "Defending" << endl;
111                     return true;
112                 }
113                 else
114                 {
```

```
115         // Return board to starting configuration
116         this->board[row][col] = EMPTY;
117     }
118 }
119 }
120 }
121 return false;
122 } // End function defend
123
124 bool Opponent::selectRandom()
125 {
126     // Iterate over board
127     for (int row = 0; row < BOARD_SIZE; ++row)
128     {
129         for (int col = 0; col < BOARD_SIZE; ++col)
130         {
131             if (this->board[row][col] == EMPTY)
132             {
133                 // If empty space is found, select space
134                 this->row = row;
135                 this->column = col;
136                 this->board[row][col] = 'O';
137                 cout << "Choosing Randomly" << endl;
138                 return true;
139             }
140         }
141     }
142     return false;
143 } // End function selectRandom
```

```
1 Welcome to Tic Tac Toe
2 You will play as x and you're up First
3 Enter your Move as <row column>, ie '1 2'
4
5 Current Board:
6 | |
7 - - -
8 | |
9 - - -
10 | |
11
12 Your Move: 1 1
13 Opponent Turn
14 Choosing Randomly
15
16 Current Board:
17 X|O|
18 - - -
19 | |
20 - - -
21 | |
22
23 Your Move: 2 2
24 Opponent Turn
25 Defending
26
27 Current Board:
28 X|O|
29 - - -
30 |X|
31 - - -
32 | |O
33
34 Your Move: 3 1
35 Opponent Turn
36 Defending
37
38 Current Board:
39 X|O|O
40 - - -
41 |X|
42 - - -
43 X| |O
44
45 Your Move: 2 1
46 Current Board:
47 X|O|O
48 - - -
49 X|X|
50 - - -
51 X| |O
52
53 You Win!
54 Game Over!
```

```
1 Welcome to Tic Tac Toe
2 You will play as x and you're up First
3 Enter your Move as <row column>, ie '1 2'
4
5 Current Board:
6 | |
7 - - -
8 | |
9 - - -
10 | |
11
12 Your Move: 1 2
13 Opponent Turn
14 Choosing Randomly
15
16 Current Board:
17 O|X|
18 - - -
19 | |
20 - - -
21 | |
22
23 Your Move: 1 3
24 Opponent Turn
25 Choosing Randomly
26
27 Current Board:
28 O|X|X
29 - - -
30 O| |
31 - - -
32 | |
33
34 Your Move: 2 3
35 Opponent Turn
36 Attacking
37
38 Current Board:
39 O|X|X
40 - - -
41 O| |X
42 - - -
43 O| |
44
45 You Lose!
46 Game Over!
```

```
1 Welcome to Tic Tac Toe
2 You will play as x and you're up First
3 Enter your Move as <row column>, ie '1 2'
4
5 Current Board:
6 | |
7 - - -
8 | |
9 - - -
10 | |
11
12 Your Move: 2 2
13 Opponent Turn
14 Choosing Randomly
15
16 Current Board:
17 0| |
18 - - -
19 |X|
20 - - -
21 | |
22
23 Your Move: 2 1
24 Opponent Turn
25 Defending
26
27 Current Board:
28 0| |
29 - - -
30 X|X|0
31 - - -
32 | |
33
34 Your Move: 1 2
35 Opponent Turn
36 Defending
37
38 Current Board:
39 0|X|
40 - - -
41 X|X|0
42 - - -
43 |0|
44
45 Your Move: 3 1
46 Opponent Turn
47 Defending
48
49 Current Board:
50 0|X|0
51 - - -
52 X|X|0
53 - - -
54 X|0|
55
56 Your Move: 3 3
57
```

```
58 Current Board:
59 0|X|0
60  _ _ _
61 X|X|0
62  _ _ _
63 X|0|X
64
65 It's a Tie.
66 Game Over!
```



```
1 Welcome to Tic Tac Toe
2 You will play as x and you're up First
3 Enter your Move as <row column>, ie '1 2'
4
5 Current Board:
6 | |
7 - - -
8 | |
9 - - -
10 | |
11
12 Your Move: 1 a
13 Make sure your values are between 1 and 3.
14 Your Move: 4 4
15 Make sure your values are between 1 and 3.
16 Your Move: asdf
17 Please Submit Your move in the form <row column>, i.e.:
18 1 2
19 Your Move: 12 1
20 Please Submit Your move in the form <row column>, i.e.:
21 1 2
22 Your Move: 01 1
23 Please Submit Your move in the form <row column>, i.e.:
24 1 2
25 Your Move: 1 1
26 Opponent Turn
27 Choosing Randomly
28
29 Current Board:
30 X|O|
31 - - -
32 | |
33 - - -
34 | |
35
36 Your Move: 1 2
37 That Space is already Played.
38 Your Move: 1 3
39 Opponent Turn
40 Choosing Randomly
41
42 Current Board:
43 X|O|X
44 - - -
45 O| |
46 - - -
47 | |
48
49 Your Move: 2 2
50 Opponent Turn
51 Defending
52
53 Current Board:
54 X|O|X
55 - - -
56 O|X|
57 - - -
```

```
58 0| |  
59  
60 Your Move: 2 2  
61 That Space is already Played.  
62 Your Move: 3 3  
63  
64 Current Board:  
65 X|0|X  
66 _ _ _  
67 0|X|  
68 _ _ _  
69 0| |X  
70  
71 You Win!  
72 Game Over!
```