VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

INTRODUCTION TO PROGRAMMING - CO1003

ASSIGNMENT

GOMOKU

HO CHI MINH CITY, AUGUST 2024

ASSIGNMENT SPECIFICATIONS

Version 1.0

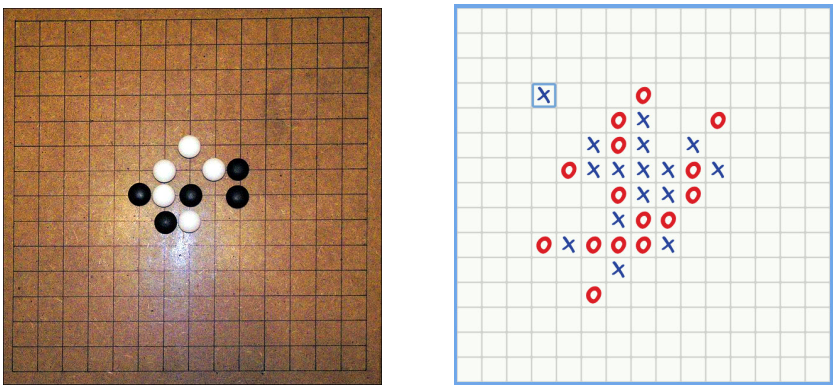
1 Outcomes

After finishing this assignment, the student is revised and can proficiently use:

- Conditional statements
- Loop statements
- Array and 2-dimensional array
- String processing
- Function and function call
- User-defined data type (struct)

2 Introduction

Gomoku, also called Five in a Row, Caro, etc., is an abstract strategy board game. It is traditionally played with Go pieces (black and white stones) on a Go board. It is played using a 15×15 board while in the past a 19×19 board was standard. Because pieces are typically not moved or removed from the board, Gomoku may also be played as a paper-and-pencil game.[1] In this assignment, you will re-implement the Gomoku game with some of the rules provided below.



(a) Gomoku on Go board (b) Gomoku on paperandpencil

Figure 1: Illustration for the game Gomoku

3 Game rules

For the sake of simplicity, the rules used in this assignment will be applied to the paper version of Gomoku (the move is denoted by X or O).

In the regular rule of Gomoku, two players will alternately place a stone on an empty square. The first player will start with stone X, the next player will use stone O and take turns placing stones down the board. The winner is the first player to form a sequence of **five (5)** moves (which are stones) in a row, unbroken horizontally, vertically or diagonally. The figure 2 depicts some of the winning states of the game.

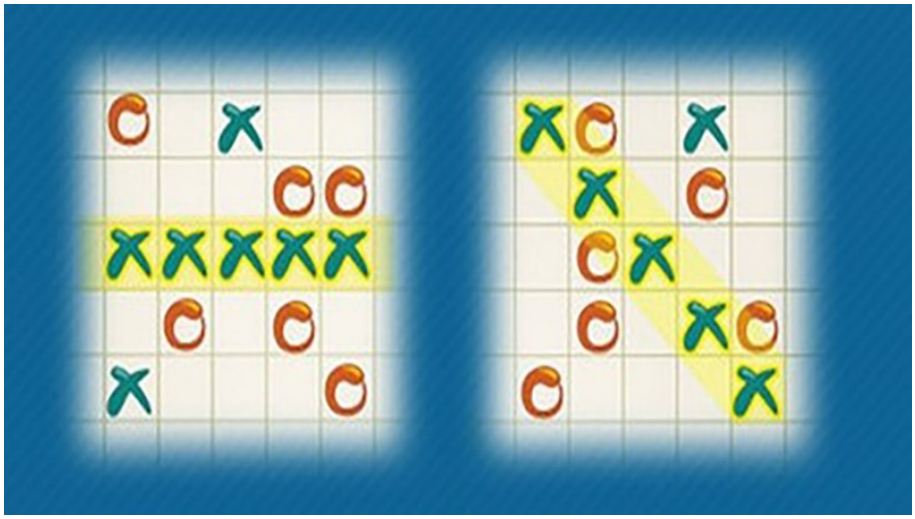
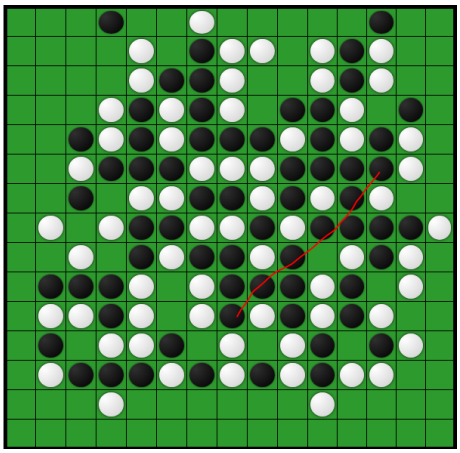


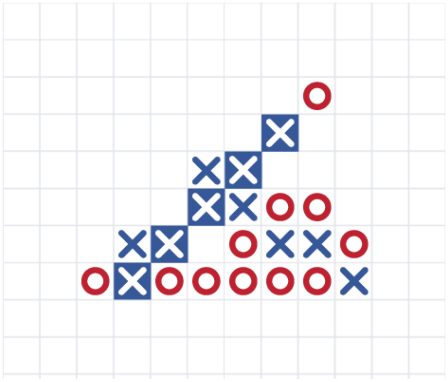
Figure 2: Winning states of Gomoku

Also, in this assignment, we will add 2 more rules to the game:

- Supplementary Rule 1 (6 moves or more): A consecutive sequence of moves with **more than five (5)** moves will **not be considered as a win**.
- Supplementary Rule 2 (2-headed Block): A sequence of five (5) consecutive moves that is **blocked by both ends** by the opponent's 2 moves will **not be considered as a win**.



(a) Illustration for the 1st complement rule



(b) Illustration for the 2nd complement rule

Figure 3: Supplementary Rules

In the figure 3(a), the red line indicates that player Black, even with 6 moves, is not considered a winner. In the figure 3(b), player O is not considered to have won even though he has made 5 consecutive O moves because he was blocked by player X with move X at both ends.

4 Program description

This section will present the information that students need to know about the program before starting to work on the requirements of the assignment. In this assignment, students are provided with a source code that has already been **implemented** including: function **main()**, function **startGame()**, function **displayBoard( )** (along with the displayBoard **lineString()** add-in function), function **displayBoardSimple()**, the enum **Stone** type and some other global variables.

When the program starts running, the console screen will show the following:

```
Welcome to Gomoku!
first.  Play games
2.  History
3.  Exit
Please select mode [1/2/3]:
```

The program will wait for the user's input to be an integer and for each corresponding input:

- If input is 1: The program will call the function **startGame()** and start the turn-based mode of the game Gomoku.
- If input is 2: The program will call the function **displayHistory()** and start the replay mode of a Gomoku game.
- If input is 3: Program will terminate.
- If the input is invalid: The program will print `Illegal input, please try again:` and ask the user to enter it again.

5 Tasks

In this assignment, students need to implement a program in C language to simulate the process of playing the game Gomoku on the console screen through the game rules described above and the tasks described below. Each task is required to write a corresponding function with parameters of which are given in the description of the task request.

Before continuing below, students are required to read and understand how the **main()** and **startGame()** functions work. It is **not** recommended to understand the content of the function **displayBoard()** and **lineString()**.

5.1 Make a move (3 points)

When the user selects the turn-based game mode, the program calls the function **startGame()**. Next, the program will wait for the user to enter a move that is a position of the board.

Gomoku are played on a square board consisting of 15 rows numbered 15 to 1 (top to bottom) and 15 columns ordered from character 'a' to 'o' (left to right). A player's move on the board will be denoted by a combination ***RowColumn*** with row and column values within the valid range of the board. Some examples of valid moves: 1a, 3d, 5o, etc.

Students are asked to write a function that determines if a player's input is a valid move or not, and records the move to the board if it is valid. Specifically, the description of the function is as follows:

- Function name: **makeMove**.
- Input parameters:
  - **board[][MAX_SIZE] (type enum Stone)**: 2-dimensional array of type enum Stone, storing information about the current state of the board.

  - **size (type int)**: The size of the board.
  - **playerMove (char*)**: String containing the player's move
  - **isFirstPlayerTurn (bool)**: Logical variable used to determine whether the current turn is the turn of the first player or not

- Function requirements:

  1. The function will check if the player's move is a valid move or not. A valid move is a move that: follows the notation of a move; the row and column values are within the valid range, and the square to be placed on the board is empty.
  2. If the move is valid, the function updates the player's move on the **board** array based on the **isFirstPlayerTurn** variable. Otherwise, the function will do nothing.
  3. Returns `true` if the board is updated with a new move, otherwise returns `false`.

Example 1: Examples of **playerMove** being an invalid move:

- **32x**: invalid because a row value of 32 and a column value of 'x' are outside the valid range of the board.
- **t3**: invalid because it does not follow the notation of a move.

The function returns the value `false` in these two cases.

Example 2: Given the current state of the board as follows:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14															
13															
12															
11						O		X							
10							X	O	X						
9						O									
8															
7															
6															
5															
4															
3															
2															
1															

With playerMove = "12i" and isFirstPlayerTurn = true, the function will update the board to:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14															
13															
12									X						
11						O		X							
10							X	O	X						
9						O									
8															
7															
6															
5															
4															
3															
2															
1															

The function returns the value `true`.

5.2 Winning checking(4 points)

We continue to observe the function **startGame()**. After the move is valid and the stone has been placed on the board, the program will continue to check if the player who just made that move has won or not. The winning conditions of the game are clearly described in the section Game rules.

Students are asked to write the following function to describe the winning checking process. The function information is as follows:

- Function name: **hasWon**.
- Input parameters:
  - **board[][MAX_SIZE] (type enum Stone)**: 2-dimensional array of type enum Stone, storing information about the current state of the board.
  - **size (type int)**: The size of the board.
  - **isFirstPlayerTurn (bool)**: Logical variable used to determine whether the current turn is the turn of the first player or not
- Function requirements: The function checks if the current state of the **board** is the winning state of player 1/player 2 (based on the variable **isFirstPlayerTurn**).
- Return result: The function returns the value `true` if the **board** is in a winning state. Otherwise, returns `false`.

Example 3: Given the current state of the board as follows:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14															
13									O	O					
12									X						
11						O		X	X	X	O				
10							X	O	X		O				
9						O		X	X	X	X	X	O		
8									O		O				
7															
6															
5															
4															
3															
2															
1															

Notice that player X has 5 moves in a row from 9h to 9l. So, player X won. The function returns the result `true`.

Example 4: Given the current state of the board as follows:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14										X					
13									O	O					
12							X		X	O					
11						O		X	X	O					
10							X	O	X	O					
9						O			X	O					
8									O	X					
7															
6															
5															
4															
3															
2															
1															

Although player O has made 5 moves in a row (from 9j to 13j) but according to the 2-Headed Block rule, player O has not won because he was blocked by player X on moves 14j and 8j. So the function returns `false` because no one has won yet.

Example 5: Given the current state of the board as follows:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14															
13							O	O	O	O	O	O			
12									X			O			
11						O		X	X	X	X	O			
10							X	O	X	X	X	X	X		
9						O			X						
8									O						
7															
6															
5															
4															
3															
2															
1															

Player O is not considered to have won even though there have been 5 consecutive moves (13g to 13k, or 13h to 13l) because those moves fell within the rules **6 moves or more**. Meanwhile, player X has 5 consecutive moves from 10i to 10m so X is the winner. The function returns the result `true`.

5.3 Watch match history (3 points)

This section asks students to implement the function of reviewing a game of Gomoku by implementing the function **displayHistory()**. Know that after a match ends, the match history will be saved in the following format:

$$a_0a_1a_2...a_i...a_n$$

In which:

- a_i: Notation of a move
- n: Number of moves of the match

After this match history mode is selected, the user will enter the match history to watch

this match. The program will continue to provide users with 3 ways to control the watching process including: Watch next move, Watch previous move and Stop.

SV was asked to write the following function to allow the user to control the replay of the match based on the history entered by the user. The details of the function are as follows:

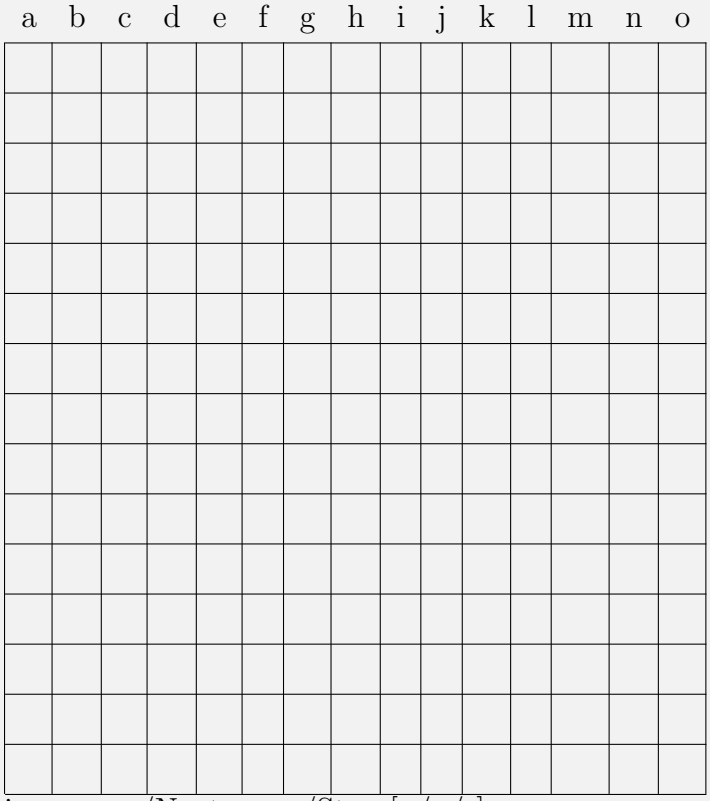
- Function name: **displayHistory**.
- Input parameters:
  - **history (char*)**: string containing match history
  - **numOfMoves (type int)**: number of moves of the match
- Function requirements:
  1. Declare a 2D array of type enum Stone (called *game*) with size 15x15 to store the match state. Initialize all elements of the array to the value NA.
  2. Displays the current state by calling the displayBoard function with the necessary arguments to display the empty board.
  3. Prints the string `inputCommand` to the screen.
  4. Reads user input as a character of type `char`.
     - If the user enters the character `'n'`: Display the board state after playing the next move (continue using the displayBoard function). However, if the current move is the last move, print the string `endOfHistory`. Then go back to step 4.
     - If the user enters the character `'p'`: Displays the board before playing the current move (using the displayBoard function). However, if the board is currently empty, `startOfGame` is printed to the screen. Then go back to step 4.
     - If the user enters the character `'s'`: Ends the function.
     - If the user enters a character other than the three above, the string `invalidInput` is printed to the screen. Then go back to step 4.
- Returns: Nothing
- Note: In this assignment, user-entered match history is **guaranteed** to be always valid and n ¡ 225.

Example 6: Here are some examples of match history string:

1. 7i7h8f8h9g10h9h9i8j6h6f9f4h6g5f7f10g11f5h5g4g10j11k6i8g11g3h2i4f4e2h1h3f2f3g5e3e3d3i (n = 34)
2. 13n10n12n8e7k9e12m7e6e8n12l12o10k14o12j12k11l9j10l9l9k8k11m13o8j (n = 25)
3. 13n10n12n8e7k9f9h7f9d8f6f7d6c7e7g8c8d6e5e5f4g9b (n = 22)

Example 7: The following example is a flow of user activity interacting with the program after the user selects the watch match history mode:

- The program prints: Please enter number of moves:
- User enters: 22
- The program prints: Please enter history:
- User enters: 13n10n12n8e7k9f9h7f9d8f6f7d6c7e7g8c8d6e5e5f4g9b
- The program prints:



Previous move/Next move/Stop [p/n/s]:

- User enters: n
- The program prints:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14															
13														X	
12															
11															
10															
9															
8															
7															
6															
5															
4															
3															
2															
1															

Previous move/Next move/Stop [p/n/s]:

- User enters: n
- The program prints:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14															
13														X	
12															
11															
10														O	
9															
8															
7															
6															
5															
4															
3															
2															
1															

Previous move/Next move/Stop [p/n/s]:

- User enters: s
- The program ends.

Example 8: Considering the match history is

`13n10n12n8e7k9e12m7e6e8n12l12o10k14o12j12k11l9j10l9l9k8k11m13o8j`

with n = 25. Here is a state of the board during playback:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14															O
13														X	
12										X	O	X	X	X	O
11												X			
10											X			O	
9						O				O					
8						O								O	
7						O					X				
6						X									
5															
4															
3															
2															
1															

After the user enters `p`, the program will print:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
15															
14															O
13														X	
12										X	O	X	X	X	O
11												X			
10											X			O	
9						O									
8						O								O	
7						O					X				
6						X									
5															
4															
3															
2															
1															

If the user enters `a`, the program will print: `Illegal input, please try again:`

The user continues to input `s`, the program terminates.

5.4 Update displayBoard function (required)

Because the output size displayed by the function **displayBoard** is very large and the system will grade the student's work by matching test cases, the student is required to replace all function calls **displayBoard** to the function call **displayBoardSimple** before submitting the work. Instead of printing the entire board like **displayBoard**, the function **displayBoardSimple** encodes the chessboard a string of numbers to speed up matching and reduce the size of the testcase. The function **displayBoardSimple** is already implemented.

6 Submission

Students download the following files from the course's e-learning site:

gomoku.cpp	Initial source code
233_FP_Assignment_Gomoku-v1.0.pdf	Assignment specifications

The file gomoku.cpp is the initial source code. Student **must** use this source code to continue writing.

When submitting assignment, students submit their work on the course's e-Learning site. Students fill in the code for the assignment like other exercises. The content to be filled in will be the entire gomoku.cpp file after it has been implemented. Students are provided with 4 places to submit their work:

- **Test place**: Students submit their work and are graded on **5 testcases** to check for possible syntax errors or basic logic errors of student work.
- **Assignment - Make a move**: Students submit their work and will be graded on **30 testcases** to check cases related to Task **Make a move**.
- **Assignment - Winning checking**: Students submit work and will be graded on **40 testcases** to check cases related to Task **Winning checking**.
- **Assignment - Watching match history**: Students submit work and will be graded on **30 testcases** to check cases related to Task **Watch match history**.

In each of the above places, excluding **Test place**, students have a maximum of **10 attempts**. For each attempt, students have **10 minutes** to submit the code and check. Only the first "Check" is scored, the following times will not be scored. The test results are displayed only after you press the "Complete" button. The highest-score attempt will be taken as the score for that task.

The student's assignment score is calculated according to the following formula:

Assignment = **(30 %)** Task Make a move + **(40 %)** Task Winning checking + **(30 %)** Task Watch match history

For example, student A gets 10 points in Task Make a move, 7 points in Task Winning checking and 8 points in Task Watch match history.:

A's assignment score = 10 * 0.3 + 7 * 0.4 + 8 * 0.3 = 8.2

Submission deadlines are announced at the submission site in the above site. By the submission deadline, the link will be automatically locked so students will not be able to submit late. To avoid possible mishaps at the time of submission, students MUST submit their papers at least **one hour** before the deadline.

Students must test their program on MinGW and Test place before submission.

7 Handling fraud

Assignment must be done BY YOURSELF. Students will be considered fraudulent if:

- There is an unusual similarity between the source code of the submissions. In this case, ALL submissions are considered fraudulent. Therefore, students must protect the source code of their assignments.
- Students do not understand the source code written by themselves, except for the parts of the code provided in the initialization program. Students can consult from any source, but make sure they understand the meaning of all the lines they write. In the case of not understanding the source code of the place they refer, students are especially warned NOT to use this source code; instead use what has been learned to write programs.
- Mistakenly submit another student's assignment on your personal account.

In the case of cheating, students will get a 0 for the entire subject (not just the assignment).

DO NOT ACCEPT ANY INTERPRETATION AND NO EXCEPTION!

After each major assignment has been submitted, a number of students will be called for random interviews to prove that the assignment has been done by themselves.

8 Change from previous version

References

[1] Gomoku, Wikipedia, `https://en.wikipedia.org/wiki/Gomoku`

——————————END——————————