Pseudo-Code: Implementation of the Draught Game in C

AUTHORED BY:

KAVIN T | I st Year AI&DS| Jan 21st 2022

Foreword :

The pseudocode is written procedure by procedure to enhance readability as opposed to jotting the steps down for the entire program as a whole.

FUNCTION: STUDENT_INTRO()

1. Store student details as character arrays student1, student2, student3 and student4
2. Display the "Welcome" message.
3. Print the student details, enclosed in the box display.
4. Wait for the user to press any key to continue.

FUNCTION: RULES_OF_THE_GAME()

1. Displays the rules of the game
2. Displays the game controls and commands
3. Wait for the user to press any key to continue.

FUNCTION: LOGFILE_GENERATION()

1. Open the game log file "Gamelog.txt" using file pointer cfPtr1 in read mode.
2. If the open operation is not successful, open a file pointer cfPtw1 in write mode and create a new file "Gamelog.txt".
3. Close the file pointer cfPtw1.
4. Open the game log file "GameCount.txt" in file pointer cfPtr2 in read mode.
5. If the open operation is not successful, open a file pointer cfPtw2 in write mode and create a new file "GameCount.txt".
6. Close the file pointer cfPtw2.
7. Close the file pointer cfPtr2.
8. Close the file pointer cfPtr1.

FUNCTION: GENERATE_GAME_NUMBER()

1. Open the file pointer cfPtr for the file gamecount.txt, if any error, display an error message.
2. Loop through the file and store the specified buffer size in a character array “line”.
3. Gamecount.txt file consists of the number of games played until now and followed by comma as a seperator.
4. Using a ‘for loop’ read the ‘line’ array and store it in the character array g_number.
5. Keep a counter ‘nc’ to check the length of the number.
6. If the length is 1, it is a single digit number. Convert the number into integer and store it in an integer variable gn.
7. If the length is 2, it is two-digit number. Convert into integer by multiplying the 1st digit by 10 and adding the second digit and store it in an integer variable gn.
8. If the length is 3, it is three-digit number. Convert it into an integer by multiplying the 1st digit by 100 and 2nd digit by 10 and adding it all to the third digit to get a single integer value. The result is stored in integer variable gn.
9. Close the file pointer cfPtr.
10. Increment the game counter by 1 by adding 1 to the variable gn.
11. Open the game count file “GameClose.txt” using the file pointer pfout in write mode, to overwrite the existing file.
12. Store the value of variable gn with suffix “,” using fprintf.
13. Close the file pointer fpout.

FUNCTION: GET_PLAYER_DETAILS()

1. Take user input of the names of player 1 and player 2 into the variable player1_name and player2_name respectively.

FUNCTION: START_CONDITION()

1. Displays the menu to start the game
2. Option 1 Starts the Game
3. Option 2 displays the Game play history
4. Option 3 Exits Game
5. Input is taken from the user and stored in the variable 'selection'. The Input is taken in character format.
6. Return the selection to the main program. Return value 49 for Play Game selection, return value 50 for Display play history selection and return value 51 for Exit Game selection.

FUNCTION: DISPLAY_GAME_HISTORY()

1. Open the game log file 'Gamelog.txt' using the file pointer cfPtr in read mode.
2. Store the line drawing ascii codes in the respective variables bs, vl, luc, ruc, hl, tt, lt, rt, pl, llc, rlc and it.
3. Display an error if the file option operations is not successful.
4. Open a while loop to iterate until the end of file.
5. Within the while loop read each line of the file gamelog.txt and store in the array 'line'.
6. Store the length of the line stored in the array in the variable 'len'
7. Using a 'for loop' read the content of variable 'line' character by character. The loop continues until the control reaches the limit of the string whose value is stored in the variable 'len'.
8. The data is separated by comma, hence comma counter variable count and variable cc is initialized. It gets incremented whenever a comma is encountered.
9. Store the value in the variable game_number until first comma is encountered in the file.
10. Store the value in the variable player1_name from the first comma to the second comma occurrence in the file.
11. Store the value in the variable player2_name from the second comma to the third comma occurrence in the file
12. Store the value in the variable winner from the third comma to the last comma occurrence in the file
13. Exit the loop if the variable cc is equal to 4.

14. Check the length of the variable game_number and store it in the variable ‘nc’
15. If the value in variable ‘nc’ is 1, it is a single digit number. Convert the number into integer and store in integer variable gn.
16. If the value in variable ‘nc’ is 2, it is two-digit number. Convert it into an integer by multiplying the 1st digit by 10 and adding the second digit and store it in integer variable gn.
17. If the value in variable nc is 3, it is three-digit number. Convert it into an integer by multiplying the 1st digit by 100 and 2nd digit by 10 and adding it all with the third digit to get a single integer value. The result is stored in integer variable gn.
18. Print the details in tabular format. Display the Game Number, the names of the Players who have Competed and the Winner.
19. Close the file point cfPtr.

FUNCTION: CHECK_KINGCHECKER()

1. Initialize the variable kc to store the result
2. Function has the input parameters active_player, player1_kingchekers, player2_kingchekers, player1_kingcheker_position, player2_kingcheker_position, checker_position.
3. Based the active player reference in the variable ‘active_player’ check whether the checker is king checker or not.
4. Using ‘for loop’ the value in variable checker_postion is checked by comparing each position in the target variable player1_kingcheker_postion or player2_kingcheker_postion.
5. The variable checker_postion has the value to be checked for king checker.
6. For active player 1, check the availability of the king checker in the array player1_kingcheker_postion
7. For active player 2, check the availability of the king checker in the array player2_kingcheker_postion.
8. If the array holds the value of checker_postion in the array, assign ‘Y’ to variable ‘kc’.
9. Return the variable ‘kc’.

FUNCTION: CHECK_FOR_KING_POSITION()

1. Variable x has the y-axis reference of the checker board.
2. For player 1, if the checker reaches row 8, the checker is converted to a King checker.
3. For player 2, if the checker reaches row 1, the checker is converted to a King checker.
4. Using If condition, the validation is done.
5. Using pointers increment the number of King Checkers for the active player, 'player1_kingchekers' for player 1 and 'player2_kingchekers' for player 2.
6. Number of existing king checkers for the player is stored in the variable 'kcnum'.
7. Append the new King Checker position to the existing list. King checker positions are stored in the variables player1_kingcheker_postion for player 1 and for player 2 it is stored in player2_kingcheker_postion.

FUNCTION: CHECK_MORE_CAPTURE()

1. New position of the checker is stored in the variable s1 and s2. Variable 's1' stores y-co-ordinate and 's2' stores the x-co-ordinate of the board co-ordinates for the checker.
2. Check whether the checker is a king checker or not using the function module check_kingcheker and the return value is stored in the variable kc. The Function returns 'Y' if the checker is king checker and 'N' if the checker is not a king checker.
3. For Player 1, Normal checker can move only in one direction that is downwards; hence the checking needs to be done only on the left and right side of the current position.
4. The vertical position is calculated. Vertical movement is incremented by adding 1 to 's1' and storing in x1 and x2.
5. The horizontal adjacent position is calculated by increasing the 's2' by 1 for the right hand side move, 'y1' stores the incremented value. Left hand side move is calculated by decreasing 1 from 's2', 'y2' stores the decremented value.
6. We need to validate the free position to move after capture. Hence the same is calculated. For right hand side movement x1 and y1 is

incremented by 1 to get the next diagonal position and it is checked for the free space using the ascii code '32'.

7. If it is free return code rt1 is assigned with 'Y'; if not free it is assigned with 'N'.
8. For left hand side movement x2 is incremented by 1 and y2 is decremented by 1 to get the next diagonal position and it is checked for the free space using the ascii code '32'.
9. If it is free return code rt2 is assigned with 'Y'; if not free it is assigned with 'N'.
10. If the checker is a king checker, there are additional directions to check. As it can move in both the directions.
11. Variables 'x3','y3','x4','y4' used for storing the co-ordinates.
12. The vertical position is calculated. Vertical movement is decremented by subtracting 1 from 's1' and storing it in x3 and x4.
13. The horizontal adjacent position is calculated by increasing 's2' by 1 for right hand side movement, 'y3' stores the incremented value. Left hand side movement is calculated by decreasing 1 from 's2', 'y4' stores the decremented value.
14. We need to validate the free position to move after capture. Hence the same is calculated. For right hand side movement x3 decremented by 1 and y3 is incremented by 1 to get the next diagonal position and it is checked for free space using the ascii code '32'.
15. If it is free return code rt3 is assigned with 'Y'; if not free it is assigned with 'N'.
16. For left hand side movement x4 is decremented by 1 and y4 is decremented by 1 to get the next diagonal position and it is checked for free space using the ascii code '32'.
17. If it is free return code rt4 is assigned with 'Y'; if not free it is assigned with 'N'.
18. If any of the return code variables rt1, rt2, rt3 or rt4 is 'Y' return 'Y'. If not return 'N'.
19. The similar exercise is done for Player 2.

FUNCTION: CAPTURE_CHECKER()

1. Number of checkers currently with player 1 and player 2 is stored in the variable ‘p1_checker’ and ‘p2_cheker’.
2. Using the function ascii_to_number the coordinates entered by the user for x-axis is converted to numerical reference.
3. Position of checker captured is stored in the variable ‘pos’.
4. Player 2’s checker can be captured by player 1, using for loop to scan through the array player2_cheker_postions and check the presence of the checker position. If it matches, update the same with ‘32’.
5. Reduce the number of checkers of player 2 by subtracting 1 from the variable player2_checkers.
6. Check whether the captured checker is a king checker or not by using check_kingchecker. If it is a king checker the function returns ‘Y’ and it returns ‘N’ for a normal checker.
7. If it is a king checker, reduce the number of king checkers from the variable player2_kingcheckers and update the position of the king checker to 0 in player2_kingcheker_position.
8. Player 1’s checker can be captured by player 2, using for loop to scan through the array player1_cheker_postions and check the presence of the checker position. If it matches, update the same with ‘32’.
9. Reduce the number of checkers of player 1 by subtracting 1 from the variable player1_checkers.
10. Check whether the captured checker is a king checker or not by using check_kingchecker. If it is a king checker the function returns ‘Y’ and it returns ‘N’ for a normal checker.
11. If it is a king checker, reduce the number of king checkers from the variable player1_kingcheckers and update the position of the king checker to 0 in player1_kingcheker_position.

FUNCTION: MOVE_CHECKER()

1. The position of the checker before move is stored in x1 and y1.
2. The position of the checker after move is stored in x2 and y2.
3. Variable 'old_pos' is updated with the old position in the same format as it is stored in the position control array.
4. Check for the king checker status by passing the 'old_pos' variable value into the function check_kingcheker. If the checker is king checker it returns 'Y', if not returns 'N'.
5. If the initiated action is 'Move' validated using the value of the 'helper_tmp' variable is equal to 'M' or 'm'. The new position variable 'new_pos' is updated with x2 and y2.
6. If the initiated action is 'Capture' validated using the value of the 'helper_tmp' variable is equal to 'C'or 'c. Further validation initiated.
7. If the checker is a normal checker. The free position after the capture is calculated by the addition of 1 to x2 for player 1 and subtracting 1 from x2 for player 2.
8. The horizontal movement position y2 is calculated, if the movement is in the right hand side y1 is less than than y2, hence y2 is added with 1. If the movement is in the left hand side y1 is greater than y2, hence y2 is subtracted with 1.
9. The new position variable 'new_pos' is updated with x2 and y2.
10. Using pointers, the new_position variable is updated with 'new_pos'.
11. For player 1, the player 1 position control variable player1_checker_postion is updated. Using 'for loop' each element is compared with 'old_pos', upon matching, it is updated with 'new_pos'. Similarly, the king checker position is also updated with the new position in the variable player1_kingchecker_postion.
12. For player 2, the player 2 position control variable player2_checker_postion is updated. Using 'for loop' each element is compared with 'old_pos', upon matching, it is updated with 'new_pos'. Similarly, the king checker position is also updated with the new position in the variable player2_kingchecker_postion.

FUNCTION: VALIDATE_TARGET_INPUT()

1. The target position and current position is stored in the variable x1, y1 and x2, y2 respectively.
2. Current position of the checker is stored in the variable checker_postion.
3. The value stored in the board for the target position is stored in the variable checker.
4. Check the row number of the capturing checker. If it is 1 or 8 decline the capture as checker cannot jump over and capture.
5. For player 1 and player 2 similar validation is done to make sure whether the requested action is feasible.
6. Checking whether the checker is a king checker or not. If it is a king checker the variable kc is updated as 'Y'.
7. For player 1, difference between x2 and x1 is equal to 1, then the move is one step. Return code variable 'rc' is updated with 1.
8. If difference between x2 and x1 is less than 1 , it means the checker is trying to move in reverse direction. Display the error message and return to main function with return code 0. However, if the checker is King checker allow this movement and update rc=1.
9. If the difference between x2 and x1 is greater than 1, then the checker is trying to move more than one step. Return back to main function with return code 0.
10. Check the horizontal movement, if the absolute value difference between y2 and y1 is 1, the movement triggered is 1 step which is accepted and allowed, Update rc=1.
11. If the absolute value of the difference between y2 and y1 is greater than 1 then, the checker trying to move more than one step Since this move is not allowed return to main function with return code 0.
12. If the absolute value of the difference between y2 and y1 is less than 1 then, the checker is trying to move more than one step Since this move is not allowed, return to main function with return code 0.
13. If the checker position ascii value is '46', which is marked as '.' In the checker board as restricted spaces, movement is not allowed. Print the error message and return to main function with return code 0.

14. If the checker position ascii value is ‘79’, which is having the player 1 checker at that place. movement is not allowed. Print the error message and return to main function with return code 0.
15. If the checker position ascii value is ‘88’, which is having the player 2 checker at that place. Check for the action requested. If the action is move, display error, and return to main function with return code 0. If the action is capture, accept the move.
16. For left to right movement, for normal checker, the check is done by incrementing x2 by 1 and incrementing y2 by 1. If the value is not ‘32’, movement not feasible. Display error message and return to main function with return value 0. For king checker the move is accepted, update rc=1.
17. For right to left movement, for normal checker, the check is done by incrementing x2 by 1 and decrementing y2 by 1. If the value is not ‘32’, movement not feasible. Display error message and return to main function with return value 0. For king checker the move is accepted, update rc=1.
18. If the final status of the variable rc is still 0, return to the main function with return code 0, if any step updated the variable rc with 1, return the code as 1.
19. For player 2, difference between x2 and x1 is equal to -1, then the move is one step. Return code variable ‘rc’ is updated with 1.
20. If difference between x2 and x1 is greater than -1 , this means the checker trying to move in reverse direction. Display the error message and return to main function with return code 0. However, if the checker is King checker allow this movement and update rc=1.
21. If the difference between x2 and x1 is less than -1, then checker is trying to move more than one step. Return back to main function with return code 0.
22. Check the horizontal movement, if the absolute value difference between y2 and y1 is 1, the movement triggered is 1 step. This is accepted and allowed, Update rc=1.
23. If the absolute value of the difference between y2 and y1 is 0 checker trying to move in the straight upward direction. This is not an allowed move, return to main function with return code 0.

24. If the absolute value of the difference between y2 and y1 is greater than 1 then, the checker is trying to move more than one step. This is not an allowed move, return to main function with return code 0.
25. If the absolute value of the difference between y2 and y1 is less than 1, Then the checker is trying to move more than one step. This is not an allowed move, return to main function with return code 0.
26. If the checker position ascii value is '46', which is marked as '.' In the checker board as restricted spaces, movement is not allowed. Print the error message and return to main function with return code 0.
27. If the checker position ascii value is '88', which is having the player 2 checker at that place. movement is not allowed. Print the error message and return to main function with return code 0.
28. If the checker position ascii value is '79', which is having the player 1 checker at that place. Check for the action requested. If the action is move, display error, and return to main function with return code 0. If the action is capture accept the move.
29. Checking the movement from left to right, for normal checker, the check is done by decrementing x2 by 1 and incrementing y2 by 1. If the value is not '32', the movement is not feasible. Display error message and return to main function with return 0. For king checker the move is accepted, update rc=1.
30. Checking the movement from right to left, for normal checker, the check is done by decrementing x2 by 1 and decremented y2 by 1. If the value is not '32', the movement is not feasible. Display error message and return to main function with return 0. For king checker the move is accepted, update rc=1.
31. If the final status of the variable rc is still 0, return to the main function with return code 0, if any step updated the variable rc with 1, return the code as 1.

FUNCTION: VALIDATE_SOURCE _INPUT()

1. The current position is stored in the variable x and y.
2. Store the coordinates from board array into the variable checker
3. If the checker variable is equal to 79 for player 1 and 88 for player 2, reject the move as their own checker is at that position.
4. If the checker variable is equal to 46, for both the players reject the move as chosen position is restricted space.

FUNCTION: PRINT_PLAYER_DETAILS()

1. Display the details of the checkers and king checkers held by player 1 from the variables player1_checkers and player1_kingcheckers.
2. Display the details of the checkers and king checkers held by player 2 from the variables player2_checkers and player2_kingcheckers.
3. Print the active player of the game from the variable active_player.

FUNCTION: INITIALIZE_BOARD()

1. Store the ascii code of the board table diagram in variables bs, vl, luc, ruc, hl, tt, lt, rt, pl, llc, rlc and it.
2. Store the board layout details in the integer array variable 'data'.
3. Use 'for loop' to update the board 2-dimention variable.
4. Update the board 2-D array with the player 1 checker positions from the player1_checker_postion variable.
5. Update the board 2-D array with the player 2 checker positions from the player2_checker_postion variable.
6. Mark the reserved space used for masking the captured checkers with a blank (ascii code=32).
7. Print the broad on the screen using the for loop and appropriate line drawings as stored in the variables.

FUNCTION: UPDATE_BOARD()

1. Store the ascii code of the board table diagram in variables bs, vl, luc, ruc, hl, tt, lt, rt, pl, llc, rlc and it.
2. Store the board layout details in the integer array variable 'data'.
3. Use 'for loop' to update the board 2-dimention variable.

4. Update the board 2-D array with the player 1 checker positions from the player1_checker_postion variable.
5. Update the board 2-D array with the player 2 checker positions from the player2_checker_postion variable.
6. Mark the reserved space used for masking the captured checkers with a blank (ascii code=32).
7. Print the broad on the screen using the for loop and appropriate line drawings as stored in the variables.

FUNCTION: ASCII_TO_NUMBER()

1. Receive the 'input' variable as a parameter from calling functions.
2. Store the value of the 'input' variable in local variable 'ch'
3. Based the value of 'ch' return the appropriate value stored in the variable 'y'