**ARUSHI RAI**

**UFID: 5898-1169**

**EMAIL: arai@ufl.edu**

# PROJECT REPORT

# NUMBRIX

## DESCRIPTION OF GAME

Numbrix is a logic puzzle game invented by Marilyn Vos Savant. Numbrix is a very fun game and does not require any math skill.

Numbrix is played on N*N grid. The objective of this game is to fill the grid with sequence of consecutive numbers from 1 to N*N, that connect vertically and horizontally (but not diagonally).

## IMPLEMENTATION:

The game is implemented in lisp language. Software used is Allegro CL 9.0.

HOW TO RUN USING ALLEGRO CL 9.0:

Open Allegro CL 9.0 and type on listener window.

Open the file using

> (setq in-file (open “path\\numbrix.lsp” :direction :input))

Load the file

> (load in-file)

Run the program

> (numbrix)

The program has 17 different boards from which a user can select any and can play either manually or let computer solve the game. All the boards have some numbers already filled and the user/computer has to fill the remaining numbers such that they satisfy the constraints.

The game can be played in two modes: Manual play and Auto play.

**Manual Play:**

In manual play, after selecting the board, the user fills rest of the numbers by entering the numbers and where he wants to place them.

The user will be asked to enter three values: row, column and number. The user should enter all the values correctly while filling any number; otherwise the program will throw some errors. There are total of four errors that a user can face while playing. First, it checks if row and column are inbound or not that is, if row and column is between 1 and N, otherwise it gives an error that row or column out of bound. Second, it checks if number is inbound or not, i.e. between 1 and N*N, else gives an error number out of bound. Third, it checks if the number is repeating or not, if it is already placed somewhere on the board then gives an error that number is repeating. Last, if all the values are correct but there is already a number present on that position then the program will give an error if that number was initially present saying that he cannot overwrite the number on that position, otherwise the program will ask the user if he wants to overwrite the number or not.

The user will keep entering the numbers till all the positions in the board are filled with different numbers. As soon as the board is filled, the program will check if all the numbers are placed correctly, satisfying the constraints of the game. To check, first the program will look for 1, if 1 is not present then there is no need to check further and the solution is not correct else it will look for next number i.e. 2 (on left, right, up and down of the number 1) and so on. If any of the number is not present in the sequence, the checking will stop and the solution is not correct. If all the numbers from 1 to N*N are present in a proper sequence then the solution is correct.

**Auto Play:**

In auto play, the user will only select the board and the computer will fill the remaining numbers using some heuristics and search techniques. After filling the board, the program will show the result generated by computer and then check if it is correct or not using the same checking function described in manual play. Also, the program will show the time taken by computer to solve the board.

Following are some of the heuristics that are used to solve the Numbrix game:

1. When there is only one option to fill the number.

- Filling the numbers that have limited choices:
  Using the constraint satisfaction search technique, fill the numbers which have only one blank option. The program will start searching for number which can be placed using

this technique, from smallest number (from 1 to N*N). If any number is not present, it checks if next smallest is there or not.
For instance, taking a small part of 8*8 board

| | | |
|---|---|---|
| | | 38 |
| | 4 | 5 |
| | 7 | 6 |

Here while placing 4, it has only one option. According to constraint that 4 should be placed next to its adjacent number (vertically or horizontally), here moving left.

- Filling the number between alternate numbers:
  Again using the constraint satisfaction technique, starting from the smallest number from 1 to N*N, checking if its next number is present or not, if not then check if number next to its next number is present or not. If this is the case, then the next number should be placed in between.
  For instance, taking a small part of 8*8 board

| | | |
|---|---|---|
| | | |
| 5 | 6 | 7 |
| | | |

Here, 6 should be placed between 5 and 7 and there is no other option for it.

Another situation based on this same concept.
When the next number to be placed actually has two options but one option is already filled by some other number then even if this case the next number to be placed has only one option.
For instance,

| | | |
|---|---|---|
| | | |
| | 6 | 7 |
| | 5 | 34 |

Here, 6 have two options but one is already filled.

2. When there are two options.

- No empty blanks:
  Again using constraint satisfaction search technique, all the blanks should be filled in order to find the solution. By taking a path which makes blanks to not get filled in the future should be avoided. For instance, taking a part of 8*8 board

| | | | |
|---|---|---|---|
| | | 2 | 3 |
| 9 | | 1 | 4 |
| 8 | 7 | 6 | 5 |

Here 1 has two options (either on left or down of 2), if we take path other than the colored one then that would make the colored cell empty and no other number can be placed there.

- No sequence breaking:
  Applying constraint satisfaction search, numbers should be placed sequentially.
  For instance, taking a part of 8*8 board

| | | | |
|---|---|---|---|
| | 1 | 2 | 3 |
| 9 | 10 | 11 | 4 |
| 8 | 7 | 6 | 5 |

Now, if we place 10and 11 in this way then that will break the sequence, since now there is no place for 12 and hence the constraint that numbers should be placed consecutively is satisfied.

3. When more than one number is missing between two numbers but there is only one possible way to fill the numbers between them.

- Only possible sequence:
  The program will start searching from the smallest number and see if there are more than one number are missing between two numbers. Calculate the distance between the two numbers and check if there is only possible path or not.
  For instance, taking a part 8*8 board

| | | | |
|---|---|---|---|
| 38 | 36 | 35 | 24 |
| | | 34 | 25 |
| | | 33 | 26 |
| | | 32 | 27 |
| | | 31 | 28 |
| | | 30 | 29 |

Here the distance between 30 and 35 is 4 and the possible path has 4 blanks. Taking other sequences would have created the problem of blanks being not filled in the future.

Another situation when the distance between two numbers is two and there are actually two possible paths but already one is feasible.
For instance, taking a part 8*8 board

| | | |
|---|---|---|
| 45 | 5 | 6 |
| | 8 | 7 |
| | | |

Here, 6 and 7 has only one option left since other side has number filled in one cell.

4. When all the above heuristics does not work, then the program will traverse the whole board starting from 1 and use depth first search and backtracking to fill the blanks and keep applying it till all the numbers are correctly placed.

**MAJOR VARIABLES**

- brd1: it's a list of numbers and blanks present on the board.
- available: it's a list containing 0 and 1, 0 for a number which is present and 1 for number which is not present.
- n1 and n: size of the board.
- row: row shows the row number of the board.
- column: column shows the column number of the board.
- val: number of the board.
- list_empty: list of empty blanks. Used to display the numbers while checking the sequence.
- fill_op1, fill_op2, fill_op3 and fill_op4: checking variables used while filling the board automatically, till when the mentioned heuristics can be applied.

**MAJOR FUNCTIONS**

- numbrix(): this is the main function, it starts the game. Provide the user with board options and ask for the mode of the game (Manual or Auto).
- board(): prints the entire board.
- set_value: set the value in the position provided.
- get_value(): gets the value present at the position provided.
- make_list(): make a copy of the list.
- check_digits(): checks the number of digits in a given number used for displaying the board.
- check_baord(): starts checking the board. Has all the empty boards used to display the board while checking.
- look_start(): look for 1 in the board.
- checking(): called by look_start() function, if 1 is found, it checks for rest of the values, whether they are correctly placed or not.
- start_play(): for manual play, ask for all the error checking functions.

- keep_play(): keeps repeating the start_play() function for user till all the blanks are filled.
- check_position_bound(): check if row and column are out of bound or not.
- check_repeat(): check if number entered by user is repeating or not.
- check_value_bound(): check if value is out of bound or not.
- Check_overwrite(): check if user is overwriting a number at any position or not.
- auto(): starts the automatic play.
- fill_one_option(): fill all the numbers which have only one option to be placed.
- fill_alternate(): fill all the numbers which can be placed between any two numbers.
- fill_two_option(): fill all the numbers which has two options but only one is feasible.
- fill_simple_distance(): fill the numbers which have only one possible path between two numbers.
- set-sqr1(): sets the value 1 or 0 in the available list.
- sqr1(): gets the value for the number from available list.
- dfs_tree(): fill the blanks by searching for all the possible ways.

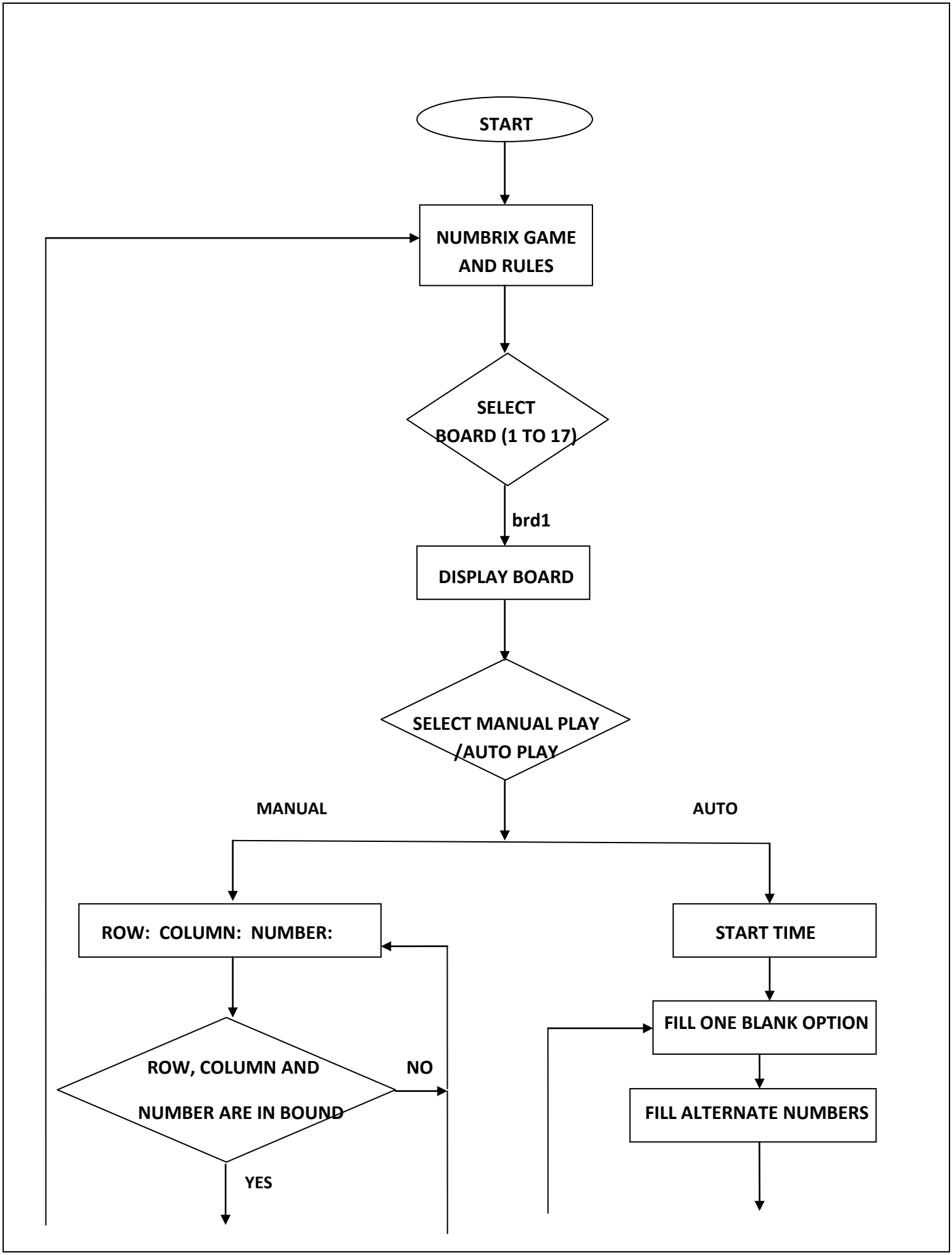
**FLOWCHART**

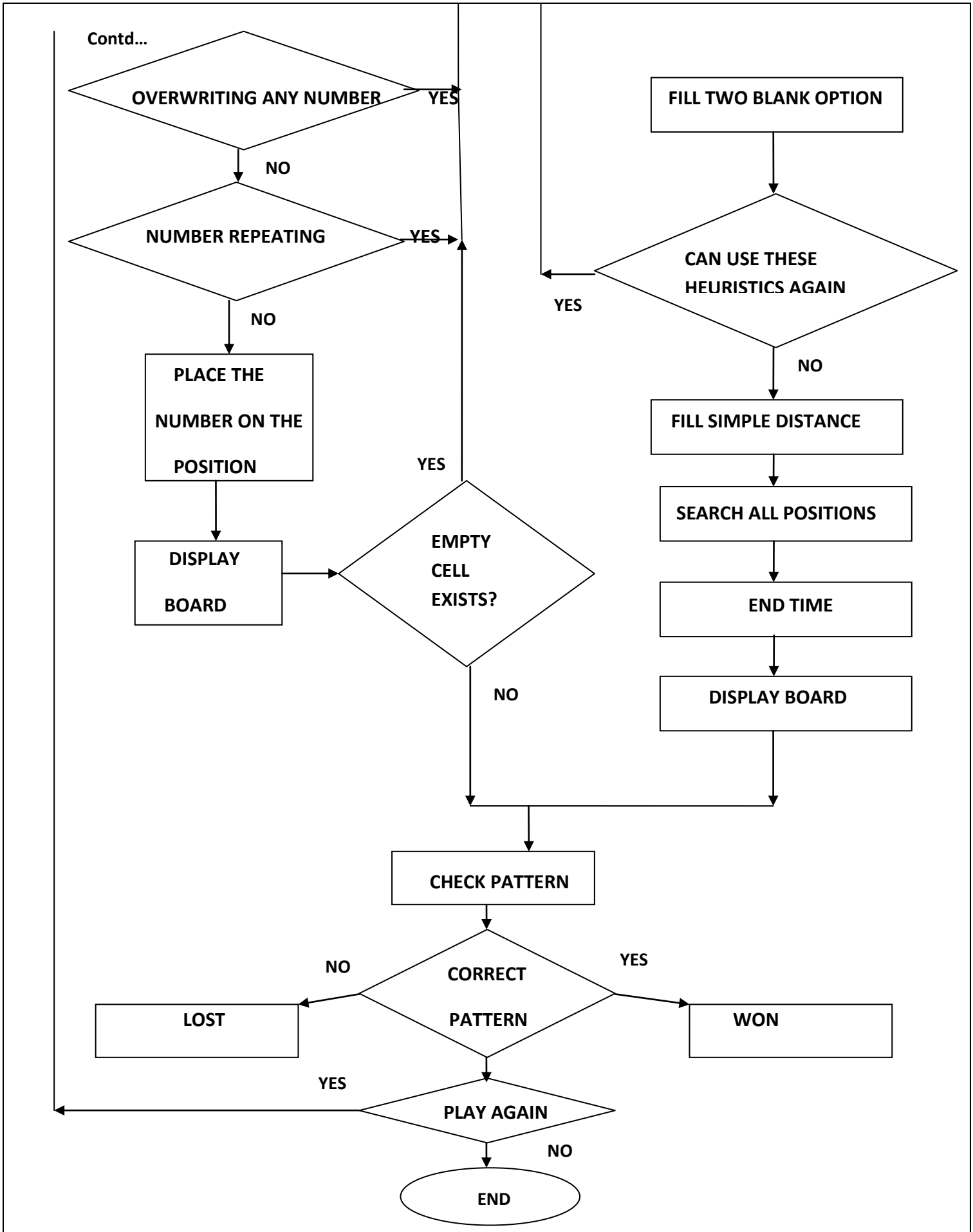
The flow for this program starts from numbrix function, this is the main function. First ask user to select the board and displays it. According to the user, the numbrix function then calls functions for manual play or auto play, start_play() or auto().

Start_play() then ask the user for position and number and calls for all the error functions and place the number if there is no error and keeps it playing till the blanks are filled.

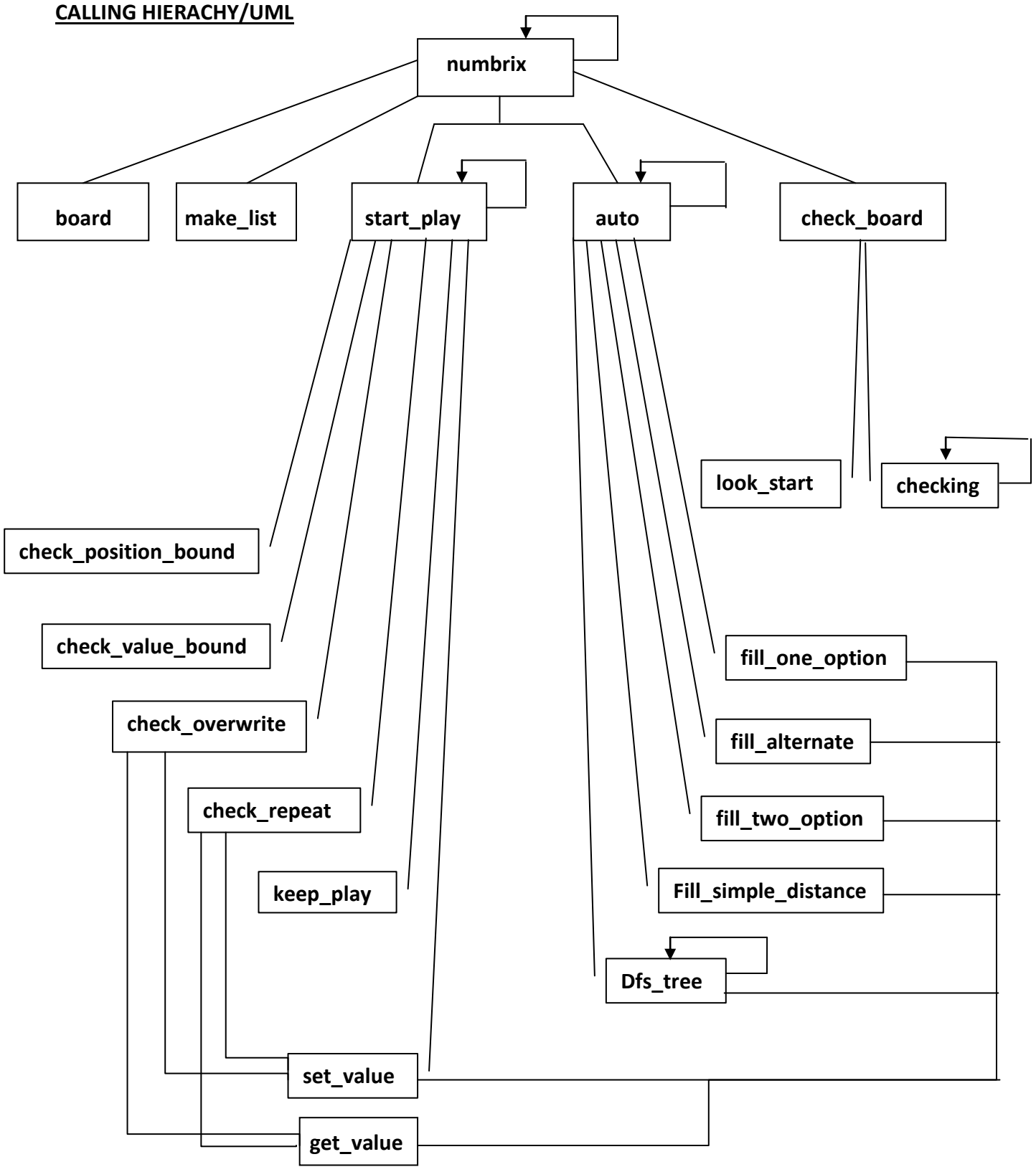
Auto() functions call all the heuristic function one by one and then finally call for the search function till the numbers are placed correctly.

Lastly, numbrix function will call the check_board function to check if numbers are correctly are correctly placed or not.


START
NUMBRIX GAME AND RULES
SELECT BOARD (1 TO 17)
brd1
DISPLAY BOARD
SELECT MANUAL PLAY /AUTO PLAY
MANUAL
AUTO
ROW: COLUMN: NUMBER:
ROW, COLUMN AND NUMBER ARE IN BOUND
NO
YES
START TIME
FILL ONE BLANK OPTION
FILL ALTERNATE NUMBERS


Contd…
OVERWRITING ANY NUMBER
YES
NO
NUMBER REPEATING
YES
NO
PLACE THE NUMBER ON THE POSITION
DISPLAY BOARD
YES
EMPTY CELL EXISTS?
NO
FILL TWO BLANK OPTION
CAN USE THESE HEURISTICS AGAIN
YES
NO
FILL SIMPLE DISTANCE
SEARCH ALL POSITIONS
END TIME
DISPLAY BOARD
CHECK PATTERN
CORRECT PATTERN
NO
YES
LOST
WON
YES
PLAY AGAIN
NO
END

**CALLING HIERACHY/UML**



**INTELLIGENCE IMPLEMENTED**

- Filling the numbers that have limited choices.
- Filling the number between alternate numbers.
- When there are two options for a number but only one is feasible because in order to avoid empty blanks left at the end and to avoid sequence breaking, only one option is left.
- When there is a gap between two numbers and there is only one possible sequence.
- Using depth first search and backtracking to fills the numbers correctly.

**INTELLIGENCE NOT IMPLEMENTED**

- When there is a gap between two numbers and there are many possible sequences. Taking one solution and checking if that path is possible or not. Using backtracking.

**WHAT I WOULD DO DIFFERENRTLY**

- I would try to implement a better heuristic rather than applying depth first search, since it takes a lot of time searching every possible solution, hence not that efficient. I would rather find two numbers and calculate the distance between them. If there is only one path then place the numbers and if there are more than one path then trying those paths in order to get to the solution and using backtracking if one doesn't work.