**IT 114 - ADVANCED PROGRAMMING FOR INFORMATION TECHNOLOGY**

**Case Studies**

**1. Formulating the Problem**

**1.1 Problem Description**

We must create a program that prompts the user to choose a starting point within an array, which will be used to see whether the puzzle is solvable or not if you start from that area. It uses a recursive method to check whether or not its solvable and prints whether it is or not at the end.

The user should have the ability to choose an integer between 0 & 9 and receive an accurate answer from the system. The program should display the answer to the user.

**1.2 Verbalization**

*What is the goal?*

Create a program that tells you whether the starting point you selected is solvable or not. It must use a recursive method to check it, and at the end it prints whether its solvable or not with that starting point or not.

*What are the givens?*   The numbers in the array, and the size of the array[10]

*What are the unknowns?*   The total amount of times the method will run

1.3 **Information Elicitation**

*Goal*

Create a program that – tells you whether the starting point you selected is solvable --- uses a recursive method to check it --- prints whether its solvable or not.

*Givens*   The numbers in the array, the size of the array[10]

*Unknowns*   How many times it will run

*Conditions*   The starting point cannot be the same as the end value 0

**2. Planning the Solution**

**2.1 Solution Strategy**

Create an array that contains integers and ends with 0. Prompt the user to choose an integer between 0 & 9.

Take the user input and create a variable to store it named ‘n’. Use variable n to extract a value form its relevant position in the array and assign that value to variable ‘a’. Check to see if the value of ‘a’ is equal to the answer 0. Create variable nleft (n-a) and see if nleft is greater than -1. Set nleft as the new value of ‘a’ and recall the method PuzzleSolver with ‘a’ as the parameter. Create variable nright (n+a) and see if nright is less than 10. Set nright as the new value of ‘a’ and recall the method PuzzleSolver with ‘a’ as the parameter. This repeats until an answer is found. If neither works, the problem isn’t solvable, return not solvable. Create a print statement in the main method that says whether the function is solvable or not

**)**

**2.2 Goal Decomposition**

*Sub-goal 1*

Create an Array

*Sub-goal 2*

Ask the user for an input

*Sub-goal 3*

Take user input

*Sub-goal 4*

Assign it to variable ‘n’

*Sub-goal 5*

Use variable n to extract a value form its relevant position in the array and assign that value to variable ‘a’

*Sub-goal 6*

Check to see if the value of ‘a’ is equal to the answer 0

*Sub-goal 7*

Create variable nleft (n-a) and see if nleft is greater than -1

*Sub-goal 8*

Set nleft as the new value of ‘a’ and recall the method PuzzleSolver with ‘a’ as the parameter

*Sub-goal 9*

Create variable nright (n+a) and see if nright is less than 10

*Sub-goal 10*

Set nright as the new value of ‘a’ and recall the method PuzzleSolver with ‘a’ as the parameter

*Sub-goal 11*

If neither works, the problem isn’t solvable, return not solvable

*Sub-goal 12*

Print whether the function is solvable or not

**2.3 Resources**

*Relevant Information*

**Puzzle Array:** 3. 6. 4. 1. 3. 4. 2. 5. 3. 0

**3. Designing the Solution**

**)3.1 Structure Chart**

*First Level Decomposition*



*Goal Refinement*

**Sub-goal 1**

Create an array

**Sub-goal 2**

Ask user for input

**Sub-goal 3**

Take input from user

**Sub-goal 4**

Assign input to variable ‘n’

**Sub-goal 5**

Use variable to extract value from its relevant position in the array and set it to variable ‘a’

**Sub-goal 6**

Create variable nleft (n-a) and nright (n+a) which show the new position in the array

**Sub-goal 7**

Check to see if the value of the of ‘a’ is equal to the answer 0

**Sub-goal 7.1**

if it is, return value is solvable. If not continue

**Sub-goal 8**

Check to see if nleft is greater than -1

**Sub-goal 8.1.1**

Check to see if nleft isn’t equal to a, if so continue

**Sub-goal 8.1.1**

if it is, set the nleft as the new value of ‘a’ and recall the method with the new position

**Sub-goal 9**

Check to see if nright is less than 10

**Sub-goal 9.1**

Check to see if nright isn’t equal to a, if so continue

**Sub-goal 9.1.1**

if it is, set the nright as the new value of ‘a’ and recall the method with the new position

**Sub-goal 10**

If neither function works, the problem isn’t solvable

**Sub-goal 10.1**

Return value isn’t solvable

**Sub-goal 11**

Print whether the function is solvable or not

*Second Level Decomposition*



The second level decomposition shows operations between the User and the System. The process starts with the user being prompted by the system to choose a integer between 0 & 9. The system then takes the integer and runs it through its method called PuzzleSolver. The method PuzzleSolver creates multiple variables using the integer. It first creates a variable ‘a’ which takes the value of the starting point chosen by the user from the array Puzzle. It then creates variables nleft and nright by either calculating n-a and assigning it to nleft or calculating n+a and assigning that to nright. It then checks to see if ‘a’ is equal to the solution 0. If not, it continues to the next function. If the value of nleft is greater than -1 and nleft isn’t equal to ‘a’ then it sets a to nleft and recalls PuzzleSolver with ‘a’ as the parameter. If that isn’t met, it checks to see if the value of nright is less than 10 and that nright isn’t equal to ‘a’. Then it sets ‘a’ to nright and recalls PuzzleSolver with ‘a’ as the parameter. If all else fails, the problem is not solvable, and the program returns problem isn’t solvable.

**3.2 Module and Data Specifications**

**Name**: Choice – Display a message to user asking them to choose a number between 0 and 9

**Input**: int(n)

**Output**: string stating whether or not the value is solvable

**Logic**: call function PuzzleSolver(n) which takes the input and solves the problem by checking for ways it can equal to 0 starting from the given value.

**3.3 Logic**

*Logic*

1. Print the user choice
2. User Inputs value between 0 & 9

3.0 Selected value is assigned a variable ‘n’

4.0 New variable is created ‘a’ is assigned the value of the position ‘n’

5.0 new variables are created nleft and nright.

5.1 nleft is equal to n – a

5.2 nright is equal to n + a

6.0 if position ‘a’ is equal to the value 0

6.1 the function returns true meaning the starting point is solvable

7.0 If nleft is greater than -1, that means the program can move left

7.1 if nleft is equal to a, don’t continue

7.1.1 the program assigns the new value to ‘a’ and restarts the method

8.0 if nright is less than 10, that means the program can move right

8.1 if nright is equal to a, meaning it’s the same value, don’t continue

8.1.1 the program assigns the new value to ‘a’ and restarts the method

11.0 if none of the functions worked, the problem isn’t solvable and the value returned is False.

12.0 system prints string stating whether or not your starting point is solvable

*Algorithm Description*

The process of doing this : Data is stored in an array called puzzle. The system gives the User an option to input a number between 0 & 9 as the starting point of the array. It then takes the input and prints a line stating whether or not the starting point is solvable by calling the recursive method PuzzleSolver.

Puzzle solver stores the array takes the input from the main method. It then creates variables nleft and nright which allows it to see if the function can progress left or right in the array. This is determined using ‘if’ statements. If the none of the requirements are met of any of the statement, as in it cannot go left or right and the value isn’t 0, the method returns false, meaning it’s not solvable. If the ‘if’ statement to see if it can go left is fulfilled the method sets the new starting point to the value of nleft and restarts to see if the requirements to end are fulfilled. If it is the solution is true and the method ends. If not it checks if it can go left, if not left it checks right, and it repeats this until it finds an answer of either true or false.

**4. Translation**

**4.1** **Source Code**

I removed the indents to try to make it easier to read in this document. With the indents it was hard to look at.

//===================================================   
// Name : Emad Tirmizi  
// SID : 31400222  
// Course : IT114   
// Section :   
// Instructor : Maura Deek  
// T.A :   
//===================================================   
//===================================================   
// Assignment # : 3  
// Date : 11/2/2018  
//===================================================   
//===================================================   
// Description: This program will check to see if the  
// if the starting point chosen by the user can be  
// solved through the use of a recursive method. It  
// informs the user whether or not it is able to be  
// solved using the starting point provided by the  
// user  
//===================================================   
  
  
**import** java.util.Scanner;  
  
**class** main{  
**public** **static** **void** main(String[] args){  
//create a scanner so that the user can select a number to choose from  
Scanner input **=** **new** Scanner(System.in);  
//print which numbers the user can choose from  
System.out.print("Enter a starting point between 0 & 9 \n");  
//set the user input to variable n  
**int** n **=** input.nextInt();  
   
//print the result  
System.out.println("The starting point of " **+** n **+** " is solvable: " **+** PuzzleSolver(n));  
}//end of main method  
   
//create recursive method  
**public** **static** String PuzzleSolver(**int** n){  
//create an array  
**int[] puzzle = new int[] {3, 6, 4, 1, 3, 4, 2, 5, 3, 0};**  
//set a variable that retrieves the value of the position the user chose  
**int** a **=** puzzle[n];  
//create a variable nleft that will be used to see if the function can move left  
**int** nleft **=** n **-** a;  
//create a variable nright that will be used to see if the function can move right  
**int** nright **=** n **+** a;  
//if the value of a, as in the current position is 0 that means the problem is solved  
   
String value **=** "";  
   
**if**(n **==** 0){  
**return** value **=** "problem solved";  
}//end of if  
   
//if nleft is greater than negative 1, meaning the function can go left, it will   
**else** **if**(nleft **>** **-**1){  
//if nleft doesn't equal to a, meaing its not the same value, then continue so it doesn't repeat  
**if**(nleft **!=** a){  
//the variable 'a' is set to the new location  
a **=** puzzle[nleft];  
//the function is recalled to see if the base case is met  
**return** PuzzleSolver(a);   
}//end of if   
}//end of if  
   
//if nright is less than 10, meaning the function can go right, it will  
**else** **if**(nright **<** 10){  
//if nright doesn't equal to a, meaing its not the same value, then continue so it doesn't repeat  
**if**(nright **!=** a){  
//the variable 'a' is set to the new location  
a **=** puzzle[nright];  
//the function is recalled to see if the base case is met  
**return** PuzzleSolver(a);   
}//end of if  
}//end of else if  
   
//if all else fails and the function cannot be solved it will return false as the answer  
**else**{  
**return** value **=** "problem cant be solved";  
}//end of else  
**return** value;  
}//end of recursion method  
}//end of class

**)4.2 Program and Module Description**

Main

The main method prints a line asking the user to input a number between 0 & 9 as the starting point of the array. It then takes the input and prints a line stating whether or not the starting point is solvable by calling the recursive method PuzzleSolver.

Puzzle solver stores the array takes the input from the main method. It then creates variables nleft and nright which allows it to see if the function can progress left or right in the array. If the function cannot go left or right, the solution is false, meaning its not solvable. If it can go left it first checks to see if the new position is equal to the end point 0. If it is the solution is true and the method ends. If not, it checks if it can go left, if not left then it checks if it can go right, it repeats this until it finds an answer.

**5. Solution Testing**

Test the program with following data domain:

Ideally, any whole number between 0 & 9. However, for some reason I could only get the starting points of 0 and 9 to work. Any other number doesn’t work and causes a recursive error.

