Grayson Hart
Thomas Mintun
CS4500-Chakraborty-Project2

Requirements Specification Document:

1. Problem Type

The people involved in the creation of this software are Thomas Mintun the user/customer, and Grayson Hart the software developer. Thomas made a wager with a non-CS student that he will be able to solve the famous n-Queens problem with more queens than the non-CS student. This document's audience is: Thomas Mintun the customer/user, Grayson Hart the developer. The scope of this project is small in size, and in fact one-time use. Thomas does not care about the source code, so Grayson will keep the executable and the source code. Thomas is only using the software once to beat his non-CS friend in a wager.

2. Problem Statement and Problem Parameters

"The size of N shall not exceed 25 or you will lose your bet!" is the first thing Grayson told Tom. Grayson made it explicitly clear that his algorithm cannot handle N larger than 25. The maximum amount of time the software will take to answer with a working chess board without attacking queens is 90 seconds. Grayson must be able to communicate this answer with Thomas.

User classes will be: USERS. Users consists of two people: Grayson and Thomas. The operating environment will be on the Grayson's computer. Implementation constraints will include: software must be functional in 5 days, Grayson Hart the developer must be there with Thomas to run the program. He must also bring his computer to a location Thomas tells him and run the code on his machine.

The use cases for these actors is detailed here:

USERS CLASS: Thomas will tell Grayson what number of N to solve for. Grayson will input the N, interpret the results, tell them to Thomas.

a. Success Scenario: The information is entered into software correctly. Grayson interprets the results and tells them to Thomas. Thomas writes his solution and beats his non-CS friend in a wager of knowledge.
b. Failure Scenario: There is a communication error between Thomas and Grayson, an incorrect N is entered, the program takes too long to find a solution, Grayson has a hardware failure.

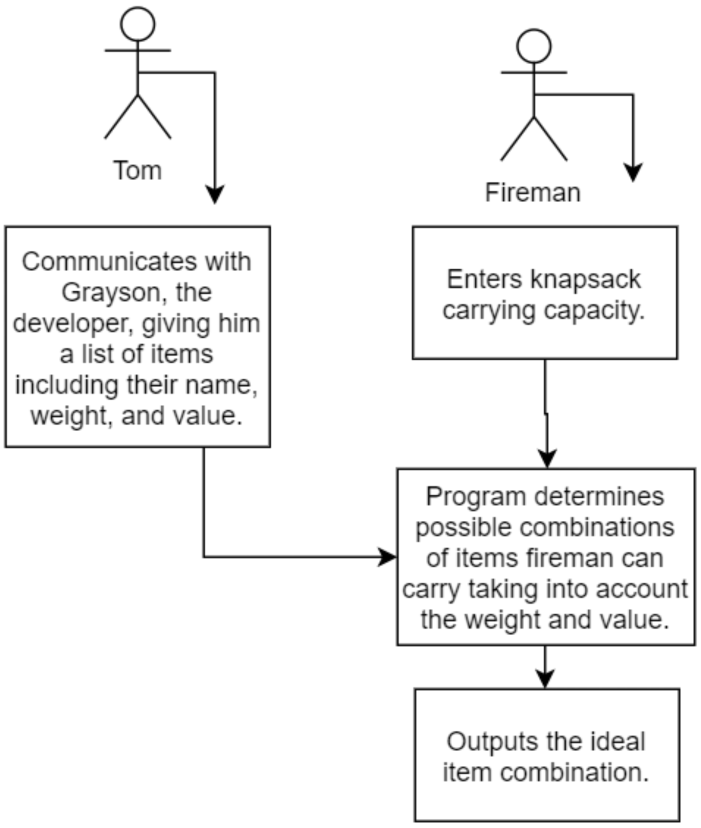
4. User Interface

The user interface is Grayson's computer. His IDE is JetBeans NetBrains IntelliJ. If a future version is sold for more users, the user interface will need to be updated.

5. Cost, Maintenance

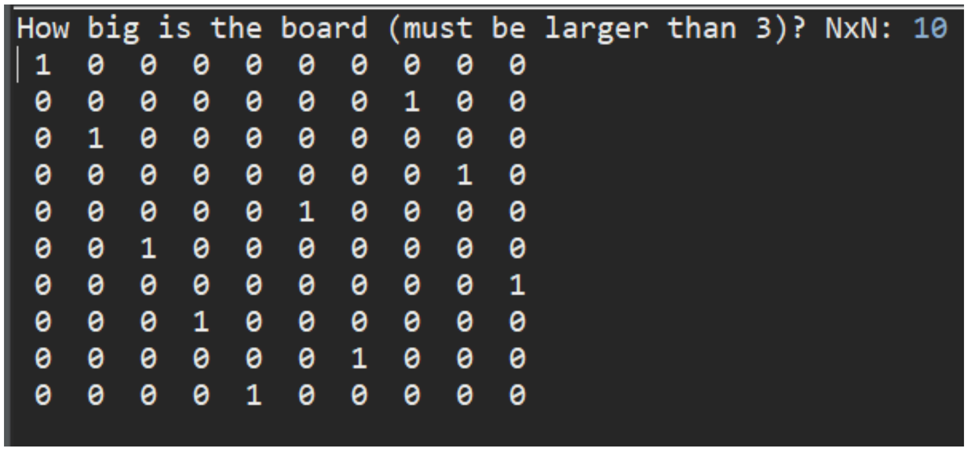
Thomas is paying Grayson $9.99 for his work. It is up to Thomas to win more than $9.99 in his bet to profit. There will be no maintenance because Thomas is only using the software once. The software is hosted on Grayson's computer.

Program Design Document:

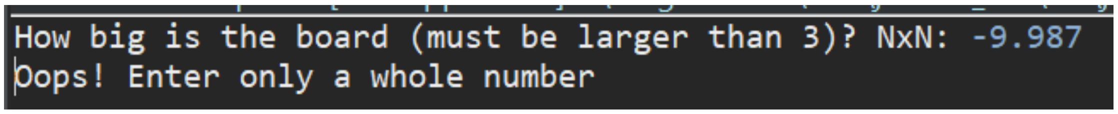

Tom
Fireman
Communicates with Grayson, the developer, giving him a list of items including their name, weight, and value.
Enters knapsack carrying capacity.
Program determines possible combinations of items fireman can carry taking into account the weight and value.
Outputs the ideal item combination.

Document Detailing Testing Strategies:

The software will be tested by Grayson. He will communicate with Tom about any issues. The software will be tested on Grayson's computer because it will be running on Grayson's computer. Grayson entered in larger numbers than 25, numbers smaller than 3, decimal numbers, negative numbers, and characters to see the output, and determined to tell Thomas that the only acceptable input is an integer between 4 and 25 inclusive. Grayson is there with Thomas at run time and runs the code on his computer. His computer must be charged and mobile, and Grayson has tested this. Grayson answers phone calls or calls back within 4 hours during the development cycle.



```
How big is the board (must be larger than 3)? NxN: 10
 1  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  1  0  0
 0  1  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  1  0
 0  0  0  0  0  1  0  0  0  0
 0  0  1  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  1
 0  0  0  1  0  0  0  0  0  0
 0  0  0  0  0  0  1  0  0  0
 0  0  0  0  1  0  0  0  0  0
```



```
How big is the board (must be larger than 3)? NxN: -9.987
Oops! Enter only a whole number
```

```
package nqueens;
import java.util.InputMismatchException;
import java.util.Scanner;

public class nqueens {
	//start in leftmost column
	//returns true if all the queens are placed
	//try all rows in column
	//- if queen can be placed mark this row
	//- if placing queen leads to a solution return true
	//- if placing queen doesn't work backtrack.
	//if nothing works return false to trigger backtracking
	void printboard(int board[][], int boardSize) {
		for(int i = 0; i < boardSize; i++) {
			for(int j = 0; j < boardSize; j++) {
				System.out.print(" " + board[i][j] + " ");

			} System.out.println();
		}
	}
	boolean isSafe(int board[][], int row, int col, int boardSize) {
		int i, j;
		for(i = 0; i < col; i++)
			if (board[row][i] == 1)
				return false;
		for (i=row, j=col; i >= 0 && j >= 0; i--, j--)
```

```
            if (board[i][j] == 1)
                return false;
        for (i = row, j=col; j >= 0 && i < boardSize; i++, j--)
            if (board[i][j] == 1)
                return false;
        return true;
    }
    boolean solveBoard(int board[][], int col, int bSize) {
        if (col >= bSize)
            return true;
        for (int i = 0; i < bSize; i++) {
            if (isSafe(board, i, col, bSize)) {
                board[i][col] = 1;
                if(solveBoard(board, col+1, bSize) == true) {
                    return true;
                }
                board[i][col] = 0;
            }
        }
        return false;
    }
    boolean solve(int boardSize) {

        int[][] board = new int[boardSize][boardSize];
        if (solveBoard(board, 0, boardSize) == false) {
            System.out.print("No Solution");
            return false;
        }
        printboard(board, boardSize);
        return true;
    }
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        nqueens queen = new nqueens();
        System.out.print("How big is the board (must be larger than 3)? NxN: ");
        try {
            int boardSize = input.nextInt();
            while (boardSize <=3 ) {
                System.out.println("Enter a positive number lager than 3:");
                boardSize = input.nextInt();
            }
            queen.solve(boardSize);
        }
        catch(InputMismatchException e) {
            System.out.println("Oops! Enter only a whole number");
        }
    }
}
```

```
package nqueens;
import java.util.InputMismatchException;
import java.util.Scanner;


public class nqueens {
	//start in leftmost column
	//returns true if all the queens are placed
	//try all rows in column
	//- if queen can be placed mark this row
	//- if placing queen leads to a solution return true
	//- if placing queen doesn't work backtrack.
	//if nothing works return false to trigger backtracking
	void printboard(int board[][], int boardSize) {
		for(int i = 0; i < boardSize; i++) {
			for(int j = 0; j < boardSize; j++) {
				System.out.print(" " + board[i][j] + " ");


			} System.out.println();
		}
	}
	boolean isSafe(int board[][], int row, int col, int boardSize) {
		int i, j;
		for(i = 0; i < col; i++)
			if (board[row][i] == 1)
				return false;
		for (i=row, j=col; i >= 0 && j >= 0; i--, j--)
			if (board[i][j] == 1)
				return false;
```