Written and completed by candidates 260104, 260121, 184100, 268940

Advanced software engineering reflective essay Group 1.

Contents

Introduction

The group was given a total of 4 deliverables to accomplish when the coursework first started, and those deliverables made up about 65% of the final grade. The group was given a variety of difficulties to solve as part of the project. Due to the group's stranded interaction with the algorithmic difficulties, the problems offered a distinct algorithmic problem stance, which faced us with challenges for each release.

Deliverable 2 n-queens:

For this task we began by researching the topic because the group was unfamiliar with just questions. We first delved into what the n queens' problem is and then we proceeded to research various successful interpretations of this problem on websites such as a GitHub. The first attempt of this problem was a n queens' problem that was completed on python using google colab. We uploaded this experimental file on GitHub just so that the group could fairly understand the context of the problem and how to approach it better and to make a final version that would be accepted into the final form requested by the head lecturer.

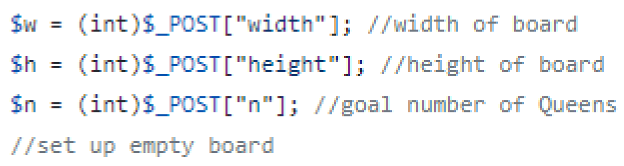
Once the group managed to complete the research into the topic, we began the final procedure we underwent to complete this deliverable which was to implement the problem correctly onto a web application that can then be submitted. The final languages that we used to submit this problem with was PHP and CSS. We took advantage and decided the use the university of Sussex webspace to display and store our website and solutions.

Methodology

1. First used CSS to make the style of the webpage
2. The application will ask the user to input the board width and height, and how many queens do they wants to put on the board.

To begin with the board size was set, because it is custom the board size can be set to any size.

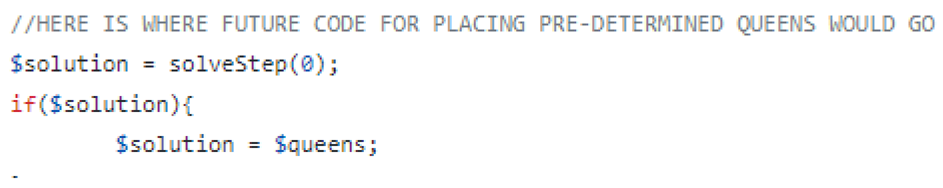
Functions were used to represent the width of the board, the height of the board, and the number of the queens.



```
$w = (int)$_POST["width"]; //width of board
$h = (int)$_POST["height"]; //height of board
$n = (int)$_POST["n"]; //goal number of Queens
//set up empty board
```

Figure 1: Functions used to represent the board and the number of queens.

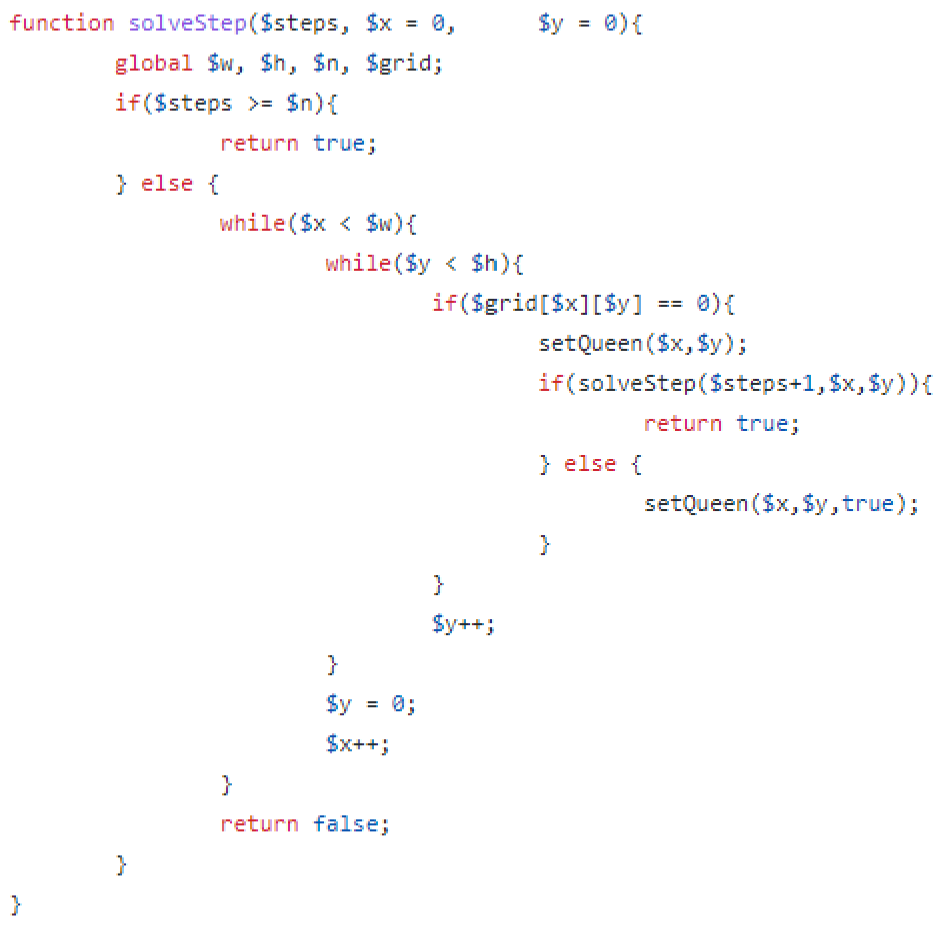
A further function was then put in place to decide where any future queens would be placed on the board.



```
//HERE IS WHERE FUTURE CODE FOR PLACING PRE-DETERMINED QUEENS WOULD GO
$solution = solveStep(0);
if($solution){
        $solution = $queens;
```

Figure 2: Function used to show where future queens would be placed on the board.

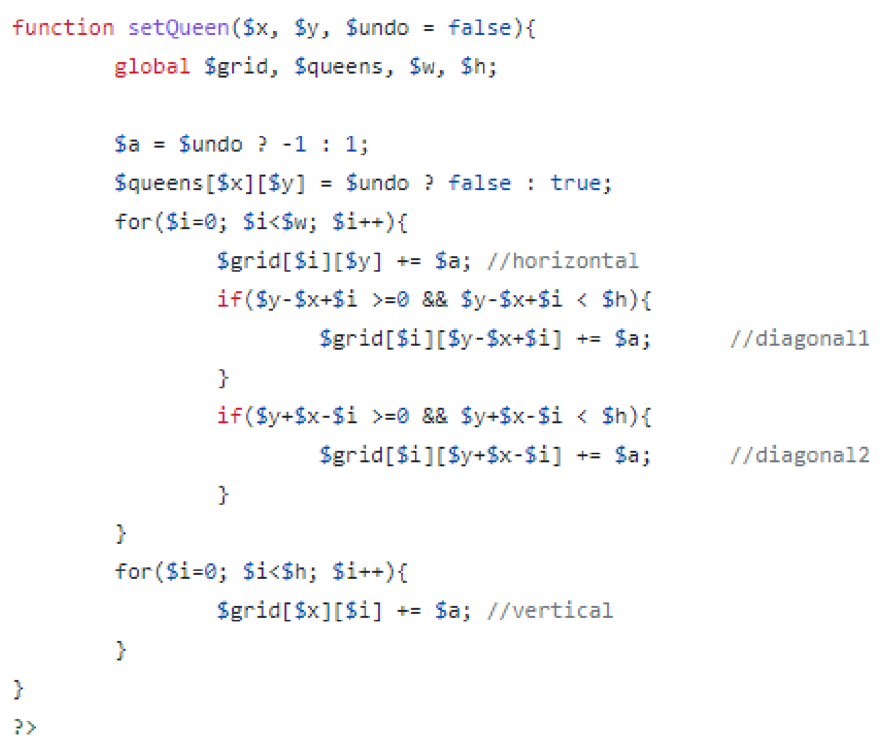
Once these parameters were put in a place a function was then created to go through the steps and solve problem using an algorithm.



```
function solveStep($steps, $x = 0,      $y = 0){
        global $w, $h, $n, $grid;
        if($steps >= $n){
                return true;
        } else {
                while($x < $w){
                        while($y < $h){
                                if($grid[$x][$y] == 0){
                                        setQueen($x,$y);
                                        if(solveStep($steps+1,$x,$y)){
                                                return true;
                                        } else {
                                                setQueen($x,$y,true);
                                        }
                                }
                                $y++;
                        }
                        $y = 0;
                        $x++;
                }
                return false;
        }
}
```

Figure 3: Function used to solve the algorithm

Following these steps, a final function was created, this function served as the algorithm that would travers the board and place the queens in the correct location. The board was traversed via horizontal, diagonal and vertical traversals.



```
function setQueen($x, $y, $undo = false){
        global $grid, $queens, $w, $h;

        $a = $undo ? -1 : 1;
        $queens[$x][$y] = $undo ? false : true;
        for($i=0; $i<$w; $i++){
                $grid[$i][$y] += $a; //horizontal
                if($y-$x+$i >=0 && $y-$x+$i < $h){
                        $grid[$i][$y-$x+$i] += $a;      //diagonal1
                }
                if($y+$x-$i >=0 && $y+$x-$i < $h){
                        $grid[$i][$y+$x-$i] += $a;      //diagonal2
                }
        }
        for($i=0; $i<$h; $i++){
                $grid[$x][$i] += $a; //vertical
        }
}
?>
```

Figure 4: Function to set the queens in the correct places on the board.

Deliverable 3 polysphere puzzle:

In order to have a general sense of the ultimate output before starting this job, we first looked at the websites the professor had provided. After that, we began researching the subject and made the decision to employ JavaScript and CSS to complete this work.

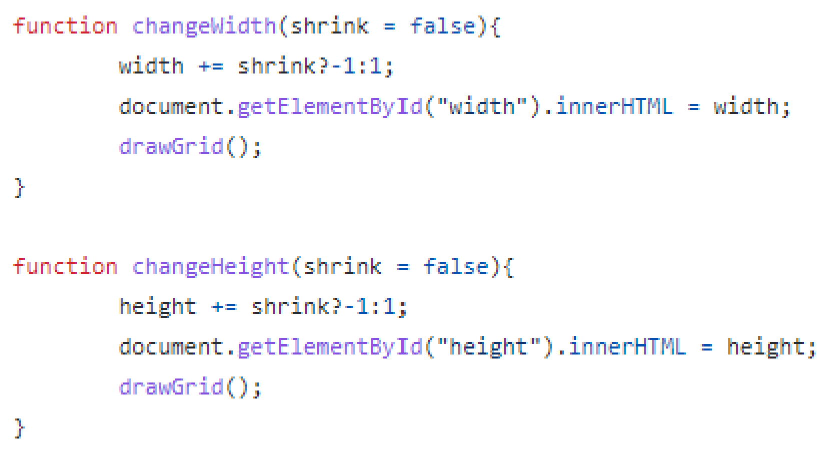
The Polysphere problem's basic idea is to fit 12 supplied forms within an 11 by 5 grid without overlapping any of them. This puzzle has frequently been used as an algorithmic test to check if a programmer can come up with an effective way to generate answers to the puzzle.

Unfortunately, in that regard our solver fails. While there are features in the solver (such as skipping shapes that can no longer fit the available grid space or ignoring rows and columns of the grid that have already been filled), this solver uses a largely brute-force method and is not efficient enough to find solutions to the classic configuration of this puzzle in a reasonable time. The intended plan for future development is to attempt to implement a 2-dimensional variant of Donald Knuth's "Algorithm X", which solves this problem much more quickly, or to find some other alternative approach.

Our solver does, however, solve such problems. Given a set of shapes and a grid, this website will find all possible configurations of those shapes contained within that grid. While the classic problem is too difficult to see results quickly, you can change the configuration to a simpler version of the puzzle to see this page in action.

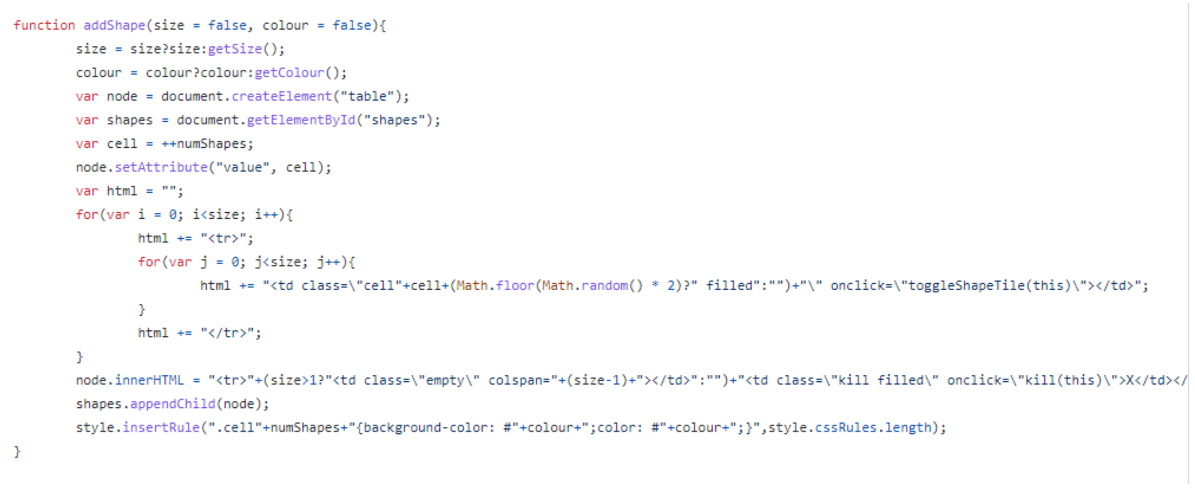
Click on the grid to the right of the page to place pre-filled blocks or click on the squares for the pieces below to reconfigure their shape. Pieces can be added with the + symbol at the end of the Shapes section and removed with the "X" above each shape. When a new shape is generated, it will ask you what size and colour you want it to be, defaulting to a size of 3 and a random colour.

Once you are happy with your configuration, you can click the solve button to see solutions to the puzzle you have generated. You can stop the solver at any time with the "Stop" button that appears and clear the board, as well as past solutions, with the clear button.



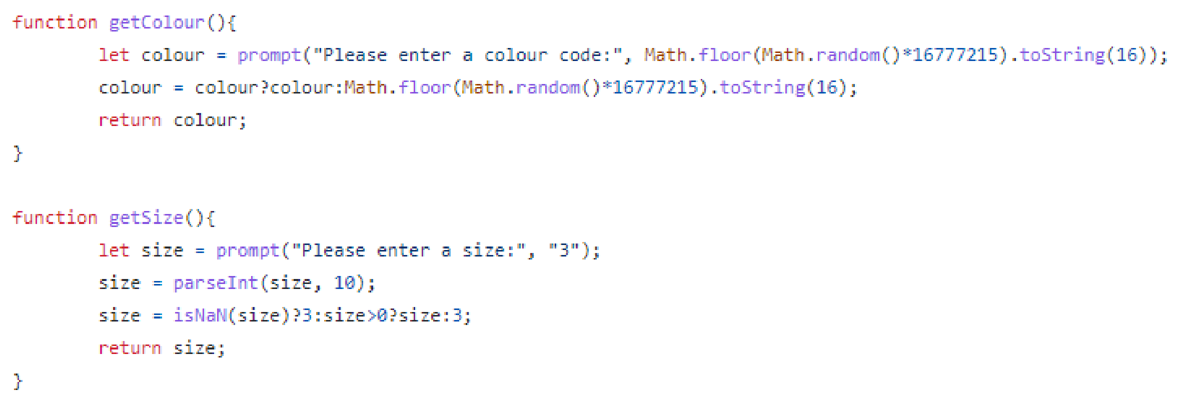
```
function changeWidth(shrink = false){
        width += shrink?-1:1;
        document.getElementById("width").innerHTML = width;
        drawGrid();
}

function changeHeight(shrink = false){
        height += shrink?-1:1;
        document.getElementById("height").innerHTML = height;
        drawGrid();
}
```

Figure 5: Function that takes the input from the user to determine the size of the board.



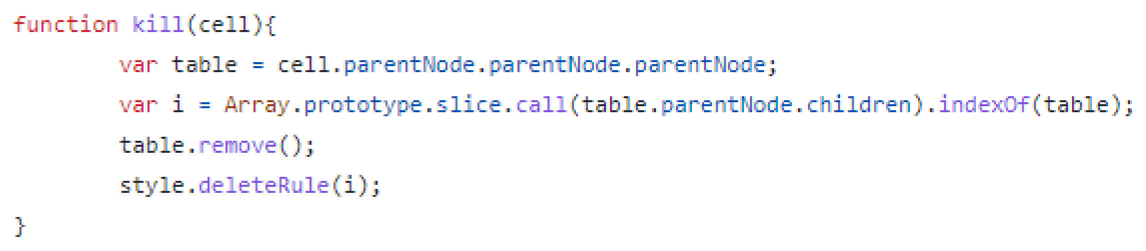
```
function addShape(size = false, colour = false){
        size = size?size:getSize();
        colour = colour?colour:getColour();
        var node = document.createElement("table");
        var shapes = document.getElementById("shapes");
        var cell = ++numShapes;
        node.setAttribute("value", cell);
        var html = "";
        for(var i = 0; i<size; i++){
                html += "<tr>";
                for(var j = 0; j<size; j++){
                        html += "<td class=\"cell"+cell+(Math.floor(Math.random() * 2)?" filled":"")+"\" onclick=\"toggleShapeTile(this)\"></td>";
                }
                html += "</tr>";
        }
        node.innerHTML = "<tr>"+(size>1?"<td class=\"empty\" colspan="+(size-1)+"></td>":"")+"<td class=\"kill filled\" onclick=\"kill(this)\">X</td></
        shapes.appendChild(node);
        style.insertRule(".cell"+numShapes+"{background-color: #"+colour+";color: #"+colour+";}",style.cssRules.length);
}
```

Figure 6: Function that will display a cross size that allows the user to click on to add a shape they want.



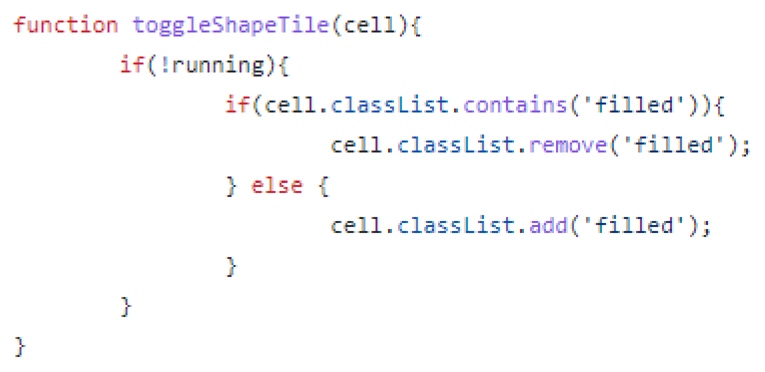
```
function getColour(){
        let colour = prompt("Please enter a colour code:", Math.floor(Math.random()*16777215).toString(16));
        colour = colour?colour:Math.floor(Math.random()*16777215).toString(16);
        return colour;
}

function getSize(){
        let size = prompt("Please enter a size:", "3");
        size = parseInt(size, 10);
        size = isNaN(size)?3:size>0?size:3;
        return size;
}
```

Figure 7: These 2 functions will run when the addShape() is called every time so that the user can determine the size and colour of that shape.



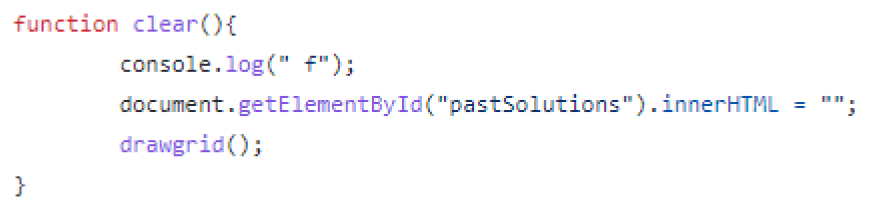
```
function kill(cell){
        var table = cell.parentNode.parentNode.parentNode;
        var i = Array.prototype.slice.call(table.parentNode.children).indexOf(table);
        table.remove();
        style.deleteRule(i);
}
```

Figure 8: This function allows users to delete the shape they do not want.



```
function toggleShapeTile(cell){
        if(!running){
                if(cell.classList.contains('filled')){
                        cell.classList.remove('filled');
                } else {
                        cell.classList.add('filled');
                }
        }
}
```

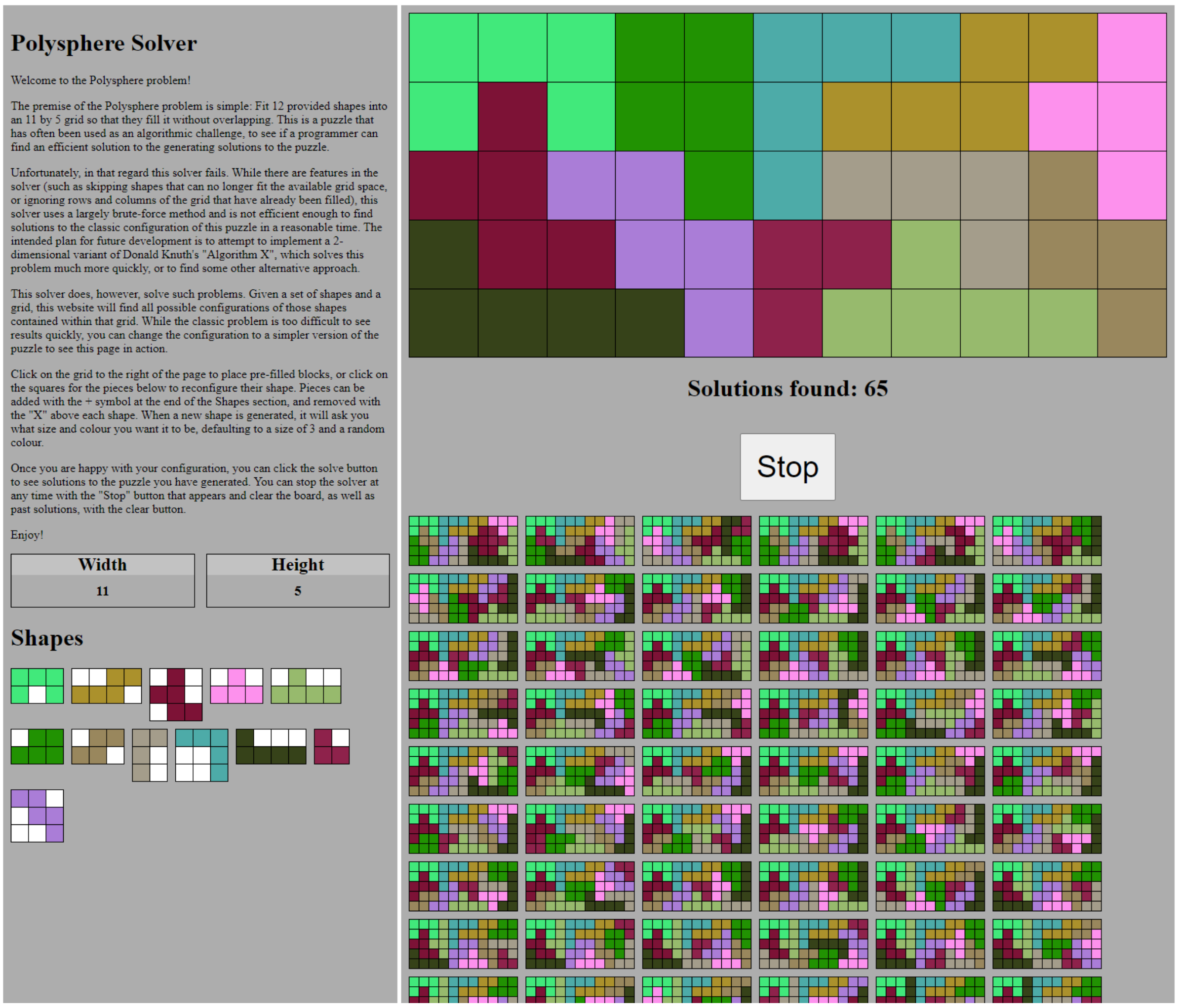
Figure 9: This function allows users to add or remove coloured tile on the shapes or on the board.



```
function clear(){
        console.log(" f");
        document.getElementById("pastSolutions").innerHTML = "";
        drawgrid();
}
```

Figure 10: This function will clear everything that is on the board.

Conclusion for this task is that our approach to this question is not as efficient as it could be. However, it does give the first solution for a 5*11 boards in 20 minutes.



Polysphere Solver

Welcome to the Polysphere problem!

The premise of the Polysphere problem is simple: Fit 12 provided shapes into an 11 by 5 grid so that they fill it without overlapping. This is a puzzle that has often been used as an algorithmic challenge, to see if a programmer can find an efficient solution to the generating solutions to the puzzle.

Unfortunately, in that regard this solver fails. While there are features in the solver (such as skipping shapes that can no longer fit the available grid space, or ignoring rows and columns of the grid that have already been filled), this solver uses a largely brute-force method and is not efficient enough to find solutions to the classic configuration of this puzzle in a reasonable time. The intended plan for future development is to attempt to implement a 2-dimensional variant of Donald Knuth's "Algorithm X", which solves this problem much more quickly, or to find some other alternative approach.

This solver does, however, solve such problems. Given a set of shapes and a grid, this website will find all possible configurations of those shapes contained within that grid. While the classic problem is too difficult to see results quickly, you can change the configuration to a simpler version of the puzzle to see this page in action.

Click on the grid to the right of the page to place pre-filled blocks, or click on the squares for the pieces below to reconfigure their shape. Pieces can be added with the + symbol at the end of the Shapes section, and removed with the "X" above each shape. When a new shape is generated, it will ask you what size and colour you want it to be, defaulting to a size of 3 and a random colour.

Once you are happy with your configuration, you can click the solve button to see solutions to the puzzle you have generated. You can stop the solver at any time with the "Stop" button that appears and clear the board, as well as past solutions, with the clear button.

Enjoy!

Width	Height
11	5

Shapes

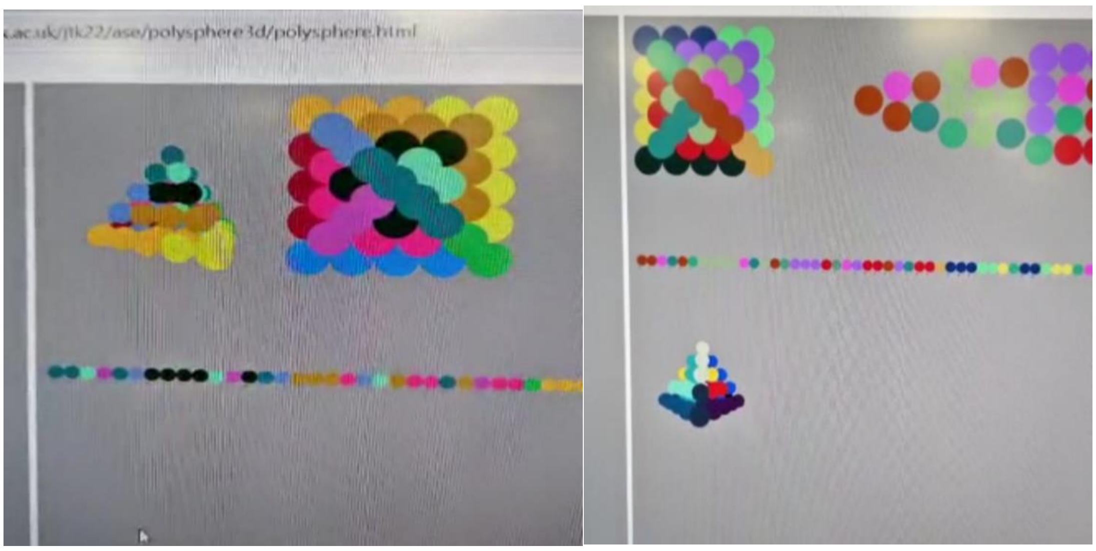
Solutions found: 65

*Figure 11: The application runs for roughly 1 hour and 30 minutes, it generated 65 solutions for 5*11 board with 12 original pieces.*

Deliverable 4 & 5 Pyramid Puzzle and freeform:

For this task we attempted to compete this project however various difficulties were faced. This project without a doubt provided the group with the biggest challenge that this group had faced, it tested and pushed our abilities to the limit.

This project required many brainstorming sessions, and we discussed the exact problem and the implementation of algorithm x to help solve this exact cover problem. So, we then implemented algorithm x through dancing link technique. Because of the problems unique form of requiring 3 dimensions shapes as well as added to the issue that was faced with the fact that shapes had to remain in the exact shape they were left in and cannot be disconnected or reorganised but can only be rotated, we encountered man difficulties along the way. Towards the final stage of testing, it was found that program was in fact running quite efficiently however due the large scope in which we had constructed the algorithm we had an issue with computer memory.



Screenshot 1 and 2: A representation of how the pyramid puzzle solver that we implemented was meant to appear as.

One of the libraries we used to implement this algorithm was a library called web workers. Web workers will allow background operations, every time a solve() function is called on the front page, another solve() function at the background will be trigger to send all possible rotations of shapes to the frontpage. These operations will happen at the same time. In the result of that, the webpage will be more responsive because it is doing with separate thread hence improving the efficiency of front page. In this task, we use MVM model to finish this, but it is kind of difference from the traditional MVM model because it is separatable model architecture that we are using. One thing to mention is that one dimensional array is mainly used for this program, 2-dimensional array is used only when user trying to manipulate the shapes.

For the freeform task. We did a 3D model that can spin around at its own. We were using a library called three.js to make it happen. It is basically a WebGL but is more user friendly to the coder.

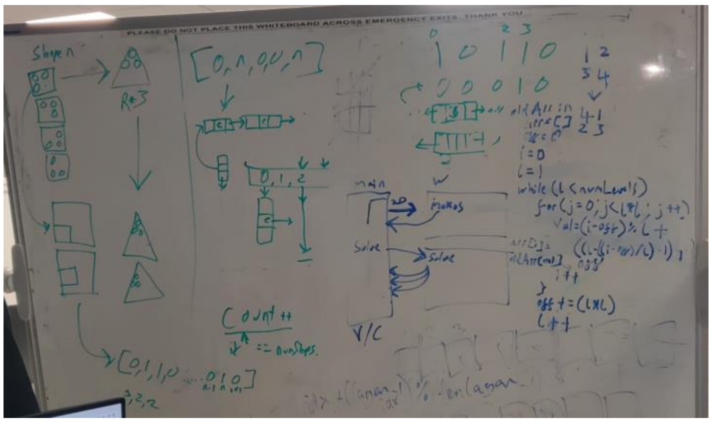
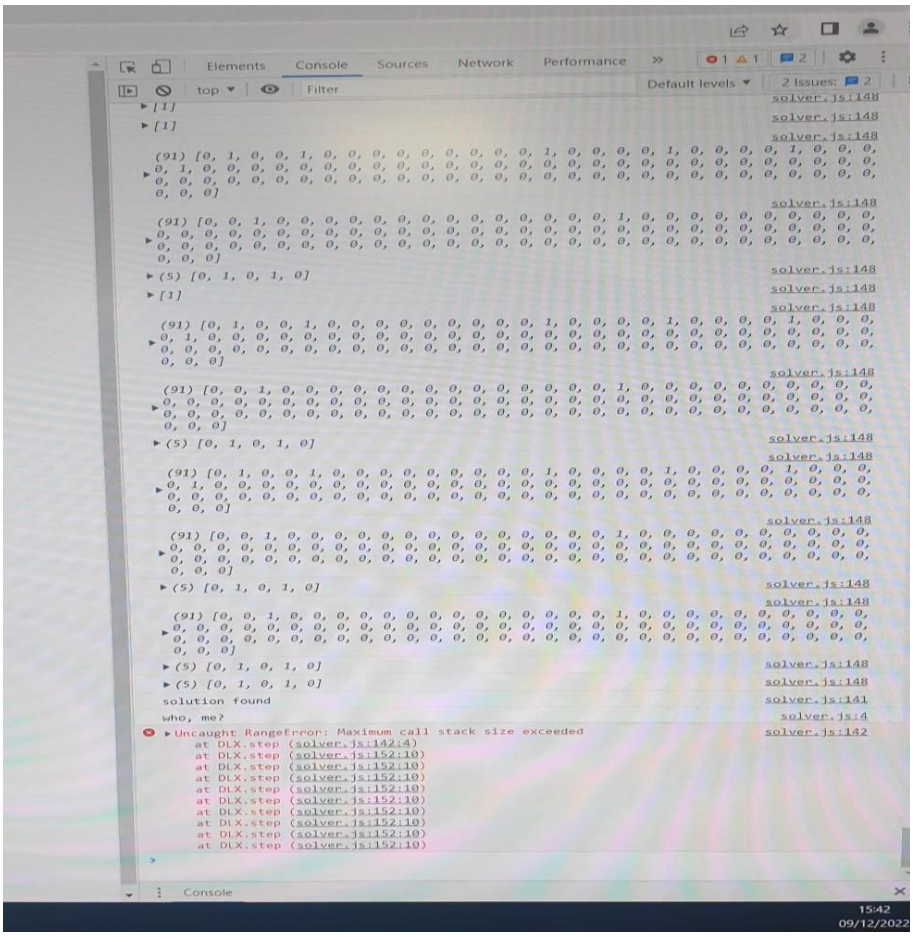


Figure 12: Blueprint mockup of web workers.



```
Elements   Console   Sources   Network   Performance   >>        ⊗1 ⚠1   ▮2   ⚙   ⋮
top ▾   👁   Filter                                  Default levels ▾   2 Issues: ▮2
                                                                        solver.js:148
 ▸ [1]
                                                                        solver.js:148
 ▸ [1]
                                                                        solver.js:148
   (91) [0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
 ▸ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0]
                                                                        solver.js:148
   (91) [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 ▸ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0]
 ▸ (5) [0, 1, 0, 1, 0]                                                  solver.js:148
 ▸ [1]                                                                  solver.js:148
                                                                        solver.js:148
   (91) [0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
 ▸ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0]
                                                                        solver.js:148
   (91) [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 ▸ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0]
 ▸ (5) [0, 1, 0, 1, 0]                                                  solver.js:148
                                                                        solver.js:148
   (91) [0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
 ▸ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0]
                                                                        solver.js:148
   (91) [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 ▸ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0]
 ▸ (5) [0, 1, 0, 1, 0]                                                  solver.js:148
                                                                        solver.js:148
   (91) [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 ▸ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0]
 ▸ (5) [0, 1, 0, 1, 0]                                                  solver.js:148
 ▸ (5) [0, 1, 0, 1, 0]                                                  solver.js:148
   solution found                                                       solver.js:141
   who, me?                                                               solver.js:4
⊗ ▸Uncaught RangeError: Maximum call stack size exceeded                solver.js:142
       at DLX.step (solver.js:142:4)
       at DLX.step (solver.js:152:10)
       at DLX.step (solver.js:152:10)
       at DLX.step (solver.js:152:10)
       at DLX.step (solver.js:152:10)
       at DLX.step (solver.js:152:10)
       at DLX.step (solver.js:152:10)
       at DLX.step (solver.js:152:10)
       at DLX.step (solver.js:152:10)
       at DLX.step (solver.js:152:10)
>
⋮  Console                                                                          ✕
                                                                               15:42
                                                                          09/12/2022
```

Screenshot 3: Proof that algorithm that we implemented is working because a solution was found.

Upon finalising of the product, it was found that while there are still some errors which we are encountering because of the size of the algorithm that we used. We had found that the algorithm which we implemented was working successfully and in fact was able to find a solution before crashing.

Blueprints

Through this image we want to show the logic behind the implementation of 3D PYRAMID. We were trying to find the exact the method so we can traverse each node by the specific formula which is mentioned in the below figure12. We were going through a lot of confusions as it is 3D so we mentioned the nodes on the top layer as “A”, on second layer as “B”, ”C”, ”D”, ”E”, on third layer as “F”, “G”, “H”, “I”, “J”, “K”, “L”, “M” and so on.

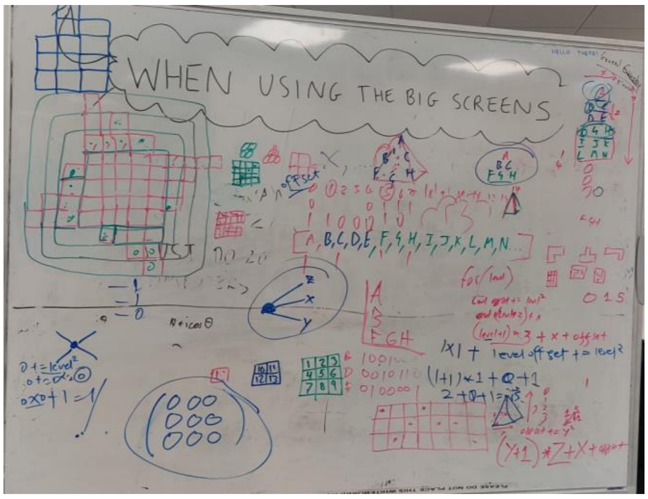


Figure13: It shows the logic behind the implementation of 3D PYRAMID.

We opted to declare the variables "up," "down," "left," and "right" after applying some logic. We attempted to connect every row and column of the matrix using this method as we are constructing the dancing link algorithm. As a result, the matrix's columns and rows will each be circular linked lists connected to one another above by up and down pointers and left and right pointers for each row. For each column list, we also set a unique node called a "list header node." Aside from that, we create three functions in general: two for removing row and column, and one for closing. Therefore, removeRow() and moveColumn() are essentially functions through which we can remove column and all the rows to which nodes of that column belongs to.

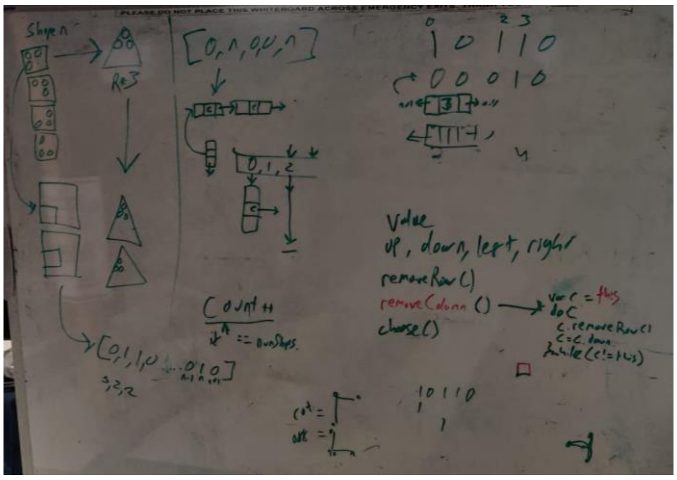


Figure14: The roughly idea of logic and as well as implementation of Dancing link Algorithm.

How Each group member performed throughout the course of this project.

This group project was completed by candidate number 260104, 260121, 184100, 268940

260104(Leo): This student provided much assistance with the deliverable required, as well as a undergoing a fair amount of report writing and as well as providing unique expertise due to his background in BSc degree. Furthermore, this student assisted with uploading some of the work that we completed to the GitHub. Score 9.5/10.

260121 (Priyanshi): this student provided a very well-rounded mathematical knowledge that was needed when completing this project. This student very much assisted in algorithmic knowledge and furthermore was the key reason as to why we had decided to implement dancing links algorithm, because of their understanding of the topic. Score 9.5/10

184100 (Josh): Without this student the project would not have been completed. This student was exceptional throughout the course of this project, providing us with their fantastic coding skills and deep knowledge into the academia of computer science, talking the group through ways in which the problem can be solved while leading us to completion. This student also acted the group leader. Score 10/10

268940(Ziad): this student was a very active member of the group; he was the student that created the GitHub account (deliverable 1). Always ensuring the project was completed, the report was written and that the group was always up to date with what was going on. The member also assisted with brainstorming sessions and in the absence of the designated group leader, he always ensured that the spot was not to be left vacant and that the members of the group who were present were always up to date and constantly working on the problems. Score 9.5/10

Conclusion

In conclusion the group found the project and the deliverables, while challenging to be completed successfully. The deliverable in which we experienced the least difficulty with was deliverable 1 and 2 respectively. While however deliverable 4 and 5 provided the most resistance. The project was a unique opportunity for the members of this group to experiment with their abilities while pushing it to the limit. The members of this group hail from a variety of STEM backgrounds consisting of induvial who specialize in mathematics, web design, artificial intelligence, and computer science for academia.

Using the groups unique skill set and varied opinions about how to complete certain projects, we believe that we have managed to complete the deliverable to our best abilities with the time we had available to us. In the future should we attempt these projects again one thing we should make sure of is to ensure we manage our time more effectively whilst ensuring we study to broaden our knowledge in this filed, specifically our knowledge on solving algorithms. All in all however we consider this project as a success.