

**Declaration of Originality**

I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the regulations on Unacceptable Academic Practice from the University’s Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
* In submitting this work I understand and agree to abide by the University’s regulations governing these issues.

Name: Edward Dugdale

Date: 08/05/17

**Consent to share this work**

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Edward Dugdale

Date: 08/05/17

**Acknowledgements**

A debt of gratitude is owed to Christine Zarges for her consistent support and motivation. Without her expertise, I would not have been able to submit a working game.

I would also like to thank Jamie Bonnett, for testing my application and giving me constructive criticism to work upon. Also, being the light at the end of the tunnel and letting me talk through any problems that occurred, be that in the project or in life.

Next to my girlfriend, Angharad Bache, for testing the application, making me enough cups of tea to sink a ship, and for putting up with my incessant, unending musings about computers and games for the last three years.

Finally, to Aberystwyth University for giving me the skills necessary to complete this project.

**Abstract**

The product I have produced is a playable game of the popular television game show *Countdown Numbers* which is derivative of the game *Countdown*. The show, originally broadcasted on Channel 4, is based off the French gameshow *Des chiffres et des* lettres (Numbers and Letters).

The reason why I have chosen to develop this project is due to an influence early in my life at Sandroyd School in southern England. At the school, Coundown: Number Round was used as an exercise by one of the teachers to practice our mental arithmetic, as well as a game that everyone could enjoy. The competitive element disguised the by-product of learning and as a result of this, I have fond memories of the game. I became not only obsessed with winning but the process that goes with it. This is the fire that captivated me into implementing an algorithm that could solve the game as well.

**Contents**

1. Background, Analysis & Process 6

1.1. Background 6

1.2. Analysis 7

1.3. Process 8

2. Design 10

2.1. Overall Architecture 10

2.2. Detailed Design 10

2.2.1. Even More Detail 11

2.3. User Interface Design 12

3. Implementation 16

3.1. The Base Game 16

3.2. The Algorithm 18

4. Testing 22

4.1. Overall Approach to Testing 22

4.2. Automated Testing 22

4.2.1. User Interface Testing 22

4.2.2. Stress Testing 23

4.3. User Testing 23

5. Critical Evaluation 25

5.1. Were the requirements correctly identified? 25

5.2. Were the design decisions correct? 25

5.3. Could a more suitable set of tools have been chosen? 25

5.4. How well did the software meet the needs of those who were expecting to use it? 26

5.5. How well were any other project aims achieved? 26

5.6. If I were starting again, what would I do differently? 27

6. Appendices 29

A. Third-Party Code and Libraries 29

B. Ethics Submission 30

C. Code Samples 32

7. Annotated Bibliography 33

# Background, Analysis & Process

## Background

Games as we know them today take their origins from the pre-historic roots of dice. Dice existed before man had even completed their foundation for language, ‘A series of 49 small carved painted stones were found at the 5,000-year-old Başur Höyük burial mound in southeast Turkey. These are the earliest gaming pieces ever found.’[[1]](#footnote-1) It shows that the link between numbers and games have been hand in hand since the inception of humanity. However, the mathematical logic that accompanied the dice was misrepresented over time into perceptions of fate and divine intervention. A notion that is carried to modern day through gambling and the term ‘lucky’. To steer as far away as possible from that concept, I planned for my game to be won by skill rather than probabilities.

To prepare for the upcoming project, I made myself familiar with the rules of Countdown: numbers. I read a variety of sections from ‘Countdown Spreading the Word[[2]](#footnote-2)’. This book describes each, and every, detail of the game so I would be prepared for what the requirements would need to be. This in turn, allowed me to make my algorithm more efficient, this will be explained further on in the report.

I also watched a few episodes of *Countdown Numbers* to see how it is laid out and get a better grasp of how it is played on television. I used this as motivation for my project and took away the colour scheme and design of the logo for inspiration.

I have always enjoyed Countdown Number Round, both because I find the concept great fun, but also because my mental math is one of my more developed abilities. As a result, the option to create Countdown Number Round made a lot of sense for topic for me to choose.

Choosing a language was difficult, I looked at Java, C++ and JavaScript as I am familiar with all of these languages. Java and C have the benefit of being more powerful languages some completing the goal would be slightly easier. However, I am better at developing Graphical User Interfaces (GUIs) in JavaScript.

I first decided to use Java, this is because I am most familiar with Java and I know it is more than capable of completing the task. Nevertheless, I ran into an issue further down the line that required I changed languages or set up my GUI in Java Swing. My issue is that I have never really used Java Swing so it could cause more problems later. Because JavaScript is so similar to Java, it was a fairly simple swap to convert all of my work that I had already completed over.

I ran into an issue using JavaScript when I wanted to use it, and that was I wanted to make a desktop application instead of having to make it like a website. Because I wanted to develop it to run like a desktop program, I needed to use a framework that would allow me to run a desktop style app in JavaScript. Electron is a new software that has been taken to by storm, allowing users to make use of Chromium to create their own cross-platform desktop applications. It uses both Chromium and Node.js to do this. It is cross platform so it would be usable on Mac, Windows and Linux.

Electron is used by a few successful applications already, one of these is Discord. Discord is a program that allows users to voice chat, it allows you to be part of many different channels and to change channels quickly. Although not a new concept, it is still more efficient than many other forms of communication such as Skype and Teamspeak. Also, because it is done in electron rather than a standard program, it is cross platform so there are not too many different versions of it. Some programs will be tailored only to one operating system.

What was your background preparation for the project? What similar systems did you assess? What was your motivation and interest in this project?

## Analysis

The problem I have been set with is that after the user has had their numbers, and the target number generated, a solution had to be found. The solution must utilise the user’s numbers to find the closest possible answer to the target number, and how many ways it can reach that number. Another thing to note is that a user does not need to use all of their numbers to reach the target number. For example; if the user had to create the target number of 150, and the user had a 50 and a 3 in their selection of numbers, they could solve this problem with just one turn. It is very rare that this happens, however it does need to be accounted for.

My original idea to solve this issue was to use A\* search. It is a shortest path algorithm that would be implemented after already plotting all possible numbers. The numbers would be placed into a tree with all the possible combinations, and A\* search would find the quickest way to the number. Although all ways of getting to the answer are recorded, it is best to print off a best solution. The reason why I chose A\* search originally was because it is one of the most efficient shortest path algorithms to date. It is used in many different scenarios, including video games. The reason why it is so efficient, is because you can tailor the algorithm to what you need it to do by adding heuristics.

To implement A\* search I would first need to create a tree out of all the possible user number combinations and then use A\* search to find the shortest path down the tree. This would find the solution with the least number of moves to display to the user once they had submitted their answer.

After completing my research my first thought was to create a standard program that will generate the target number by using the user’s numbers in a random operation. There were however parameters to stop results that would be unreasonable. First off all, I needed to make sure that the target number could not reach a negative number as countdown prohibits any negative numbers. Countdown also prohibits decimals, so using modulus worked well here. The final parameter I needed to implement was to make sure that the target number stays above 100 and below 999. This was simply done by placing the construction of the target number in a while loop.

The analysis I made helped me decompose the problem by not only deciding on which algorithm would be best implemented to solve the issue, but ways of making it more efficient as well. Finding the nearest possible answer could be difficult especially with the amount of numbers that can be made. Using factorial function, it is easy to see just how many possibilities there are, however permutation without repetition must be used, as you can you cannot use the same number twice.

## Process

I chose Scrum to be my agile methodology, this is because I work best when tasks are laid out clearly with estimated completion dates. Scrum uses small iterations of work known as ‘sprints’ to organize the work flow.

Figure 1: The planned completion dates versus the actual completion dates

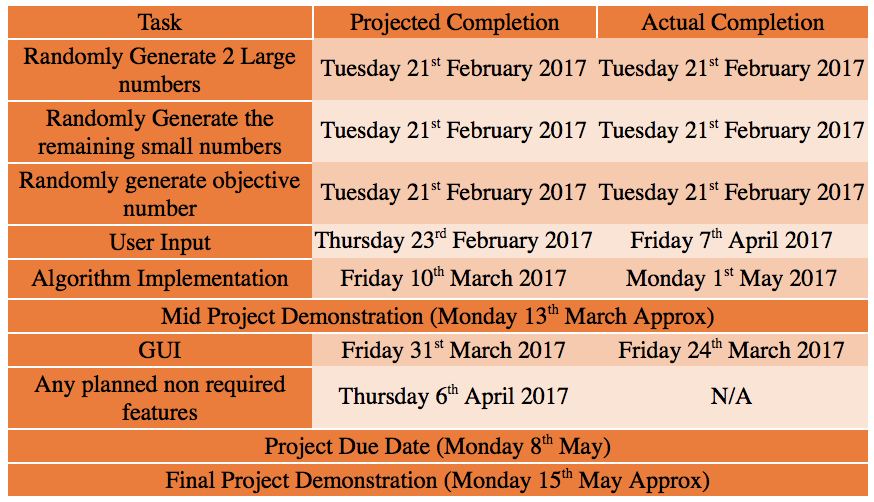


Figure 1 shows what I had planned for my sprints versus what actually happened. There were some modifications for certain sections such as “Randomly generate 2 large numbers”, this is because according to Countdown rules, the user choses between zero to four large numbers. I had originally chosen to set it to two because this is what I was used to whilst I played it at school, however this could be because I was fairly young so the teacher may not have wanted to have set too many large numbers.

Also, it is clear from figure 1 that certain sections were swapped around. I swapped the GUI to be completed just after the mid project demonstration, this is because I had already completed the basic way of playing the game without the implementation of the main algorithm. I decided it would be best to do the GUI then so that I had something working that I could demonstrate, if anything were to go wrong.

I implemented the algorithm last and this is because, although I understood how A\* search would work with my situation, I struggled with thinking of how to implement it. By completing this last, it allowed me to focus on the core playability of the game with the user being able to select how many large numbers they would like as well as being able to submit an answer.

Scrum allowed me to keep a track of what I was doing as well as what I needed to do next. I would also feedback to Christine Zarges (my project supervisor) as to what I had completed and what I needed to do next. Christine acted as the Scrum Master for my project my keeping me on task and focused.

# Design

## Overall Architecture

The architecture of my application is reliant on Electron. Electron uses Node.js as the back-end of the application whilst Chromium runs the front-end. An Electron application relies on three core files, “package.json” which is a metadata file that stores the information about the package. It stores the name of the application, the version string, and the name of the main script file that it will run. For my project, the main file is index.js however I have many different JavaScript files, this is because index.js will run as the main file and the others are referenced in an object orientated manner. The package.json file is a npm file, this is a file type that is included in Node.js. NPM stands for node package manager, it is required for Electron, because Electron is a package.

The second file in the set of core files is the main JavaScript file, in my case this is index.js. This file is initiated by package.json and will then be the base of the JavaScript.

The final core file is index.html. This serves as the Graphical User Interface (GUI). It sets up the divs for JavaScript to later give a value to or to manipulate to create the GUI I have designed.

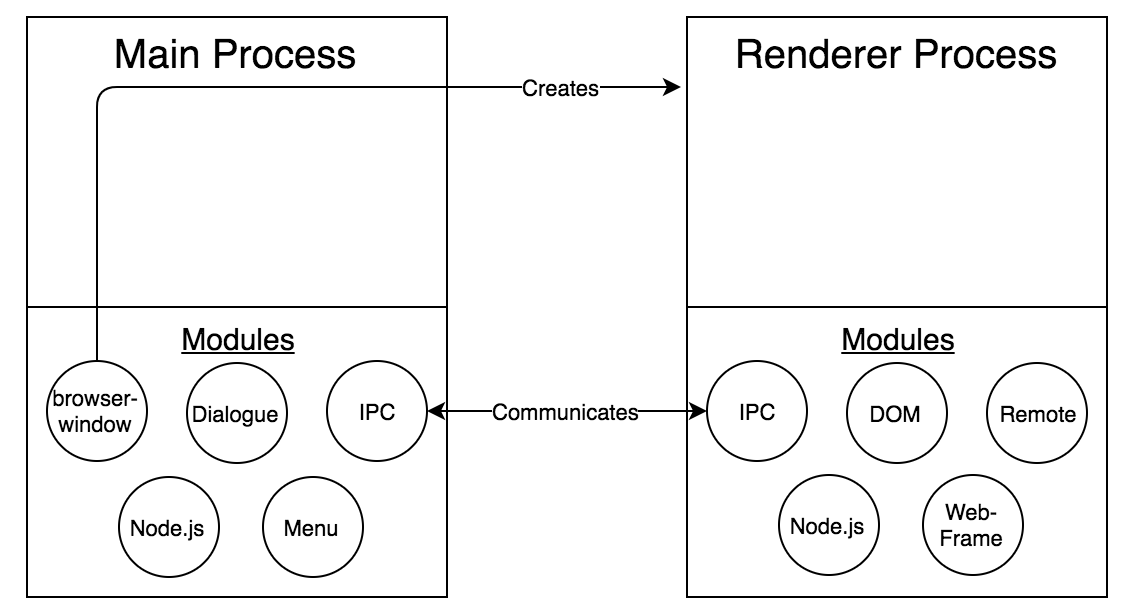
## Detailed Design

Electron uses two processes to run, one is the main process and the other is the renderer process. The main process is the lifecycle of the whole application and controls when the application is starting or ending. It has different modules that are built into electron specifically for the main process.

The dialog module allows the interaction with open file dialogues and system user interface elements. The menu module allows the creation of system level menus as well as there being full access to the node application programming interfaces (APIs). Next is the Inter Process Communication (IPC), this module handles asynchronous and synchronous messages sent from the renderer process. The last module is the browser-window; the module allows the creation of the renderer process.

The renderer process also has the IPC module, this allows the renderer process and the main process to communicate with each other. Figure 1 shows the lifecycle of Electron and how the IPCs link.

Figure 2: UML Diagram of Electron’s lifecycle



I have not really used Electron to its full potential in my project, I have used Electron to set up the window as a desktop based application rather than using all of the modules it provides. This is because for what I wanted to program to do, it wasn’t necessary as the program could be made solely in JavaScript without any issues, however I wanted to develop the application as a desktop based application rather than a website.

### Even More Detail

When I first started, the design was to originally create the project in Java, however due to Java Applets not being applicable for my project as they are outdated as well as my lack of experience with Java Swing, I needed to rethink my approach and ended switching to JavaScript.

When I started the design of my project I needed to make sure that I had a good base to work off from, so instead of diving head first into the algorithm, I thought it wise to test the waters instead. I first set out to create the target number as well as the solution in one go, this was not the main goal of the project, but would give me a good starting point. I managed to do this by randomly selecting the user’s numbers and a random operator, performing the equation and adding the number back to the user’s numbers until all numbers had been used. I had to make sure there were some parameters to create the target number, as to not get any results that would be against the rules of Countdown.

Because of having this original design, as well as then having to create an algorithm later that would randomly generate the target number and search to see if it was possible to be created from the user’s numbers, it ended up being fairly object orientated (OO). Because of it being object oriented, I needed to design a logical set of classes.

First of all, I had the main class that would call the other classes when required as well. Then I would create a class to create the user’s numbers, this would work by taking the user’s choice of how many large numbers they desired, followed by filling the rest in with smaller numbers.

Once the user’s numbers were created there would be a class that would call the user’s numbers to create the target number and solution to create the foundation of my project.

There would also be another class that would contain the algorithm that would solve a randomly generated target number using the user’s numbers. This class would also call the user’s numbers so that it can try and do this.

## User Interface Design

For the design of the game, I originally wanted to create it using Java Applets. As a result, I didn’t want to make too many different pages and have it all set up on one instead.

The original layout is depicted in figure 3, it was going to have a main section called “solution”, this box would contain the target number at the top and then would have the solution printed out after the user had chosen their answer.

The menu bar across the bottom of the screen was going to be where the user selects which difficulty they would like or when to start or restart the game.

On the right hand side of the GUI there were going to be two different boxes, the “user numbers” box which is self-explanatory and would contain the user’s numbers.

The second box is the sidebar which would contain the equations the user had made so they could keep track on what they had done. This would allow users to be able to remember which numbers they had originally, so if they wanted to backtrack on what they had done.

There is also a dark blue to light blue gradient behind it all, I decided that would be a good idea because otherwise it would be too bland. However, upon creating the design I stuck with later on, it does seem like a bad design when I look back.



Figure 3: Original design for the Graphical User Interface

I later decided that this design was an inefficient use of space as the solution section would be used once and the user would be limited to a small on the side of the screen, this means that the application would have to be full screen for the user to play and would limit the functionality of the application.

Also Java Applets is an outdated part of Java, this means that I needed to redo the relatively small amount of code that I had produced by this stage. I chose JavaScript to continue my project with, as some of the concepts of Java and JavaScript are very similar.

Because I had chosen JavaScript the program would be run in a web browser as opposed to a desktop application, this is the reason why I then chose Electron. It means that I had all the benefits for GUI design that CSS, JavaScript and HTML give me but with the added benefit of having a nicer looking desktop application.

Because of the new choice of language, I redesigned the GUI layout with a much more attractive theme. I still kept the blue design idea but without the gradient. This is because the Countdown theme is a blue background with white boxes with black font inside the boxes.

I decided that my redesign of the application should allow for a user to be able to play the game in full screen as well as a windowed application, this is made easier using electron and bootstrap to make the application reactive. Electron also allows me to remove the title bar that would normally be there during a webpage, this makes the application look a lot smoother.

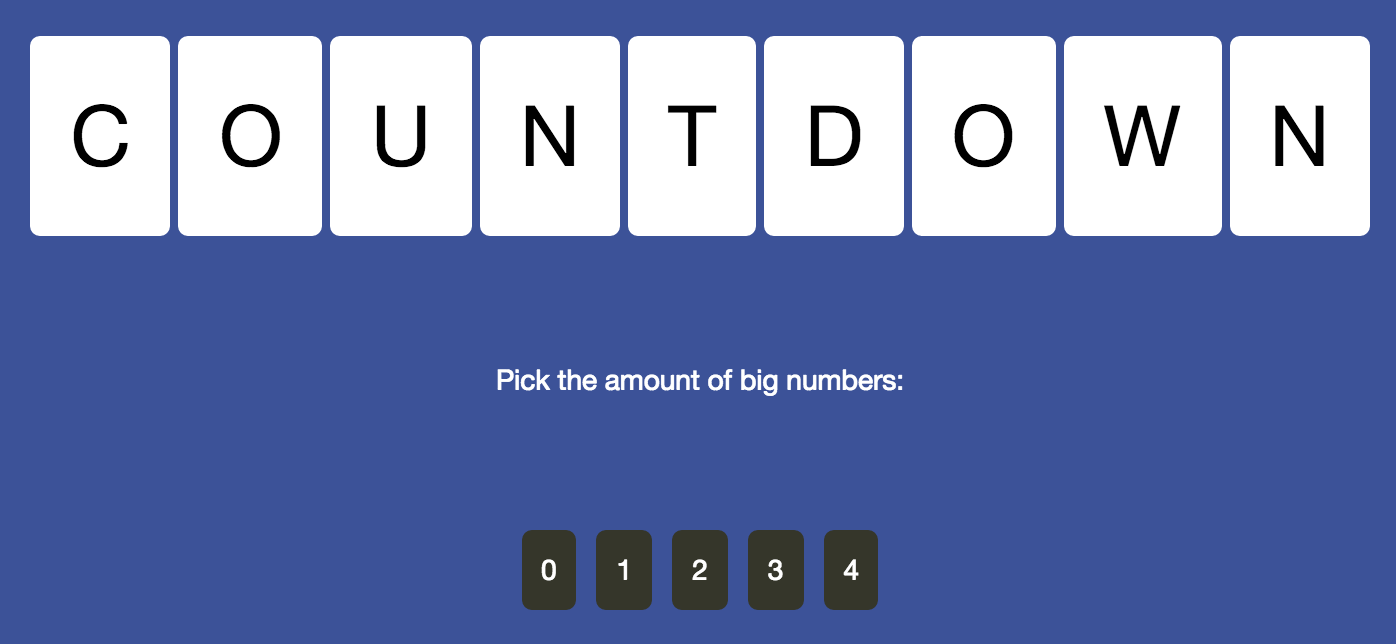
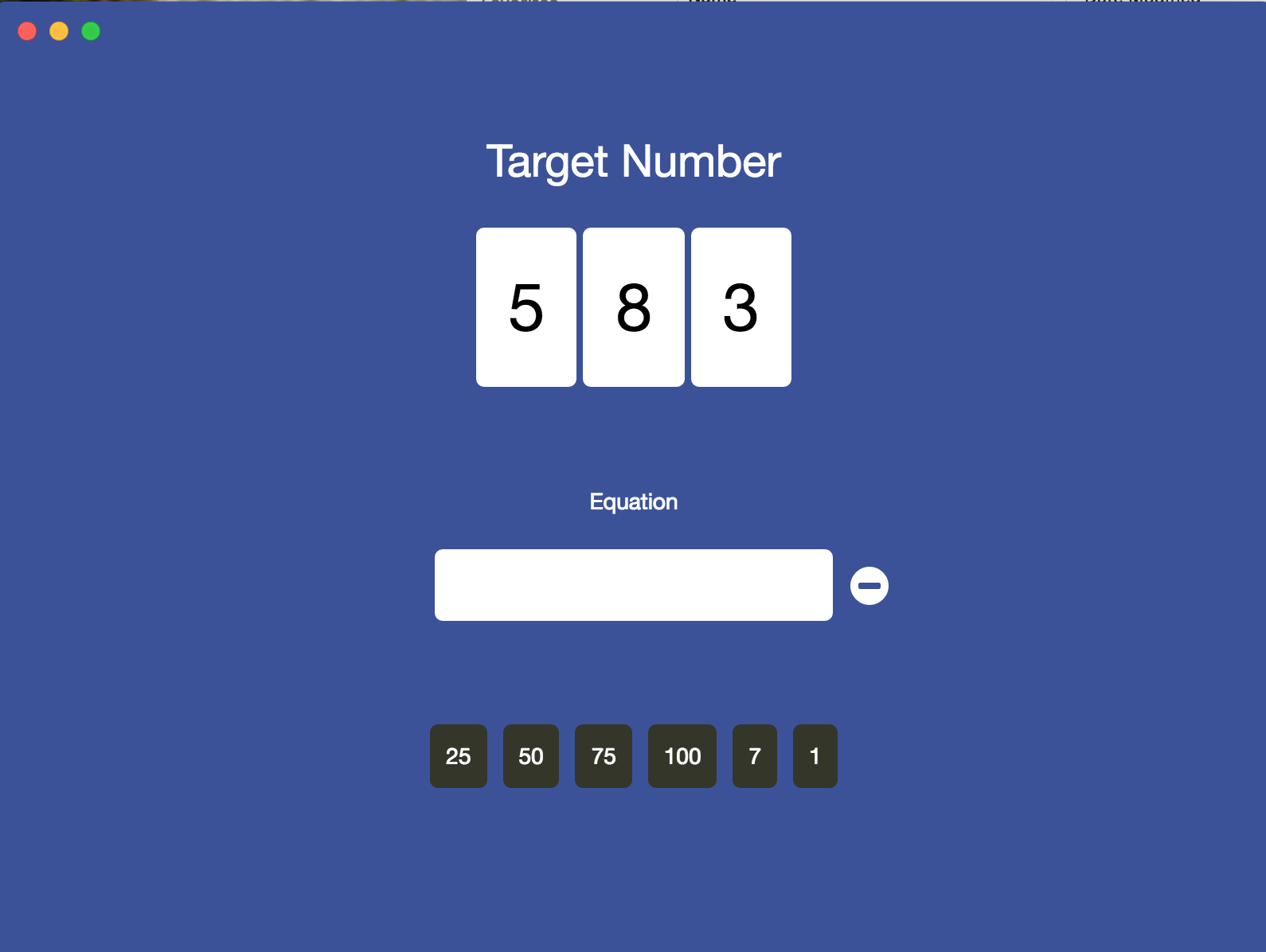


Figure 4: The design for the page where the user can select their large numbers

Figure 4 shows the new design I have chosen for the user to select their numbers, It keeps the Countdown theme with a blue background that is not overpowering, with white boxes and black writing. I chose to place the user’s interactive buttons in a dark grey box with white writing so that it looks different to the rest of it. This is so the user can clearly discern that they are buttons.

Figure 5: The playing stage of the game



In Figure 5 it is clear to see that the title bar is not present in the application, the Mac OSX; close, minimise and maximise buttons are all placed in the top left of the application. These clearly show that although the title bar is not there with the applications name (e.g. the name of the folder in finder) but does however have the ability to close the application.

The target number is generated at random and placed in white boxes with black font in the horizontal centre of the screen. Below is the equation box, when a user clicks on one of their numbers, it is “pushed” into the equation box. The numbers will then swap to operators so the user can select which operator they would like to perform.

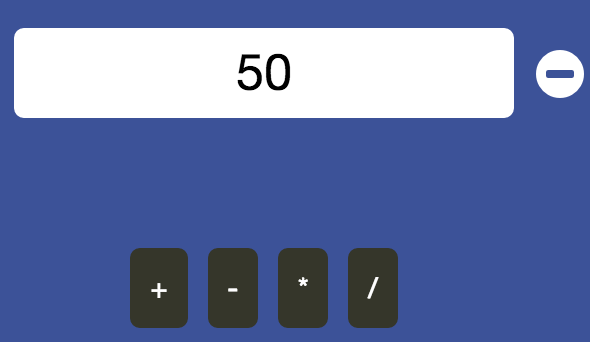
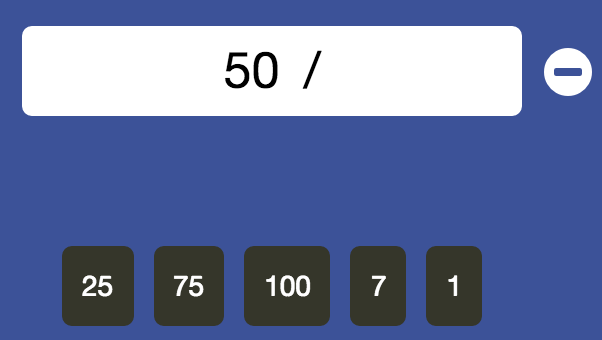


Figure 6: The user selecting an operator

In Figure 6, the user has selected 50 as their first number, they can then select an operator to perform on the number. To make the numbers disappear and instead have the operators take their place, I used a button swap state.

If the user were to change their mind about which number they had selected, they could click the remove button on the right hand side of the equation box. This looks like a minus symbol inside a white circle. The reason I have chosen this design is to show the symbol they are removing something, and by placing it next to the equation box should show the user that it is removing something from there.

Figure 7: The user finishing the first equation



In Figure 7, the user has selected a division operator to perform on their first number, this will then bring them back to the number selection screen. Note that the first number the user selected, in this example, 50, has been removed from the options for the user. This is because according to Countdown rules, the same number cannot be used more than once.

# Implementation

## The Base Game

When I first started my project, I decided to make a base game which would not implement a sophisticated algorithm to solve the problem, but rather solve the problem whilst creating the target number at the same time. To do this I would use the user’s numbers to create the target number by cycling through using random operators to perform the actions and once all numbers were left, that was the target number. This means that by storing each step in a Tuple object, I would also have the answer saved.

The reason as to starting my project like this, allowed me to get buttons and user input working correctly as well as designing a GUI that looked good for the task at hand.

The first step in all of this was to create the user’s numbers. When I was first planning my project, I was originally looking at only having two large numbers for the user to have as well as four smaller numbers. However, I decided it would be wiser to implement the user input at the start rather than refactoring the code later on. This means that once the program was run, it would prompt the user to answer how many large numbers they would like. The difference with the large numbers is that I had to place 25, 50, 75 and 100 in an array and then remove them from that string and place it into the user’s number array. This is because if a user were to choose 4 large numbers, they would get 25, 50, 75 and 100 as opposed to randomly selecting out of them four times. This is due to the fact that numbers cannot be duplicates whereas the smaller numbers between 1 and 10 can be. Therefore, to create the user’s smaller numbers I would take six and subtract the choice they made of big numbers (e.g. if the user selects to have four big numbers, the code would effectively run 6 – 4).

After the user numbers had been created, I needed to create the code that would create the target number within some set parameters using the user’s numbers.

This was fairly straight forward, I had to use two while loops. The first one would have the parameters that whilst the target number is equal to a value less than 100 or greater than 999 it would loop through the while loop. At the beginning of the while loop it would set the user’s numbers back to what they originally were and would also set the operations to be blank.

After this there is the second while loop, nested within the first. This while loop continues to loop through until the user’s number array length is equal to one. In this while loop I will need to loop through the array, so I need to create two variables that will lock onto the indices of the array. These are indexX and indexY. IndexY will always be larger than indexX. Then I need to get the value of indexX and indexY so I can perform the operations on them. So I set x and y to be equal to their respective values. Then I need to make sure that certain operators cannot be used depending on the values of x and y. For example, if the value of x is less than y, then subtraction cannot be used and neither can division. I also need to make sure that division does not leave the result in decimal numbers, so I used modulus to make sure that if x is divided by y, then if the result is not equal to 0, then division cannot be performed.

To do this I create a variable called numOperators, this variable will be equal to a maximum of four. If certain operators cannot be performed such as the number would end up negative or in decimal places, then the numOperators is changed. If the equation cannot be completed using subtraction, then the numOperators is set to two and if it is not able to be completed using division, then the numOperators is set to three.

The reason as to why this variable is being set to these values is because next I use a switch statement. This switch statement will randomly select a case number from one to four and then apply the operator, one being addition, two is multiplication, three is subtraction and four is division. These switch statements had to be in this order as division is not possible if subtraction is not, so division goes last. Also addition and multiplication can always be performed so subtraction will go after these.

Using a simple equation of;

let operator = Math.floor(Math.random() \* (numOperators - 1 + 1)) + 1;

This will set a variable; ‘operator’, that will have the value ranging between zero and four depending on the value of numOperators, the reason why I need this despite having a section that determines the number of operators on which ones can be performed, is because I need to be able to randomly select one of the operators from the ones that are possible to be executed properly.

Each switch statement case will perform the relevant operation on x and y and then set that as the value of numberOut. It will then store the value of x, and y and the operator that was used in a tuple within the Tuple object. It will then print the value of x, the operator used and the value of y followed by the result.

To conclude the while loop, the numberOut that is produced from the switch statement is added to the userNumbers array to be looped through again. Also the number at indexX and indexY are removed from the array. In essence, the numbers used to calculate the new number, are removed and the new number is inserted.

As aforementioned, this process continues until there is only one index remaining in userNumbers. This number is then checked and if it is between 100 and 999 then the number is accepted and the game begins with that number as the target number, if the number is not within this range, then the while loop begins again with the same userNumbers that were generated originally until a valid target number is created.

## The Algorithm

The algorithm has three objects that it used throughout its lifecycle. These are; “number\_value.js”, “operator.js” and “combination.js”.

The “number\_value.js” file stores the value and has the functionality to convert the number (e.g an integer) to a string. It also checks to see if the value that is being passed in is equal to the value set at the top of the object. It also checks to see if the value being passed in is the same value as NumberValue (a constructor set at the beginning of number\_value.js).

Next is “operator.js”, this sets the operators so I do not have to set them dynamically throughout the program. The operators are set using the correct mathematical symbols, for addition and subtraction this is easy as we can just use a predetermined string of ‘+’ and ‘-‘, however for multiplication and division it is a little harder. For multiplication and division to be correctly displayed to make the game more user friendly, I had to use Unicode to set the display of the operator.

The object is also calculating the result itself my taking x and y and performing the given operator on them. It is also looking to see if the operation is commutative, this checks to see if the operation is the same either way around (e.g. x + y and y + x). This later on stops the duplication of operations to make the algorithm more efficient.

Finally, within operator.js there is a function named “canCalculate”. This stops illegal or redundant operations from being performed. For addition, it is set to return true, this is because there are no illegal or redundant operations for addition. For multiplication however, you can’t make a negative number or decimal place, the algorithm stops the multiplication of one. This is because there is no benefit of multiplying by one as the user can submit an answer without using all numbers and the result of multiplying by one will just return the same number. For subtraction it makes sure that x is greater than y, this is to make sure that no negative numbers are created as this would break the rules of Countdown. Finally, there is division, it makes sure that again you cannot divide by one as it would be a redundant move as well as stopping from diving into decimal places by using modulus to check to see if the result would be a decimal or not.

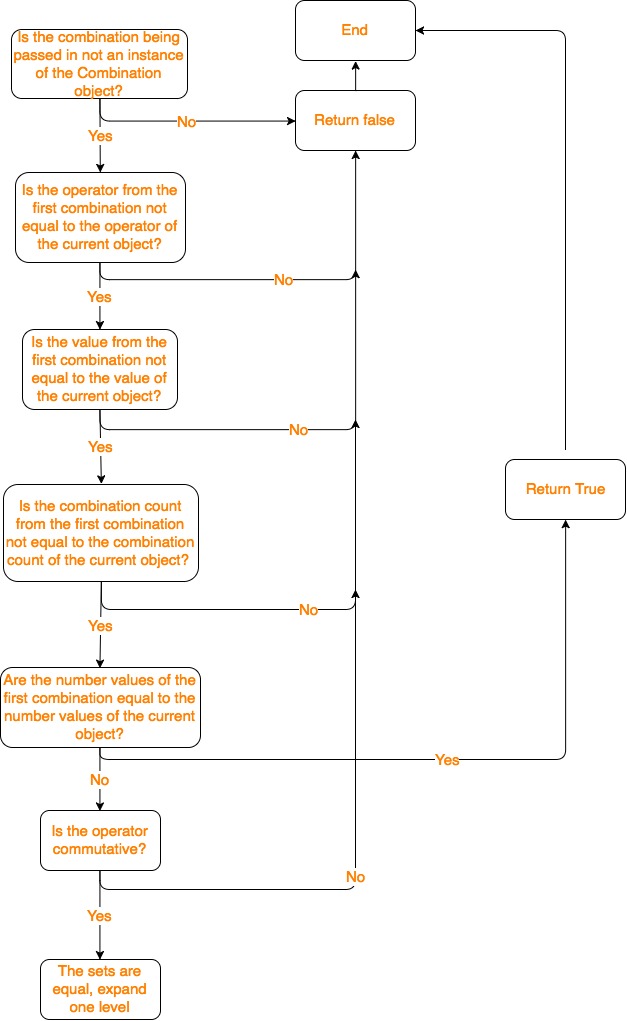


Figure 8: A flow diagram of how I check previous combinations

The final object that the algorithm uses throughout its lifecycle is “combination.js”. This object has a few different sections to it.

The first section checks to see if the combination that is currently being processed is equal to the previous one.

To the right is Figure 8 which shows a flow diagram of how this function is working within combination.js. It starts by checking if the combination that is being passed into the function is not already an instance of the ‘Combination’ object. If it is already an instance, then it will return false and the process ends with that check. If the combination is not an instance, then it will move onto the next check.

The next check is too see if operator from the first combination is not equal to the operator of the current object. If it is equal to the current object’s operator then it will return false, and end the process of checking that combination. If it is not equal to the current object’s operator, it will move onto the next check.

If the value from the first combination is not equal to the value of the current object, then it will move onto the next section, however if they are the same, then it will return false and end there.

If the combination count from the first combination is not equal to the combination count of the current object, then it will move onto the next section, whereas if it is the same it will return false and end the check.

At the next stage, instead of checking if one value is not equal to the other value, it checks to see if the number values of the first combination are equal to the current objects values. If they match, then it returns true and ends the check. If they do not match, it checks to see if the operator is commutative. This is because the numbers may be equal but the wrong way around. If the operator is commutative, then the sets are equal and it will expand one level however if the operator is not commutative that it returns false and end.

The ‘expandOneLevel’ function checks to see if the first value is an instance of the current object, and if the second values operator is equal to operator, the it will add both value one and value two to the first values array, but if one of them isn’t true, it just adds value one. There is a sample of the code in the “code samples” section of this report.

After it has done that, it does the same thing but checks to see if value two is an instance of combination, and checks to see if value two’s operator is equal to operator, if these are both true, it adds both value one and value two to the second values in the values array, otherwise it adds value two. There is a sample of the code in the “code samples” section of this report.

The next step is checking if the arrays are equal, not just in length but also in content and that is precisely what arraysAreEqual does. First it performs an if statement that checks if the first array’s length or “arr1” is not equal to the second array’s length or “arr2”. If they are not equal, it will return false. Then it will check to see if the first arrays length is equal to 0, if it is it returns true. The finally it uses a nested for loop to perform a two dimensional array to see if there are any values within the first and second arrays. If they are, it will return true. If it gets to the end of the function and they are not the same, then it will return false.

Now that I have explained the three core objects that the solution uses, I can begin to explain how my code solves the issue. Within the solver.js object, there is a function there is a function called “getCombinations”, this is where the magic happens.

It starts be setting some variables at the top, however further down there are three different nested for loops. The first two for loops are setting up a two dimensional array, the first sets the value of i1 to be 0, it then sets ‘len’ to be equal to number of values within the values array. This is so it can slowly iterate over the array, what’s important in this section is that it also enters a variable called i1Max. This value is equal to len – 1, this is because in the second for loop, there is a value declared, i2. This variable is equal to i1 + 1, this is because they are both set to be indices in the array, hence why they are i1 and i2. This is also why I have set i1Max to be equal to len – 1, because it needs to have a point where i2 cannot exceed the maximum length of the string, so i1 needs to stop 1 before the end.

In the third for loop, the solution to the problem is created. It will loop through i1 and i2 with every operator to create every possible value. So first i1 will start at index 0 of the user numbers, this being the first item in the user numbers array. Then i2 will start at index 1, the second item in the array. It will use every possible operator whilst taking into account the operators that cannot be run or would be pointless to run (multiplying by one for instance).

Once they have run every possible operator, i2 will increment by one, this means it will move from the second item in the array to third, whilst i1 remains at position 0. It will perform all the applicable operators once more and move on. Once i2 has reached the end of the array, i1 will increment by one to index 1 in the array (the second item). Then i2 will start at one higher than i1 and loop through the array once more.

The ‘combineIfValid’ function is being called in the third for loop. Each time an operation is made, if the operation is valid, it will the operator, val1 and val2. Val1 is set to be equal to the index that i1 represents within the values array. The i2 variable does the same for val2.

It then loops through the array again with the numbers that have been changed, to create a combination for the solution.

Below these is a loop that checks to see if the solution generated from the for loop is within ten of the target number, if it is, then it is added to combination.

At the bottom of the solver.js there is a function called sort. This sorts through all the combinations and places the ones closest to the target number first. This is because my algorithm will find every possible number within 10 of the target number, the closer it is to the target number the closer it is to the top of the list of the solutions (with exact matches being at the top).

In the sorter.js object, there is a function called solve. This function has a for loop that will loop through the solutions to see if the solution I have is the closest solution, if it is not and the new one that has been presented is closer, then the current solution is overwritten to equal the new solution.

And that is how my algorithm works. It is not as efficient as A\* search would be, however it does find the closest possible numbers and ranks them in order of which ones are best. It does complete the requirements of the project however, it could be more efficient.

# Testing

## Overall Approach to Testing

To test my work, I first had to think not only about what was required of me, to make the project what it was designed to be, but also about what I wanted from the project. The project clearly stated to make a playable version of Countdown number round, another requirement is that it would have an algorithm find a solution to a randomly generated target number.

I needed to make sure that the tests would show what I have accomplished in terms of what was set as a task, as well as what I wanted from this project myself. I have asked two friends to user test my application for me.

## Automated Testing

### User Interface Testing

Because my application is GUI based with no room for the user to break it as there are fixed boxes for them to select from as well as integrated ways of stopping certain inputs. Because of this I have only really tested the User Interface Testing.

To complete my user interface testing (UIT) I created some test cases with the desired output, I would record what the actual output was as well as record whether the test passes or fails. Below is a table with all of these tests and their results.

|  |  |  |  |
| --- | --- | --- | --- |
| What is being tested | What is the desired response | What was the actual response | Pass? |
| Selecting large numbers | The user selects the quantity, and the game begins with the correct amount | The user successfully selects the quantity of large numbers and the game begins with that amount | Yes |
| Entering a number | The user will select their first number to start an equation | The user successfully selects their first number and it is entered into the equation box | Yes |
| Entering an operator | The user will select their operator to perform for their equation | The user successfully selects the operator for the equation | Yes |
| Removing the last item added to the equation box | The user clicks the remove button, the last value is removed | The user successfully removes the last entered value from the equation box | Yes |
| The user can close the program | The user clicks the red cross in the top bar of the screen | The user successfully closes the program | Yes |
|  |  |  |  |
|  |  |  |  |

### Stress Testing

Although I have mainly tested through User Interface Testing, I have also decided to include this Stress Testing section. This is because my algorithm does not always work flawlessly, however I am able to see in the console exactly what it is doing.

My project is fairly difficult to stress test as there is no real way to push it further than it can normally go. The user will always be able to select how many large numbers they would like and they will always be the same four large numbers.

Occasionally the algorithm will fail to find a solution and instead of regenerating a new target number, it will just fail and the application will need to be restarted, or refreshed as it is a webpage. This is however, a very rare occurrence and I don’t understand why it happens, as it has only happened a few times in the many times I have run the application to test it.

## User Testing

I have asked two real users to test my application for me. The first one I asked was my girlfriend, Angharad Bache. I watched as she played the game and asked for her feedback on the application as a whole. She was able to click on buttons however she was unaware of how to remove the numbers from the equation box afterwards.

Her feedback was to make it more obvious that the button next to the equation box was to remove the last value entered, or to create a quick tutorial on what each section does before the game starts.

My second real user was Jamie Bonnet. Jamie found it very easy to navigate and found it very intuitive, however he did say that I could perhaps add more information on the first screen where it asks the user how many large numbers they would like. He said that “If the user is playing Countdown for the first time, they would not know what the large numbers are or of what use they will be”.

Overall there seems to be a connection, this is that I should include a tutorial page which could be launched upon a button click or just simply some text on the first screen. It would allow users to easily navigate through the game with a better idea of what does what.

# Critical Evaluation

## Were the requirements correctly identified?

I believe the requirements were correctly identified as well as met. I have designed an algorithm that will find the nearest possible solution to the target number using the user’s numbers. If the user’s numbers cannot make a number that is within ten of the target number, then a new target number is randomly generated.

Although the requirement has been identified and met, I could nevertheless have made a more complicated but more efficient algorithm such as A\* search. I was not able to implement A\* search due to my coding abilities. However, it would make the game faster and more reliable.

Not only does my code find the nearest possible solution to the target number, it will also find every other solution that can be made within ten of the target number, it will then rank them in order of how close they are. If the solution is an exact match, it will be placed at the top, whereas if it is ten away and there are better options, then it will be closer to the bottom.

I also made a fairly intuitive GUI for simple and easy gameplay. This also resizes the content depending on the size of the window that the user has chosen. This application is also a cross platform desktop application because of the framework I am using. This means that it works on Windows, Mac and Linux.

## Were the design decisions correct?

I think the design decision I made was a good one. Because I made the application using Electron, it allowed for me use Bootstrap to help design the application. Electron is a framework that allows the creation of desktop applications created in a web page. This is done by using Chromium as the front end of the application whilst Node.js does the backend heavy lifting.

This design helped me a lot because not only is it far more powerful to do it this way compared to the likes of Java Swing but I am also far more competent with CSS and JavaScript than I am with Java Swing.

Also, I was able to create a reactive design which allows the user to shrink or expand the application without sacrificing playability. If the user wanted a very small window, it would move the design around to fit to the size they chose.

## Could a more suitable set of tools have been chosen?

This is a difficult question as the answer is both yes and no. Electron is still a very young framework so I struggled with using it a lot at the beginning of the project. By the end of it I became fairly competent by designing a high end application that was reactive to the user changing the sizing as well as stopping the user from pressing certain buttons that would create an illegal move such as moving into negative numbers.

But on the other hand, Java and C++ are far more powerful languages that JavaScript meaning I could implement a better algorithm in general. They also have better memory management that JavaScript. This means that they would run into less issues.

The reason why I chose JavaScript in the end is because I knew that it could create this algorithm and work fine, but it would be able to make a much better looking and easier to navigate GUI.

## How well did the software meet the needs of those who were expecting to use it?

I tested the software on two different people. These people also range in their capabilities of technology as well as gender. Angharad Bache was my first user tester who was able to use the program with ease.

She thought it worked great and was impressed with the design as it was effective as well as attractive. Though, she did say it could use a tutorial which I can understand as if you have never played Countdown before, you would have no idea where to start.

Jamie Bonnett was my second tester, he is also a computer scientist so a program may be easier for him to use. He also thought the design was good and liked how I kept it in theme with the television series design of Countdown. Yet, he was not without his complaints. He too thought I could include more information on the home screen where the user is prompted to select the amount of large numbers. Although he knew how to work it, he gave me some constructive criticism and said that it could use even text based instructions at the bottom of the screen if not on a screen before the number selection screen.

I must admit I overlooked instructions for the game as I have had such a long history of playing Countdown since I was a small boy, also because I have been working on it for the past few months, it has become second nature to me.

## How well were any other project aims achieved?

My aim was to also create an attractive design. Although it was not specified in the project requirements to create a good looking GUI, I wanted to make sure that my application would look professional.

I spent quite a few days designing the GUI and went through two different designs entirely. The first was quite a bog standard design and would not look good at all, however it wouldn’t require many page updates or button swap states.

I then decided to design a much more attractive design which is the design I have ended up with. I watched a few episodes of Countdown before deciding upon the design. My issue with the first design is that I used a rather unattractive gradient in the background, whereas in the new design, I use a nicer subtle blue background as well as white boxes with black writing in it. It is inspired by Countdown and I hope it gives users a sense of nostalgia if they are as familiar with Countdown as I am.

## If I were starting again, what would I do differently?

If I were to start again there would be a few changes I would make. First of all, I would not have wasted so much time at the beginning creating Java code and then having to convert it to JavaScript later on due to the outdated Java Applets.

Similarly, I would also try to use the benefits of Electron more. The only real benefit of Electron that I have used it to make it into a desktop application that works on different operating systems as opposed to a web project.

Likewise, I would also spend more time researching and understanding A\* search. This is because A\* search would make my code much more efficient as well as more reliable. I would have liked to have implemented A\* search however with the looming deadline, I had to make the call to implement a much simpler algorithm.

Moreover, I would also have spent more time on my project so that I could also implement extra features at the end such as a timer and a clock to display it. This would add an extra level of difficulty to the game as well as making less monotonous. I could then add extra features such as being able to set the time limit you have to answer the question so that more confident people are able to set a lower time limit and people who maybe find mental math a bit more difficult, could set a time limit so that they can still enjoy the game.

I would also have liked to be able to implement some sound effects so that the user knows when they have clicked a button. When the user first selects their large numbers, the algorithm has to work to solve the target number. But because it takes a few seconds for the algorithm to finish, there is no way the user can know for certain that they have actually clicked the button.

Although I have included the base game in the file system, where the target number is guaranteed to be made as it is generated out of user numbers. I have not been able to make it so the user can select it to run that method instead of the algorithm. This is another feature I wish I have been able to solve before the end.

There are also special episodes of Countdown where instead of the large numbers being 25, 50, 75 and 100, they are replaced with 12, 37, 62 and 87[[3]](#footnote-3). This would allow for more difficult scenarios than large numbers that are set to increment every 25.

.

# Appendices

* 1. Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

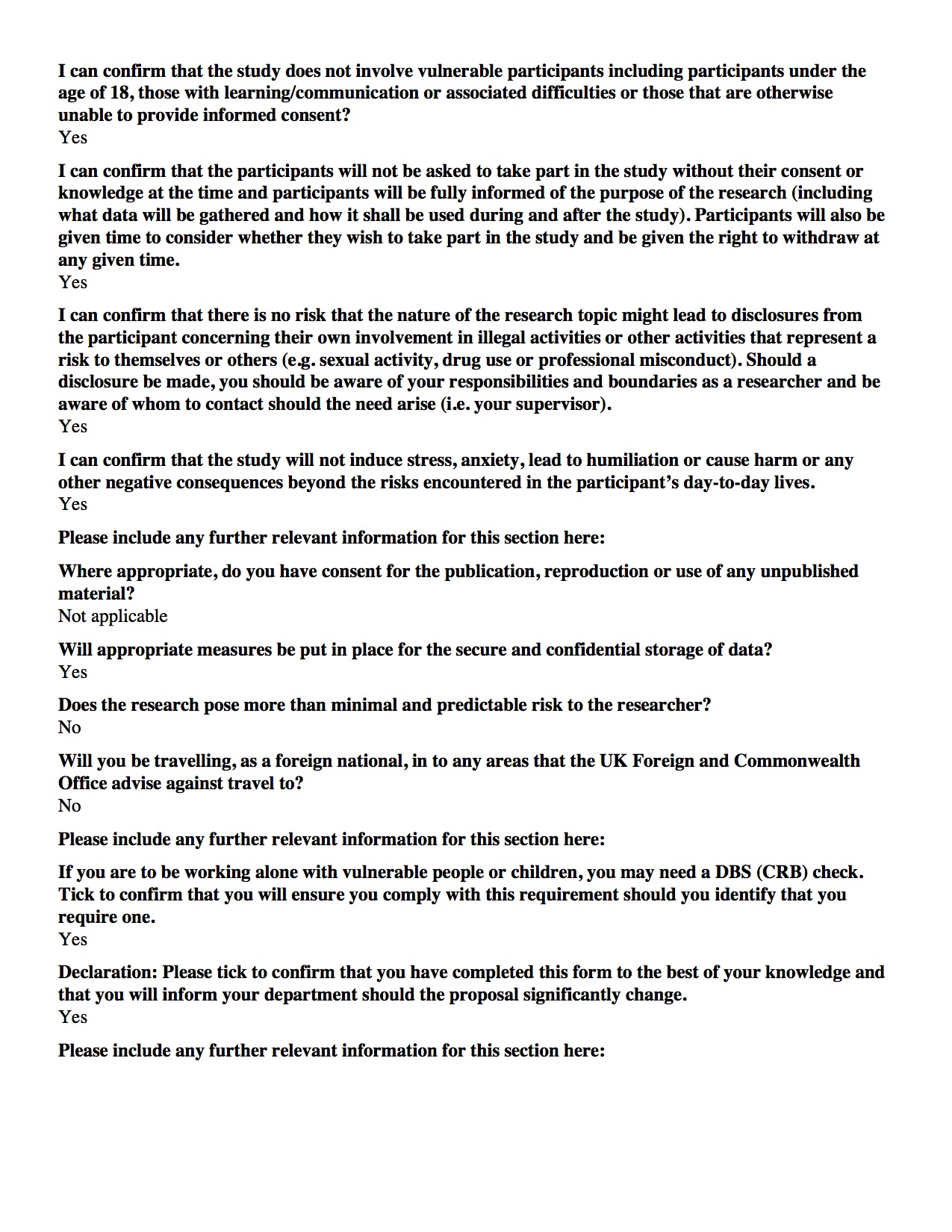
As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. If third party code or libraries are used, your work will build on that to produce notable new work. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

As an example, you might include a definition such as:

**Apache POI library** – The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client’s existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [5]. The library is released using the Apache License [6]. This library was used without modification.

* 1. ../Downloads/6822.pdfEthics Submission



Application reference number: 6822

* 1. Code Samples

The below code shows the expandOneLevel function within combination.js. The first part of the code is checking to see if we add the first and second value to the first value of the values array, whereas the second snippet of code is doing the same thing, but with the first and second value being added to value two.

Value One addition:

if ((this.val1 instanceof Combination) && this.val1.operator == this.operator) {

values.push(this.val1.val1, this.val1.val2);

} else {

values.push(this.val1);

}

Value Two addition:

if ((this.val2 instanceof Combination) && this.val2.operator == this.operator) {

values.push(this.val2.val1, this.val2.val2);

} else {

values.push(this.val2);

}

# Annotated Bibliography

This final section should list all relevant resources that you have consulted in researching your project. Each reference should also include a brief annotation.

1. Countdown - UKGAMESHOWS. <http://www.ukgameshows.com/ukgs/Countdown>, undated, Accessed May 8th 2017.

This website was invaluable in explaining the finer points of countdown. It conveyed to me some extra information about Countdown including special episodes and the history of where it all began.

1. The Full History Of Board Games – The Startup – Medium". *Medium*. N.p., 2017. Web. 8 May 2017.  
     
   This article took me in the midst of the history of boardgames. I ended up spending a great deal time delving into each section of the timeline and learning about Backgammon in 2000BC or The Landlord’s Game in 1903.
2. Wylie, Michael, and Damian Eadie. Countdown: Spreading The Word. 1st ed. London: Granada Media, 2001.

This book is the best companion to the show and proves to be an amazing insight into what the producers had in mind.

1. Neil Taylor, “MMP: Final Report and Technical Work”, 2017 (Online) Available at: <http://blackboard.aber.ac.uk/> Accessed 26th April 2017.

This template by our lecturer gave me the springboard to actually begin the write up instead of being confounded by the prospect of 10,000 words.

1. The Full History Of Board Games – The Startup – Medium". *Medium*. N.p., 2017. Web. 8 May 2017. [↑](#footnote-ref-1)
2. Wylie, Michael, and Damian Eadie. Countdown: Spreading The Word. 1st ed. London: Granada Media, 2001. Print. [↑](#footnote-ref-2)
3. 1. Countdown - UKGAMESHOWS. <http://www.ukgameshows.com/ukgs/Countdown>, undated, Accessed May 8th 2017.

   [↑](#footnote-ref-3)