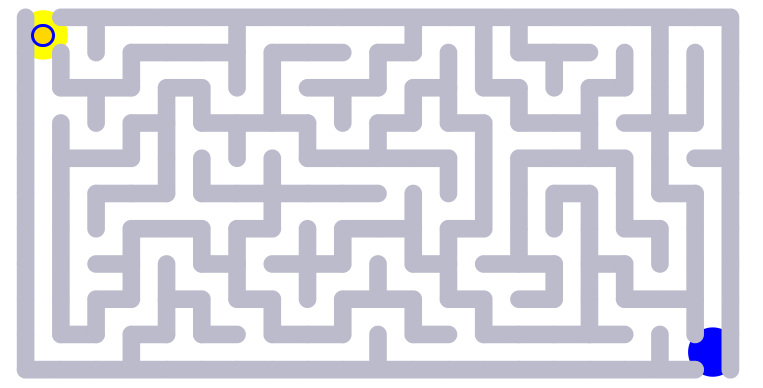
Origin of the project

*Maze games are a very popular form of computer games and there have been many successful ones such as The Amazing Maze Game which is an arcade game developed by Midway, released in 1976, Ali Baba and 40 Thieves which is an arcade game released by Sega in 1982, Aardvark which is a video game for the Commodore 64, and Commodore 16 published by Bug-Byte in 1986, Bomberman which is a windows game released by Hudson Soft in 2003 and many, many more.*

*A link to a full list of maze games can be found* [*here*](https://en.wikipedia.org/wiki/List_of_maze_video_games)*:*





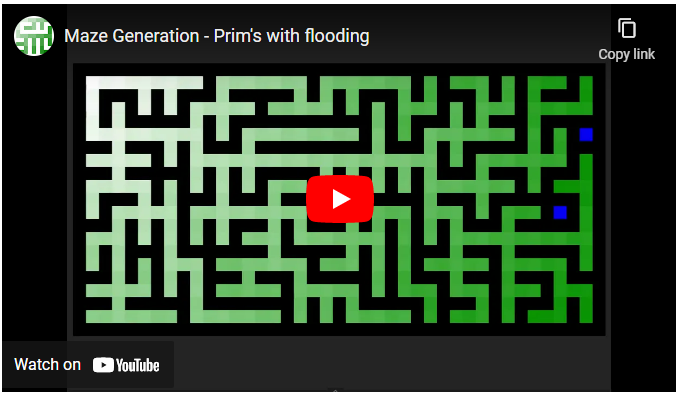
**Figure 1** - A maze game from [mathisfun.com](https://www.mathsisfun.com/games/mazes.html)

*I decided to put my own twist on the maze game and include A-level standard maths questions to grant a limited amount of moves depending on the difficulty of the question. I then had to choose topics for my questions and also the size of the mazes on each level etc. For this I asked my end-user who is a fellow A-level maths student Jacob. The interview can be seen below in the section labelled “End-user discussion” (page 10-11). I also needed to do background research on an algorithm to use for generating the maze with random paths that work (ensuring the user has at least one possible route from start to end of the maze). I ended up finding Prim’s algorithm for path finding and applying this to maze generation.*

Background Research

Prim’s algorithm was the algorithm I decided to use. Here is some more information on it.

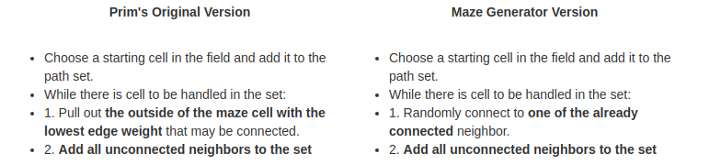
Here is a link to a video on it: <https://youtu.be/EcwGyx1gIF0>



*Prim’s Maze Generator is a randomised version of Prim’s algorithm which is a method that produces a minimal spanning tree from an undirected weighted graph.*

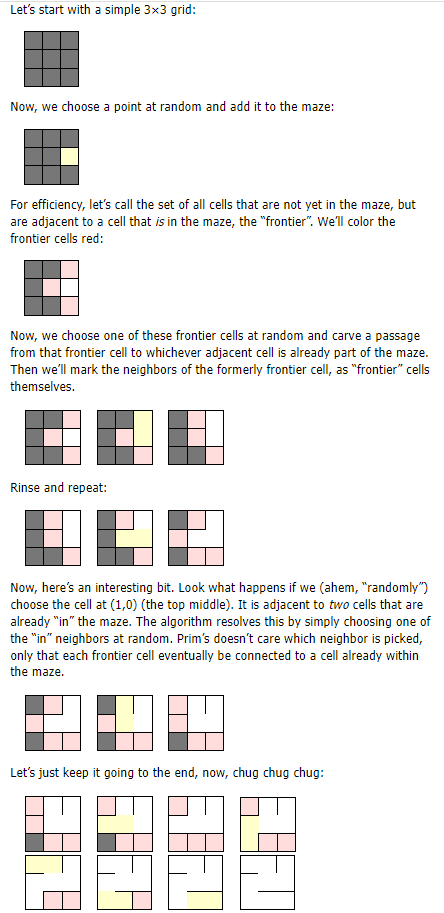
*Prim’s algorithm creates a tree by getting the adjacent cells and finding the best one to travel to next. Generating mazes using Prim’s algorithm uses a random cell to travel to the next one. It also takes up storage proportional to the size of the maze. Here are the steps below (****Figure 2****):*





Here is a demonstration of Prim’s algorithm on a 3x3 graph from:

<https://weblog.jamisbuck.org/2011/1/10/maze-generation-prim-s-algorithm.html>















Below is an image of the tree that Prim’s algorithm creates (**Figure 3**)

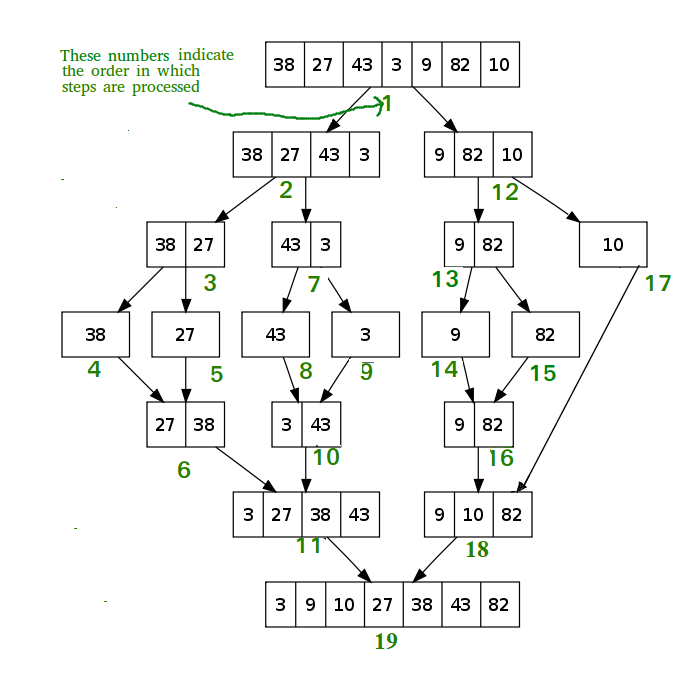




I also decided to use Merge sort to sort out the high scores in order.

*Merge sort is a linear sorting algorithm used to sort a list of integer values in descending or ascending order. It works by dividing the list until each item is in its own separate array. Then it combines items 2 at a time in order until the full array is merged together and it is in order.*

Below is an example of merge sort (**Figure 4**):



As you can see if the array is an odd number of items, the more items are taken into the first half of the array when it is divided into 2 arrays and it merges back in order.

Merge sort has its advantages. However there are disadvantages:

* Slower compared to the other sort algorithms for smaller tasks.
* The merge sort algorithm requires an additional memory space of O(n) for the temporary array.
* It goes through the whole process even if the array is sorted.

*A trace table of the merge sort algorithm below can be found in Appendix A*

Below is a pseudocode example of Merge sort which I have written myself:

array = [2,4,6,5,3,1,7]

SUBROUTINE sort(array):

IF array.length > 1 THEN

m ← (array.length+1) DIV 2

array1 ← array(0,m)

array2 ← array(m,m)

sort(array1)

sort(array2)

i ← 0

j ← 0

k ← 0

WHILE i < array1.length AND j < array2.length:

IF array1[i] <= array2[j] THEN

array[k] ← array1[i]

i ← i + 1

ELSE

array[k] ← array2[j]

j ← j + 1

ENDIF

k ← k + 1

ENDWHILE

WHILE i < array1.length:

array[k] ← array1[i]

i ← i + 1

k ← k + 1

ENDWHILE

WHILE j < array2.length:

array[k] ← array2[j]

j ← j + 1

k ← k + 1

ENDWHILE

RETURN array

ENDIF

ENDSUBROUTINE

OUTPUT sort(array)

Core Objectives

1. **An algorithm to generate the maze**
   1. This will be Prim’s algorithm for maze generation
   2. Other algorithms were considered but this was decided upon based on my coding skills and also its simplicity.
   3. It was also recommended by my end-user (interview on page 10-11).
2. **Maths questions that can be stored in an array**
   1. These will be based on the topics that the end-user gives (interview on page 10-11)
   2. These will be stored in a file then loaded into an array when the game runs
   3. This is an important aspect of the game as it adds creativity to the concept of the game and can also be used to improve a user’s maths skills
   4. It also makes the project unique as not many maze games incorporate the element of having maths questions that the user can answer.
3. **Ability to move character around using keyboard**
   1. This is a key idea as many of the best games use arrow keys to move around a character - without this the game wouldn’t be able to function
   2. Alternate key binds such as WASD could be added as well.
4. **Can select answers to maths questions**
   1. If the user wouldn’t be able to answer the maths questions then the game wouldn’t be able to function
   2. There will be a choice of 4 answers and each answer will have its own area on the screen that you can click.
5. **Interactive GUI**
   1. It is essential for the user to have a fun time while playing the game and a command-line interface wouldn’t suit this
   2. 2 possible graphic interface modules I could use are Pygame and Tkinter
   3. I will be using a combination of these 2 as it is easier to incorporate buttons on Tkinter but it is easier to generate a maze with a character that can traverse through it in Pygame.
6. **Scoring system**
   1. How this will be decided is a big decision in how the game will keep score
   2. Merge sort algorithm will be run on the array to sort out the high scores in descending order
   3. Score will be added based on questions answered correctly and also a bonus will be added on at the end based on how many moves were made - more points for less moves.
7. **File handling to store high scores**
   1. A file will be used to store the high scores of players who have beaten the game before
   2. The file will be opened in append mode and your score added to the document when you finish a level/beat the game
   3. Dictionaries will also be used to store the high scores in run time with them (dictionaries) allowing for an ability to sort the data.
8. **Limited number of moves**
   1. You will start of with up to 3 moves at the start of the game
   2. Moves will be used to move the player one block at a time
   3. The user will not be able to move if they have 0 moves left and a question will display thereafter for a chance to gain moves
   4. This is so the user cannot complete the maze without answering at least one maths question.
9. **Moves given after answering question correct**
   1. One move will be given if the question is not answered correctly and three moves will be given if the question is answered correctly.
10. **Set amount of lives**
    1. There will be a certain amount of lives based on the difficulty (of questions) selected by the user at the start of the game
    2. There will be 3 lives on easy, 5 on medium and 7 on hard
    3. Lives will be lost if the user crashes into a wall 5 times or answers a question wrong (as decided by end-user - interview on page 10-11)
    4. The game will end when the user loses all lives.
11. **Hints given with correct solution**
    1. The user will have an option to buy a hint
    2. This will then show them part of the correct solution to the maze
    3. Recursive solving will be used to find the correct path and store it in an array.
12. **Ability to save and load games**
    1. This will be useful as the user will then be able to close the game and pick back up from where they left off.
13. **Able to select difficulty of maths questions**
    1. This will be a menu choice at the start of the game and allow the user to choose how hard they want the maths questions to be
    2. If the user chooses harder questions, then more lives will be given.
14. **Background**
    1. This is the least important of the core objectives
    2. There is most likely a chance that there will be a background
    3. Having a background is necessary to appeal to the aesthetics of the project
    4. If there are multiple levels, each level will have a different background
    5. However, whether the background is moving or not will depend on time constraints.

Advanced Objectives

1. **Aesthetics of maze and player**
   1. The aesthetics of the maze and player will not matter as much until the concept of the game, maze generation, scoring system etc are considered and completed.
   2. The images used for the project will need to be small and will be decided depending on the theme of the overall game (decided by the end-user - interview on page 10-11).
2. **Background image**
   1. This will also depend on the theme of the overall game.
3. **Multiple levels increasing in difficulty**
   1. If there is time, multiple levels will be considered with the size of the maze increasing with difficulty of each level.
4. **Theme of levels**
   1. Each level could have its own slightly different theme.
5. **Personalised settings**
   1. These could be accessibility settings and other options which would allow the user to customise their game experience.
6. **Different game modes**
   1. There could be a time trial mode as well as a classic mode where instead of having a limited amount of moves, the user has a limited amount of time in which to complete the maze and answering questions correctly gives the user more time to complete the maze.
   2. This would be hard to implement though as “time.sleep()” would cause the whole program to stop.
   3. There could also be a mode that only focuses on answering questions (an arcade mode) and doesn’t give bonus points for completing the maze in a small amount of moves.

End User Discussion

For my end-user I selected a fellow A-level maths student called Jacob. He is quite good at maths and would be an excellent test subject for the maths side of things. I also selected my supervisor to be a fellow Computer Science student to assist me in making sure my game doesn’t have any game-breaking bugs and to help the functionality of the game be as smooth as possible. Below are the questions I asked Jacob about the program and the answers he gave.

**Q) What topics in A-level maths do you feel are the most important to appear in the game?**

A) I feel like logarithms, trigonometry and circles should come up in the hard section, graphs and transformations, differentiation and integration should come in the medium section. In the easy section should be binomial expansion, quadratics, vectors and algebra. There should also be a section for GCSE questions.

**Q) Do you prefer a game because of its graphics or the concept behind it?**

A) I believe that the concept of the game is more important as in a time constrained environment the graphics would not matter as much. However, the best games are the ones that have the best graphics but they are developed with large teams over long periods of time.

**Q) How big should the mazes be for each level?**

A) I think that the first maze should be 5 x 5, the second 7 x 7, the third 10 x 10, the fourth 12 x 12 and the last one 15 x 15.

**Q) Do you think Prim’s algorithm is a good algorithm to generate the paths of the maze?**

A) There are a few other algorithms you could have chosen such as Kruskal’s algorithm, Depth First Search and Recursive division. However I believe that Prim’s algorithm is the easiest to implement and generates a maze well enough.

**Q) What would you prefer to happen when you get a question correct e.g receive limited amount of moves, add time on to time limit etc?**

A) I would prefer for a limited amount of movies to be received when a question is answered correctly. Also 1 move could be received if a question is answered incorrectly. Also, in a different game mode a time limit could be given and time could be added on but because of time constraints it may not be possible to add this mode.

**Q) What should happen when you get a question wrong e.g lose a life, go back moves etc?**

A) I believe that a life should be lost when a question is answered wrongly and also if the user hits a wall say 5 times. A limited amount of lives should be given depending on the difficulty. For a different game mode, maybe the user could be sent back a move or 3 moves, but given the time constraints this mode may not be a possibility.

**Q) Should you be able to undo moves and if so, how many?**

A) I think that moves should be able to be undone. This would be a key aspect of the game in making it unique and also adding functionality and appeal to the game. I think a maximum of 5 previous moves should be stored.

**Q) What should be the overall theme of the game?**

A) I believe that the theme of the game should be pirates. There are a few good maths games out there that are pirate themed such as Pirate Treasure Numicon Shape Count so I believe that this would be a good unique theme to have.

I also used an alternative end-user. Mr Ninson is a maths teacher at my school who teaches A-level maths and has a good idea about which topics should appear in the game. Below are some questions I asked him and the answers he gave.

**Q) Which topics do you feel are the weakest among A-level and GCSE maths students?**

A) I believe that these following topics should be included in the game:

* *Making mathematical proofs*
* *Sketching gradient functions*
* *Solving problems involving areas between curves and lines*
* *Understanding implicit differentiation and using it to solve modelling problems*
* *Understanding the varying methods of integration*
* *Recurrence relations and modelling with series*
* *Vectors in 3D problems*

**Q) Would you recommend this game to students to help them with their revision?**

A) Any game that supports students learning in these topics is always a good and useful learning tool.

**Q) Do you think Prim’s algorithm is a good algorithm to generate the paths of the maze?**

A) I believe that is an efficient way of generating a maze and could be put to good use

*SUMMARY:*

So, overall from these interviews, I can say that my objectives are mostly unchanged as the end-users mostly agreed with my ideas (final objectives found in Appendix A).

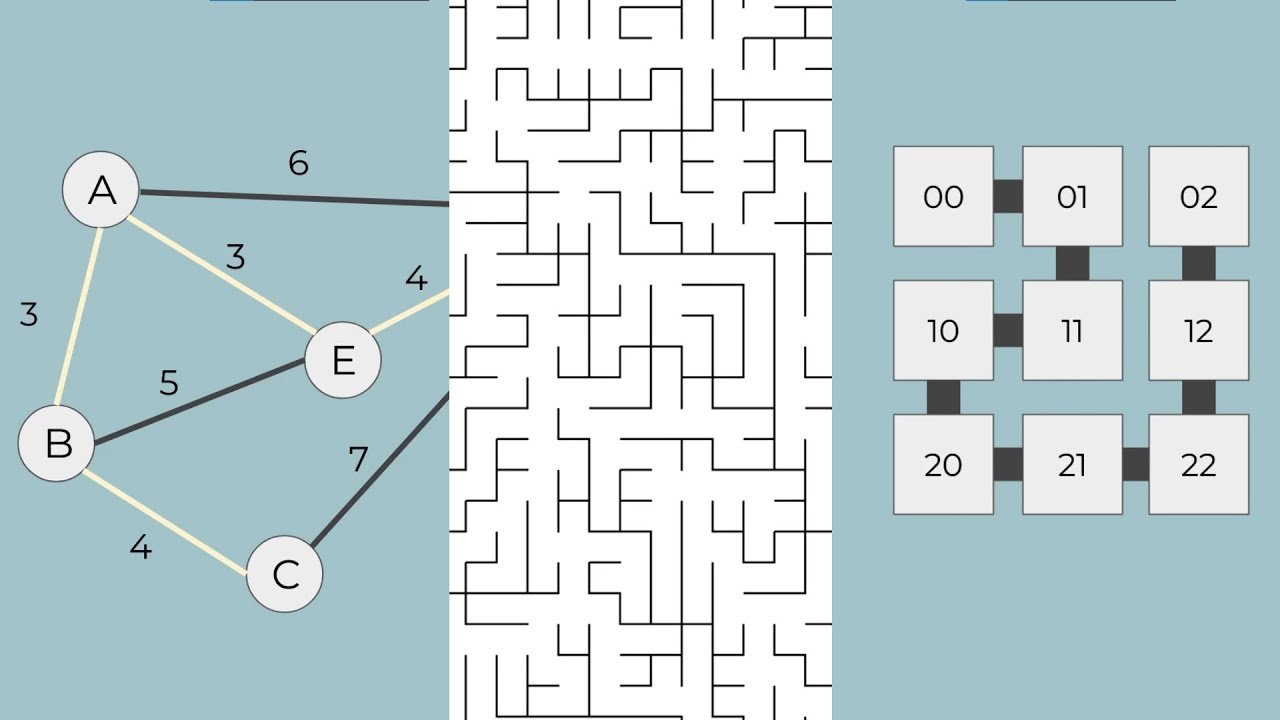
Some things that were decided by the end-users in the interviews include:

* There will be 5 categories for the questions (examples of questions found in Appendix A):

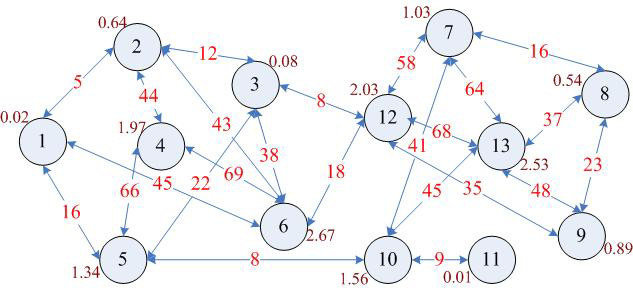
1. Easy which will include binomial expansion, quadratics, vectors and algebra
2. Medium will include graphs and transformations, differentiation and integration
3. Hard will include logarithms, trigonometry and circles
4. GCSE will include angles and right-angled triangles, graphs, sine and cosine rules, proportion, probability and more
5. Ninson’s specials will include mathematical proofs, problems involving areas between curves and lines, implicit differentiation, integration and vectors in 3D problems

* If there are multiple levels, the first maze will be 5 x 5, the second 7 x 7, the third 10 x 10, the fourth 12 x 12 and the last one 15 x 15
* Prim’s algorithm will be used as it was recommended by the end-users
* 3-5 moves will be received (depending on difficulty) when the user answers a question correctly and 1 move when the user answers a question incorrectly
* A life will be lost when the user answers a question incorrectly and when the user crashes into a wall
* The user will be able to ‘undo’ a maximum of 5 moves
* The overall theme (i.e. character and backgrounds) of the game will be pirates with different levels being set at different places in an imaginary pirate world

More on Prim’s Algorithm

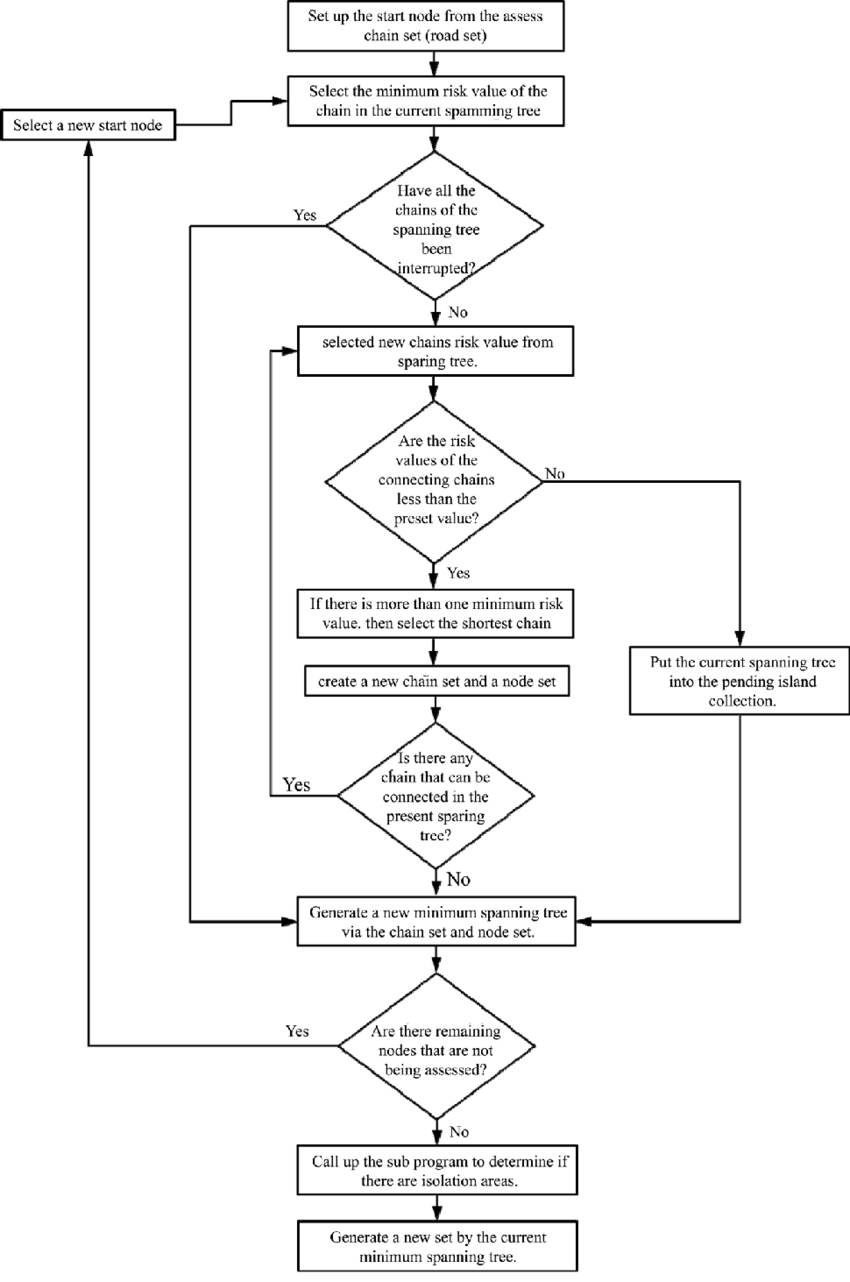


*Normally Prim’s algorithm is used to create a minimum spanning tree for path finding algorithms where a weighted graph is given and the shortest/least expensive path is the path wanting to be found. For example from this graph (****Figure 5****), the shortest path from 1 to any other node can be found using Prim’s algorithm (trace table for this in Appendix A).*



On the next page (**Figure 6**) can be seen a flow diagram of how Prim’s algorithm works.





Recursive solving algorithm

*Recursive solving is where a starting X and Y value are given. If the starting X and Y values aren’t a wall, then the method will call itself using adjacent X and Y values.*

*It will make sure that it hasn’t repeated any values and if it reaches the end X and Y values, then it will save the path in a correct path array. Below is a pseudo code representation of this algorithm that I have written myself (more info on this in Appendix A):*

maze ← []

wasHere ← []

correctPath ← []

startX ← 0

startY ← 0

height ← 10

width ← 10

endX ← width-1

endY ← height-1

FUNCTION solveMaze(maze):

Maze ← generateMaze(maze)

FOR row = 0 TO width

FOR col = 0 TO height

wasHere[row][col] ← FALSE

correctPath[row][col] ← FALSE

ENDFOR

ENDFOR

b ← recursiveSolve(row, col)

RETURN b

ENDFUNCTION

SUBROUTINE recursiveSolve(x,y):

IF x = endX and y = endY THEN

RETURN TRUE

ENDIF

IF maze[x][y] = wasHere[x][y] THEN

RETURN FALSE

ENDIF

wasHere[x][y] ← TRUE

IF x <> 0 THEN

IF (recursiveSolve(x-1, y)) THEN

correctPath[x][y] ← TRUE

RETURN TRUE

ENDIF

ENDIF

IF x <> width - 1 THEN

IF (recursiveSolve(x+1, y)) THEN

correctPath[x][y] ← TRUE

RETURN TRUE

ENDIF

ENDIF

IF y <> 0 THEN

IF (recursiveSolve(x, y-1)) THEN

correctPath[x][y] ← TRUE

RETURN TRUE

ENDIF

ENDIF

IF y <> height - 1 THEN

IF (recursiveSolve(x, y+1)) THEN

correctPath[x][y] ← TRUE

RETURN TRUE

ENDIF

ENDIF

RETURN FALSE

ENDSUBROUTINE

OUTPUT solveMaze(maze)

Similar games and how they work

For this section I decided to find a maths game in python and also a maze game in python and analyse them. The link to the maths game is below:

<https://www.w3resource.com/python-exercises/math/python-math-exercise-63.php>



*Line 1 imports the random library to be used later on in the program*

*Line 4 sets up the ‘display\_intro’ function*

*Line 5 sets the title*

*Lines 6-8 print the title with asterisks above and below it*

*Line 11 sets up the ‘display\_menu’ function*

*Line 12 sets the menu list*

*Lines 13-17 prints the menu list line by line*

*Line 20 sets up the ‘display\_separator’ function*

*Line 21 prints a ‘-’ 24 times*

*Line 24 sets up the ‘get\_user\_input’ function*

*Line 25 takes an integer input from the user and stores it as ‘user\_input’*

*Line 26 sets up a WHILE loop to check for invalid menu choice options*

*Line 27 and 28 print out ann error message and take another user input*

*Line 29 and 30 are executed if the while loop conditions aren’t met and returns the user input to the main program*

*Line 33 sets up the ‘get\_user\_solution’ function with a parameter of ‘problem’*

*Line 34 is a print statement*

*Line 35 prints ‘problem’ with an empty string at end*

*Line 36 takes ‘result’ as an integer input from the user*

*Line 37 returns ‘result’*



*Line 40 sets up the ‘check\_solution’ method with 3 parameters*

*Line 41 - 44 is a IF statement which checks if the user solution matches the actual solution and if it does, increments the score by 1, returns it and prints ‘Correct’*

*Line 45-47 is ELSE following the IF statement and returns count and prints ‘Incorrect’*

*Line 50 sets up the ‘menu\_option’ function with 2 parameters*

*Line 51 and 52 generate 2 random numbers between 1 & 21 and store them as number1 and number2*

*Line 53 sets up a series of IF statements for the ‘index number’*

*Line 54 sets the problem as a string of ‘number 1 + number2’*

*Line 55 gets the solution of ‘number1 + number2’*

*Line 56 calls the ‘get\_user\_solution’ function and stores the output as ‘user\_solution’*

*Line 57 calls the ‘check\_solution’ function and stores the output as ‘count’*

*Line 58 returns count to the main function of the program*

*Line 59 sets up the first ELIF statement for the ‘index number’ being 2*

*Line 60 sets the problem as a string of ‘number 1 - number2’*

*Line 61 gets the solution of ‘number1 - number2’*

*Line 62 calls the ‘get\_user\_solution’ function and stores the output as ‘user\_solution’*

*Line 63 calls the ‘check\_solution’ function and stores the output as ‘count’*

*Line 64 returns count to the main function of the program*

*Line 65 sets up the second ELIF statement for the ‘index number’ being 3*

*Line 66 sets the problem as a string of ‘number 1 \* number2’*

*Line 67 gets the solution of ‘number1 \* number2’*

*Line 68 calls the ‘get\_user\_solution’ function and stores the output as ‘user\_solution’*

*Line 69 calls the ‘check\_solution’ function and stores the output as ‘count’*

*Line 70 returns count to the main function of the program*

*Line 71 sets up the ELSE statement for the ‘index number’*

*Line 72 sets the problem as a string of ‘number 1 // number2’*

*Line 73 gets the solution of ‘number1 // number2’*

*Line 74 calls the ‘get\_user\_solution’ function and stores the output as ‘user\_solution’*

*Line 75 calls the ‘check\_solution’ function and stores the output as ‘count’*

*Line 76 returns count to the main function of the program*



*Line 79 sets up the ‘display\_result’ function with 2 parameters*

*Lines 80-82 is an IF statement for if the total is more than 0 and calculates ‘result’ as a percentage rounded to 2 decimal places*

*Lines 83-84 is an IF statement for if the total is 0 and sets ‘percentage’ to 0*

*Line 85 and 86 print the score and total*

*Line 89 sets up the ‘main’ function*

*Lines 90-92 call the ‘display\_intro’, ‘display\_menu’ and ‘display\_separator’ functions*

*Line 94 calls the ‘get\_user\_input’ function and stores the output as ‘option’*

*Lines 95-96 set ‘total’ and ‘correct’ to 0*

*Line 97 creates a WHILE loop when the option isn’t 5*

*Line 98 increments ‘total’ by 1*

*Line 99 calls the ‘menu\_option’ function and stores the output as ‘correct’*

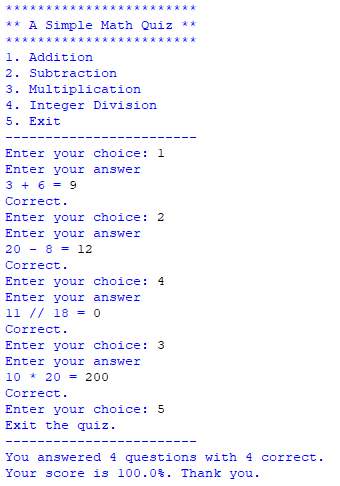
*Line 100 calls the ‘get\_user\_input’ function and stores the output as ‘option’*

*Line 102 is a print statement to signify the end of the quiz*

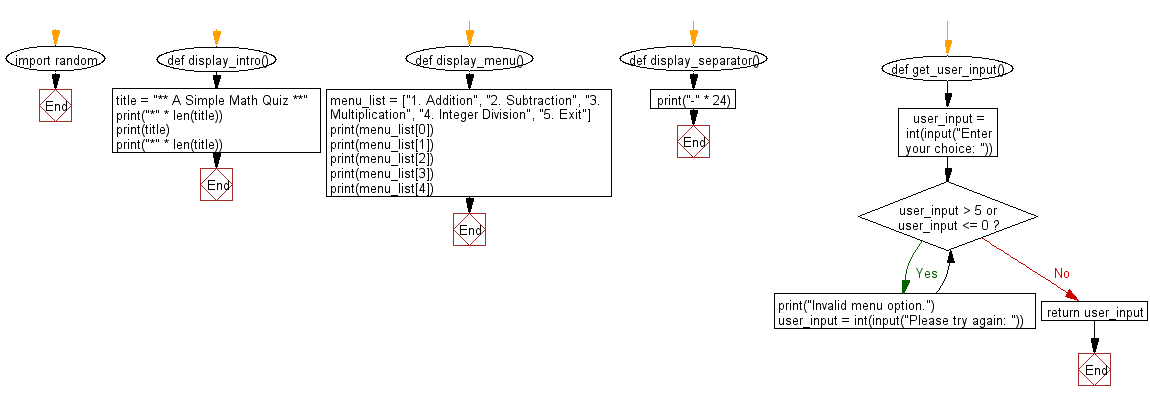
*Lines 103-104 call the ‘display\_separator’ and ‘display\_result’ functions*

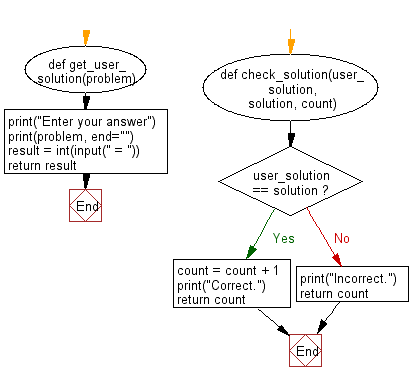
*Line 106 calls the ‘main’ function*

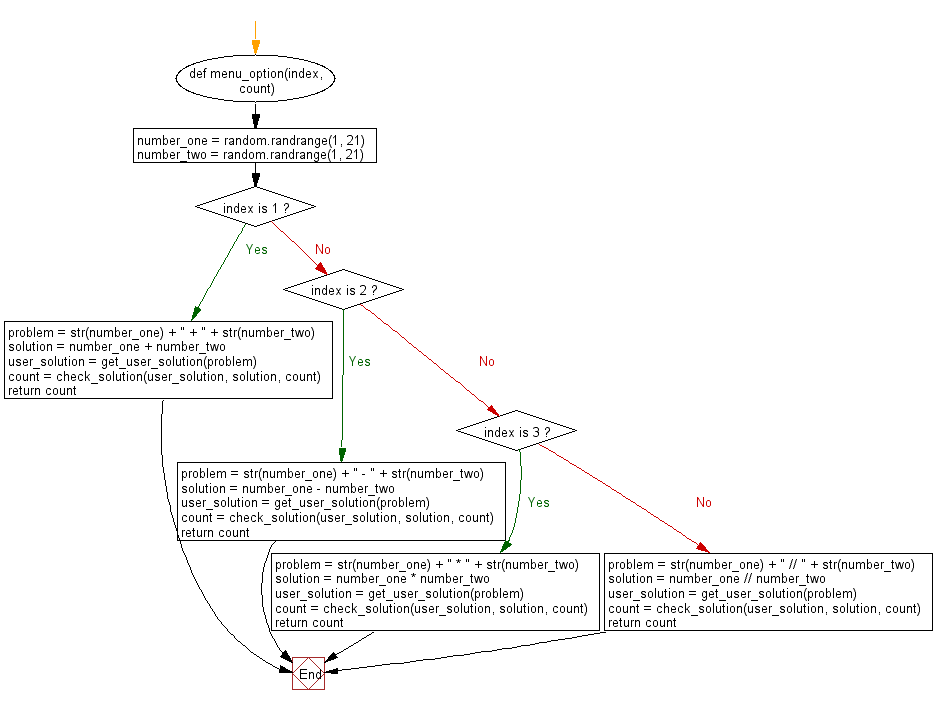
This is a screenshot of me playing this maths game.



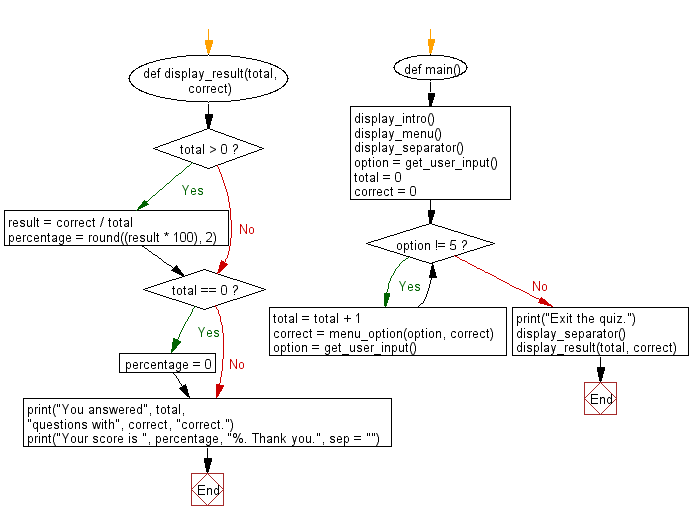
Below are some flow charts of the game (**Figure 7.1, 7.2, 7.3, 7.4**)





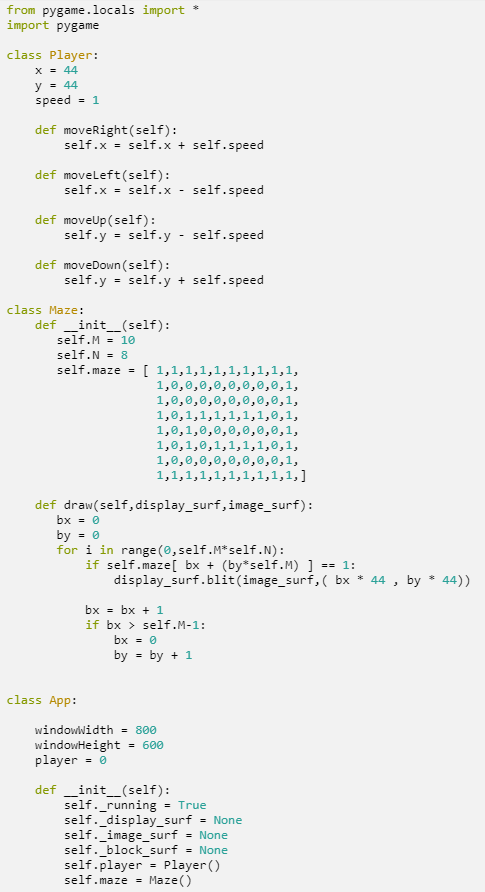






I also found a maze game in python. The link to this is below:

<https://pythonspot.com/maze-in-pygame/>



*Lines 1-2 import pygame and other necessary libraries*

*Line 4 sets up the ‘Player’ class*

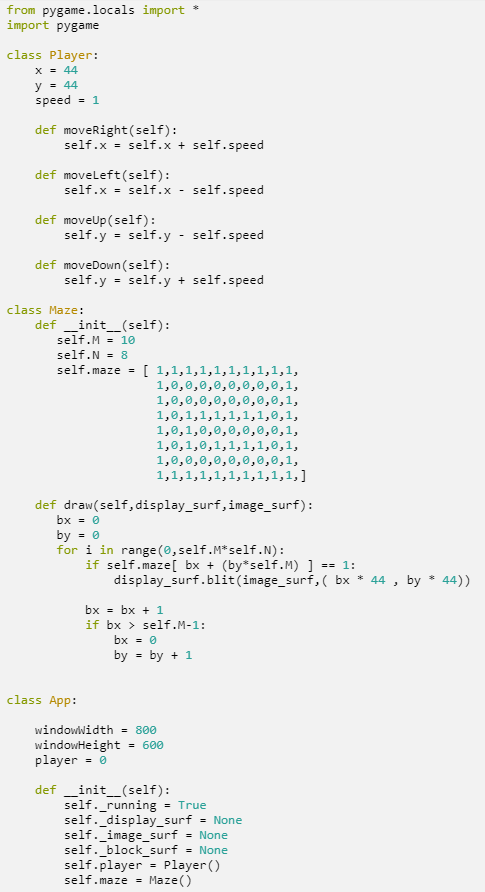
*Lines 5-7 set up the attributes of the class*

*Lines 9-10 sets up the ‘moveRight’ method which increments the ‘x’ attribute by the ‘speed’ attribute*

*Lines 12-13 sets up the ‘moveLeft’ method which decrements the ‘x’ attribute by the ‘speed’ attribute*

*Lines 15-16 sets up the ‘moveUp’ method which decrements the ‘y’ attribute by the ‘speed’ attribute*

*Lines 18-19 sets up the ‘moveDown’ method which increments the ‘y’ attribute by the ‘speed’ attribute*



*Line 21 sets up the ‘Maze’ class*

*Line 22 is the constructor method of the class*

*Lines 23-32 set up the attributes of the class including an array for the maze*

*Line 34 sets up the ‘draw’ method of the class with 2 parameters*

*Lines 35-36 set the variables bx and by to 0*

*Lines 37-45 traverse through the maze array and blit the walls of the maze if the index in maze is 1*

*Line 48 sets up the ‘App’ class*

*Line 50-52 set up the ‘windowWidth’, ‘windowHeight’ and ‘player’ variables*

*Line 54 is the constructor method*

*Lines 55-60 set up the attributes for the class*



*Line 62 sets up the ‘on\_init’ method of the class*

*Line 63 initiates pygame*

*Line 64 sets the window with dimensions as previously initialised variables and hardware accelerated in fullscreen mode (cannot be maximised/minimised) and overrides the attribute ‘display\_surf’*

*Line 66 sets the title of the pygame window*

*Line 67 sets the ‘running’ attribute to True which is used for the game loop*

*Lines 68-69 load the images for the blocks and player and store them as attributes*

*Line 71 sets up the ‘on\_event’ method of the class which takes 1 parameter*

*Lines 72-73 is an IF statement which checks if the user has pressed ‘QUIT’ and if it has been pressed then sets the ‘running’ attribute to False*

*Lines 75-76 sets up the ‘on\_loop’ method of the class and does a pass*

*Line 78 sets up the ‘on\_render’ method of the class*

*Line 79 fills the ‘display\_surf’ with black*

*Line 80 blits the player onto the ‘display\_surf’*

*Line 81 calls the ‘draw’ method of the ‘Maze’ class with 2 arguments*

*Line 82 updates the display*

*Lines 84-85 sets up the ‘on\_cleanup’ method which quits pygame*

*Line 87 sets up the ‘on\_execute’ method*

*Lines 88-89 uses an IF statement to check if ‘on\_init’ is False and if it is breaks the game loop by setting the ‘running’ attribute to False*

*Lines 91 - 102 is the game loop which allows the player to move when an arrow key is pressed in that direction or if the ‘ESC’ key is pressed sets ‘running’ to False*

*Lines 103 - 105 call the ‘on\_loop’, ‘on\_render’ and ‘on\_cleanup’ methods of the class*

*Line 107 is an IF statement that checks if ‘\_\_name\_\_’ is equal to ‘\_\_main\_\_’*

*Line 108 creates an instance of the ‘App’ class*

*Line 109 calls the ‘on\_execute’ method of the ‘App’ class*

Here is a screenshot of me playing the game:

