**Evaluation**

When developing my program, I’ve run into many challenges. For example, in the very beginning I didn’t know how to code in python at all! I had to teach myself from video tutorials on YouTube to be able to create a basic mockup for my program. Thankfully, over the many months I’ve improved my python skills and got better at coding.

Another challenge I’ve run into was the fact that I didn’t know how to implement certain algorithms. For example, I didn’t know I would implement recursive backtracker or the A\* algorithm. Thankfully, after doing research I came across several solutions on the internet that I’ve highlighted in my **3.** **Research**. I was able to understand their basic idea, and understood the logic behind these algorithms to be able to implement them myself, whilst making some tweaks that specifically suited my program.

In terms of success criteria, I had achieved every single one of my initial objectives with the program. All 33 success criteria have been fully met, however that doesn’t mean that there’s no room for improvement! I have some ideas that I could improve my program if I had been given extra time and wanted to implement new features.

**1. Efficiency**

In my program, I’ve used the A\* algorithm to path-find mazes. The A\* algorithm is the best for path-finding mazes, however that largely depends on the heuristic. The fewer nodes it has to process, the faster it will solve the maze. In honesty, I haven’t implemented an A\* algorithm that is necessarily efficient. My algorithm just “works” and does the job. If I had spent some time optimising and fine-tuning it, then I could possibly make it more efficient and solve mazes in less time.

**2. 3d Mazes**

My program also generates 2D mazes from 15x15 to 75x75 cells, so the mazes can have anywhere between 225 up to 5,625 total cells. However, I think it would be significantly more interesting for the mazes generated to be 3D instead. What I mean by this, is that instead of cells being represented as mere squares, they would instead be ‘blocks’, for example I could have a 75x75x75 3D maze that has 421,875 pathways inside of it. Possibly I’d have to use a completely different maze-generation algorithm rather than a backtracker algorithm. For example, [Kruskal's algorithm](https://en.wikipedia.org/wiki/Kruskal%27s_algorithm). Also, as an alternative to a 3D cube maze, I could have a [triangle maze](https://www.youtube.com/watch?v=_h65x15ULXE). So this would make my program significantly more resource-taxing on the computer, but it would be more unique than other programs that also generate mazes.

**3. Accessibility**

In my maze program, I could also add accessibility settings in the future, for example voice navigation. Some users would be blind or won’t be able to see, so I could implement a feature that reads text out-loud to them. Sadly though, mazes would be extremely difficult to represent with voice. That’s what brings me to my next point.

**4. 3d-printable mazes**

While this wouldn’t make my mazes actually 3D, it would allow them to be printed and accessible to blind users. As an example, when a user saves a maze they could also have an option to “export” it and print it if they have a 3D printer available. This would print the maze’s walls as actual walls that can be felt with fingers, and empty cells would have nothing in them. For the “solution” to the maze, which is normally highlighted in blue, the maze could use a different texture to help blind users distinguish between a cell that is a part of the solution, and an empty cell.

**5. Code maintainability**

I’ve come to realise that my code could be referred to as ‘spaghetti code’, while it might not seem like a problem at this point, in the future when I want to add extra functionality to my program, it could be difficult to implement some changes because, for example I only slightly utilised object-oriented programming (OOP). OOP’s main principle is that a program is broken down into modules that interact with other modules, but are mostly in isolation. In my program, I only have 3 classes, and ideally I’d have more classes, maybe 10-15 that could each be separately worked on. If I wanted to change a core feature of the program right now, I could possibly introduce a bug that would be very difficult to fix, and I’d have to rewrite a lot of code. If I had utilised OOP better, it’d be way easier to add new features to the program by just changing a single module.