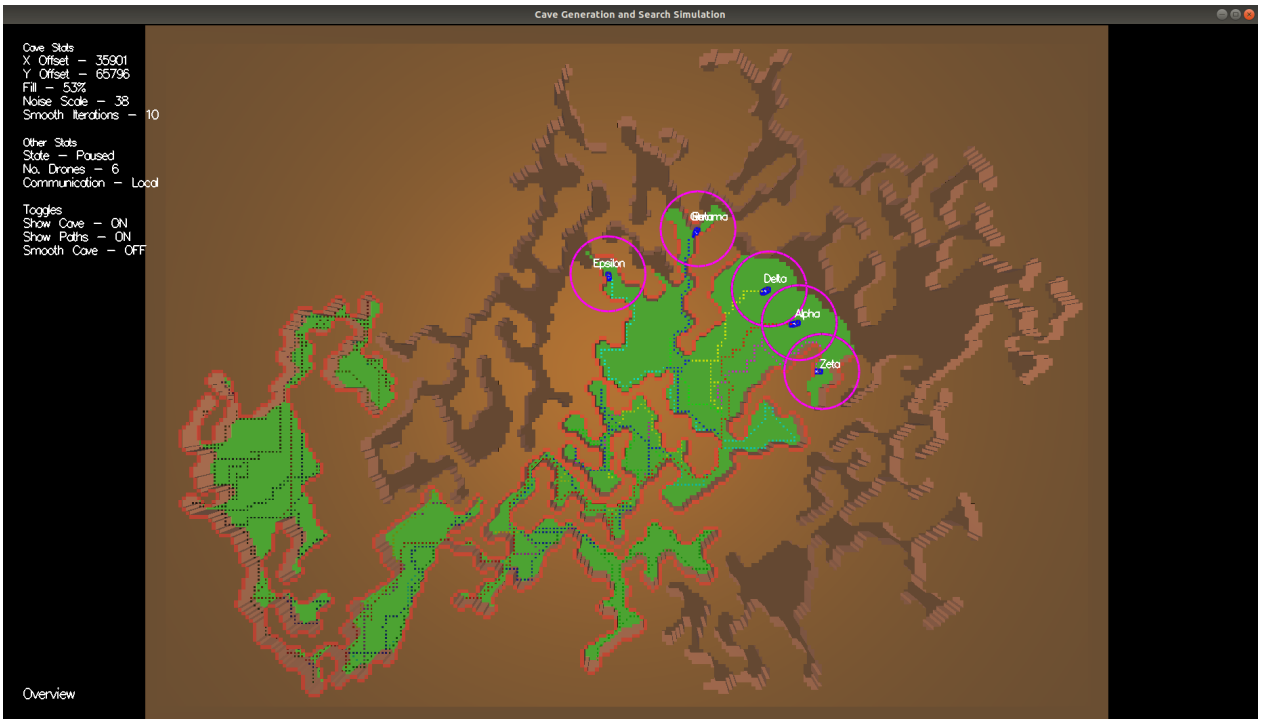
Using Swarm AI to efficiently map a cave network.

Kyle Gough

April 2019

Author Email: K.Gough@warwick.ac.uk
Supervisor: Dr Yulia Timofeeva
Year of Study: 3rd Year



Contents

1 Abstract

This project aims to find an efficient solution to cave exploration and mapping by simulating multiple autonomous flying drones, all initially unaware of their environment in a visualised and custom generated cave environment. The project tackles various problems in the field of robotics such as: sensing, frontier searching, teamwork, route planning and target selection. Drones will build an occupancy grid of its environment then proceed to map the cave by repeatedly identifying and searching an optimal frontier cell which are boundaries between unoccupied areas and unknown areas. Incorporating multiple drones into the simulation allows for communication of information to propel search efficiency and thus reduce search duplication achieved by mimicking swarm behaviour. Such behaviour includes exploring frontiers away from nearby drones, moving away from nearby drones, and branching out at junctions. Realistic caves for the simulation are generated using a random but reproducible process combining Simplex Noise, Cellular Automata and Flood fill. The project aims to identify how various factors of the drone's behaviour such as how many drones exist or their search radius may affect the performance of the search.

2 Keywords

- Artificial Intelligence
- Robotics
- Swarm
- Cave
- Searching
- Mapping
- Teamwork

3 Acknowledgements

The project files *SimplexNoise.cpp* and *SimplexNoise.h* are solely the work of user SRombauts on GitHub[5], taken from a repository[17] and used in this project to generate 2D simplex noise for cave generation. The repository is licensed under an MIT license[18] allowing copying, publishing and modification of the original code. The project utilises the code in the repository but does not modify any existing code within.

4 Introduction and Background

Caves are indiscriminately located all over the world from mountainous regions to desert regions. They are vastly diverse and can range from the size of a small car to hundred of miles long such as the 405 mile long Mammoth cave[8] in Kentucky, United States. Caves are greatly important for tourism, ecology, archaeology and hobbyists. Presently many unexplored caverns and mine-shafts are dangerous for cave exploration and mapping due to various factors such as unpredictability, visibility, flooding, hypothermia, falling rocks, tight spaces and harmful substances such as contracting Histoplasmosis[12] from bat droppings. Additionally, mapping is very high-effort, time-consuming and also susceptible to human error if processed manually.

Over the last decade the general perception of drones have shifted from military weapon applications to commercial and personal recreational use. This is partly due to the falling prices of drones and the increasing availability as hobbyists can now purchase drones from retail stores for recreational use.[4].

Consequently, this project proposes a safer and risk-free method of cave exploration and mapping by using a team of UAVs, specifically quad-copter based drones for their portability and small size, ideal for cave environments. The proposed method would be fast - as the drones are not limited by human speed or capability, efficient - as multiple drones can communicate and reduce the search space, and would mitigate human danger - as the drones will work autonomously.

The drones will use Swarm Artificial Intelligence to avoid contact with each other and the cavern walls whilst simultaneously identifying potential unexplored paths and avoid crowding by splitting up wherever possible to increase the efficiency of the exploration. GPS cannot penetrate rocks in a cave and thus the idea of 'global' communication is not present in a cave exploration scenario. Communication can only be achieved through electromagnetic waves where two drones have direct line of sight and are close enough to each other. As this project has applications beyond the scope of cave exploration where GPS and other forms of communication are in fact available, the project will allow for such scenarios where GPS would be available (such as crop harvesting) and where GPS is not available (cave exploration).

This project aims to identify, simulate and visualise the specific behaviours required to increase the efficiency of cave exploration for instances with a single drone and instances with multiple drones. The simulation will model the 2D case with altitude not present, an extension of the project into 3D was considered but was eventually found to be unfeasible due to time constraints and the additional research and resources that would be required.

5 Motivations

5.1 Why would we want to explore caves?

There are various reasons why research into cave exploration would be useful. This includes: Locating fossils and other archaeological finds for research purposes; Cave rescue - helping identify the location of lost or trapped spelunkers or tourist; and planning safe routes for hobbyists of cave exploration. Additionally, teamwork based AI in drones has various other applications outside of cave exploration where similar techniques and methods found in this project could be applied to, such as[16]:

- Surveillance - e.g. identifying criminals in crowded areas via facial recognition mounted on flying drones.
- Law enforcement - e.g. searching potential buildings for dangerous activities.
- Search and rescue[11] - e.g. locating missing persons in difficult to search locations such as woods.
- Searching in large bodies of water - e.g. drones adapted for traversal and vision in water equipped with sonar.
- Crop Harvesting - yielding the maximum amount of crop in the minimum amount of time with a group of autonomous harvesting drones.
- Autonomous home vacuuming - cleaning all areas in a home requires an initial search and mapping. Very similar to 2D cave exploration as these drones only operate in two dimensions.
- Mass deliveries - e.g. distribution of items via multiple flying drones.

5.2 Why did I choose this project?

An inspiration for this project was fuelled by the Thailand Cave Rescue news story[3], which occurred on 23rd of June 2018 when a group of 12 boys and their football coach became trapped 3 km deep into the Tham Luang cave due to excessive flooding from rainfall. The news story and subsequent rescue attempts in the following fortnight quickly became international news. As part of the rescue attempt drones were utilised to attempt to locate the boys with thermal imaging. This international story posed questions about how this accident could have been mitigated or how the rescue attempts could have been more efficient, a useful case study if this were to occur in the future.

An additional motivation for this project was a scene from the 2012 film 'Prometheus' directed by Ridley Scott[13]. In the scene multiple futuristic drone-like orbs are released which autonomously explore a cave structure and relay the mapping information back to the crew. Whilst the film is partly a futuristic Sci-Fi, the technology showcased is not implausible from

what may be possible in the next few decades, particular with the trend of how drones are improving and becoming smaller in recent years.

6 Current Technology

Currently there exists some technology that possesses the capability to explore and map cave systems.

Firstly, the Zebedee[15] - a hand-held device which contains a laser scanner in constant rotational motion. A human operator holds the device whilst traversing an area and the signals received from the device can be used to create a map of the surrounding area. It allows the creation of a complex and detailed map of the area, however it is restricted by the speed and reachability of the human operator in the environment. Because it is hand held by a human operator the risks of cave exploration are still present and even more dangerous as both hands are not free. This tool only performs mapping and doesn't not search the cave autonomously and so search efficiency is fully dependant on the human operator.

Secondly, the Hovermap[14] - is an attachment to hover drones which uses LIDAR technology to map the entire surrounding environment. It is similar to Zebedee but isn't required to be held and is not restricted by human speed or reachability of the environment. As this requires no human operator it mitigates the risk factor significantly however the Hovermap is not yet capable of interacting with other similar drones to increase exploration efficiency.

In conclusion, currently there are no solutions which exploit multiple drones working together. This is where this projects attempts to bridge the gap between individual drones and a swarm of drones mapping an environment.

7 Project Management

7.1 Software Development Approach

An Agile software development approach was decided early on in the planning of the project. This was for various advantages such as:

- It allowed adaptation to the requirements and design easier, which became useful where predicted components of the project became too difficult or unfeasible.
- Allowed a working product at all stages of development so components which needed improvement or fixing could be identified early and returned to at a later date.
- Modularity allowed components to be worked on separately. Slow development of one component had a lower chance of slowing progress of other components. For example, the 'Customisable Generation' component of the project was expected to be completed by December 24th but was delayed as other components were determined to be of a higher priority. This however did not impede the progress of other components due to the chosen development approach.

Additionally, previous experiences and projects where an agile approach had been employed had been successful, and also more familiarity with the agile approach was possessed over other software development approaches such as Waterfall.

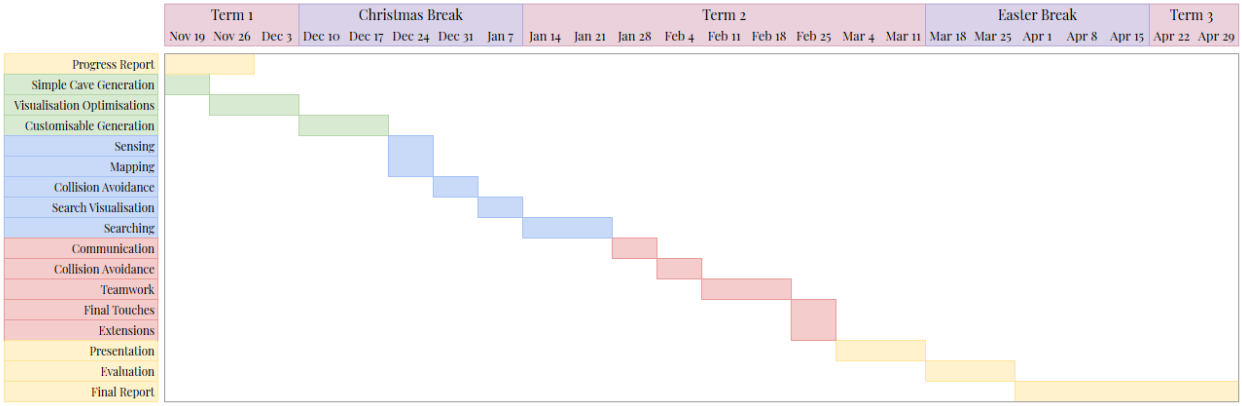


Figure 1: Project Gantt Chart

7.2 Project Components and Timetable

Three main stages required for completion of the project were identified. These were 'Cave Generation', 'Individual Searching' and 'Teamwork Searching' as coloured in the project Gantt chart in green, blue and red respectively.

Firstly 'Cave Generation' concerns creating realistic cave environments from noise to be used by the visual tool and simulation. Three sub-components were identified which start from creating some simple caves with little variation, then optimising the visualisation with OpenGL, and finally allowing more complex generation which is customisable by the user.

Secondly, 'Individual Searching' concerns a single drone and its operations including: Searching, Frontier identification, Frontier selection and navigation. Combining these operations will allow the drone to successfully explore and map the entire cave.

Finally, 'Teamwork Searching' concerns multiple drones exploring a cave. They employ swarm AI behaviour by branching out at junctions or when nearby and communicate mapping information to each other to increase search efficiency.

A component not individually tied to the above three but continuously implemented and refined was the visualisation tool. This tool allows the user to see the generated cave, each drone's movement, path, internal map and target in real-time. Sufficient controls were implemented for ease-of-use, tracking and toggling visuals, and search statistics were shown on screen to aid analysis of performance.

At the end of each sub-component in development, a small amount of time and resources were dedicated to testing that the sub-component worked as intended. If this was not the case then it was refined and refactored to fix the problems identified. This ensured problems did not cascade unintentionally to the end of the project development, and that there was a continual working product at the end of each stage of development.

7.3 Project Overview

The project is written in the C++ language on both 'Ubuntu' and 'Red Hat' Linux distributions. Atom[2] was used as the text editor for Linux. Visualisation was achieved using GLUT[9]: an OpenGL utility toolkit which contains libraries written in C++. This allowed the creation of a GUI to showcase the generated caves and drones exploring and mapping the caves.

Implementation using Unity with the C# language was originally considered but eventually discarded due to lack of experience and familiarity. Due to the time constraints of the projects a familiar language was chosen instead due to previous experience in projects using both C++ and OpenGL together.

Git[1] was used for source-code management and version controlling. GitHub[5] was used to allow the Git repository to be stored remotely allowing development to be completed and synced at home and at university, whilst also ensuring the project was backed-up at all times in the event of corruption or data loss.

7.4 Project Files

To improve modularity, debugging and readability of written code, similar functions and data were grouped together in individual files. The modularisation of code in the project provided

easier testing of individual components and provided a solid structure when refactoring. Additionally, it will help future maintainability of the code, where further improvement or extensions may be required.

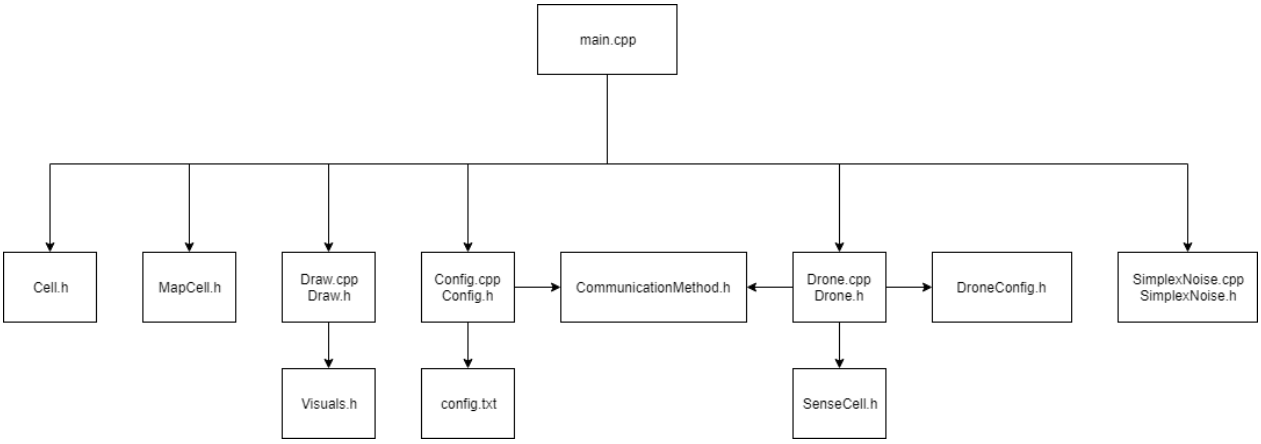


Figure 2: Project components file structure and hierarchy.

- *main.cpp* - Serves as the entry point of the program. Contains all methods for initialising the GUI environment, keyboard and mouse inputs and associated response actions, and cave generation. Also consolidates all code from other files together to create a finalised working product.
- *Cell.h* - A simple cell structure that can store an x and y coordinate. Used as the elementary object for the cave data structure.
- *MapCell.h* - Contains a single enum declaration for the available cell types which are: Free, Occupied, Unknown, Frontier. This means cells can be assigned named values instead of coded numbers, which also improves readability.
- *Draw.cpp, Draw.h* - Contains methods used to draw various objects onto the GUI such as polygons, circles and text. Also includes functions for displaying a drone, the path of a drone, a drone's internal map and the cave environment.
- *Visuals.h* - Contains definitions for materials and lights used by the GUI and colour variables for text and different components of the cave objects such as unique path colours for each drone.
- *Config.cpp, Config.h* - Reads the text file config.txt and parses the values set in the file. Attempts to assign the cave preset variables with the contents in the configuration text file, if unsuccessful default values are set instead.
- *config.txt* - Plain text file that allows a user to change the parameters for 5 different cave presets, the drone communication method, and search radius.

- *CommunicationMethod.h* - Contains a single enum declaration for the communication method to be used. Either Local or Global.
- *Drone.cpp, Drone.h* - Provides a structure for drone objects including data members for the internal map, list of frontier cells, current configuration, previous configurations, target, path to target and statistical values. Additionally, functions are included for sensing, communication, frontier identification, frontier selection and navigation.
- *SenseCell.h* - Another simple cell structure used during sensing. Stores the x and y coordinate of a cell and a range value corresponding to the distance between the cell coordinates and a drone's coordinates.
- *DroneConfig.h* - A structure that defines the configuration of a single drone at a single point in time. Values stored include a drone's x and y coordinates, orientation, and timestep when the configuration was recorded.
- *SimplexNoise.cpp, SimplexNoise.h* - Generates 2D Simplex Noise efficiently. These files have been taken from a GitHub repository[17] under an MIT license and the contents within is solely the work of user SRombauts.

8 Objectives

The analysis phase of this project identified numerous functional and non-functional requirements to outline and shape the end product. Most requirements were met, however some did require alterations or became dismissable due to infeasibilities. These alterations were simple, fast and non-detrimental due to the agile approach chosen. Here is a list of all the requirements set out in the progress report and how they have either been met or altered of the course of development.

8.1 Functional Requirements

1. *"To be able to create realistic cave environments with a cell resolution of 10cm by 10cm and maximum size of at least 250 m by 250 m (i.e. max cave dimensions of 2500x2500). The representation of the cave should have different values to represent different types of cells (e.g. unoccupied and occupied) to be used by the visualiser and for statistical analysis." (Core, High, Independent)*

   The implemented cave generation did produce realistic cave environments with cells either representing unoccupied or occupied areas. However, the proposed cave dimension requirement of 2500x2500 was not fully met due to complexity and performance. Many implemented algorithm's performance were based on the size of the cave such that a cave of this size caused performance bottlenecks primarily in the visualisation due to insufficient polygon optimisation techniques where too man vertices were used to represent the cave, but also in the flood fill and A* search algorithms.

2. *"Allow cave creation customisability for the following parameters: Maximum environment size in all dimensions, amount of open space, jaggedness of cavern walls, size of tunnel areas, length of cavern sections and variability/variety of cave." (Core, Medium, Dependant on 1)*

   After further research into random cave generation customisability for length of cavern sections and variety of cave were found to not be directly adjustable. The end result of cave generation however did provide a wide range of variety in the caves including jaggedness, open space, length of cavern sections, smoothness, area etc. The current cave generation doesn't allow you to directly specify certain criteria such as the length of cavern sections, however there are generation parameters which could influence the behaviour to best match the ideal criteria.

3. *"Ability to view the generated cave environment on a GUI." (Core, High, Dependant on 1)*

   This requirement was met by using OpenGL and was extended to include drone statistics, different camera views, visual toggles for the cave, drones, pathing and internal maps of each drone.

4. *"Drones should be able to move in all directions when given a request and possess the ability to search its local environment up to a specified range." (Core, High, Independent)*

   Pathing of the drones allowed them to move in all 8 directions when navigating to a designated target. A parameter *searchRange* editable in the configuration file allows specifying the distance at which occupied and unoccupied cells could be searched.

5. *"Drones should be able to detect and avoid collisions with the cavern walls." (Core, High, Dependant on 4)*

   Paths calculated using A* from the drone's current position to its target inherently avoids collisions with the cavern wall. Because localisation is assumed to be 100% accurate in the simulation, collisions do not have to be accounted for. However if extended to include noisy localisation or other errors, then the drone would perform localisation before sensing and path planning to avoid collisions.

6. *"Drone should be able to locally store known areas of the cave environment. The drone object will use a quad-tree data structure to store occupied cells." (Core, High, Dependant on 4)*

   Each drone is fully capable of storing their own known areas of the cave environment is what is known as the internal map. This stores occupied, unoccupied, frontier and unknown cells. A Quad-tree based implementation was initially chosen due to it being expandable at run-time, unaffected by sparse or dense regions and reduces the memory allocation at the start. Internal maps were initially implemented as Quad-trees but were re-implemented as a C++ vector of vectors due to complications with integrating Quad-trees with flood fill and the A* search algorithm where performance was greatly affected. One negative aspect of changing the implementation meant that unknown cells

would be stored resulting in an increased memory overhead, however the difference was small enough to not consider this as a problem. On the other hand a positive aspect of this meant due to its simplicity it shortened development time as it was trivial how to integrate the data structure with other algorithms and components.

7. *"The drone will be able to calculate frontier cells and use an algorithm to decide the optimal frontier cell to explore next." (Core, High, Dependant on 4,6)*

   Frontier cells are calculated using a simple iterative algorithm checking each unoccupied cell for any unknown neighbours. An appropriate algorithm is chosen to select the optimal frontier for an individual drone based on distance since last discovered, euclidean distance and bearing in comparison to other nearby drones. The frontier selected is not always the global optimal choice as the algorithm can be greedy towards the local optimum, but also adds some randomness to avoid multiple drones targeting the same frontier cell.

8. *"The drone should be able to track its path from its starting location, so it can be used when exploration is complete and shown visually." (Stretch, Low, Dependant on 4,6)*

   Each drone successfully stores its location at every timestep in a vector of data structures designed to store the configuration of a drone. The GUI has the capability to toggle showing each drone's path. The path is shown for each drone in a distinct colour going from low brightness to high brightness to show progress of the the drone over time.

9. *"In instances with multiple drones, collision avoidance techniques should be used to avoid drone-to-drone collisions." (Core, High, Dependant on 4)*

   Drone to drone collision avoidance was not implemented for the 2D case as it can be assumed they would avoid collision in the 3rd dimension (i.e. altitude).

10. *"When multiple drones are nearby they will communicate mapping information to avoid search repetition, increasing efficiency. Both drones will use the received communication to update their locally stored map." (Core, High, Dependant on 4)*

    Two different communication techniques have been implemented corresponding to different real world situations. Firstly local communication operates between drones within a set distance apart where both have line of sight of the other, and secondly global communication where all drones can communicate regardless of distance or line of sight. The communication operation successfully merges two drones' internal maps and results in an increase in search efficiency in both cases.

11. *"Display search statistics in real-time and at simulation completion such as: Total time taken to explore the cave, percentage/area explored by each individual drone and ratio of explored areas to unexplored areas mapped over time." (Stretch, Low, Dependant on 3, 4, 8)*

    Multiple statistics were able to be tracked and displayed. Total time taken to explore the cave was replaced with total distance traversed as this value was more reliable for

analysis. Additional statistics displayed include: unoccupied cells discovered, occupied cells discovered, unoccupied cells discovered from communication, and occupied cells discovered from communication.

12. *"Extend the project to be able to create environments and simulate drone behaviour in 3D." (Stretch, Medium, Dependant on all previous objectives)*

    This stretch goal was originally an optimistic goal to set and indeed became unfeasible due to the time constraints and additional work required to shift many algorithms and data structures from 2D into 3D. Additionally, many algorithms that required modification for 3D would gain an increase in time complexity which would slow down performance of the simulation.

13. *"Assume the drones cannot localise themselves with 100% accuracy and so use localisation techniques with the sensed data to localise themselves in the environment accurately." (Stretch, Low, Dependant on 4, 6, 8)*

    This stretch goal was not implemented due to how previous components had been implemented which were not accommodating for localisation, as well as time constraints. If this requirement had been implemented some method incorporating 'Simultaneous Localization and Mapping (SLAM)'[7] which both maps an unknown area and keeps track of an agent inside the area simultaneously would have been used. Localisation is important when other factors beyond a drone's control causes unpredictable movement such as wind or animal interactions, or interpreting noisy signals received from sensors.

8.2 Non-Functional Requirements

1. *"To be able to create realistic cave environments in less than 10 seconds."*

    This requirement was significantly exceeded as random caves can be generated in well under a second. This is due to the low complexity and efficiency of the cave generation algorithms which were implemented. For larger caves generation is noticeably slower but still well within the 10 second time constraint.

2. *"The searching simulation should run close to real-time speed."*

    Simulations using a single drone perform at real-time speed with no lag. With the increase of more drones there is more processing to be performed including sensing, A*, communication etc, and with this arises lag. Significant lag occurs when a drone is calculating a path to a target cell that is far away as the A* search algorithm must search a large proportion of the cave. Hence the A* search algorithm is the bottleneck of performance for multiple drones and the main concern for improvement. Despite this, the simulation runs mostly smooth with some lag spikes when the A* search algorithm calculates a complex path. As the end result of the simulation is most important the lag spikes can be ignored as the simulation still completes in a feasible time.

3. *"To be able to view informative data on the environment and drones within the GUI."*

Statistical information for each drone is displayed at the bottom of the GUI and updated in real-time. Cave generation parameters and visual toggles are displayed on the top-left side of the GUI.

4. *"UI should be responsive, usable and aesthetically pleasing."*

   On start-up a manual page is shown listing all the keybinds needed to operate the the simulation. Cave cells are colour coded appropriately to help the user identify certain cells such as yellow cells for a drone's current target. Additionally there are dedicated keybinds for visual toggles which can be turned on or off, this includes:

   - Enable/Disable Smooth cells - smooths cave edges using the marching squares algorithm.
   - Show/Hide path - toggle to show or hide paths for each drone in a distinct colour.
   - Show/Hide cave - toggle to show or hide the original cave so a drone's internal map can be seen independently.
   - Show/Hide controls - toggle to show or hide the list of controls and keybinds.

9 Cave Generation

This section describes the process of creating random, realistic cave environments from initial continuous noise to be used by the visualisation tool and simulation. The generated cave is a simple 2D array of integers either 0 or 1. Where 0 represents free, unoccupied cells where a drone or human is free to operate in, whereas 1 represents cavern walls, occupied cells where a drone or human cannot operate in or traverse through. The goal of the 'Cave Generation' stage was to create realistic, reproducible and random 2D cave environments from an initial noise. This means given the same initial noise and generation parameters a given cave could be reproduced, allowing easier testing and analysis.

Whilst this component of the project is a precursor for cave searching and mapping, it has important applications in other fields such as level design in computer games[10] such as first-person shooters, rogue-likes or real-time strategy games. This is because having an unlimited resource of possible caves or environments generated automatically means an improvement on unpredictability and reduction in development time.

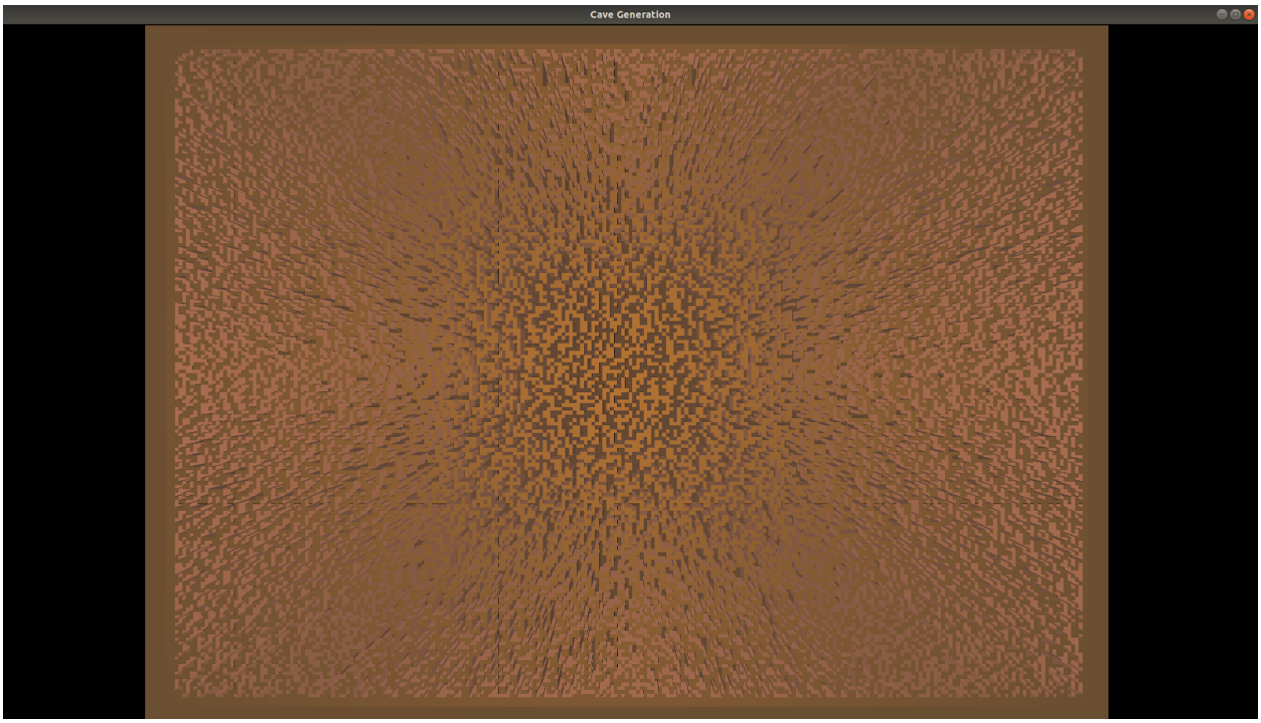


Figure 3: Initial Generated Noise that was used to create a cave environment.

The cave generation process has been organised into four distinct steps. When combined together this creates random, realistic, reproducible and fast cave environments.

1. Generating and thresholding 2D Simplex noise.
2. Smoothing the noise by using a Cellular Automata for a defined number of iterations.

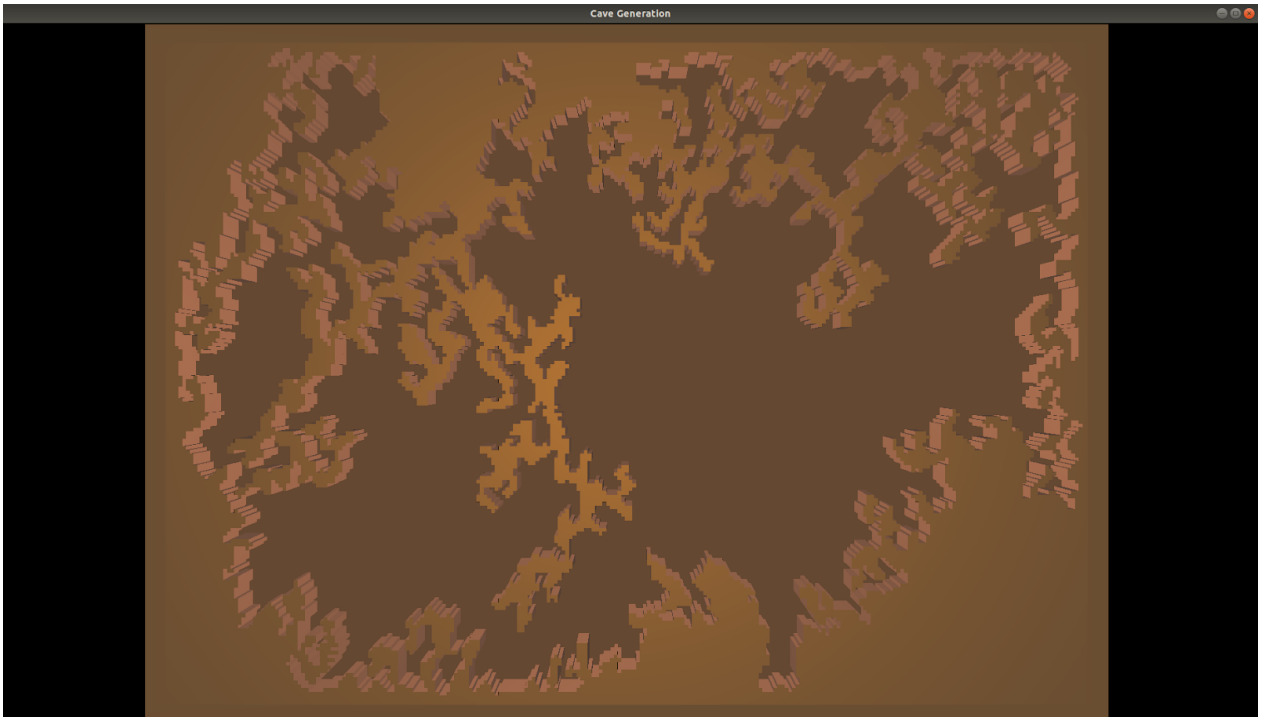


Figure 4: Produced realistic 2D cave environment from the above noise.

3. Removing inaccessible areas by using Flood fill by identifying the largest connected area the removing everything else.

4. Removing stand-alone areas by using a series of Flood fills.

9.1 Prototype

An initial naive prototype was developed. Starting from random discrete noise generated from a pseudo-random number generator, a cellular automata algorithm "smoothed" the cells to produce a more cave-looking environment. In the figure below cavern walls/occupied cells are shown as black whilst unoccupied/free cells are in white.

The cellular automata smoothed the cave for a total of i iterations. In each iteration every cells is looked at individually for the number of free cells in its local neighbourhood. In this implementation the local neighbourhood was defined as the 3x3 block centred on the cell not including the centre cell (8 cells in total). Each cell has a opportunity to change state if its neighbours have a bias towards occupied or unoccupied. If the number of unoccupied neighbours exceeds 4 then the cell changes to unoccupied, and if the number of occupied neighbours exceeds 4 then the cell changes to occupied, otherwise the state of the cell remains unchanged.

To avoid directional interference and bias when smoothing, an additional temporary data structure, *tempCave*, was created to store a copy of the original cave data structure, *currentCave*, so values could be changed in each iteration without affecting the previous iteration.

For the prototype 20 iterations of smoothing proved to produce the best-looking cave environments. The cellular automata performed at a time complexity of $O(wh)$ where w and h are the width and height of the cave respectively. Below is the algorithm for one smoothing iteration.

Listing 1: Cellular automata smoothing iteration.

```
1     for i = 0 to caveWidth
2       for j = 0 to caveHeight
3         n = number of free neighbours for currentCave(i,j)
4         if n > 4 then
5           tempCave(i,j) = Free
6         else if n < 4 then
7           tempCave(i,j) = Occupied
8         else
9           tempCave(i,j) = currentCave(i,j)
10    currentCave = tempCave
```

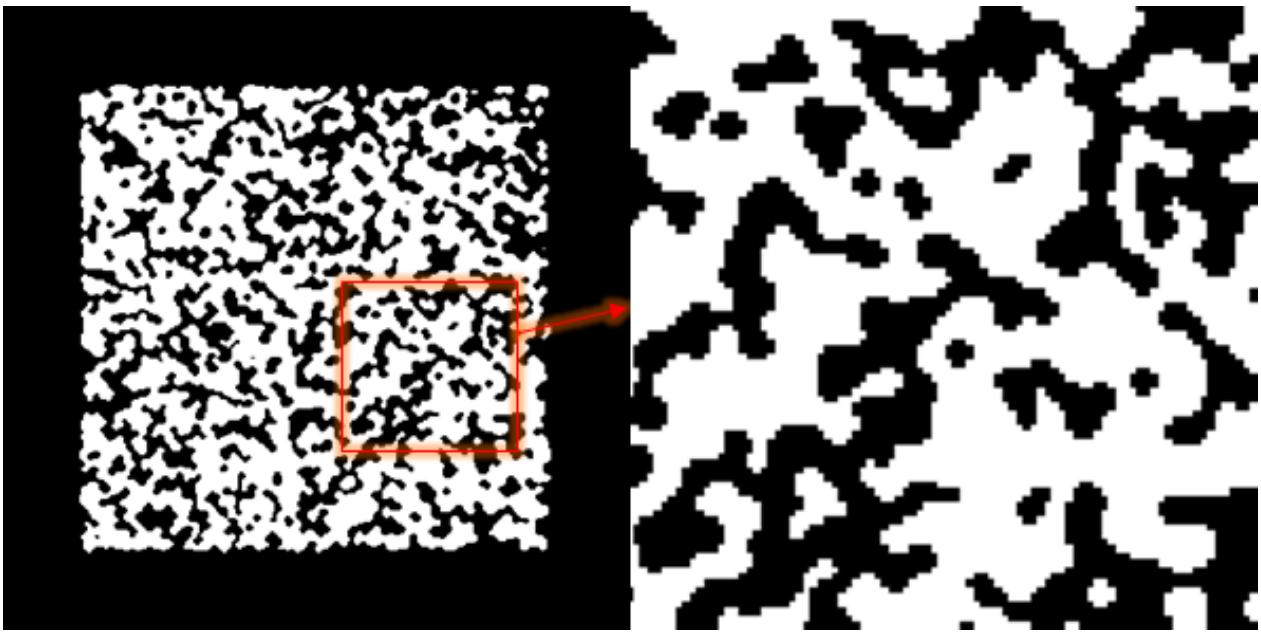


Figure 5: Cave generation prototype, with the highlighted region further zoomed in on the right.

9.2 Research

The prototype did produce cave-like environments, however the caves were not easily producible, featured low customisability and possessed little variety in terms of cave structure. Therefore, additional research was conducted on how to improve the variety and customisability of cave generation.

An important factor for variety and realism of the generated caves was found to be heavily dependant on the initial noise such as the ratio of occupied to free cells[10]. Thus two well

known and similar pseudo-random noise generation algorithms were identified: Perlin Noise and Simplex Noise. This would provide the required customisability and variety not present in the prototype.

To ensure as close to possible realism, examples of real world caves were viewed and compared with the generated caves. Many caves used for reference were taken from surveys[19] of many Yorkshire Dales caves. This provided an understanding of how some caves are formed and the general structure that the cave generator should aim for.

9.3 Simplex Noise

After creating the generation prototype, further research was conducted on possible improvements and a list of four main steps were constructed to create the final cave generator. The first step in improving cave generation was to use a different noise function. Previously noise was generated from a simple pseudo-number generator which produced caves with little variety. Simplex noise was the logical choice for improvement due to its fast speed, low computational cost and high variety. Simplex noise is a procedural noise function and an improvement on Perlin noise. In two dimensions it has low computational cost and allows a greater diversity in cave environments as the noise contains some structure.

An advantage of Simplex noise over random noise was that you can specify an offset value (O_x, O_y) for each dimension which acted as the generation seed, and a scale value, S, which affected the cave structure and scale of the cave. These values allowed the creation of random but reproducible caves. Values obtained from the simplex noise function were continuous in the range of 0-1, however the desired output was discrete data of either 0 or 1 for unoccupied and occupied cells respectively. To solve this, values were scaled to the range 0-100 then thresholded by a parameter, t. A low value of t produced a sparse cave with little cave walls, whilst a high value of t produced a dense cave.

An ideal range for t was identified at 40-60. For random cave generation a Gaussian distribution (μ=50,δ=5) was implemented to pick the value of t. Additionally, the scale parameter S was chosen from another Gaussian distribution (μ=55,δ=20), and finally the offset values O_x and O_y were chosen randomly from the range 0-100,000.

Listing 2: Obtaining and thresholding a simplex noise value for a single cell in the cave.

```
1     noiseValue =  SimplexNoise(i / caveWidth * S + Ox,
2                                j / caveHeight * S + Oy)
3     if noiseValue <= t
4       Cell (i,j) = Occupied
5     else
6       Cell (i,j) = Free
```

The function Simplex noise was imported from a preexisting implementation of Simplex noise[17] on GitHub released by user SRombauts under an MIT license. Attempting to replicate the Simplex noise function would have consumed considerable time and resources as it features complicated mathematics and was not the focus of this project.

9.3.1 Border Cells

To prevent drones attempting to navigate outside the bounds of the cave area, an artificial border of occupied cells was added to every cave. This consists of a 3-cell wide border along each side of the cave which prohibits changes in cells outside the border.

Figure 6: (Cave a) An example of a cave obtained from simplex noise.

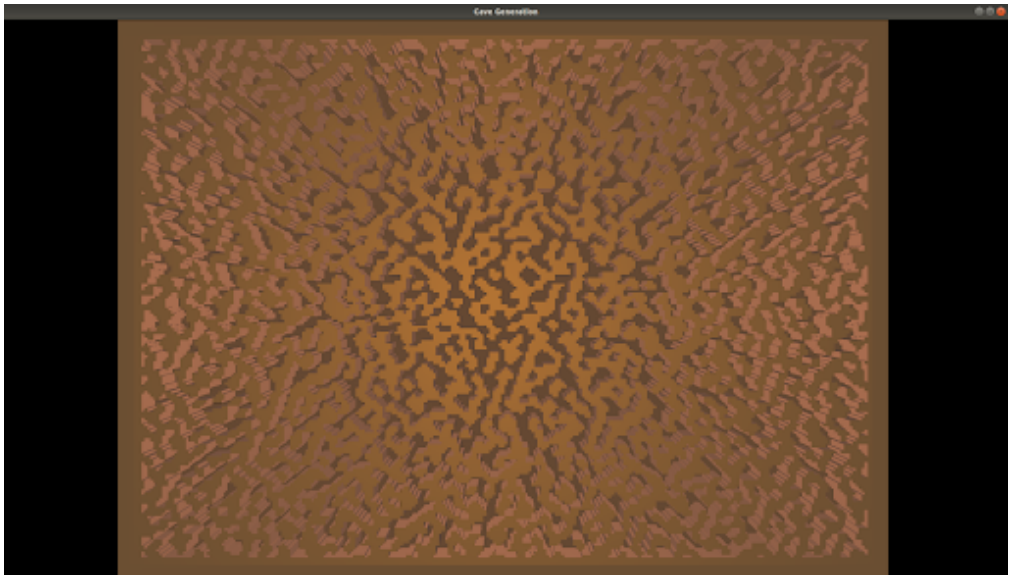


Figure 7: (Cave b) Another example of a cave obtained from simplex noise, this time with a higher scale value.

9.4 Cellular Automata

The second step involves applying i iterations of the cellular automata as described in the prototype section to the thresholded Simplex noise. A low value of i produced a jagged cave whilst a higher value of i produces a smoother cave. As i increases however fewer and fewer cells are changed as the cave reaches higher entropy. For random caves the value of i is chosen by a Gaussian distribution (μ=10,δ=5).

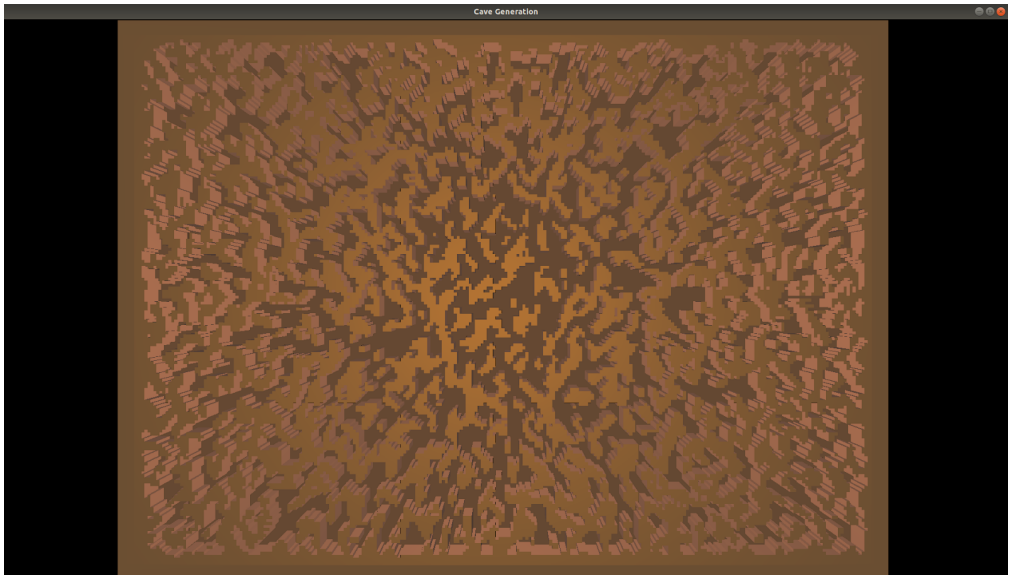


Figure 8: (Cave a) Smoothed by cellular automata for a low value of i.

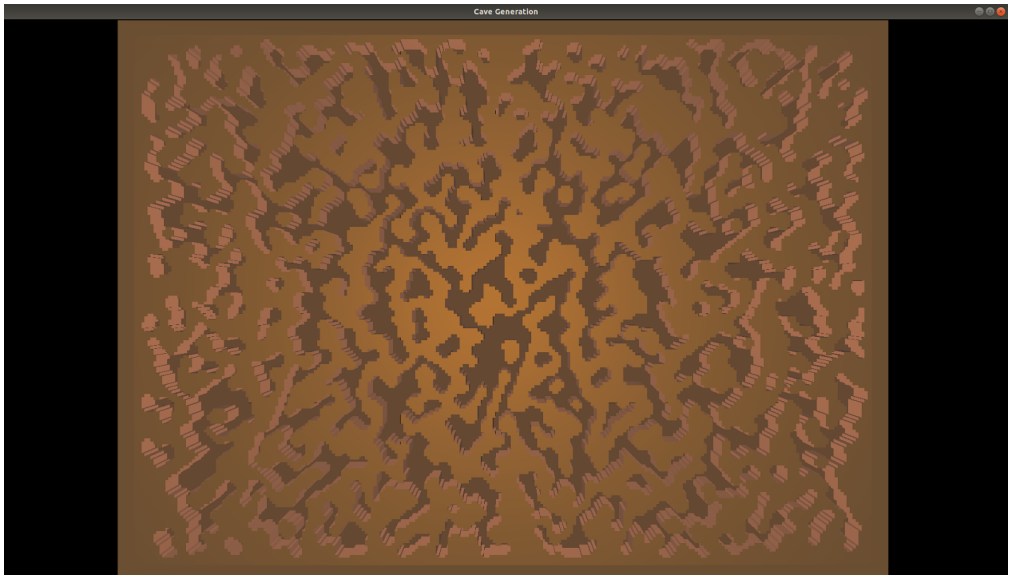


Figure 9: (Cave b) Smoothed by cellular automata for a high value of i.

9.5 Inaccessible Areas

The third step aims to identify the largest free connected area in the generated cave, and remove everything else. This is because we want the drone to search the largest connected area and remove other areas as they are redundant because the drone cannot reach them.

Firstly, a simple flood fill algorithm is used to find the largest free connected region. For each free cell in the cave a flood fill counts the number of cells in the region and simultaneously sets the cells to occupied. This is so the same region is not processed multiple times which saves a lot of computational power and time. The implemented flood fill is an iterative queue-based 8-directional variant. A stack-based implementation was not chosen because of the memory overhead required for large regions. Once the largest region has been established, all other cells are set to occupied thus removing other redundant regions.

Listing 3: Algorithm to find a cell that is contained within the largest connected free region.

```
1     max = 0
2     for i = 0 to caveWidth
3   for j = 0 to caveHeight
4     count = floodFillCount(i,j,Occupied)
5     if count > max
6         max = count
7         startCell = (i,j)
```

The function 'floodFillCount' counts the number of free cells in the region starting from cell (i,j), and in the process converts all free cells to occupied cells.

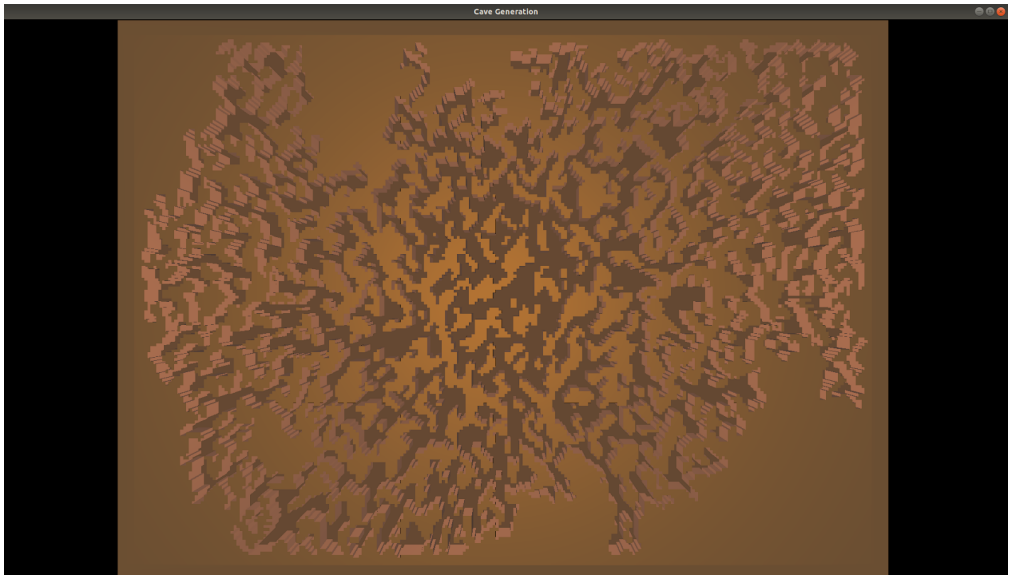


Figure 10: (Cave a) with inaccessible areas removed.

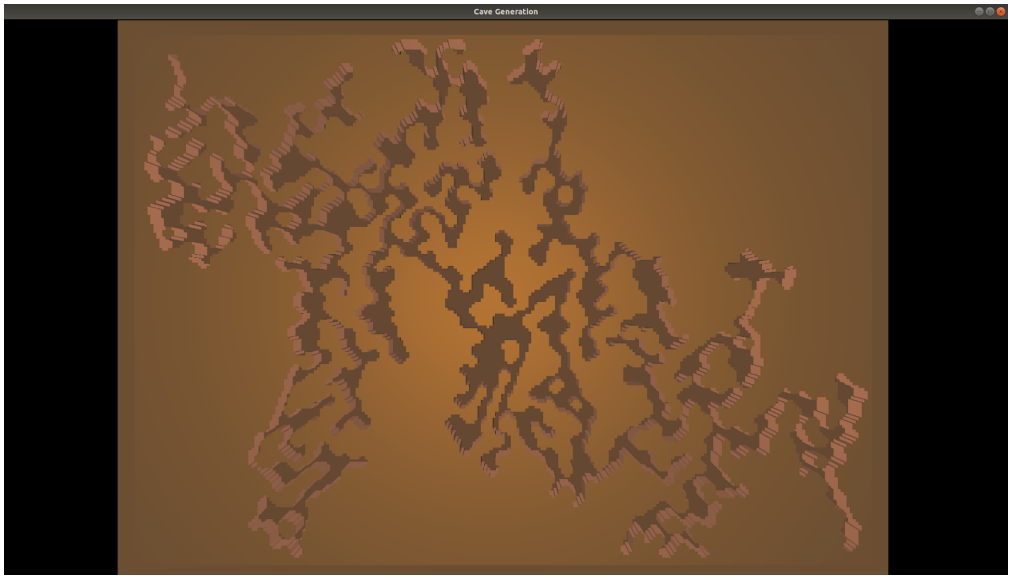


Figure 11: (Cave b) with inaccessible areas removed.

9.6 Stand-alone Sections

The fourth and final step removes occupied cave features not connected to the cave border. Again a flood fill algorithm is utilised to filter only occupied cells that are connected to at least one border cell. This has an effect of creating large open areas which improves the variety of cavern structures, however removes all instances of loops within the cave which are present in real world examples for example, pillars. As real world caves do contain loops and pillars the extraction of these stand-alone sections is random per cave. If a cave has an odd value for the smoothing iteration value, i then stand-alone sections are maintained, and if i is even then stand-alone sections are removed. At this stage the requirements set out in the objectives were satisfied.

9.7 Presets

To accommodate streamlined analysis of drone performance and testing the notion of cave presets were materialised. This allowed via the functions keys F1-F5 to generate 5 distinct predefined caves. For testing and analysis purposes the 5 different caves had sufficient differences between them and were bound to these presets. An external text file 'config.txt' is editable by the user to change the generation parameters for each preset. This includes:

- Offset X - Simplex noise offset in the 1st dimension, used as a seed value.
- Offset Y - Simplex noise offset in the 2nd dimension, used as a seed value.
- Fill Percentage - threshold value for Simplex noise continuous values.
- Noise Scale - scale factor of Simplex noise.

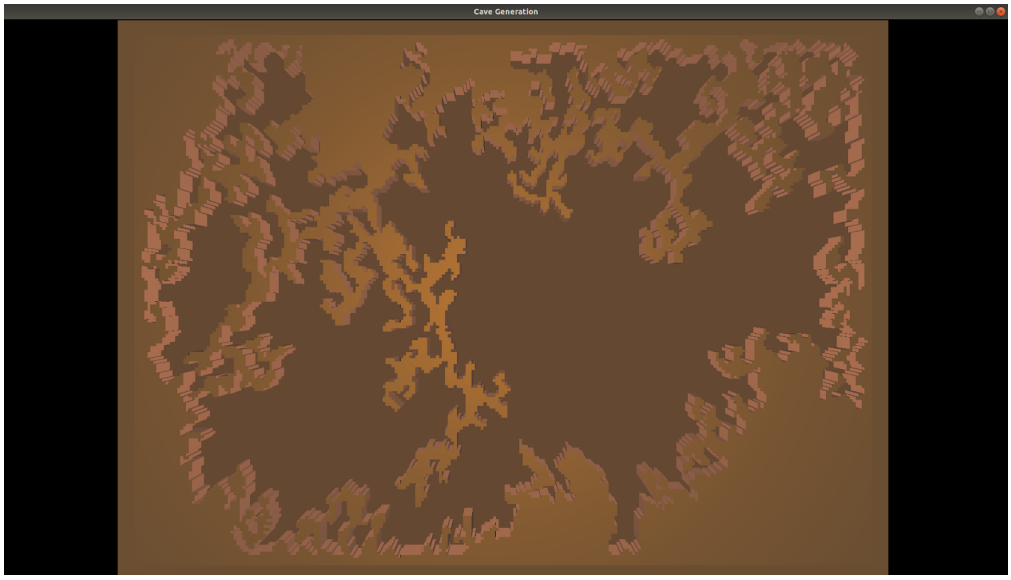


Figure 12: (Cave a) with stand-alone sections removed.

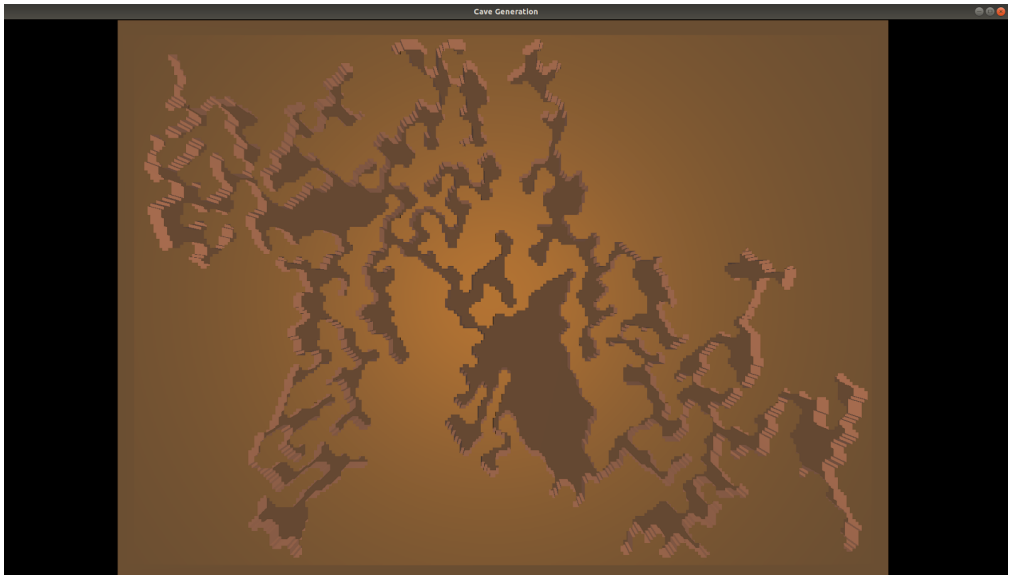


Figure 13: (Cave b) with stand-alone sections removed.

- Iterations - number of iterations of the cellular automata algorithm to execute.

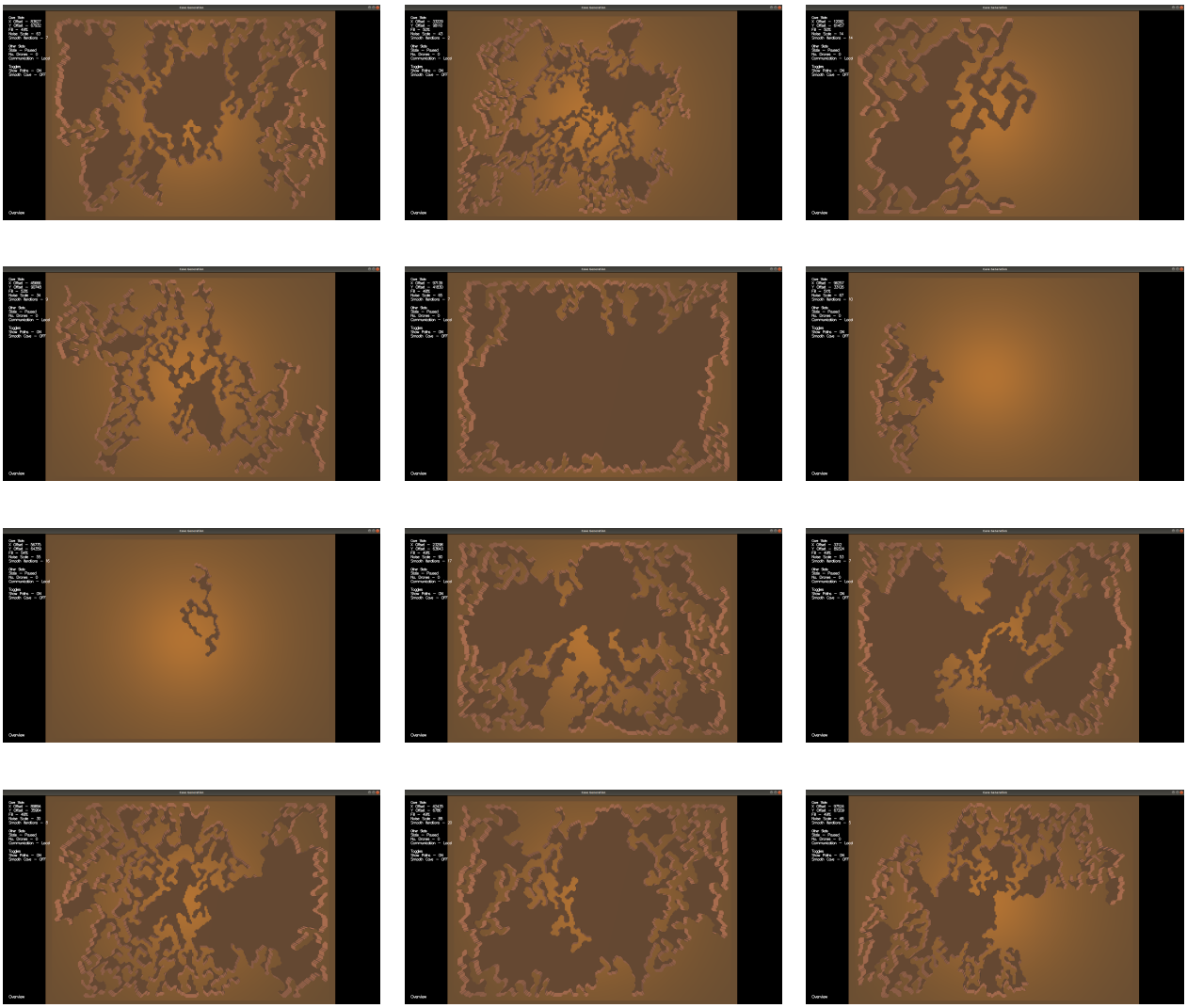
For testing and performance analysis, 5 caves were defined in the configuration file and reused for comparing approaches, algorithms and number of drones. Here are the parameters defined for each preset:

Preset	***Offset X***	***Offset Y***	***Fill Percentage***	***Noise Scale***	***Smoothing Iterations***
1	42435	6786	49	88	2
2	3312	89324	49	53	7
3	45666	90749	53	34	9
4	33229	98110	50	43	2
5	42435	6786	49	88	20

Table 1: Cave preset parameters

9.8 Cave Examples

Here are some examples of caves which could arise from the cave generator. The method of incorporating Gaussian distributions for some generation parameters meant most caves conformed to an ideal structure but the generation allows anomalies and a sparse range of cave types. In the examples cave structures range from small to large, jagged to smooth and tight to open.



10 Individual Search

10.1 Overview

This section describes how an individual drone performs sensing, frontier detection, optimal frontier selection and pathfinding. In each unique timestep the drone first performs a sense operation revealing all cells in the vicinity which are in line of sight of the sensing drone. Then sensed data is merged with the drone's own version of the cave called the internal map. Next new frontier cells are identified as unoccupied cells adjacent to an unknown cell. Once all the frontier cells have been identified a single frontier is chosen as the next target to navigate to using the A* search algorithm.

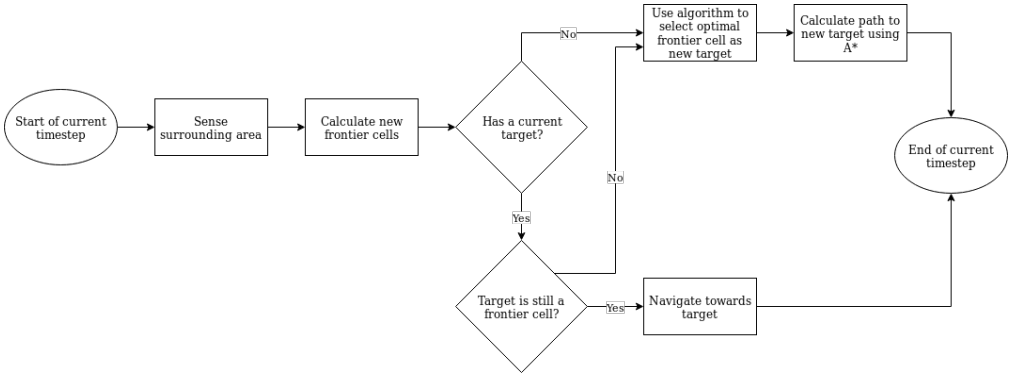


Figure 14: Individual search overview for one timestep.

10.2 Sensing

Firstly, sensing of the local environment must be executed as a prerequisite to all other search processes needed to map the cave. Sensing involves identifying all cells in the cave which the drone has line of sight of (i.e. no occupied cell intersects the line from the drone to the target cell) and is within a specified distance r from the drone. The value of r can be set in the configuration file.

A real-world implementation would primarily utilise LIDAR which is a combination of 3D scanning and laser scanning, particularly useful for airborne applications. Due to the effectiveness and accuracy of laser technology decreasing the further away an object is, the model reflects this issue by ignoring all sensed data beyond a given search radius. Additionally laser scanning cannot penetrate cavern walls and so can only detect objects visible directly from the position of the scanner, this is reflected in the model by only sensing cave cells which have line of sight with the drone.

The model's implementation is organised into multiple steps to identify the set of cells which can be merged with the drone's internal map.

1. For each cell in a rxr block surrounding the drone's position, calculate the distance between the cell and the drone and add these cells to set a. Where r is the search radius.

2. Remove cells in set a having a distance from the drone greater than r.

3. Sort the cells in set a by distance in ascending order.

4. Copy cells from set a to set b where the distance is less than or equal to 1, maintaining order.

5. For each cell in set a starting in order from the smallest distance: If the drone has direct line of sight with the cell add it to set b otherwise do nothing. Use all cells with less or equal distance in set a to determine collisions.

6. Output set b as the newly identified sensed cells.

The line of sight checking function requires knowledge of all cells that intersect a line going from the drone's position to the target. Research concluded that Bresenham's line algorithm was not sufficient as it only would identify one point per axis. An alternate algorithm 'Bresenham-based supercover line algorithm'[6] was found to achieve the goal of identifying all the points intersecting a line, however due to the application of this algorithm for line rasterisation and its complexity it was decided not to be implemented. Instead a custom made function was created using the parametric form of the equation of the line. For each step in x, one or more cells can be identified to intersect the line and then checked against the cave. For lines with a gradient greater than 1, the roles of x and y are reversed for efficiency. The algorithm for this can be found in more detail in section 11.2.1.

Figure 15: Drone Alpha and its sensed local environment with the cave overlayed with its internal map.

In the visualisation tool, cells are colour coded to separate cell types. Green indicates a free cell, Cyan cells are frontier cells (also free cells which have an unknown cell as a neighbour), Red cells are occupied cells, and the single Yellow cell is the drone's current target which it

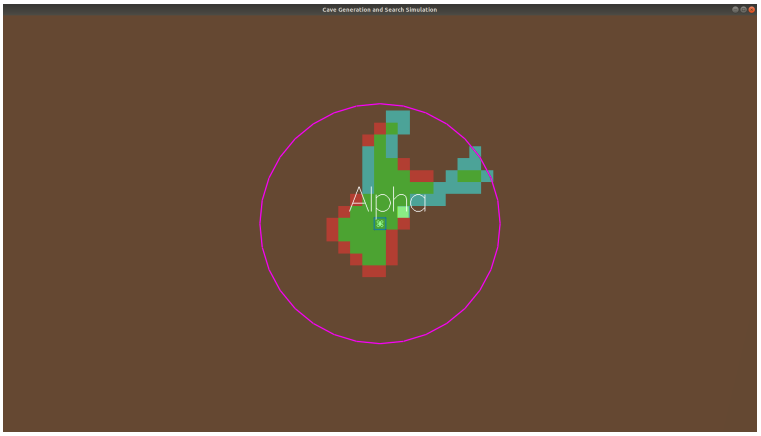


Figure 16: Drone Alpha and its sensed local environment with only its internal map.

is attempting to navigate towards. The search radius of the drone (in this instance it is set to 10 cells) is shown as the purple circle surrounding the centre of the drone. Any cells that have their centre coordinates on or inside this circle are processed by the sensing algorithm in the model.

10.3 Internal Map

Each drone contains a data structure named the 'Internal map' for the accessing, storage and retrieval of known cave cells. It is implemented as a vector of a vector of integers providing it with a 2D structure reflecting its real world counterpart. This provides constant time complexity for many of the most common operations performed on the data structure such as element access, element assignment and element comparison. All cells within the internal map are initialised to unknown cells.

At each timestep data obtained from sensing is merged with the current internal map. The map can only distinguish between free, occupied and unknown cells. Frontier cells are stored elsewhere in a separate list to improve efficiency with accessing, storage and removal.

10.4 Frontier Detection

Frontiers are the keystones for exploration in the cave, they indicate a potential for further exploration. They are identified after sensing and defined as unoccupied cells that are adjacent to at least one unknown cell.

There some advantages to using frontier cells over the preexisting unknown cells. Frontiers indicate a cell where the drone can freely navigate to without causing a collision because it is also a free cell, unknown cells could be free or occupied and could be unreachable by the drone. Additionally as frontier cells only exist on the boundary between free and unknown cells there are inherently less of them and as such less of them to analyse meaning faster operations to calculate the optimal frontier.

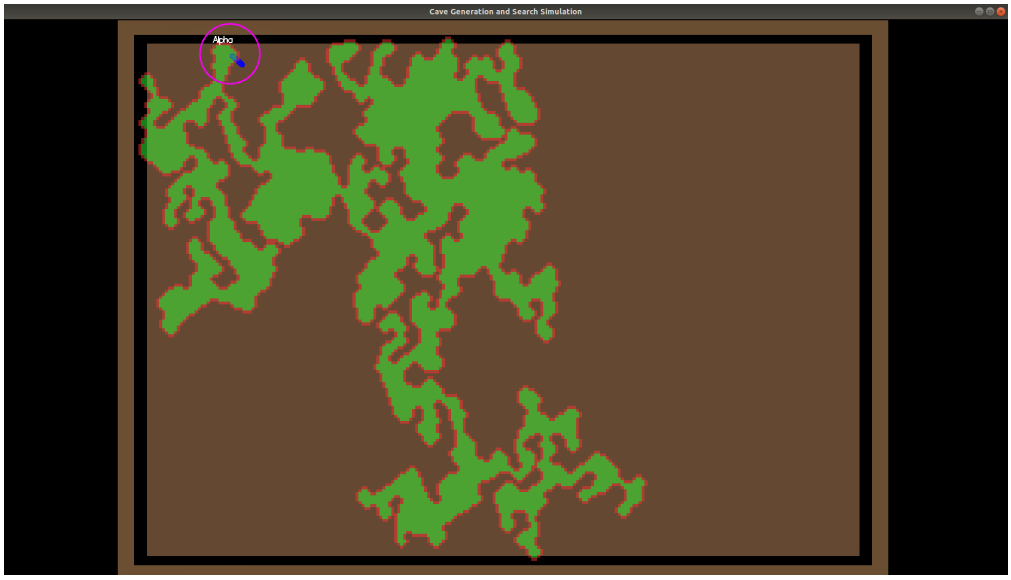


Figure 17: Internal map after a complete search of a cave has been completed.

The drone will continuously select a frontier cell and navigate to it. As the drone navigates to its target, sensing is performed simultaneously and in the process discovers more cells thus identifying more frontier cells. Frontier cells are defined as unoccupied cells that are adjacent to an unknown cell. Here is the algorithm to identify new frontier cells from the set of sensed cells in the current timestep.

1. Let set a be the set of sensed cells in the current timestep, let set b be an empty set, and let m be the drone's internal map.
2. For each occupied cell i in set a - check the adjacent neighbours of i, if a neighbour is a frontier cell in m then add the neighbour to set b and set the neighbour in m to free.
3. For each free cell i in set a - add the cell to set b and set the cell in m to free.
4. For each cell i in set b - if cell i has an unknown neighbour set cell i in m to a frontier.

This algorithm identifies frontier cells which are located at a distance d or less from the drone. It avoids recalculations of frontiers by ignoring cells outside this range to make the algorithm as efficient as possible.

10.5 Frontier Selection

At each timestep there is a set S of current frontier cells. The drone must choose using an appropriate algorithm a single frontier cell from this set to navigate to. The drone will continue traversing towards the frontier cell until it is terminated by either: sensing the unknown cells adjacent to the target frontier cell causing the target frontier cell to change to

an unoccupied cell, or during merging of internal maps with another drone where the target cell has been identified as an unoccupied cell in the communicating drone. In both cases navigation terminates due to the target frontier cell changing state to an unoccupied cell. Once navigation is terminated a new target cell must be chosen from the remaining frontiers in the set S.

Two frontier selection approaches were constructed and compared by a performance metric based on the total euclidean distance travelled by a drone over the entire search of an environment.

10.5.1 Nearest Frontier

The nearest frontier approach chooses a single frontier from set S that has the minimum Euclidean distance between itself and the drone's current position. This is a greedy approach that chooses the local optimum which may not necessarily be the optimal choice globally, however does perform suitably well for a simple and naive approach. In some circumstances particularly apparent in confined and dense caves, a frontier may be chosen that appears to have the lowest euclidean distance, but is obstructed by a cavern wall between the frontier and drone. This leads to inefficient behaviour as a longer path is required to divert around the obstruction and can lead to extensive backtracking.

Figure 18: Nearest frontier approach identifies an optimal frontier as the next target.

10.5.2 Latest Frontier

Due to the inefficient behaviour exerted by the nearest frontier approach, a second approach was formulated to combat this and improve on efficiency. The latest frontier approach selects a single frontier that was most recently discovered. To be able to identify this frontier, a value corresponding to the timestep when the frontier was discovered must be stored alongside every identified frontier. In the case of multiple frontiers possessing this criteria, the frontier with

Figure 19: Nearest frontier approach identifies a sub-optimal frontier as the next target. A region WSW of the drone would be the optimal target choice.

the lowest euclidean distance to the drone is chosen. This approach minimises the inefficient behaviour present in the nearest frontier approach as backtracking is greatly discouraged.

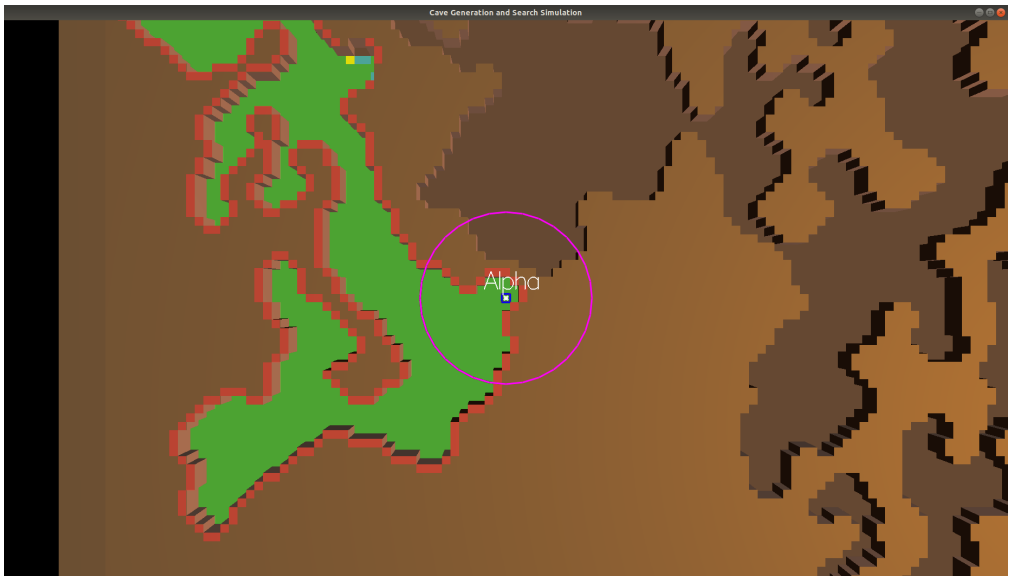


Figure 20: Latest frontier approach identifies an optimal frontier as the next target.

10.5.3 Analysis

To verify that the behaviour exerted by the nearest frontier approach was sub-optimal and that the latest approach was indeed an improvement, 5 distinct caves were generated. A single drone explored each of these 5 caves using each of the approaches mentioned and the total distance travelled during each simulation was recorded. In conclusion the latest frontier

approach outperformed the nearest frontier approach for every cave and on average achieved a 10.6% speed up. Therefore, the latest frontier approach was chosen as the ideal method for individual searching and as the base case for searching with multiple drones.

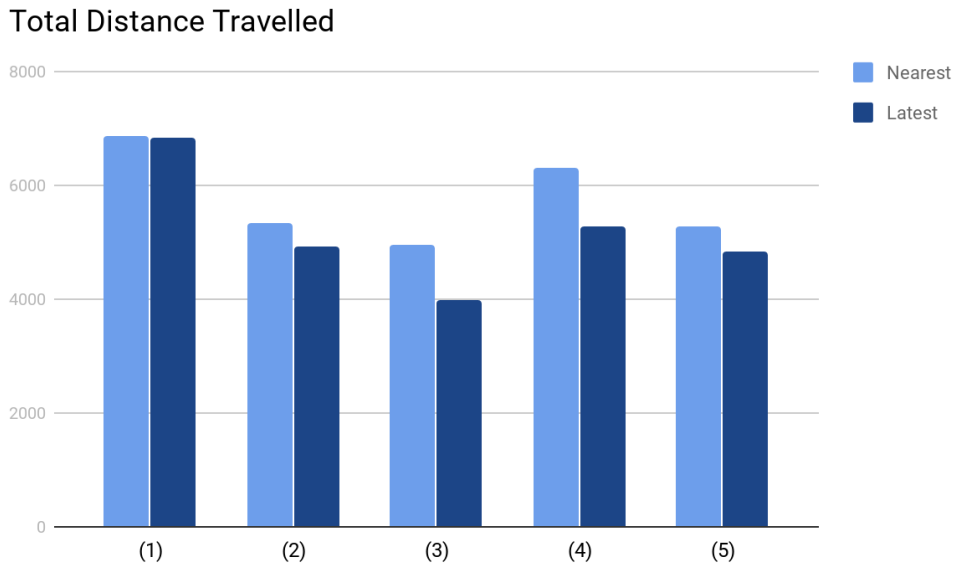


Figure 21: Comparison between the Nearest and Latest Frontier approaches for the total distance travelled.

10.6 Pathfinding

Once a target cell has been established from the latest frontier approach, the drone must proceed to navigate towards it along an efficient path that has a minimal distance. The A* search algorithm was implemented to calculate the shortest path between the drone's current position and the target cell.

10.6.1 A* Search Algorithm

The A* search algorithm is a pathfinding and graph traversal algorithm that is an extension to Dijkstra's algorithm as it uses heuristics to guide the path in the direction of the target. To model the drone in a cave, the algorithm was adapted to only account for free and frontier cells and allowed to navigate in 8 directions. The algorithm outputs the calculated path as a list of cells starting from the drone's current position and ending at the target cell. In most cases the output path is optimal, in cases where it is sub-optimal the performance is not hindered dramatically.

10.6.2 Unreachability Problem

An unforeseen and rare problem arose during implementation concerning navigating to frontier cells that were unreachable. Unreachable frontier cells could rarely arise from instances of caves where small tunnels restrict the drone's sensing capability creating a disconnected region of frontier and/or free cells. If a target cell was selected as one of these frontier cells in a disconnected region then the A* search algorithm would attempt to search the entire cave. This greatly slowed down the performance of the simulation and eventually the algorithm would fail to output a valid path and the program would throw a segmentation fault.

An initial solution was constructed by implementing a detection of these unreachable frontiers by analysing the output of the A* search algorithm. If the path was valid then the target was identified as reachable. If the path was invalid or incomplete then the target was identified as unreachable. If a target was unreachable then the target was removed from the list of frontiers and a new target was calculated until a valid target was found. Whilst this solution did prevent the arisen segmentation faults, the program continued to perform slow in the presence of these unreachable targets.

Finally, to prevent the performance bottleneck, these unreachable frontiers were identified significantly earlier in the A* search by performing the search in the opposite direction (starting from the target). This stopped the performance bottleneck as unreachable frontiers could be identified in significantly fewer iterations of the search algorithm.

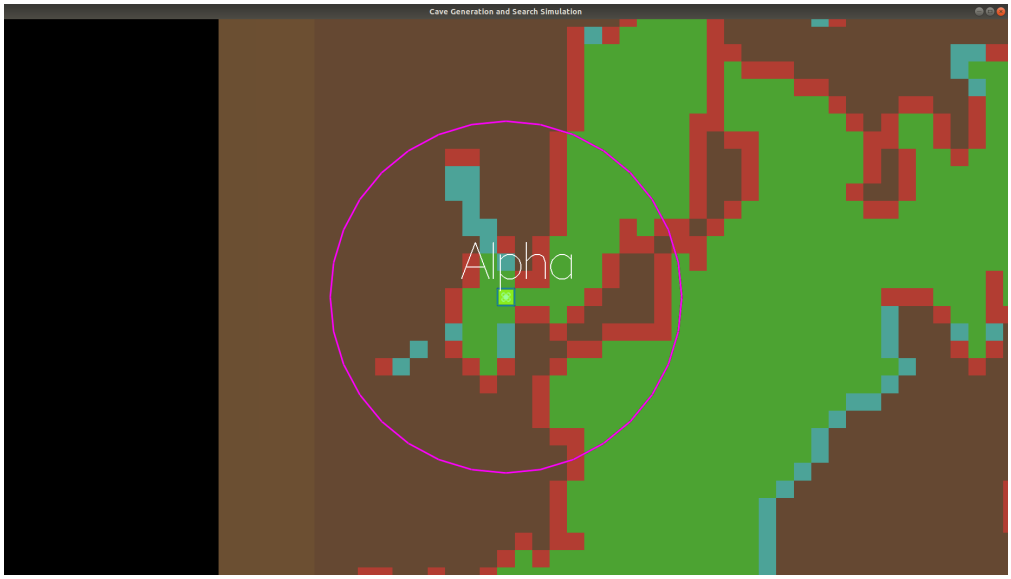


Figure 22: An example of a disconnected region containing an unreachable frontier cell.

10.6.3 Finishing Exploration

Whilst the drone has a current target it will continue to be in a navigation mode and at each timestep move closer and closer to its target following a defined path whilst continuously

and simultaneously searching the local environment. In the case where the current target is included in the local search and identified as free then the target is cleared and a new target is calculated. Additionally a new target may be calculated if the current target is identified as unreachable, or in an instance with multiple drones, if the target cell has been identified by another drone as free and communicated to the drone. If the drone doesn't have a current target then a new target is calculated from the list of frontier cells, if no frontiers remain then the cave has been fully explored and the simulation is terminated.

11 Teamwork Searching

11.1 Overview

This section describes the extension of individual cave exploration and mapping for more than one drone. The drone's behaviour has been altered with the addition or updating of particular behaviours and operations for communication and frontier selection.

An important added operation is the availability to communicate information between drones. This allows two drones to combine their internal maps, which reduces the area of the cave searched as previously searched areas by another drone can be ignored. Less area is repeatedly explored and thus caves are potentially explored with greater efficiency and speed.

Now that drones can effectively share information, the base approach of frontier selection would mean every drone would attempt to navigate to the same target cell. This is inefficient as the performance would be the same as if just a single drone was exploring the cave. The base case fails to improve efficiency for multiple drones, therefore, frontier selection must now include some variation and randomness so each drone explores different paths and branch out whilst still prioritising optimal frontiers. Ideally at a junction in the cave where two drones are approaching, each drone should traverse down opposite paths, drones should avoid contact with each other and keep distance to minimise repeated exploration. These behaviours mimic swarm intelligence such as separation.

The upper bound of improvement for a cave of area a with n drones present would be each drone exploring an area of a/n. Realistically this is unachievable as there is inherently some overlap in the areas explored by each drone, however this value provides a target performance and a benchmark for the ideal efficiency.

11.2 Communication

There are two implemented communication protocols for communication: local and global communication. Local communication reflects the conditions in caves as GPS and other forms of communication cannot penetrate through cavern walls, so communication can only be performed if two drones are close enough and have direct line of sight. Global communication is different in that every drone can communicate regardless of distance or line of sight. Communication using a local protocol is irregular and uncommon, whilst communication using a global protocol is regular and constant. To reduce computation and repeated operations a minimum time must have passed between when two drones can communicate. This is so two drones cannot communicate in consecutive timesteps where little or no change in the internal maps are present. This buffer value was set to 25 timesteps before two drones were allowed to begin communication after a previous interaction.

The communication operation is strictly one-way between two drones, therefore for two drones: *Alpha* and *Beta* the protocol follows this procedure:

1. *Alpha* sends its internal map information to *Beta*.

2. *Beta* combines the received information with its internal map.
3. *Beta* sends its internal map information to *Alpha*.
4. *Alpha* combines the received information with its internal map.

Once information has been received, the drone's internal map and received information must be combined. This is a O(wh) operation, where w is the cave width and h is the cave height. Combination of the two maps is both trivial in the cases where both maps are identical, however frontier cells and cells adjacent to frontier cells require more elaborate checks before they can be assigned appropriate values.

There are three cases which require alterations in the internal map if found. For a cell in the received map, r, and the corresponding cell in the internal map, i:

1. If r is Occupied and i is Unknown.
   - Set i to Occupied
   - Add any immediate neighbours (directly N,E,S,W) of i which are frontiers to the set *frontierCheck*.
2. If r is Free and i is either Unknown or Frontier.
   - Remove the cell from the *frontierCells* set.
   - Set i to Free.
   - Add any immediate neighbours (directly N,E,S,W) of i which are frontiers to the set *frontierCheck*.
3. If r is a Frontier and i is either a Frontier or Unknown.
   - Remove the cell from the set *frontierCells* if i is Frontier.
   - Set i to Free.
   - Add the cell to the set *frontierCheck*.

Listing 4: Algorithm for combining the received refMap and intMap.

```
For i = 0 to  caveWidth - 1
For j = 0 to caveHeight - 1
  If (refMap(i,j) == Unknown)
    Continue
  Else if (refMap(i,j) == Occupied && intMap(i,j) == Unknown)
    intMap(i,j) = Occupied
    If (i - 1 >= 0 && intMap(i-1,j) == Frontier)
      Add cell (i-1,j) to frontierCheck
    If (i+1 < caveWidth && intMap(i+1,j) == Frontier)
      Add cell (i+1,j) to frontierCheck
    If (j-1 >= 0 && intMap(i,j-1) == Frontier)
      Add cell (i,j-1) to frontierCheck
```

```
    If (j+1 < caveHeight && intMap(i,j+1) == Frontier)
      Add cell (i,j+1) to frontierCheck
  Else if (refMap(i,j) == Free && intMap(i,j) != Free)
    If (intMap(i,j) == Frontier)
      Remove cell (i,j) from frontierCells
    intMap(i,j) = Free
    If (i - 1 >= 0 && intMap(i-1,j) == Frontier)
      Add cell (i-1,j) to frontierCheck
    If (i+1 < caveWidth && intMap(i+1,j) == Frontier)
      Add cell (i+1,j) to frontierCheck
    If (j-1 >= 0 && intMap(i,j-1) == Frontier)
      Add cell (i,j-1) to frontierCheck
    If (j+1 < caveHeight && intMap(i,j+1) == Frontier)
      Add cell (i,j+1) to frontierCheck
  Else if (refMap(i,j) == Frontier && intMap(i,j) != Free)
    If (intMap(i,j) == Frontier)
      Remove cell (i,j) from frontierCells
    intMap(i,j) = Free
    Add cell (i,j) to frontierCheck


For each cell(x,y) in frontierCheck
  If (x-1 >= 0 && internalMap(x-1,y) == Unknown)
    internalMap(x,y) = Frontier
  Else If (x+1 < caveWidth && internalMap(x+1,y) == Unknown)
    internalMap(x,y) = Frontier
  Else If (y-1 >= 0 && internalMap(x,y-1) == Unknown)
    internalMap(x,y) = Frontier
  Else If (y+1 < caveHeight && internalMap(x,y+1) == Unknown)
    internalMap(x,y) = Frontier
```

- *refMap* - is a 2D data structure containing the internal map information received from another drone.
- *intMap* - is a 2D data structure containing the internal map information of the drone.
- *frontierCheck* - is a set of cells, used to storage cells which are potential frontiers.
- *frontierCells* - is the set of all frontier cells identified by the drone.

11.2.1 Local Communication

Local communication concerns communication limited by distance and line of sight between two drones, imitating the conditions within a cave system where communication methods such as GPS are not available. To model local communication every unique pair of drones must be looked at individually, to check they are within a specified distance of each other and that there is an unobstructed line of sight between the two drone positions.

Figure 23: Drone Alpha's internal map before communication with drone Beta.

Figure 24: Drone Alpha's internal map after communication with drone Beta.

Calculating the distance between two drones is trivial and performed first as to eliminate the most candidates for communication. The line of sight check is more computationally demanding and so it is performed second if the distance criteria is met. The linear complexity function *lineOfSightCheck* identifies if there is or is not an obstruction between two given points. Given one drone's coordinates (ax,ay) and another drone's coordinates (bx,by) the function returns a boolean value by checking all cells intersecting the line from (ax,ay) to (bx,by).

Listing 5: Algorithm for identifying whether a line between two points is obstructed.

```
1 //Two points have the same x value.
2 if (ax == bx)
3   for (i = min(ay,by); i <= max(ay,by); i++)
4     if (cave(ax,i) == Occupied)
5         return false
6
7
8 //Two points have the same y value.
```

```
else if (ay == by)
  for (j = min(ax,bx); j <= max(ax,bx); j++)
    if (cave(j,ay) == Occupied)
        return false


//Two points are not aligned by either axis.
else
  for (x = min(ax,bx); x <= max(ax,bx); x++)
    t0 = (x - 0.5 - ax) / (bx - ax)
    t1 = (x + 0.5 - ax) / (bx - ax)
    y0 = ay + (t0 * (by - ay))
    y1 = ay + (t1 * (by - ay))
    ymin = max((int)floor(floor((min(y0,y1) * 2) + 0.5) / 2), min(ay,by))
    ymax = min((int)ceiling(floor((max(y0,y1) * 2) + 0.5) / 2), max(ay,by))
    for (y = ymin; y <= ymax; y++)
      if (cave(x,y) == Occupied)
          return false

return true
```

Due to the criteria required for local communication to operate, the frequency of interactions between drones is limited and dependant on search radius, the type of cave and the size of the cave. One method of increasing the frequency of communication would be to backtrack periodically to locations where a previous encounter between drones took place, this however is highly inefficient and performance is better without attempting to increase the frequency of interactions through behavioural means.

11.2.2 Global Communication

Global communications concerns communication not limited by any factors such as distance, imitating conditions where a central unit can communicate with all drones simultaneously without obstruction, such as automatic home vacuuming using Wi-Fi or automated crop harvesting machines communicating via satellite or GPS.

Modelling global communication is trivial as there is no criteria for communication to operate. The frequency of communication is solely limited by a buffer value that limits the time between communication for a pair of drones. This is to reduce computation as consecutive timesteps will yield little or no change in the drone's internal maps.

To further increase efficiency whilst still retaining the spread of information, the number of communication operations was reduced to a minimum amount. Previously communication occurs between every pair of drone, for n drones this would equal a total of $2\sum_{k=1}^{n-1} k$ communication operations. Remember, communication is strictly one-way.

Listing 6: Previous implementation of global communication with $2\sum_{k=1}^{n-1} k$ communication operations.

```
for (i = 0; i < droneCount - 1; i++)
    for (j = i + 1; j < droneCount; j++)
```

```
3           communicate(i,j)
4           communicate(j,i)
```

The number of communication operations was reduced to $2(n-1)$, a considerable computational speedup as each communication operation is expensive. This was achieved by applying an order and flow to the data flow between drones. For n drones: drone 1 will communicate to drone 2, drone 2 will then communicate to drone 3, and so on until drone n. Next drone n will communicate with drone 1, then drone 1 will communicate with drone 2 and so on until drone n-1. This is analogous to the ring topology in computer networks, and therefore has the same disadvantage in that if one drone in the network fails the communication process as a whole fails.

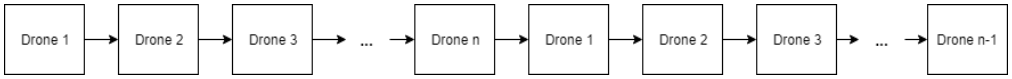


Figure 25: Improved communication flow between drones for global communication using the ring topology.

Listing 7: Improved global communication implementation with $2(n-1)$ communication operations.

```
1 for (i = 0; i < droneCount - 1; i++)
2           communicate(i,i+1)
3
4 communicate(droneCount-1,0)
5
6 for (i = 0; i < droneCount - 2; i++)
7           communicate(i,i+1)
```

	Number of Drones								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Previous	*0*	2	6	12	20	30	42	56	72
Improved	0	2	4	6	8	10	12	14	16

Table 2: Number of communication operations required for n drones for both the naive and improved approaches.

An alternative approach analogous to the star topology would be to have a central unit such as a satellite dedicated to receiving internal maps from all drones, combing them all together and distributing the combining map. This would mean all processing of combining the maps was solely handled by one dedicated unit and would not require the drones to communicate with each other, so if one drone begins to fail it will not affect any other drone in the network. This approach also reduces the number of communication operations to $2n$, requiring two more operations than the ring topology method. Additionally due to requiring combination of multiple maps and handling frontiers appropriately the combination function would require modification, therefore for simplicity the method analogous to the ring computer network topology was chosen.

Due to the higher frequency of communication operations in global communication over local communication, it had been hypothesised that global communication would achieve better efficiency and performance in instances with multiple drones, due to a larger proportion of cave area being less repeatedly explored.

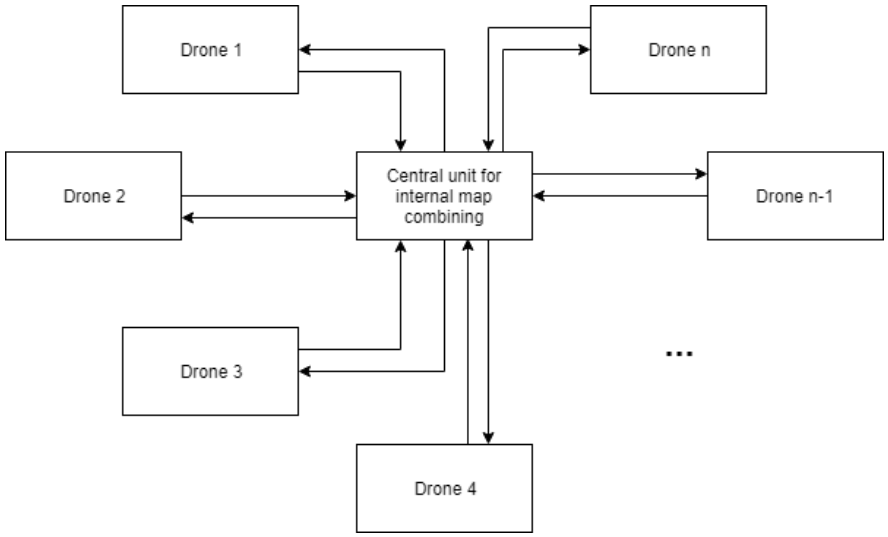


Figure 26: Improved communication flow between drones for global communication using star topology.

11.3 Frontier Selection

Previously, the best case for frontier selection had been established as picking the frontier that was most recently discovered. In the presence of multiple candidates the nearest of these were chosen. This method proved effective for instances with a single drone, however leads to the same performance in instance with multiple drones as every drone would choose the same frontier as their target and explore the same region of the cave. For example if a drone, *Alpha* approaches a new region, R, then fully explores it and leaves, then later another drone, *Beta* approaches the same region R it will explore the region in exactly the same fashion as drone *Alpha*.

This now presents a problem where ideally the drones should explore the best frontier, but having every drone explore it would be highly inefficient after a long period of time. Even a solution of distributing the n best frontiers to n drones proves inefficient and ineffective as there is high probability of these frontiers being in close proximity to each other.

Additionally, a desired behaviour in swarms is required where multiple drones are in close proximity. A frontier in a direction away from a nearby drone should be favoured over a frontier in the same or similar direction to a nearby drone. This helps prevent two drones choosing the same frontier, causes branching out at junctions, and separates drones so they cover a larger area.

Now there are two variables to consider when selecting a frontier: the time difference between the current time and when the frontier was discovered, and the bearing of the frontier in relation to any nearby drones. Furthermore, to reduce the chances of drones choosing the same target and encourage further branching out, some variation and randomness is added to the frontier selection algorithm.

11.3.1 Timestep Weight

Now it is not sufficient enough to simply choose the frontier that was most recently discovered, due to the planned implementation of variation, randomness and combination with other weight values. The time since last discovered of each frontier will still influence the decision of frontier selection however. Every frontier must be considered and assigned a weight value in the range of 0-1. A weight of 0 specifies an undesirable frontier, whilst a weight of 1 specifies a highly desirable frontier that should be prioritised.

For every frontier, the timestep weight value, *W(t)* is a given a value based on the when the frontier was discovered, *timestep*, in proportion to the minimum and maximum timestep values across all frontiers and scaled to the range accordingly.

$$W_t = (timestep - minTimestep)/(maxTimestep - minTimestep)$$

11.3.2 Bearing Weight

During attempted communication between two drones, a signal is sent from one drone to another and vice versa. If a signal is received by a drone that means another drone has line of sight and is in close proximity. The drone receiving the signal records the position of the drone sending the signal in order to calculate the bearing of the originator.

The bearing weight is a value in the range of 0-1 where again a weight of 0 specifies an undesirable frontier, whilst a weight of 1 specifies a highly desirable frontier that should be prioritised. This bearing weight is calculated from the product for all nearby drones of 1 minus the product density function of the difference in bearing of the frontier and nearby drone. This assigns a weight value which has a low value for frontiers in the direction of any nearby drone, and a high value for frontiers in the direction away from nearby drones.

$$W_b = \prod_{Drone} 1 - F(|\theta_{\text{frontier}} - \theta_{\text{Drone}}|, 0, \frac{\pi}{8}^2)$$

Where F is the probability density function of a normal distribution with the mean as 0 and the variance as $\frac{\pi}{8}^2$. If there are no nearby drones present then the weight value received of 1 is uniform for every frontier.

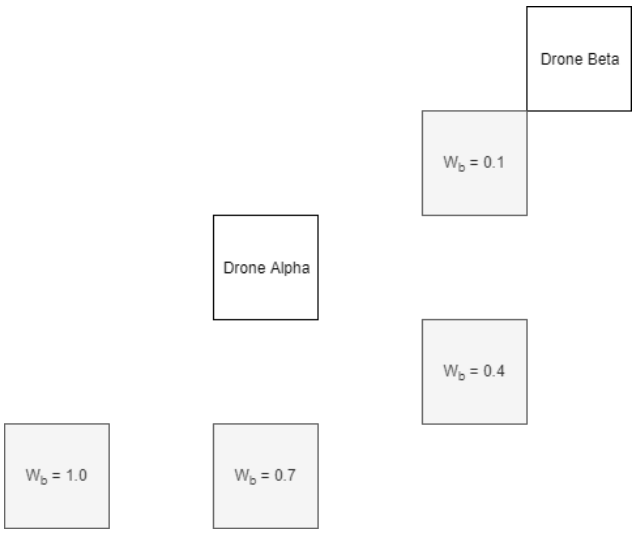


Figure 27: Bearing weight values of 4 different frontiers highlighted in grey for drone Alpha. Drone Beta is a nearby drone that influences the weights of the frontiers.

11.3.3 Updated Frontier Selection

Combining the bearing weight and timestep weight value produces a single weight value for all candidate frontier cells. A high value closer to 1 represents a frontier that was discovered recently and is in the direction away from any nearby drone.

$$W = W_t^6 * W_b$$

Now that the final weight value has been calculated some variation and randomness must be added whilst attempting to prioritise frontiers with a higher weight.

1. All frontiers are sorted in descending order according to the associated weight value W.
2. The cumulative weight, C is calculated by summing all weight values of each frontier.
3. A random value, r, between 0 and C is generated.
4. Finally the first frontier in the sorted list that has a weight value greater or equal to r is chosen as the target.

This concludes the updated frontier selection algorithm for simulation instances with multiple drones. If a drone requires a new target it will select a target using the base case defined in individual searching by choosing the single frontier which was discovered most recently (latest frontier approach) if and only if there are no nearby drones present. If however there is at least one nearby drone the frontier selection switches to the updated frontier selection algorithm incorporating the timestep and bearing weights.

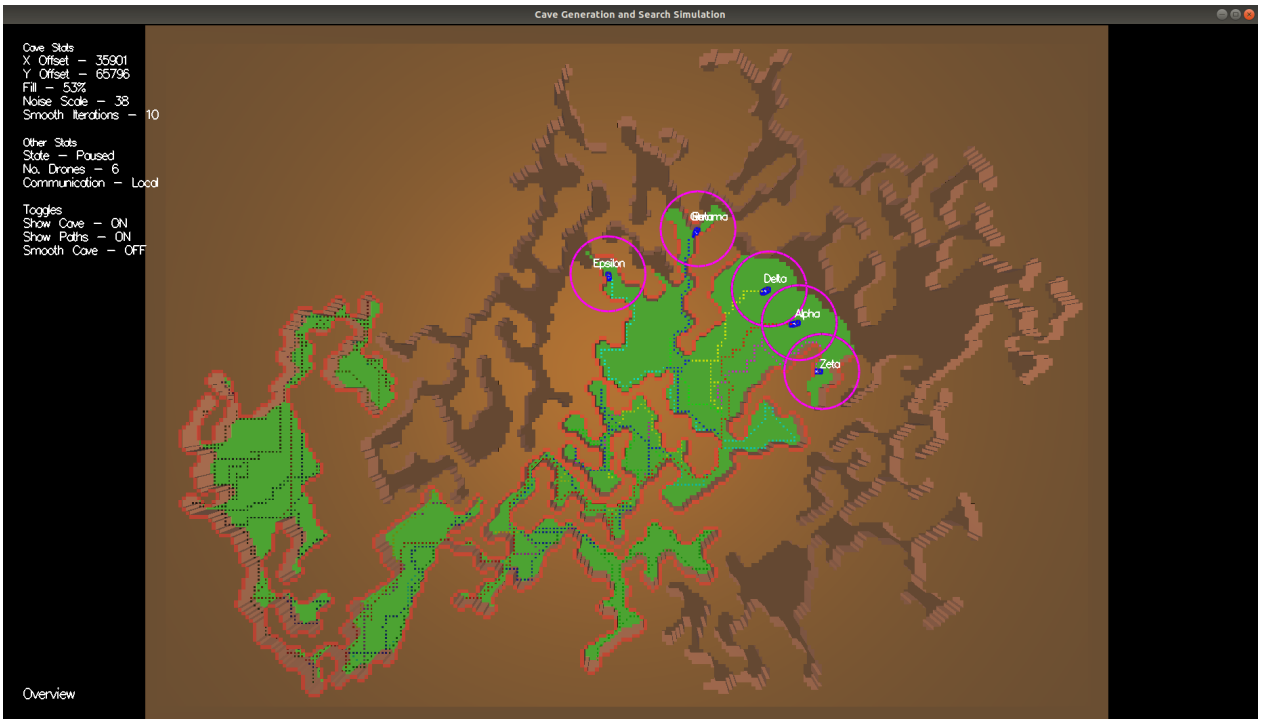


Figure 28: Simulation with multiple drones branching out exploring different sections of the cave.

12 User Manual

12.1 Controls and Keybinds

Here is a list of keybinds and corresponding actions for the GUI. All keybinds are case insensitive and are shown as an overlay when the program is first ran to help users.

Keybind	***Action***
'[' or Mouse scroll up.	Zoom in.
']' or Mouse scroll down.	Zoom out.
Up Arrow	Pan up.
Down Arrow	Pan down.
Left Arrow	Pan left.
Right Arrow	Pan right.

Table 3: Camera Control Keybinds

Keybind	***Action***
'h'	Shows/Hides the control overlay. Default: Shown.
'p'	Shows/Hides drone paths. Default: Shown.
't'	Enables/Disables the marching squares algorithm for cave wall smoothing.

Table 4: Visual Toggle Keybinds

Keybind	***Action***
'r'	Stops any ongoing simulation and generates a random cave.
F1-F5	Stops any ongoing simulation and generates a preset cave using parameters obtained from the configuration file 'config.txt'.

Table 5: Cave Generation Keybinds

Keybind	***Action***
'q'	Closes the program.
, ,	Resumes/Pauses the current simulation.
'v'	Rotates through the different camera modes. This includes an overview mode, and individual modes for each drone present which follows the drone's movement as it explores the cave.
1-9	Creates n drones for the simulation at the starting point.

Table 6: Simulation Keybinds

Figure 29: Overlay of the program controls shown on program startup.

12.2 Startup

To start the program navigate to the project files and type the following two commands to compile and run the program respectively.

1. ./build.sh
2. ./main

On program startup the configuration file is read and sets the preset cave variables as well as the specified communication method and search radius. The GUI is shown with the control and keybinds overlay which lists all the keybinds and their associated actions. This overlay can be toggled on and off with the keybind 'h'.

12.3 Generating caves

Before any simulation is run a suitable cave environment must be generated. On program startup a random cave is automatically generated and shown to the user. New random caves can be generated fast by pressing 'r'. If you desire a preset cave (one defined previously in the config.txt file) then pressing the function keys F1 through to F5 will yield the respective preset cave.

Once a cave has been generated, the parameters used to create it are shown in the top-left corner of the screen. These can be noted down to allow the cave to be reproduced by changing values in the configuration text file.

12.4 Running the simulation

Once a suitable cave has been generated, a number of drones from 1 to 9 must be spawned. Pressing the keybinds 1-9 will spawn the respective number of drones. Once the drones have imported the simulation is ready to commence. Pressing the spacebar will start the simulation and all present drones will explore and map the cave. Pressing spacebar again will pause the simulation.

Pressing 'v' will cycle through the different camera views. The default view is a stationary overview that shows the entire cave. This view will show a combined internal map of all drones. Other views will follow a single drone as it moves through the cave and will only show the internal map of the respective drone. The simulation will terminate automatically once the cave has been fully explored by all drones, or when the simulation is manually reset by generating a new cave.

13 Evaluation

13.1 Testing

To evaluate the performance of different frontier selection approaches and weights for instances with multiple drones, a series of test simulations were ran. Each test simulated n drones on a specific cave preset using either local or global communication. A total of 90 final tests were ran, 45 tests using a local communication protocol and 45 using a global communication protocol. Of these tests 5 different caves were generated and 9 tests were simulated on each cave using 1 to 9 drones respectively. The total distance travelled by each drone in a simulation was recorded and an average was taken across all drones. This average distance travelled value was taken as the performance metric to compare approaches and identify improvements in efficiency during the lifetime of the project.

Due to time constraints of the project and the speed and computational power required to run a simulation only 90 test could be performed. This was sufficient for accurate outputs as multiple tests all correlate similar findings, however in the future additional tests are to be carried out to further justify the findings and more accurately identify the specific parameters for optimal efficiency.

13.2 Performance

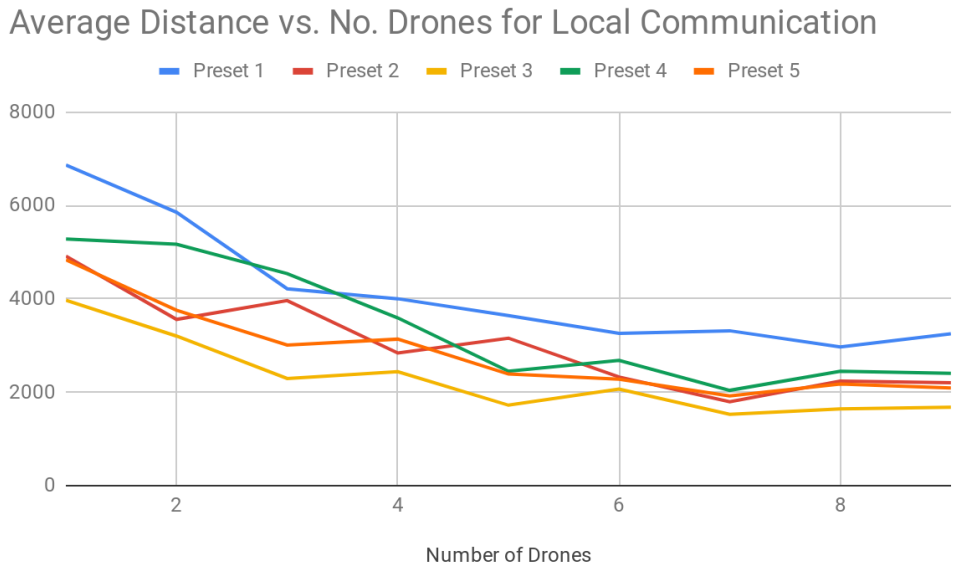


Figure 30: Average distance vs. Number of drones using local communication for 5 different presets.

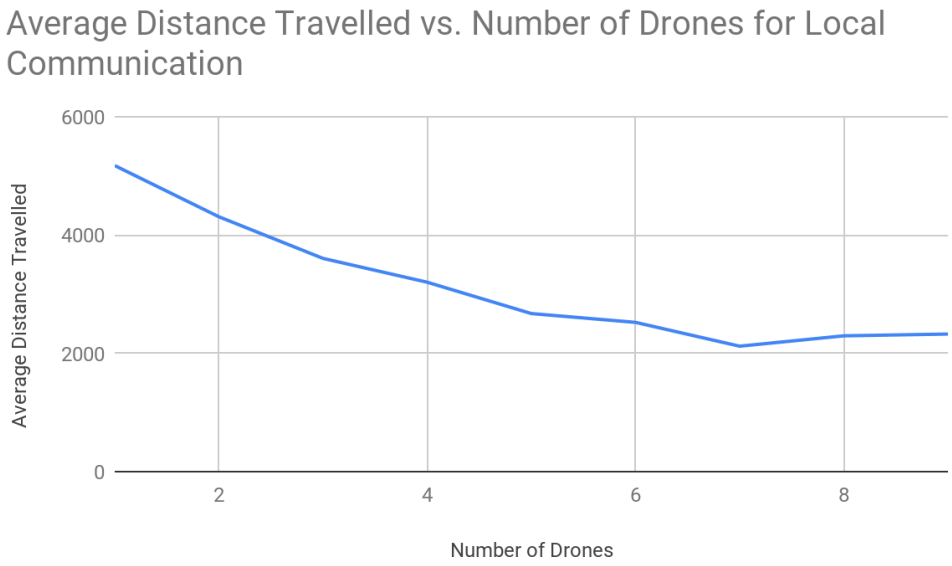


Figure 31: Average distance vs. Number of drones using local communication for 5 different presets with values averaged across the presets.

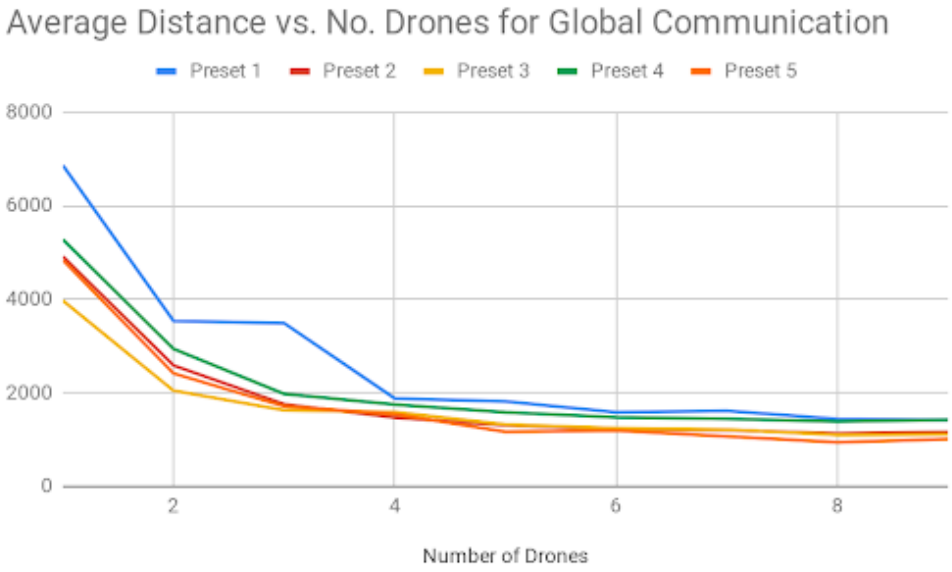


Figure 32: Average distance vs. Number of drones using global communication for 5 different presets.

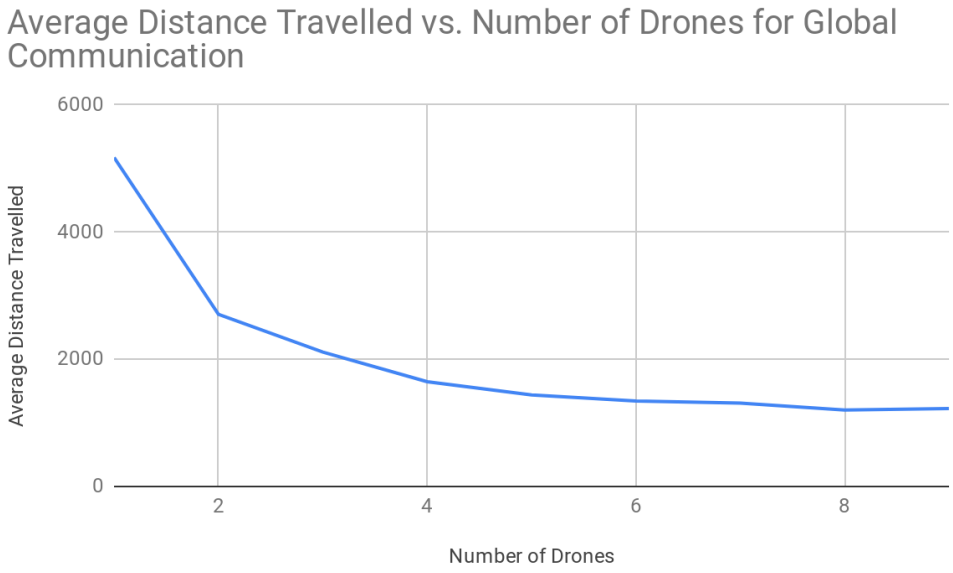


Figure 33: Average distance vs. Number of drones using global communication for 5 different presets with values averaged across the presets.

As hypothesised previously, the addition of multiple drones increased the efficiency of searching and mapping of cave environments indicating a successful result. By analysing the above graphs one can infer that local communication possesses the ability to speedup searching and mapping by up to 220% from its initial performance with a single drone, whilst global communication possesses the ability to speedup searching and mapping by up to 400% from its initial performance with a single drone. This is to be expected as global communication is not limited by distance and line of sight between drones and so communication occurs far more frequently. The increased efficiency with local communication is heavily dependant on the frequency at which communication occurs. This can be improved by increasing the search radius allowing communication to occur at a greater distance apart.

From the graphs, increasing the number of drones reduces the average distance travelled which is directly proportional to the time taken to complete the search. However the decrease in average distance travelled becomes smaller and smaller as the number of drones increases. Therefore, average distance travelled appears to be asymptotic to the number of drones which is further supported in that both communication protocols exhibit the same behaviour. This implies that there is a limit and optimal number of drones to employ for a particular simulation so that additional resources are not wasted. For the cave sizes and presets tested, using 7 drones had the lowest average distance travelled for a local communication protocol, whilst for a global communication protocol additional drones past 6 provided minimal improvement to the average distance travelled.

13.3 Conclusion

In conclusion the addition of multiple drones into a simulation improves the efficiency at which exploration and mapping is performed. Improvement appears asymptotic to the number of drones in a simulation where eventually increasing the number of drones present will yield negligible or reduced performance. This means an optimal number of drones to employ could present itself for given parameters. Employing the global communication protocol yields a far greater efficiency increase over the local communication protocol due to the increased frequency at which communication operations occur. In the real world one can hope that this may be implemented so that tasks such as cave exploration can be completed more efficiently and safely to benefit researchers, archaeologist and hobbyists, as well as contributors to other domains where this project has additional applications.

Overall this projects achieves the majority of goals initially set in the project objectives and requirements. Some of these goals eventually proved unfeasible due to time constraints or experience such as localisation, therefore additional resources were shifted into other components of the project such as swarm behaviour and cave generation.

Cave generation was initially foreseen as a small component of the model used to streamline the simulation and testing processes. However, the complexity of implementation as well as its potential applications outside of the project caused the component to have a higher focus, due to personal interests fuelling motivation. I am satisfied with the outcome of this component and have gained valuable insight and knowledge for procedural generation, which may seed an additional project focused heavily on procedural cave generation for video games.

13.4 Further Improvements

Overall I am satisfied with the outcome of the project however there are some aspects which given enough time, knowledge and resources would have been done differently. For example due to the speed at which simulations were able to run and the time constraint of the project, the number test simulations was limited. These test were used to identify the optimal weight values and approaches to use in the final model. Further tests would have helped to fine tune these parameters and potentially further increase the efficiency and performance.

Secondly, cave customisability is limited to either the randomness of generating new caves or manually specifying the generation parameters. This requires knowledge of how an individual parameter will affect the generated cave. Customisability could be improved by adding keybinds that can increase or decrease an individual parameter to provide a better understanding of the cave generation process. Furthermore the ability to bind a generated cave to a preset could be implemented with a single keybind (e.g. SHIFT+F1 to bind the current cave to preset 1), instead of requiring manual input into the configuration plain text file.

Thirdly, the visualisation of the cave required numerous draw operations for polygons particularly the OpenGL primitive GL_QUADS. Each cave cell is represented by a single polygon containing 4 vertices, the majority of these vertices are shared amongst multiple polygons but the current rendering implementation does not take advantage of this fact. Due to the visualisation not being the focus of the project, optimisation of the visualisation was excluded. To reduce the number of vertices and polygons present on each draw operations, alternative representations of the cave could be used such as triangle strips or by combining adjacent polygons into an extended polygon with considerably less vertices. For small cave environments, visualisation is smooth and fast, however as cave environment grow larger, there are more cells which require a larger number of vertices and polygons which leads to slow performance.

Fourthly and finally, navigation of a drone to its target frontier uses the A* search algorithm. The performance of the algorithm is the primary bottleneck in performance. Visible lag in the program can be observed when a path is created from drone to target that is of a considerable distance away or of sufficient complexity. This occurs as the heuristic attempts to guide the path but could end up attempting to explore every cell in the cave as the path may not always be direct. A solution to this project would be to explore alternative searching algorithms such as Dijkstra's, Depth-First search or Breadth-First search. If no alternative algorithms provide a solution or performance increase then further analysis of the current implementation is required to see how it could be improved. One such suggestion would be store when each individual cell was first discovered and when it was most recently detected to be used to aid backtracking. This would reduce the search space of the A* algorithm at the cost of additional memory overhead.

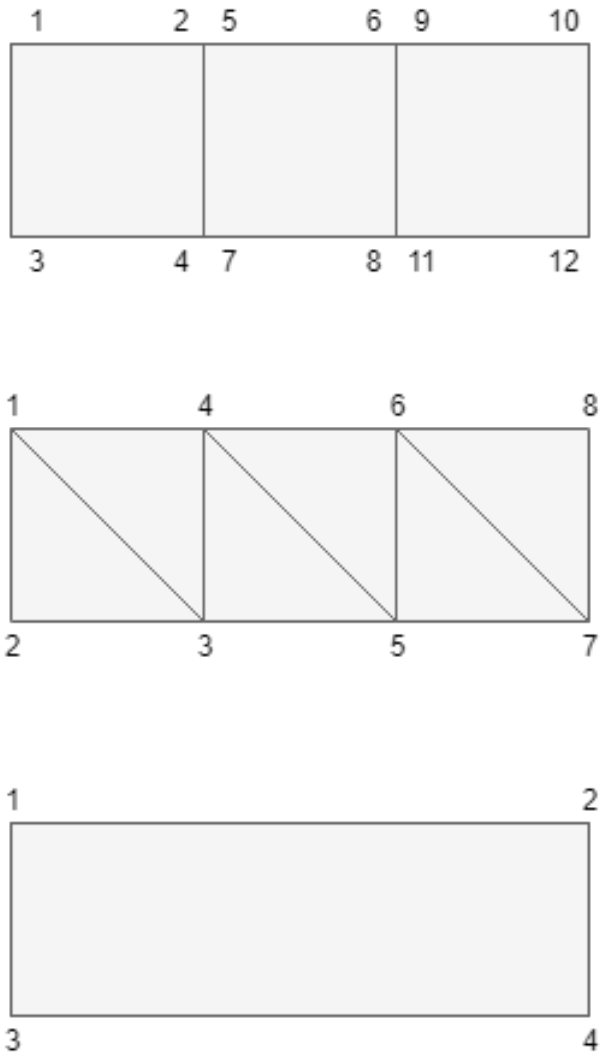


Figure 34: Top: Current implementation of 3 cells requiring 12 vertices. Middle: Triangle strip implementation of 3 cells requiring 8 vertices. Bottom: Extended polygon implementation of 3 cells requiring 4 vertices.

13.5 Future Directions

One main ambition of this project is to fully integrate it to a real world application by mapping a cave using a swarm of quad-copter drones. Currently the state of the project doesn't allow this as simulations are run in two dimensions. Expansion into three dimensions would be required which would require altercations and new development to make the data structures and algorithms work for the 3D case. The algorithms for modelling sensing, cave generation, communication, line of sight checking, storing the internal map, visualisation, combining maps, frontier selection and target navigation would all require extension into 3D. This task would require considerable time and resources, as such was not completed for this project. The 3D version of these algorithms would perform slower due to the added com-

plexity of the third dimension, and would also require a larger memory overhead. Therefore, additional research would have to be undertaken to maximise the performance to evaluation whether extension into 3D is feasible.

The localisation problem is a prominent challenge when designing robotic based applications. A robot cannot always be 100% certain of its position in a map and must used previously gathered data and its current sensed data to find its most probable location. All drones in the simulation currently are assumed to be able to localise themselves with 100% accuracy, i.e. knowing exactly their position in the map at all times. A real world application cannot make this assumption and so a future extension could be to model this inaccuracy. One such solution would be to use an implementation of SLAM (Simultaneous Localisation and Mapping) in conjunction with Kalman and particle filters to identify the position of the drone in the internal map as well as aid mapping and reduce errors in the presence of interference or noise that could present itself in recording devices.

References

[1] URL: https://git-scm.com/.

[2] *A hackable text editor for the 21st Century*. URL: https://atom.io/.

[3] BBC. *The full story of Thailand's extraordinary cave rescue*. https://www.bbc.co.uk/news/world-asia-44791998. Accessed: 18-03-2019.

[4] Padraig Belton. *Drones taking off as prices plummet*. https://www.bbc.co.uk/news/business-30820399.

[5] *Build software better, together*. URL: https://github.com/.

[6] Eugen Dedu. URL: http://eugen.dedu.free.fr/projects/bresenham/.

[7] H. Durrant-Whyte and T. Bailey. "Simultaneous localization and mapping: part I". In: *IEEE Robotics Automation Magazine* 13.2 (2006), pp. 99–110. DOI: 10.1109/mra.2006.1638022.

[8] National Geographic. *A Guide to Kentucky's Mammoth Cave National Park*. https://www.nationalgeographic.com/travel/national-parks/mammoth-cave-national-park/. Accessed: 08-10-2018.

[9] Khronos Group. *The Industry's Foundation for High Performance Graphics*. URL: https://www.opengl.org/resources/libraries/glut/.

[10] Lawrence Johnson, Georgios N. Yannakakis, and Julian Togelius. "Cellular automata for real-time generation of infinite cave levels". In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games - PCGames 10* (2010). DOI: 10.1145/1814256.1814266.

[11] Rob Matheson. *Fleets of drones could aid searches for lost hikers*. http://news.mit.edu/2018/fleets-drones-help-searches-lost-hikers-1102. Accessed: 22-03-2019.

[12] Mayo Foundation for Medical Education and Research. *Mayo Clinic*. https://www.mayoclinic.org/diseases-conditions/histoplasmosis/symptoms-causes/syc-20373495. Accessed: 08-10-2018.

[13] *Prometheus*. 2012.

[14] Commonwealth Scientific and Industrial Research Organisation. *Hovermap*. https://research.csiro.au/robotics/hovermap/. Accessed: 01-10-2018.

[15] Commonwealth Scientific and Industrial Research Organisation. *Zebedee*. https://research.csiro.au/robotics/zebedee/. Accessed: 01-10-2018.

[16] Panda Security. *Why are drones so suddently popular?* https://www.pandasecurity.com/mediacenter/news/what-is-the-future-of-drones/. Accessed: 25-03-2019.

[17] SRombauts. *SRombauts/SimplexNoise*. Oct. 2018. URL: https://github.com/SRombauts/SimplexNoise.

[18] SRombauts. *SRombauts/SimplexNoise - License*. Oct. 2018. URL: https://github.com/SRombauts/SimplexNoise/blob/master/LICENSE.txt.

[19] *Yorkshire Dales Cave Maps*. URL: http://cavemaps.org/.