# Town Planning with AI

By Philip Crispin

*Disclaimer: Please note that building construction plans throughout this project are programmatically taken from GrabCraft.com.*

# Summary

## Project Aim

The project aims to optimise town planning using AI with the Minecraft Framework (HTTP).

This project is a part of a group project with Aaron Fletcher that locates a suitable site and builds a town. Aaron's submission provides the building site locations and plot sizes. This project will deal with three related aspects: building roads, deciding what to build where, and the town's aesthetics.

I will build roads from each building site to each of the other building sites, where possible. **Roads will be built with minimum lengths while maintaining a realistic style.** For example, building two parallel roads immediately adjacent to one another might minimise the absolute distances for two routes, but that would not resemble a real-world town. Success can be achieved when the road lengths are near a minimum. However, a real-world style can only be decided objectively based on a birds-eye view of the city.

**The town facilities will be placed in the locations which are most accessible by inhabitants.** Several public buildings will be built, some houses and one or more blocks of flats. It will be assumed that inhabitants visit each public building a set number of times. For example, inhabitants visit factories five times per week. Success for this objective can be quantified when the total weekly distance travelled by the inhabitants is a minimum.

**All buildings will also be aesthetically consistent and resemble the local environment.** An element representative of the local environment will be used in each construction. Aaron Fletcher's Selective Project will provide the element. It ought to be easy to observe if this objective has been met in the town.

## Approach

An A* algorithm was used to build roads with the lowest possible distances between each building site. The A* algorithm systematically searches for routes faster than other algorithms because of a heuristic factor that guides the search towards the goal.
The roads were made to look more realistic by not allowing parallel roads to be built too close together. For this, a *snap-to-grid* cost factor made the algorithm prioritise routes along a predetermined grid.
All roads are three blocks wide. However, the processing cost increases significantly when checking for obstacles for wider roads. So, instead, a map was drawn where all obstacles were two squares too big on each side. The algorithm thus searched for roads a single square wide against this modified map, thereby massively reducing the processing costs while still avoiding obstacles.
The algorithm checked each square on the road to see if it was part of another road with the same destination. This prevented processing from being repeated. Such roads were more likely to be found because all roads tend towards a predetermined grid on the terrain.

To minimise inhabitants' weekly distance, I used a Breadth-First Search (BFS) algorithm to determine which public building should go in which site. This algorithm is complete and optimal, which means it considers every possible combination of buildings in each location and finds the absolute best solution. However, the processing time increased exponentially with the number of buildings, limiting the search to a maximum of 12 sites. So I incorporated a Monte Carlo Tree Search (MCTS) for larger towns.
MCTS does not consider every possible combination and does not necessarily find the best solution, but it does find a solution much quicker. MCTS tries different combinations, partly systematically and partly randomly. The algorithm mainly prioritises well-performing combinations, but it also explores uncharted combinations and occasionally re-explores less well-performing combinations just in case they had scored poorly from unlucky random choices. It is necessary to tune the algorithm with a constant to achieve a good balance in this exploration.

## Related Work

### Source

***Route optimisation issues and initiatives in Bangladesh: The context of regional significance***
*by Mahir Shahrier, Arif Hasnat, February 2021*
http://10.1016/j.treng.2021.100054

### Summary of Source

This source analyses the transportation system of Bangladesh, a country home to 2.18% of the world population, for the use of AI-based route optimisation algorithms in relation to country-specific transportation-related problems. First, the authors discuss a range of AI algorithms with their strengths and weaknesses. These strengths and weaknesses are then used to categorise the issues faced by Bangladesh's transport infrastructure. The source then makes recommendations on the suitability of specific AI algorithms in Bangladesh. The overall objective of the paper is to aid the development of an Intelligent Transport System.

### Relation to the project

The source directly addresses issues involved with combining AI and town planning, which was the goal of this project.

A large part of the source discusses the merits of algorithms such as A*, genetic algorithms and neural networks for route optimisation. In addition, the source details techniques and their applications. This provides an excellent high-level overview of methods to implement a route-finding solution.

The source delves into practical considerations required for implementing an Intelligent Transport System. This includes economic, ecological and geographical factors. In addition, the source also discusses social and personal concerns in journey decisions that can impact a transport system. This provides many aspects which could potentially influence the project design, thereby improving the realisticity of the town created.

The compatibility of Bangladesh and specific AI route-planning solutions is considered, which provides a real-world analysis of implementation and the difficulties. Finally, specific AI recommendations for Bangladesh's challenges are made, providing examples from which to learn for the project.

### Source Critique

Bangladesh and Minecraft are very different domains. The real-world practical considerations are primarily irrelevant for Minecraft. For example, the parameters involved in the real world can include safety, vehicle ownership, traffic jams, reckless driving, law enforcement levels and the type of road. Few of the recommendations or issues raised applied to a simulated world.

The breakdown of the different AI route-finding solutions was helpful, but I would not recommend this source as there may be better alternatives.

# Figure

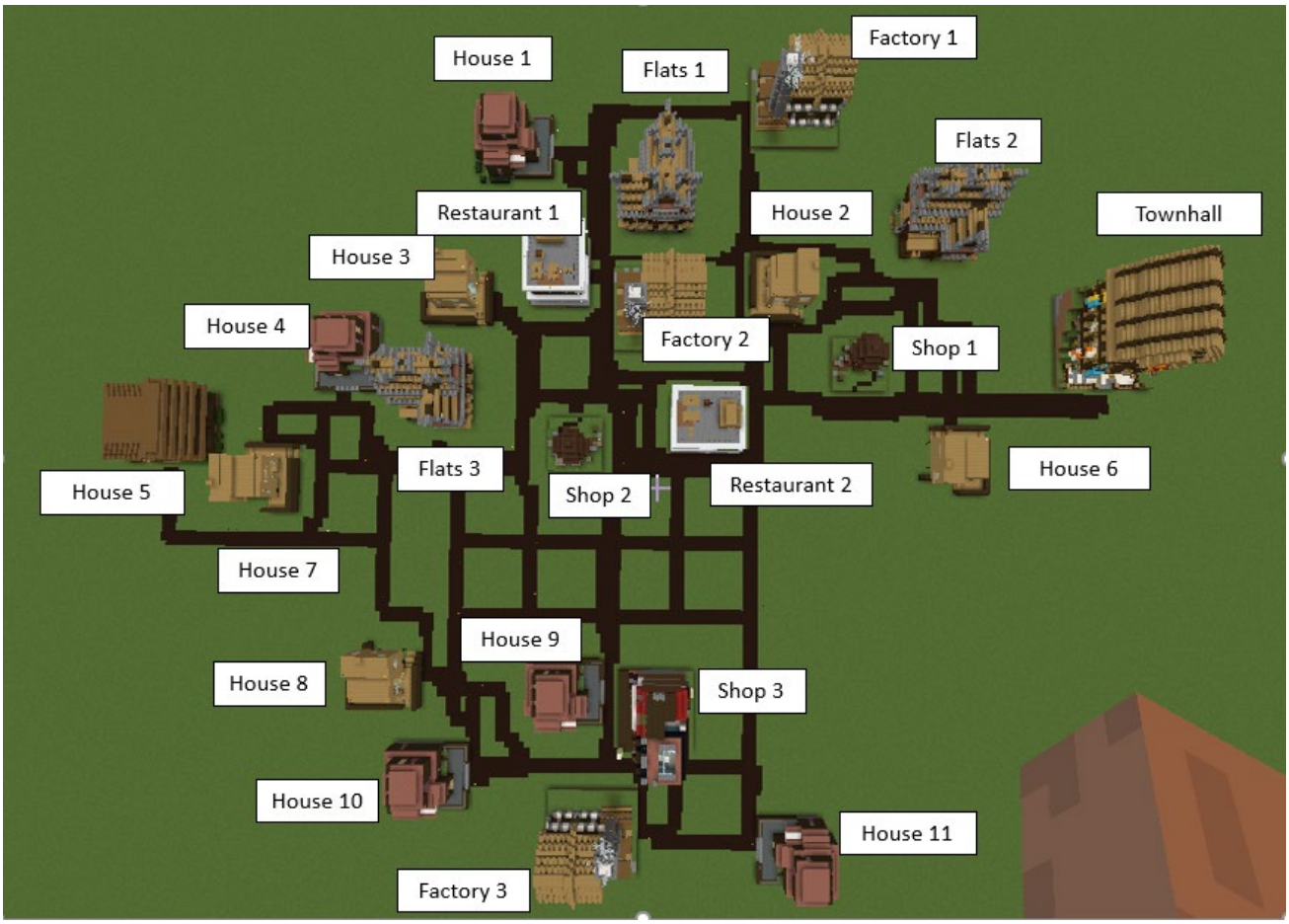



FIGURE 1 – BIRDS-EYE VIEW OF TOWN CREATED IN THE PROJECT.

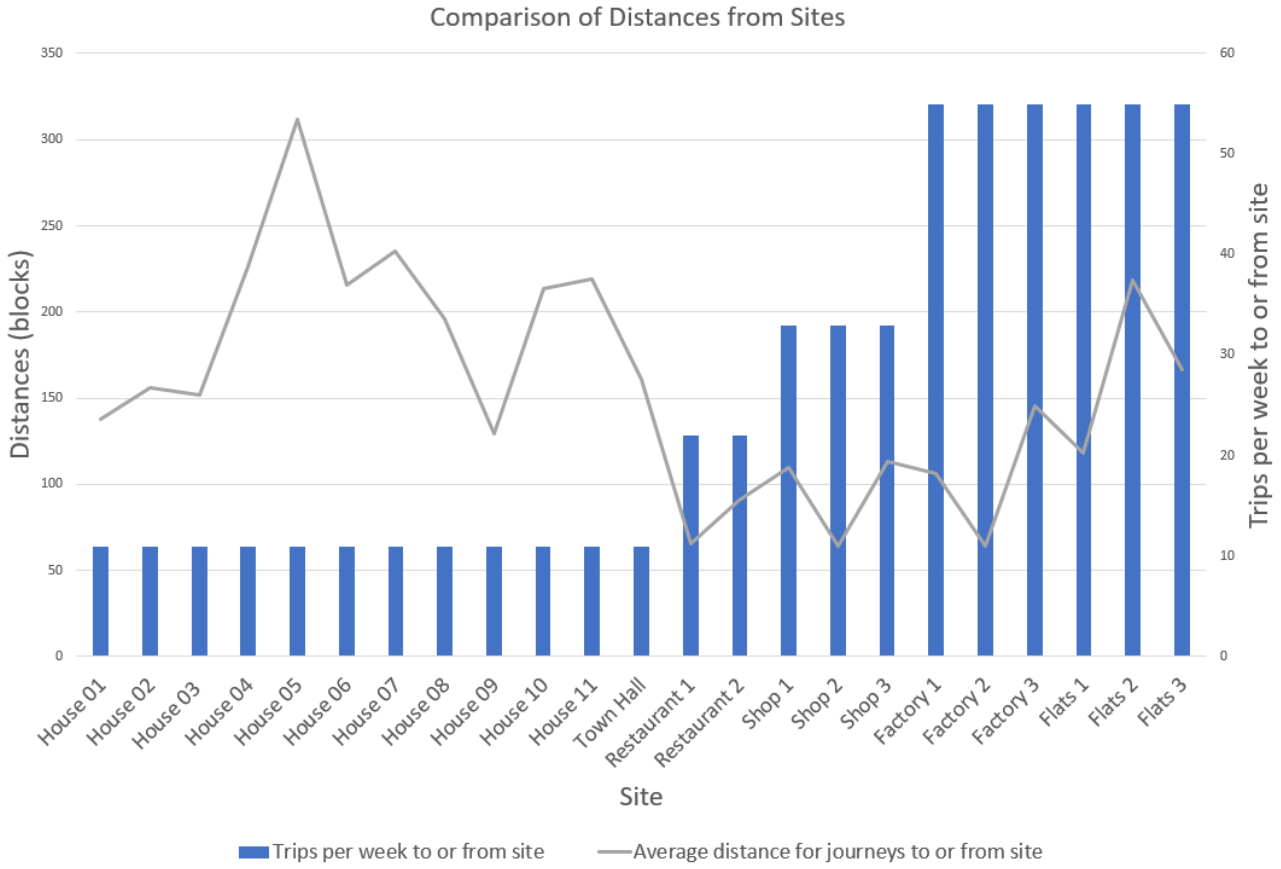
Figure 1 shows a birds-eye view of a town created during the project with the buildings labelled outside Minecraft.

There is a road from every building to every other building. The roads have successfully adhered to a realistic Manhattan-style grid for the most part.

The public buildings have been assigned to the sites where the least amount of travel is necessary to reach them from the residential buildings. Generally, the residential sites were on the map's outer parts, and the often-visited public buildings were more central.

Another factor that affected the positioning of buildings was the building site size. Aaron provided me with the location and the size of the site, making some sites nonviable for some of the more significant buildings.

## Graph




## Analysis

The above graph explores the distances from each site in a town created with a Monte Carlo Tree Search algorithm. The town was designed to minimise the distance travelled by the town's inhabitants. The town is pictured in Figure 1 on the previous page.

The inhabitants are assumed to visit each public building at a set frequency, as shown below.

| Building | Frequency (per week) |
|---|---|
| **Town Hall** | 1 |
| **Restaurant** | 2 |
| **Shop** | 3 |
| **Factory** | 5 |

The blue bars show how many journeys start or end at each site (with the axis on the right), which is a measure of how busy each site is. Both the densely populated flats and the public buildings, with the exception of the Town Hall, were much busier.

The green line shows the average distance of the journeys to or from each site. The journeys were mostly longer for the less busy sites and shorter for the busier sites. The algorithm clearly assigned less favourable locations to those buildings less used. These buildings tended to be at the edge of the map.

However, there is a divergence from this generalisation for some locations. For example, some houses had shorter journeys, and *Flats 2* and *3 had longer journeys*. There were two possible explanations. First, some building sites were not large enough for the more significant buildings like flats, which could have prevented their locations from being optimal. Also, it might not have been physically possible to design the perfect town with the locations provided.

# Conclusion

Overall, the project has gone well, and the main objectives have been met.

***Roads will be built with minimum lengths while maintaining a realistic style.***

The roads found with the A* search algorithm were, generally, as short as practical. The road's lengths were similar to the shortest possible distance to the destination (i.e., the Manhattan distance) while allowing for obstacles.

Roads were also reasonably natural-looking and coordinated (see *Figure 1* on page 4). They were generally on a grid and did not meander directly across the landscape, intertangled with one another. There were, however, a few areas where small squares were created unnecessarily. For example, see next to *Shop 1* in *Figure 1*.

***The town facilities will be placed in the locations which are most accessible by inhabitants.***

The Breadth-First Search (BFS) algorithm provided the optimal solution for smaller towns with fewer locations. We can be sure that the distance travelled by the inhabitants was a minimum because every possible combination had been checked.

But for larger towns with more than 12 sites, the incomplete Monte Carlo Tree Search (MCTS) algorithm was used. This greatly improved the processing speed, but we cannot guarantee the best solution.

I compared the results from BFS and MCTS and found that MCTS always found a result equal to or very close to BFS for towns with 12 or fewer sites. Larger towns were not checked due to BFS processing times. I am confident that the MCTS renders acceptable results and found building locations well.

The processing time for 12 and fewer sites with BFS was much lower than above 12. For example, 12 sites might take up to ten minutes on my laptop, but 13 sites took nearly four hours. This cannot be explained by examining the processing complexity of the calculations alone. Instead, I suspect that some threshold to the underlying workings of Python 3 had been met, perhaps memory management, which vastly increased the processing time. If I were to do the project again, I might use another programming language with the hope that 20 sites were manageable.

I would recommend that anyone undertaking a similar project limit their ambitions on town size and use BFS, or only use MCTS.

***All buildings will also be aesthetically consistent and resemble the local environment.***

Aaron Fletcher's project produces a Minecraft block that reflects the local environment (see his project for details). The construction plans for the buildings are taken from GrabCraft.com. One or more of the components in the structures were replaced with the block provided by Aaron's project. This created a town that reflects the local environment, and the buildings all have a common theme. See the images on the front cover for an example where all the buildings used a dark brown block called *minecraft:dark_oak_wood*.