AgentCraft: An Agent-Based Minecraft Settlement Generator

Ari Iramanesh, Max Kreminski
University of California, Santa Cruz
{airanman, mkremins}@ucsc.edu

Abstract

AgentCraft is an entry for the Generative Design in Minecraft (GDMC) AI Settlement Generation Competition, including the optional Chronicle Generation challenge. It makes use of an agent-based simulation to organically produce a plausible settlement and a written history of the settlement's development. Uniquely, AgentCraft utilizes the HTTP Server version of the competition framework to show the agents constructing the settlement in real time—a visual technique that can't be achieved with the earlier MCEdit framework. In this paper, we aim to provide a point of reference for future agent-based settlement generators by describing how our competition entry works and discussing the benefits and downsides of the agent-based approach. Additionally, we propose a new optional challenge for the GDMC competition, centering on the development of settlement simulations whose inhabitants can be directly observed or interacted with by the player.

Introduction

This paper presents AgentCraft, a Minecraft settlement generator that uses agent-based simulation to create a plausible settlement with adaptations for local terrain in a bottom-up way. It was created as an entry to the Generative Design in Minecraft (GDMC) AI Settlement Generation Challenge (Salge et al. 2020), which scores entries based on four high-level criteria: Adaptability, Functionality, Narrative, and Aesthetics. In the 2021 iteration of the competition, AgentCraft took second place out of 20 total entries and earned the second-highest Narrative score of any GDMC entry to date (just behind the overall winner in 2021.)

Though not the only competition entry to use an agent-based approach, AgentCraft is (to the best of our knowledge) unique in its incorporation of lightweight social simulation elements, loosely inspired by the social simulation framework Talk of the Town (Ryan and Mateas 2019). Furthermore, AgentCraft goes beyond the scope of GDMC by rendering the settlement's development (including the movement and interaction of the agents that created the settlement) in real time, allowing the player to watch the settlement develop before their eyes.

Figure 1: An example settlement generated by AgentCraft, showcasing a variety of buildings connected by roads and built from appropriate materials for the local environment.

Our goals in this writeup are twofold. First, we aim to provide a reference description of an agent-based Minecraft settlement generator that can serve as an inspiration and jumping-off point for future agent-based competition entries. To this end, we attempt to document our approach more thoroughly than other competition entries have done to date and provide a brief accounting of significant difficulties we encountered in the process of developing our generator. Additionally, we provide links to a GitHub repository containing the complete AgentCraft source code[1] and video documentation of the generator in action.[2,3]

Second, beyond the current explicit goals of the GDMC competition, we believe that a compelling new challenge might be found in the live simulation of settlement inhabitants that players can observe or even interact with in real time. To that end, we suggest that the GDMC competition could adopt a new optional "Liveness Challenge" (in the same vein as their current Chronicle Generation challenge) oriented around this goal.

[1] https://github.com/aith/agentcraft
[2] https://www.youtube.com/watch?v=Gc5h54Bus_k
[3] https://www.youtube.com/watch?v=DGLiWOxWLig

Figure 2: An aerial view of the road network in a generated settlement, showing how roads conform to the terrain and connect buildings in an organic-looking way.

Starting the Generator

The AgentCraft generator runs as a CLI, with a few optional flags. Upon starting, the generator searches for a suitable settlement location within the user-specified boundaries, or the *build area*. This location must:

- Be on land
- Have water nearby (otherwise a well will be placed)
- Be traversable, i.e., flat and unobstructed enough to be walked across
- Have suitable locations for a building, a road, and the initial agents
- Be at minimum 32x32 blocks in size

If a valid build location was found, the generator will create a small cast of agents, an initial building, and the first node of a road network. It will then launch an iterative settlement construction process that repeatedly steps through each agent and performs appropriate actions based on the current state of the settlement.

Information about the build area is stored as a grid of nodes overlaid on the surface of the world. Each node is composed of nine surface blocks, arranged in a 3x3 manner. When the generator requires information about a node, it inspects those nine tiles to classify the node as belonging to one of several high-level categories. This information is used in many ways, including for agent-based pathfinding and road network placement.

Roads

Traffic-Based Placement

Real-world settlement traversal and design revolves around roads. Real-world buildings are built along roads, and footpath-based roads are typically built for the purpose of facilitating traversal through a popular route. AgentCraft replicates these features in order to achieve realism of simulation. To accomplish this, each node stores a *prosperity* value that measures how busy that node is. A node's prosperity increases by a task-specific amount when an agent performs an action within that node, and decays over time. After a node reaches a certain prosperity threshold, the generator will decide whether to generate a road from that node. Agents will also favor roads when pathfinding to a destination, so that the agents and their roads work cohesively. This synergism of behavior fulfills the positional realism of roads.

Our prosperity-based approach to road generation is based on the one used in TownSim (Song and Whitehead 2019), which in turn was inspired by a survey of road network patterns present in complex real-world cities (Batty 2007) and earlier work on the generation of naturalistic terrain-adaptive road networks (Emilien et al. 2012). Additional related work on road network generation is surveyed by Smelik et al. (2014) and Kelly and McCabe (2006), as well as in the TownSim paper.

Traversability and Efficacy-Based Placement

Real-world road design aims to satisfy several requirements. Generally, roads are expected to:

1. Minimize the user's travel time to a destination
2. Be connected to pre-existing roads
3. Be traversable
4. Avoid redundancy with other roads

For the purposes of realism, AgentCraft aims to fulfill each of these requirements. Requirements (1) and (2) tend to conflict in practice, since the fastest path to a destination is simply a straight line to the destination, or the path that minimizes the Euclidean distance from the start to the destination. To fulfill both (1) and (2) in a reasonable and realistic-looking manner, when the generator decides to link a particular node to the road network, it first attempts to do so by generating a straight-line path from the target node to the nearest road-type node. However, this often doesn't lead to paths that satisfy Requirement (3): buildings, trees, or steep slopes can intercept these paths.

If the desired straight-line path is not traversable, the generator will first attempt to split this straight line into two connected linear paths to get around an obstacle. If one or both of the resulting split paths are themselves not traversable, the generator will then perform a series of raycasts outward from the target node in a circle to identify a "good enough" plausible path that still connects the target node to the road network. If no traversable path can be found in this manner, the generator gives up on connecting this particular node to the road network for now, though it may try again later once the road network has expanded further.

Traversability checks involve two key components. First, to be traversable, a path must never require an agent to move from one block to another when the change in height between the two blocks is greater than the designated agent jump height (by default, one block.) Second, to be traversable, a path must not move through any nodes that have been classified as non-traversable due to the types of blocks they contain (for instance, water or lava blocks.)

Figure 3: The initial road network can't be extended in a straight line to reach the nearest tree, so we first find a route around the building, extend the road network slightly, and construct a straight-line road to the tree from the new extension.

The combination of these checks and algorithms is computationally more expensive than some simpler road network generation algorithms, but results in an iterative and organic road system that builds on top of itself, effectively replicating real-life road systems—particularly those found in older cities with less-than-modern city-planning origins.

Agents

Agent behavior in AgentCraft is context-free and local to each agent: when deciding what to do next, agents do not consider past experiences or (except in the case of specifically social actions) the state of other agents. The agents have seven different actions, or *motives*, that they choose from based on a set of rules, similar to cellular automata. The behaviors that are necessary to retain agent "health", such as gathering water or getting rest, are decided by non-stochastic rules. The decision to grow or log a tree is stochastic, unless there are no trees nearby.

Here are all of their possible actions, in order of priority:

1. **Socialize**, if another willing agent is in the same node
2. **Rest** in a building, if tired
3. **Drink** from a water source, if thirsty

Figure 4: Several screenshots of agents as they appear in the world, carrying out their business. During the settlement generation process, agents visibly walk from place to place, collect resources, and interact with other agents.

4. **Propagate** to produce a new "child" agent, if lover-lover interaction threshold met with another colocated agent
5. **Build** using shared resources, if enough resources exist
6. **Log** OR **grow** a tree, picking stochastically
7. **Grow** a tree if no trees exist within a fixed radius

Notably, all actions except socializing require movement to a destination. This works synergistically with the prosperity system, which results in roads being generated often and in the places where agents go most (hearkening back to road efficacy.) Agent pathfinding is accomplished via A* search on the two-dimensional grid of nodes that AgentCraft overlays on the surface of the world.

The **socialize** action occurs when two otherwise unoccupied agents are positioned in the same node. There are three possible relationship types for the agents: *lovers*, *friends*, and *enemies*. All of these are mutual relationships, with the relationship between each pair of agents being chosen at random. If an agent has performed enough **socialize** interactions with their lover and neither has any higher-priority actions to perform, they will both pathfind to a building and **propagate** to produce a new "child" agent. **Socialize** interactions between friends and enemies affect the interacting agents' happiness, but in the current version of AgentCraft, happiness has no visible effect on the agents.

Agents are rendered as Minecraft *armor stand* entities, which offer expressive options that allow them to visually represent Minecraft player models. Each agent carries a

motive-specific item in one hand (indicating to the player what action this agent is currently performing) and a randomly chosen agent-specific "favorite item" in the other. An agent's clothing is colored based on a randomly-chosen color representing the settlement as a whole, while their shoes represent the overall prosperity of the settlement (with agents from richer settlements having higher-quality shoes.)

Optimizations

Because the GDMC competition imposes a 10-minute cap on the time that a generator may take to produce a settlement, and because we perform fairly in-depth simulation of agent activity, the implementation of AgentCraft required many unexpected optimizations to be performed. Optimizations were especially needed for large build areas, such as the 1000x1000 build area used as the upper bound for build area size in the 2021 iteration of the GDMC competition. On a 4-Core Thinkpad T480, given a 1000x1000 block build area, the retrieval of world data from the HTTP Server competition framework alone took 5 minutes—leaving only 5 minutes for the generator to produce a plausible settlement. This was not an easy target for us to hit, given that AgentCraft constructs settlements iteratively through the actions of agents over time.

The following optimizations were necessary to make the generator produce a complete and plausible settlement in under 10 minutes:

1. Pre-compute legal A* paths for agents on startup
2. Recompute legal A* paths when a change in a block leads to a change in reachability from surrounding blocks, in a depth-limited fashion
3. Lazy-load block and type information from the HTTP server on a per-node basis
4. Store block data in an internal buffer in the generator, rather than repeatedly requesting it from the Minecraft world
5. Disable mid-simulation rendering of all changed blocks and agents, if desired

Optimization 1 entails calculating inter-traversability between blocks and their neighbors. This guarantees that pathfinding to an unreachable location fails immediately. Without this, A* would iterate through every block to see if a path is attainable; in a 1000x1000 build area, around 1,000,000 surface blocks would have to be searched every time an agent attempted to go to an unreachable location.

Trickily, the reachability of specific blocks has to be recalculated whenever a block's traversability is changed in the world—for instance, when an agent places a building or chops down a tree. Recalculating traversability for every block would be enormously expensive, given how many times each block might be changed during the simulation. For this reason, we implemented **Optimization 2**, which limits recalculation of block traversability to only those blocks whose reachability is known to have changed. Furthermore, we made this computation depth-limited, accounting for the furthest distance that an agent could actually reach from their current location.

Optimization 3 ensures that node data structures (which cache information about specific parts of the build area's surface) are only created when needed for decision-making. Each node stores a fairly large amount of information, so this helps to reduce memory usage during simulation.

Optimization 4 separates the generator's model of the Minecraft world from the live, player-facing version of the Minecraft world after the generator starts up. Unfortunately, this means that a modification to the Minecraft world during the simulation may lead to incoherent results. Because this could potentially compromise the illusion of the settlement as a real, living and responsive place, we would like to find a way for the generator to receive local updates to the actual Minecraft world during simulation; for the time being, however, we leave this to future work.

Optimization 5 is available as a CLI option for AgentCraft users. It simply holds off all simulation-to-world rendering until the generator completes. Live rendering of agent behavior and block updates during simulation is one of AgentCraft's key features, so disabling it runs counter to our intended player experience—but for the time-limited GDMC competition, which does not take live rendering into account during judging, this optimization is essentially mandatory.

Chronicle Generation

GDMC currently offers one optional bonus challenge for settlement generators: the Chronicle Challenge, which involves generating a written history, or chronicle, of the generated settlement. To date, most competition entries have not attempted to address this optional challenge: of the 20 entries to the 2021 iteration of the competition, only three (including AgentCraft) attempted to participate in the Chronicle Challenge.

Since AgentCraft creates a settlement incrementally through agent behavior, producing a chronicle was an easy fit for our approach. In the current version of AgentCraft, whenever an agent performs an action, there is a chance that the action will be described in the chronicle. To add some light variation to the generated text, we choose randomly between two potential sentence templates for each high-level action type; swap agent names into these templates where appropriate; and add random adjectives (drawn from a list) to specific fixed points in most of these sentences. These action descriptions are then written to a Minecraft book in the order that the actions were performed (Fig. 5). In the future, a lightweight text generation tool such as Tracery (Compton, Kybartas, and Mateas 2015) could be used to add procedural variation to these descriptions. Future attempts at extending the chronicle generation components of AgentCraft could also borrow methods from NaNoGenMo, as surveyed by van Stegeren and Theune (2019).

In our original plan for AgentCraft, we intended to generate per-agent diaries that gave separate and sometimes-conflicting accounts of the same events in the settlement's history, inspired loosely by the biased per-character perceptions of narrative in *Why Are We Like This?* (Kreminski et al. 2020a,b). In practice, this proved to be out of scope for our

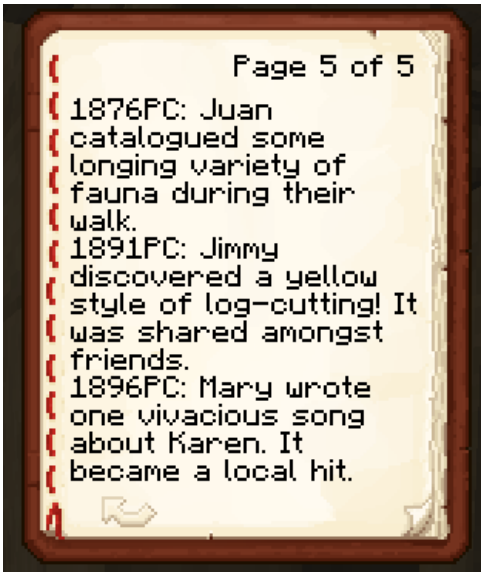

Figure 5: A page in a generated Chronicle describing three various events in time.

Figure 6: A group of generated Chronicles for a settlement procedurally named Verdictland.

entry this year; consequently, we leave per-agent diary generation to future work.

Building Placement

In some ways, using agents to create settlements significantly increases the complexity of settlement generation—which is already a difficult task on its own. For this reason, we chose to strategically limit complexity in other areas, especially in the potentially thorny area of building generation. Rather than procedurally generate new building geometry from scratch, our agents construct buildings based on human-authored schematics. Nonetheless, we do lightly process these schematics to allow some flexibility and variation in which blocks are used to construct them. This entails allowing "air" blocks in schematics to be replaced by blocks already in the world; wood blocks to change based on the type of logs that agents have gathered; and directional blocks (such as staircases) to adjust their directionality based on the rotation of the building. In addition, to make these buildings sit correctly on uneven terrain, we generate a flat scaffolding foundation underneath them.

Liveness: A New Challenge

In the process of developing AgentCraft, we rapidly realized that it was fun to watch the agents go about their business in the world and construct a settlement in real time—maybe even more fun than to walk around the resulting settlement once the agents had completed their task. Others who have seen the agents in action (including the GDMC organizers) seem to agree: people consistently find the liveness of agent behavior compelling.

In light of this reaction, it stands out to us as unusual that, even in building-focused games like Minecraft, non-player characters that attempt to systematically change their environment over time are almost unheard of. In AgentCraft, our simulated agents are not fully realized NPCs, because the player cannot meaningfully interact with them—but in principle, nothing seems to preclude the possibility of further developing AgentCraft-like agents into NPCs that can interact with the player through (at least) trade and combat, much like the villagers of vanilla Minecraft.

With this goal in mind, we propose a new optional challenge for the GDMC competition, centered on *liveness*: the creation of live settlement simulations whose inhabitants can be observed or even interacted with by the player. Like the current Chronicle Generation bonus challenge, a prospective Liveness Challenge might serve as an optional objective that favors generators with a deeper understanding of the artifacts they create—and thereby steers the community as a whole toward better *orchestration* (Liapis et al. 2018) across multiple distinct facets of generative design and game AI.

Conclusion

AgentCraft's success (taking second place overall in the 2021 GDMC competition) indicates that the use of social agents to create a settlement can result in organic design, if done with care. By integrating the simulation of agent needs with a prosperity-based approach to road network generation, AgentCraft generates realistic-seeming road networks that adapt to local terrain and resource availability. Additionally, an agent-based approach to settlement generation makes first-pass chronicle generation relatively easy and provides lots of hooks for deeper chronicle generation going forward, especially if integrated with a slight deepening of social simulation. However, agent-based approaches to settlement generation come at a performance and development cost that grows with simulation scale—necessitating more optimization effort than might otherwise be required and limiting the amount of time and energy that can be spent on other aspects of settlement generation, such as procedural building design.

Nonetheless, we find that generated settlements are made more interesting when you can directly witness the process by which they were constructed—especially when this process involves the live simulation of believable characters. For this reason, we suggest that future iterations of the GDMC competition might want to adopt a new optional bonus challenge centered on the development of live settlement simulations—in which plausible settlements go hand in hand with the plausible NPCs that live inside them.

References

Batty, M. 2007. *Cities and Complexity: Understanding Cities with Cellular Automata, Agent-Based Models, and Fractals*. MIT Press.

Compton, K.; Kybartas, Q.; and Mateas, M. 2015. Tracery: an author-focused generative text tool. In *International Conference on Interactive Digital Storytelling*, 154–161. Springer.

Emilien, A.; Bernhardt, A.; Peytavie, A.; Cani, M.-P.; and Galin, E. 2012. Procedural generation of villages on arbitrary terrains. *The Visual Computer* 28(6): 809–818.

Kelly, G.; and McCabe, H. 2006. A survey of procedural techniques for city generation. *ITB Journal* 14(3): 342–351.

Kreminski, M.; Dickinson, M.; Mateas, M.; and Wardrip-Fruin, N. 2020a. Why Are We Like This?: Exploring writing mechanics for an AI-augmented storytelling game. In *Proceedings of the Electronic Literature Organization Conference*.

Kreminski, M.; Dickinson, M.; Mateas, M.; and Wardrip-Fruin, N. 2020b. Why Are We Like This?: The AI architecture of a co-creative storytelling game. In *Proceedings of the Fifteenth International Conference on the Foundations of Digital Games*.

Liapis, A.; Yannakakis, G. N.; Nelson, M. J.; Preuss, M.; and Bidarra, R. 2018. Orchestrating game generation. *IEEE Transactions on Games* 11(1): 48–68.

Ryan, J.; and Mateas, M. 2019. Simulating character knowledge phenomena in Talk of the Town. In *Game AI Pro 360*, 135–150. CRC Press.

Salge, C.; Green, M. C.; Canaan, R.; Skwarski, F.; Fritsch, R.; Brightmoore, A.; Ye, S.; Cao, C.; and Togelius, J. 2020. The AI Settlement Generation Challenge in Minecraft. *KI-Künstliche Intelligenz* 34(1): 19–31.

Smelik, R. M.; Tutenel, T.; Bidarra, R.; and Benes, B. 2014. A survey on procedural modelling for virtual worlds. *Computer Graphics Forum* 33(6): 31–50.

Song, A.; and Whitehead, J. 2019. TownSim: Agent-based city evolution for naturalistic road network generation. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*.

van Stegeren, J.; and Theune, M. 2019. Narrative generation in the wild: Methods from NaNoGenMo. In *Proceedings of the Second Workshop on Storytelling*, 65–74. Association for Computational Linguistics.