Space segmentation and multiple autonomous agents: a Minecraft settlement generator

University of Southern Denmark, Odense.
Sebastian S. Christiansen, *sebac17@student.sdu.dk*

Abstract - This paper describes and illustrates a submission to the generative design in Minecraft (GDMC) competition. It introduces a novel approach to traversability partitioning of three-dimensional game space; The imperfect space segmentation, and introduces a novel implementation of a two-system brain model for autonomous agent simulation. Imperfect space segmentation is used in conjunction with a grid-wise segmentation to produce a contextual representation of the game space. This is input for the settlement generation using autonomous agents, where each agent is controlled by their system 1 impulses, and their system 2 reasoning-action brain model. The two-system brain model is novel to autonomous agent simulation and is described both theoretically and by its implementation. The imperfect space segmentation is described both theoretically and by its runtime.
Keywords: *Generative design, AI, space segmentation, partitioning, autonomous agent, two-system brain model, simulation.*

1 Introduction

This paper introduces a combined set of space segmentation algorithms and simulated set of autonomous agents as a submission for the Generative Design in Minecraft (GDMC) Competition.

The focal point of the GDMC competition is to write Artificial Intelligence (AI) implementations to run in a Minecraft map editor software called McEdit. The McEdit software supports *filters* which are Python scripts, implementing a *perform()* function as the entry point for functionality.

GDMC entries are graded based upon four categories:

- Adaptability,
- Functionality,
- Narrative, and
- Aesthetics

While the solution proposed in this paper focuses on only the first two categories of grading; adaptability and functionality, it does so via meaningful space segmentation and an approach to simulated autonomous agents. The grading category of *narrative* remains largely implicit in the settlement by; placement of the settlement and shelter clumping. Though implicit in the generated settlements, a pseudo-narrative can be extracted from every agent at any point during simulation.

The fourth grading category of Aesthetics is dependent upon the placement, variety, and design of structures in a settlement. While both narrative and aesthetics may become properties of the generated settlement, these were not the primary focus of the project.

The goal of the GDMC competition is for AI to eventually rival the settlement generation capabilities and believability of humans. Furthermore all implementations must finish settlement generation within 10 minutes for a 256x256 block region.

This paper will answer the following research question by use of theory, technological descriptions and examples.

RQ: "How can a sequence of space segmentation algorithms be used as input for multiple autonomous agents to generate a Minecraft settlement with realistic properties inherent to its location?"

2 Related works

The paper [2] proposes a multi-agent simulation approach to settlement generation which proves to lend itself well, to fulfilling the third category of grading - Narrative - as each agent lives a simulated life in real time while cooperatively generating a settlement. Each simulated agent can generate logs of actions and motivations which can then be converted into a human-readable format and included as part of the settlement. Although in little detail the paper briefly describes a grid-based segmentation algorithm of the walkable space to determine suitable plots of land on which to place houses.

In the field of robotics the imperfect space segmentation approach is referred to as 'traversal segmentation'. In the paper [1] the authors introduce a snake-like robot to navigate obstructed and complex three dimensional environments using traversal segmentation to segment the space into traversable and non-traversable segments based on depth sensing.

In the book *"Thinking, fast and slow"*[3] Daniel

Kahneman describes a two-system representation (simplification) of the human brain; system 1, and system 2. The system 1 brain is described as unconscious and automatic, a system in which the agent (human) has no control, it is influenced by natural desires which makes it impulsive in nature. System 2 represents the opposite and is described as conscious and capable of reasoning, system 2 is the problem solver for the problems identified by system 1. The book further details systems 1 and 2 behaviours more applicable in real-time systems, such as; priming, contextual analysis of events, and causality analysis of events. In real-time systems these behaviours would amplify the power of system 2, as it would allow for further analysis as compared to step-wise simulation of a predetermined environment.

3 Walkable floor detection

With the goal to split the surface of the selected area in the Minecraft world into segments, the first action must be to identify the surface. A surface block is defined as; any block in three dimensional space without another block above it. Exceptions must be made to this rule, as every visible entity, including; grass, flowers, water and lava are all defined as blocks. As shown in figure 1, the definition of a block becomes problematic as decorative grass blocks would constitute a surface block leaving the underlying grass block not included in the set of walkable blocks. The definition of a floor block thus becomes; a visibly solid block, on which the player can stand, above which decorative blocks may reside. The definition of decorative blocks is any block with which the player has no game state altering interaction or collision.

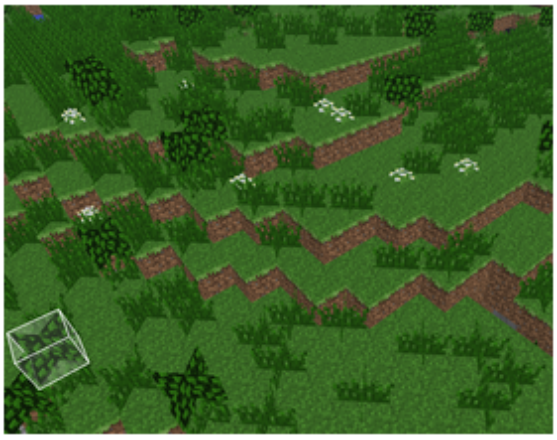

Figure 1: Obstructions found on the Minecraft world surface

The process of finding walkable floor blocks is to iterate over all integer coordinates in the selected box region of the world. Firstly iterating over x and z axis coordinates from lowest to highest, then for each of those sets, iterating over the y-axis coordinates from highest to lowest, breaking when a floor block is discovered.

Run-time can be lowered significantly by calling *break* when a block is discovered. The consequence of this action is that overlapping space is not handled, thus multi-story walkable floor space is not possible. One can imagine an overhanging cliff in the Minecraft world, naturally both the surface of the overhang and the surface over which the cliff hangs would be considered surface blocks. However this implementation limits the discoverability of surface blocks, in order to remain within the competitions ten minute runtime limit, while spending time on more important tasks. Identifying surface blocks does not constitute an important task, as any realism breaking classification would be the result of a rare edge case. Likewise much is to be gained from one floor block per (x,z) coordinate combination within the selected box, as this ensures that caves, mine-shafts, and tunnels are not explored for floor blocks. These naturally occurring features of the Minecraft world are undesirable for settlement generation, thus the floor blocks within are of no importance.

4 Imperfect space segmentation

Segmentation algorithms are used throughout data-science to meaningfully differentiate between classifications of data points. While blocks in Minecraft contain classifications via *blockIDs* this singular property does not capture the class property of a set of blocks. A dirt block and a grass block adjoined constitute ground blocks (floor blocks) though the blocks contain differing blockIDs.

When implemented efficiently, space segmentation algorithms in general can;

- Produce contextual understanding of the in-game world for agents,
- Perform the human task of categorisation and grouping in-game spaces, and
- Enable dynamic behaviours within the classification regions

As described in the related works section, traversal segmentation is already applied in the studies of robotics, but the principle could easily apply to segmentation of three-dimensional game space. The technique with which the segmentation is achieved in the

robotics studies does not apply to the game space, as depth sensing cannot adequately describe traversable space in Minecraft.

The move-set of a player character in Minecraft is assumed by the segmentation agent, starting from an origin block, it will then traverse all legal-step neighbours of the origin block. Neighbours will be explored when no more neighbours are found from the current block, as they are added to the search space. The search space thus gradually increases from including only the origin point, to including all legal neighbours of the origin, to the neighbours of neighbours. McEdit applies the filter in a square region which is manually selected, thus when initialising the segmentation at the lowest point in the two dimensional representation of floor blocks in the square region, the search space never exceeds:

$$\sqrt{w^2 + d^2}$$

Where w is the width of the region, and d the depth of the region. This follows the Pythagorean theorem for right triangles. Figure 2 illustrates the search space at a given time for a square region, the green pixel shows the origin block, the red shows the last block identified. The Pythagorean properties of the search space are self evident, as the search space is ensured to be rectangular for all selections. As this proposed algorithm assumes the lowest possible point as the origin of search and the player move-set as the mechanism of traversal the search space will be traversed linearly, with a maximal open set of non-visited blocks as described earlier.

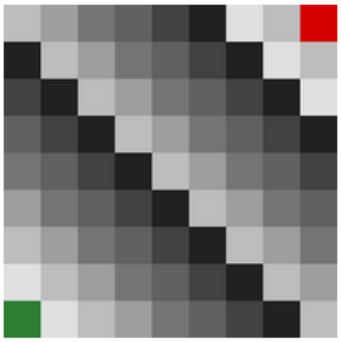

Figure 2: Search space illustration

A set of traversable blocks using the player move-set defines a segment. While not illustrated above, imperfections in the Minecraft world space stems from the stringent definition of floor blocks as described in the prior section. Floor blocks are simply blocks with no block above it, thus blocks under water are not defined as floor blocks. Traversable segments are thus the set of connected floor blocks, where each floor block included in a section is removed from the search space. It is thus ensured that a floor block belongs to exactly one segment, with a segment possibly consisting of a singular block. Figure 3 displays a selected region from an arbitrary Minecraft world, segmented by Imperfect space segmentation. Each segment is shown using wool blocks, one colour is used per segment.

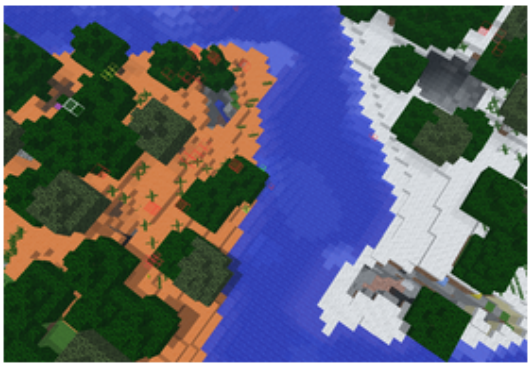

Figure 3: Imperfect space segmentation on region with two landmasses

The result of Imperfect space segmentation is a shallow tree structure, the root of which being the floor block list. Only leafs are added onto the root, each leaf contains a fully traversable set of walkable floor blocks. The advantage of this tree structure is, to allow for fast search and retrieval of the largest fully traversable set of floor blocks, called a segment. The largest segment is used throughout the remainder of both segmentation and simulation, as this segment has the greatest potential for a vast settlement to be generated.

5 Grid segmentation

Inspired by the briefly introduced idea of grid-wise segmentation in the AgentCraft paper [2], the Square region segmentation segments a specific set of blocks in three-dimensional space into square regions of 5x5 blocks. The prior step in the segmentation flow resulted in a shallow tree structure with walkable segments of the root node as leafs. Grid-wise segmentation utilises the work of the imperfect space segmentation, with the largest segment in the imperfect space as input, it produces a grid representation of the three-dimensional game space. Each grid cell consists of 5x5 floor blocks found within the largest segment of the selected in-game space.

The search space for this segmentation agent is the provided set of floor blocks in two dimensional space, where each block is matched to a predefined definition

of a square. The agent will overlay the square template at the block location to compute a set of blocks that must be found in the search space for the square to be valid. If all blocks within the template overlay are found in the search space, then a complete square segment has been found. All blocks found within a square template of a valid square region are removed from the search space to avoid overlap. Likewise, a block in the search space which did not lead to a valid square is removed from the search space, as it cannot be part of another square while not resulting in a valid square region itself. To ensure that the former assumption remains true, each iteration of the search space matches the square template to the lowest point in the (x,z) search space.

Figure 4 below shows the resulting grid, overlaid the region from which it was created, each colour indicates a separate segment. Notice the empty areas in the region, these show the blocks removed from the search space as they led to no valid square region. Any immovable object like trees, or missing blocks as a result of holes in the ground, will result in invalid square regions if found within the template matching area.

Figure 4: Grid-wise segmentation of an island

The agent returns a set of grid segment objects, where each segment object contains its blocks, a produced heightmap, and an integer value equal to the deviations of its blocks around its center. The runtime of this agent is O(n) as the entire search space is to be matched with the square template, but with the assurance that every block is checked once. A checked block is either included in a grid square or removed from the search space entirely. In practice the runtime is less than O(n) directly proportional to lack of obstructions in the space. Each valid square region removes all blocks within, thus one block can lead to a search space reduction of 25 blocks. The more square regions are found, the less runtime is required. Equally a complex and obstructed space will potentially require the full runtime of O(n) if no valid squares can be found.

While grid segmentation produces a useful board game-like set of cells, the static nature of 5x5 cell identification leads many blocks to being discarded and removed from the search space. An optimal approach would involve identification of optimal shapes within the search space (as opposed to squares), this appears a scarcely researched problem, though seemingly an NP-hard problem.

6 Agents

To simulate a 'realistic' settlement generation, one must introduce agents (settlers) to act upon the environment; place housing, build farms and reproduce. The actions and movements of settlers is to simulate realistic behaviour of real-world settlers and if randomness is introduced into the movement of settlers, the settlements will be unique for each simulation.

If the initial state of all settlers is equal one must expect equal actions due to the deterministic nature of impulses. Impulses are to be understood as a set of one-word representations of needs of the settler, some needs will take precedence over others - hunger being the primary need, followed by shelter, rest and reproduction. In most cases humans (after which the settlers are modelled) tend to reproduce when their basic needs are met, though in some cases they reproduce in the hopes that their children can help them meet their basic needs. The second case is outside the scope of this implementation and is therefore not explored.

Settler behaviours can be modelled as a set of impulses by system 1, with reasoning and interpretation by system 2. This model leads to a reasoning-action pattern internal to the settler agent. To efficiently generate realistic settlements with settler agents, it only makes sense to perform a multi-agent simulation. For this purpose, n settlers will act upon the environment with a lifetime of s steps. The importance of grid segmentation upon the imperfect space becomes important here, as settlers must know the bounds of the settlement area; the largest imperfect space segment. The settler must also understand the land as a grid of visit-able locations on which to perform actions. The environment of the settler is thus perceived through a discrete set of fully observable grid segments. Each grid segment will be dynamic as other settlers can act upon the state of a grid segment to build housing upon a grid segment.

Figure 5 illustrates how each settler receives im-

pulses from system 1 and acts upon those impulses with system 2. Naturally the settler will have no influence over which impulses system 1 gives as input to system 2.

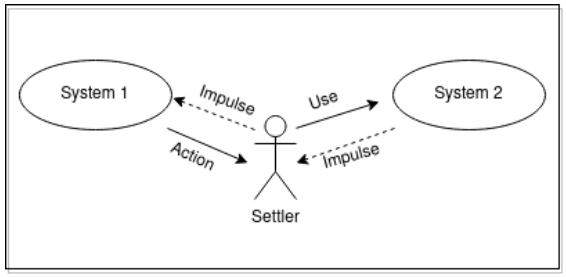

Figure 5: The two-system brain inputs and outputs

Figure 6 shows the decision tree within system 2, this is used to translate impulses into action. The system 2 is in control of the settler, it also differentiates between actions and reasoning (as described earlier). Reasoning in system 2 consists of optimisations, and comparisons; when a settler places a house the system 2 checks (thinks) if the grid cell is already occupied, and if it is, it finds an alternative cell on which to place a house. The thought of finding an alternative grid cell on which to place a house then results in an action of the settler to move to that cell.

System 1 outputs a dictionary of weighted impulses which is input for system 2. System 2 will attempt to understand the impulses by finding the highest priority impulse from the weighted impulses and find a way to reach the goal of the impulse. As impulses are merely a one-word representation of the needs of the settler, system 2 - as the system for reasoning - can understand impulses and compute a way to translate an impulse into an action or a set of actions. System 2 must be able to command the settler to action, the settler will remain ignorant to the reasoning behind the action but will execute the action without the option to resist or counter.

6.1 Simple example

As an example assume system 1 has produced the following dictionary of impulses:

{HUNGER: 0.98, SLEEP: 0.03, SHELTER: 0.02, CHILDREN: 0.0}

System 2 will evaluate the received impulses although ordered in this case, this is not guaranteed. The evaluation of impulses results in one impulse on which to execute for the current step, here the impulse of *HUNGER* is chosen. The *HUNGER* impulse is associated with hunting, to accomplish this task system 2 will identify a suitable grid location on which to hunt and instruct the settler to move to that cell. Currently both successes and failures of the hunting action of a settler is determined by a random variable, with a 0.5 probability of success.

Figure 6 depicts the decision tree of a settler, this tree is an idealised representation for the purpose of clarity. As settlers are a prototype implementation, some behaviours depicted in the decision tree have not been implemented.

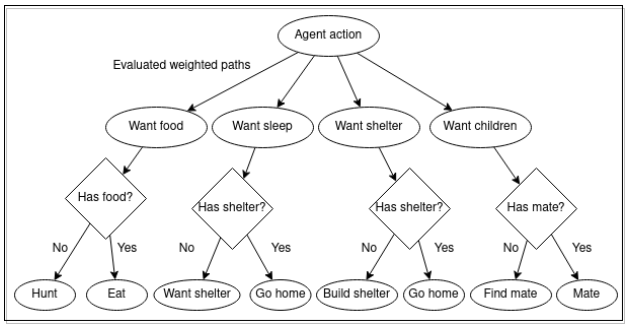

Figure 6: Decision tree for impulses and their resulting actions

6.2 Initial conditions

Once a settlement is instantiated, it instantiates n settlers placed on the center cell of the grid (as described in Section 5). Fertility of a settler is derived from a random variable with a normal distribution, the mean of which is 1.9, with a standard deviation of 0.5. Likewise the standard deviation of the random variable representing the number of children produced, is also derived from a normally distributed random variable with a mean of 0.8 and a standard deviation of 0.25.

The reasoning for using a random variable for both the fertility and the standard deviation of the birth distribution is two-fold. Firstly, the randomness of both properties simulate the randomness of humans, this leads to a more natural simulation result. Secondly this introduces further randomness into the settlement generation, while not inherently a preferable property, it minimises the predictability of each settlement.

The initial set of settlers will be simulated for 25 steps, or until they have children and die. The children of settlers will be simulated for only 90% of their parents full simulation steps. This is to disable an otherwise potentially infinite simulation.

6.3 Building shelter

A settler can build a shelter (5x5x5 blue cube) on any grid cell if the grid is not already claimed for a house

by another settler in the settlement. Grid cells can be considered plots of land, as they have been identified to contain a complete square of floor blocks, they can thus contain a simple shelter for a settler. The current implementation of the system 2 of settlers does not evaluate a plot of land in terms of proximity to a water source or food source. If a plot of land is free, then a settler may claim it.

6.4 Collaboration

Though minimal, collaboration is enabled through the use of a specific *settlement* class, which instantiates the initial set of settlers and is used to simulate the settlement generation through step-wise simulation of the settlers of the settlement. The settlement class handles; creation and removal of settlers, children of settlers, houses built and plots claimed, roads and paving them. With a dedicated settlement class for simulation and for back-reference by settlers, it becomes trivial to simulate the settlement with a dynamic number of settlers. As settlers die after s steps and can have a random number of children with other settlers it becomes important to manage the life-cycle of settlers, simulating only those still alive, while maintaining references to houses already built. As houses belong to the settlement while only being represented as a boolean flag within a settler (has_shelter)

6.5 Reproduction

Settlers can reproduce as a response to an impulse generated when the basic needs of the settler is met. If the another settler has a shelter they are then considered suitable mates. Having a shelter represents having basic needs met if only temporarily. For each suitable mate identified from the settlement, the initiating settler has a 0.5 probability of consent, after which the settlement instantiates a set of new settlers. The number of new settlers is based upon a random variable with a normal distribution and a mean of 1.8, and a standard deviation of 1.2. In practice this results in a settlement population averaging a decreasing population. The initiating settler and the other settler are both removed from the settlement after reproducing. This is due to the limited implementation of settler behaviour, once a settler has reproduced there will be no more new actions to take. It is therefore more convenient to remove the settler and save computational time. New settlers are born mature and thus initiate impulse - action behaviour immediately. One could introduce limitations upon impulses based on the lifetime of the settler, though this has not been done in this project. The simulation is ended whenever no more settlers are alive.

6.6 Decisions as a tree

Though not explored in detail, all settlers store a linked list of *decisions* where each decision contains; a human-readable text representation of the decision, the impulse upon which system 2 acted and the weighted set of impulses. As described prior, system 1 produces a set of weighted impulses as input for system 2. These are the impulses of a decision. The text representation of the action is produced within system 2, it is to be understood as the reasoning behind an action. The impulse upon which system 2 acts is the highest weighted impulse of the set of impulses.

Once a settler has lived out their life, one can generate a narrative of their life by simply traversing the linked list of decisions made. Each decision will lead to another until no more are to be found. The *Decisions* class contains the root decision, which is their birth, from which it can traverse the linked list. The examples below shows the printable decisions of a settler, with more work put into this implementation one could generate a complete and human-readable narrative of the life of a settler.

Each settler can be programmed to output the full linked list of decisions at any stage, the following two examples are produced when settlers have been fully simulated.

Example 1: "I'm alive! -> Went to hunt, and found nothing.. -> Went to hunt, and found 1 food! -> Successfully built a shelter -> Had 2 children"

Example 2: "I'm alive! -> Went to hunt, and found nothing.. -> Went to hunt, and found 1 food! -> Successfully built a shelter -> Got no consent from suitable mates -> Had 1 children"

6.7 Limitations

Settlers lack actions to take after reproducing. Furthermore settlers lack internal goals which would motivate individualistic action. Internal goals could be an internal understanding of specific jobs; being a blacksmith, or being a barkeep. Any internal goal would lead to unique housing and the creation of workplaces

7 Results

The following sections will detail the resulting settlements as a result of using the imperfect space segmentation, followed by the grid segmentation which is input for the settlement step-wise simulation of settler agents.

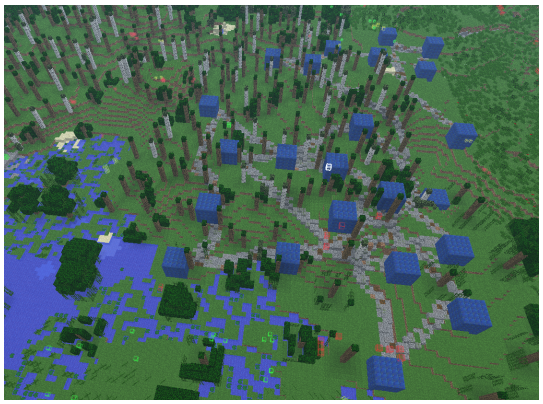

Figure 7: A settlement generated around a lake

Figure 7 shows a settlement generated on a large segment of fully traversable floor. The imperfect space segmentation has correctly identified the boundaries of the traversable floor, and returned the largest land mass. The grid segmentation has produced a set of complete squares (5x5 blocks of floor) upon which settlers move and build. The current demonstration implementation removes all blocks above the floor blocks to better show generated settlement details.

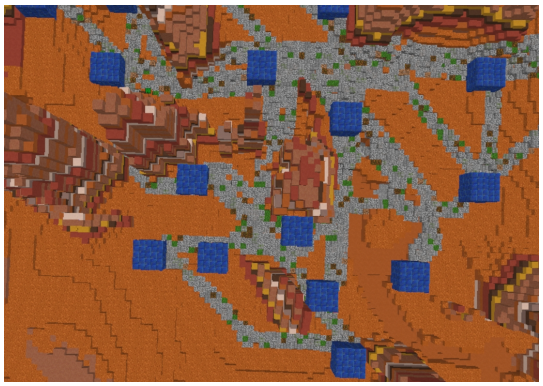

Figure 8: A settlement generated around difficult obstacles

Figure 8 shows another generated settlement using the same approach as on the regular biome as seen in figure 7. The Minecraft mesa biome is an interesting biome on which to generate a settlement as the floor is different to the grass of the regular biome. Furthermore the mesa biome includes canyons and mountainous rises, this results in a difficult terrain in which to traverse.

Figure 9: Example snippet of generated roads between shelters

Figure 9 shows the roads generated between shelters, the blocks for roads are chosen with a random variable taking on a normal distribution around the mean block; Cobblestone. While simplistic, the road network between shelters adds believability and realism to the settlement.

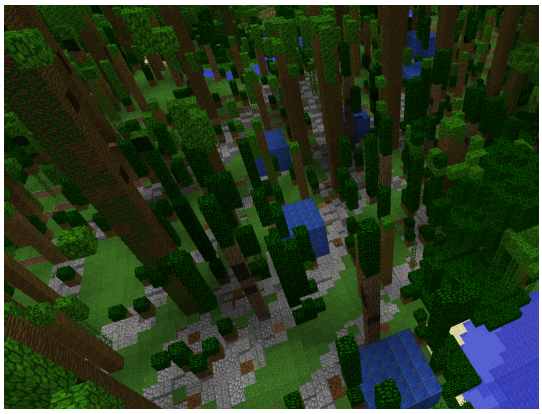

Figure 10: Settlement generated in a jungle biome

Figure 10 shows a settlement generated within a jungle biome, while it looks slightly rough it handles this difficult biome well. The difficulty of the jungle biome is due to the leaves blocks placed upon the ground. As can be seen in the figure, the floor detection works, and both segmentation algorithms help to produce a settlement within this biome. One lacking feature is for settlers to clear the paths between houses for leaves and other decorative blocks. A settlement within a jungle biome is only due to all blocks being

removed above the set of detected floor blocks. While this solution proves helpful for illustrative purposes, it is not an ideal approach.

8 Conclusion

Imperfect space segmentation has proven very useful for identifying traversable block sets. Using the largest block set as input for the grid segmentation provides a useful grid representation of the largest landmass. The usefulness of a grid representation stems from the minimisation of choices of settlers during simulation. Settlers move in unpredictable patterns when searching for food, or when searching for a plot of land on which to place a shelter. When moving between grid cells the possible search space for specific actions associated with cells is greatly decreased.

As floor block detection runs for only seconds even for large selections of an arbitrarily complex Minecraft world and compresses the useful block data from three-dimensions to two-dimensions. The complexity of any further searches is greatly decreased, while still allowing for conversion between two and three dimensional space.

The introduction of a settlement class for simulation proved impressively useful for generating organic settlements, ones in which shelters are scattered but with a clear grouping. Though the system 2 implementation of reasoning-action patterns was not fully explored, the concept seems promising.

Settlements as generated by settlers appear satisfactory, though little has been accomplished in aesthetics. The settlements suit the environment surrounding them well, with adaptability to terrain features. Settlements have so far been functional and fully traversable. Realism of a generated settlement is achieved through the road network connecting buildings and though the placement of shelters in proximity to other shelters with slight branching. The organic nature of both placement of shelters and of road pavements evokes the sense of realism, an interest to find out more about the civilisation.

Imperfect space segmentation as input for grid segmentation leading to simulation of autonomous agents upon the grid space, has generated both realistic and adaptable settlements. Settlements adapt to the inherent properties of their placement; hills and reachable spaces. All paths between shelters are fully functional paths, with shelters placed well within the terrain. Settlement simulation provides an interestingly organic pattern of formation.

9 Future work

The simulated settlements proved interesting, though the implementation described in this paper is limited. It would be interesting to see the possibilities of a full settler implementation, one in which settlers follow pseudo-unique internal goals. Leading to settlers generating workplaces for their professions.

Further work could investigate the possibility of the children of settlers to further the shelter of their parents. Children are born with a reference to their parents including a reference to their birthplace. Settler children could build upon their parents shelter to expand or improve the shelter.

The step-wise simulation of autonomous settler agents allows for the introduction of events into the simulation. Future works could inspect the impacts of simulated disease or natural disasters upon settlers, granted more adaptive settler behaviour was to be implemented.

Future works could inspect implementation improvements for the pseudo-narratives generated by settlers; to record the settlement history in in-game books. The generalised structure of the pseudo-narrative is already in place for all settlers, though not fully explored in this project. The narrative aspect of settlers was introduced to visually inspect and debug settler behaviours, but proved to be more useful than initially thought.

References

[1] Zhao Y. Smith J. Vela P. Chang A., Feng S. Autonomous, monocular, vision-based snake robot navigation and traversal of cluttered environments using rectilinear gait motion. 2019.

[2] Kreminski M. Iramanesh, A. Agentcraft: An agent-based minecraft settlement generator. 2021.

[3] D. Kahneman. *Thinking, fast and slow.* Farrar, Straus and Giroux, 2013.