Space segmentation and multiple autonomous agents: a Minecraft settlement generator

1st Sebastian S. Christiansen
University of Southern Denmark
Odense, Denmark
sebac17@student.sdu.dk

2nd Marco Scirea
Game Development and Learning Technology Lab
University of Southern Denmark
Odense, Denmark
msc@mmmi.sdu.dk

***Abstract*—This paper describes and illustrates a system designed as part of a submission to the Generative Design in Minecraft (GDMC) competition. It introduces an approach to partitioning of a three-dimensional game space novel to the domain of Minecraft settlement generation: traversal segmentation. Moreover, the paper introduces a novel implementation of a two-system brain model for autonomous agent simulation. Traversal segmentation is used in conjunction with a grid-wise segmentation method to produce a contextual representation of the game space. This is used as 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 resulting settlements, generated by settlers upon grid-wise segmentation of the traversable space, boasts the properties; organic settlement evolution, and adaptability to its surrounding terrain, though not realism when compared to settlements procedurally generated by Minecraft.**

***Index Terms*—Generative design, space segmentation, partitioning, autonomous agent, two-system brain model, procedural content generation, simulation.**

I. Introduction

This paper introduces a joint set of space segmentation algorithms and simulated set of autonomous agents as a submission for the Generative Design in Minecraft (GDMC) Competition. Minecraft is a voxel-based video game, in which the world is made up of blocks with which the player interacts. All blocks are placed in three-dimensional space, with a unique identifier for its block type. The focal point of the GDMC competition is to procedurally generate settlements within the game of Minecraft. Algorithms can be applied to a Minecraft world using the McEdit[1] software, which supports Python scripts as *filters*.

GDMC entries are graded based upon four categories: adaptability, functionality, narrative, and aesthetics. The solution proposed in this paper focuses on only the first two categories of grading – adaptability and functionality – via meaningful space segmentations and an approach to simulated autonomous agents. The grading category of *narrative* remains largely implicit through the placement of the settlement and shelter clumping. Though implicit in the generated settlements, our system proposes a pseudo-narrative that can be extracted from every agent at any point during simulation.

[1]https://github.com/mcedit/mcedit

The goal of the GDMC competition is for AI (Artificial Intelligence) 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?"

II. Related works

This section discusses theories used within the described system and examples of similar research.

A. AgentCraft

The paper *"AgentCraft: An Agent-Based Minecraft Settlement Generator"* [1] 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 be included as part of the settlement. A similar implementation of agent Chronicles is utilised for the implementation of settlers in this project, though settler logs for this project contain every action with its reasons for all settlers, as opposed to only occasionally entering agent actions into the town chronicle as done for AgentCraft.

B. Traversal segmentation

In the paper *"Autonomous, Monocular, Vision-Based Snake Robot Navigation and Traversal of Cluttered Environments using Rectilinear Gait Motion"* [4], 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. The concept of traversability segmentation as described in the paper is directly applied to the target domain of Minecraft, where the segmentation agent traverses the Minecraft world assuming the movement set of a human

player and segments the game world into sections of cohesive and traversable segments.

C. Procedural content generation

In the book *"Artificial and computational intelligence in games"* [7], the authors describe the goal of Procedural content generation (PCG); the ability of systems to generate quality content at multiple levels of granularity, while taking game design constraints into account. While the authors describe numerous challenges of PCG, one of which will be addressed by this project; the challenge of search space construction. This challenge is defined as the problem of defining the space within which content is to be generated. The search space construction must retain features of the underlying game space with representations of reasonable structures. Both traversal segmentation [4] and the grid segmentation briefly introduced in AgentCraft [1] may lead to a reasonable representation of the search space for PCG.

D. Agent brain model

In the book *"Thinking, fast and slow"* [2], 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 system 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, and would allow for further analysis as compared to simplified stepwise simulation upon a predetermined environment, as done in this project.

E. Multi-agent systems

In their book *"Handbook of Knowledge Representation"* [8, Chapter 24], the authors describe Multi-Agent Systems (MAS) as a solution for knowledge representation, with agents (usually software agents) reasoning about the environment. Agents must be capable of reasoning about other agents in the multi-agent system wherein agents may share a common purpose. The authors further describe cognitive models of rational agents, which entails formalised rational agents with predictable transitions from beliefs and desires to actions. This idea of cognitive models for rational agents fits perfectly with the two system brain model described by Kahnemann [2], wherein system 1 produces impulses (desires) which are translated to actionable steps by system 2 (actions).

F. Decision trees

Finally, this project includes the use of a decision tree, the book *"Data Mining and Knowledge Discovery Handbook"* [5, Chapter 9] describes decision trees as a classifier, expressed as a recursive partition of the instance space. It is thus a directed tree, with a *"root"* node with no incoming edges, while all subsequent nodes will have incoming edges. All paths through the directed tree will terminate in a *"leaf"* node containing singular values. This is a straightforward method to formalise an agent's decision-making process in game development; in this domain leaves usually represent behaviours rather than values.

III. Implementation

The implementation of this project follows a pattern of sequential operation, upon the underlying game space; Surface detection, Traversal segmentation, grid-wise segmentation, multi-agent simulation. Figure 1 visualises the sequence of operations of the implemented segmentation flow and multi-agent simulation upon the defined game space.

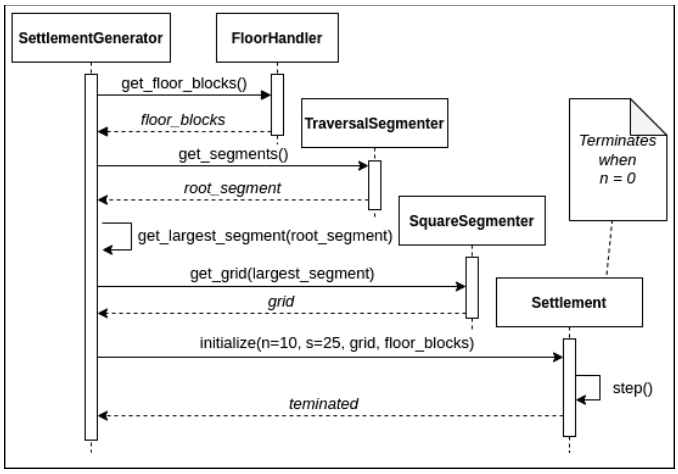

Fig. 1. Sequence diagram depicting the sequence of operations for the implementation described in the coming sections.

Firstly in the flow, we focus on splitting the surface of the selected area in the Minecraft world into segments to identify cohesive areas of walkable surface land. A surface block is defined as *a visibly solid block on which the player can stand, above which decorative blocks may exist*, whereby 'decorative block' we mean any block with which the player has no game-state altering interaction or collision. Figure 2 illustrates the complexity of defining the surface of the Minecraft worlds.

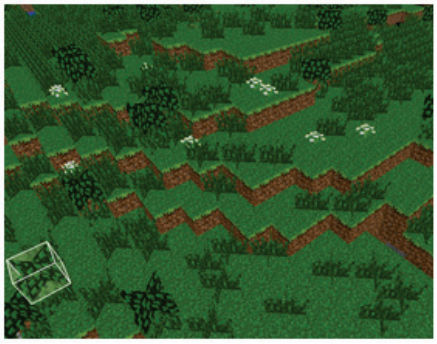

Fig. 2. Typical example of obstructions found on the Minecraft world surface

The process for finding surface blocks is analogous to creating a heightmap of the terrain, while ignoring decorative blocks. This means that the 3D voxel representation of the Minecraft world is reduced to a 2D top-down view. As such, there is a loss of information regarding possible building spaces located underneath overhangs. Since this type of situation is uncommon within most Minecraft biomes (types of environments defined within the game), we believe it's an acceptable loss of information. Moreover, this representation has the double advantage of ignoring possible space within underground caves and lowering the processing time significantly.

The second part of the flow aims to produce a grid of square regions within the largest cohesive surface area. Lastly settlers are created and simulated in a step-wise manner upon the grid of squares.

IV. Traversal segmentation

When implemented efficiently, space segmentation algorithms can:

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

As described in the related works section, traversal segmentation has already been applied in the studies of robotics, but the principle could easily apply to segmentation of three-dimensional game spaces. 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 segmentation agent will assume the move-set of the player and perform a breadth-first search upon the set of floor blocks detected prior. McEdit applies filters 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 3 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, with a constant moveset for the segmentation agent of single-block steps (excluding diagonal steps), using breadth-first exploration of neighbouring blocks. 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.

Fig. 3. Search space illustration

A set of traversable blocks found by 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. Traversable segments consist of connected surface blocks, where each surface block included in a section is removed from the search space. It is thus ensured that a surface block belongs to exactly one segment, with a segment possibly consisting of a singular block. Figure 4 displays a selected region from an arbitrary Minecraft world, segmented by Traversal segmentation. Each segment is shown using coloured blocks, one colour is used per segment.

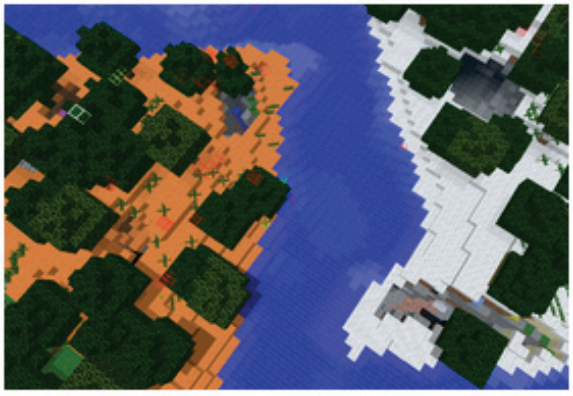

Fig. 4. Traversal segmentation on region with two landmasses, traversable and cohesive segments are coloured individually for illustrative purposes.

The result of Traversal segmentation is a shallow tree, the root of which being the floor block list. Each leaf of the root contains a fully traversable set of walkable floor blocks. The largest traversable segment is used throughout the subsequent segmentation and simulation steps, as this segment has the greatest potential for a vast settlement to be generated.

V. Grid segmentation

Inspired by the briefly introduced idea of grid-wise segmentation in the AgentCraft paper [1], the square region segmentation splits 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 leaves. Grid-wise segmentation utilises the largest segment from the traversal

segmentation process as input, it produces a 5x5 block grid representation of the three-dimensional game space.

The search space for this segmentation agent is the provided set of cohesive surface blocks, 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 segmentation process matches the square template to the lowest remaining point in the largest traversable segment. The result is a list of square segments identified on the largest traversable segment of the selected area.

Figure 5 shows the resulting grid, overlaid the region from which it was created, each colour indicates a separate grid cell.

Fig. 5. Grid-wise segmentation of an island

The runtime of this agent is $O(n)$ as the entire search space needs to be matched with the square template, but with the assurance that every block is only checked once. In practice, the runtime can be lower than $O(n)$ proportionally to the 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.

VI. Agents

To simulate a 'realistic' settlement generation, we introduce agents (settlers) to act upon the environment: place housing, build farms, and reproduce. The actions and movements of settlers are designed to simulate realistic behaviour of real-world settlers.

Settler behaviours is 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 each settler agent. To efficiently generate realistic settlements with settler agents, multi-agent simulation is utilised. For this purpose, n settlers will act upon the environment with a lifetime of s steps. The grid-wise segmentation upon the largest traversable surface segment becomes important here, as settler understand grid squares as visitable locations upon which to perform actions. The environment of the settler is thus perceived through a discrete set of fully observable grid segments. Each grid segment can be seen as dynamic since other agents can act upon its state.

Figure 6 illustrates how each settler receives impulses from system 1 and acts upon those impulses using system 2. Naturally, the settler will have no direct influence over which impulses system 1 gives as input to system 2.

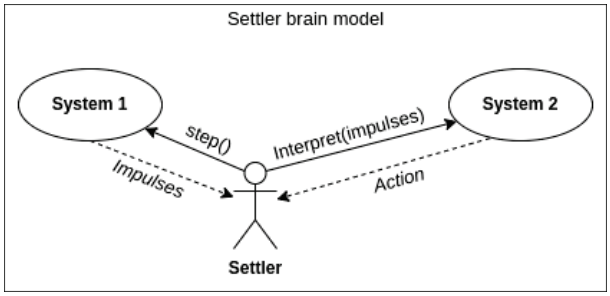

Fig. 6. The two-system brain inputs and outputs

System 1's impulses are expressed as a dictionary of weighted one-word representations of needs of the settler. While initially configured to arbitrary values, these could be configured in accordance with availability of food in the environment. The set of impulses are **hunger**, **sleep**, **shelter**, and **children**. These weights associated to impulses vary through the simulation based on the actions of agents and the passage of time. Moreover, some needs will take precedence over others: hunger being the primary need, followed by shelter, rest, and reproduction.

System 2 represents logic and problem-solving of the settler. It evaluates the impulses from System 1 through a decision tree (see Figure 7), translating the highest weighed impulse into an actionable goal with reasoning consisting of optimisations and comparisons. For example, the impulse 'shelter' leads system 2 to evaluate cells upon which to build a shelter, and then to perform the building action.

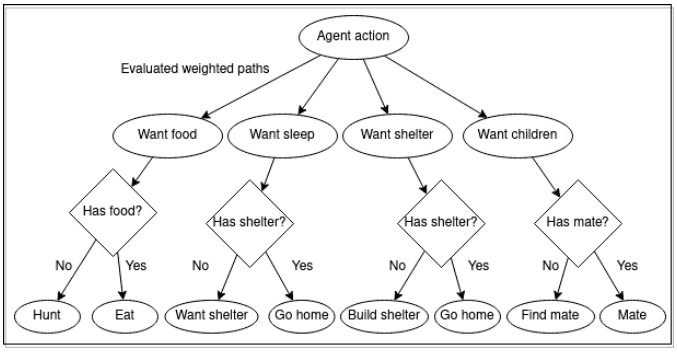

Fig. 7. Decision tree for impulses and their resulting actions

A. Initial conditions

Once a settlement is instantiated, *n* settlers are placed on the centre cell of the grid (see Section V). Fertility of a settler is derived from a random variable with a normal distribution $N(1.9, 0.5)$. Likewise, the standard deviation of the random variable representing the number of children produced, is also derived from a normally distributed $N(0.8, 0.25)$.

There are two main reasons for using a random variable for both the fertility and the standard deviation of the birth distribution: i) the randomness of both properties simulates the randomness of humans, which we believe leads to a more natural simulation result, and ii) it introduces a measure of randomness into the settlement generation which we believe would minimise the predictability of the results.

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

B. Building shelter

A settler can build a shelter (5x5x5 blue cube) on any grid cell if the cell is not already claimed by another settler. Grid cells can be considered plots of land, as they have been identified to contain a complete square of surface blocks, they can thus contain a simple shelter for a settler.

C. Simulation management

The *settlement* class instantiates the initial set of settlers, and is used to simulate the settlement generation through stepwise simulation of all settlers. The settlement class handles; creation and removal of settlers, children of settlers, houses built, plots claimed, and roads. 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 keeping references to houses already built. As houses belong to the settlement while only being represented as a Boolean flag within a settler.

D. Reproduction

Settlers can reproduce as a response to an impulse generated when the basic needs of the settler is met. If 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 a set of new settlers is instantiated. The number of new settlers is based upon a random variable with a normal distribution $N(1.9, 0.5)$, resulting in a gradual decrease of the population. The initiating settler and the mate 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.

E. 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. Once a settler has lived out their life, one can generate a narrative by traversing the linked list of decisions made. The *Decisions* class contains the root decision; their birth, from which one can traverse the linked list. The example below shows the printable decisions of a settler.

Settler life example: "I'm alive! $\rightarrow$ Went to hunt, and found nothing. $\rightarrow$ Went to hunt, and found 1 food! $\rightarrow$ Successfully built a shelter. $\rightarrow$ Got no consent from suitable mates. $\rightarrow$ Had 1 child."

VII. Results

This section describes an analysis of runtime requirements of the system and the results of a quantitative experiment.

A. Experiment

A survey was carried out to evaluate resulting settlements by their adaptability, realism and organic evolution properties. Minecraft settlements (such as are generated by the base game) were used for comparison to the ones generated through our system. Since our current systems focuses on the settlement layout and does not create actual houses, all houses were homogenised to blue cubes.

The experiment includes three different biomes, selected at random from a random Minecraft world as to evaluate the adaptability of the system: Plains, Desert, Savanna. For each biome participants were asked comparative questions regarding an image of a Minecraft village and one from our system. Martinez and Yannakakis [3] suggest that ranking produces more consistent and reliable data when annotating affect information. The order in which in-game and in-place generated settlements were displayed (A and B) was altered throughout the survey. Participants in the survey were asked:

- **To which degree does x conform to the environment?** (x being settlement A then B) 5-point Likert scale

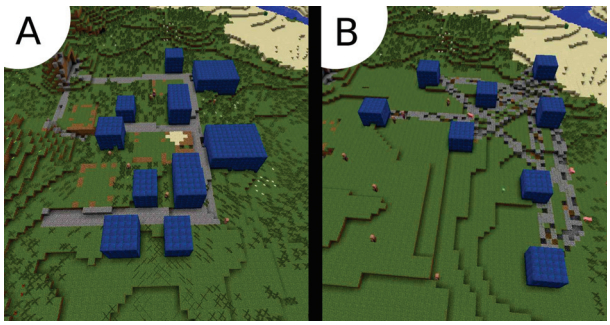

Fig. 8. Leftmost is a homogenised version of a Minecraft settlement generated by the game, while the rightmost image is a settlement generated by the proposed segmentation and stepwise simulation approach.

- **Which of the two settlements above looks like a real city?** *"A"/"B"*
- **Which of the two settlements above could you imagine to have evolved organically over time?** *"A"/"B"*

The following sections discuss the results obtained through the experiment. As a shorthand we will refer to the criteria related to the above questions with the labels: *adaptability*, *realism*, and *organic*. For the analysis of the *adaptability* criterion we provide an additional analysis which only consider definitive answers (i.e., where the participant expresses a clear preference). Under the definite choice constraint, the data becomes Boolean: the answers are either "user preferred the first set-up" or "user preferred the second set-up". To analyse this data a two-tailed binomial test is used, with the null hypothesis that both categories are equally likely to occur and, as we have only two possible outcomes, that probability is 0.5. The Binomial Effect Size Display (BESD) [6] is another way of looking on the effects of treatments by considering the increase of successes through interventions. This is an interesting measure, as it elucidates how much of an effect is created, in our case, by the usage of our generative system.

Figure 8 shows an example of the images from which participants would evaluate the properties of settlements.

1) Demographics

The data collected corresponds to 66 participants, of these 41 were male, 19 were female, 5 reported as "other" gender, and 1 preferred not to say. The participants' age has an average of $\approx$ 23 years (stdev $\approx$ 5.8). Regarding other demographic features, expressed in 5-point Likert scale (0-4), most people self-reported a considerable experience with Minecraft ($avg \approx 3,05, mode = 4$). Regardless of population subdivisions, the results are not significantly different, possibly because of the participants' relative homogeneity.

2) Adaptability

indicate that settler generated settlements were consistently evaluated to be more adaptive to their surrounding environment. Figure 9 displays the survey results of the degree to which each settlement adapts to its environment. We can observe that the participants gave a higher frequency of positive ratings for the generated settlements for the Plains ($32/66$ vs. $22/66$) and Savanna biomes ($41/66$ vs. $21/66$). On the other hand, the difference is less marked for the Desert settlements ($10/66$ vs $11/66$), this may be the result of inherent difficulties of settlement generation upon the terrain of the given biome, for both our system and in-game Minecraft settlements.

TABLE I
PARTICIPANTS' PREFERENCES OVER THE "ADAPTABILITY" CRITERION. ALSO SHOWN ARE THE p-VALUES, CALCULATED USING A TWO-TAILED BINOMIAL TEST, AND THE BINOMIAL EFFECT SIZE DISPLAY. p-VALUES UNDER 0.05 ARE HIGHLIGHTED IN BOLD.

	Minecraft	Generated	Binomial test	BESD
Biome: Plains	24	37	1,24E-01	21.3%
Biome: Desert	15	41	**6,86E-04**	46.4%
Biome: Savanna	16	32	**2,93E-02**	33.3%

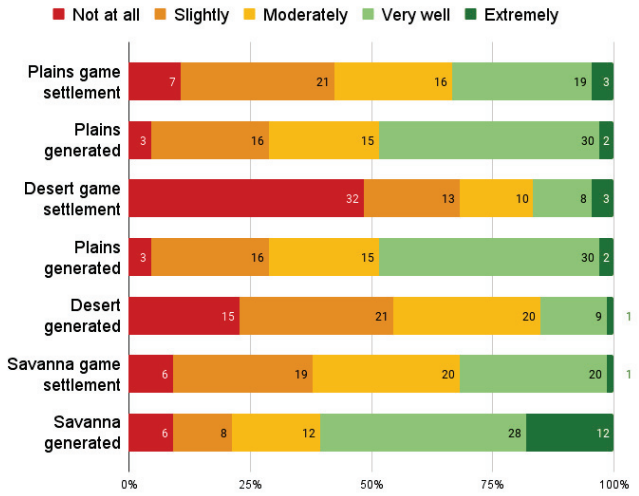

Fig. 9. Evaluated degree to which each settlements conform to their environment, partitioned by biome. Note that 'generated' settlements are the evaluations of our settlements, and 'game' settlements are the evaluations of settlements generated by the game.

We also tried to categorise the answers to this question as a preference: a preference was recognised if the participant had given two distinct ratings to the two images. As can be seen in Table I, a strong statistical significance can be observed for the Desert and Savanna biomes, but not for the Plains biome (although, there remains a preference towards our generator). For those two biomes the null hypothesis can be refuted and a difference in distribution can be inferred between preferring the two generators. This shows how our system is clearly perceived as having better adaptability to the environment in two of the three biomes. The BESD values reflect what can be inferred by the p-values, especially highlighting how, while we do not have a strong significance for the Plains biome, a moderate increase in successes can be observed. Finally, it's worth noting that we can observe a surprisingly large amount of non-preferences for the Savanna biome (see Table II), which might be a hint that the two systems do not produce a clearly different result in that scenario.

TABLE II
NUMBER OF EQUAL ANSWERS FOR THE ADAPTABILITY CRITERION, DETERMINED AS WHEN THE PARTICIPANT GAVE THEM THE SAME SCORE.

	No preference
Biome: Plains	5
Biome: Desert	10
Biome: Savanna	18

TABLE III
PARTICIPANTS' PREFERENCES OVER THE "REALISM" CRITERION. ALSO SHOWN ARE THE p-VALUES, CALCULATED USING A TWO-TAILED BINOMIAL TEST, AND THE BINOMIAL EFFECT SIZE DISPLAY. p-VALUES UNDER 0.05 ARE HIGHLIGHTED IN BOLD.

	Minecraft	Generated	Binomial test	BESD
Biome: Plains	52	14	**2,82E-06**	-57.6%
Biome: Desert	21	45	**4,27E-03**	36.4%
Biome: Savanna	48	18	**2,87E-04**	-45.5%

3) Realism

As can be observed in Table III, we found statistically significant results for all biomes in regard to the *realism* criterion. It appears that the Minecraft algorithm creates settlements that are perceived as more realistic than our system in the Plains and Savanna biomes. The situation is reversed for the Desert biome where our system is perceived as more realistic but, as previously mentioned, that might be an artefact of the desert biome being generally more difficult to handle for both algorithms. The BESD values highlight that there is a significant decrease in preferences towards our system, especially when in the Plains biome.

4) Organic evolution

Finally, we can observe from Table IV that our system is consistently perceived as creating more *organic* settlements. These results are corroborated by both the p-values and the BESD, showing a significant increase in preferences for our system

B. Settlements on varied terrains

Figure 10 shows how settlements can be generated around difficult terrain; lakes, trees, hills, mountains and how the settlement houses adhere to its environment by following terrain features. The figures also illustrate how the traversal segmentation ensures that the settlement is generated upon the largest landmass (see lower right picture). The approach of traversability segmentation ensures that all possible travel paths through the region within which the settlement is to be generated can be traversed for road generation.

TABLE IV
PARTICIPANTS' PREFERENCES OVER THE "ORGANIC" CRITERION. ALSO SHOWN ARE THE p-VALUES, CALCULATED USING A TWO-TAILED BINOMIAL TEST, AND THE BINOMIAL EFFECT SIZE DISPLAY. p-VALUES UNDER 0.05 ARE HIGHLIGHTED IN BOLD.

	Minecraft	Generated	Binomial test	BESD
Biome: Plains	20	46	**1,86E-03**	39.4%
Biome: Desert	21	45	**4,27E-03**	36.4%
Biome: Savanna	21	45	**4,27E-03**	36.4%

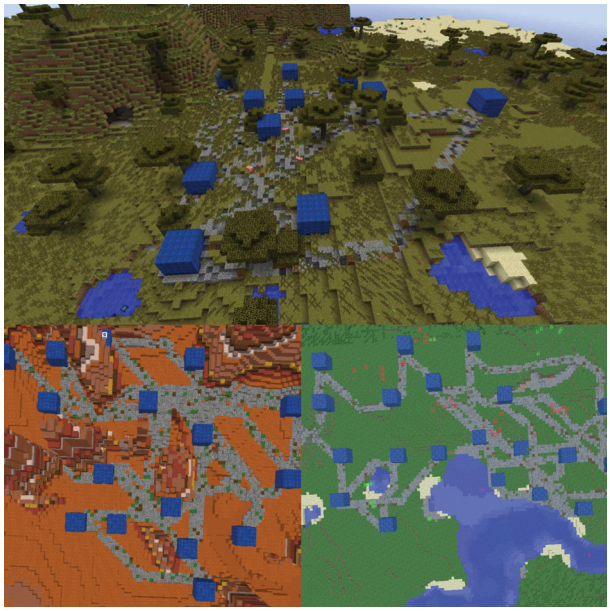

Fig. 10. Sprawling cities evolved over multiple generations of settlers in various biomes with different inherent challenges

VIII. CONCLUSION

Traversal segmentation has proven particularly useful for identifying traversable block sets. Using the largest block set as input for the grid segmentation supplies a useful grid representation of the largest landmass.

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. This means that the complexity of any further searches is greatly decreased, while still allowing for conversion between two- and three-dimensional space.

Traversal segmentation as input for grid segmentation leading to simulation of autonomous agents upon the grid space, has generated adaptable and organic looking settlements. Adapting to the inherent properties of their placement; hills and reachable spaces. The system described provides an interestingly organic settlement layout patterns which are recognised as being more adaptive to a given environment.

The usefulness of a grid representation stems from the minimisation of choices for settlers during simulation. Settlers move in unpredictable patterns when searching for food or 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 reduced. While grid-wise segmentation produces a useful board-game like set of cells, the static nature of 5x5 cell identification leads to many blocks being discarded and removed from the search space. A possible improvement would involve identification of optimal polygons within the search space.

Settler agents currently lack actions to take after reproducing and internal goals which would motivate individualistic action. Internal goals could be seen as an internal understanding of specific jobs, e.g. being a blacksmith or a barkeep. Any internal goal could lead to unique housing and the creation of workplaces. As described in section VI-E, settlers can output a readable set of actions with associated reasoning for each action.

The experiment discussed in Section VII-A provided statistically significant quantitative evidence to the adaptability of settler generated settlements to their environments when compared to in-game generated villages.

A. Survey results

The survey results for realism favoured in-game generated over settler generated settlements in all cases but for the Desert biome, this was likely due to the in-game settlement producing a house in water, whereas the settler village remained on land. Realism was an assumed property resulting from organic settlement evolution over time, likewise the property was assumed to hold for the generated roads and their adherence to and integration into the game world. The survey results showed a clear pattern as to the lack of realistic properties of our settlements. This result was surprising since we assumed a correlation between being ”organic” and ”realistic”. Organic evolution over time was achieved by settler generated settlements with survey results showing a clear bias towards settler generated over in-game generated settlements.

B. Future work

Future work could explore the possibilities of generating thorough chronicles of individual settler lives and analysing the emerging narratives. Further work could investigate the possibility of the children of settlers to improve/expand their parents' shelter. We expect this could create more variety and possibly lead to ”architectural narrative”. Moreover, the current implementation of the settlement behaves like a closed system. Future works could inspect the impacts external events, like simulated disease or natural disasters upon settlers.

References

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

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

[3] Hector P Martinez, Georgios N Yannakakis, and John Hallam. Don't classify ratings of affect; rank them! *IEEE transactions on affective computing*, 5(3):314–326, 2014.

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

[5] Lior Rokach and Oded Maimon. *Decision Trees*, pages 165–192. Springer US, Boston, MA, 2005.

[6] Robert Rosenthal and Donald B Rubin. A simple, general purpose display of magnitude of experimental effect. *Journal of educational psychology*, 74(2):166, 1982.

[7] Julian Togelius, Alex J. Champandard, Pier Luca Lanzi, Michael Mateas, Ana Paiva, Mike Preuss, and Kenneth O. Stanley. Procedural Content Generation: Goals, Challenges and Actionable Steps. In Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius, editors, *Artificial and Computational Intelligence in Games*, volume 6 of *Dagstuhl Follow-Ups*, pages 61–75. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2013.

[8] Wiebe van der Hoek and Michael Wooldridge. Chapter 24 multi-agent systems. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 887–928. Elsevier, 2008.