2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI) | 978-1-6654-0898-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICTAI52525.2021.00182

Multi-agent Story-based Settlement Generation

Jonathan Demke, Robert Morain, Connor Wilhelm and Dan Ventura
Computer Science Department
Brigham Young University
demkejon001@gmail.com, robert.morain@gmail.com, wilhelmconnorr@gmail.com, ventura@cs.byu.edu

***Abstract*—Video game content creation is a creative task that has typically been performed by 3D artists. While procedurally generated worlds provide the opportunity to create arbitrarily large cohesive environments, if the generated content lacks an integrated narrative, the environment may begin to feel generic and auto-generated; human artists can use story narratives to help them create 3D worlds which feel more "alive". This paper describes the StoryViz system which uses a short story as inspiration for generating a 3D settlement in Minecraft, leveraging swarm intelligence to optimize a set of rule-based interest functions. A user survey evaluating the system's generated settlements provides a baseline for further development of the story visualization task.**

***Index Terms*—computational creativity, narrative visualization, settlement generation, Minecraft, generative design**

I. Introduction

Places tell stories. Each place comes embedded with a history of events, characters, and culture. Archaeologists know that history leaves a mark—and these marks can be weaved together to form a cohesive, consistent narrative. Even the novice observer possesses a keen enough sense of attention to detail that the absence or repetition of detail becomes a hallmark of the artificial.

While archaeologists are tasked with translating from place to story, 3D artists translate from story to place. They must use their imagination and creativity to incorporate rich, contextual glints of culture, function, and personality into their designs to enchant and delight their audience.

Procedural content generators are typically rule-based systems used to create 3D worlds which would be too difficult or time-consuming to design by hand. Recent success in titles such as No Man's Sky [1] provide evidence of the utility of this approach. However, as the setting of a story changes, so must the rules and procedural generation methods. In the default Minecraft generator, village generation follows basic, generic rules and largely ignores terrain. The village has no story or history to differentiate it from others, and, in playing the game, once you've seen one of these villages, you've seen them all—for lack of a better expression, they have no soul.

In this paper, we tackle the task of story visualization—converting a written story into 3D rendered artwork. For now, we limit the scope to settlements embedded into generated Minecraft environments. The goals for this task follow evaluation criteria put forth by Salge et al. [2] for the Generative Design in Minecraft Competition[1] (GDMC), including responding to the environment in terms of physical materials and natural barriers, adding sufficient details to satisfy the extrapolation from imagination to visual art, and converging on a scenario consistent with the given story.

We introduce StoryViz, a system that generates narrative-focused 3D settlements in Minecraft. StoryViz uses a provided short story to identify relevant pre-built Minecraft structure assets. These assets are then placed within a Minecraft world using a swarm intelligence algorithm, inspired by the work of Teodorovic [3], which optimizes a set of rule-based interest functions for each asset, similar to the approach of Emilien et al. [4]. We describe in detail the resources used, the system design, as well as the technical difficulties encountered. A user survey evaluates the quality of generated settlements in terms of narrative integration, novelty, functionality, adaptability to the environment, and perceived creativity of the system. A discussion of how our methods influence the novelty and quality of generated settlements follows.

This paper also aims to explore the limitations of current rule-based techniques in the field of computational creativity. Contributions of this work include the following:

1) Casting the creative process as independent multi-agent sub-processes interacting to satisfy conflicting constraints to form an artifact
2) Casting the agent interactions as a multi-objective optimization problem mediated with a set of *interest functions* that impose agent-specific constraints
3) Progress toward narrative-based world generation
4) Demonstrating transformation from one creative domain (short stories) to another (video game world)

II. StoryViz System

The StoryViz system is composed of several major modules, including a *schematic database*, *a schematic manager* and a *multi-agent positioning system*. The schematic manager is itself composed of submodules: *story interpreter*, *story entity selector*, and *general entity selector*. As shown in Figure 1, the system takes as input a story in plain text format, identifies its main elements as keywords and uses those keywords to select appropriate entities for their representation (in this case we limit entity types to architectural constructs, but the system could be extended to include additional types such as people, animals, items, etc.) Once the story-specific entities are defined, they are supplemented with other general entities to fill out the settlement and make the visualization more rich. Each story entity is treated as an autonomous agent,

[1] https://gendesignmc.engineering.nyu.edu/

978-1-6654-0898-1/21/$31.00 ©2021 IEEE
DOI 10.1109/ICTAI52525.2021.00182

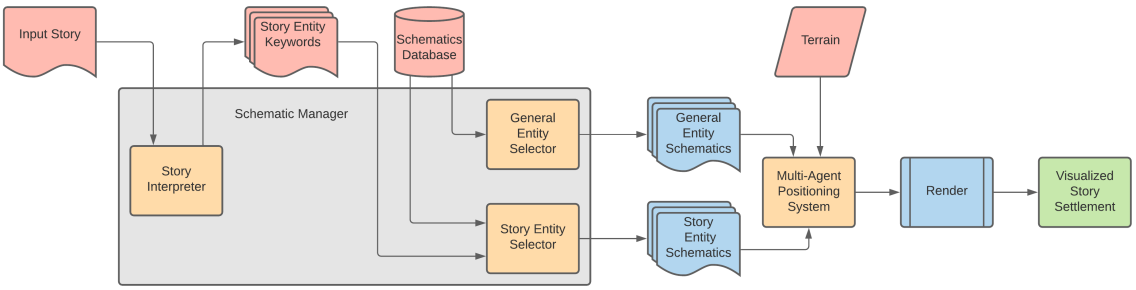

Fig. 1. Architectural overview of the StoryViz system.

with a schematic that defines its 3D structure and rules that define how it is positioned in an environment. Once all the entity/agents are assembled, they are passed to the Multi-Agent Positioning System (MAPS) which positions the structures into suitable absolute and relative locations on the terrain.[2]

A. Story Analysis and Entity Identification

The StoryViz system does not utilize all information within a story; rather, it only utilizes the story's *entities*. These entities are most pertinent to creating a settlement based on the story, as they can be represented by different types of structures. We used the spaCy natural language processing library[3] to identify the entities within the story. To do so, the system extracts all noun phrases from the text. The system uses spaCy's large English model, en_core_web_lg, to tokenize and perform the part-of-speech tagging, allowing the system to separate entities from other information based on their parts-of-speech. Any tokens considered part of a noun phrase are extracted as one entity. Additional cleanup is performed on the extracted phrases, including removing stopwords and repeat (entity) instances. Nouns are extracted with their modifiers, which can provide information required for identifying an entity; for example, a pirate ship is different than a cruise ship. The extracted tokens are vectorized and used to select corresponding schematics for placement in Minecraft.

B. Schematic Selection

Entities are represented within the generated settlement by a static structure. These structures consist of Minecraft blocks and may represent buildings, statues, vehicles, terrain, trees, farms, etc. In order to avoid the complex problem of building a structure block-by-block, the StoryViz system utilizes pre-built structures, referred to as *schematics*, created by the Minecraft community.

Since the schematics were scraped en masse from the web, the only easily available information about the schematics is the filename F. Extracting more information from the 3D array of blocks in the schematic file is unnecessary, assuming the files are properly and clearly named. In addition to the information provided by the filename, the system categorizes each schematic into one of four structure types: *residential*, *rural*, *commercial* and *public*. This type information is used to determine placement rules.

Type classification for a schematic with filename F is performed by vectorizing the type names and the words in F and finding the type with the smallest average cosine distance from the words in the filename:

$$\tau(F) = \underset{t \in T}{\operatorname{argmin}} \left\{ \frac{1}{|F|} \sum_{u \in F} \delta(\mathtt{vec}(u), \mathtt{vec}(t)) \right\}$$

where $T = \{\text{“residential”}, \text{“rural”}, \text{“commercial”}, \text{“public”}\}$, $\mathtt{vec}()$ is the spaCy vectorization function and δ is the cosine distance between vectors.

1) Story Entities: In order to represent story entities within Minecraft, the system connects schematics to key phrases using spaCy word vectorization and cosine distance. To associate a schematic with a key phrase, we find the schematic filename with the smallest average cosine distance between the words in the filename and the words in the key phrase:

$$\sigma(K) = \underset{F}{\operatorname{argmin}} \left\{ \frac{1}{|F||K|} \sum_{u \in F} \sum_{v \in K} \delta(\mathtt{vec}(u), \mathtt{vec}(v)) \right\}$$

where F is the filename and K is the key phrase.

2) General Entities: Since the story is not likely to provide enough detail to generate an entire settlement, more general-purpose structures, such as houses, farms, stores, and churches, are needed to augment the story schematics. Because general entities should serve as part of the background for the visualized story, we heuristically selected a subset of the schematic database that contains only the more generic-looking schematics for each structure type. Using this restricted schematics database, the system can randomly select general entity schematics that do not distract from the main story elements, resulting in a more cohesive visualization. In contrast, story element schematic selection utilizes the full database of scraped schematics, allowing story schematics to stand out amongst the general schematics.

C. Multi-Agent Positioning System (MAPS)

Once all schematics are identified, they are provided to the positioning system, along with their type classification.

[2]Examples and code at https://mind.cs.byu.edu/projects/storyviz/ and https://github.com/rmorain/storyviz.

[3]https://github.com/explosion/spaCy and https://spacy.io/

MAPS creates a grid that represents the Minecraft terrain. Each entity structure starts with a randomly assigned position $\mathbf{p} = (x, y)$ and iteratively changes its own position over multiple timesteps, using local information, similar to self-organizing systems as described by Couzin and Krause [5]. Each iterative change in position, $\Delta\mathbf{p}$, is called the *interest vector*. The interest vector is computed from many different interest functions, similar to Emilien et al.; however, where Emilien et al. use interest functions to calculate the probability of a structure being placed, our interest functions return a vector that points towards a better position. We use the following interest functions: *sociability*, *centroid*, *slope*, *nearness to water*, *nearness to road*, *geographic domination*, and *collision avoidance*. Each structure type may employ a unique subset of these interest functions.

1) Sociability: uses attraction and repulsion to group structures of the same type. It builds a vector as a linear combination of attraction vectors, one for each of the k-nearest structures N_τ of a specific type (e.g. residential structures are sociable with other residential structures). Attraction is positive for each neighbor outside an *attraction radius* r_a and negative for each neighbor within a *repulsion radius* r_r.

$$f_b(\mathbf{p}) = \frac{1}{\max(1, Z_b)} \left[\sum_{\substack{q \in N_\tau \\ ||q-p|| > r_a}} (\mathbf{q} - \mathbf{p}) - \sum_{\substack{q \in N_\tau \\ ||q-p|| < r_r}} (\mathbf{q} - \mathbf{p}) \right]$$

where Z_b is a normalization term that ensures the result is bounded by the unit circle. The attraction radius helps control village size—increasing the radius results in larger villages and decreasing the radius can result in multiple smaller villages. The repulsion radius controls how congested the village feels.

2) Centroid: is similar to sociability, however rather than being attracted or repulsed, a structure tries to move towards the center of its k-nearest neighbors N_τ of some type.

$$f_c(\mathbf{p}) = \frac{1}{\max(1, Z_c)} \left[\left(\frac{1}{|N_\tau|} \sum_{q \in N_\tau} \mathbf{q} \right) - \mathbf{p} \right]$$

Many practical structures require ease of access to neighbors. For example, a store needs to be easily accessible to its consumers and a barn needs to be close to all of its farms. This adds functionality and realism to the village design.

3) Slope: takes the terrain's elevation into account by computing the gradient at each structure's position and moving the structure down (or up) the gradient.

$$f_s(\mathbf{p}) = \pm \frac{1}{\max(1, Z_s)} \nabla \mathbf{p}$$

This rule allows structures to slide downhill so that villages are built in valleys and plains or to slide uphill to form a more mountainous village. Moving a structure uphill is also useful for specific structures such as watchtowers that should be placed on high ground.

4) Geographic Domination: Like Emilien et al., we take the terrain's elevation into account by trying to position a structure in a higher location than its neighbors and in a location with more neighbors in its influence radius r_g. Random locations near the structure are sampled, and the structure is moved in the direction of the highest valued location:

$$f_g(\mathbf{p}) = \frac{1}{\max(1, Z_g)} \left[\left(\operatorname*{argmax}_{s \in S} \sum_{q \in N(s)} \frac{h(\mathbf{p}) - h(\mathbf{q})}{1 + \|\mathbf{p} - \mathbf{q}\|^2} \right) - \mathbf{p} \right]$$

where S is the set of randomly sampled locations, $N(s)$ is the set of structure positions that lie within r_g of s, and $h(\mathbf{x})$ is the elevation at position $\mathbf{x}$. Geographic domination is useful for placing important structures, such as churches or fortifications, in locations that can be seen from everywhere.

5) Nearness: to a material such as water or roads is useful for fulfilling the needs of structures. Farms need to be placed near water for their crops, while houses need to be near roads. We move these structures toward their respective material(s) and measure the distance with a family of functions:

$$f_?(\mathbf{p}) = \frac{1}{\max(1, Z_?)} (\mathbf{m} - \mathbf{p})$$

where '?' is a wildcard that represents the material in question and $\mathbf{m}$ is the location closest to $\mathbf{p}$ that contains that material. For this version of StoryViz, water and roads are the only materials we use to calculate nearness vectors, but this family of functions could easily be expanded to any type of material, resource, structure (type), etc.; for example, quarries need to be near rocky mountains. These functions add functionality and realism to the village by placing structures near resources.

6) Collision Avoidance: moves structures away from each other when they are colliding (defined as any overlap between the bounding boxes of multiple structures). To avoid collisions, a structure is moved away from the centroid of the set C of structures with which it is currently colliding:

$$f_a(\mathbf{p}) = -\frac{1}{\max(1, Z_a)} \left[\left(\frac{1}{|C|} \sum_{q \in C} \mathbf{q} \right) - \mathbf{p} \right]$$

Structures that collide can cause unwanted, unrealistic artifacts when placed. To prevent these artifacts, all structures use the collision avoidance interest function.

7) The Interest Vector: is a linear combination of the vectors returned from a set of type-specific interest functions:

$$\begin{aligned} \Delta\mathbf{p} = \alpha_b f_b(\mathbf{p}) + \alpha_c f_c(\mathbf{p}) + \alpha_s f_s(\mathbf{p}) + \alpha_g f_g(\mathbf{p}) \\ + \alpha_w f_w(\mathbf{p}) + \alpha_r f_r(\mathbf{p}) + \alpha_a f_a(\mathbf{p}) \end{aligned}$$

where weights are non-zero only for those interest functions that apply to the structure type. Changing the weights of the interest function vectors affects the behavior of the agents by increasing or decreasing their interest in a given goal.

8) Final Positioning: of the structures occurs in two phases: *free positioning* and *annealed positioning*. For the free positioning phase, structure locations are initialized uniform randomly, and structures move around for T_f time steps, with MAPS updating positions using the interest vector at each time step: $\mathbf{p}_{t+1} = \mathbf{p}_t + \Delta\mathbf{p}_t$. The purpose of the free positioning phase is to a) find initial structure locations for the annealing phase, and b) estimate an *annealing radius*:

$$\tilde{r} = \frac{1}{T_f|B|}\sum_{t=0}^{T_f}\sum_{b\in B}\left\|\Delta\mathbf{p}_t^b\right\|$$

where B is the set of all structures.

When free positioning is complete, MAPS enters the annealed positioning phase, which establishes the structures' final positions. In this phase iterative re-positioning continues; however, whenever the length $\|\Delta\mathbf{p}_t^b\|$ of a structure's interest vector is shorter than the annealing radius, if the structure position is not on top of water, roads or other structures, then the structure position is fixed with probability $(\tilde{r}-\|\Delta\mathbf{p}_t^b\|)/2\tilde{r}$.

Whenever a structure is placed, it is connected to the road network using A*, similar to the technique in Mizdal and Pozzer [6]. To avoid local minima, non-fixed structure positions are occasionally randomly re-initialized. At the end of the annealed positioning phase, any unplaced story entity is placed at its current position, and any unplaced general entity is removed (see Figure 2 for an example MAPS output).

D. System Aesthetic

The system evaluates the quality of an artifact by its settlement dissatisfaction score, calculated using the final interest vector $\Delta\mathbf{p}_*^b$ of each structure in the settlement. We interpret the magnitude of each structure's final interest vector $\|\Delta\mathbf{p}_*^b\|$ as a simple measure of the structure's "dissatisfaction". The average structure dissatisfaction over the entire settlement can provide an indication of the quality of the final layout:

$$\mathcal{L}(B) = \frac{1}{|B|}\sum_{b\in B}||\Delta\mathbf{p}_*^b||$$

Because the annealing algorithm is stochastic, each MAPS run will likely produce a different artifact. The best artifact (lowest $\mathcal{L}$) can then be rendered by MCEDIT[4] as a Minecraft world. Exploring multiple artifacts with MAPS increases the chances of finding a good solution for difficult terrain, resulting in better settlement realism. In less challenging terrain, MAPS will find many solutions, increasing the novelty of the artifacts without sacrificing quality.

III. Evaluation

To evaluate the system and its output, we wrote eight stories and ran the system a single time for each (i.e., we did not try to optimize the artifact for the story); from the eight resulting settlements, we chose six and surveyed 91 participants:

1) participants were informed that the purpose of the system was to map a story to a setting in Minecraft

[4]https://www.mcedit.net

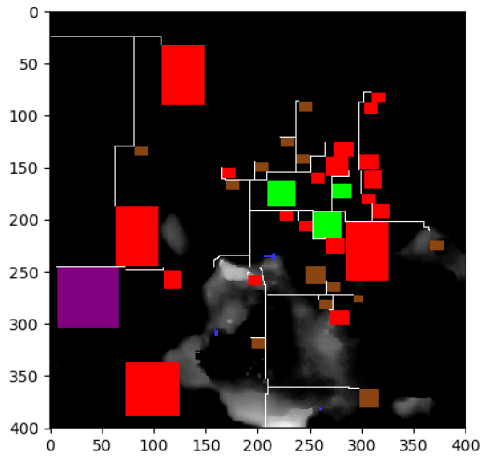

Fig. 2. Output of the MAPS module: terrain height is greyscale; bodies of water are blue; roads are white; residential structures are red, rural brown, commercial green, and public purple. Note how structures of the same type are clustered together and that most structures are constructed on low flatland.

2) participants were required to read a story and watch a video of screenshots of each settlement
3) participants were asked four questions for each of the generated artifacts
4) After seeing all 6 settlements, participants were asked two questions regarding the overall system.

For the assessment questions, participants were asked to respond to the following Likert items on a scale from 1 to 10 (1 meaning "not at all", 10 meaning "completely"). The first four questions apply to each generated settlement:

- How well does the world capture the story?
- How functional/realistic is the world?
- Does the world handle environmental constraints, such as rivers, valleys, mountains, etc?
- How creative is this world?

The final two questions apply to the overall system:

- How different were each of the worlds created?
- In your opinion, how creative is the system?

The results from the survey can be seen in Figures 3-4. While the *story* scores are uniformly distributed, the *functional*, *responsive*, and *creative* scores are clearly right-skewed. The overall system results are an average score of 6.55 for novelty and 6.32 for creativity. It is interesting to note that lower story consistency did not negatively affect ratings for other characteristics nor overall system evaluation—even though StoryViz was not always able to transform the word domain into the visual domain with high fidelity, its attempts to do so often result in artifacts that the participants consider realistic, functional and creative. As a result, they also consider the system fairly novel and creative; however, the lower score in story consistency does indicate room for improvement.

Our evaluation queries similar features as the GDMC which uses *adaptability*, *functionality*, *narrative*, and *aesthetic*. The high scores for each feature, based on 3 years of competition among researchers and hobbyists, are 5.82, 5.18, 5.02 and

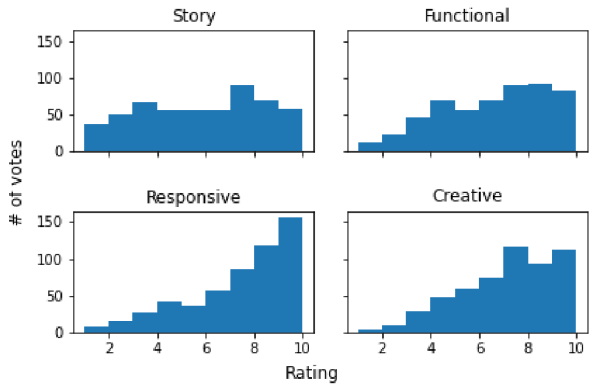

Fig. 3. Combined ratings from 91 respondents for six generated settlements. Averages: 5.40 (story), 6.10 (functional), 7.02 (responsive), 6.73 (creative).

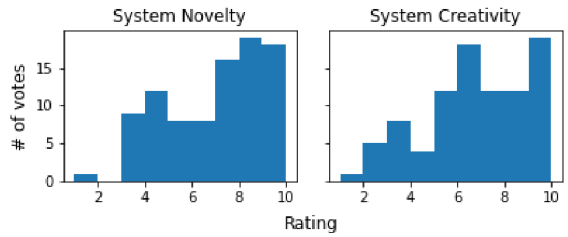

Fig. 4. Overall system ratings. Averages: 6.55 (novelty), 6.32 (creativity)

5.91 respectively, with the average of those averages being 5.48. Our scores, shifted from the Likert scale to GDMC's 0-10 scale, would be *responsive*: 6.69, *story*: 4.89, *functional*: 5.67, *creative*: 6.36, and our overall average would be 5.90. Note that our results come from novice observers, whereas GDMC is critiqued by a panel of experienced judges. GDMC's goals differ slightly as well, in that StoryViz aims to build a settlement based on a story while GDMC systems focus on building a settlement that will visually tell a story [7].

IV. Discussion

The multi-agent creativity exhibited here should not be confused with independent high-level creative agents interacting in an eco-system [8] but is rather a demonstration of low-level subprocesses interacting to facilitate a (single, dependent) emergent creative process. This multi-agent approach, rather than treating the subprocesses as one homogeneous step or as a temporally constrained sequence of steps, makes the creative process a continuous negotiation between agents—subprocesses that continually change their behavior until they have settled on a global solution that satisfices each of them.

Although we limited this to homogeneous agents, i.e. structures, it could be extended to less homogeneous settings: people, tools, animals, landscape features, etc.; in other domains, examples might include lyric, melody and harmony agents interacting to form a song; or shape, color, and content agents interacting to create a website.

The system's extraction of story information can be improved by leveraging additional kinds of linguistic information, including prepositions, modifiers and verbs. The current version of the system only extracts entities from the story; it does not extract actions or events. Event extraction would provide the system with additional information; however, performing potential actions during or after settlement placement is nontrivial if the extracted information is un-ordered. Utilizing a semantic network could allow this information to be ordered in a logical way, allowing events, actions, and relative positions to be correctly extracted.

Increasing the (meta-)information available for schematics (currently, only descriptive filenames are used) would likely improve the key-phrase-based schematic selection, which would in turn likely improve the recognizability of the inspiring story given the generated village.

The MAPS module could also be improved in various ways. Preventing structures from being placed in undesirable terrain, such as partially underground or on trees, would improve the village quality. Additional rules could aid structures in finding more realistic stable states. Additional evaluation functions could be considered; for example, functionality could be estimated via an AI agent exploring the settlement in Minecraft to evaluate the settlement's navigability.

Finally, it may be possible to have a machine learning model discover (hyper)parameters that optimize human evaluation of generated villages. However, introducing such additional complexity may increase the difficulty of finding stable solutions.

References

[1] Hello Games, "No Man's Sky," 2016.

[2] C. Salge, M. C. Green, R. Canaan, and J. Togelius, "Generative design in Minecraft (GDMC) settlement generation competition," in *Proceedings of the 13th International Conference on the Foundations of Digital Games*, 2018.

[3] D. Teodorovic, "Transport modeling by multi-agent systems: A swarm intelligence approach," *Transportation Planning and Technology*, vol. 26, pp. 289–312, 2003.

[4] A. Emilien, A. Bernhardt, A. Peytavie, M.-P. Cani, and E. Galin, "Procedural generation of villages on arbitrary terrains," *The Visual Computer*, vol. 28, no. 6-8, pp. 809–818, 2012.

[5] I. D. Couzin and J. Krause, "Self-organization and collective behavior in vertebrates," *Advances in the Study of Behavior*, vol. 32, no. 1, pp. 10–1016, 2003.

[6] T. Mizdal and C. Pozzer, "Procedural content generation of villages and road system on arbitrary terrains," in *Proceedings of the 17th Brazilian Symposium on Computer Games and Digital Entertainment*, 2018, pp. 556–562.

[7] C. Salge, C. Guckelsberger, M. C. Green, R. Canaan, and J. Togelius, "Generative design in Minecraft: Chronicle challenge," in *Proceedings of the Tenth International Conference on Computational Creativity*, 2019, pp. 311–315.

[8] K. Jennings, "Developing creativity: Artificial barriers in artificial intelligence," *Minds and Machines*, vol. 20, pp. 489–501, 11 2010.