An Academy of Spatial Agents: Generating Spatial Configurations with Deep Reinforcement Learning

Conference Paper · September 2020

CITATIONS
0

READS
344

2 authors:

Pedro Luís Alves Veloso
Carnegie Mellon University
27 PUBLICATIONS **41** CITATIONS
SEE PROFILE

Ramesh Krishnamurti
Carnegie Mellon University
109 PUBLICATIONS **1,016** CITATIONS
SEE PROFILE

An Academy of Spatial Agents

Generating spatial configurations with deep reinforcement learning

Pedro Veloso[1], Ramesh Krishnamurti[2]
[1,2]Carnegie Mellon University
[1]pedroveloso13@gmail.com [2]ramesh@andrew.cmu.edu

Agent-based models rely on decentralized decision making instantiated in the interactions between agents and the environment. In the context of generative design, agent-based models can enable decentralized geometric modelling, provide partial information about the generative process, and enable fine-grained interaction. However, the existing agent-based models originate from non-architectural problems and it is not straight-forward to adapt them for spatial design. To address this, we introduce a method to create custom spatial agents that can satisfy architectural requirements and support fine-grained interaction using multi-agent deep reinforcement learning (MADRL). We focus on a proof of concept where agents control spatial partitions and interact in an environment (represented as a grid) to satisfy custom goals (shape, area, adjacency, etc.). This approach uses double deep Q-network (DDQN) combined with a dynamic convolutional neural-network (DCNN). We report an experiment where trained agents generalize their knowledge to different settings, consistently explore good spatial configurations, and quickly recover from perturbations in the action selection.

Keywords: *space planning, agent-based model, interactive generative systems, artificial intelligence, multi-agent deep reinforcement learning*

INTRODUCTION

In this paper we investigate a generative model for spatial design based on learning agents, which supports fine-grained and real-time human-computer interaction.

Typically, computational agents are used for modeling natural or artificial phenomena that unfold over time based on the interactions with the shared environment (Agent-based modeling, ABM) (Wilensky and Rand, 2015). In the context of multi-agent space planning (MASP) (Veloso, Rhee and Krishnamurti, 2019), an agent can represent a spatial entity with local control, interweaving individual perception and action to decide how space should be shaped, occupied, or partitioned. The 'environment' comprehends elements of the space that are independent of the agents and that support the different interactions. During a simulation, agents receive signals from the environment, neighboring agents, or even the designer and make decisions to change the

spatial configuration. This is in contrast to many of the prevailing generative methods.

Overall, agent-based models support many characteristics that are beneficial for design exploration. They enable the control of configurations that are not feasible with a centralized modeling strategy. Agents can produce patterns or behaviors that are not necessarily predictable from the perspective of their internal programs. They operate on a shared environment and use a local action space, facilitating a fine-grained interaction with the designer. The distributed decision making of multiple spatial agents in a simulation results in partial design states with valuable information about design trade-offs and opportunities for local changes.

Gap in multi-agent space planning

Agent-based models used in spatial synthesis include swarm algorithms (flocking and pheromone navigation), cellular automata, reaction-diffusion, and physics simulation (Herr and Ford, 2015; Veloso, Rhee and Krishnamurti, 2019). Cellular automata enable the emergence of patterns and has been successfully applied to conceptual form generation, urban morphology (Koenig, 2011), and even to building design (Araghi and Stouffs, 2015), but the cell-based computation imposes strict restrictions for the satisfaction of architectural requirements. Physics simulation (rigid or soft body) is a more general technique that conciliates simple control with intuitive interaction and can be adapted for different spatial problems and objectives. However, physics-based agents are reactive agents that approximately follow laws of physics. They do not have any sophisticated policy (i.e. a probability of choosing actions in every state) to manage spatial conflicts or to explore the design space. Bio-inspired models, such as swarm algorithms simulate exogenous phenomena (social navigation) in which the units can move and interact in space.

While these models enable the incorporation of certain architectural requirements, such as area or adjacency, it is usually harder to adapt them to produce conventional architectural forms or satisfy other requirements, without resorting to some form of destructive technique to improve their configurations. The challenge for agent-based, generative models is to develop control strategies that incorporate specific architectural requirements and preserve the fine granularity of the simulation.

Developing self-learning agents

Reinforcement Learning (RL) is a viable alternative to develop control strategies that incorporate specific architectural requirements and to preserve the fine granularity of the simulation. With RL, agents can learn how to build spaces by interacting with the environment and improving their performance with the respect to certain goals. After training, designers can interact with the spatial agents in real-time to explore design alternatives.

In CAAD, RL has been recently applied to varied topics, such as intelligent adaptive building control (Smith and Lasch, 2016), autonomous robots (Hosmer and Tigas, 2019), fire egress evaluation (Jabi et al., 2019), and machine feedback (Xu et al., 2018). For the generation of spatial configuration, RL has been adopted for automatic decision-making with shape grammars (Ruiz-Montiel et al., 2013) and a natural selection algorithm has been used to improve the policy network of competing teams of agents that create clusters of building blocks on a grid (Narahara, 2017).

In this paper, through a proof of concept we describe an instance of our method to train agents to generate spatial configurations by interacting in an environment. We briefly describe the representation of the environment, agents, actions and objectives. Then, we focus on the description of the algorithm, which uses RL and spatial objectives provided by the designer to train the agents. Particularly, we use custom RL techniques based on deep learning for the multi-agent setting – which is referred to as multi-agent deep reinforcement learning (Nguyen, Nguyen and Nahavandi, 2018). Finally, we evaluate the performance of our proof of concept in an architectural application.

SPATIAL REPRESENTATION OF ENVIRONMENT AND AGENTS

Environment and agents

Information is organized as stacked grids of information, which we refer to as arrays. For instance, the environment is an array of shape (d_{env}, h, w). The width (w) and height (h) define the size of the grid, and d_{env} defines the number of layers of information. The basic layer of information of the environment contains two sets of cells: empty (0) and obstacle (-1). Only empty cells can be occupied by agents. More information can be added to the array as additional layers.

The agent is an array with shape $(d_{\text{agent},h,w})$. It represents a spatial partition, such as a building, a room or an area for certain activity. In this proof of concept, the spatial partition is a polyomino with no holes (PnH, see figure 1, row 1). A polyomino is a set of grid cells that share edges. By avoiding the existence of holes, the PnH has certain interesting properties. For instance, it forms a polygon with orthogonal edge boundaries and avoids containment of one agent by another. Therefore, it can form custom diagrams of floorplans, comprehending organizations such as tight packing and loose packing. The shapes can approximate any polygon, such as orthogonal and non-orthogonal rectangles, or free forms such as amoeba-like shapes. The resulting polyominoes can be post-processed to depict bubble diagrams or support the development of a parametric model. In our setting, each PnH induces different types of cells (see figure 1, row 2).

Action space

The actions available to the agents are (1) single-cell expansion, (2) single-cell retraction, and (3) no-action. The set of legal cells by which the agent's expansion and retraction is defined consists of cells that are not blocked by an obstacle of the environment, preserve the PnH-ness of all the agents, and are inside the action grid. It is important to notice that the mask or action grid (see figure 1, row 3) has shape $(h_{action}, w_{action}) < (h, w)$ and is placed on the environment based on the current centroid of the agents' PnH. Therefore, as the agents expand the PnH in a certain direction, they also move the centroid and, consequently, the action grid. The basic actions are building blocks that can be combined to form complex interplays such as blocking, pushing, pulling, or attraction. For example, if the agent eliminates all the cells of its PnH using retraction, it then can jump to any legal cell inside the current action grid, using a single expansion. The agent also has the option of not executing any action this turn.

Spatial objectives

Our proof of concept provides for the definition of a variety of objectives. In our first experiments, we focused on simple objectives based on neighborhood information, such as smooth adjacency, or strictly based on local information, such as area and shape – a metric that stimulates shapes with few folds, such as rectangle, L or U (for more details, see Veloso and Krishnamurti, in press). For each of these objectives, there is a utility function and a spatial representation. The utility functions $(u_{adj}, u_{area}, u_{shape})$ return a value in the unit interval representing the performance of the agent in that state with respect to a goal. The spatial functions $(g_{adj}, g_{area}, g_{shape})$ return an array with size (h, w) containing spatial hints for the performance of an agent, which intends to ease training and enable parametrization by the user (see Figure 1, rows 4-6). The total utility (u_{total}) of an agent state is based on a combination of the selected goals.

A structured representation

The state representation is composed of the different layers of information. In our proof of concept, we organized an array $(7, h, w)$ with the obstacles (environment), types of cells, utility, mask, folds (g_{fold}), area (g_{area}), and adjacency (g_{adj}) (see Figure 1). The organization of the state as a fixed array conditions the learning and enables the use of efficient computer vision algorithms and machine learning models.

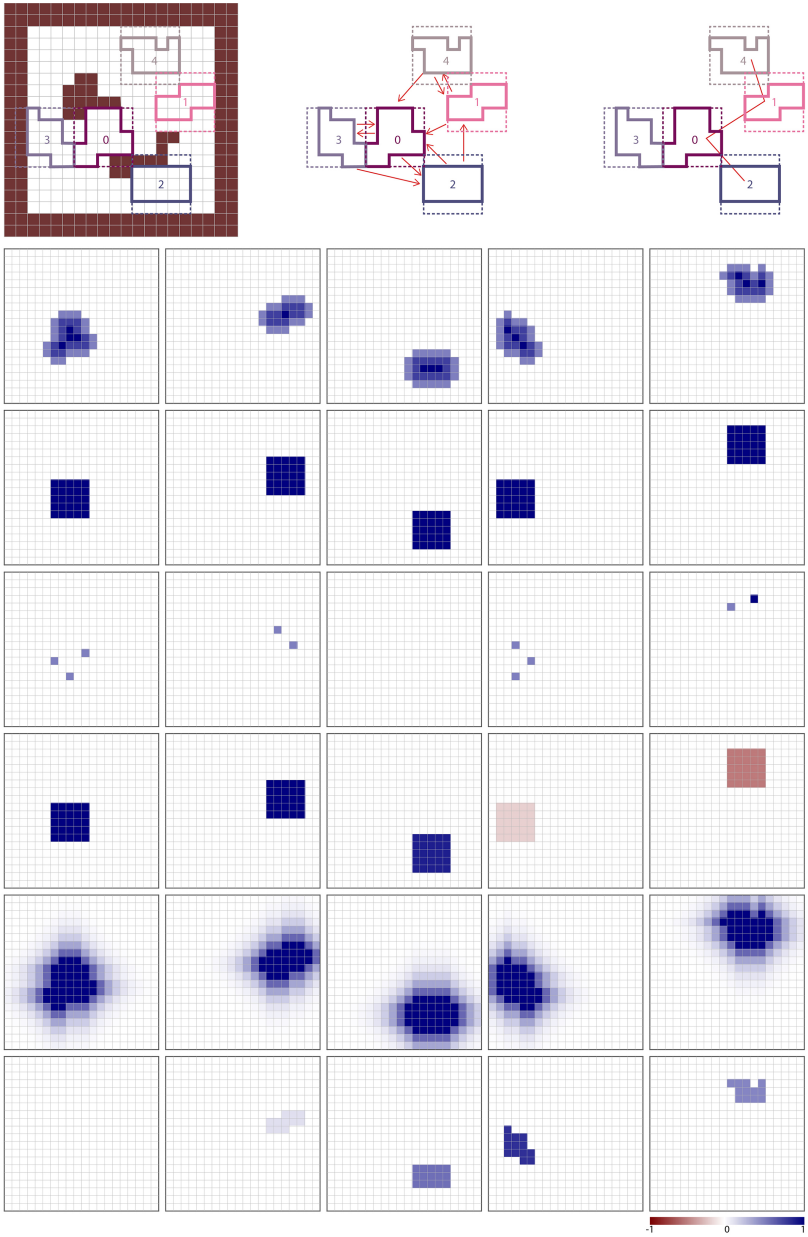


Figure 1
Row 1: left: simplified visualization of agents in the grid with obstacles; middle: closest neighbors; right: adjacency goals. Row 2: classification of the cells: structure (dark blue), surface (medium blue), and offset (light blue); row 3: mask with the action space of the agent; row 4: g_fold: indication of folding cells: L-folds (light blue), U-folds (dark blue); row 5: g_area: the agents have the respective (area/ target area): 0: (17/ 2), 1: (11/ 4), 2: (15/ 8), 3: (13/ 16), 4: (12/ 25); row 6: g_adj: soft adjacency values; row 7: the current utility of the agent (u_total) represented in the internal cells of the agent.

LEARNING

Deep Reinforcement Learning

In Reinforcement Learning, agents learn by interacting with the environment and discovering behaviors that can maximize their performance with the respect to certain goals. The mathematical framework for this interaction is a Markov decision process (MDP) (Sutton and Barto, 2018, pp. 47-57).

In our proof of concept, the MDP consists of: a set of states defined by the configuration of the agent and of the environment; a set of actions (expansions, retractions, or no-action) that can change the state of the agent; a set of reward signals based on the Δ of the combined utility functions (u_{total}) between two states. An agent interacts with the environment by observing its current state (s) and taking an action (a) in the MDP. The environment returns the next state (s') and an evaluative feedback in the form of a reward signal (r).

To train our agent, we use an off-policy, model-free approach called Q-learning, which relies on estimating the future cumulative rewards for the state-action pairs (Q-values) of the MDP. The agent interacts with the environment using a policy π derived from the Q-values estimates. Given an observation (s, a, r, s'), the algorithm updates the estimate $Q(s, a)$ towards a target defined by the recursive relationship between subsequent optimal Q-values in the MDP (see formula 1). This target includes a discount rate $\gamma \in [0, 1]$ to avoid infinite returns.

$$r + \gamma \max_{a'} Q(s', a') \tag{1}$$

Conventional Q-learning algorithms store every Q-value in a tabular data structure. However, to address the large state-action spaces of our setting, we estimate the Q-values using neural networks. More specifically, we use the Double Deep Q-Network algorithm (DDQN) (van Hassel, Guez and Silver, 2016). DDQN uses two similar deep neural networks (Q_θ and $Q_{\theta'}$) to estimate the Q-values.

In our version of DDQN, the agent interacts with the environment in multiple episodes, where, the objective parameters, environment values, and PnH are randomly initialized. It relies on the alternate execution of two steps: (1) interaction with the environment and (2) update of the network estimates.

In step 1, the agent interacts with the environment using an ε-greedy policy to generate observations (s, a, r, s') – i.e. it selects a random action with ε probability or an action that maximizes expected cumulative reward according to the online Q-network (Q_θ) otherwise. At the beginning of a training session, ε is a large value, so the policy explores random trajectories in the environment. As ε is reduced over training, the agent switches to the exploitation of the best estimates. The observations are augmented with rotation and reflection, then added to a large memory buffer D, from which they are later retrieved for training. The buffer randomizes the data, removing the correlation between observation samples and reducing the variance for the updates.

In step 2, the algorithm updates the current estimate of Q_θ with a batch of observations (S, A, R, S') retrieved from D. Q_θ is trained with a gradient descent method using the target:

$$R + \gamma Q_{\theta'}(S', \text{argmax}_a Q_\theta(S', a)) \tag{2}$$

Then, $Q_{\theta'}$ is slowly updated to approximate Q_θ using a step-size τ, which reduces the correlation between Q estimates and target values.

Multi-agent deep Reinforcement Learning

In the previous section we described the algorithm for a single agent. However, our prototype requires multi-agent deep reinforcement learning (MADRL) methods to address the problem of the interaction of multiple spatial agents that compete and cooperate to improve individual and collective rewards. In this context, the formalization of the MDP generalizes to a Markov Game, where the state contains the information of all the agents and the action space is defined by joint actions (Nguyen, Nguyen and Nahavandi, 2018, p. 10). MADRL imposes several additional challenges to RL, such as exponential growth of the joint-action space, non-stationarity due to the observation of other agents, heterogeneous agents,

partial observability, and credit assignment of global rewards.

Our approach to the multi-agent setting relies on a dynamic convolutional neural network (DCNN) to approximate the Q-values of the multiple agents. DCNN estimates the Q-values after receiving three arrays as the input:

- An array $\left(n_{\text{agents}}, 7, h, w\right)$ with the state information (see section "A structured representation").
- An array $\left(n_{\text{agents}}, m\right)$ with the indices of the m closest neighbors to reconfigure DCNN.
- An array $\left(n_{\text{agents}}, k\right)$ with the indices of the adjacent goals to reconfigure DCNN.

Each agent is represented in a strand and there are five main network blocks (see Figure 2).

Over the sequence of blocks, the network captures a larger and larger receptive field, defined by neighbors and neighbors of neighbors. The strands of the network for each agent not only share weights but also gradients. Therefore, the backpropagation step of DDQN enables cooperative adjustment of weights between agents. As a result, while the number of closest neighbors (m) and maximum connected agents (k) should be fixed, the number of agents in the model can be changed during execution, by adding or removing strands.

The first block receives the state array of each agent and uses a convolutional neural network to create a richer representation. The second and third block use convolutional neural networks on the resulting arrays of the closest neighbors and connected agents. The fourth block uses a convolutional neural network on the combination of the resulting arrays from previous blocks. Finally, the fifth block is a Q-network that receives the resulting array from the previous block associated with the mask of the agents and computes the respective Q-values using fully connected layers.

EXPERIMENT

env. grid	16 x 16 (w/padding)	**loss function**	Smooth L1 Loss
action grid	5 x 5	**optimizer**	RMSProp
actions space	26	**batch size**	64
n of agents	6	**learning rate (α)**	0.0001
adacencies (k)	3	**τ**	0.02
neighbors (m)	3	**capacity of D**	10000
episodes	43350	**initial** ε	0.9
steps per episode	64	**minimum** ε	0.1
γ	0.92	ε **reduction**	custom schedule

Table 1
Configuration of training.

Our approach to interactive generative design is to train spatial agents that build custom spatial configurations and adapt to changes devised by real-time interactions with the designer. This interaction includes not only changes in the parameters of the simulation (ex: number of agents, area, adjacency, and amount of randomness in action selection) but also direct changes to the PnH configuration of the agents and of the environment.

In this section, we report an experiment that evaluates the quality of the spaces generated by the agents, and their capacity to generalize their knowledge to unknown situations and to react to perturbations in the cell configuration.

In the training setting (see Table 1), six agents are randomly initialized and interact in randomly generated environments during multiple episodes of 64 steps. The reward signals are derived from the following utility function:

$$u_{\text{total}} = 0.5u_{adj}(u_{area} + u_{shape}) \quad (3)$$

In the experimental setting we use twelve agents to represent the spaces of a generic two-bedroom house (see Figure 3). There are two custom environments: a terrain on the top of a hill with irregular boundaries and a terrain with a bridge over a stream. For each environment, the agents interact for 6500 steps, which are divided in three stages that are separated by two phases of perturbations:

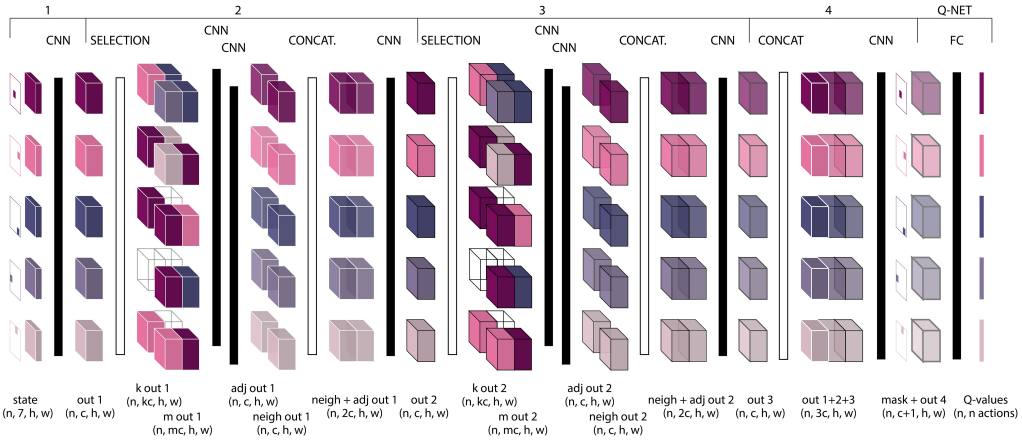


Figure 2
Simplified graphical representation of the layers, operations, and arrays of DCNN. Every CNN is composed of 2 convolutional layers with 64 filters of size 3x3, stride 1, padding 1 and a Leaky ReLU activation. The final FC network has 3 linear layers with 104, 52, and 26 neurons. Its two first layers use Leaky ReLU activation, while the last uses a hyperbolic tangent function (Tanh).

- Stage 1 (0-1999): the agents start at random positions and use the learned policy 95% of the time and random actions 5% of the time – i.e. ε-greedy policy with $\varepsilon = 0.05$.
- Perturbation 1 (2000-2099): actions are selected randomly.
- Stage 2 (2100-3099): agents use an epsilon greedy policy with $\varepsilon = 0.05$.

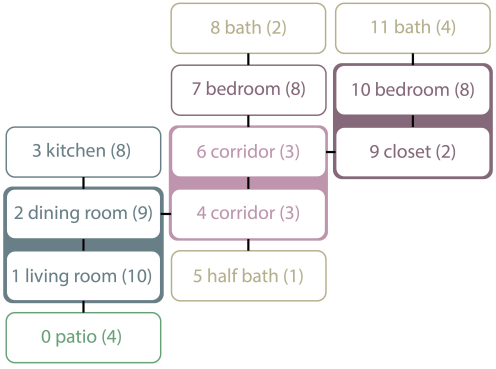


Figure 3
Program of the house for the experiment. The leading number is used to identify the spaces in the next images. The numbers in parenthesis indicate area. The connections indicate adjacency. The colored grouping indicates spatial integration between spaces.

- Perturbation 2 (3100-3499): actions are selected randomly.
- Stage 3 (3500-6499): agents use an epsilon greedy policy with $\varepsilon = 0.05$.

The cell configuration of the obstacles and agents is the input of a parametric model which generates the geometry for visualization, with floors, walls, windows, and openings for doors. The openings are inserted randomly in the intersection between adjacent environments. The windows are inserted in one of the external edges of the PnH (bedrooms and kitchens) or in one of the external edges of a cell of the PnH (bathrooms). Some partitions are integrated to create open spaces, such as in the dining and living room or in the bedroom and closet. The social spaces have glass walls, the patio does not have any walls, and the remaining spaces have walls on the edges without doors or windows.

Figure 4 and Figure 5 show some snapshots of the resulting configuration for the two environments over 6500 steps. The agents generated multiple configurations with high scores and adapted to the obstacles. Overall, the agents improved the performance of the global configuration either from random initialization or from a random perturbation in all the scenarios. After the perturbations, the agents were able to re-organize in different spatial configurations, which shows the potential for direct interaction with the designer. Also, interesting adaptive spatial patterns that were not part of the objectives emerge from the interaction of the agents with local

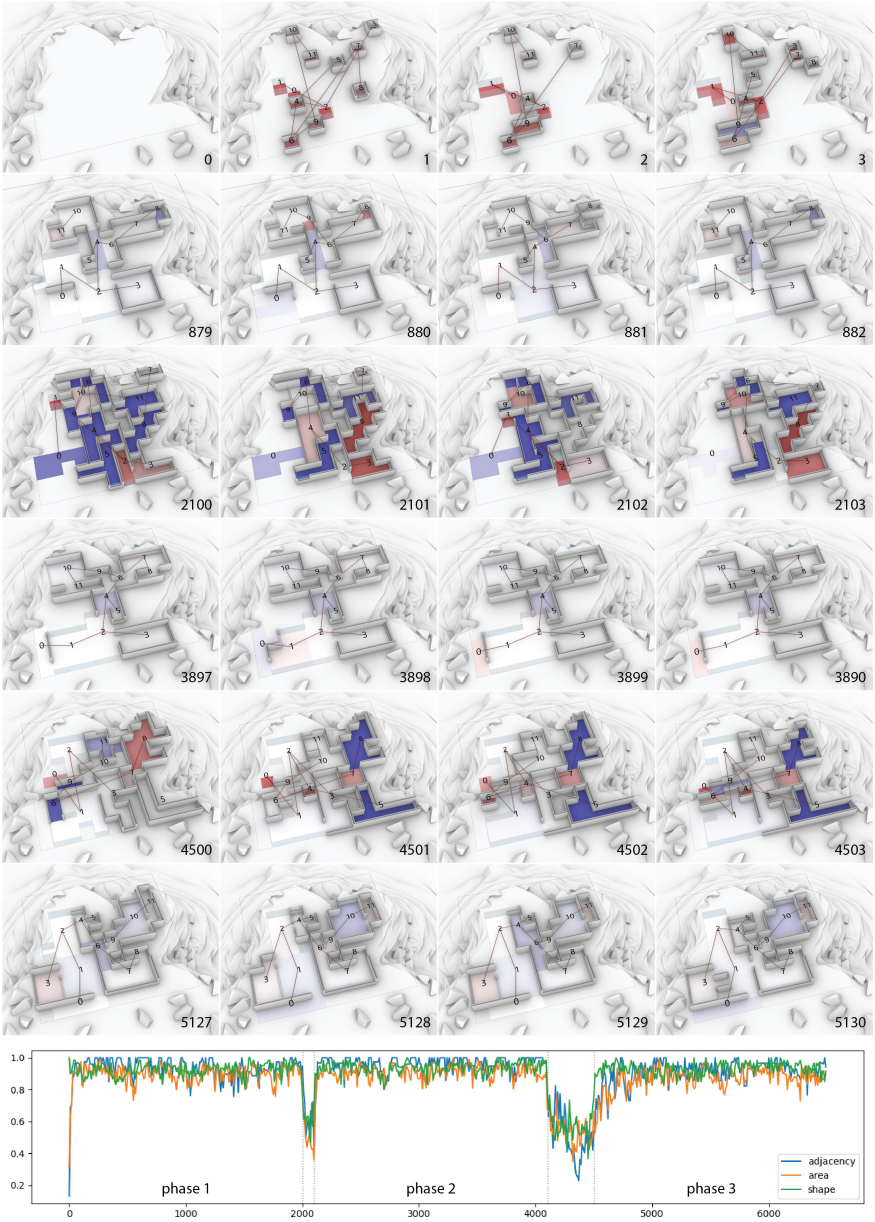
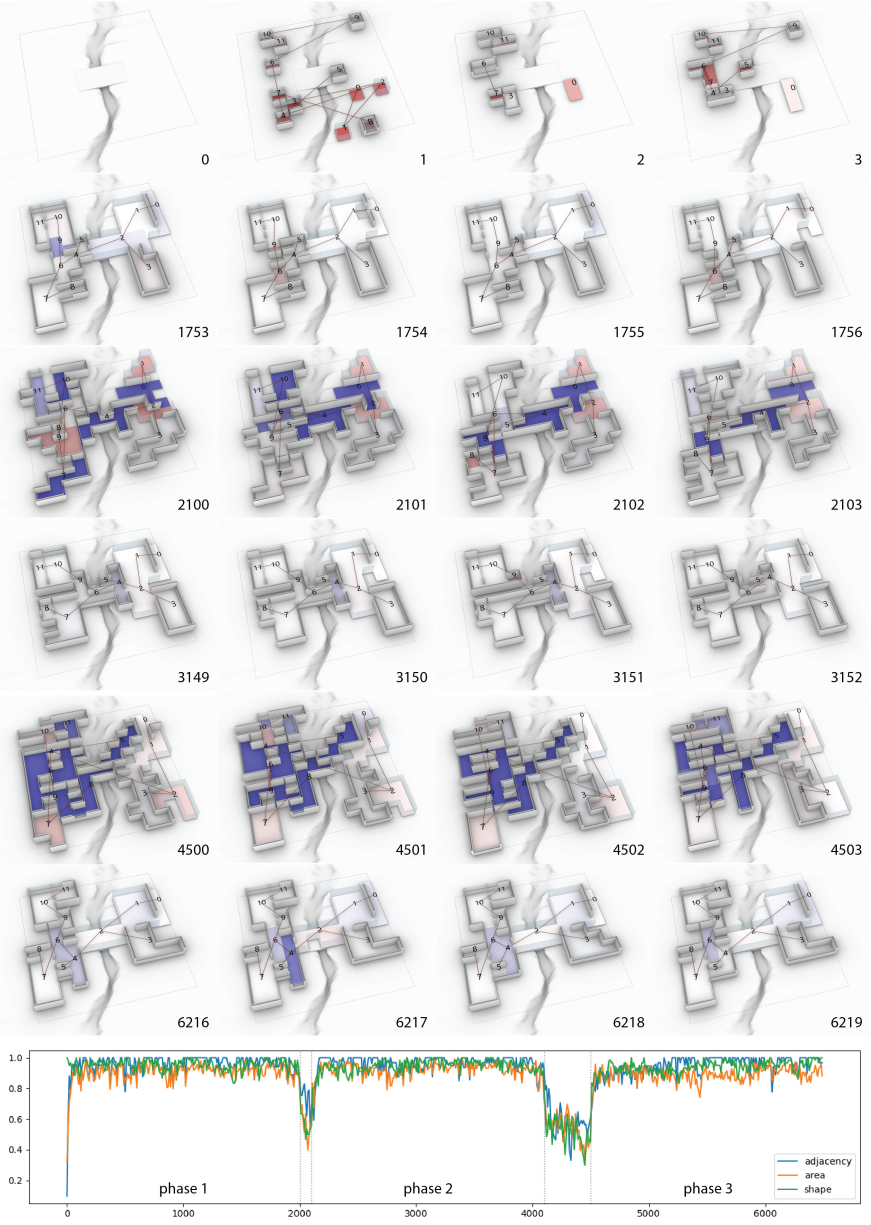


Figure 4
Rows 1-6: simulation of 12 agents in a terrain on the top of a hill with irregular boundaries. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. Row 7: graph with the average performance of the agents (y axis) over the episode (x axis). The areas between dotted lines indicate the periods where action was selected randomly (perturbation).

Figure 5
Rows 1-6: simulation of 12 agents in a terrain with a bridge over a stream. Every pair of rows contain the first 4 steps in a stage of the simulation and 4 steps selected visually. Row 7: graph with the average performance of the agents (y axis) over the episode (x axis). The areas between dotted lines indicate the periods where action was selected randomly (perturbation).



situations. For example, an agent created a protuberance to increase the area and to adapt to local obstacles (Figure 4, patio at 3898 or kitchen at 5128) or a small courtyard emerged between the sectors of the house (Figure 5, at 6216, and 6219).

CONCLUSION

In this paper, we introduced a workflow to train agents for spatial diagramming and planning. The agents successfully learned how to generate spaces, addressed specific spatial goals, preserved fine-grained interaction with the representation and were able to react to perturbations in the simulation. This algorithm will be improved and integrated to a game engine, so designers can use custom tools to interact with the model by changing parameters of the simulation and the PnH configurations of the environment and agents in real-time. By promoting the conversation between computational agents and the designer, we expect to support improvisation and cyclic reasoning in generative design.

ACKNOWLEDGMENT

This research was supported in part by funding from the Carnegie Mellon University Frank-Ratchye Fund for Art @ the Frontier as well as a PhD scholarship granted by the Brazilian National Council for Scientific and Technological Development (CNPq).

REFERENCES

Araghi, SK and Stouffs, R 2015, 'Exploring cellular automata for high density residential building form generation', *Automation in construction*, 49, pp. 152-162

van Hassel, H, Guez, A and Silver, D 2016 'Deep Reinforcement Learning with Double Q-learning', *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, Phoenix, pp. 2094-2100

Herr, CM and Ford, RC 2015 'Adapting Cellular Automata as Architectural Design Tools', *Emerging Experience in Past, Present and Future of Digital Architecture: Proceedings of the 20th CAADRIA conference*, Daegu, pp. 169-178

Hosmer, T and Tigas, P 2019 'Deep Reinforcement Learning for Autonomous Robotic Tensegrity', *Ubiquity and Autonomy: Proceedings of the 39th ACADIA conference*, Austin, pp. 16-29

Jabi, W, Chatzivasileiadi, A, Wardhana, NM, Lannon, S and Aish, R 2019 'The synergy of non-manifold topology and reinforcement learning for fire egress', *Architecture in the age of the 4th Industrial Revolution: Proceedings of the XXXVII eCAADe and XXIII SIGraDi Conference*, Porto, pp. 85-94

Koenig, R 2011, 'Generating Urban Structures: a Method for Urban Planning Supported by Multi-Agent Systems and Cellular Automata', *Przestrzen I Forma*, -(16), pp. 353-376

Narahara, T 2017 'Collective Construction Modeling and Machine Learning: Potential for Architectural Design', *Sharing Computational Knowledge!: Proceedings of the 35th eCAADe Conference*, Rome, pp. 593-600

Nguyen, TT, Nguyen, ND and Nahavandi, S 2018, 'Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications', *arXiv:1812.11794 [cs, stat]*, -, p. -

Ruiz-Montiel, M, Boned, J, Gavilanes, J, Jiménez, E, Mandow, L and Pérez-de-la-Cruz, JL 2013, 'Design with shape grammars and reinforcement learning', *Advanced Engineering Informatics*, 27(2), pp. 230-245

Smith, SI and Lasch, C 2016 'Machine Learning Integration for Adaptive Building Envelopes: An Experimental Framework for Intelligent Adaptive Control', *Proceedings of the 36th ACADIA Conference: Posthumans Frontiers*, Ann Arbor, pp. 98-105

Sutton, RS and Barto, AG 2018, *Reinforcement learning: An introduction*, The MIT Press, Cambridge

Veloso, P and Krishnamurti, R (in press) 'Self-learning Agents for Spatial Synthesis', *Proceedings of the 5th International Symposium on Formal Methods in Architecture (5FMA)*

Veloso, P, Rhee, J and Krishnamurti, R 2019 'Multi-agent Space Planning: a Literature Review (2008-2017)', *Hello, Culture!: Proceedings of 18th CAAD Futures conference*, Daejeon, Korea, pp. 52-74

Wilensky, U and Rand, W 2015, *An Introduction to Agent-based Modeling: modeling natural, societal and engineered complex systems with netlogo.*, The MIT Press, Cambridge

Xu, T, Wang, D, Yang, M, You, X and Huang, W 2018 'An Evolving Built Environment Prototype', *Learning, Adapting and Prototyping: Proceedings of the 23rd CAADRIA conference*, Beijing, pp. 207-215