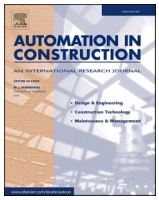
Automation in Construction 154 (2023) 104997

Contents lists available at ScienceDirect

Automation in Construction

journal homepage: www.elsevier.com/locate/autcon



Spatial synthesis for architectural design as an interactive simulation with multiple agents

Pedro Veloso [a,*], Ramesh Krishnamurti [b]

[a] Fay Jones School of Architecture and Design, University of Arkansas, USA
[b] School of Architecture, Carnegie Mellon University, USA

ARTICLE INFO

Keywords:
Spatial synthesis
Interactive simulation
Artificial intelligence
Agent-based modeling
Multi-agent deep reinforcement learning

ABSTRACT

Motivated by reflection-in-action in architectural design, this article introduces a spatial synthesis artifact that relies on multi-agent reinforcement learning to address spatial goals with fine-grained control in a simulation. It relies on parameter sharing with proximal policy optimization and a parameterized reward function to train robust agent policies in random environments with random spatial problems. The agents are evaluated in three design cases: a house design with 12 agents in three sites, a museum with 18 agents in an interstitial urban site, and a speculative design of a housing complex with 96 agents on a large empty site. The policies performed well in all the cases and produced morphologically consistent solutions. However, in cases with a larger number of agents, the system largely benefited from a spring layout algorithm for the initialization. Future research will address more complex spatial synthesis problems and mechanisms for human-computer interaction.

1. Introduction

In computer-aided architectural design (CAAD), the terms spatial synthesis ([15], 425), automated facility layout ([13], 197), and space planning ([4], 242) refer to the problem of generating architectural spaces that comply with specific objectives and constraints using computational methods. Spatial synthesis in CAAD mainly relies on satisficers and optimizers [8,12,13,15].

Satisficers require the creation of design rules or transformations that structure an implicit design space with derivational dependencies to produce spatial configurations incrementally ([15], 453–68) in the very large space of architectural problems ([1], 422). As the search tree grows exponentially with the branching factor as the base and the depth as the exponent, spatial synthesis algorithms must constrain the search for spatial configurations to restricted portions using different control strategies ([1], 435–40).

Current optimizers rely on black box methods, such as meta-heuristics, direct search, and model-based methods, to explore designs in a bounded parametric space without the need for expressing the design problem in analytical form [35]. Designers interact with optimizers either by controlling input parameters related to the problem and to the algorithm and then visualizing catalogues and dashboards with correlations between design alternatives and their respective performance [11,23,34].

Satisficers and optimizers require anticipating knowledge about the problem, the consequences of the decision making, and the forms of evaluation as collections of facts and rules. Besides, due to the large size of architectural layout problems, they are typically solved by computational agents with sophisticated strategies to search for good design solutions isolated from design interaction and without real-time guidance. This is particularly problematic in architectural design, where thinking, perception, and action are coupled in the circumstance of the design task, providing opportunities for ad hoc responses to unforeseen contingencies [19,20]. Design problem and solution co-evolve as a conversation between architects and the design media, such as drawings or physical models. The progressive actions over a design model provide feedbacks for the designers, which can influence their ideas and establish a session of reflection-in-action [19]. In this process, "(…) the designer constructs the design world within which he/she sets the dimensions of his/her problem space, and invents the moves by which he/she attempts to find solutions" ([20], 11). Not surprisingly, classic spatial synthesis methods have been criticized by the lack of support for design knowledge and interaction between problem solving and problem definition ([8], 182) and by its emphasis on an engineering approach to architectural problems ([14], 158–59).

Motivated by the creation of interactive spatial synthesis methods for

* Corresponding author.
E-mail addresses: pveloso@uark.edu (P. Veloso), ramesh@andrew.cmu.edu (R. Krishnamurti).

https://doi.org/10.1016/j.autcon.2023.104997
Received 12 July 2022; Received in revised form 20 May 2023; Accepted 15 June 2023
Available online 12 July 2023

reflection-in-action, this research proposes an alternative approach centered on fine-grained interaction in a sequential decision structure. Fine-grained interaction refers to the increase of speed and granularity of the turn-taking in order to enable humans and computational agents to be continuously active and responsive to unforeseen situations in a shared task ([27], 685–86). However, seamlessly interacting with designers in spatial synthesis is not trivial, because it requires responding to real-time updates of the environment based on human behavior, which makes the task environment non-stationary and dynamic. To address this setting, our approach relies on a game to be solved by a multi-agent system in a simulation, which can be shared with human designers. Interactive simulations with agents support fast and fine-granular turn-taking and enable the sequential generation of spatial configurations in large state spaces under uncertainty. For that, they rely on computational agents that can handle multiple spatial objectives and conflicts in real-time, and that are resilient to variations in the initial setup, order of operations, and control strategies.

In this article, we prove the feasibility of this unorthodox spatial synthesis approach by presenting the formulation, training, and evaluation of a design construct. The formulation relies on ideas from agent-based modeling to create cellular agents that collaborate and occupy cells in a shared environment grid, forming spatial partitions shaped as polyominoes without holes (*PnH*). Multi-agent deep reinforcement learning (MADRL) is used to train an ecology of these agents in random spatial problems using a series of parameterized reward functions. The evaluation is based on three distinct design cases: a house design, a museum in an interstitial urban site, and a speculative house complex on a large empty site. To the extent of our knowledge, this research resulted in the first successful design artifacts that address the problem of interactive spatial synthesis in CAAD with MADRL.

2. Background

2.1. Agent-based modeling and multi-agent spatial synthesis

Agent-based computing encompasses decentralized computational methods based on a collection of agents, which are autonomous computational entities that have sensors, actuators, and internal programs that allow them to interact locally in a shared environment. The domain of agent-based computing comprehends many areas of study, such as computer science, life sciences, ecological sciences and social sciences ([16], 479–80).

In the field of Artificial Intelligence (AI), agents are computational constructs that operate autonomously to solve a problem. Multiagent task environments require multiple computational agents that can compete and/or collaborate to solve a task ([18], 43–45). In Multi-Agent Systems (MAS), agents are defined both by their capacity for autonomous decision-making and for interacting with other agents and engaging in analogues of social activities ([33], 4–5). AI and MAS include algorithms for distributed constraint-satisfaction problems, distributed optimization, multiagent learning, and topics such as game theory, communication, and social choice [22].

In Agent-based modeling (ABM), agents are algorithmic descriptions of entities in a shared environment that can take local decisions based on the perception of its neighborhood, and they are used as a modeling methodology to capture the changing dynamics of complex systems ([32], 203–76). As the computation typically happens in the local representation of the agents during the simulation, it avoids the exponential joint action spaces of a centralized controller, and it enables a variable and large number of agents during a simulation. Besides, ABM also supports the visualization of different behaviors and spatial patterns over time and under uncertainty. It is used to capture the arising of novel and coherent patterns on the macro-level of the system from the interaction of its constituent elements at the micro-level – i.e. emergence ([9], 115–24; [3], 3; [32], 6). These properties make ABM a strong candidate as a method that can both increase the speed and granularity of turn-taking and support the exploration of novelty in spatial synthesis.

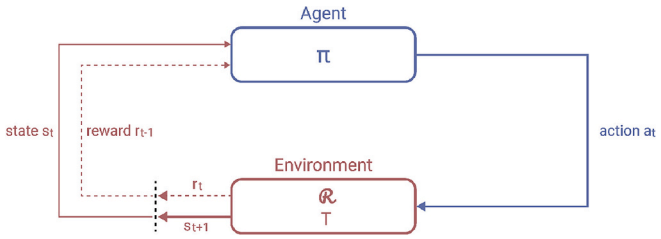


Fig. 1. The agent-environment interaction in the case of a MDP — based on ([25], 48). Notice that the time index of the reward is aligned with the action that generated it and not with the state produced, which simplifies the notation for the trajectory and for the description of the equations and algorithms.

Despite the growth of application of ABM and simulation in architecture ([24], 2), they are barely present in the general literature reviews on spatial synthesis in CAAD [10,13–15,30]. Still, Veloso, Rhee, and Krishnamurti [29] identify a series of research prototypes in CAAD that use ABM and simulation to generate spatial configurations, which they refer to as multi-agent space planning (MASP).

The human-computer interaction in MASP can potentially happen in a real-time simulation, at every step of the generation of the design alternatives and can support the exploration of emerging spatial patterns from the interaction of multiple agents. However, that relies on typical properties of ABM, such as the restriction of the agent's perception to the neighborhood information and the limited access to the model of the environment by the agent's policy. As a result, it is not trivial to design computational agents that both operate in real-time and follow a local policy capable of handling multiple spatial objectives and conflicts. To address this trade-off between operating in a simulation and the satisfaction of multiple realistic goals, such as adjacency, area, room shapes, and daylight access, researchers have explored hybrid techniques. These rely on the use of ABM and simulation to define general diagrams of the spatial organization and other techniques, such as metaheuristics [2,6] or generative adversarial networks [17,28], to convert them into more refined layouts.

2.2. Reinforcement Learning for training spatial agents

To develop an approach that relies on the behavior of multiple agents to respond to realistic layout goals and constraints, we focus on reinforcement learning (RL), a branch of machine learning that addresses sequential decision problems by training an agent to maximize the expected cumulative reward from its interaction with the environment (see Fig. 1). In RL, the decision making is idealized and modeled as a Markov Decision Process (MDP), so the probability distributions that define the interactions at a given timestep are dependent only on the current state and actions ([25], 49).

The RL agent is the learner and decision-making entity encoded into a behavioral function commonly known as a policy $\pi(a_t|s_t)$, which maps the current state s_t to the probability of selecting an action a_t. The environment $(S, A, T, \mathscr{R})$ comprehends everything that is external to the agent, which is defined by a state in the state space ($s \in S$) that can be influenced by the actions of the agents in the action space ($a \in A$). It uses a transition function $T(s_{t+1}|s_t, a_t)$ to map the action taken by the agent at a certain time and state, to the probability of reaching a new state. With the new state defined, the reward function $\mathscr{R}(s_t, a_t, s_{t+1})$ maps both states and the action of the agent to a certain reward signal r_t.

In large state spaces or in the face of partial information about s, it is useful to rely on function approximators, such as deep neural networks, to represent the agent policy. In this case, instead of explicitly storing the states in a table to support decision making, it is only necessary to store the description of the neural network and to manipulate a fixed-sized

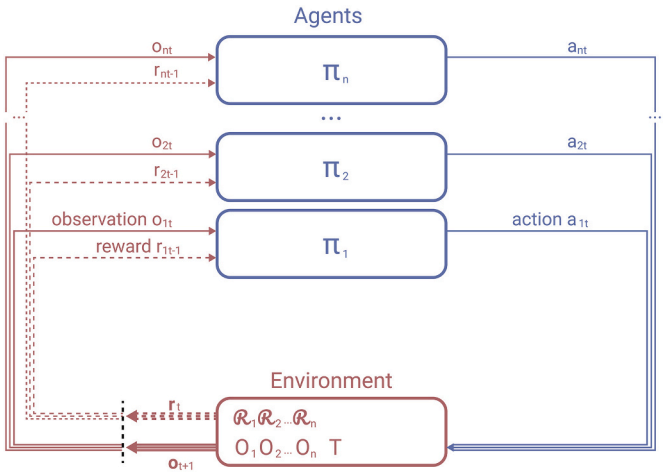


Fig. 2. The agent-environment interaction in the prototype as a variation of a partially observable stochastic game.

vector $\theta \in \mathbb{R}^d$ with its learnable parameters. Neural networks are also good for generalization, so with a good distribution of samples, it can be trained to learn a hierarchy of features and generalize the knowledge acquired to unknown situations ([25], 197).

Besides using deep neural networks, the proposed formulation of spatial synthesis also requires specific types of environments and training algorithms for multiple agents, which are part of the sub-field of multi-agent deep reinforcement learning (MADRL). This research focuses on the parameter sharing approach, where all the agents share the same individual policy [7,26]. To ensure that a single policy can express a diversity of actions and agent types, the information in the observation is customized for the agent, which can include part of its own internal state, its id, or even global information. Parameter sharing is a simple approach that enables the use of single RL algorithms with minimal modifications. The use of the shared policy enables training it with the data from all the agents, provides some level of centralization in the learning that can potentially mitigate non-stationarity, accelerate training [26], and lead to emergent cooperative behavior [7].

3. Formulation of the prototype

In this section we present the design and implementation of a design artifact based on ABM and MADRL that supports real-time, sequential, and granular construction of the design alternative with reduced assumptions about setup and order of the solution procedure. The problem is defined as a sequential decision process of an ecosystem of N agents sharing the same policy, reward function, and environment. Formally, this decision model is a variation of a partially observable stochastic game defined by the tuple $(S, N, A, T, \Omega, O_i, \mathscr{R}_i)$, where:

- S is the set of possible states for the environment
- N is the number of agents
- A is the action space for the agents
- T is the transition function
- Ω is the observation space for the agents
- O_i is the observation function that is customized to the different agents
- $\mathscr{R}_i$ is the reward function that is customized to the different agents

The multiple agents observe, act in the environment together, and receive individual observation and reward signals (see Fig. 2). Notice that instead of separate reward functions, there is a single function that is parameterized to support heterogeneous goals and behavior. This approach assumes that a single policy and a single reward function are

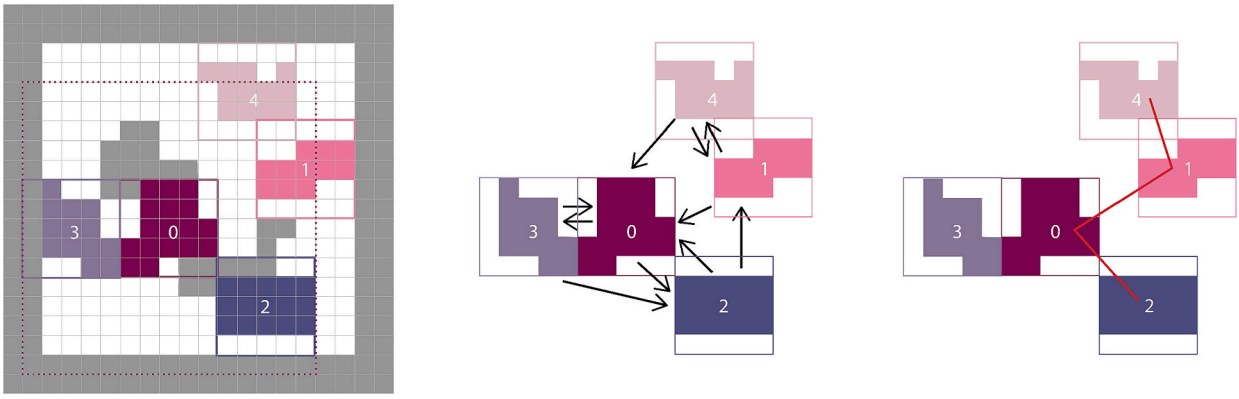


Fig. 3. Left: simplified visualization of agents in the grid with obstacles. The squares with solid lines around the agents represent the boundaries of their action grids and the square with dashed lines represents the boundary of the observation grid of agent 0. Middle: closest neighbors with k = 2: agent 0: [agent 3 and agent 2], agent 1: [agent 4 and agent 0], agent 2: [agent 0 and agent 1], agent 3: [agent 0 and agent 2], agent 4: [agent 1 and agent 0]. Right: example of adjacency goals with l = 2: agent 0: [agent 1, agent 2], agent 1: [agent 0, agent 4], agent 2: [agent 0], agent 3: [], agent 4: [agent 1]. By Author.

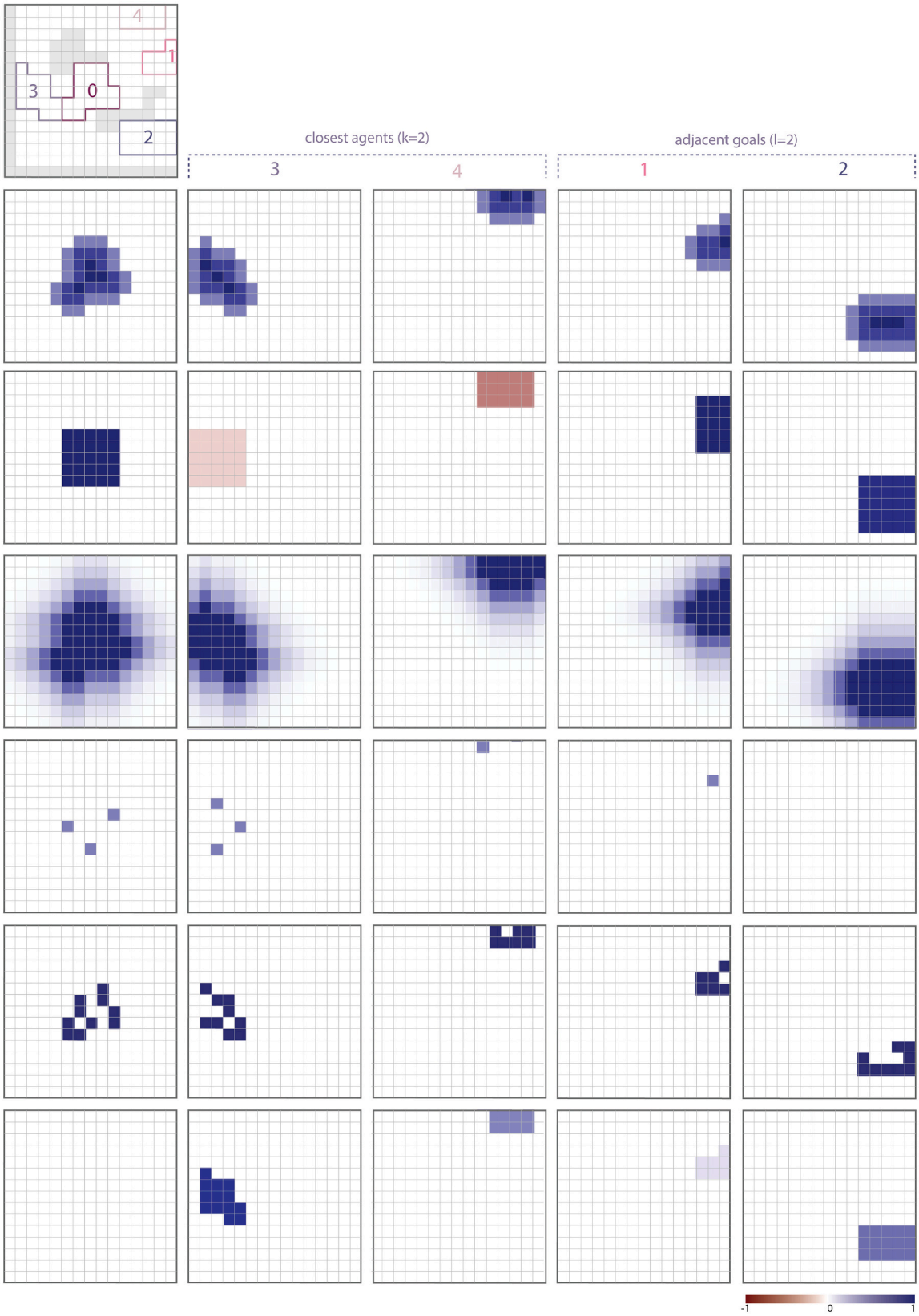


Fig. 4. Observation for agent 0 with action grid of shape $(5, 5)$, observation grid of shape $(15, 15)$, $k = 2$, and $l = 2$. Row 1: grid with obstacles (agents are added just for visualization). The following rows are based on the l agents that share a relationship of adjacency with agent 0 (1, and 2) and its k closest neighbors that do not share a relationship of adjacency (3 and 4). Row 2: classification of the cells: structure (dark blue), surface (medium blue), and legal offset (light blue); row 3: score of the agent with the respect to the area over the action grid; row 4: soft adjacency values based on Manhattan distance; row 5:indication of folding cells: L-folds (light blue), U-folds (dark blue); row 6: indication of the cells with daylight access; row 7: the current utility of the agent represented in the internal cells of the agent. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

enough to model the range of behaviors in the system.

3.1. Spatial and topological information

Each agent is a polyomino with no holes (*PnH*) forming a connected set of cells that represents a spatial boundary (see Fig. 3). The *PnH* discretizes different geometric entities into a homogeneous representation, which can approximate any polygon, such as conventional room shapes (rectangles, L-shapes, U-shapes, T-shapes, etc.), unconventional and irregular shapes, or even shapes with free form. These agents and the shared environment are represented spatially in a state space $S \subset \mathbb{R}^{w_{env} \times h_{env} \times d_{env}}$ that contains overlapping grids of information that are stored in multi-dimensional arrays of shape $(w_{env}, h_{env}, d_{env})$. While (w_{env}, h_{env}) defines the spatial dimensions of each information grid, d_{env} defines the number of grids stored in the environment.

The basic environment layer contains cells in two states: empty or obstacle. That allows the representation of existing buildings, elements of the landscape, or other elements defined by designers as spatial constraints. Besides, all the information that is relevant for the agents can be discretized and stored in additional layers, increasing d_{env}. The agent layers contain the state information of the agent expressing the configuration and the goals of the agent (see the following sections for further description) to facilitate learning. In the design artifact, $d_{agent} = 6$, which means that each agent has 6 layers:

- the classification of the cells based on the *PnH*
- the score of the agent with the respect to the area
- the soft adjacency values based on Manhattan distance from its body
- the indication of the cells on its concave corners – i.e., L- and U-folds
- the indication of cells with daylight access
- the current utility of the agent represented in the internal cells of the agent.

In terms of topological information, each agent i shares a relationship of adjacency to a maximum of l agents defined by the spatial synthesis problem and has a relationship of proximity with the k-closest and non-adjacent neighbors, which are dynamically defined during the simulation (see example in Fig. 3 and Fig. 4 with $k = 2$, $l = 2$).

3.2. Agent action

The basic actions available for the agents are single-cell expansion and retraction inside an action grid. The action grid is located on the current centroid of the PnH_i and defines the boundaries for the action space (Fig. 3 – Left). Therefore, as the agent expands the PnH_i into a certain direction, it also moves the centroid and, consequently, the action grid. The shape of the action grid (w_{act}, h_{act}) plus the option of not taking any action define the maximum size of an agents' discrete action space as $w_{act} \times h_{act} + 1$.

Inside the action grid, the action space is defined by the cells that are not blocked by an obstacle of the environment and that preserve the *PnH*ness of all the agents in the simulation if they are selected by the agent i. For the latter, the cells are classified into four types:

- legal expansion: cells in the von Neumann neighborhood of the *PnH* that if expanded, will not create a hole in the *PnH*.
- illegal expansion: cells outside of the von Neumann neighborhood of the *PnH* or cells in the von Neumann neighborhood of the *PnH* that if expanded, will create a hole in the *PnH*.
- legal retraction / agent surface: internal cells that if retracted will not create a hole or divide the *PnH* into two separate parts.
- illegal retraction / agent structure: internal cells that if eliminated will create a hole or divide the *PnH* into two separate parts.

Cell expansion and retraction 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.

3.3. The transition and observation functions

The changes in the environment rely on the transition function $T(s_t, \boldsymbol{a}_t)$, which updates the environment configuration to a following state, given the previous state and all the agents' actions $\boldsymbol{a}_t$ at once. The use of synchronous/simultaneous plays simplifies the process of training a variable number of agents and reduces the pre-defined assumptions about the sequences of decision from the perspective of the user. To make T deterministic, internally, the environment engine sorts the agents based on their current performance (best agent moves first) and solves the actions sequentially based on the constraints of obstacles and *PnH*s.

Inspired by agent-based modeling, the information available for the agent i is managed by the observation function $O(s_{t+1}, i)$, which maps the updated environment and agents' layers into a partial observation o_i in the space $\Omega \subset \mathbb{R}^{w_{obs} \times h_{obs} \times d_{obs}}$. This agent-centric observation only depends on the current configuration of the neighborhood of agent i, to support a computational method that is adaptable to problems with different environment sizes and with variable number of agents in a simulation. More specifically, this observation o_i is represented by an array of shape $(w_{obs}, h_{obs}, d_{obs})$. The information in o_i is sliced by the agent's observation grid of shape (w_{obs}, h_{obs}), which is smaller than the environment grid and is centered at the centroid of the current PnH_i (Fig. 3 and Fig. 4). It is defined by a series of d_{obs} layers related to the environment, to the agent i, and to other inter-related agents, such as the l agents that share a relationship of adjacency with i and the k-closest and non-adjacent agents. Overall, d_{obs} consists of a stack with

- a layer with the environment obstacles
- d_{agent} layers with specific information about agent i (check first column of Fig. 4)
- $l \times d_{agent}$ layers with information about the l agents that share a relationship of adjacency with agent i
- $k \times d_{agent}$ layers with information about the k closest agents that do not share a relationship of adjacency with agent i

For the design construct, we used $w_{obs} = h_{obs} = 15$, $d_{agent} = 6$, $k = 4$, and $l = 3$, so the observation o_i is a multi-dimensional array of shape $(15, 15, 49)$. Fig. 4 shows a simplified example of the observation of an agent with an action grid of shape $(5, 5)$, observation grid of shape $(15, 15)$, $k = 2$, $l = 2$, and $d_{agent} = 6$.

3.4. Spatial objectives and reward function

The goals of the agents are modeled by functions f^j that return a scalar value in the unit interval representing the performance of the agent. These goal functions are parameterized by the agents so they can be dynamically modified during the generation of a solution to reframe the design problem on the fly. For each goal function f^j, there is a complementary spatial function g^j that provides a perceptual map related to the agent's performance w.r.t a goal j in the form of an array of size (w_{env}, h_{env}) — see Fig. 4: rows 3–6. These maps augment S to facilitate training a single policy that can address multiple goals. For the design construct, there are goal functions to indicate area, smooth adjacency, not-folding (stimulates shapes with few concave corners, such as rectangle, L or U shapes), and daylight access (stimulates agent cells adjacent to open spaces).

To provide a dense signal related to adjacency, the function g^{adj} labels the cells in the environment based on their distance to the PnH_i. The values are 1 for the cells that are inside or adjacent the PnH_i, then they

Table 1
Specification of two settings used in training.

		Setting 1	Setting 2
RL	RL algorithm	**PPO**	**PPO**
	ɣ	0.99	0.99
	buffer size	10,000	10,000
	value function weight	0.5	0.5
	entropy weight	0.01	0.01
	epsilon clip (ϵ in L_{clip})	0.3	0.3
	max grad. norm. (clipping)	0.5	0.5
	GAE λ	0.85	0.85
	reward normalization	1	1
	value clip	1	1
Environment	n agents in training	16	16
	n agents in test	4, 8, 12, 16, and 20	4, 8, 12, 16, and 20
	max l adjacencies	3	3
	max k closest neighbors	4	4
	action grid shape	(5, 5)	(5, 5)
	observation grid shape	(15, 15)	(15, 15)
	action space	Discrete (26,)	Discrete (26,)
	observation space	(15, 15, 49)	(15, 15, 49)
	board initialization shape	(8–31, 8–31)	(15–45, 15–45)
	board configuration in training	random obstacles	random carving
	board padding	(7, 7, 7, 7)	(7, 7, 7, 7) or (9, 9, 9, 9)
	steps per episode	128	256
Training	optimization algorithm	Adam	Adam
	learning model	CNN (shared) + FC (heads)	CNN (shared) + FC (heads)
	learning rate	0.0003	0.0003
	Loss function	PPO policy and value losses	PPO policy and value losses
	batch size	64	64
	n of training env.	64	32
	n of test env.	32	8
	epochs	300	200 (one with 300)
	episodes per epoch	512	512
	repeat per collect	2	2
	agent samples per epoch	262,144	262,144

decay over the Manhattan distance until they reach 0 at the maximum distance ($dist_{max}$) beyond the adjacent cells and on – see Fig. 4: row 4. The function f^{adj}_{min} evaluates the maximum values in g^{adj} that intersect with PnH_i for all the agents that share a relationship of adjacency with it (the set D_i). Then, it returns the minimum value in that set, which represents its worst performance in terms of adjacency.

$$f^{adj}_{min}(i, dist_{max}, c) = \min_{d \in D_i}\left(max\left(g^{adj}(d, dist_{max}, c) \bigcap PnH_i\right)\right) \quad (1)$$

$$g^{adj}(i, dist_{max}, c) = \left(\frac{dist_{max} - \text{clip}(\text{MDistMap}(PnH_i) - 1, 0, dist_{max})}{dist_{max}}\right)^c \quad (2)$$

The function f^{area} is a piecewise linear function based on the relationship between the agent area and its target area. It returns 1 when the agent reaches the target area. This value linearly decreases when the area of the agent moves away from the target. It becomes 0 when its area is 0 or when it is twice or more than the target area. The spatial representation g^{area} labels the cells of the action grid with the ratio between the agent and target area shifted by one and bounded by 1. It labels the action grid with −1 when the agent has area 0, 0 when it reaches the target, and 1 when it has twice the target area or more.

$$f^{area}(i) = \begin{cases} \frac{area_i}{target_i} & if\ area_i \le target_i \\ 2 - \frac{area_i}{target_i} & else\ if\ area_i \le 2\ target_i \\ 0 & else \end{cases} \quad (3)$$

$$g^{area}(i) = ActionGrid_i min\left(\frac{area_i}{target_i} - 1, 1\right) \quad (4)$$

To prevent irregular shapes, the spatial function g^{fold} assigns the value of $1/fold_{max}$ to the concave corners of the agent's PnH_i that are surrounded by two adjacent edges (a L-fold) and assigns $2/fold_{max}$ to the ones with three adjacent cell edges (U-fold) – see Fig. 4: Row 5. The parameter $fold_{max}$ is controlled by the designer to define how regular the shapes should be. The function f^{fold} returns a value of one minus the sum of all the values in g^{fold}, bounded by zero to prevent negative scores. The functions fold_L and fold_U are indicators that return a zero array of shape (w_{env}, h_{env}) with ones in the position of the respective folds.

$$f^{fold}(i) = max\left(1 - sum\left(g^{fold}(i)\right), 0\right) \quad (5)$$

$$g^{fold}(i, fold_{max}) = \frac{\text{fold}_L(\text{i})}{fold_{max}} + \frac{2\text{fold}_U(\text{i})}{fold_{max}} \quad (6)$$

Finally, for daylight, we assume that only the cells in the PnH_i that are adjacent to one or more empty cells in the environment have daylight access. Thus, the daylight function f^{lit} divides the percentage of agent cells that have daylight access by the target percentage and bounds the result by 1. $litcells(i)$ is an indicator function that returns a zero array of shape (w_{env}, h_{env}) with ones in the position of the daylit cells. g^{lit} returns a map with the daylit cells of an agent tagged with the value of f^{lit}.

$$f^{lit}(i, lit_{\%}) = min\left(\frac{sum(lit\_cells(i))}{area_i}, lit_{\%}\right) \Big/ lit_{\%} \quad (7)$$

$$g^{lit}(i, lit_{\%}) = litcells(i) f^{lit}(i, lit_{\%}) \quad (8)$$

The utility $f(s, i)$ is a function that combines a subset of these goal functions with a weighted average or with a multiplication of the terms to incentivize simultaneous exploration of goals. For each agent i, there is a parameterized reward function $\mathscr{R}(s_t, s_{t+1}, i)$ that maps the consecutive environment states and the agent information to dense scalar reward signals, to condition heterogeneous behaviors and accelerate training. $\mathscr{R}$ is defined as the difference between the utility values of two consecutive states according to the objectives and the agent parameters: $f(s_{t+1}, i) - f(s_t, i)$.

4. Training

In this design artifact, we train a shared policy to express the diversity of agent behaviors conditioned on custom agent observation and rewards. It is a simple approach that enables the use of model-free and single-agent RL algorithms with minimal modifications. The advantage of this approach is that it enables training the same policy network with the data from all the agents, providing a level of centralization in the learning.

Table 1 contains the specifications of the two iterations in the training setting of the design artifact. The next sections will describe the algorithm, model, initialization, and training results. To test the feasibility of developing custom agents with different spatial behaviors, the model in both settings were trained with various objectives (see Table 2).

4.1. Training algorithm

We use Proximal Policy Optimization (PPO, [21]) for training. PPO

Table 2
Utility functions used in the training. The rows in bold refer to the policies that were evaluated in this article.

	Name used in this article	Utility function	Setting 1	Setting 2
Goals	Adjacency and Area	$f_{min}^{adj} f^{area}$	X	X
	Adjacency, Area, and Not Folding (3)	$f_{min}^{adj} \frac{\left(f^{area} + f^{fold}\right)}{2}$ **with** $fold_{max} = 3$	**X**	
	Adjacency, Area, and Not Folding (5)	$f_{min}^{adj} \frac{\left(f^{area} + f^{fold}\right)}{2}$ **with** $fold_{max} = 5$	**X**	**X**
	Adjacency, Area, and Daylight (75%)	$f_{min}^{adj} \frac{\left(f^{area} + f^{lit}\right)}{2}$ **with** $lit_{75\%}$	**X**	
	Adjacency, Area, Not Folding (5), and Daylight (50%)	$f_{min}^{adj} \frac{\left(f^{area} + f^{fold} + f^{lit}\right)}{3}$ **with** $lit_{50\%}$		**X**
	Adjacency, Area, Not Folding (5), and Daylight (75%)	$f_{min}^{adj} \frac{\left(f^{area} + f^{fold} + f^{lit}\right)}{3}$ with $lit_{75\%}$	X	X

modifies the original objective J of an advantage actor-critic algorithm to overcome some problems of stability, such as variance and performance collapse, and makes the training more sample-efficient. It uses the relative performance identity – i.e., the difference in performance between two policies – to ensure non-negative improvement in a conservative policy iteration ([5], 165–94). In practice, this identity is approximated by using trajectories from the old policy adjusted with weights based on the ratio of action probabilities between the successive policies, which upweights the action that are more likely under the new policy. This creates a surrogate objective function based on importance sampling that guarantees monotonic improvement within an error bound.

Particularly, we opted for a PPO algorithm with a surrogate objective J_{π}^{CLIP}, which uses clipping inside the objective to limit the ratio to an ε-neighborhood to guarantee monotonic improvement (see Algorithm 1). The algorithm relies on Generalized Advantage Estimation (GAE), which uses a weighted average of the advantages for the different n-step returns to reduce its variance. There is also an entropy regularization term H that is subtracted from the loss function to increase exploration in the training by incentivizing policies with more uniform distributions. The base implementation of PPO is derived from the library Tianshou 0.3.0 [31], which was extended with custom code to support the proposed MADRL formulation with parameter sharing. Basically, this extension creates a multi-agent environment that handles transitions with arbitrary N actions, rewards, and state updates. Benefiting from the fact that agents share the same policy network, the multi-agent experience is collected from instances of this environment using different settings and is later merged into a large training batch with the experience of all individuals.

Algorithm 1. PPO-Clip with multiple environments – based on ([5], 177–78; [31]).

1: Initialize actor (θ_A) and critic (θ_C) networks with random weights
2: Initialize the buffer D, and parameters such as β, ε, and K
3: For epoch in K epochs do
4: For each training environment
5: Reset training environment
6: Collect an episode using θ_A and store in D
7: Compute $\pi'(a|s)$ with θ_A
8: Compute values V_{tar}^{π} with θ_C and trajectory
9: Compute advantages A_{GAE}^{π}
10: For each minibatch m
11: Compute new values V_{new}^{π} with θ_C
12: Compute $\pi(a|s)$ with θ_A
13: Compute the probability ratio $r(\theta_A)$ using $\pi'(a|s)$ and $\pi(a|s)$
14: Compute the entropy H with θ_A
15: $J^{CLIP}(\theta_A) = \mathbb{E}[\min(r(\theta)A_{GAE}^{\pi}, clip(r(\theta_A),1-\varepsilon,1+\varepsilon)A_{GAE}^{\pi})]$
16: $L_{pol}(\theta_A) = J^{CLIP}(\theta_A) - \beta H$
17: $L_{val}(\theta_C) = MSE(V_{new}^{\pi}, V_{tar}^{\pi})$
18: Update θ_A using $L_{pol}(\theta_A)$
19: Update θ_C using $L_{val}(\theta_C)$
20: Empty D
21: Test the policy during one episode for each test environment

4.2. Training model

The neural network used for training has a shared body (initial layers) and separate network heads (the final layers) for the actor and for the critic (see Table 3). This architecture allows the model to build a common representation based on the agent observation, then establishes custom features for their distinct tasks in the independent portion of the neural network.

4.3. Random initialization

To learn a robust and general policy, we implemented two methods of random initialization for training: random obstacles and random carving. The random obstacles algorithm populates the board with x-aligned and y-aligned obstacles (see Fig. 5). The random carving algorithm excavates an initial board full of obstacles by sampling cells using a factor proportional to the number of empty neighbors, which

Table 3
Architecture of the training model.

	Layer Type	Input shape	Output shape	Kernel	Stride	Padding	Activation
Shared Body	Convolution 2D	(b, 15, 15, 49)	(b, 15, 15, 64)	(3,3)	(1,1)	(1, 1)	ReLU
	Convolution 2D	(b, 15, 15, 64)	(b, 15, 15, 32)	(3, 3)	(1, 1)	(1, 1)	ReLU
	Convolution 2D	(b, 15, 15, 32)	(b, 15, 15, 16)	(3, 3)	(1, 1)	(1, 1)	ReLU
	Convolution 2D	(b, 15, 15, 16)	(b, 15, 15, 8)	(3, 3)	(1, 1)	(1, 1)	ReLU
	Convolution 2D	(b, 15, 15, 8)	(b, 15, 15, 4)	(3, 3)	(1, 1)	(1, 1)	ReLU
	Flatten	(b, 15, 15, 4)	(b, 900)	–	–	–	–
	Fully Connected	(b, 900)	(b, 256)	–	–	–	Linear
Critic Head	Fully Connected	(b, 256)	(b, 128)	–	–	–	ReLU
	Fully Connected	(b, 128)	(b, 64)	–	–	–	ReLU
	Fully Connected	(b, 64)	(b, 1)	–	–	–	Linear
Actor Head	Fully Connected	(b, 256)	(b, 128)	–	–	–	Relu
	Fully Connected	(b, 128)	(b, 26)	–	–	–	Linear

incentivizes cave-like shapes (see Fig. 6). After the obstacles are defined, the engine randomly initializes the areas, adjacency targets, layers, and initial positions for the agents, then runs several steps with random actions so they settle in a random initial configuration.

4.4. Training results

All the policies in setting 1 were trained for 300 epochs (see Fig. 7). Due to time restriction, the policy Adjacency, Area, Not Folding (5) was trained over 300 epochs but the other policies were trained over 200 epochs in the second setting (see Fig. 8).

5. Results

We evaluate if the proposed formulation and training can produce policies for reflex agents that address multiple spatial objectives in three unseen design cases: a single-family house to be implemented in different sites; a single-story complex of art galleries; a speculative complex of eight single-family houses. This evaluation comprehends a few goals. First, having an effective method to train agents to satisfy multiple spatial objectives in a simulation is still an open problem in multi-agent spatial synthesis. Secondly, it tests the capacity of the method to produce policies that support step-by-step formation of valid designs in face of adversity, such as new sites, programmatic conditions, or state perturbations by humans. While a policy trained with the data from a specific design case could perform better in the same case, policies that are robust to unseen situations and variations can be used in design tools because they reduce the need of training on-demand, and they allow designers to provide guidance. Finally, we also evaluate the morphological consequences of the different reward functions based on a visual analysis of the resulting configurations.

In this article, we focus on four policies:

- Adjacency, Area, and Not Folding (3) from setting 1
- Adjacency, Area, and Daylight (75) from setting 1
- Adjacency, Area, and Not Folding (5) from setting 2
- Adjacency, Area, Not Folding (5), and Daylight (50) from setting 2

To demonstrate the integration of the artifact with design workflows, the resulting configurations are post-processed in a parametric modeling editor, which can be controlled in real-time during a simulation. The cell configuration of the agents is the input of a parametric model that generates the geometry for visualization, with floors, walls, windows, and openings for doors. These are the basic steps of the model (Fig. 9):

A) Extracting polyominoes and other information from the environment
B) Converting the polyomino of each agent into a single mesh
C) Combining agents that represent a single space, based on user input
D) Extracting internal and external walls
E) Defining the rooms that do not have external walls, such as porches, based on user input
F) Defining the glass walls for the external walls of the rooms selected, based on user input
G) Defining the window positions based on a percentage of glazing and on a heuristic
H) Defining the door positions between rooms that share a relationship of adjacency by randomly selecting one of the cell edges in the shared walls
I) The resulting parametric model.

5.1. Design case 1

The first design case consists of a single-family house to be implemented in three different sites with obstacles (Fig. 10): (1) a site on the top of the hill, (2) a site with a central obstacle, (3) a site with two stream banks connected by a bridge. The program consists of 12 agents that represent the spaces of a two-bedroom house with a free floor plan (see Fig. 11).

This case was originally created in [36] to assess a previous research prototype, which was still limited to a fixed spatial representation of the environment. In this design construct it is used to evaluate the robustness and performance of a realistic policy: Adjacency, Area, and Not Folding (5). For each episode, the agents interact for 6500 steps, which are divided in three stages that are separated by two phases of perturbations:

- Initialization: empty agents start at random positions
- Stage 1 (0–1999): policy selects actions
- Perturbation 1 (2000–2099): random action selection for randomization.
- Stage 2 (2100–3099): policy selects actions
- Perturbation 2 (4100–4499): random action selection for randomization.
- Stage 3 (4500–6499): policy selects actions

In a visual and qualitative analysis, it is possible to observe that the agents developed a functional and consistent behavior (Fig. 12). The layouts are mostly mosaics of rectangular and quasi-rectangular shapes. The agents were also able to react to the constraints of the site. For example, on site 3, the bridge incentivized a separation of the social and private areas of the house to the two banks, mediated by the corridors in the center, which is beneficial for the adjacency requirements.

The policy was also evaluated 50 times in each of the environments. The results are on the following graphs (Fig. 13). Notice that on average, the agents converged to good results in all the three scenarios either starting from random initialization or from a random perturbation. There is a big improvement after a few iterations, and over the remaining iterations the agents keep refining the configuration and solving local problems. The average adjacency performance in the site

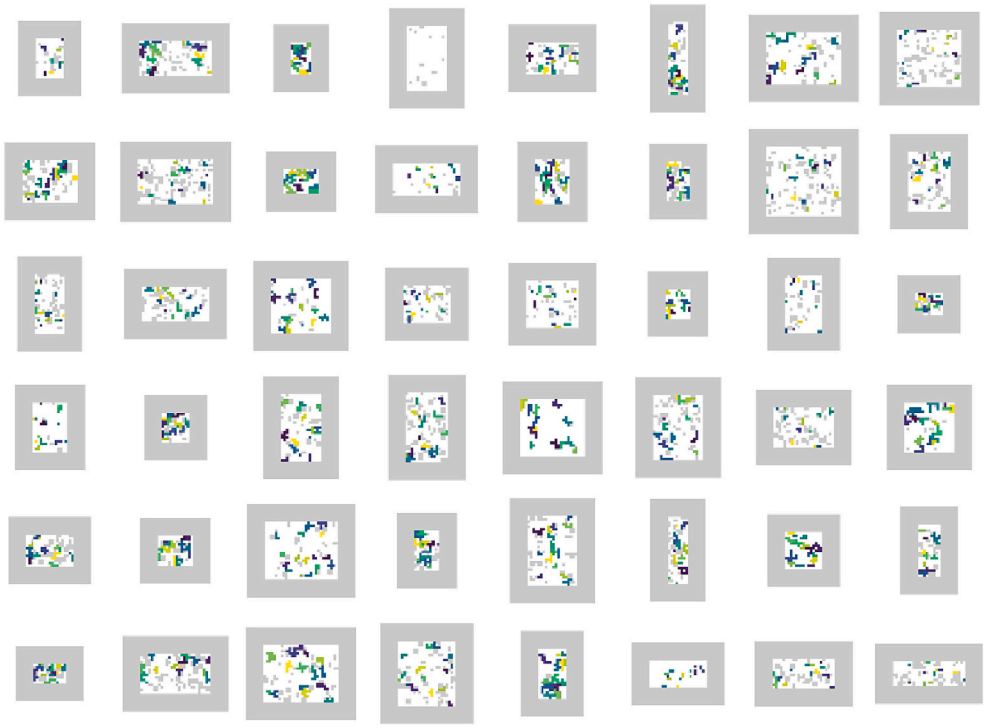


Fig. 5. 48 examples of agent initialization with random obstacles.

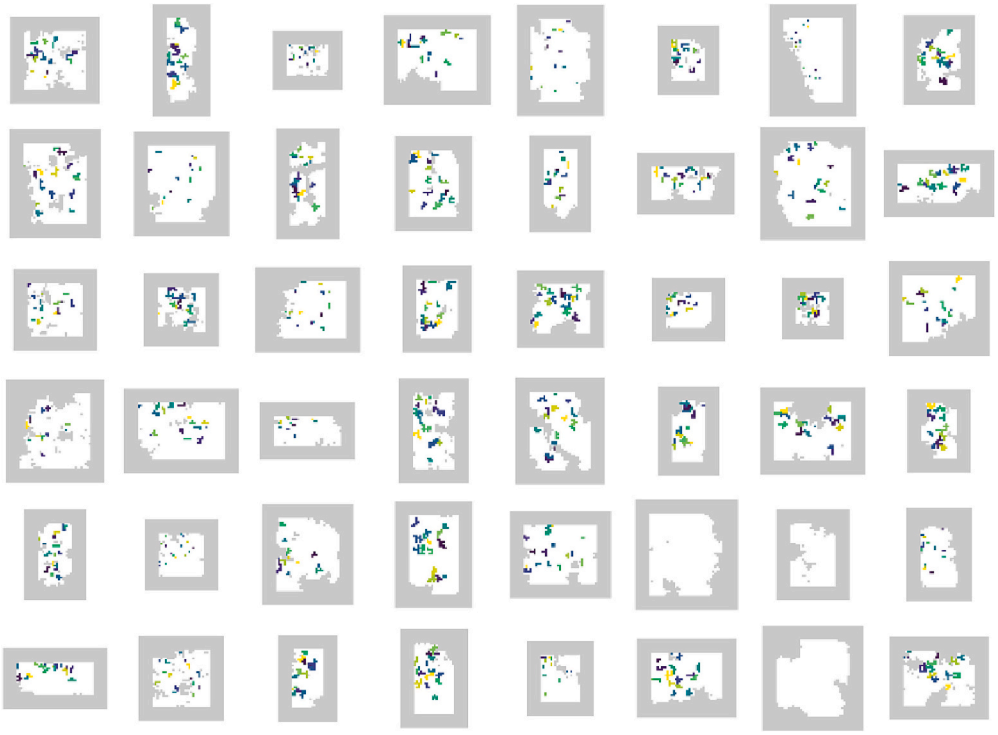


Fig. 6. 48 examples of agent initialization with random carving.

with central obstacles is the only one that drops below 0.9.

5.2. Design case 2

Design case 2 is a single-story complex of galleries for local and regional artists, with a focus on the promotion of visual arts. It was designed to test the capacity of generalization of the different policies. It consists of an art gallery complex that is modeled by 18 agents with more connections and a loop – see Fig. 14. The floorplan of the complex should have a fluid configuration, which provides opportunities to evaluate the morphological implications of the policies derived from different spatial goals.

The site is located between urban lots with one to two-story buildings and an existing park (see Fig. 15). It is defined by an irregular boundary

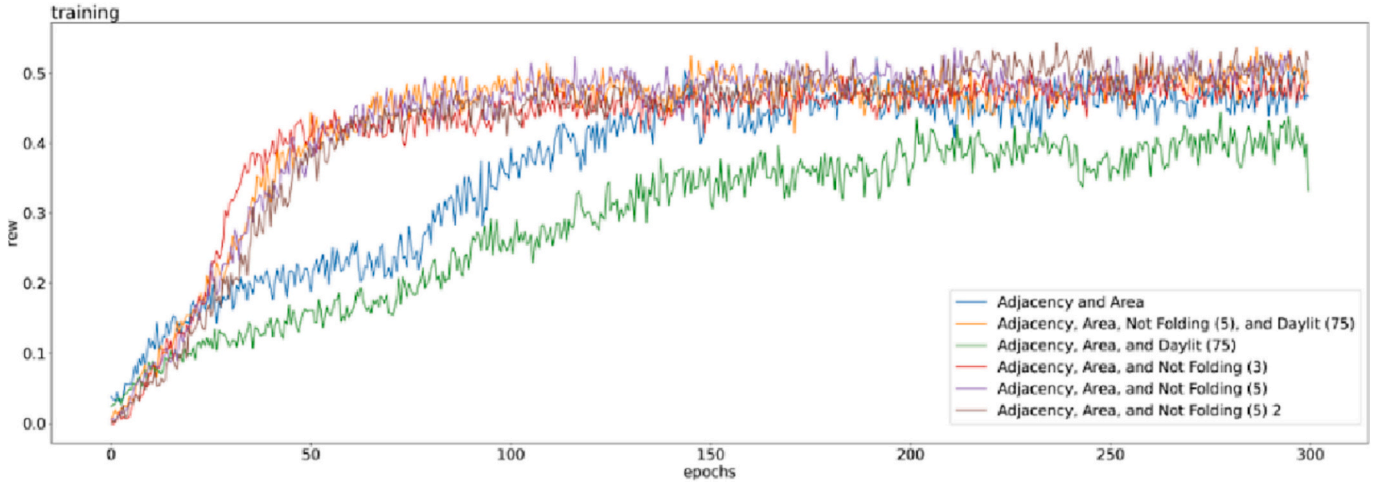


Fig. 7. Training graphs with training rewards for setting 1.

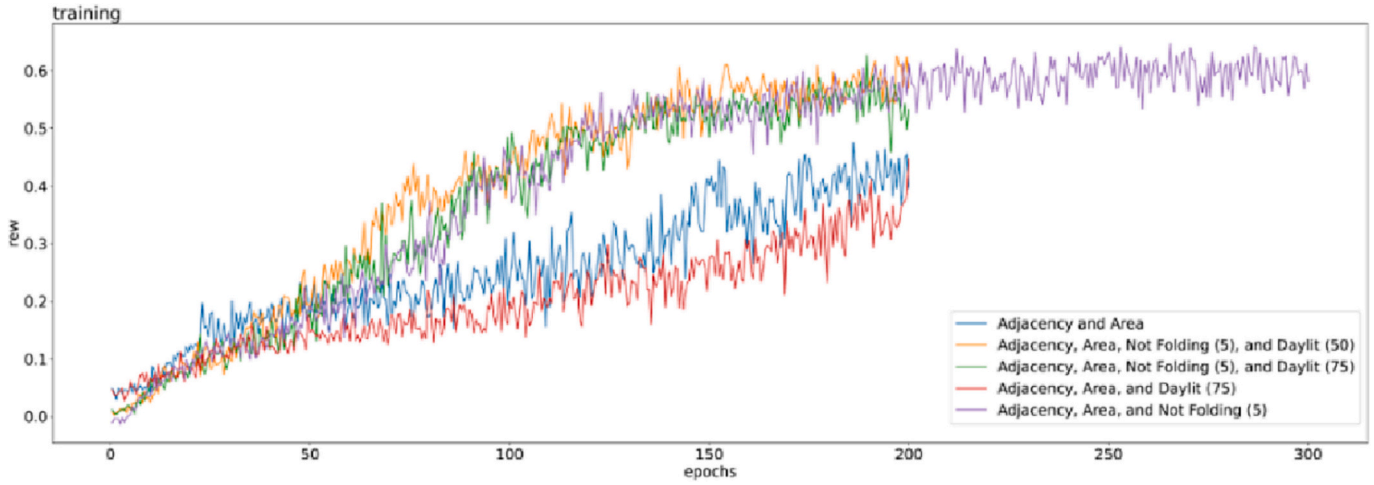


Fig. 8. Training graphs with training rewards for setting 2.

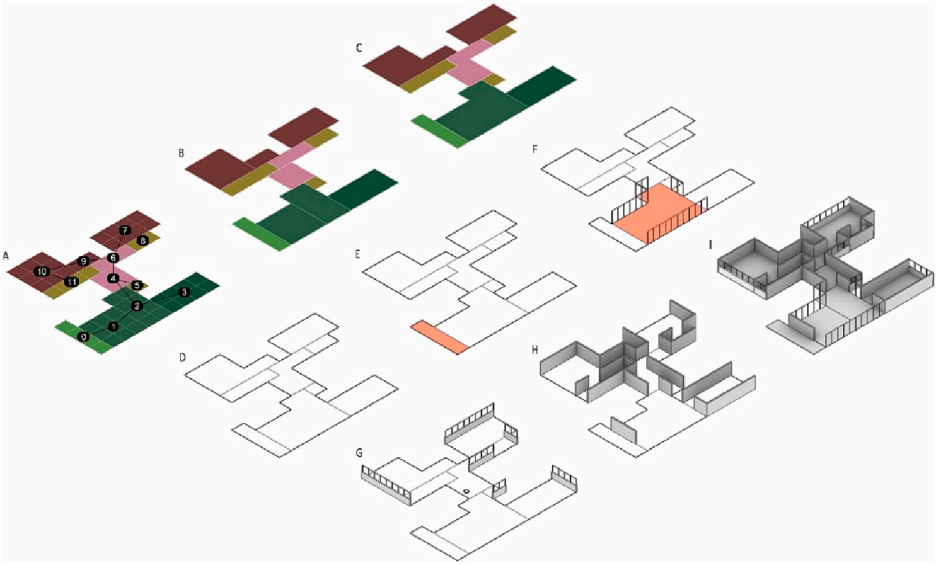


Fig. 9. Basic steps for the parametric model derived from the agent configuration.

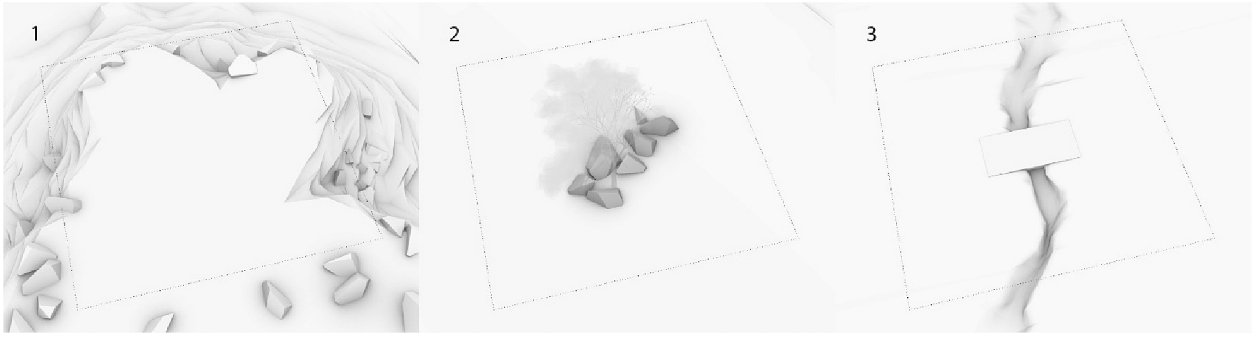


Fig. 10. Three different sites: (1) site on the top of the hill, (2) site with central obstacles, (3) site with stream and bridge.

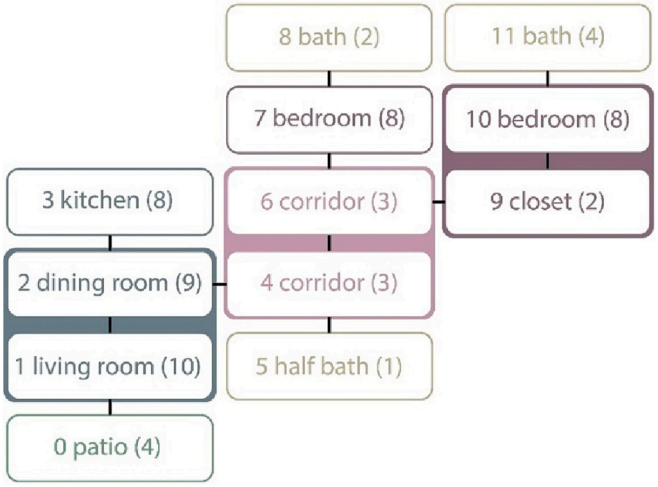


Fig. 11. Program of the house in case 1. The leading number is used to identify the spaces in the next images. The numbers in parenthesis indicate the target areas. The connections indicate adjacency. The colored grouping indicates spatial integration between spaces.

behind the buildings, fences, and many natural elements such as boulders or trees, some of which are also present inside the site and should be preserved.

Like in design case 1, we tested the performance of the agents over several steps to evaluate the flexibility, robustness, and convergence of the agent to good configurations under different conditions. However, considering the larger number of policies and the good performance of the training, the number of total steps and stages were reduced. The agents interact in the environment for 2200 steps, which are divided in two stages that are separated by a single perturbation:

- Initialization: empty agents start at random positions defined by a spring layout method using the adjacency graph of the program
- Stage 1 (0–999): policy selects the actions
- Perturbation 1 (1000–1199): random action selection
- Stage 2 (1200–2199): policy selects the actions

In the policy Adjacency, Area, and Not Folding (3), adding a single fold to a room shape has a large impact in the score. Therefore, the agents opted for generating rows of single cells, which eliminates folds in the formation process and reduces the distance between agents (see Fig. 16: left column). However, the action grid restricts the extension of the rows to 5 cells, which is smaller than all the target areas in the art gallery program, resulting in undersized rooms. The agents had good performance in terms of adjacency, not folding, and even daylight access, which was further improved in the early stage with the spring layout initialization (Fig. 18).

In the case of the policy Adjacency, Area, and Daylight (75%) there is a strong pressure to increase the external walls adjacent to open spaces. Thus, each agent tends to create irregular polyominoes with multiple concavities, which results in rooms formed by interconnection of corridors and niches (see Fig. 16: right column). Overall, this spatial pattern results in a maze-like floorplan with small external patios in between the agents.

Adjacency, Area, and Not Folding (5) resulted in arrangements of rectangular and L-shaped rooms. With the spring layout initialization, the agents are positioned further away from the neighbors, so they must move closer to resolve spatial conflicts and satisfy adjacency. Consequently, it results in a freeform layout that expands from the center (hall and patios) to centrifugal gallery and administrative blocks in the periphery, which results in a better performance in terms of adjacency (see Fig. 17: left column).

Adjacency, Area, Not Folding (5), and Daylight (50%) results in rectangular arrangements like the ones from Adjacency, Area, and Not Folding (5). However, the agents themselves organize a more disperse and centrifugal layout, which provides daylight access on the scale of the building arrangement (see Fig. 17: right column). It also minimizes the number of cells that are adjacent to the borders and obstacles of the site. Overall, the agents opt for having some flexibility in terms of area to improve the other metrics.

In terms of performance, the policies generally satisfy their target spatial goals in phase 1, where there is the support of the spring layout initialization. The exception is the area in the policy Adjacency, Area, and Not Folding (3), which opted for sub-optimal configurations with linear rooms, and Adjacency, Area, Not Folding (5), and Daylight (50) which allows the area to change to create more access to daylight. After the random perturbation, the policies achieve good configurations, but the overall performance is typically worse than in stage 1, which benefits from the spring layout algorithm.

5.3. Design case 3

Case 3 comprehends the speculative design of a large single-floor complex composed of 8 units with the same area and adjacency specification as in case 1. The complex occupies an empty and flat site of shape $(50, 40)$. In total, there are 96 agents in the simulation, which are initialized with the spring layout algorithms and run for 2000 steps. This was repeated 50 times for each policy. The goal of this case is to

- evaluate if the policy can generalize to a design case with many more agents than in the random training
- evaluate the possibility of the system to support the parallel design of multiple buildings on a site
- visualize morphological qualities that emerge from the interaction of many agents with the different policies

The morphological results are consistent with the previous case but there are some spatial patterns that emerged with the interaction of a

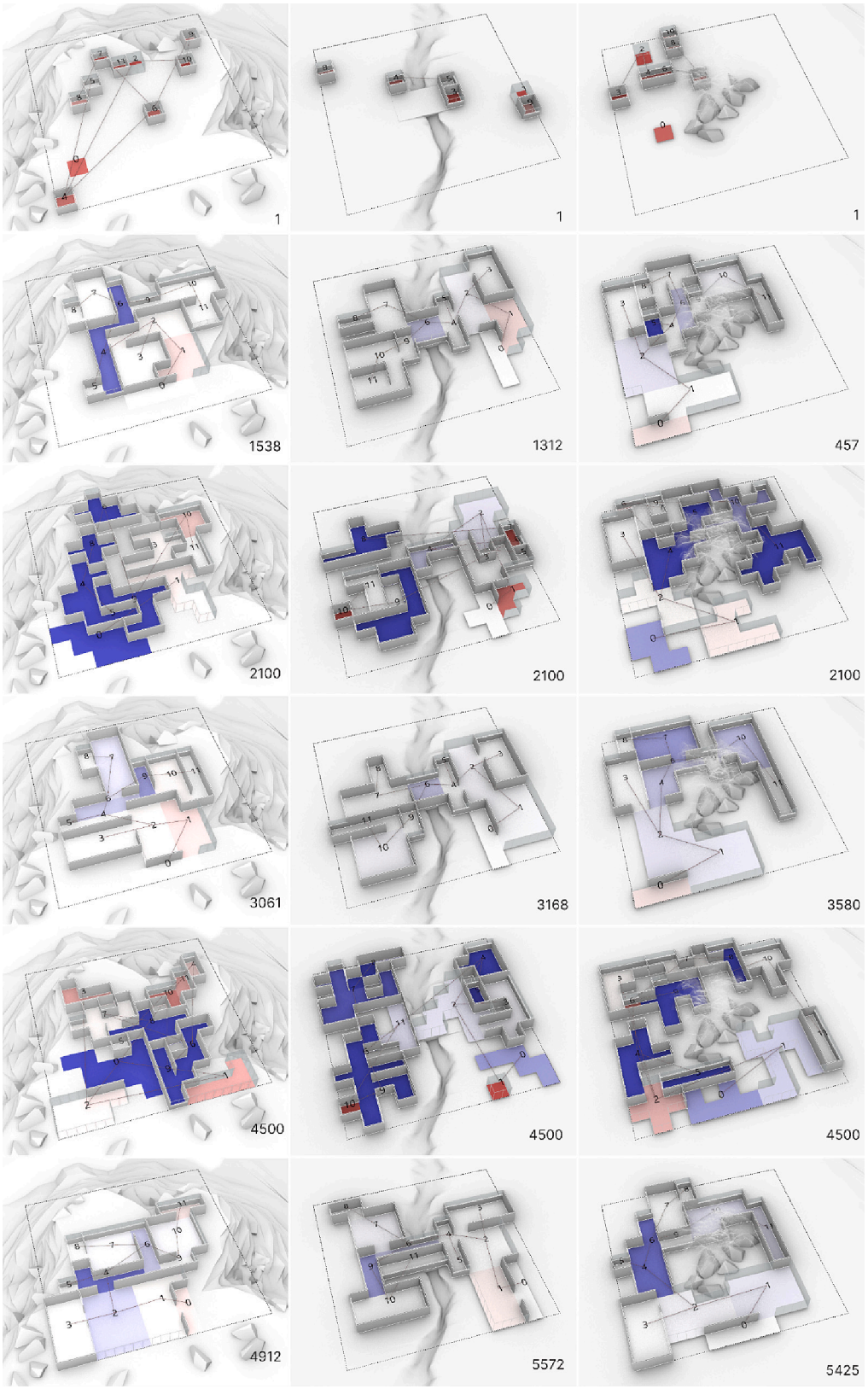


Fig. 12. Selected iterations from simulation of 12 agents using the policy Adjacency, Area, and Not Folding (5). Left Column: site on the top of the hill. Middle Column: site with bridge over a stream. Right Column: site with central obstacles.

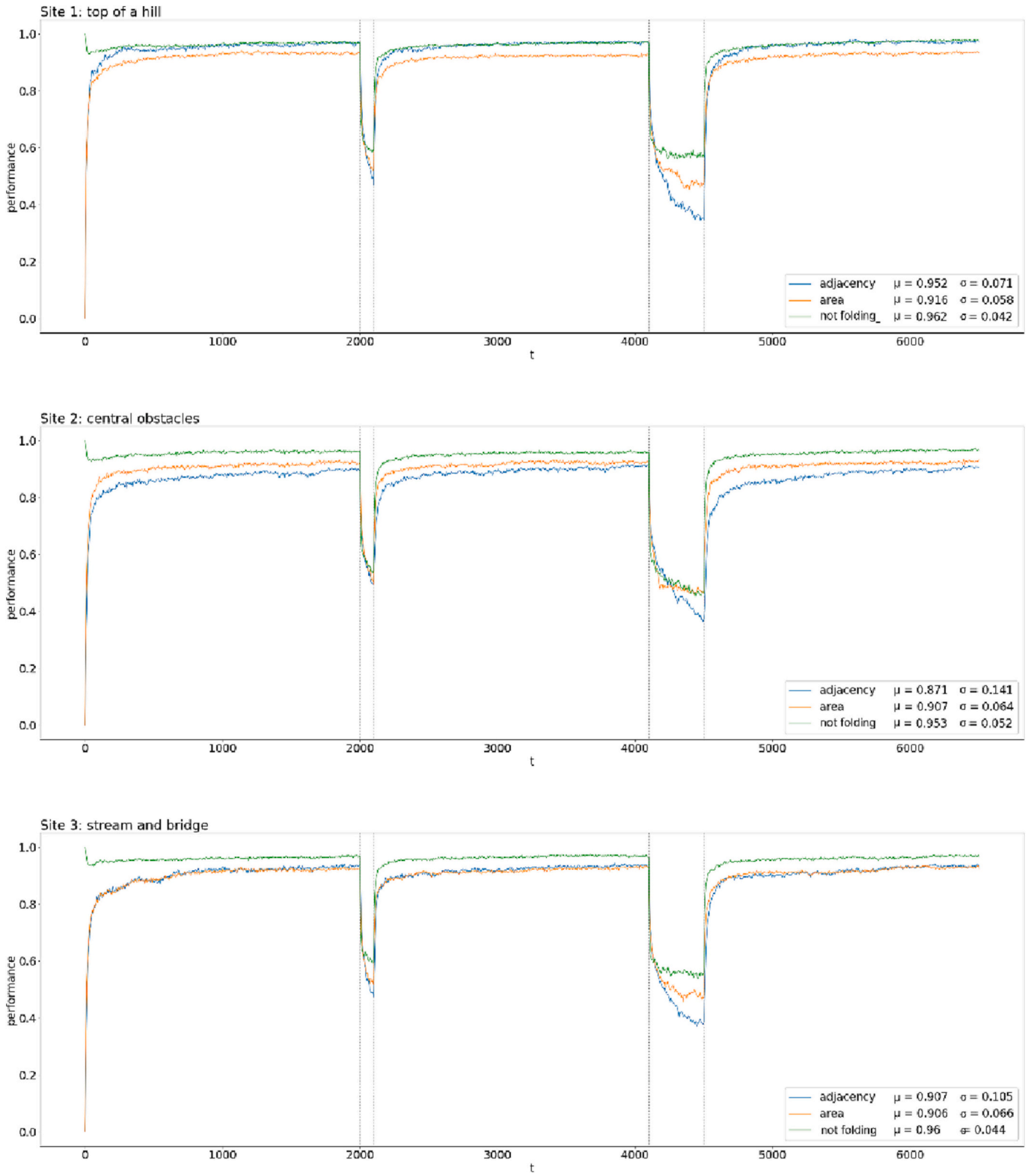


Fig. 13. Graphs with the average performance of the agents over 50 episodes for each of the three environments. The areas between dotted lines indicate the periods where action was selected randomly (perturbation). The means (μ) and standard deviations (σ) consider the three stages with ε-greedy action selection but not the periods with random perturbation.

larger number of agents in case 3.

Adjacency, Area, and Not Folding (3) generates rows of single cells restricted by the extension of 5 cells, which eliminates folds in the shape and reduces the distance between agents (see Fig. 19). This policy results in an agglomeration of linear rooms. In the case of the grouped agents without internal walls, there are also rectangular, L-shaped, and S-shaped rooms. With 96 agents, large multiple quasi-rectangular patios are created between the building units, which contribute to good daylight access. However, as the layout is based on an agglomeration of linear shapes, there are many cases where an internal room is surrounded by

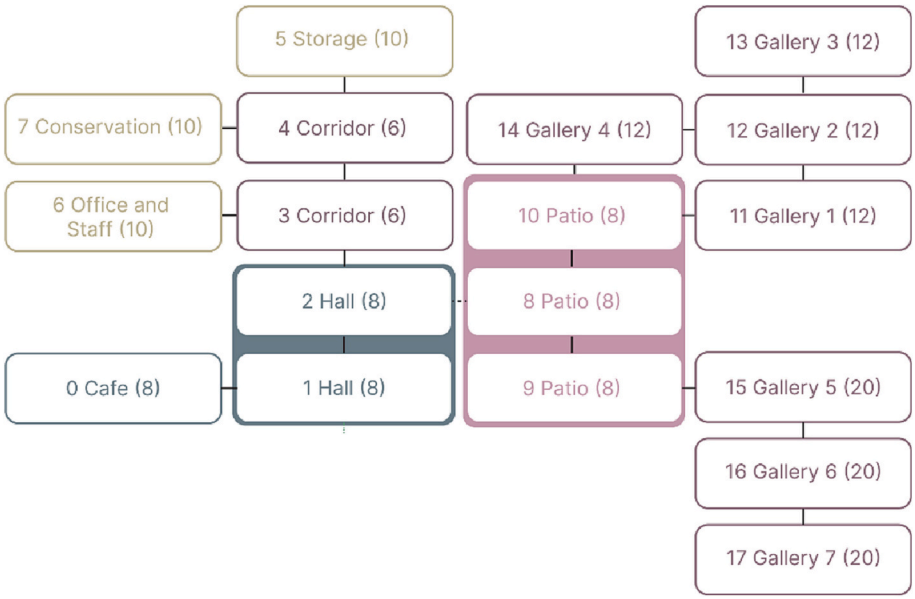


Fig. 14. Program for art gallery. The leading number is an identifier of the spaces. The numbers in parenthesis indicate the target areas. The connections indicate adjacency. The colored grouping indicates spatial integration between spaces.

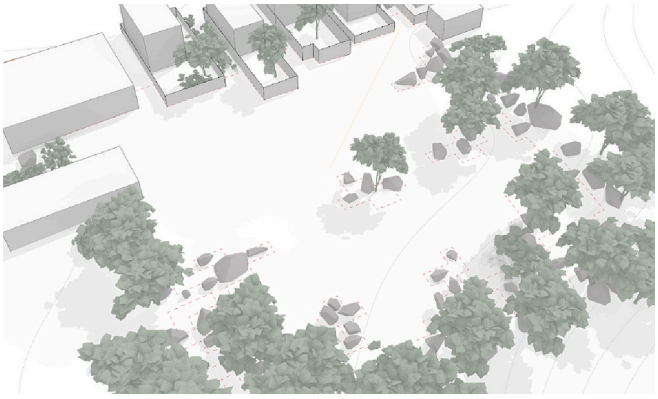


Fig. 15. Site for the art gallery. Dotted red line represents the boundary and obstacles for the agents. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

internal walls, which drastically reduces daylight access.

Adjacency, Area, and Daylight (75%) increases the extension of the external walls by creating irregular polyominoes with multiple concavities, which results in a maze-like configuration of corridors, small patios, and niches (see Fig. 20). Evidently, these concavities are not common in the design of houses but that can be useful for other programs, such as the galleries in case 2.

Adjacency, Area, and Not Folding (5) results in arrangements of rectangular and L-shaped rooms in a free form and centrifugal layout. With the large number of agents and limited space, buildings are connected, creating a series of irregular courtyards of different sizes (see Fig. 21).

Adjacency, Area, Not Folding (5), and Daylight (50%) results in building configurations like the ones from the previous policy but with more intricate outdoor configuration. To provide more access to daylight, each agent exposes more cells to the exterior, which results in a hierarchy of outdoor spaces, comprehending a network of courtyards of different sizes, and small corridors and patios between buildings (see Fig. 22).

Quantitatively, the agents quickly converge to a configuration with high-performance (see Fig. 23) and they spend most of the simulation doing small adjustments in the layout. In some cases, the agents were not able to solve a few high-level conflicts in terms of adjacencies that would require long-term collaboration of multiple agents.

6. Discussion and future research

Overall, the policies were able to address the three design cases, were efficient with respect to the different goals and were robust to the random perturbations in the simulations. They succeeded not only in terms of performance but also in producing a diversity of design alternatives that share essential spatial qualities. With more agents in the simulation and a larger site, specific cluster configurations that were not embedded directly into the policy emerged, such as a hierarchy of irregular courtyards, boulevards, and patios between housing units (see Fig. 24).

By training and deploying custom agents for interactive exploration of consistent spatial configurations, the research artifact proves the viability of the proposed framework for fine-grained interaction in spatial synthesis. It suggests a form of human-machine interaction that is an alternative for collaboration and shared authorship in design workflows based on AI, where designers can use ad hoc responses to the circumstances of the design generation inside the loop. Besides, it incentivizes the incorporation of principles from agent-based computing to architectural morphology and to the solution of practical architectural problems.

Some of these aspects will be discussed in another research under development, where a version of this prototype was integrated into a design tool to evaluate the experience of designers. Besides, future versions of the design artifact could benefit from code optimization, further parameter fine-tuning, longer training, or additional design knowledge. There are a lot of opportunities for future research under this framework, such as

- using different representations and expanding them to 3-D spaces
- extending the environment and observation grid to provide more contextual information
- increasing the action grid to allow shapes with higher resolution
- extending the number of adjacencies to enable more complex layouts

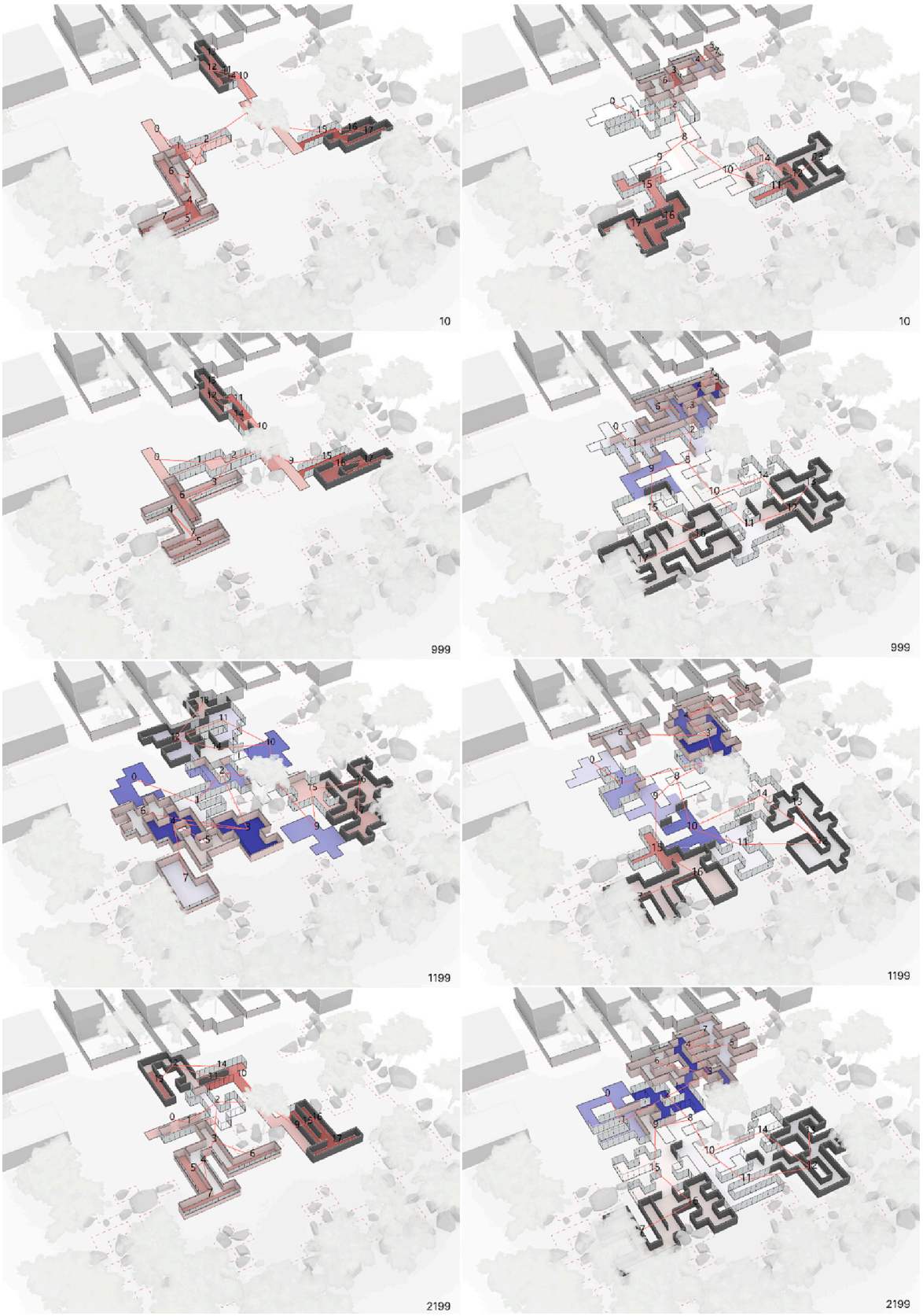


Fig. 16. Left: agents from setting 1 trained for adjacency, area, and not folding (3) goals. Right: Left: agents from setting 1 trained for adjacency, area, and daylight (75) goals.

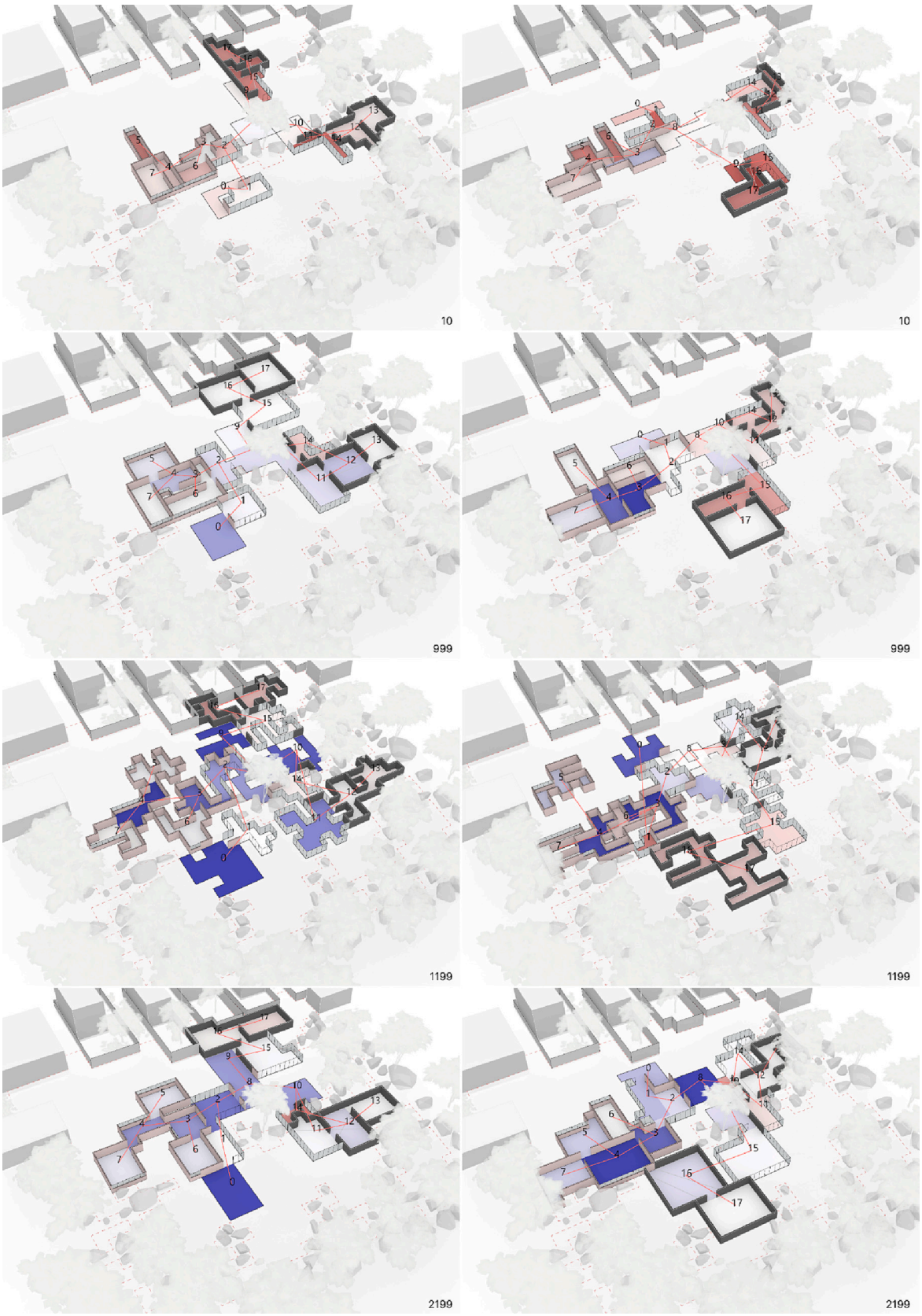


Fig. 17. Left: agents from setting 1 trained for adjacency, area, and not folding (5) goals. Right: Left: agents from setting 1 trained for adjacency, area, not folding (5), and daylight (50) goals.

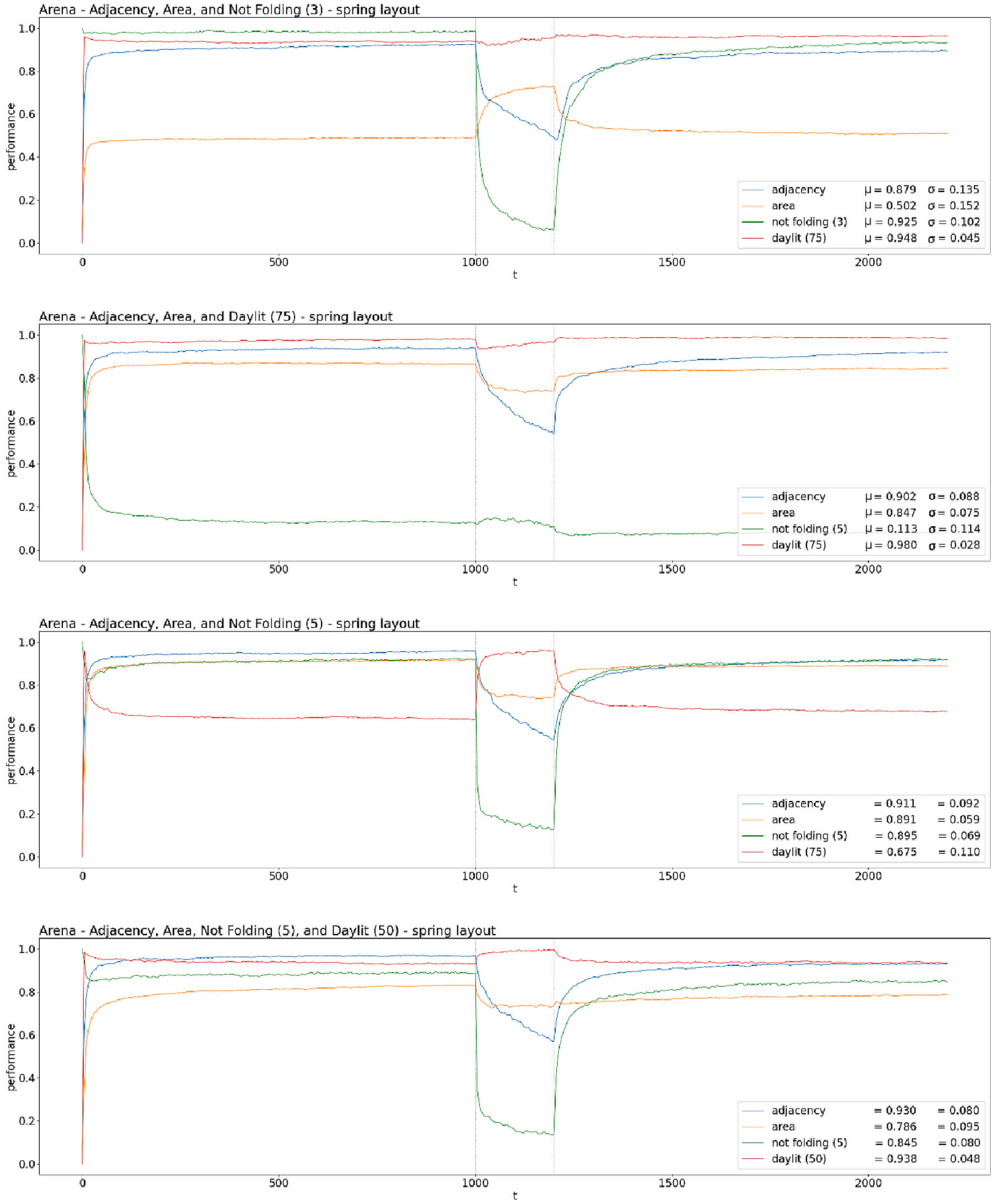


Fig. 18. Performance of the four policies in case 2. The areas between dotted lines indicate the periods where action was selected randomly (perturbation). The means (μ) and standard deviations (σ) consider the periods following the policy.

Fig. 19. 50 configurations produced by policy Adjacency, Area, and Not Folding (3) after 2000 steps. The configurations highlighted were selected for a more detailed description at the bottom.

- leaving all the goal parameters free, including not folding or daylight access, to incentivize the existence of more heterogenous agents in the same simulation
- incorporating objectives based on the direct interaction with the designer or on design examples to make the framework more adaptive to specific design practices

7. Limitations

This research explores an unorthodox approach to spatial synthesis based on a simulation with fine-granular interaction, which has a few limitations.

First, it is not straightforward to compare it with other algorithms off-shelf, because it breaks many assumptions of classical methods, such as the reliance on a pre-defined decision structure or hierarchy of

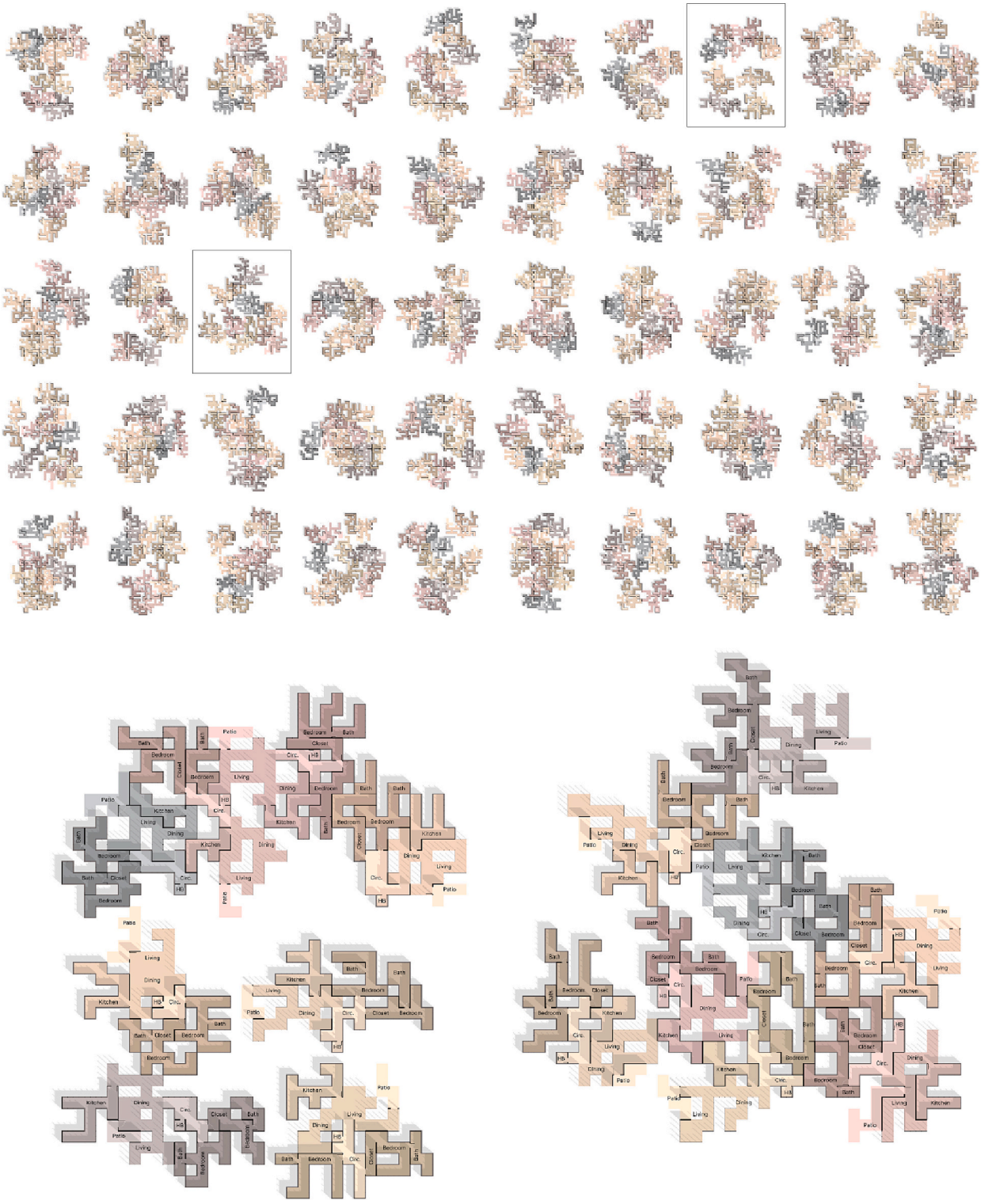


Fig. 20. 50 configurations produced by policy Adjacency, Area, and Daylight (75%) after 2000 steps. The configurations highlighted were selected for a more detailed description at the bottom.

decisions to make the exploration of the search tree more efficient or the idea that the designer is outside of the loop. For example, centralized approaches to heuristic search or to optimization would require some research on how to conciliate acting on the exponential action space imposed by the agents and supporting fine-grained interaction.

Another limitation is that MADRL presents a set of technical challenges for research. The target function for RL is nonstationary, RL algorithms are data hungry, and the sequential nature of the agent-environment interaction restricts parallelization. In our design construct, we trained the models in a laptop with Intel® Core™ i9-9880H CPU @2.30 GHz, 2304 MHz, 8 cores, 16 logical processors, 32 GB (RAM), and NVIDIA GeForce RTX 2080 with Max-Q Design. Working with computers with better graphical processing units in a cloud computing service barely improved the performance because the

Fig. 21. 50 configurations produced by policy Adjacency, Area, and Not Folding (5) after 2000 steps. The configurations highlighted were selected for a more detailed description at the bottom.

processes that rely on parallelization (backpropagation and forward passes) consumed minimal time compared to the production of observations by the training environments. To increase the efficiency of training and produce better models it will be necessary to work with an implementation that allows the use of multiprocessing for the training environments.

Finally, a theoretical limitation of our problem formulation is that it relies on an agent-centric reward function, which restricts its capacity to model top-down qualities of an architectural composition that depend on the configuration of many agents, such as axiality or symmetry. For that, it would be necessary to extend our approach with some environmental mechanism that is able to translate global properties into perceptual and reward feedback for each agent.

Fig. 22. 50 configurations produced by policy Adjacency, Area, Not Folding (5), and Daylight (50%) after 2000 steps. The configurations highlighted were selected for a more detailed description at the bottom.

8. Conclusion

Motivated by situated cognition and reflection-in-action, this research addressed the specific problem of creating a computational framework for spatial synthesis to support designers inside the generative loop with fine-grained interaction. To address this setting, our approach relies on a game to be solved by a multi-agent system in a simulation, which can be shared with human designers.

To prove the feasibility of this computational framework, we developed a research artifact that integrates agent-based modeling and multi-agent deep reinforcement learning to support the design an ecology of custom agents for interactive spatial synthesis. The method uses parameter sharing and a parameterized reward function to train multiple heterogeneous spatial agents that address multiple spatial synthesis

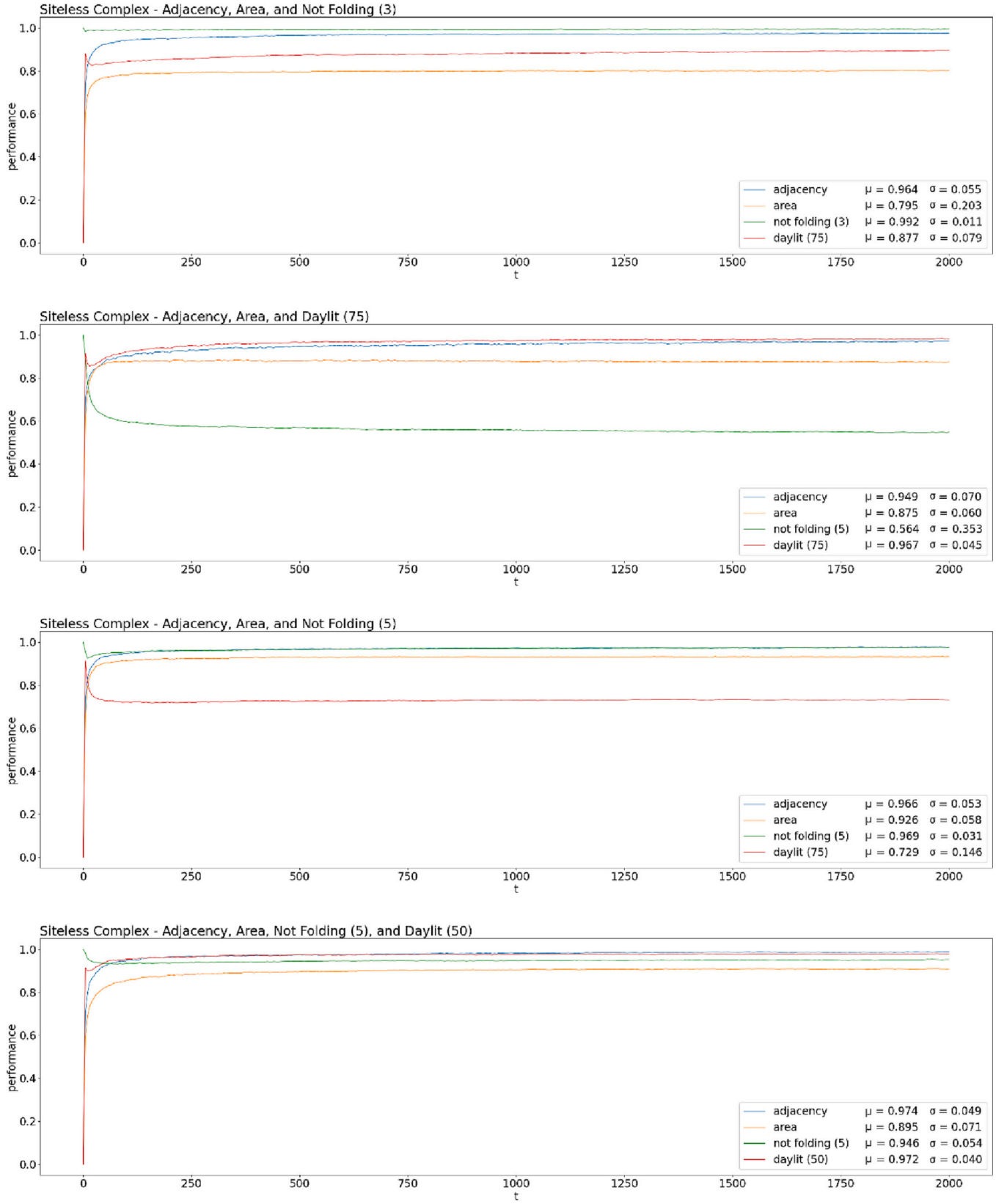


Fig. 23. Average performance of different policies over 50 episodes and with spring layout initialization.

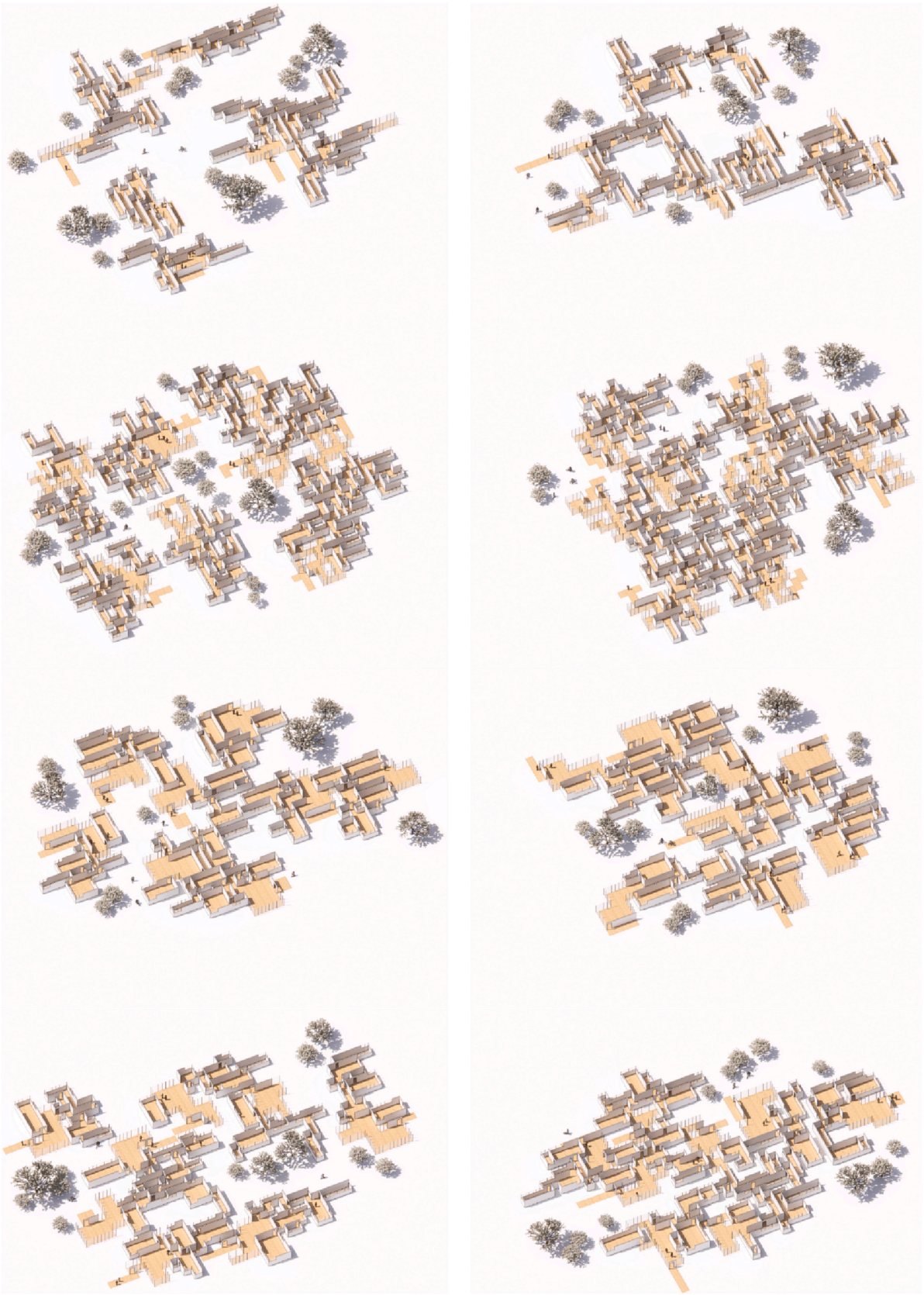


Fig. 24. 3D models of the selected samples from design case 3 showing the consistent morphology row by row.

goals and are largely independent of a predefined order of operation or initialization. The design artifact succeeded in addressing multiple design cases and was able to produce variations of design alternatives with shared spatial qualities. The interactive simulations with agents supported fast and fine-granular turn-taking and enabled the sequential generation of spatial configurations in large state spaces under uncertainty.

The success of the artifact opened doors to different lines of inquiry, from the gamification of human-machine interaction to the incorporation of structural innovations in the artifact to address more complex spatial problems.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

We would like to express our gratitude to the Brazilian National Council for Scientific and Technological Development (CNPq) for granting the first author a PhD scholarship (grant #201374/2014-5).

References

[1] Ömer Akin, Rana Sen, Navigation within a structured search space in layout problems, Environ. Plann. B 23 (4) (1996) 421–442, https://doi.org/10.1068/b230421.
[2] Silvio Carta, Stephanie St Loe, Tommaso Turchi, Joel Simon, Self-organising floor plans in care homes, Sustainability 12 (11) (2020) 4393, https://doi.org/10.3390/su12114393.
[3] Tom De Wolf, Tom Holvoet, Emergence versus self-organisation: different concepts but promising when combined, in: Sven A. Brueckner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, Radhika Nagpal (Eds.), Engineering Self-Organising Systems: Methodologies and Applications, Lecture Notes in Computer Science, Springer, Berlin, 2005, https://doi.org/10.1007/11494676_1.
[4] Charles M. Eastman, Representations for space planning, in: C.L. Lawson (Ed.), Communications of the ACM 13 (4), 1970, https://doi.org/10.1145/362258.362281.
[5] Laura Graesser, Wah Loon Keng, Foundations of Deep Reinforcement Learning: Theory and Practice in Python, in: Addison-Wesley Data & Analytics Series, Addison-Wesley Professional, 2019 (ISBN 978-0-13-517248-3), https://www.pearson.com/store/p/foundations-of-deep-reinforcement-learning-theory-and-practice-in-python/P200000009486/9780135172483 (ISBN 978-0-13-517248-3).
[6] Zifeng Guo, Biao Li, Evolutionary approach for spatial architecture layout design enhanced by an agent-based topology finding system, Front. Architect. Res. 6 (1) (2017) 53–62, https://doi.org/10.1016/j.foar.2016.11.003.
[7] Jayesh K. Gupta, Maxim Egorov, Mykel Kochenderfer, Cooperative multi-agent control using deep reinforcement learning, in: Gita Sukthankar, Juan A. Rodriguez-Aguilar (Eds.), Autonomous Agents and Multiagent Systems, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2017, pp. 66–83, https://doi.org/10.1007/978-3-319-71682-4_5 (ISBN 978-3-319-71682-4).
[8] Max Henrion, Automatic space-planning: A postmortem? in: Jean Claude Latombe (Ed.), Artificial Intelligence and Pattern Recognition in Computer Aided Design North-Holland, Amsterdam, 1978, pp. 175–191 (ISBN 978-0-444-85229-8).
[9] John H. Holland, Emergence: From Chaos to Order, Basic Books, New York, 1998 (ISBN 978-0-7382-0142-9).
[10] Ying-Chun Hsu, Robert J. Krawczyk, New generation of computer aided design in space planning methods - a survey and a proposal, in: K. Keatruangkamala, W. Nakapan (Eds.), Proceedings of the 8th International Conference on Computer Aided Architectural Design Research in Asia (CAADRIA), Rangsit University, Bangkok, 2003, pp. 101–116, https://doi.org/10.52842/conf.caadria.2003.101.
[11] Reinhard Koenig, Matthias Standfest, Gerhard Schmitt, Evolutionary multi-criteria optimization for building layout planning - Exemplary application based on the PSSA framework, in: Fusion - Proceedings of the 32nd eCAADe Conference 2, Department of Architecture and Built Environment, Newcastle upon Tyne, England, UK, 2014, pp. 567–574, https://doi.org/10.52842/conf.ecaade.2014.2.567.
[12] Robin S. Liggett, The quadratic assignment problem: an analysis of applications and solution strategies, Environ. Plann. B 7 (2) (1980) 141–162, https://doi.org/10.1068/b070141.
[13] Robin S. Liggett, Automated facilities layout: past, present and future, Autom. Constr. 9 (2) (2000) 197–215, https://doi.org/10.1016/S0926-5805(99)00005-9.
[14] Danny Lobos, Dirk Donath, The problem of space layout in architecture: a survey and reflections, Arquiteturarevista 6 (2) (2010) 136–161, https://doi.org/10.4013/arq.2010.62.05.
[15] William Mitchell, Computer-Aided Architectural Design, Mason Charter, New York, 1977 (ISBN 978-0-88405-323-1).
[16] Muaz Niazi, Amir Hussain, Agent-based computing from multi-agent systems to agent-based models: a visual survey, Scientometrics 89 (2011) 479–499, https://doi.org/10.1007/s11192-011-0468-9.
[17] Morteza Rahbar, Mohammadjavad Mahdavinejad, Amir H.D. Markazi, Mohammadreza Bemanian, Architectural layout design through deep learning and agent-based modeling: a hybrid approach, J. Build. Eng. 47 (2022), 103822, https://doi.org/10.1016/j.jobe.2021.103822.
[18] Stuart J. Russell, Peter Norvig, Artificial intelligence: A modern approach, in: Pearson Series in Artificial Intelligence, 4th ed., Pearson, Hoboken, 2021.
[19] Donald Schön, The Reflective Practitioner: How Professionals Think in Action, Basic Books, New York, 1984 (ISBN 978-0-465-06878-4).
[20] Donald Schön, Designing as reflective conversation with the materials of a design situation, Knowl.-Based Syst. 5 (1) (1992) 3–14, https://doi.org/10.1016/0950-7051(92)90020-G.
[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, Proximal Policy Optimization Algorithms, arXiv, 2017, https://doi.org/10.48550/arXiv.1707.06347.
[22] Yoav Shoham, Kevin Leyton-Brown, Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations, 1st ed., Cambridge University Press, Cambridge, 2008 (ISBN 978-0-521-89943-7).
[23] Christian Sjoberg, Christopher Beorkrem, Jefferson Ellinger, Emergent syntax: machine learning for the curation of design solution space, in: Takehiko Nagakura, Skylar Tibbits, Mariana Ibañez, Caitlin Mueller (Eds.), ACADIA 2017: DISCIPLINES & DISRUPTION - Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture, ACADIA, Cambridge, MA, 2017, pp. 552–561, https://doi.org/10.52842/conf.acadia.2017.552.
[24] David Stieler, Tobias Schwinn, Samuel Leder, Mathias Maierhofer, Fabian Kannenberg, Achim Menges, Agent-based modeling and simulation in architecture, Autom. Constr. 141 (2022), 104426, https://doi.org/10.1016/j.autcon.2022.104426.
[25] Richard S. Sutton, Andrew G. Barto, Reinforcement Learning: An Introduction, 2nd ed., The MIT Press, Cambridge, 2018 (ISBN 978-0-262-03924-6), http://incompleteideas.net/book/ (ISBN 978-0-262-03924-6).
[26] J.K. Terry, Nathaniel Grammel, Ananth Hari, Luis Santos, Benjamin Black, Revisiting Parameter Sharing in Multi-Agent Deep Reinforcement Learning, arXiv, February, 2021, https://doi.org/10.48550/arXiv.2005.13625.
[27] Tran Luciani, Jonas Löwgren Danwei, Jonas Lundberg, Designing fine-grained interactions for automation in air traffic control, Cogn. Tech. Work 22 (4) (2020) 685–701, https://doi.org/10.1007/s10111-019-00598-9.
[28] Pedro Veloso, Jinmo Rhee, Ardavan Bidgoli, Manuel Ladron de Guevara, Bubble2Floor: a pedagogical experience with deep learning for floor plan generation, in: Jeroen van Ameijde, Nicole Gardner, Kyung Hoon Hyun, Dan Luo, Urvi Sheth (Eds.), POST-CARBON - Proceedings of the 27th CAADRIA Conference, CAADRIA, Sydney, 2022, pp. 373–382, https://doi.org/10.52842/conf.caadria.2022.1.373.
[29] Pedro Veloso, Jinmo Rhee, Ramesh Krishnamurti, Multi-agent space planning: A literature review (2008-2017), in: Ji-Hyun Lee (Ed.), "Hello, Culture!" - Proceedings of the 18th International Conference, CAAD Futures 2019, CAAD Futures, Daejeon, Korea, 2019, pp. 52–74 (ISBN 978-89-89453-05-5), http://papers.cumincad.org/cgi-bin/works/paper/cf2019_009 (ISBN 978-89-89453-05-5).
[30] Ramon Elias Weber, Caitlin Mueller, Christoph Reinhart, Automated floorplan generation in architectural design: a review of methods and applications, Autom. Constr. 140 (August) (2022), 104385, https://doi.org/10.1016/j.autcon.2022.104385.
[31] Jiayi Weng, Minghao Zhang, Alexis Duburcq, Kaichao You, Tianshou v0.3.0, in: Repository, GitHub, 2020. https://github.com/thu-ml/tianshou/releases/tag/v0.3.0.
[32] Uri Wilensky, William Rand, An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo, The MIT Press, Cambridge, 2015 (ISBN 978-0-262-73189-8).
[33] Michael Wooldridge, An Introduction to MultiAgent Systems, 2nd ed., John Wiley & Sons, 2009 (ISBN 978-0-470-51946-2).
[34] Thomas Wortmann, Surveying design spaces with performance maps: a multivariate visualization method for parametric design and architectural design optimization, Int. J. Archit. Comput. 15 (1) (2017) 38–53, https://doi.org/10.1177/1478077117691600.
[35] Thomas Wortmann, Giacomo Nannicini, Introduction to architectural design optimization, in: Athanasia Karakitsiou, Athanasios Migdalas, Stamatina T. Rassia, Panos M. Pardalos (Eds.), City Networks: Collaboration and Planning for Health and Sustainability, Springer International Publishing, Cham, 2017, pp. 259–278, https://doi.org/10.1007/978-3-319-65338-9_14.
[36] Veloso Pedro, Ramesh Krishnamurti, An Academy of Spatial Agents, in: Liss C. Werner, Dietmar Köring (Eds.), *Anthropologic - Architecture and Fabrication in the Cognitive Age. Proceedings of the 38th eCAADe Conference* 2, CUMINCAD, Berlin, 2020, pp. 191–200. https://doi.org/10.52842/conf.ecaade.2020.2.191.