Fall 2021

Sampling and Robustness in Multi-Robot Visibility-Based Pursuit-Evasion

Trevor Vincent Olsen

Sampling and Robustness in Multi-Robot Visibility-Based Pursuit-Evasion

by

Trevor Vincent Olsen

Bachelor of Science
Palm Beach Atlantic University, 2013

Master of Arts
University of Miami, 2015

Doctor of Philosophy
University of South Carolina, 2020

Master of Science
University of South Carolina, 2021

Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy in

Computer Science

College of Engineering and Computing

University of South Carolina

2021

Accepted by:

Jason O'Kane, Major Professor

Stephen Fenner, Committee Member

Ioannis Rekleitis, Committee Member

Marco Valtorta, Committee Member

Éva Czabarka, Committee Member

Nicholas Stiffler, Committee Member

Tracey L. Weldon, Vice Provost and Dean of the Graduate School

Dedication

To McKauley Kane and Orlie Martin for helping me turn childhood hobbies into a lifelong career. Love and miss you both.

Acknowledgments

I'd like to thank, to the greatest extent, Jason O'Kane for being an outstanding advisor. You always applied the perfect amount of pressure to me to maximize my academic progress. From the patience you exuded as I learned the ropes of a completely new field, to the motivation during the month of a paper deadline, I was never overwhelmed or stagnant. Of course, I owe so much of my career as a researcher to my math advisors Éva Czabarka and László Székely. Both of you prepared me to apply myself no matter the task. I'm also remarkably grateful to have worked with Nicolas Stiffler, who provided constant encouragement and knowledge that was pivotal to my robotics research. Thank you to the additional members of my committee: Stephen Fenner, Ioannis Rekleitis and Macro Valtorta for your guidance.

I would have never been able to succeed to this extent without the unconditional love and support from my mother Kelly, my father Rick and my grandmother Juanita.

Thank you to all of the professors and collaborators that helped me reach this point. In particular, thank you Drs. Boulware, Coomes, Dankelmann, de Oliveira, Diemer, Draghici, Fowler, Girardi, Kaliman, Mackritis, Pembamoto and Swick.

Lastly, I would like to show appreciation for my friends and family, especially: Amanda, Anne, Austin, Cassie, Christopher, Corey, Dan, Danielle, Dylan, Edelio, Elodie, Gabby, Gary, Grace, Griffin, Jack, Jacob, Jim, Jon, Katechka, Kati, Kim, Lisa, Logan, María, Marios, Max, Mitchell, Molly, Nick, Pedro, Ryan, Seyver, Tim and Tyler. I'd also like to thank all of my colleagues from Inver Hills Community College, Palm Beach Atlantic University, the University of Miami and the University of South Carolina.

Abstract

Given a two-dimensional polygonal space, the multi-robot visibility-based pursuit-evasion problem tasks several pursuer robots with the goal of establishing visibility with an arbitrarily fast evader. The best-known complete algorithm for this problem takes time doubly exponential in the number of robots. However, sampling-based techniques have shown promise in generating feasible solutions in these scenarios.

Existing sampling-based algorithms have long execution times and high failure rates for complex environments. We first address that limitation by proposing a new algorithm that takes an environment as its input and returns a joint motion strategy which ensures that the evader is captured by one of the pursuers. Starting with a single pursuer, we sequentially construct data structures called Sample-Generated Pursuit-Evasion Graphs to create such a joint motion strategy. This sequential graph structure ensures that our algorithm will always terminate with a solution, regardless of the complexity of the environment.

Another aspect of this problem that has yet to be explored concerns how to ensure that the robots can recover from catastrophic failures which leave one or more robots unexpectedly incapable of continuing to contribute to the pursuit of the evader. To address this issue, we propose an algorithm that can rapidly recover from catastrophic failures. When such failures occur, a replanning occurs, leveraging both the information retained from the previous iteration and the partial progress of the search completed before the failure to generate a new motion strategy for the reduced team of pursuers.

The final contribution is a novel formulation of the pursuit-evasion problem that modifies the pursuers' objective by requiring that the evader still be detected, even in spite of the malfunction of any single pursuer robot. This novel constraint, whereby two pursuers are required to detect an evader, has the benefit of providing redundancy to the search, should any member of the team become unresponsive, suffer temporary sensor disruption/failure, or otherwise become incapacitated. The proposed formulation produces plans that are inherently tolerant of some level of disturbance.

For each contribution discussed above, we describe an implementation of the algorithm and provide quantitative results that show substantial improvement over existing results.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Related Work

1.1 Introduction

Autonomous reconnaissance tasks, in which robots strive to observe salient features of objects or agents within their environments, remain one of the most active threads of research within the robotics community. Such tasks have wide-ranging application domains such as environmental monitoring [10], [11], [22], [55], [56], surveillance [2], [3], [9], [13], and search-and-rescue [20], [31], [48]. Many of these tasks can be framed as two-player games played amongst opposing teams: evaders (who wish to evade capture) and pursuers (who seek to capture them). This dissertation is concerned with a specific form of this two-player game, wherein a team of pursuers must locate an evader (or group of evaders) in a polygonal environment.

Specifically, we address one such problem where a group of pursuers, each equipped with an omni-directional sensor that extends to the polygonal boundary, must form a motion plan to locate an arbitrarily fast evader in a polygonal environment. Figure 1.1 illustrates this scenario.

The literature has a number of results for this problem in the single-pursuer case, including algorithms with strong guarantees such as completeness [17] and optimality [52]. However, the case in which multiple pursuers cooperate is not nearly as well understood. A complete algorithm is known, but it runs in time doubly-exponential in the number of pursuers [49].

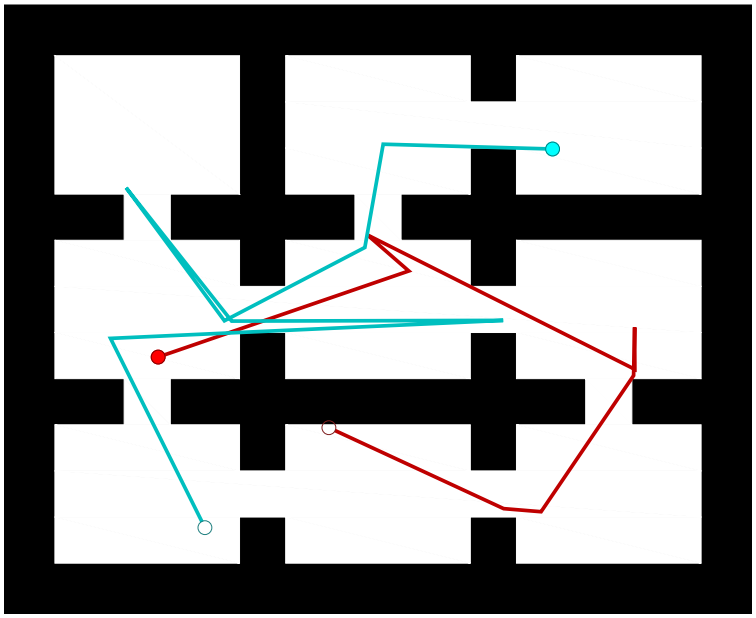


Figure 1.1: An example multi-pursuer strategy that ensures the capture of any number of evaders.

Generating solutions using sampling techniques has proven useful in the past, although existing results [50] left substantial room for improvement. In subsequent chapters of this dissertation, we discuss improved sampling methods and solution refinement techniques. Additionally, we present algorithms that can solve variations of the multi-robot visibility-based pursuit-evasion such as determining the number of pursuers recommended for an environment during execution. Lastly, we present methods to quickly recover from catastrophic robot failure as well as generating solutions which remain solutions regardless of any single robotic failure.

The contributions of this dissertation are:

- Two novel sampling methods that ensure that all regions of the environment are examined (Chapter 3).
- Methods for modifying the main data structure to include an additional pursuer. This allows us to eliminate the number of pursuers as an input (Section 4.3.1).

- A greedy solution refinement technique which greatly reduces the length of solution paths (Section 4.3.2).
- Failure recovery mechanisms which allow the algorithm to adapt and replan in real time should a catastrophic robotic failure occur (Section 5.3).
- Robust solution generation, that is, generating a solution that remains a solution given any single robotic malfunction (Section 6.3).

1.2 Related Work

The research presented in this dissertation blends ideas from two vibrant threads of prior robotics research: visibility-based pursuit-evasion and sampling-based motion planning in high dimensional spaces.

The literature on pursuit-evasion problems can be understood by organizing according to underlying models, including differential game theory, graph variants, and geometric variants. Though this work considers the geometric formulation, we present a brief synopsis of the contributions in the domains of differential game theory and graph theory.

The seminal work of Isaacs [21] and Ho et. al. [19] was the first to adapt the pursuit-evasion problem to a dynamic game-theoretic framework. This remains an active research area [8], [28], [41], [58]. Recent results include continued progress by utilizing techniques such as reinforcement learning [60] and the exploitation of rich representations such as Voronoi partitions to aid in the search [32], [62].

A different formulation in which the domain is modeled as a discrete graph was initially proposed by Parsons [37] and is referred to as the edge-searching problem. Petrov later independently rediscovered some of Parsons' results in the context of differential game theory [38]. Golovach later showed that both problems considered an equivalent discrete game on graphs [15]. A number of survey papers [1], [4],

[7] provide overviews of the many problem variants that can be realized within the graph model, such as specifying the rules of movement for the pursuers and for the evaders [43], the kind of graph [26], etc.

The visibility-based pursuit-evasion problem posed in this dissertation can be thought of as a specialization of the broader problem of *search and target tracking*. The common theme across this work is the pursuit of an agent (or agents) by one (or more) pursuers to either establish or maintain visibility of the target [44], [45], [63].

This work is most closely aligned with the problem first introduced by Suzuki and Yamashita [54], in which an evader operating in a geometric environment seeks to locate an unpredictable evader capable of moving arbitrarily quickly. This work was later expanded by Guibas, Latombe, LaValle, Lin, and Motwani [17] who provide a complete algorithm for the single pursuer scenario in simply-connected environments for a pursuer with an omnidirectional field-of-view. Park, Lee, and Chwa [36] identified necessary and sufficient conditions for a search to be feasible for a single pursuer. Other results for the single pursuer scenario that build upon this foundation provide results such as completeness [17], optimality [52], or consider more restrictive scenarios with respect to pursuer parameters such as sensing and actuation [6], [14], [39], [40], [42], [51], [53], [57].

More recently, the community has placed increased emphasis on the study of richer scenarios where a team of pursuers cooperate during the search [12], [16], [25], [50]. Stiffler and O'Kane [49] present an algorithm utilizing a cylindrical algebraic decomposition that, while complete, relies upon constructing a graph whose size is doubly exponential in the complexity of the environment. That work subsequently served as motivation for approaches that utilize heuristics [50] that seek to overcome the problem complexity by utilizing sampling techniques.

More generally, sampling based techniques have been employed in a number of planning contexts where computing an exact solution proves computationally in-

tractable such as motion planning [23], [24], [29], [30], [47] and manipulation planning [18], [27], [46]. One caveat of sampling-based methods is that they quite often suffer from the *curse of dimensionality* [5] whereby, as the number of dimensions increases, the search space becomes so vast that the number of samples required for adequate coverage of the space increases dramatically. A number of different approaches have been proposed to combat this problem. One recent result draws samples in lower-dimensional subspaces to search for a feasible solution, and incrementally reasons about higher dimensions while utilizing the information gained in the lower dimensional graph [61]. For the specific multi-robot case in which the configuration space is a Cartesian product of the configuration spaces of individual agents in the system, one novel approach seeks to reason about each agent independently (a subdimension), and only when the agents reach a point where they interact with one another is there a lifting to a higher-dimensional space [59].

The work presented in Chapter 4 is joint work with Anne M. Tumlin, Nicholas M. Stiffer and Jason M. O'Kane. It appeared in the proceedings of ICRA 2021 [35]. Chapter 5 was a collaborative effort with Nicolas M. Stiffer and Jason M. O'Kane and was presented during IROS, 2021 [33]. Lastly, the contents of Chapter 6, which is cooperative work with Nicholas Stiffler and Jason O'Kane, has been submitted to ICRA, 2022 [34].

Chapter 2

General Problem Statement and Background

While each of the remaining chapters in this dissertation address unique variations of the multi-robot visibility-based pursuit-evasion problem, they each share a wealth of common definitions and notation. To avoid redundancy, the common terms will be discussed in this chapter.

2.1 General Problem Statement

The *environment* F is a closed, bounded, and connected polygonal region in $\mathbb{R}^2$. A team of n *pursuers*, who can travel throughout the environment at bounded speed, are equipped with omnidirectional sensors whose range is only bounded by line of sight within the environment. That is, a pursuer at point $q \in F$ can detect anything within its *visibility polygon* $V(q) = \{r \in F \mid \overline{qr} \subset F\}$. We denote the location of the i^{th} pursuer as a function of the time t by the continuous function $f_i(t) : [0, T] \to F$, in which T is some termination time which the pursuers may choose. The n-robot *joint pursuer configuration* (JPC) at time t is the vector $\langle f_1(t), f_2(t), \ldots, f_n(t)\rangle \in F^n$.

A single *evader* seeks to avoid detection by the pursuers by moving continuously within the environment, without any bound on its speed. We denote its location, as a function of time, by the continuous function $e(t) : [0, \infty) \to F$, unknown to the pursuers. Observe that, because we plan for the worst case, any strategy for the pursuers that guarantees detection of a single evader can also guarantee detection for each of potentially many evaders.

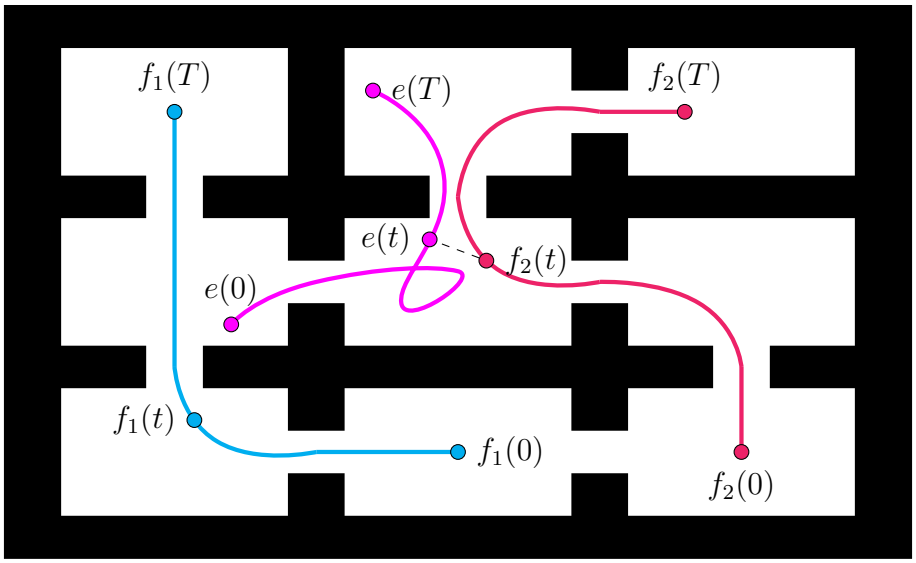


Figure 2.1: Two pursuers execute a search. Note that at time t, the pursuer on path f_2 is capable of detecting the evader e.

The pursuers' objective is to establish visibility with the evader. Thus, for a given environment F, the goal is to choose the termination time T and the functions $f_1, f_2, \ldots, f_n$ to ensure that for any evader trajectory e, there exists a time $t_0 \in [0, T]$, such that $e(t_0) \in \bigcup_{i \leq n} V(f_i(t_0))$. Figure 2.1 illustrates the notation.

2.2 Background

The primary difficulty in this type of visibility-based pursuit-evasion concerns reasoning about the regions of the environment that are not currently visible to the pursuers at the present time. To resolve that difficulty, Guibas, Latombe, LaValle, Lin, and Motwani [17] introduced a reformulation of the problem, based upon tracking which, if any, of the regions of the environment not currently perceptible by the pursuers might contain an as-yet-undetected evader.

2.2.1 Shadows and Shadow Labels

To formalize this idea, define the *shadow region*, $S(t) = F \setminus \bigcup_{i \leq n} V(f_i(t))$ as the portion of the environment unseen by any pursuer at time t. The maximal connected

components of $S(t)$ are called *shadows*. The terms *cleared* and *contaminated* can be applied to a shadow to reflect the relative *status* of the shadow at that point in the pursuers' search. A cleared shadow is an unseen area of the environment that, based upon the pursuers' motions up to the current time t, is guaranteed to not contain an unseen evader. Any shadow that is not cleared is called contaminated. Figure 2.2 displays both cleared and contaminated shadows.

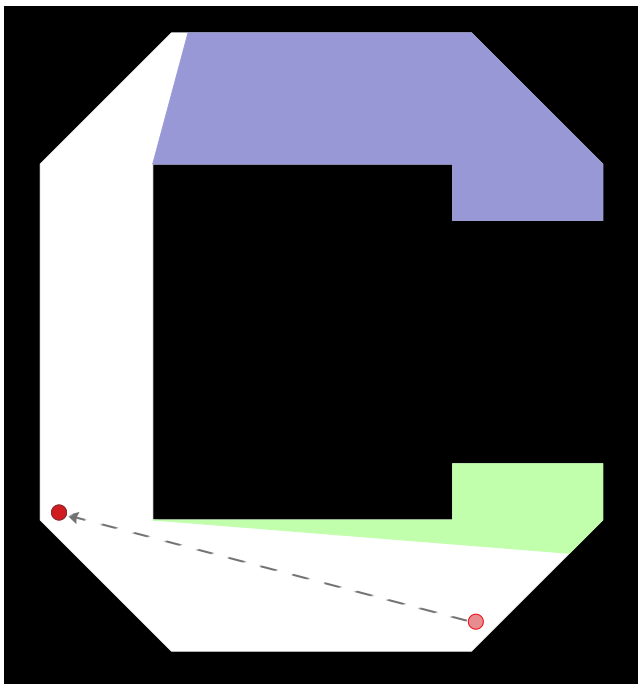


Figure 2.2: As the pursuer moves left, it loses vision of the bottom right corner. Based on the pursuer's path, it is impossible for an evader to be hidden in the newly formed shadow, thus, it is cleared.

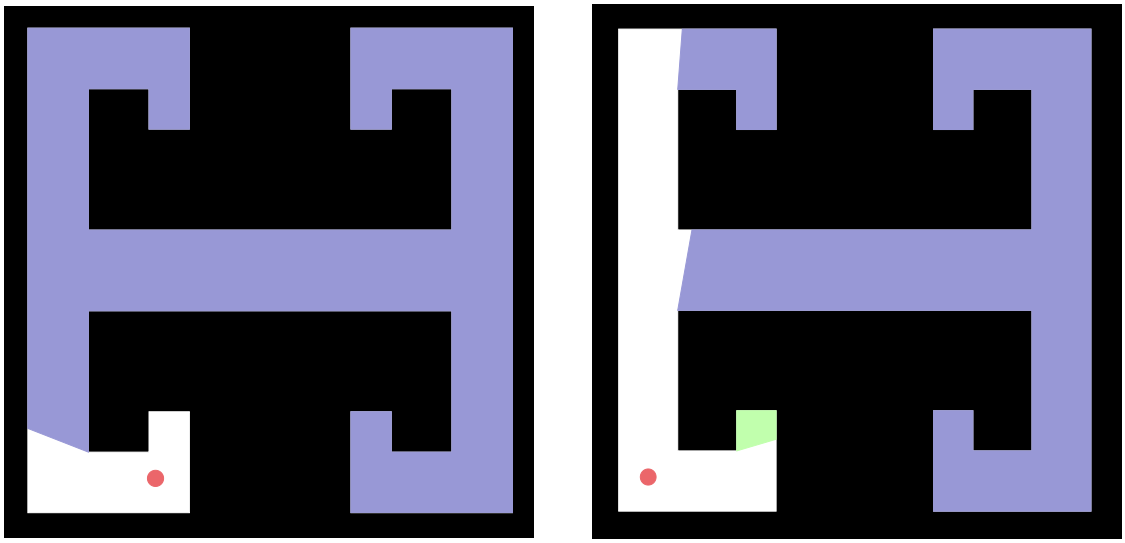
To compactly describe the status for all of the shadows, we utilize a *shadow label*, which is a binary string comprised of one bit for each shadow, in which the i^{th} bit is 1 if the i^{th} shadow (in an arbitrary but fixed ordering) is contaminated, and 0 otherwise.

2.2.2 Shadow Events

Though shadows change continuously as the pursuers move within F, the cardinality of the shadows and their labels can change only when a shadow event occurs, i.e. a shadow appears, disappears, splits, or multiple shadows merge into a single shadow. Examples of appear and split shadow events are shown in Figure 2.3.

- *Appear*: A shadow appears when the pursuers move in a way that makes a previously observed part of the environment no longer visible. In this case, the new shadow is assigned a cleared status.
- *Disappear*: If a pursuer gains vision of a shadow, we say the shadow disappears. Here, the shadow bit is deleted.
- *Split*: If a pursuers' vision disconnects a shadow, we say that the shadow has split. Each new component is given the label of the original pre-split shadow.
- *Merge*: If two or more shadows become a single connected component, we say the shadows have merged. The newly merged shadow takes on the cleared status if and only if each merging component was cleared prior to the merge. Otherwise, the merged shadow is considered to be contaminated.

This formulation of the problem in terms of clear and contaminated shadows is valuable because it enables a planner to reason over those shadows and labels, rather than directly over the space of all possible evader paths. That is, the overall visibility-based pursuit-evasion problem can be restated as a search for pursuer motions that lead to a system state in which the binary string of shadow labels contains all zeroes, indicating that every shadow is clear.



(a) The initial location of a single pursuer. Here, the shadow label is 1.

(b) Movement of a pursuer that would cause one shadow to appear and another to split. The result has one cleared and two contaminated shadows. In our implementation, this scenario is assigned shadow label 110.

Figure 2.3: An example of shadow events and labels.

2.2.3 SG-PEG

The SG-PEG is a data structure the represents a roadmap of valid joint paths for a team of pursuers in a known environment F, augmented with information about the shadow labels that can be achieved by executing those paths. We present here a concise overview; additional detail may be found in the original paper [50].

An SG-PEG is a directed graph $G = (V_G, E_G)$, in which one vertex v_0 is designated as the *root vertex*. Each SG-PEG is constructed for a specific number n of pursuers. Each vertex $v \in V_G$ corresponds to a specific JPC $\langle p_1, \ldots, p_n \rangle \in F^n$. Each directed edge $e \in E_G$ connects two vertices $v, u \in V_G$ for which it is possible for every pursuer to make a collision-free straight line motion between the representative configurations. That is, the existence of an edge from v to u means that, for each $1 \leq i \leq n$, $\overline{v_i u_i} \subset F$.

In addition to this graph structure, each vertex v maintains a set of reachable shadow labels. Specifically, a shadow label ℓ will be recorded at a particular vertex

v as a reachable shadow label if there exists a walk from v_0 to v that results in the shadow marked clear within ℓ indeed being clear.

The primary operation that can be performed on a SG-PEG is AddSample($\langle p_1, \ldots, p_n \rangle$), which accepts a collision-free JPC as input and performs the following steps:

(i) It inserts a new vertex v at the given JPC.

(ii) For every existing vertex u for which the segment $\overline{uv}$ is collision free in F^n, it adds the edges $\overrightarrow{uv}$ and $\overrightarrow{vu}$. The operation then computes a mapping that describes how the shadows at vertex u evolve as the pursuers move from the JPC at vertex u to the JPC at vertex v. (The inverse mapping is applied to $\overrightarrow{vu}$).

(iii) Finally, the reachable shadow label information across the graph is updated by propagating the reachable shadow labels, using the mappings attached to each edge, recursively across the graph, to determine what new reachable shadow labels, if any, arise due to the inclusion of the new sample v.

The SG-PEG data structure is useful for our problem because, starting from a root vertex at the pursuers' initial positions, executing a sequence of AddSample operations can eventually lead to a vertex being marked with an all-zero reachable shadow label. From there, a sequence of JPCs solving the problem can readily be extracted by walking backwards through the graph.

2.2.4 Fast Label Propagation via Shadow Influence Caching

To accelerate the propagation of shadow labels across edges of the SG-PEG, we construct shadow influence relations for each edge. For a given edge $w \to u$, let S_w and S_u denote the shadows at w and u. The *shadow influence relation* is a relation $R \subseteq S_w \times S_u$, in which $(s_w, s_u) \in R$ if and only if there exists a trajectory for the evader to travel undetected from s_w to s_u as the pursuers move from w to u. These

shadow influence relations can be computed according to the update rules described in Section 2.2.2. This is a time-consuming computational geometry operation, but it must be performed only once for each edge-pursuer pair. Once the relation R is computed, any future shadow label can be propagated efficiently by setting each shadow s_u at u to be contaminated if and only if there exists a contaminated shadow s_w in the source shadow label for which $(s_w, s_u) \in R$.

Chapter 3

General-Purpose Sampling Techniques

Each of the subsequent chapters rely on sampling methods to solve variations of the problem presented in Chapter 2. This chapter summarizes two novel sampling techniques that will be used to solve these variants.

3.1 Web Sampling (WS)

Stiffler and O'Kane proposed a handful of sampling distributions, but none of them proved significantly more effective than simple uniform random sampling of the joint configuration space. This approach appears not to be particularly effective in this domain, because the environment is comprised of regions whose surveyance is essential to finding a solution. To combat this issue, we propose a new strategy for generating samples, which specifically takes into account the visibility component of the problem.

A *web*, $W = P \cup Q$, is the union of two sets of points from the environment. The *initial points* P are placed sequentially and uniformly at random outside of the visibility polygon of any previously placed initial points. This process continues until $\bigcup_{p \in P} V(p) = F$. Next, we construct the intersection points Q. For each unique pair of points $(p_i, p_j) \in P \times P$, we add to Q a uniformly random point from $V(p_i) \cap V(p_j)$ if $V(p_i) \cap V(p_j) \neq \emptyset$ and $Q \cap V(p_i) \cap V(p_j) = \emptyset$. The initial points ensure that we have complete visibility coverage of the environment while the intersection points provide straight-line connectivity between the initial points, allowing a pursuer to move freely about an entire web. Webs generated in this way are used to generate sample JPCs, as described next. Figure 3.1 illustrates this concept and Algorithm 1 formalizes it.

Algorithm 1 GenerateWebPoints(F)

Input: an environment F

1: $P \leftarrow \emptyset$
2: $Q \leftarrow \emptyset$
3: $F' \leftarrow F$
4: **while** F' contains a shadow **do**
5: $\quad p \leftarrow F'$.RandomPoint()
6: $\quad P \leftarrow P \cup \{p\}$
7: $\quad F' \leftarrow F' \setminus V(p)$
8: **for each** $p_i, p_j \in P$, with $i < j$ **do**
9: $\quad A \leftarrow V(p_i) \cap V(p_j)$
10: $\quad$ **if** $A \neq \emptyset$ **and** $A \cap Q = \emptyset$ **then**
11: $\quad\quad q \leftarrow A$.RandomPoint()
12: $\quad\quad Q \leftarrow Q \cup \{q\}$
13: $W \leftarrow P \cup Q$
14: **return** W

We utilized webs to generate samples from F^n by generating a separate web for each pursuer and sampling, without replacement, from those points. If the points in any of the webs are exhausted, we generate another set of webs and continue.

It should be noted that the second conditional in Algorithm 1 line 10 greatly reduces the number of points in a web, as displayed in Figure 3.2. This allows the addition of more useful and unique samples to be added with significantly less data redundancy.

3.2 Robust Cycle Samples (RCS)

We wish to generate samples in a way that is effective in maintaining high connectivity. To do this, we construct a graph H whose vertex set is a web W and whose edge set consists of all unique pairs of points from $W \times W$ who can be joined by a line segment contained within the environment. The construction of webs ensures that H is, in essence, a visibility roadmap of F. Once H is constructed, a random vertex h is selected as the root node for a depth first search (DFS) on H. Throughout the DFS, we construct an ordered list D of vertices visited, both on their initial discovery and

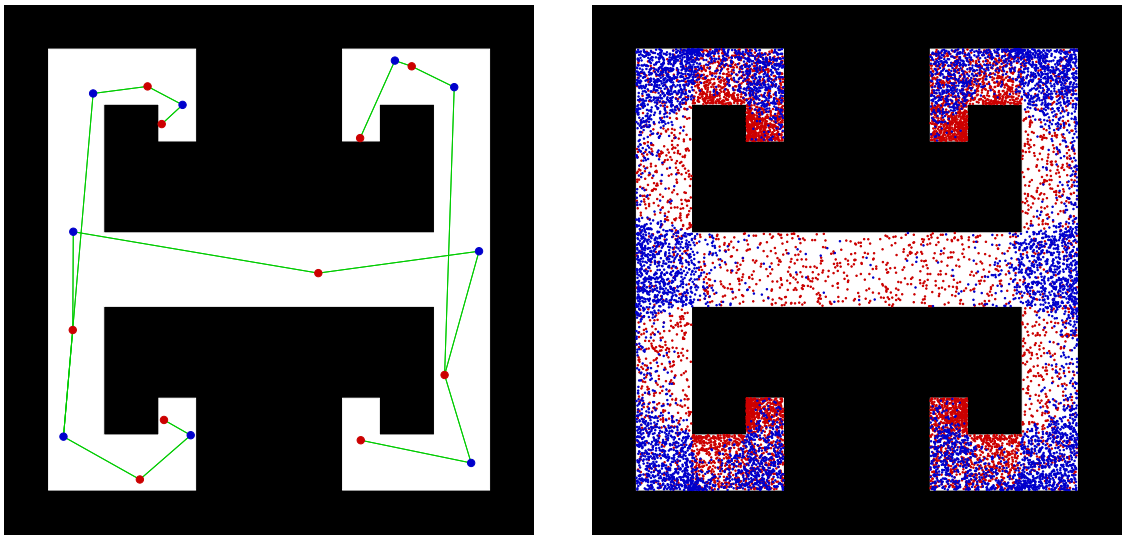
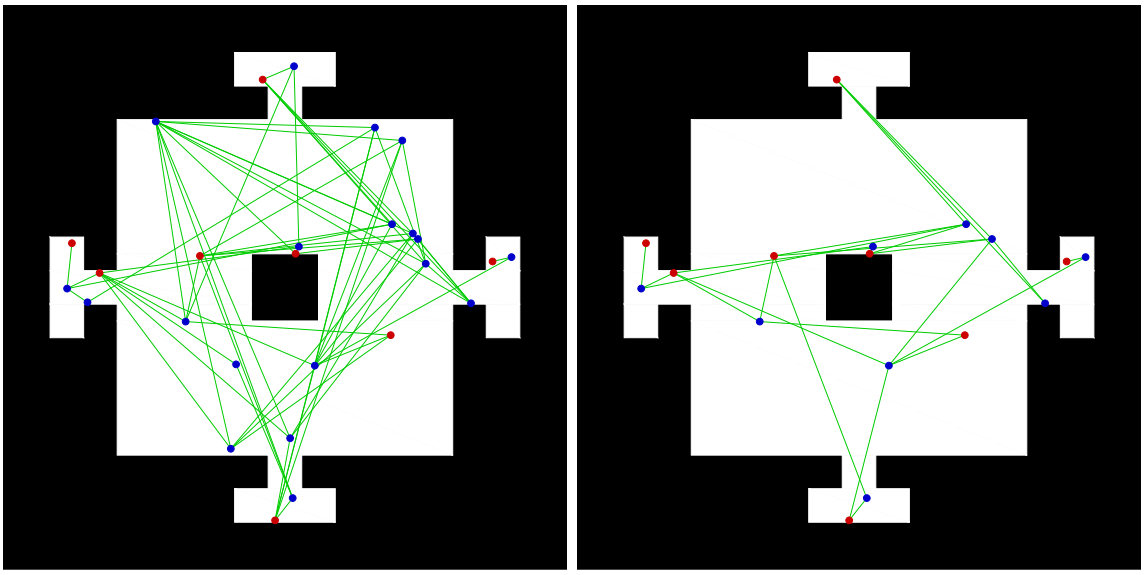


Figure 3.1: [left] An example web. Red points are initial points; the blue are intersection points. Green edges connect each intersection point to the initial points from which it is induced. [right] The distribution induced by web sampling, based upon 750 generated webs. Note the higher density at junctions and corners.

via backtracking. We continue this process until each point of H has been discovered and we successfully backtracked to the root node. We do not add the root node to D during the final stage of the DFS backtracking. This process produces a list of points $D = (D[0], \ldots, D[d-1])$ which contains every point from W at least once. The construction also guarantees connectivity between any adjacent points in D, giving a spanning walk, with repeats, around W. Since h is the first element added to D, $h = D[0]$.

To generate n-robot JPCs, we evenly space the n pursuers along D and walk along the cycle generated by the DFS on H. That is, we create the first sample JPC by choosing $D[(i-1) \cdot d/n]$ for each robot $i = 1 \ldots n$. Subsequent sample JPCs are generated in a similar fashion. See Figures 3.3, 3.4 and 3.5. To generate the k^{th} subsequent sample, we select $D[((i-1) \cdot d/n + k \mod d)]$ for each robot $i = 1 \ldots n$ and for each $k = 1, \ldots 2d/n$. Having k range over $1, \ldots, 2d/n$ ensures that each point in D has been visited by at least two pursuers, which is essential for the results of Chapter 6 and inconsequential in Chapters 4 and 5.



(a) Web with redundancy. (b) Web without redundancy.

Figure 3.2: A visualization of the noise reduction in webs. Again, the red points represent the initial points while the intersection points are drawn in blue.

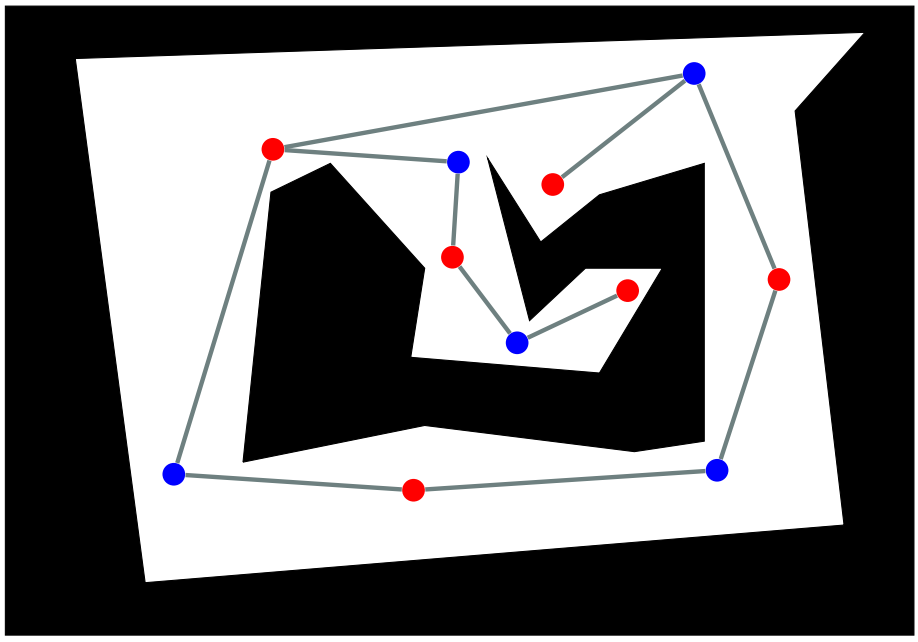


Figure 3.3: RCS stage 1: Generate a web.

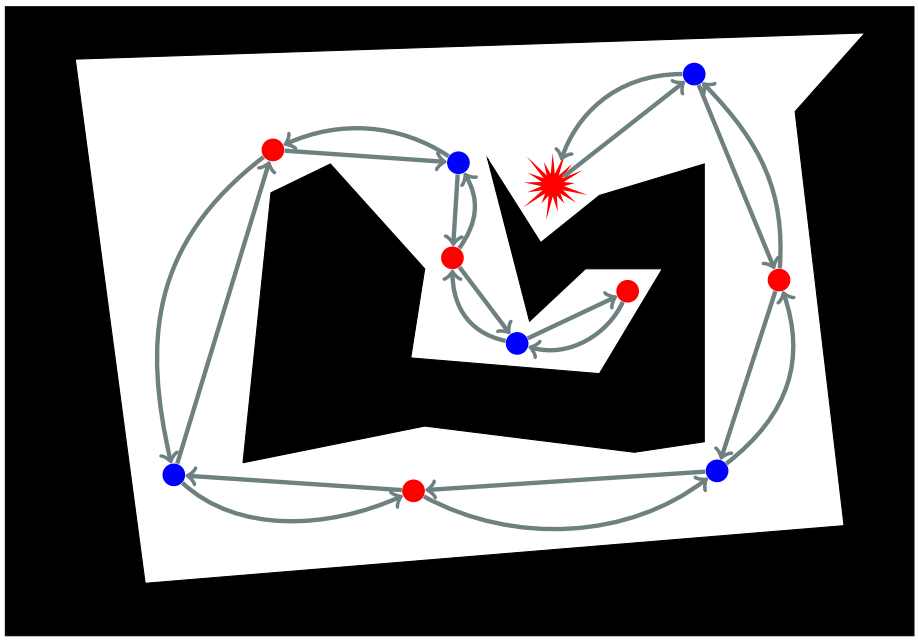


Figure 3.4: RCS stage 2: To form the walk D, DFS is performed on the web starting at the star vertex. In this example, the length the walk D is $d = 20$.

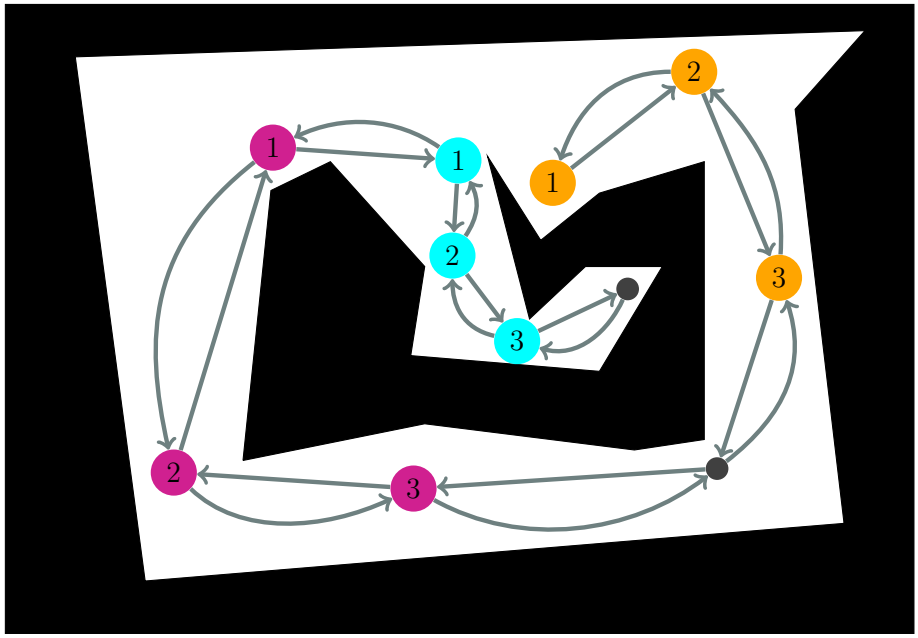


Figure 3.5: RCS stage 3: The first three samples generated by RCS. Notice that the 3 pursuers start $\lceil d/n \rceil = 7$ units apart from each other on D.

Chapter 4

Visibility Roadmap Sampling and Sequential Graph Construction

4.1 Introduction

One approach to overcome the computational challenge posed by this multiple-pursuer pursuit-evasion planning task [50], showed the feasibility of sampling-based techniques to generate joint motion strategies. Nevertheless, that approach has several important limitations.

(i) The existing algorithm requires a predetermined number of pursuers as input; it cannot adapt the number of pursuers to the complexity of the environment.

(ii) The solutions generated by this approach are of poor quality, in the sense that there nearly always are motions by one or more of the pursuers that do not actively contribute to the search.

This chapter makes two new contributions that address the limitations of the existing algorithm.

(i) We describe a method that eliminates the need for the number of pursuers to be provided as input, instead iteratively increasing the size of the team (Section 4.3.1).

(ii) We present a post-processing algorithm that improves solution quality (Section 4.3.2).

Section 4.4 presents quantitative evaluations of these new improvements on fixed environments while Section 4.5 does the same for randomly generated environments. Section 4.6 briefly discusses these results.

4.2 Objective

We consider both scenarios in which the number of pursuers is known and fixed, and in which the number of pursuers to utilize is determined by the algorithm as the plan is generated.

The algorithmic problem we address is to find a vector of motion strategies $\langle f_1, f_2, \ldots, f_n \rangle$ such that for any arbitrary evader curve e, we have $e(t) \in V(f_j(t))$ for some $t \geq 0$ and $j \in \{1, \ldots, n\}$. Such a collection of motion strategies is called a *solution*. Specifically, we consider two related problems.

Fixed: Given an environment F and a positive integer n, generate a solution using exactly n pursuers. Algorithms for this problem can be evaluated by examining both the time needed to generate a solution as well as the time needed for the pursuers to execute that solution.

Variable: Given an environment F, generate a solution that uses as few pursuers as possible. In addition to the run time and execution time criteria mentioned above, algorithms for this variant of the problem can also be judged by the number of pursuers utilized by the computed solution.

4.3 Algorithm Description

Our approach to this problem is based on two significant additions and one modification to the prior algorithm of Stiffler and O'Kane [50]. The basic idea of that prior algorithm is to generate random samples in F^n and use them to construct an SG-

PEG, continuing until the SG-PEG indicates that it contains a solution. Algorithm 2 shows the enhancements that we propose. New elements in comparison to the prior algorithm are highlighted: modifications to the sampling strategy in purple text and new additions in blue text. Note that, n, the number of robots has been ~~removed~~ in our variant. Details about these changes appear below.

Algorithm 2 SOLVE(F, ~~n~~, C, S)

Input: an environment F, ~~a number of pursuers n~~,
an expansion criterion C and a sampler S

1: $G \leftarrow$ empty SG-PEG for ~~n pursuers~~ 1 pursuer
2: $q \leftarrow$ S.GETSAMPLE()
3: G.ADDSAMPLE(q)
4: G.SETROOT(q)
5: **while** no solution has been found **do**
6: $\quad 1 \leftarrow S$.GETSAMPLE()
7: $\quad G$.ADDSAMPLE(q)
8: $\quad$ **if** C is met **then**
9: $\quad\quad G$.ADDPURSUER()
10: $X \leftarrow$ EXTRACTSOLUTION()
11: $X' \leftarrow$ REFINESOLUTION(X)
12: **return** X'

4.3.1 Variable numbers of pursuers

In the prior algorithm, the number of pursuers in the solution was required as an input. This information was necessary because the SG-PEG data structure stores joint configurations drawn from F^n; thus n must be known to construct an SG-PEG.

To alleviate this limitation, we instead propose a sequential process, in which the number of pursuers is gradually increased as the algorithm proceeds. Realizing this approach in the planner requires us to resolve two complications.

First, the algorithm requires a mechanism to transition, in mid-stream, from an n-pursuer SG-PEG to an $(n+1)$-pursuer SG-PEG (Line 9). One straightforward approach is to simply discard the existing vertices and edges and restart the search with an additional pursuer. (This is referred to as the 'Clear' option in Section 4.4.)

However, it may be preferable to ensure that our previous effort is not wasted. To this end, we propose a new method that *clones* the first pursuer in each vertex of the SG-PEG. That is, for each vertex in the SG-PEG, we replace its joint configuration $(p_1, p_2, \ldots, p_n)$ with $(p_1, p_2, \ldots, p_n, p_1)$. This adds an additional pursuer to the graph, without changing any of the reachable labels or edges. Thus, the cloning option is extremely efficient, but it leads to an SG-PEG in which the newly-added pursuer moves in parallel with another pursuer. Future edges added at the $(n+1)$-th layer will correspond to independent motions for these two robots (as they draw from their own unique set of webs).

Next, we must decide when to expand the number of pursuers (Line 8). We consider two options. First, we propose a method that devotes *fixed effort* to each stage of the search. The process begins by rapidly generating a trivial solution by placing pursuers until their visibility fully covers the environment. This gives a (generally very loose) upper bound N on the number of pursuers required. Then, given a target total run time of T_{limit} seconds, we apportion the time between the possible numbers of pursuers $1, \ldots, N$ via a Poisson distribution. The Poisson distribution was selected due to the placement of the mean, as well as its skew and shape. We choose a tunable parameter α which determines the fraction of time to spend on the final step that utilizes N pursuers, so that according to the definition of the Poisson distribution, we have $\alpha = \lambda^N e^{-\lambda}/N!$. From this equation, the algorithm numerically computes the Poisson parameter λ and allocates time $T_{\text{limit}}\lambda^{i-1}e^{-\lambda}/(i-1)!$ to the search with i pursuers before proceeding to $i+1$. Because of the existence of the upper bound N, this method will always produce a solution, although it may require a large number of pursuers.

As an alternative, we also consider a **stalled progress** approach, based upon monitoring the minimum sum of the contaminated shadow area across all vertices of the graph. (n.b. We have a solution if and only if this value reaches 0.) If this value

fails to improve by at least 5% after adding a fixed number of samples, M, we add a new pursuer. To enable a fair comparison to the fixed effort method, we once again return a trivial solution if no solution is found with fewer pursuers.

4.3.2 Solution refinement

Because of the sampling-based nature of this algorithm, its outputs are likely to have extraneous motions. This issue is noticeably more severe in our context than for traditional sampling-based motion planning because the generated solutions may travel several times along certain edges in an effort to clear specific shadows. In this section, we introduce a post processing method which takes a joint motion strategy and optimizes it by removing unnecessary pursuer motions.

Our method is similar to standard shortcut-based path smoothing. We select two points z_a and z_b at distance c along the solution path in F^n, and check whether taking a shortcut directly from z_a to z_b yields a path that is still a correct solution. Figure 4.1 depicts the process. Our check features one important difference from traditional path smoothing: In addition to ensuring that the refined solution is collision-free, we must also ensure that it remains a correct solution, i.e. that the refinement does not allow any shadows to remain contaminated. We do this by tracking forward through the shortened path, applying the shadow events experienced along that path.

Given this shortcut operation, we greedily optimize the path, proceeding systematically over decreasing values of c and increasing positions of z_a. (From these two, z_b is readily computed.) Each time we discover a shortcut yielding a correct solution, that shortened solution replaces the previous solution, and the process continues. See Figure 4.2.

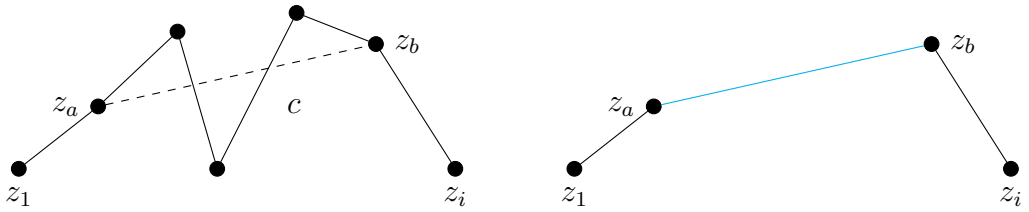


Figure 4.1: A cut of length c during solution refinement.

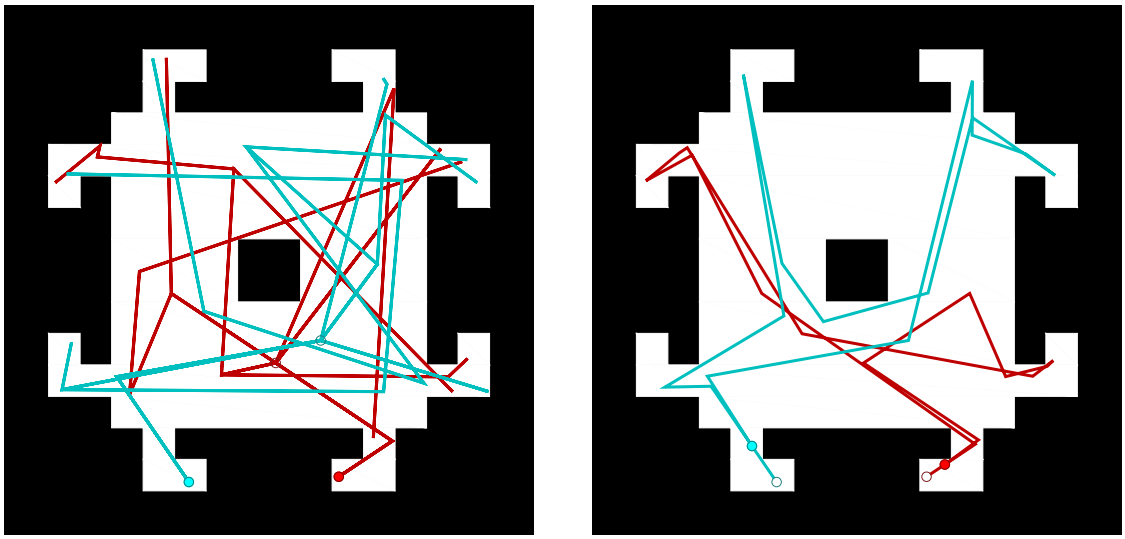


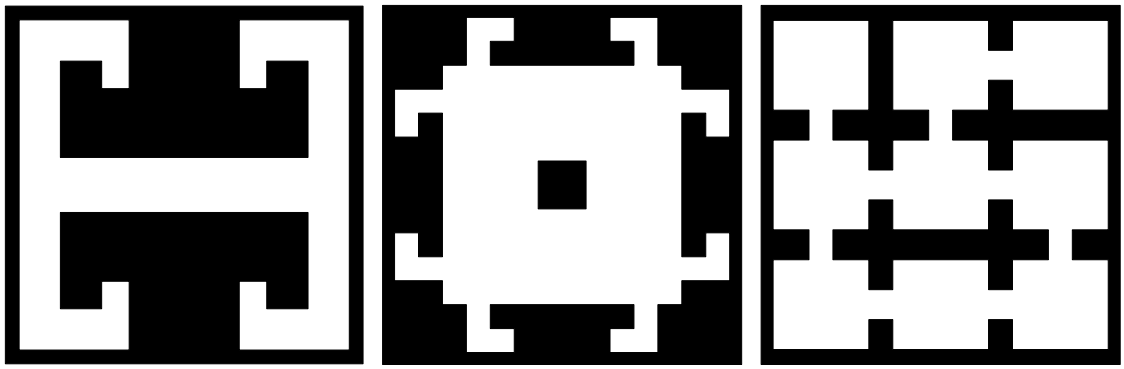
Figure 4.2: Refining a solution. [left] Before. [right] After.

4.4 Selected Environment Evaluation

This section evaluates the performance of the proposed approach. We implemented the algorithm in C++ and executed it on a computer with an Intel i5-9600K processor and 16GB of memory, running Ubuntu 20.04.2 64-bit.

We performed simulations in the environments shown in Figure 4.3. This selection of environments is intended to provide a variety common environmental traits. The H environment (Figure 4.3(a)) contains narrow corridors, the Spider environment (Figure 4.3(b)) possesses numerous hard to reach areas, and the Office environment (Figure 4.3(c)) is somewhat uniformly spread out, with a complex boundary.

We generated solution strategies for these environments using both classes of algorithms described thus far: those that utilize a fixed number of pursuers and those



(a) The H environment. (b) The Spider environment. (c) The Office environment.

Figure 4.3: The environments we used for our simulations.

that can vary the number of pursuers. In the former case, we considered RCS, WS (see Chapter 3) and uniform random sampling (SO14) [50] and varied the fixed number of robots between 2 and 5. In the latter case, we considered all four combinations of expansion criterion (fixed effort (FE) or stalled progress (SP)) and graph expansion method (clone pursuer (Clone) or clear progress (Clear)) with RCS as the sampling method.

For each of these scenarios, we executed 25 trials and recorded the computation time in seconds, number of pursuers used in the returned solution, as well as the total numbers of vertices and edges generated. Each experiment was allotted 10 minutes of computation time; if no solution was produced in that time, we considered the trial to be a failure. Failed trials are excluded from the statistics, but it should be noted that if they were included, they would contribute a run time of at least 10 minutes. The results, summarized via the means (μ) and standard deviations (σ), appear in Tables 4.1-4.5.

A number of conclusions may be drawn from the results.

RCS outperforms WS which outperforms SO14: First, in all of the tested environments, our RCS and WS methods outperformed SO14 by a wide margin.

Table 4.1: Simulation results ($n = 2$).

	success rate	planning time (s)		Vertices		Edges	
		μ	σ	μ	σ	μ	σ
Figure 4.3(a)							
RCS	100%	4.61	1.60	22.96	4.17	25.36	6.10
WS	100%	4.83	2.40	26.88	7.61	27.72	10.78
SO14	100%	38.96	16.87	93.72	27.06	202.48	112.36
Figure 4.3(b)							
RCS	100%	31.18	13.24	33.04	7.09	54.96	21.05
WS	100%	25.48	6.90	30.60	6.61	45.20	13.71
SO14	28%	413.03	124.82	89.43	12.88	761.00	162.60
Figure 4.3(c)							
RCS	100%	17.10	14.12	32.48	11.06	39.96	26.52
WS	100%	17.33	7.16	30.44	5.67	37.44	12.04
SO14	100%	35.41	18.49	80.76	21.48	99.64	51.01

Table 4.2: Simulation results ($n = 3$).

	success rate	planning time (s)		Vertices		Edges	
		μ	σ	μ	σ	μ	σ
Figure 4.3(a)							
RCS	100%	2.82	0.90	19.04	3.49	15.60	4.30
WS	100%	5.55	3.30	27.76	8.49	25.08	9.49
SO14	100%	54.04	35.02	351.60	700.96	4645.12	19181.24
Figure 4.3(b)							
RCS	100%	19.81	8.19	27.28	7.57	26.04	9.79
WS	100%	30.82	11.99	32.56	7.87	34.52	12.10
SO14	44%	433.23	107.05	72.36	11.77	451.36	211.79
Figure 4.3(c)							
RCS	100%	15.82	12.04	26.24	7.41	24.28	11.52
WS	100%	20.26	10.50	31.92	6.18	30.60	10.11
SO14	100%	37.90	28.84	302.92	101.71	356.28	257.82

RCS proved to be a modest improvement over WS, this is likely due to the high connectivity of the samples drawn.

Clone and Clear appear to be complementary: For the simulations with a variable number of pursuers, we look to compare the performance between the cloning method and the clear method when expanding a graph. When using both FE and SP as the expansion criterion, the results are remarkably similar between the two

Table 4.3: Simulation results ($n = 4$).

	success rate	planning time (s) μ	planning time (s) σ	Vertices μ	Vertices σ	Edges μ	Edges σ
Figure 4.3(a)							
RCS	100%	3.17	1.66	13.24	3.53	9.92	4.90
WS	100%	6.31	2.25	24.76	3.35	21.00	4.12
SO14	100%	84.48	49.37	838.20	1058.34	5468.88	16473.30
Figure 4.3(b)							
RCS	100%	20.64	8.35	21.04	4.92	19.68	6.94
WS	100%	34.81	13.66	36.44	5.60	35.08	9.48
SO14	64%	391.07	149.00	74.44	42.42	375.44	663.06
Figure 4.3(c)							
RCS	100%	14.60	9.11	18.32	6.08	14.40	7.26
WS	100%	17.75	10.05	30.92	6.87	26.04	9.39
SO14	100%	104.94	89.89	1619.36	1007.63	2593.36	3345.66

Table 4.4: Simulation results ($n = 5$).

	success rate	planning time (s) μ	planning time (s) σ	Vertices μ	Vertices σ	Edges μ	Edges σ
Figure 4.3(a)							
RCS	100%	2.97	1.22	9.72	3.20	6.36	3.51
WS	100%	7.47	3.32	23.80	3.93	19.20	4.81
SO14	92%	110.87	57.91	1463.13	1204.17	3027.48	4456.63
Figure 4.3(b)							
RCS	100%	17.54	6.71	16.52	4.05	13.52	5.34
WS	100%	37.10	18.85	36.92	5.07	33.44	7.20
SO14	72%	303.74	99.81	121.06	149.16	1060.11	3392.10
Figure 4.3(c)							
RCS	100%	12.37	4.44	14.08	3.38	10.24	4.10
WS	100%	20.19	12.12	30.48	6.94	24.32	8.86
SO14	80%	257.61	117.79	4511.50	1385.38	3211.15	1850.81

methods. At first glance, one may assume the cloning method should outperform the clearing method due to the larger amount of information the graph contains. This however, is not always the case, because an increased number of vertices means an increased amount of time to add each new sample, due to the propagation of reachable labels across the graph.

Allowing the number of pursuers to vary incurs only a modest computational cost: Lastly, we compare the fixed number of pursuers using RCS, to the

Table 4.5: Variable simulation results.

	num robots		planning time (s)		Vertices		Edges	
	μ	σ	μ	σ	μ	σ	μ	σ
Figure 4.3(a)								
FE Clone ($\alpha = 0.001$)	2.00	0.00	6.18	2.46	35.84	5.81	35.68	8.75
FE Clear ($\alpha = 0.001$)	2.00	0.00	6.10	1.92	39.96	5.54	43.36	8.61
SP Clone ($M = 50$)	2.00	0.00	5.75	1.92	34.72	4.67	32.60	5.80
SP Clear ($M = 50$)	2.00	0.00	6.13	1.77	42.36	3.99	45.08	6.41
Figure 4.3(b)								
FE Clone ($\alpha = 0.001$)	2.60	0.76	40.59	24.01	41.12	7.28	49.96	15.73
FE Clear ($\alpha = 0.001$)	2.40	0.50	36.15	17.01	55.04	10.84	71.80	24.45
SP Clone ($M = 50$)	2.00	0.00	32.40	12.07	38.72	6.02	54.52	12.78
SP Clear ($M = 50$)	2.00	0.00	35.71	12.87	53.92	7.12	82.04	18.16
Figure 4.3(c)								
FE Clone ($\alpha = 0.001$)	2.08	0.28	28.26	18.99	48.36	11.59	58.56	25.43
FE Clear ($\alpha = 0.001$)	2.04	0.20	24.29	11.94	56.04	8.95	69.40	17.58
SP Clone ($M = 50$)	2.00	0.00	22.47	7.78	46.80	8.55	53.04	13.70
SP Clear ($M = 50$)	2.00	0.00	20.34	7.10	52.80	6.63	60.72	11.32

variable number of pursuers methods. For each environment, the run times of the variable number of pursuers were within approximately a factor of 2 of that of the fixed number of pursuers. A portion of this additional time comes from the variable number of pursuers constructing a single pursuer graph, in environments which all require at least 2 pursuers to solve.

Finally, for each environment, we refined all 25 solutions generated by WS with 2 pursuers. The results are summarized in Table 4.6, which shows the mean and standard deviation of the computation time (in seconds) along with the length of the solution path (in meters) before and after it was refined. The results show that this approach effectively and consistently reduces the path lengths.

4.5 Random Environment Evaluation

In this section, we evaluate the increasing complexity of environments with additional holes and rooms. We randomly generated 20 environments for each combination of 0-4 holes and 0-4 rooms. These environments can be found in Appendix A. Each

Table 4.6: Refinement results for the solutions produced by WS ($n = 2$).

	comp time (s)		length before (m)		length after (m)	
	μ	σ	μ	σ	μ	σ
H	6.4	5.6	256.5	109.4	100.0	31.7
Spider	50.9	61.9	360.3	222.3	168.8	139.0
Office	29.6	37.5	254.5	114.5	136.9	97.4

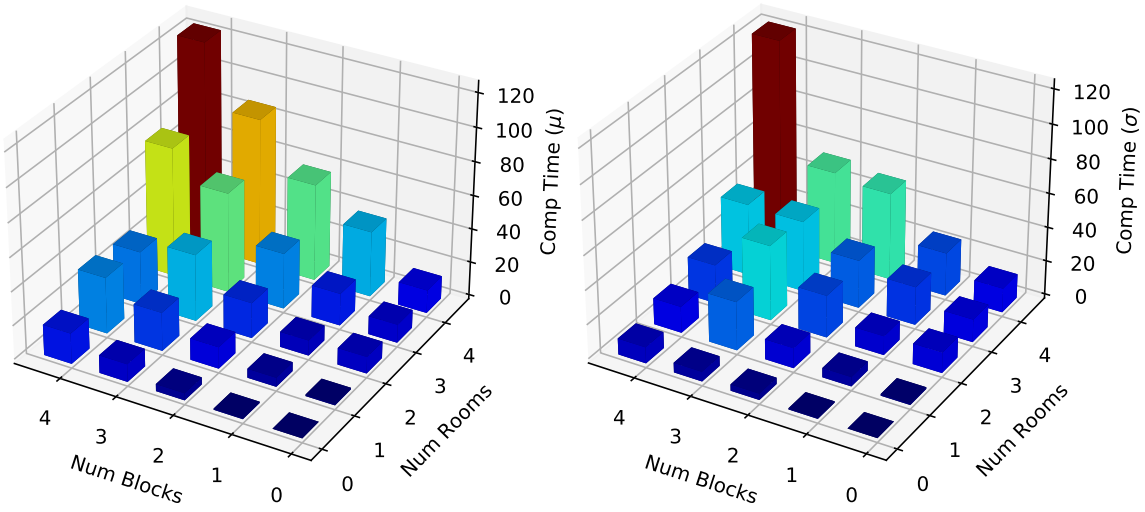


Figure 4.4: The mean [left] and standard deviation [right] of the computation time over random environments.

experiment was allotted 10 minutes to generate a solution utilizing 3 pursuers and RCS sampling. For each environment, we ran 10 simulations. Figures 4.4-4.6 display the average and standard deviation for the computation time (in seconds), as well as the number of vertices and edges in the resulting SG-PEG. These statistics were generated over all trials with a rooms and b blocks, that is, the average and standard deviation considered all 10 trials over all 20 such randomly generated environments (200 simulations in total). We are able to conclude that increasing the number of blocks and rooms increases the mean and standard deviation of the computation time, number of vertices and number of edges. The number of blocks and rooms seem to add an equal amount of difficulty, as Figures 4.4-4.6 are roughly symmetric about the diagonal.

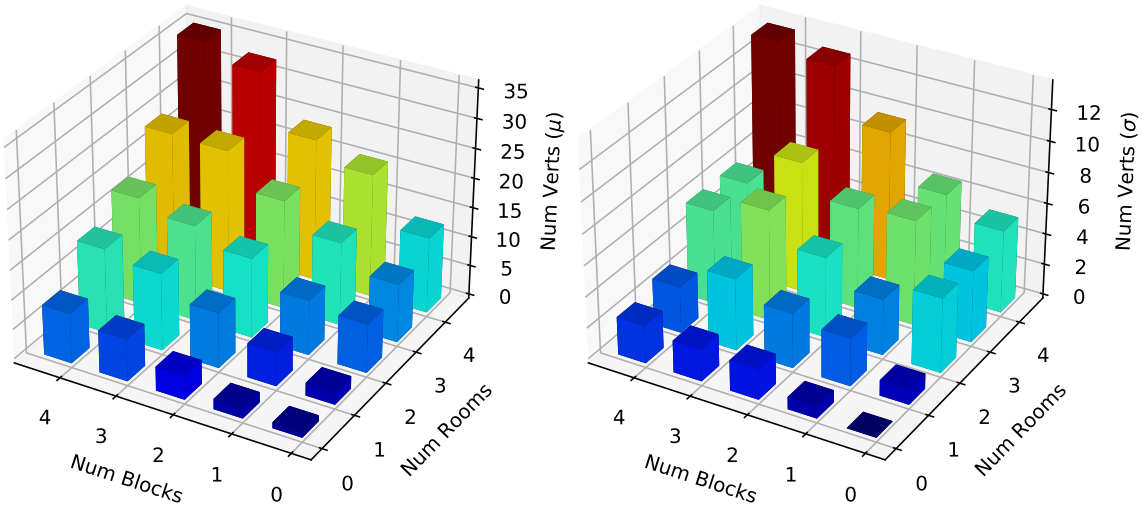


Figure 4.5: The mean [left] and standard deviation [right] of the number of vertices over random environments.

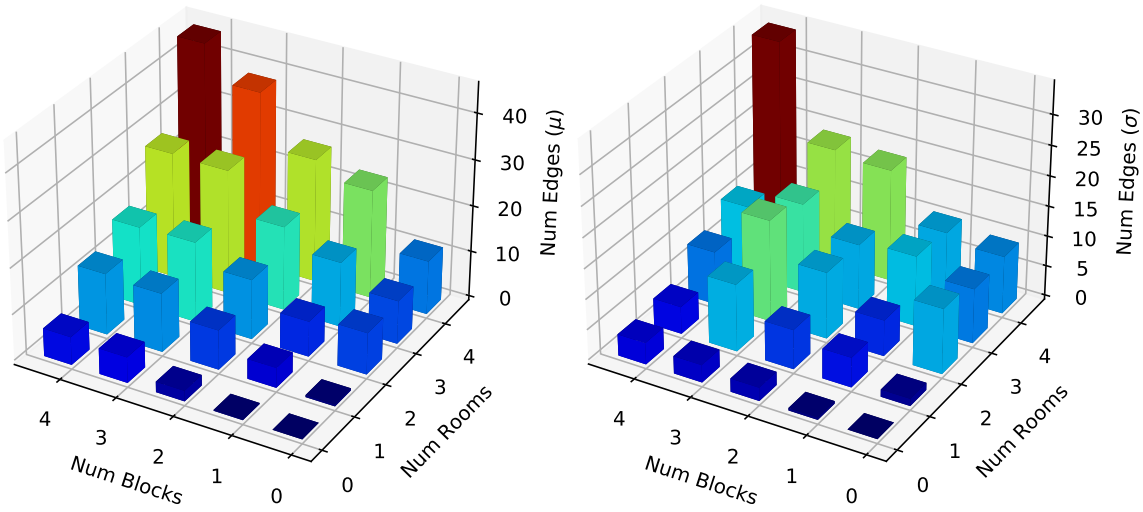


Figure 4.6: The mean [left] and standard deviation [right] of the number of edges over random environments.

4.6 Conclusion

This chapter presents a sampling-based algorithm for a visibility-based pursuit-evasion problem that generates a joint motion strategy for a team of robots in a polygonal environment. The contributions included an iterative algorithm for generating a joint motion strategy for the pursuers, and a post-processing path-smoothing algorithm that refines the strategy returned by the main algorithm. The algorithm was shown to outperform existing techniques.

Chapter 5

Rapid Recovery from Robot Failures

5.1 Introduction

Though progress has been made on several forms of the multi-robot visibility-based pursuit-evasion problem, a key limitation within much of the existing work is an inability to adapt in cases of unrecoverable failures of individual robots. Such failures are particularly likely to occur in the very application domains for which these methods are well-suited.

Figure 5.1 provides a simple illustration of this problem, in a context wherein each pursuer is equipped with an omnidirectional sensor that can detect the evaders within its line of sight. The objective is to move the pursuers along paths that are guaranteed to locate the evaders, even if the evaders may move arbitrarily quickly. An initial plan to search this environment using three robots appears on the left of Figure 5.1. Now suppose the robot stationed at the top left of the environment fails during the plan's execution, as shown on the right of Figure 5.1. In this case, any evader hiding in the center of the environment at that time can now escape to the top left corner without being detected. Thus, the initial plan is no longer guaranteed to locate the evader.

In response to this limitation, this chapter proposes a new approach for this form of visibility-based pursuit-evasion problem, suitable for contexts in which failures of robots are likely. Because generating pursuit plans for this domain is computationally challenging [49], we introduce a method for replanning in cases of robot failure that

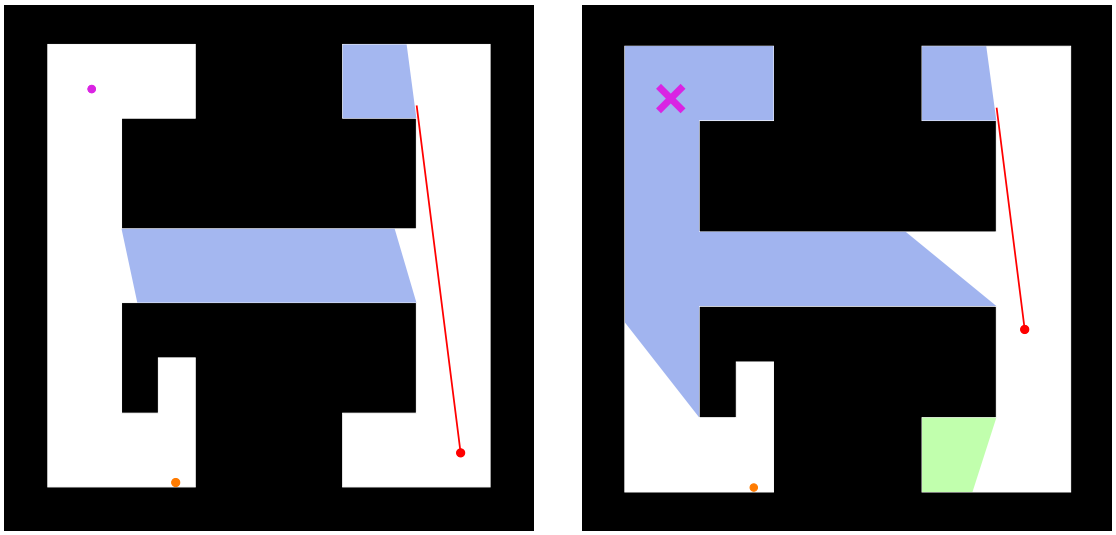


Figure 5.1: [left] An initial joint plan for 3 robots to search a simple environment for an evader. Two robots on the left remain stationary and monitor that side of the environment while a third moves to search the right. [right] The robot in the top left corner fails unexpectedly. As a result, the initial plan is no longer correct, because it cannot locate an evader that moves into the top left corner of the environment.

leverages information derived from the previous plan to accelerate the construction of new plans for the unexpectedly-smaller teams. Figure 5.2 shows an example of recovery from the failure depicted in Figure 5.1.

Recall that, for planning the motions of n robots in this pursuit-evasion problem, the SG-PEG graph represents the connectivity of the joint configurations space with vertices that represent joint configurations and edges that represent collision free movements, much like a traditional probabilistic roadmap. In addition, each vertex is labeled with information that encodes which hidden portions of the environment can be cleared of unseen evaders along some walk in the graph ending at that vertex. The new replanning approach modifies this data structure when a robot fails to reflect the removal of that robot from the search. The remaining graph then provides a valuable starting point for the process of planning to solve the problem with the remaining $n-1$ robots. The algorithm then expands this graph by adding additional vertices, attempting to recover the contributions made by the failed robot.

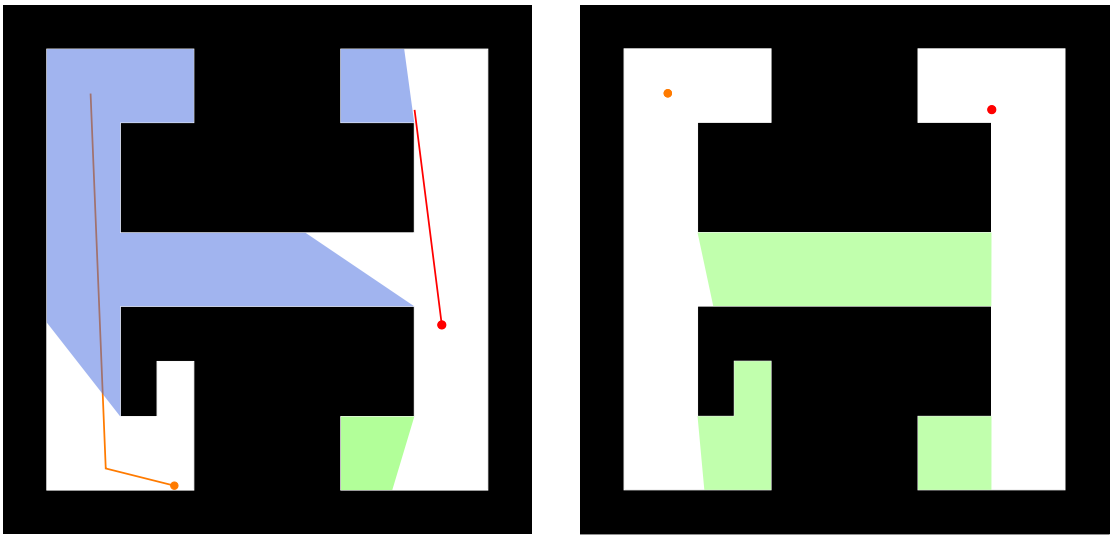


Figure 5.2: [left] The robots from Figure 5.1 replan, using the proposed algorithm, to form a correct plan for the remaining 2 robots. [right] The plan completes successfully.

This work is, to the best of our best knowledge, the first to address the problem of recovery from pursuer failures for this form of pursuit-evasion problem. In the remainder of this chapter, we describe robot failures in the context of our problem as well as discuss and evaluate the algorithm (Sections 5.3.1, 5.3, and 5.4 respectively) before concluding with discussion (Section 5.5).

5.2 Pursuer Failures

The basic problem described so far has been addressed in prior work in a number of different ways. The new contribution in this chapter is to consider this problem in an online setting, in which the pursuer robots may fail during the execution of a plan. Such failures are assumed to be both unpredictable and permanent, but known to all of the pursuers when they occur.

Recall from Chapter 2, we consider the case in which the n pursuers are executing the paths $f_i : [0, T] \to F$ for $i \in \{1, \ldots, n\}$, with termination time T. Suppose that at some time $0 \leq \ddot{t} < T$, the k^{th} pursuer fails. The *replanning problem* addressed in

this chapter is to generate new pursuer paths $f'_i : [\ddot{\hat{t}}, T'] \to F$ for $i \in \{1, \ldots, n\} \setminus \{k\}$. These revised paths must begin at the pursuers' locations at the time of failure, so that for each surviving pursuer, we have $f'_i(\ddot{\hat{t}}) = f_i(\ddot{\hat{t}})$. The revised paths end at a new termination time T'.

Generalizing the objective from the original failureless model, we say that the pursuers' execution of the prefix of $f_1, \ldots, f_n$ up to time $\ddot{\hat{t}}$, followed by $f'_1, \ldots, f'_n$ from time $\ddot{\hat{t}}$ to T' is a *solution* if, for any evader trajectory e, either (i) there exists a time $t_0 \in [0, \ddot{\hat{t}}]$, such that $e(t_0) \in \bigcup_{i \leq n} V(f_i(t_0))$, or (ii) there exists a time $t_0 \in [\ddot{\hat{t}}, T']$, such that $e(t_0) \in \bigcup_{i \leq n; i \neq k} V(f'_i(t_0))$. That is, we seek to guarantee that the evader is seen by any of the robots at some time before or after the pursuer failure, or by one of the non-failing robots at some time after the failure. Similar but notationally tedious generalizations can be made for multiple failures within a single execution.

Fortunately, we can also generalize the notions of shadow label updates to account for the abrupt change in shadows that occur at time $\ddot{\hat{t}}$. Specifically, a shadow extant immediately after a robot failure is contaminated if and only if it intersects with a contaminated shadow from immediately before the failure occurred. Notice, for example, in the right portion of Figure 5.1, that the large shadow encompassing the center and upper left portion of the environment is marked contaminated because it overlaps the central shadow which was contaminated before the failure. In contrast, the smaller shadow in the lower right has a clear label after the failure, because the only pre-failure shadow with which it intersects (namely, itself) had a clear label. This feature of the definition of success, which allows shadows to remain clear even across a failure of one of the pursuers, is crucial because it allows the pursuers the possibility of retaining some of their progress (i.e. cleared shadows) toward completing the task, rather than starting from scratch each time.

5.3 Algorithm Description

This section provides a detailed description of our algorithm. As in Chapter 4, we take a sampling-based approach. The basic idea is to construct a roadmap within the pursuers' joint configuration space, using the sample-generated pursuit-evasion graph (SG-PEG) (recall Section 2.1 [50]). We leverage this data structure in a new way by introducing new sampling strategies designed to rapidly re-acquire a solution in cases where a pursuer must be removed.

The core of the algorithm is a method called DropRobot which, given a solution path for k robots (for some k), uses an SG-PEG to attempt to rapidly generate a solution for $k-1$ robots, using the original k-robot solution as a guide. Our algorithm relies upon DropRobot both to generate an initial solution for the full set of n robots —by iteratively reducing from a rapidly-generated trivial solution— and for replanning when a pursuer fails.

The remainder of this section presents details of the method. We describe the DropRobot method (Section 5.3.1) and how that method is used to generate the initial solution and for replanning (Section 5.3.2).

5.3.1 Dropping a robot

Suppose k pursuers are at some JPC q with shadow label ℓ, and have computed a sequence of future JPCs to visit that will solve the problem from that point, reaching JPC with an all-clear shadow label. How can we use this information to construct a new solution that can be executed from this point by only $k-1$ of these pursuers, removing one particular pursuer from the solution? Notice that this scenario applies both to the case of a failed pursuer (in which case q and ℓ can be derived from the current state when the failure occurred, and ℓ may mark some shadows as clear) and to a complete solution starting from the pursuers' starting position and all-contaminated

shadow label. To simplify the notation below, we assume without loss of generality that the n^{th} pursuer is the one being removed.

The DropRobot method, shown in Algorithm 3, solves this problem. The algorithm constructs an SG-PEG G_{k-1}, starting with a root vertex at which the n^{th} pursuer has been removed and the shadow label has been updated accordingly. From there, it adds a collection of *junction samples*, designed to recover information lost due to the removal of the n^{th} pursuer at each step of the existing solution. If G_{k-1} does not contain a solution after that step, DropRobot continues by inserting additional samples generated by RCS are designed to provide good coverage, in the sense of visibility, of the environment. The process continues until a solution is found, or until some arbitrary timeout expires. Details about junction sampling and web sampling appear below.

Algorithm 3 DropRobot$(F, k, q_1, \ldots, q_m, \ell)$

Input: An environment F; a positive integer k; a sequence $q_1, \ldots, q_m$ of k-pursuer JPCs; a shadow label ℓ for q_1.

Output: A sequence $q'_1, \ldots, q'_{m'}$ of $(k-1)$-pursuer JPCs leading to an all clear shadow label at $q'_{m'}$ or Failed.

```
1: G_{k-1} ← new SG-PEG for k − 1 pursuers
2: ⟨p_1, ..., p_k⟩ ← q_1
3: r ← G_{k-1}.AddRoot(⟨p_1, ..., p_{k-1}⟩)
4: ℓ' ← ℓ updated for the removal of p_n
5: r.AddReachable(ℓ')
6: for i ← 1, ..., m do
7:     AddJunctionSamples(k, G_{k-1}, q_i)
8: while G_{k-1} has no solution and time remains do
9:     q ← RCS(G_{k-1})
10:    G.addSample(q)
11: if G_{k-1} has a solution then
12:    return G_{k-1}.ExtractSolution()
13: else
14:    return Failed
```

Algorithm 4 AddJunctionSamples(k, G_{k-1}, q)

Input: A positive integer k; an SG-PEG G_{k-1} for $k-1$ pursuers; a k-pursuer JPC q
Output: No return value, but samples are added to G_{k-1}.

1: $\langle p_1, \ldots, p_k \rangle \leftarrow q$
2: G_{k-1}.addSample($\langle p_1, \ldots, p_{n-1} \rangle$)
3: **for** $i \leftarrow 1, \ldots, n-1$ **do**
4: **if** $V(p_i) \cap V(p_n) \neq \emptyset$ **then**
5: $z \leftarrow$ random point in $V(p_i) \cap V(p_n)$
6: G.addSample($\langle p_1, p_2, \ldots, z, \ldots, p_{n-1} \rangle$)
7: G.addSample($\langle p_1, p_2, \ldots, p_n, \ldots, p_{n-1} \rangle$)

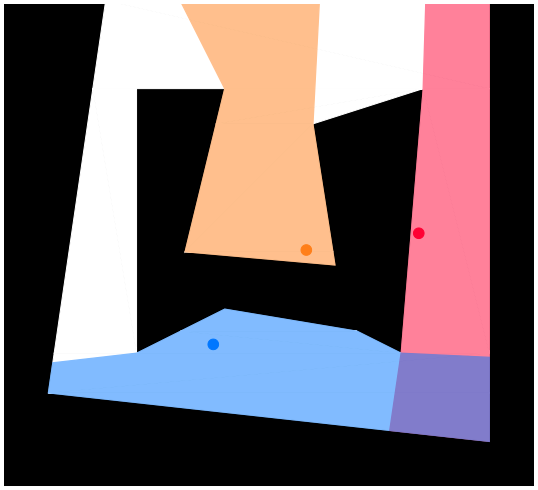
Junction sampling

The objective in junction sampling is, informally, to add vertices and edges to the SG-PEG that allow remaining pursuers to 'fill in' for the removed robot, wherever possible. Figure 5.3 shows a simple example of a pursuer removed from a JPC during DropRobot. In this example, the lower pursuer is removed, leaving the bottom portion of the environment unobserved. Junction sampling adds new samples that provide a path within the SG-PEG for the rightmost robot to visit the site of this lower portion.

This process, called AddJunctionSamples, is formalized in Algorithm 4. In the general case, the algorithm identifies a remaining pursuer at a position p_i for which the visibility polygon intersects the visibility polygon of the position p_n of the removed pursuer. When this relationship is detected, we add a sample that places the i^{th} pursuer in the intersection of the visibility polygons (see Figure 5.3c) and another that places the i^{th} pursuer at the former location of the n^{th} pursuer (Figure 5.3d). This process is repeated for each i and, via repeated calls to AddJunctionSamples, each step of the previous k-pursuer solution.

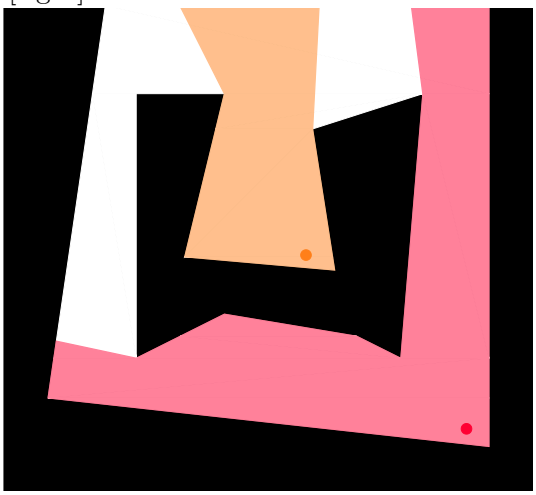
RCS

Though the structures introduced by junction sampling may be sufficient to build a SG-PEG that can generate a solution with $k-1$ pursuers, such success cannot be

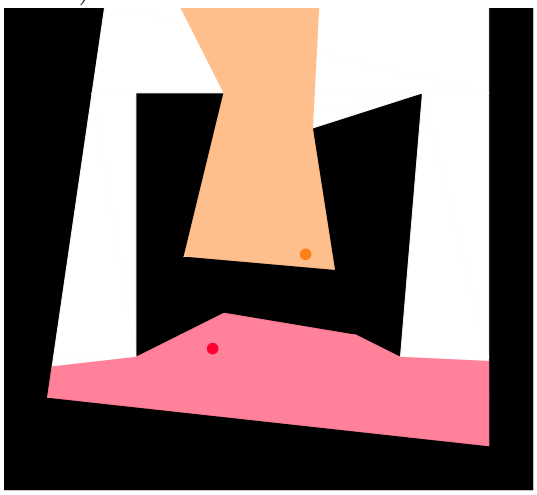


(a) The initial JCP. The n^{th} pursuer is blue [bottom], and the i^{th} pursuer is red [right].

(b) The first sample that will be added. The n^{th} pursuer is removed (Algorithm 4, line 2).



(c) The second sample. The n^{th} pursuer is removed and the i^{th} pursuer moves to a random point in $V(p_i) \cap V(p_n)$. (Algorithm 4, line 6).



(d) The third sample that will be added. The n^{th} pursuer is removed and the i^{th} pursuer takes its place. (Algorithm 4, line 7).

Figure 5.3: An example of junction sampling.

guaranteed. Therefore, after exhausting the junction samples, Algorithm 3 continues with a broader sampling strategy called RCS (Chapter 3).

5.3.2 Planning, execution, and replanning

Armed with the DropRobot method, we can consider how to use that algorithm for the overall problem.

Generating the initial solution

To begin, we must generate an initial solution that the full complement of n robots can begin to execute. To do so, we simply call RCS to generate n pursuer JPCs until a solution is found.

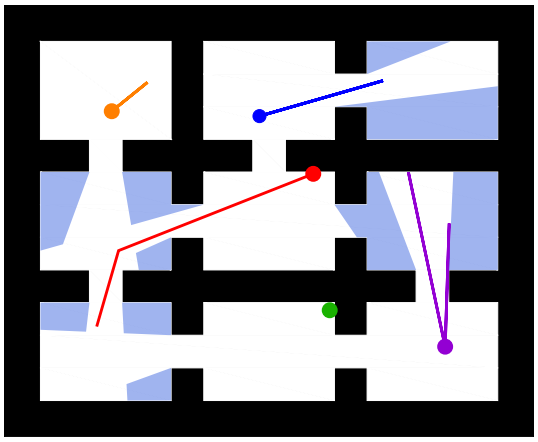
Replanning after pursuer failures

If, during the execution of the search, a pursuer fails for some reason, a replanning operation is required. In that case, we pause the pursuers' movement until a new solution with one fewer pursuer is generated. This new solution may be generated directly by DropRobot. Notice that the inputs to that algorithm include the current state of the search (including the current JPC and the current shadow label), which are leveraged to replan more rapidly than planning from scratch each time. Once a new solution is computed, the pursuers resume their search.

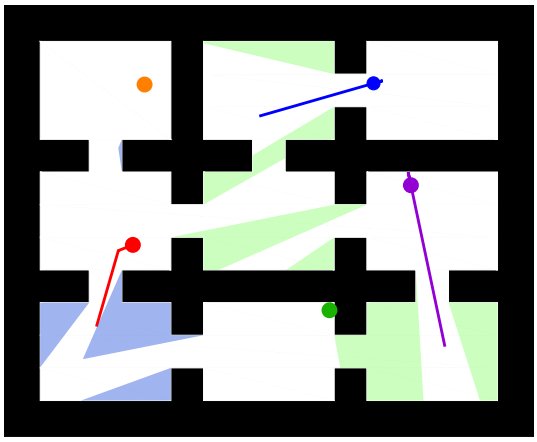
5.4 Evaluation

We implemented our algorithm in C++ and executed the code on an Ubuntu 20.04 desktop equipped with an Intel i5-9600K CPU and 16GB of RAM.

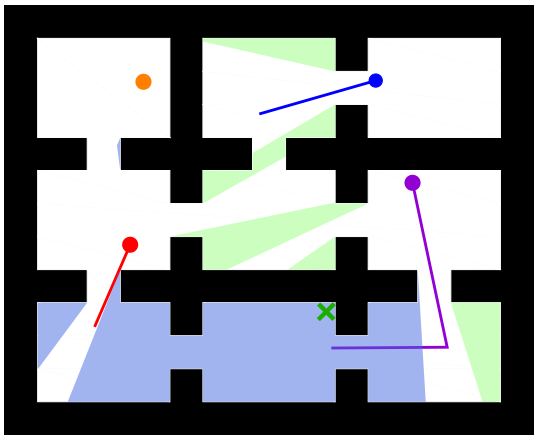
An example execution is illustrated in Figure 5.4. First, an initial solution is generated (Figure 5.4a). Next, Figures 5.4b,c represent the input and output of



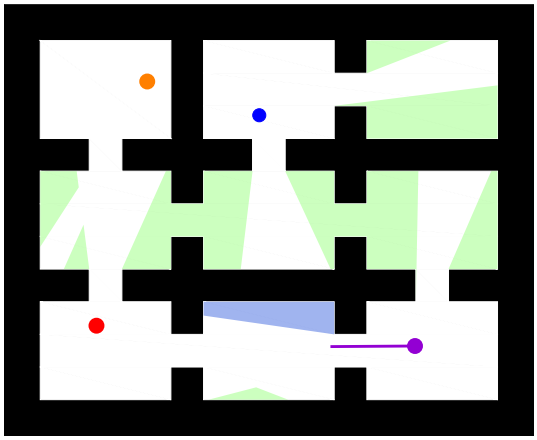
(a) An initial solution that utilizes 5 pursuers.



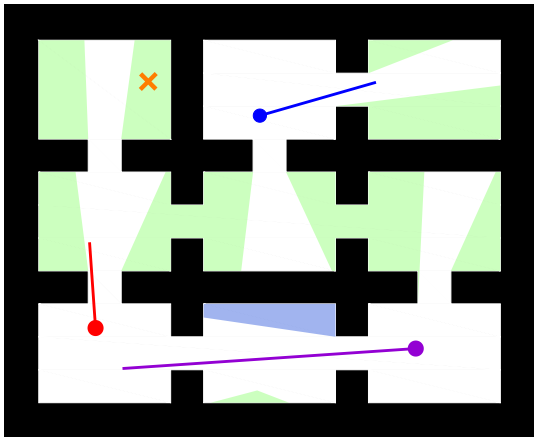
(b) The problem state right before the green pursuer fails.



(c) The new solution paths generated after the green pursuer fails.



(d) The problem state right before the orange pursuer fails.

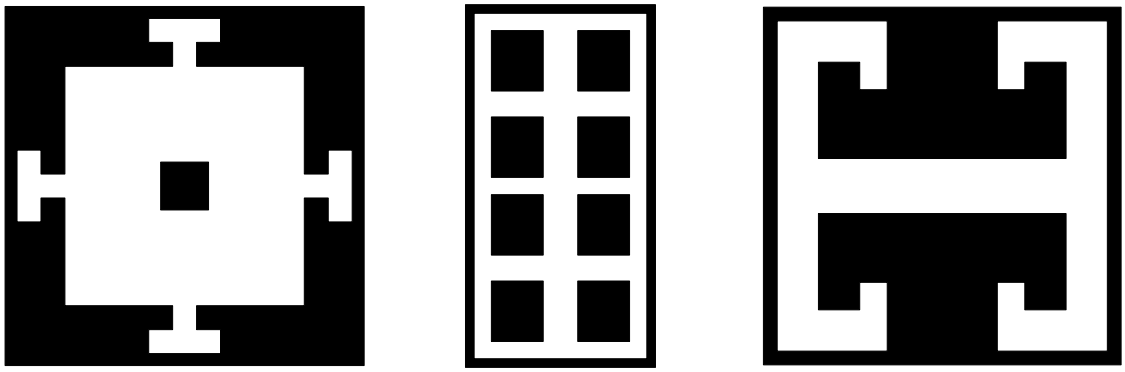


(e) The new solution paths generated after the orange pursuer fails.

Figure 5.4: Snapshots of our algorithm through a single successful execution ($n = 5, m = 2$).

Algorithm 3 when the green pursuer malfunctions. Similarly, Figures 5.4d,e show the state before and after the failure of the orange pursuer.

We simulated teams initially consisting of $n = 5$ pursuers in three different environments, depicted in Figure 5.5. These environments were selected because they highlight several interesting attributes, such as hard to reach corners, narrow corridors, and evenly spaced obstacles. In particular, we compare the algorithm presented in Section 5.3 ('Chp') against our previous algorithm utilizing RCS from Section 4.3 ('RCS'), which was designed for the failure-free setting, as a baseline. During each execution, we simulated m pursuer failures. For each failure, a randomly-selected pursuer was removed when the pursuers had completed a percentage β of their planned paths. For RCS, the algorithm was executed from scratch for the initial solution and at each robot failure. Runs were conducted for all four combinations of $m \in \{2, 3\}$ and $\beta \in \{30\%, 70\%\}$.



(a) The Star environment. (b) The Waffle environment. (c) The H environment.

Figure 5.5: The environments used in our simulations.

Each trial was limited to at most 10 minutes of run time, including both planning time and (simulated) execution time. If, after that time, the robots had not yet successfully cleared all shadows, the simulation would have been considered a failure. In the results presented here, none of the trials failed.

For each combination of environment, algorithm, team size n, number of failures m, and failure time β, we conducted 25 trials. The success or failure of the run

and total computation time spent planning and replanning were recorded. Planning time is summarized by the mean (μ) and the standard deviation (σ) over all trials. Tables 5.1 and 5.2 report the results, from which a few conclusions may be drawn.

Replanning is beneficial: Recall from Section 5.3.1 that junction sampling was developed to "recover" information in the event of a pursuer failure. The improvements for the proposed algorithm compared to RCS can be attributed to efficiencies gained by re-planning rather than starting from scratch. In each type of simulation, the proposed algorithm was roughly the same or marginally better over RCS.

Later failures are easier to recover: For the trials with $\beta = 70\%$, the total planning time was less than when $\beta = 30\%$. This is likely due to the fact that allowing more time to traverse the solution path will, in many cases, provide the next planning stage with an improved shadow label (i.e. more cleared shadows), reducing the difficulty of the replanning problem. Additionally, the gap between Chp and RCS was greater as β increased. Here, Chp was likely able to quickly recover the lost information (which, in the event of a later failure, is less information).

Impacts of the number of failures Increasing from $m = 2$ to $m = 3$ increased the planning time for both algorithms. We speculate that this can be attributed to the additional pursuer failure for which both the proposed algorithm and RCS are required to recompute strategies.

5.5 Conclusion

We presented a method of deconstructing higher dimensional solutions in order to alleviate the issue of potential robotic failures in a visibility-based pursuit-evasion problem. We did this by building a new sampling strategy which allowed us to utilize

Table 5.1: Simulation results (formative = 5, broken count = 2).

	success rate	planning time (s) μ	σ
Figure 5.5(a)			
Chp ($\beta = 30\%$)	100%	9.03	5.80
RCS ($\beta = 30\%$)	100%	10.00	3.80
Chp ($\beta = 70\%$)	100%	5.14	3.21
RCS ($\beta = 70\%$)	100%	8.65	3.81
Figure 5.5(b)			
Chp ($\beta = 30\%$)	100%	82.53	72.75
RCS ($\beta = 30\%$)	100%	77.14	48.51
Chp ($\beta = 70\%$)	100%	41.96	40.70
RCS ($\beta = 70\%$)	100%	71.89	48.55
Figure 5.5(c)			
Chp ($\beta = 30\%$)	100%	6.53	5.38
RCS ($\beta = 30\%$)	100%	5.92	2.81
Chp ($\beta = 70\%$)	100%	4.87	4.74
RCS ($\beta = 70\%$)	100%	9.42	20.24

previously computed information. Our algorithm was able to out-perform existing algorithms in the context of our problem.

Table 5.2: Simulation results (formative = 5, broken count = 3).

	success rate	planning time (s) μ	σ
Figure 5.5(a)			
Chp ($\beta = 30\%$)	100%	10.78	7.13
RCS ($\beta = 30\%$)	100%	12.88	3.97
Chp ($\beta = 70\%$)	100%	5.69	3.29
RCS ($\beta = 70\%$)	100%	10.65	4.87
Figure 5.5(b)			
Chp ($\beta = 30\%$)	100%	162.16	117.25
RCS ($\beta = 30\%$)	100%	171.89	109.21
Chp ($\beta = 70\%$)	100%	86.39	101.17
RCS ($\beta = 70\%$)	100%	127.89	90.55
Figure 5.5(c)			
Chp ($\beta = 30\%$)	100%	8.61	6.33
RCS ($\beta = 30\%$)	100%	7.62	3.13
Chp ($\beta = 70\%$)	100%	5.35	5.40
RCS ($\beta = 70\%$)	100%	11.94	20.26

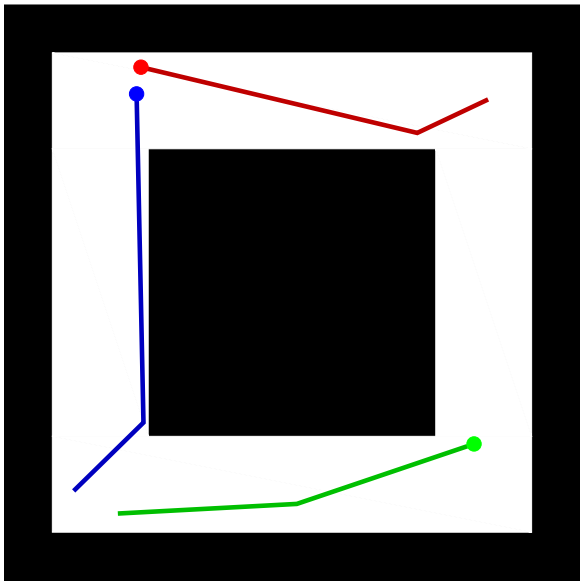
Chapter 6

Robust-by-Design Plans for Multi-Robot Pursuit-Evasion

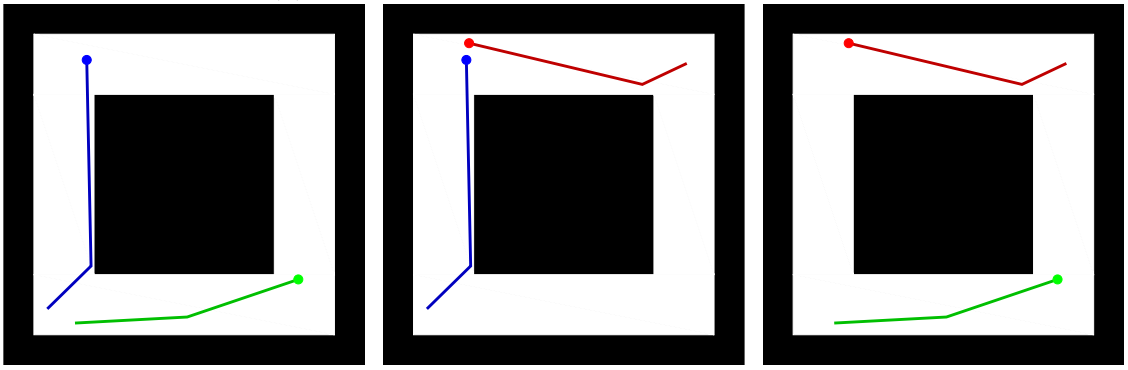
6.1 Introduction

This chapter investigates how one might generate strategies for the pursuers that are robust in the face of malfunctions which may inhibit members of the team, as shown in Figure 6.1. A few such defects include utter failure, intermittent sensor failure (i.e. false negatives), and incorrectly transmitted information. Chapter 5 considered the situation where members of the pursuer team suffered complete and catastrophic failure, necessitating that the pursuer strategy be re-planned online. This replanning step introduces execution delays and requires all of the remaining pursuers to be aware of failures when and where they occur. This chapter addresses a larger range of potential robotic defects by generating joint motion strategies for the pursuers which guarantee detection of the evaders, regardless of any single pursuer malfunction and without the need to detect such malfunctions when they occur nor to halt the current execution to update the pursuers' motion strategies.

Beyond the previously-studied pursuit-evasion component, which must reason about the regions of the environment which may contain an evader, the major challenge in generating such 'failure-robust' joint pursuer strategies is the need to track these 'contaminated' regions separately for each possible pursuer failure, while ensuring that a single solution can be extracted when the process is complete. Note that the best known complete algorithm for multi-robot visibility-based pursuit-evasion



(a) A 1-failure robust solution with 3 pursuers.



(b) Red failure (c) Green failure (d) Blue failure

Figure 6.1: A simple example of a 1-failure robust solution for $n = 3$ robots. Removal of any single pursuer does not preclude the remaining pursuers from capturing an evader in the $n = 2$ subsolution.

solves the problem in time doubly exponential in the number of pursuer agents. To address this shortcoming, sampling-based methods have been developed that utilize a graph structure whose vertices encode the pursuers pose information in addition to the regions of the environment where the evader may be; whereas edges indicate feasible pursuer motions within the environment.

This chapter builds on this line of sampling-based approaches by providing an augmented graph structure that encodes the "robustness" of the search (i.e. reasoning over any possible pursuer failure). In response to this added layer of complexity,

a sampling method (RCS) is utilized to create condensed sample sets that ensure capture of the evader via a highly connected roadmap.

The remainder of the chapter begins with a formal description of the problem, which can be found in Section 6.2. Section 6.3 outlines the algorithm utilized to generate robust joint motion strategies. Simulation results, in which the algorithm was employed in several different representative environments, are described in Section 6.4 before the chapter concludes with a summary in Section 6.5.

6.2 k-failure Robust Solutions

Recall that it is the pursuers' objective is to guarantee to locate the evader. We are interested in solutions that can still guarantee the detection of the evader, even if a single pursuer robot fails.

Definition Given a set of robots, R, where $|R| > k$, a solution X is *k-failure robust* if it remains a solution utilizing only the robots $R \setminus B$, for any $B \subset R$ with $|B| \leq k$.

Notice that, according to these definitions, a solution is a 0-failure robust solution. In this chapter, we address the problem of generating 1-failure robust solutions.

6.3 Algorithm Overview

This section describes an algorithm to generate a 1-failure robust solution for a given environment and number of pursuers.

The algorithm builds upon earlier sampling-based approaches to visibility-based pursuit-evasion Chapter 3. The basic idea in that prior work is to construct a roadmap data structure that represents the pursuers' ability to move through the environment and to clear various collections of shadows. As JPCs are sampled and inserted into the roadmap, the prior algorithm tracks a set of reachable shadow labels attached to each vertex. When the data structure determines that an all-clear shadow label

is reachable at some vertex, the algorithm extracts that solution and terminates successfully.

The algorithm we propose here differs from that baseline in an important way, necessitated by the need to produce robust solutions. The data structure is augmented to track shadow labels not for all n robots, but instead for each of the n distinct subsets of size $n-1$. This ensures that, if *all* of these shadow labels achieve an all clear status, the resulting solution will be 1-failure robust. See Section 6.3.1.

6.3.1 Robust Sample-Generated Pursuit-Evasion Graphs (rSG-PEG)

Here, we describe how we enhanced the existing sample-generated pursuit-evasion graph (SG-PEG) data structure [50] to generate 1-failure robust solutions. An rSG-PEG, G, is a directed graph in which one vertex is designated as the root. Each vertex of G is labeled with a JPC; a directed edge $u \to w$ indicates that there exists a coordinate-wise straight line, collision free movement between the JPCs of u and w. For the sake of compactness, a vertex with JPC $w_1, \ldots, w_n$ will be named w.

Each vertex is also associated with a collection of *failure shadow labels*, each of which is an n-tuple $L = (\ell_1, \ldots, \ell_n)$, where each ℓ_i is a single shadow label. The interpretation is that the existence of failure shadow label $(\ell_1, \ldots, \ell_n)$ at vertex w implies that there exists a path in G from the root to w, such that for each $1 \leq i \leq n$, the pursuers in $\{1, \ldots n\} \setminus \{i\}$ would reach shadow label ℓ_i by following that path. Thus, reaching a vertex with a failure shadow label in which all n sub-labels are all cleared results in a 1-failure robust solution within G.

An rSG-PEG is constructed by repeated calls to its primary operation, G.AddSample(w), which performs the following steps.

1. A vertex w is added to G.

2. An edge $w \to u$ is added between between w and each other vertex u of G if the line segment $\overline{wu}$ is fully contained in F^n. Similarly the twin edge $u \to w$ is added to G.

3. During this process, each time an edge $a \to b$ is created, for each failure shadow label L attached to a, the algorithm computes a failure shadow label L' for b, computed by propagating each shadow label ℓ_i in L across the edge $a \to b$, as described in Section 2.2.4. If L' is not dominated by any other failure shadow label at b, it is retained at b. Here, dominance of failure shadow labels is defined by generalizing the idea of dominance of shadow labels. This process continues recursively, spreading new reachable failure shadow labels across G as needed.

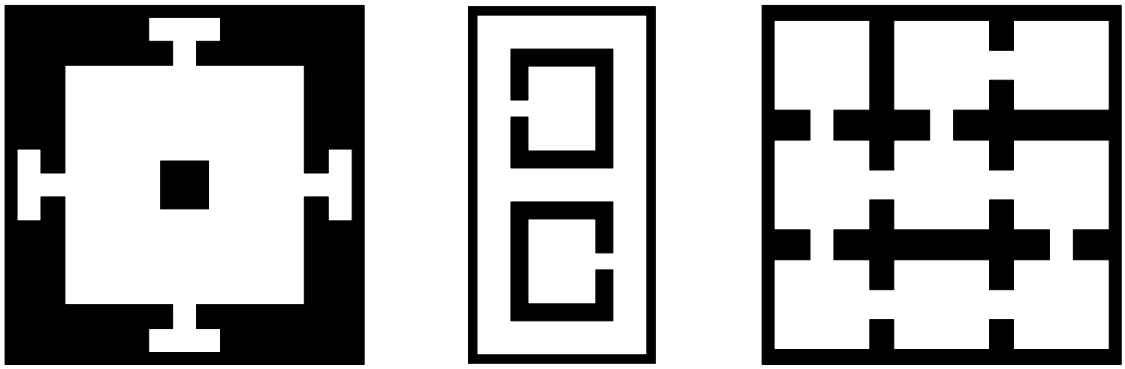
Adding a sequence of samples to an rSG-PEG is enough to form a 1-failure robust solution. We present a summary of our 1-failure robust solution simulations in the following section.

6.4 Evaluation

We implemented the algorithms described in Section 6.3 in C++. All of the experiments below were conducted on a single core of a 6 core Intel i5-9600K CPU running 64-bit Ubuntu 20.04 at 3.7 GHz with 16 GB of RAM. Figure 6.3 shows an example of a 1-failure robust solution computed by this implementation, using RCS.

To evaluate the algorithm quantitatively, we compared (a) the caching based shadow label updating system described in Section 2.2.4 against the original naive approach, and (b) the RCS technique to web sampling (WS), both of which are described in Chapter 3.

For (a) we executed the main algorithm, using RCS, to find a 1-failure robust solution utilizing 3 pursuers in the environment shown in Figure 6.2(a). We conducted 25 trials using shadow influence caching and 25 trials using the naive method that



(a) The Star environment. (b) The Brick environment. (c) The Office environment.

Figure 6.2: The environments we conducted our simulations on.

computes shadow influence anew each time. All 50 trials successfully found 1-failure robust solutions. The average computation time (in seconds) utilizing shadow influence caching was 47.02 with a standard deviation of 15.12; without shadow influence caching, the average was 425.88 seconds, with a standard deviation of 512.64. The results demonstrate an overwhelming benefit to the use of shadow influence caching.

For (b), we compared the efficiency of planning with RCS in contrast with planning with WS. To do so, we attempted to find a 1-failure robust solution utilizing $n = 3$, 4, and 5 pursuers across the 3 environments found in Figure 6.2. For each such scenario, we conducted 50 simulations. Tables 6.1, 6.2 and 6.3 encapsulate the results, showing the mean (μ) and standard deviation (σ) of the planning time (in seconds) as well as number of vertices and edges in the rSG-PEG. Each such simulation was allotted a timeout of 10 minutes; if the corresponding simulation failed to produce a 1-failure robust solution in that time, it was deemed a failure. The results for 3 robots (Table 6.1) show that RCS is, at worst, a marginal improvement over WS in all experiments. The addition of another robot further separated the results of these sampling methods. In particular, the standard deviation of the planning time was significantly decreased, resulting in more consistent run times. The highly connective nature of RCS allows us to increase the number of robots without severely suffering from the curse of dimensionality.

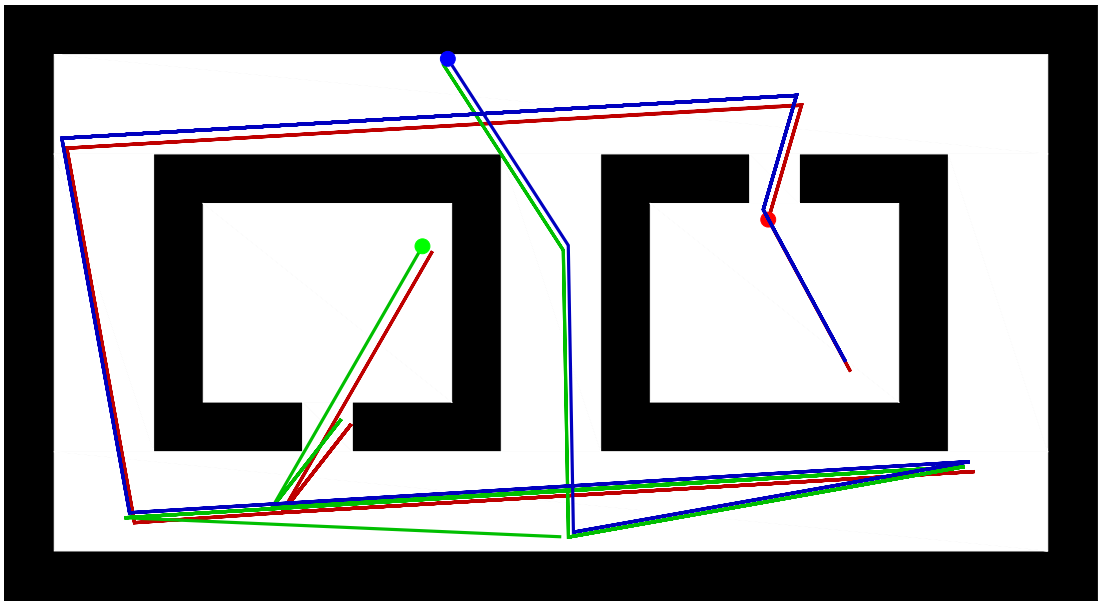


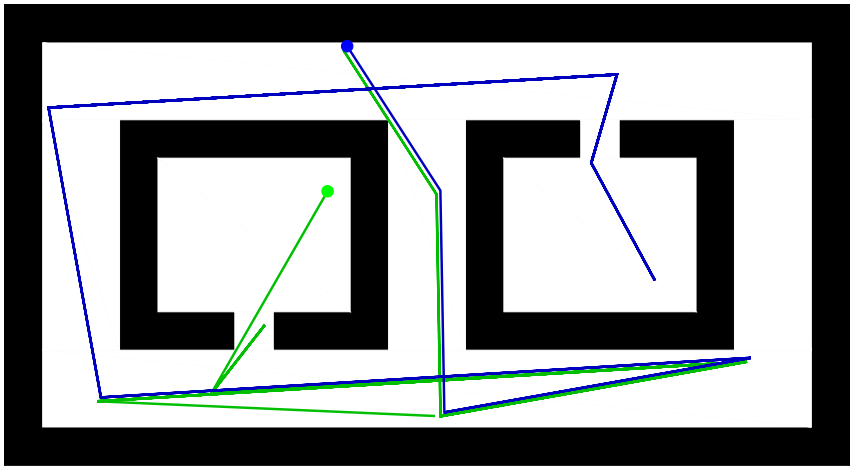
Figure 6.3: A 1-failure robust solution. Notice each unique region of the environment is examined by at least 2 pursuers. The paths taken by the pursuers were slightly shifted for illustrative purposes.

Table 6.1: Simulation results using 3 pursuers.

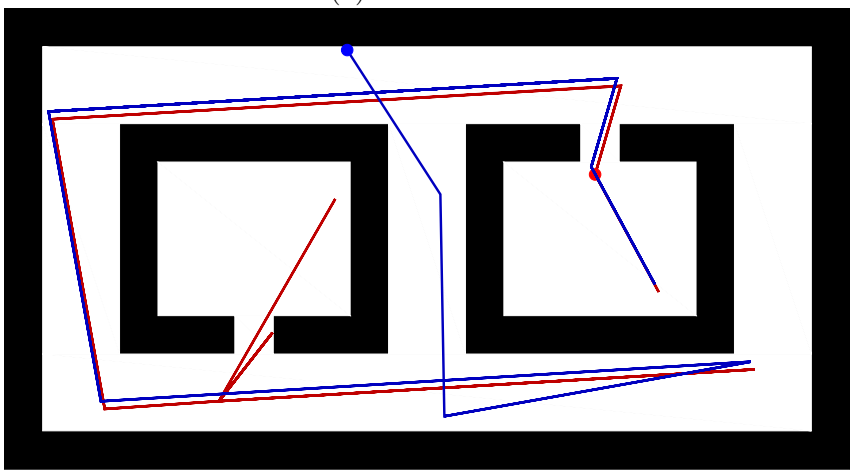
	success rate	planning time (s)		Vertices		Edges	
		μ	σ	μ	σ	μ	σ
Figure 6.2(a)							
RCS	100%	50.85	20.10	23.36	7.19	30.62	11.62
WS	100%	53.01	20.22	26.56	8.14	34.44	15.04
Figure 6.2(b)							
RCS	100%	89.84	47.37	47.30	18.09	72.32	39.65
WS	100%	109.00	57.44	50.02	18.24	77.70	41.80
Figure 6.2(c)							
RCS	100%	131.86	64.67	50.86	19.77	63.58	29.68
WS	98%	151.10	50.98	59.76	16.75	73.92	26.58

6.5 Conclusion

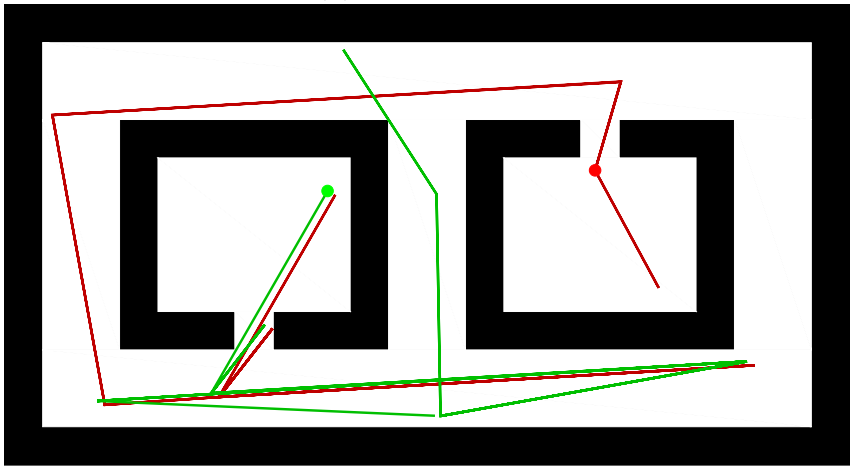
This chapter addressed the issue of potential single robot failures in the multi-robot visibility-based pursuit-evasion problem. We introduced a modification to an existing data structure to ensure that each solution generated by our algorithm remains a solution in the event of any single robotic failure.



(a) Red failure.



(b) Green failure.



(c) Blue failure.

Figure 6.4: The three subsolutions of Figure 6.3.

Table 6.2: Simulation results using 4 pursuers.

	success rate	planning time (s)		Vertices		Edges	
		μ	σ	μ	σ	μ	σ
Figure 6.2(a)							
RCS	100%	58.21	21.82	23.26	24.01	28.34	40.63
WS	100%	75.40	30.11	26.56	9.21	28.54	10.95
Figure 6.2(b)							
RCS	100%	82.79	44.73	38.34	26.62	46.10	38.11
WS	100%	113.08	57.75	46.34	25.11	52.60	33.74
Figure 6.2(c)							
RCS	100%	140.57	60.01	64.72	64.57	74.34	87.02
WS	100%	194.53	74.96	87.14	90.13	104.10	133.88

Table 6.3: Simulation results using 5 pursuers.

	success rate	planning time (s)		Vertices		Edges	
		μ	σ	μ	σ	μ	σ
Figure 6.2(a)							
RCS	100%	84.93	46.79	88.48	238.54	258.76	1102.28
WS	100%	96.86	42.19	57.34	87.28	80.12	162.37
Figure 6.2(b)							
RCS	100%	86.71	47.72	65.56	95.32	90.30	183.12
WS	100%	129.01	70.41	78.50	95.09	93.82	149.29
Figure 6.2(c)							
RCS	100%	182.10	69.91	251.26	305.28	304.30	425.75
WS	98%	255.51	128.64	237.27	605.98	367.49	1233.18

Chapter 7

Conclusion

To review, the content of this dissertation considers numerous variations of the pursuit-evasion problem. These problems are useful to consider and explore due to their utility in field robotics, including but not limited to: environmental monitoring, surveillance and search-and-rescue. In particular, the core pursuit-evasion problem addressed in this dissertation is a variation where a team of several pursuers work in unison to establish visibility with any number of evaders hiding within an environment. The remainder of this chapter presents concise summaries of my novel contributions to field of multi-robot visibility-based pursuit-evasion, as well as possible directions for future work.

Chapter 3 presented two novel sampling techniques. Both of these sampling methods relied on random sets of points from the environment called webs. By construction, webs guarantee that each region of the environment is visible to at least one point in the set. Additionally, webs are connected, that is, it is possible to travel from any web point to another using straight-line collision-free movements between web points. The first sampling method, web sampling (Section 3.1), generated a web for each pursuer and sequentially drew points from each web. The second sampling method, robust cycle sampling (Section 3.2), preformed depth first search on a web to generate a visiting order. Pursuers were placed evenly along this list, and samples were generated by exhausting the list. This ensured that samples were highly connected. Both of these methods were significant improvements over existing sampling

techniques, and proved to be essential in solving certain variations of the pursuit-evasion problem.

Next, Chapter 4 provided a way to progressively add additional pursuers to the search, as well as a technique for refining solutions. Being able to add additional pursuers to the search allowed us to remove of the number of pursuers as an input. All known existing algorithms required the number of pursuers as an input, which could result in situations where it is impossible to find a solution. This input was removed by defining expansion conditions that, when met, added another pursuer to the search. A cloning method was discussed that allowed an additional pursuer to be added without needing to recalculate any information within the graph. A greedy solution refinement technique helped us significantly improve the solution path by reducing its length.

An online failure recover method was presented in Chapter 5. Here, we introduced a way of deconstructing our data structure in such a way that allowed the removal of a single pursuer without significant information loss. This deconstruction method allowed us to recover, in real time, from catastrophic robotic failures. A sampling method was introduced which allowed the remaining functioning pursuers to quickly recapture the information that the failing robot possessed. This recovery method was a significant improvement over the naive approach of replanning from scratch each time a failure occurred.

Lastly, Chapter 6 explored how one may generate solutions that remain solutions despite any single pursuer malfunction. These solutions were generated offline, prior to execution, unlike the previously mentioned failure recovery method. Additionally, this method allowed for a wider array of failures, rather than strictly catastrophic failures. Some such failures include: limited sensing range, incorrectly transmitted information and turning radius limitations.

The first direction of future work should extend the results of Chapter 6 to allow more than a single pursuer malfunction. This would likely require a new data structure capable of effectively maintaining all reachable shadow labels given certain failures. It is possible that this information could be concisely stored by considering a lattice of subsets, where the top level is the set of all pursuers and each subset represents a collection of malfunctioning robots. While Chapter 6 considers a wide range of malfunctions, it may be of interest to study some of these malfunctions as separate problems. In particular, crafting new sampling methods tailored to the each specific malfunction would likely result in significantly faster and fewer calculations. Another possible topic of interest would examine how the environment affects the difficulty of these problems. Section 4.5 briefly explores this concept, but a reasonable amount of analysis remains to be done.

Bibliography

[1] T. V. Abramovskaya and N. N. Petrov, "The theory of guaranteed search on graphs", *Vestnik St. Petersburg University*, vol. 46, no. 2, pp. 49–75, 2013.

[2] J. J. Acevedo, B. C. Arrue, I. Maza, and A. Ollero, "A decentralized algorithm for area surveillance missions using a team of aerial robots with different sensing capabilities", in *Proc. IEEE International Conference on Robotics and Automation*, 2014.

[3] T. Alam, M. M. Rahman, L. Bobadilla, and B. Rapp, "Multi-vehicle patrolling with limited visibility and communication constraints", in *Military Communications Conference*, 2017, pp. 465–479.

[4] B. Alspach, "Searching and sweeping graphs: A brief survey", *Matematiche*, vol. 59, pp. 5–37, 2004.

[5] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[6] D. Bhadauria, K. Klein, V. Isler, and S. Suri, "Capturing an evader in polygonal environments with obstacles: The full visibility case", *International Journal of Robotics Research*, vol. 31, pp. 1176–1189, 10 2012.

[7] R. Borie, S. Koenig, and C. Tovey, "Pursuit-evasion problems", in *Handbook of Graph Theory*, J. Gross, J. Yellen, and P. Zhang, Eds., Chapman and Hall, 2013, ch. 9.5, pp. 1145–1165.

[8] J. Chen, W. Zha, Z. Peng, and D. Gu, "Multi-player pursuit–evasion games with one superior evader", *Automatica*, vol. 71, pp. 24–32, 2016.

[9] P. Dames, P. Tokekar, and V. Kumar, "Detecting, localizing, and tracking an unknown number of moving targets using a team of mobile robots", *International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1540–1553, 2017.

[10] M. Duarte, J. Gomes, V. Costa, T. Rodrigues, F. Silva, V. Lobo, M. M. Marques, S. M. Oliveira, and A. L. Christensen, "Application of swarm robotics systems to marine environmental monitoring", in *MTS/IEEE OCEANS - Shanghai*, 2016.

[11] M. Dunbabin and L. Marques, "Robots for environmental monitoring: Significant advancements and applications", *IEEE Robotics and Automation Magazine*, vol. 19, pp. 24–39, 2012.

[12] J. W. Durham, A. Franchi, and F. Bullo, "Distributed pursuit-evasion without mapping or global localization via local frontiers", *Autonomous Robots*, vol. 32, pp. 81–95, 2012.

[13] S. W. Feng, S. D. Han, K. Gao, and J. Yu, "Efficient algorithms for optimal perimeter guarding", in *Proc. Robotics: Science and Systems*, 2019.

[14] B. P. Gerkey, S. Thrun, and G. Gordon, "Visibility-based pursuit-evasion with limited field of view.", *International Journal of Robotics Research*, vol. 25, no. 4, pp. 299–315, 2006.

[15] P. Golovach, "A topological invariant in pursuit problems", *Differentsial'nye Uraveniya (Differential Equations)*, vol. 25, pp. 923–929, 1989.

[16] L. Gregorin, S. Givigi, E. Freire, E. Carvalho, and L. Molina, "Heuristics for the multi-robot worst-case pursuit-evasion problem", *IEEE Access*, vol. 5, pp. 17 552–17 566, Aug. 2017.

[17] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani, "Visibility-based pursuit-evasion in a polygonal environment", *International Journal on Computational Geometry and Applications*, vol. 9, no. 5, pp. 471–494, 1999.

[18] K. Hauser, "The minimum constraint removal problem with three robotics applications", *International Journal of Robotics Research*, vol. 33, no. 1, pp. 5–17, 2014.

[19] Y. C. Ho, A. Bryson, and S. Baron, "Differential games and optimal pursuit-evasion strategies", *IEEE Transactions on Automatic Control*, vol. 10, no. 4, pp. 385–389, Oct. 1965.

[20] G. Hollinger, A. Kehagias, and S. Singh, "Probabilistic strategies for pursuit in cluttered environments with multiple robots", in *Proc. IEEE International Conference on Robotics and Automation*, 2007.

[21] R. Isaacs, *Differential Games*. New York: Wiley, 1965.

[22] V. Isler, N. Noori, P. Plonski, A. Renzaglia, P. Tokekar, and J. V. Hook, "Finding and tracking targets in the wild: Algorithms and field deployments", in *Proc. IEEE International Symposium on Safety, Security, and Rescue Robotics*, 2015.

[23] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning", *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, Jun. 2011.

[24] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Jun. 1996.

[25] A. Kolling and S. Carpin, "Multi-robot pursuit-evasion without maps", in *Proc. IEEE International Conference on Robotics and Automation*, 2010, pp. 3045–3051.

[26] A. Kolling and S. Carpin, "Pursuit-evasion on trees by robot teams", *IEEE Transactions on Robotics*, vol. 26, pp. 32–47, Mar. 2010.

[27] A. Krontiris and K. E. Bekris, "Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner", in *Proc. IEEE International Conference on Robotics and Automation*, Sweden, 2016.

[28] S. S. Kumkov, S. L. Ménec, and V. S. Patsko, "Zero-sum pursuit-evasion differential games with many objects: Survey of publications", *Dynamic Games and Applications*, vol. 7, pp. 609–633, 2017.

[29] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects", in *Proc. Workshop on the Algorithmic Foundations of Robotics*, 2000.

[30] J. Marble and K. E. Bekris, "Asymptotically near-optimal planning with probabilistic roadmap spanners", *IEEE Transactions on Robotics*, vol. 29, no. 2, pp. 432–444, 2013.

[31] N. Mimmo, P. Bernard, and L. Marconi, "Avalanche victim search via robust observers", in *Proc. IEEE International Conference on Robotics and Automation*, pp. 4066–4072.

[32] A. V. Moll, D. Casbeer, E. Garcia, and D. Milutinovic, "Pursuit-evasion of an evader by multiple pursuers", in *Proc. International Conference on Unmanned Aircraft Systems*, Jun. 2018, pp. 133–142.

[33] T. Olsen, N. M. Stiffler, and J. M. O'Kane, "Rapid recovery from robot failures in multi-robot visibility-based pursuit-evasion", in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021.

[34] T. Olsen, N. M. Stiffler, and J. M. O'Kane, *Robust-by-design plans for multi-robot pursuit-evasion*, 2021.

[35] T. Olsen, A. M. Tumlin, N. M. Stiffler, and J. M. O'Kane, "A visibility roadmap sampling approach for a multi-robot visibility-based pursuit-evasion problem", in *Proc. IEEE International Conference on Robotics and Automation*, 2021.

[36] S. Park, J. Lee, and K. Chwa, "Visibility-based pursuit-evasion in a polygonal region by a searcher", in *Proc. International Colloquium on Automata, Languages and Programming*, Springer-Verlag, 2001, pp. 281–290.

[37] T. D. Parsons, "Pursuit-evasion in a graph", in *Theory and Application of Graphs*, Y. Alavi and D. R. Lick, Eds., Berlin: Springer-Verlag, 1976, pp. 426–441.

[38] N. N. Petrov, "The cossack-robber differential game", *Differentsial'nye Uraveniya (Differential Equations)*, vol. 19, pp. 1366–1374, 1983.

[39] A. Quattrini Li, F. Amigoni, R. Fioratto, and V. Isler, "A search-based approach to solve pursuit-evasion games with limited visibility in polygonal environments", in *Proc. International Conference on Autonomous Agents and Multiagent Systems*, 2018, pp. 1693–1701.

[40] S. Rajko and S. M. LaValle, "A pursuit-evasion bug algorithm", in *Proc. IEEE International Conference on Robotics and Automation*, 2001, pp. 1954–1960.

[41] U. Ruiz and R. Murrieta-Cid, "Time-optimal motion strategies for capturing an omnidirectional evader using a differential drive robot", *IEEE Transactions on Robotics*, vol. 21, no. 3, Jun. 2013.

[42] S. Sachs, S. M. LaValle, and S. Rajko, "Visibility-based pursuit-evasion in an unknown planar environment", *International Journal of Robotics Research*, vol. 23, no. 1, pp. 3–26, 2004.

[43] F. Shkurti and G. Dudek, "On the complexity of searching for an evader with a faster pursuer", in *Proc. IEEE International Conference on Robotics and Automation*, 2013, pp. 4062–4067.

[44] F. Shkurti and G. Dudek, "Topologically distinct trajectory predictions for probabilistic pursuit", in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017, pp. 5653–5660.

[45] F. Shkurti, N. Kakodkar, and G. Dudek, "Model-based probabilistic pursuit via inverse reinforcement learning", in *Proc. IEEE International Conference on Robotics and Automation*, 2018, pp. 7804–7811.

[46] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps", *International Journal of Robotics Research*, no. 23, 2004.

[47] K. Solovey, O. Salzman, and D. Halperin, "Finding a needle in an exponential haystack: Discrete rrt for exploration of implicit roadmaps in multi-robot motion planning", in *Proc. Workshop on the Algorithmic Foundations of Robotics*, 2014.

[48] N. M. Stiffler and J. M. O'Kane, "Visibility-based pursuit-evasion with probabilistic evader models", in *Proc. IEEE International Conference on Robotics and Automation*, 2011, pp. 4254–4259.

[49] N. M. Stiffler and J. M. O'Kane, "A complete algorithm for visibility-based pursuit-evasion with multiple pursuers", in *Proc. IEEE International Conference on Robotics and Automation*, 2014, pp. 1660–1667.

[50] N. M. Stiffler and J. M. O'Kane, "A sampling based algorithm for multi-robot visibility-based pursuit-evasion", in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 1782–1789.

[51] N. M. Stiffler and J. M. O'Kane, "Pursuit-evasion with fixed beams", in *Proc. IEEE International Conference on Robotics and Automation*, 2016, pp. 4251–4258.

[52] N. M. Stiffler and J. M. O'Kane, "Complete and optimal visibility-based pursuit-evasion", *International Journal of Robotics Research*, vol. 36, pp. 923–946, 88 Jul. 2017.

[53] N. M. Stiffler and J. M. O'Kane, "Planning for robust visibility-based pursuit-evasion", in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020.

[54] I. Suzuki and M. Yamashita, "Searching for a mobile intruder in a polygonal region", *SIAM Journal on Computing*, vol. 21, no. 5, pp. 863–888, Oct. 1992.

[55] K. Tiwari and N. Y. Chong, *Multi-robot Exploration for Environmental Monitoring: The Resource Constrained Perspective*. Academic Press, 2019.

[56] P. Tokekar, D. Bhadauria, A. Studenski, and V. Isler, "A robotic system for monitoring carp in Minnesota lakes", *Journal of Field Robotics*, vol. 27, no. 3, pp. 681–685, 2010.

[57] B. Tovar and S. M. LaValle, "Visibility-based pursuit-evasion with bounded speed", *International Journal of Robotics Research*, vol. 27, pp. 1350–1360, 12 2008.

[58] J. Vander Hook and V. Isler, "Pursuit and evasion with uncertain bearing measurements", in *Proc. Candadian Conference on Computational Geometry*, 2014.

[59] G. Wagner, M. Kang, and H. Choset, "Probabilistic path planning for multiple robots with subdimensional expansion", in *Proc. IEEE International Conference on Robotics and Automation*, 2012.

[60] Y. Wang, L. Dong, and C. Sun, "Cooperative control for multi-player pursuit-evasion games with reinforcement learning", *Neurocomputing*, vol. 412, pp. 101–114, 2020.

[61] M. Xanthidis, J. M. Esposito, I. Rekleitis, and J. M. O'Kane, "Motion planning by sampling in subspaces of progressively increasing dimension", *Journal of Intelligent and Robotic Systems*, 2020.

[62] Z. Zhou, W. Zhang, J. Ding, H. Huang, D. M. Stipanović, and C. J. Tomlin, "Cooperative pursuit with voronoi partitions", *Automatica*, vol. 72, pp. 64–72, 2016.

[63] R. Zou and S. Bhattacharya, "On optimal pursuit trajectories for visibility-based target tracking game", *IEEE Transactions on Robotics*, vol. 35, pp. 449–465, 2 Apr. 2019.

Appendix A

Randomly Generated Environments

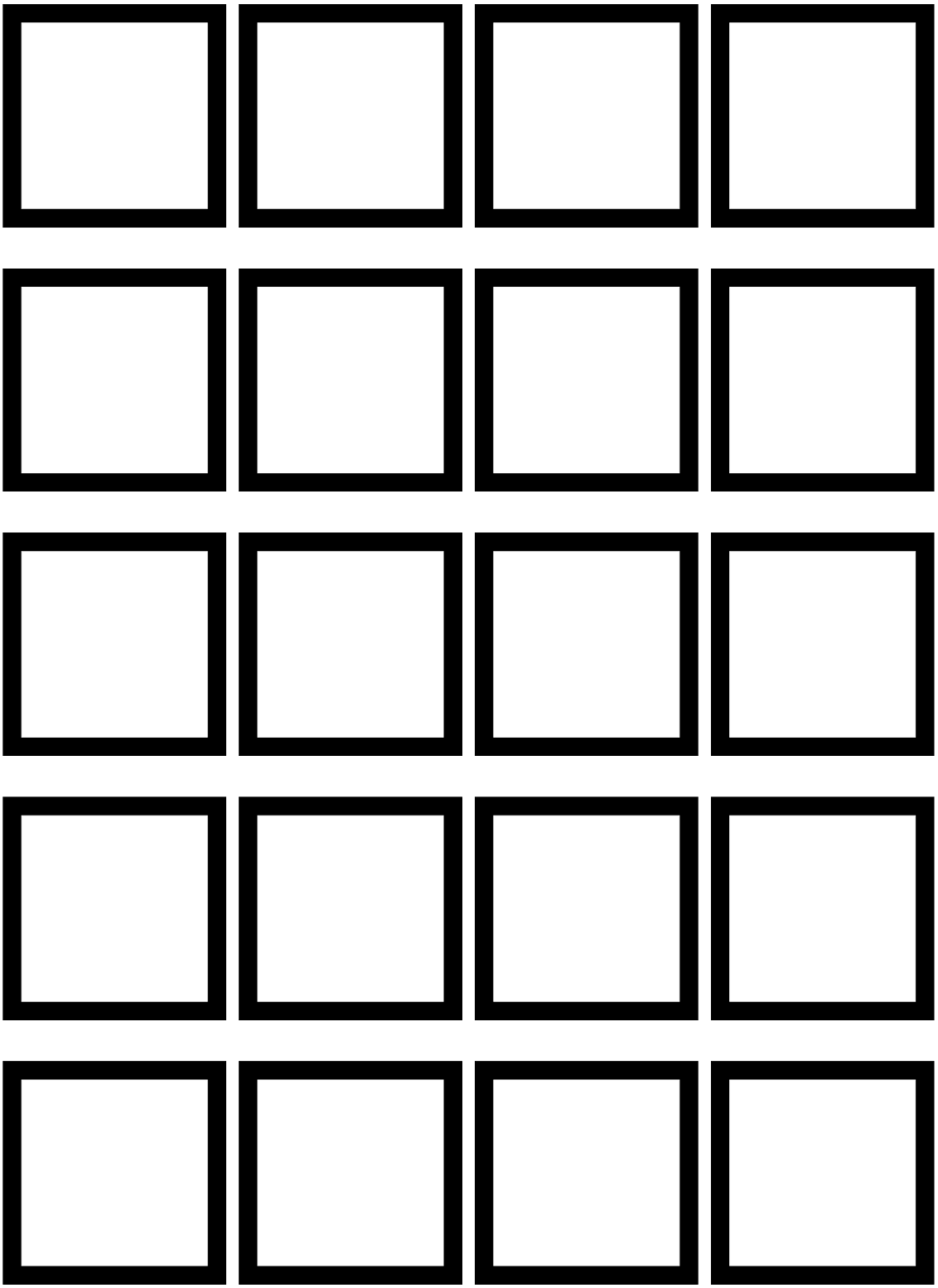


Figure A.1: Randomly generated environments containing 0 blocks and 0 rooms.

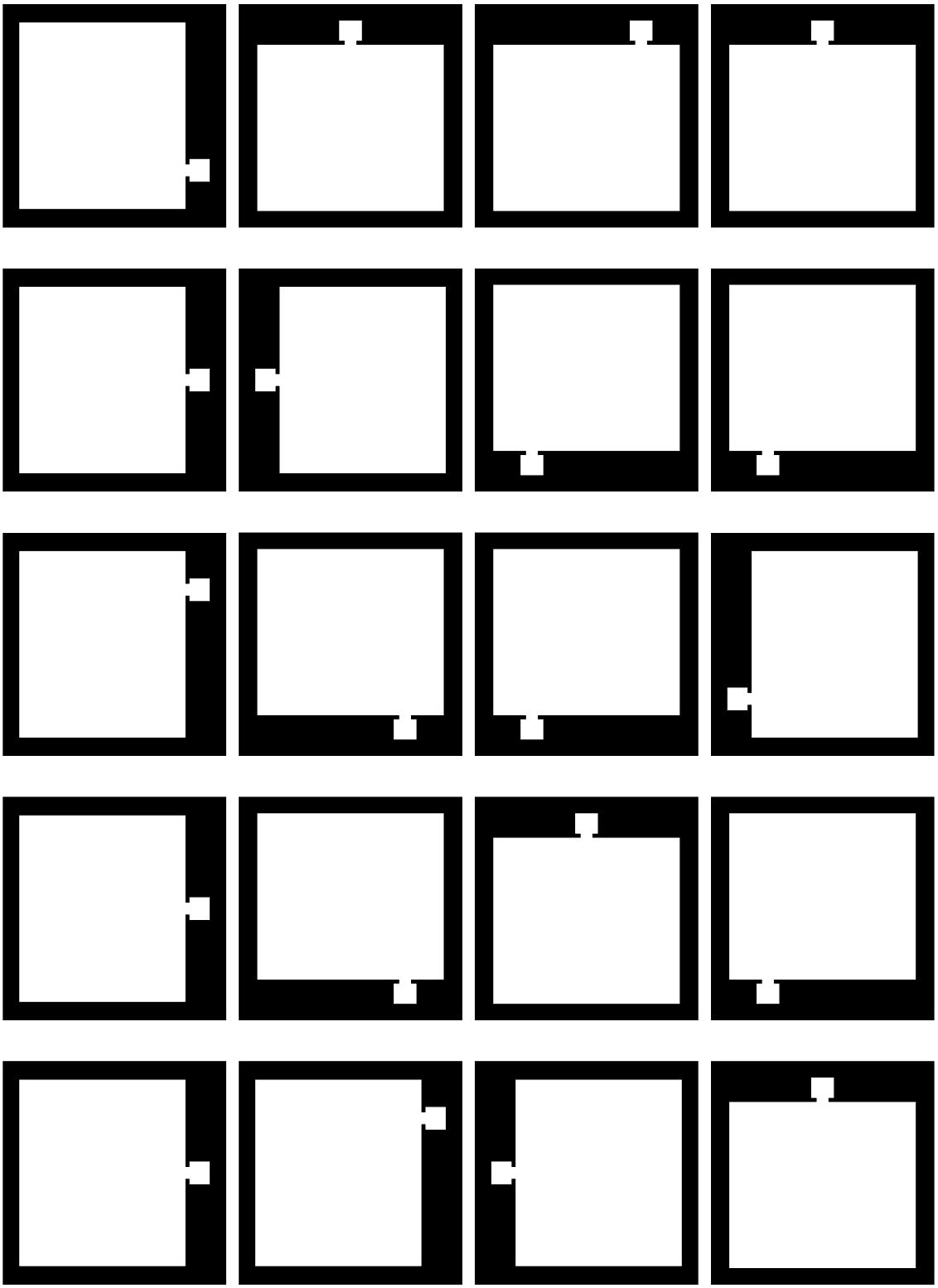


Figure A.2: Randomly generated environments containing 0 blocks and 1 room.

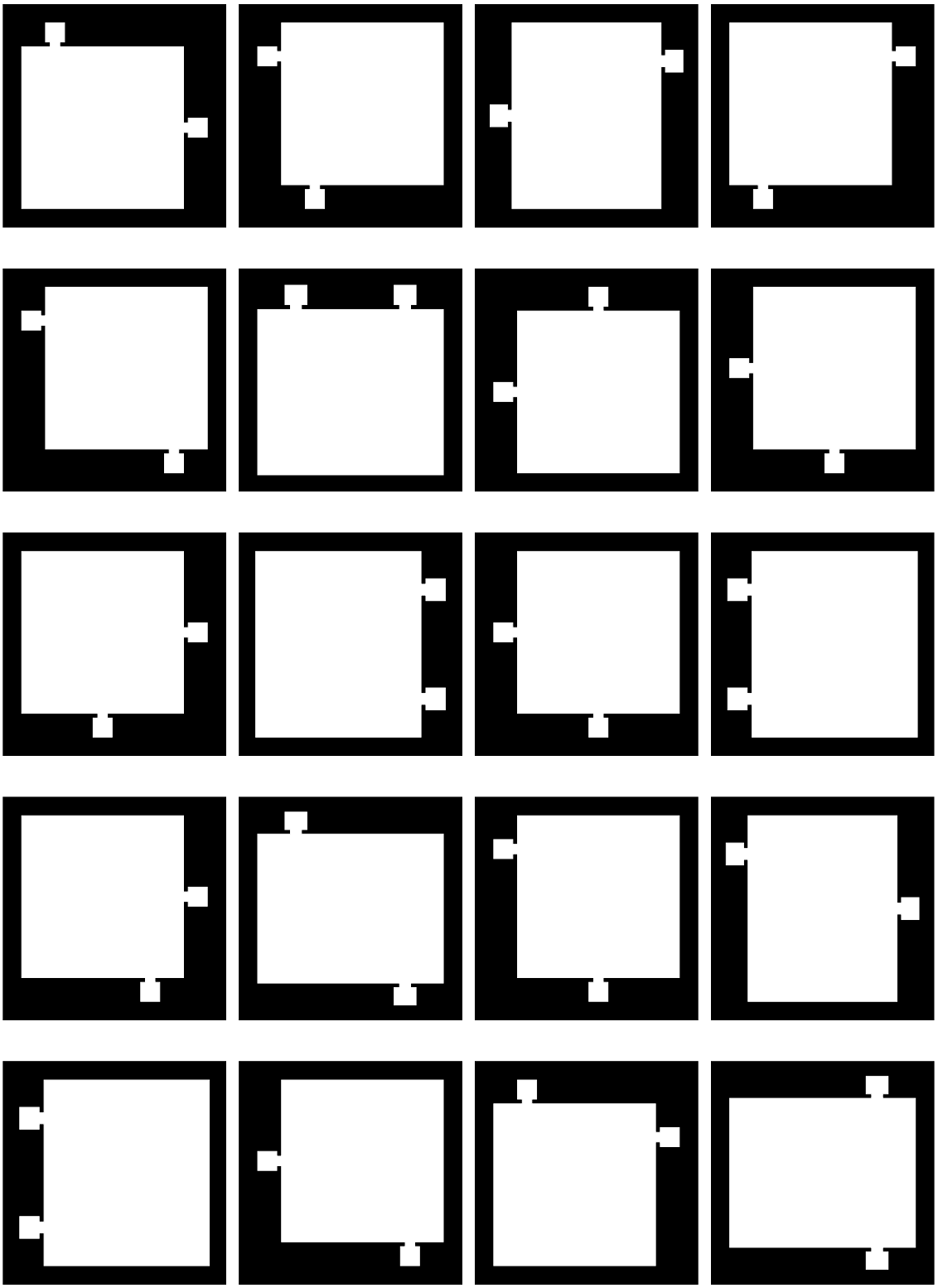


Figure A.3: Randomly generated environments containing 0 blocks and 2 rooms.

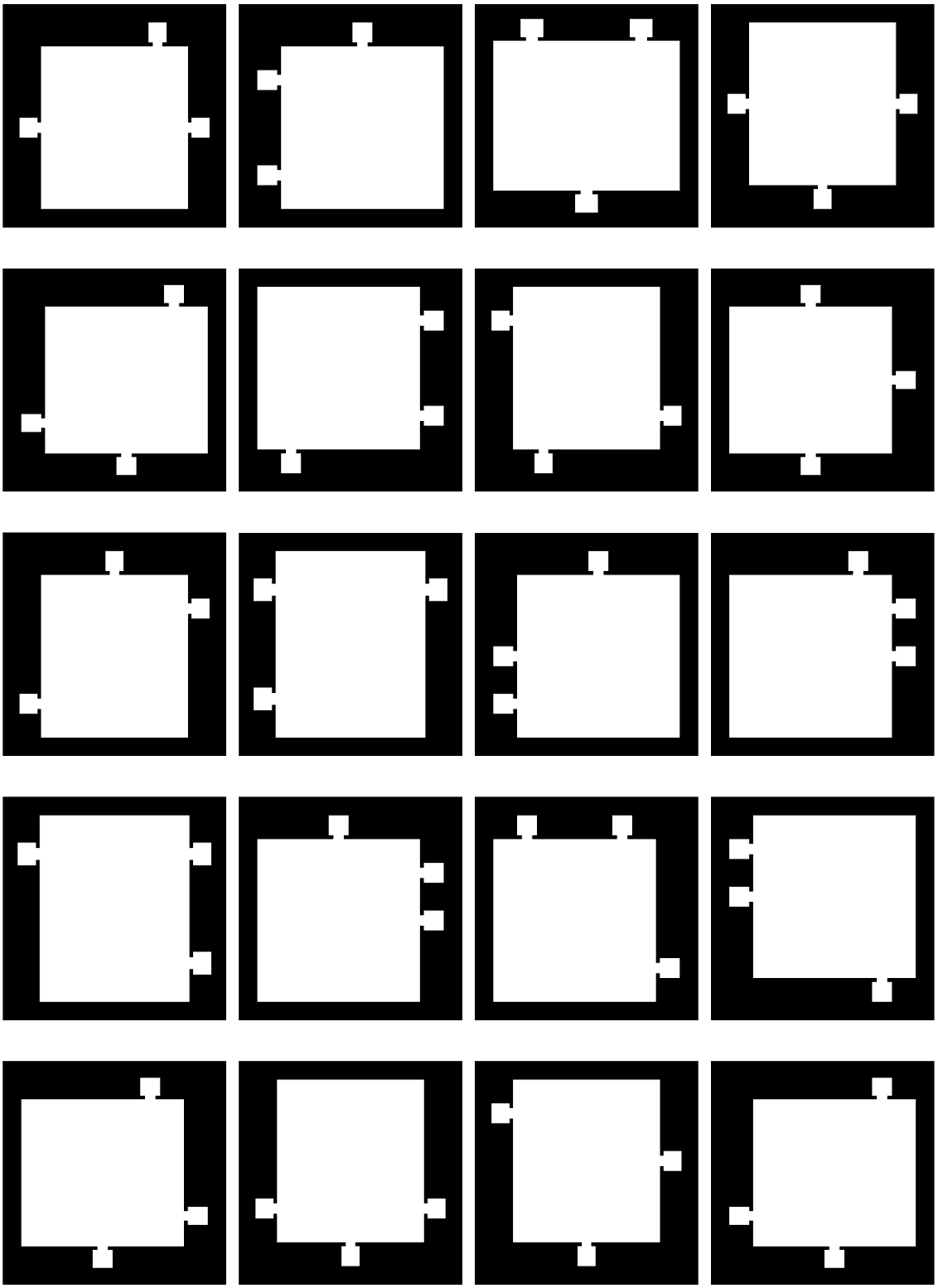


Figure A.4: Randomly generated environments containing 0 blocks and 3 rooms.

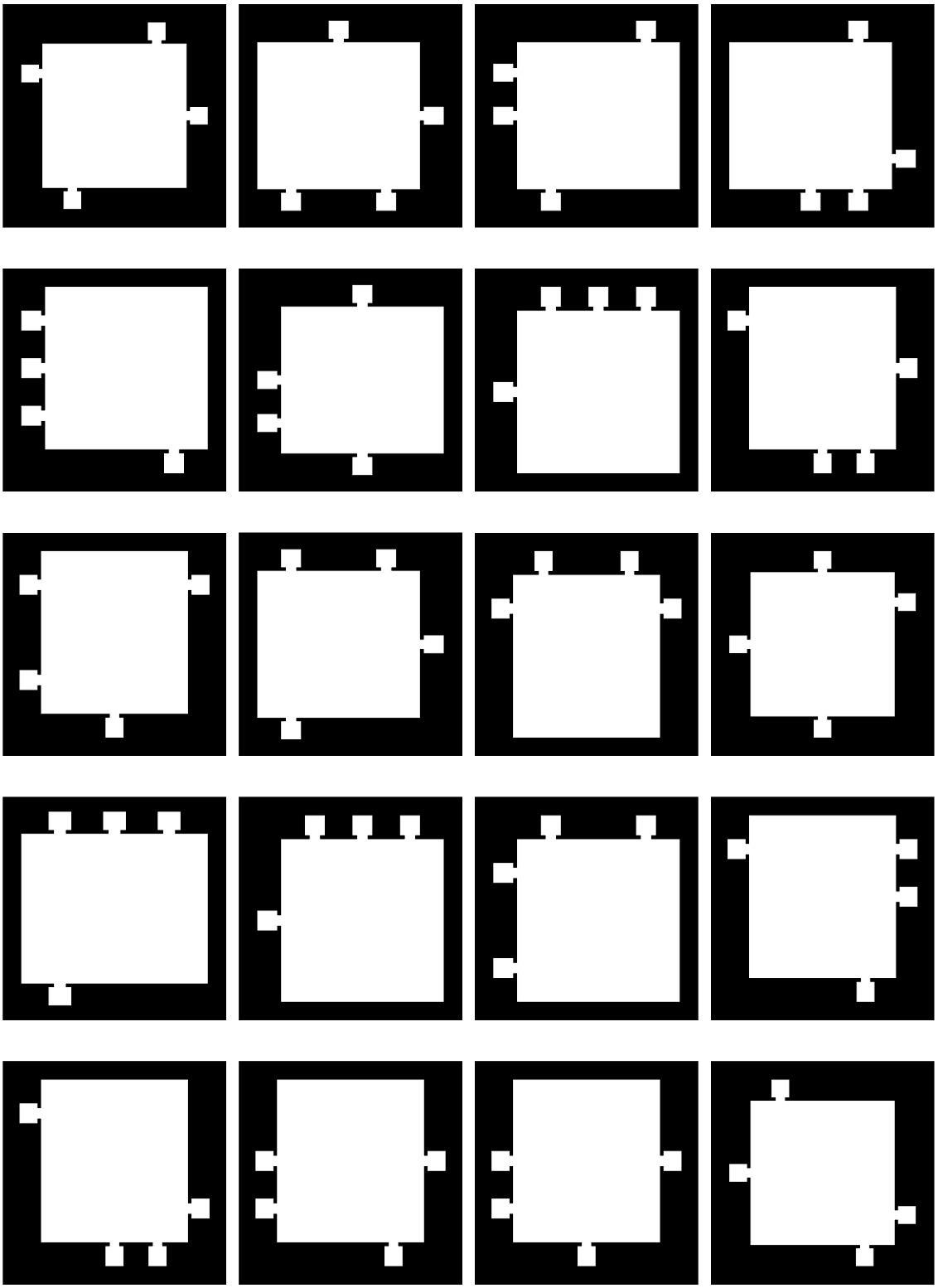


Figure A.5: Randomly generated environments containing 0 blocks and 4 rooms.

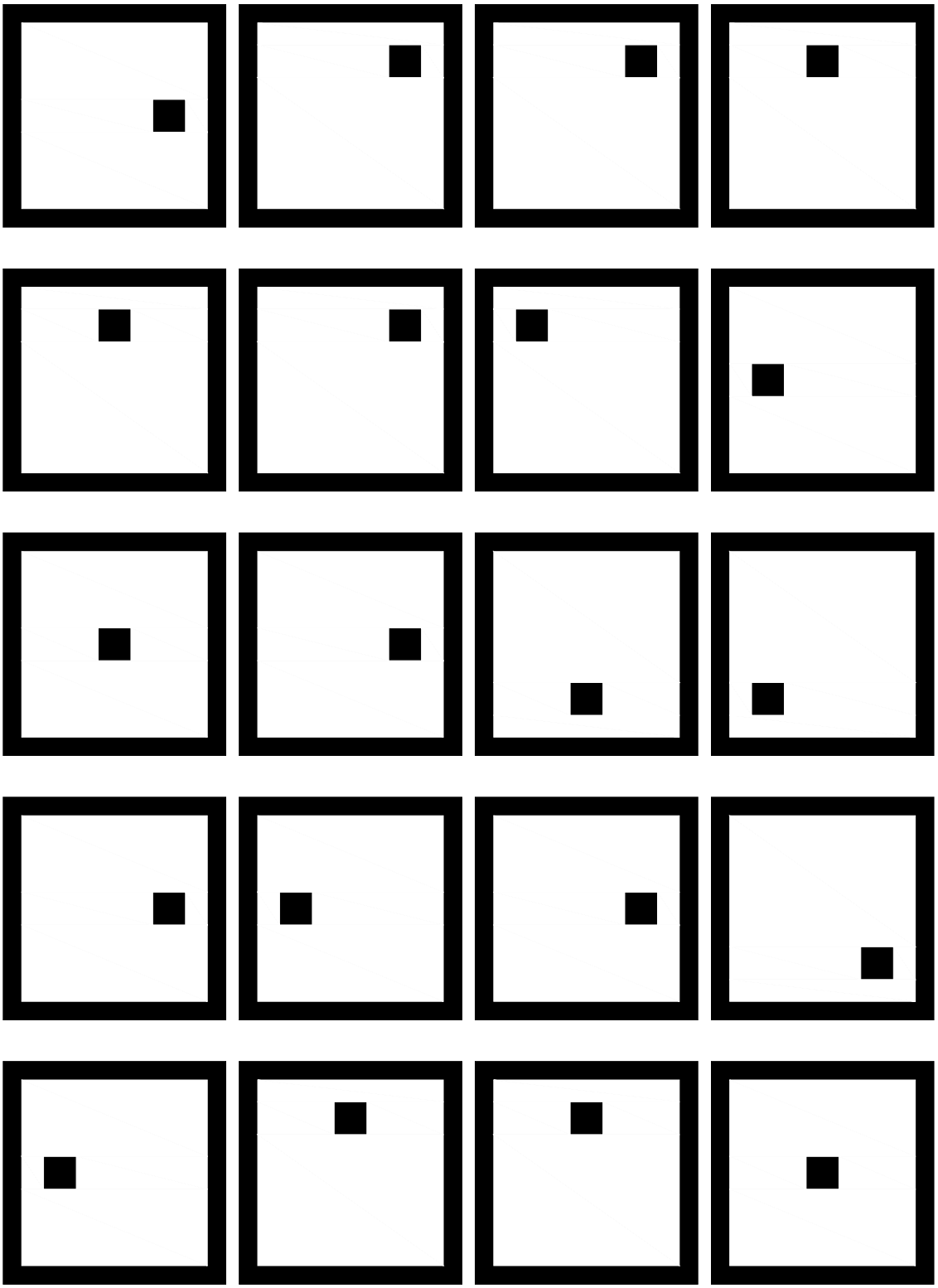


Figure A.6: Randomly generated environments containing 1 block and 0 rooms.

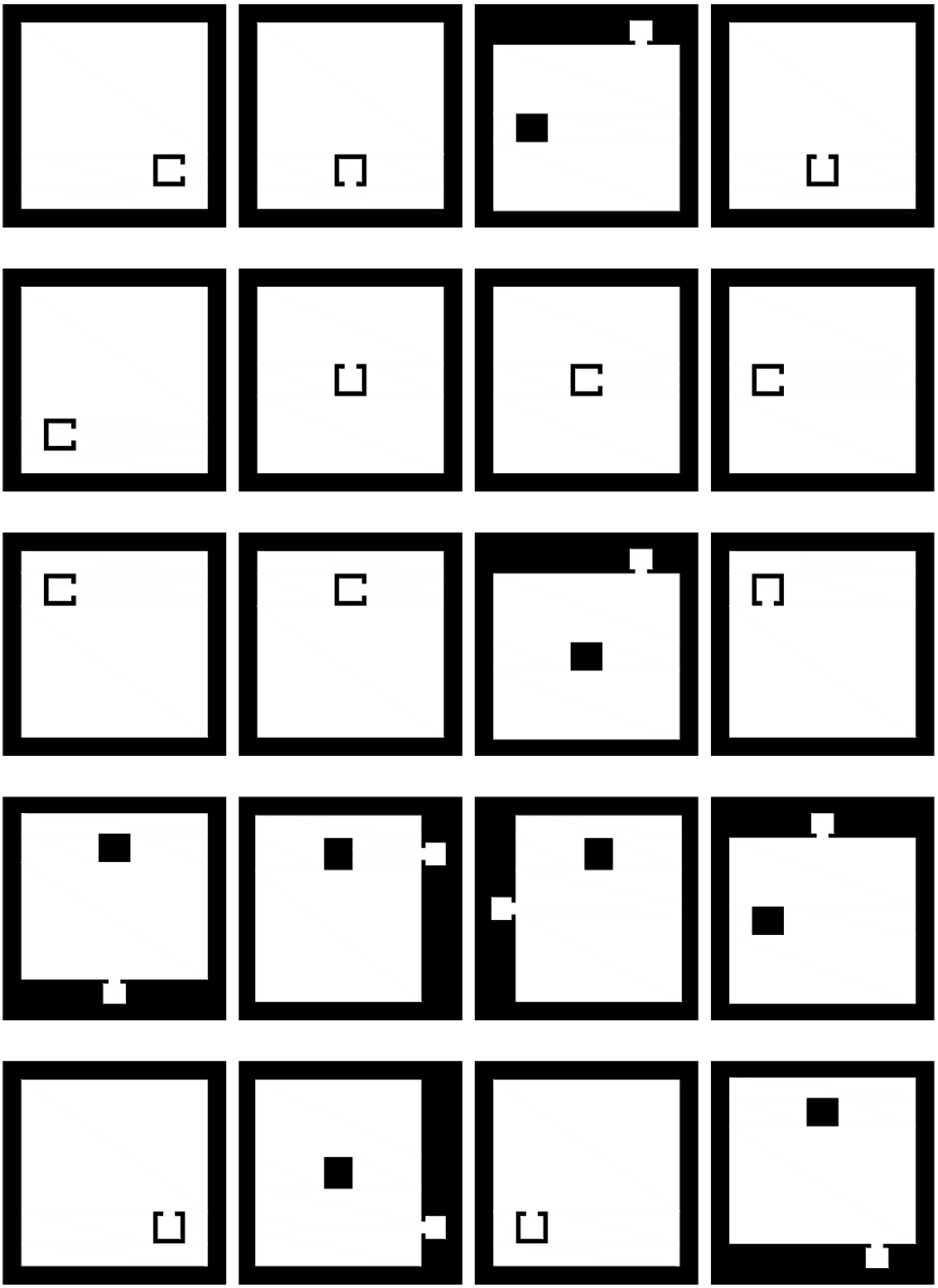


Figure A.7: Randomly generated environments containing 1 block and 1 room.

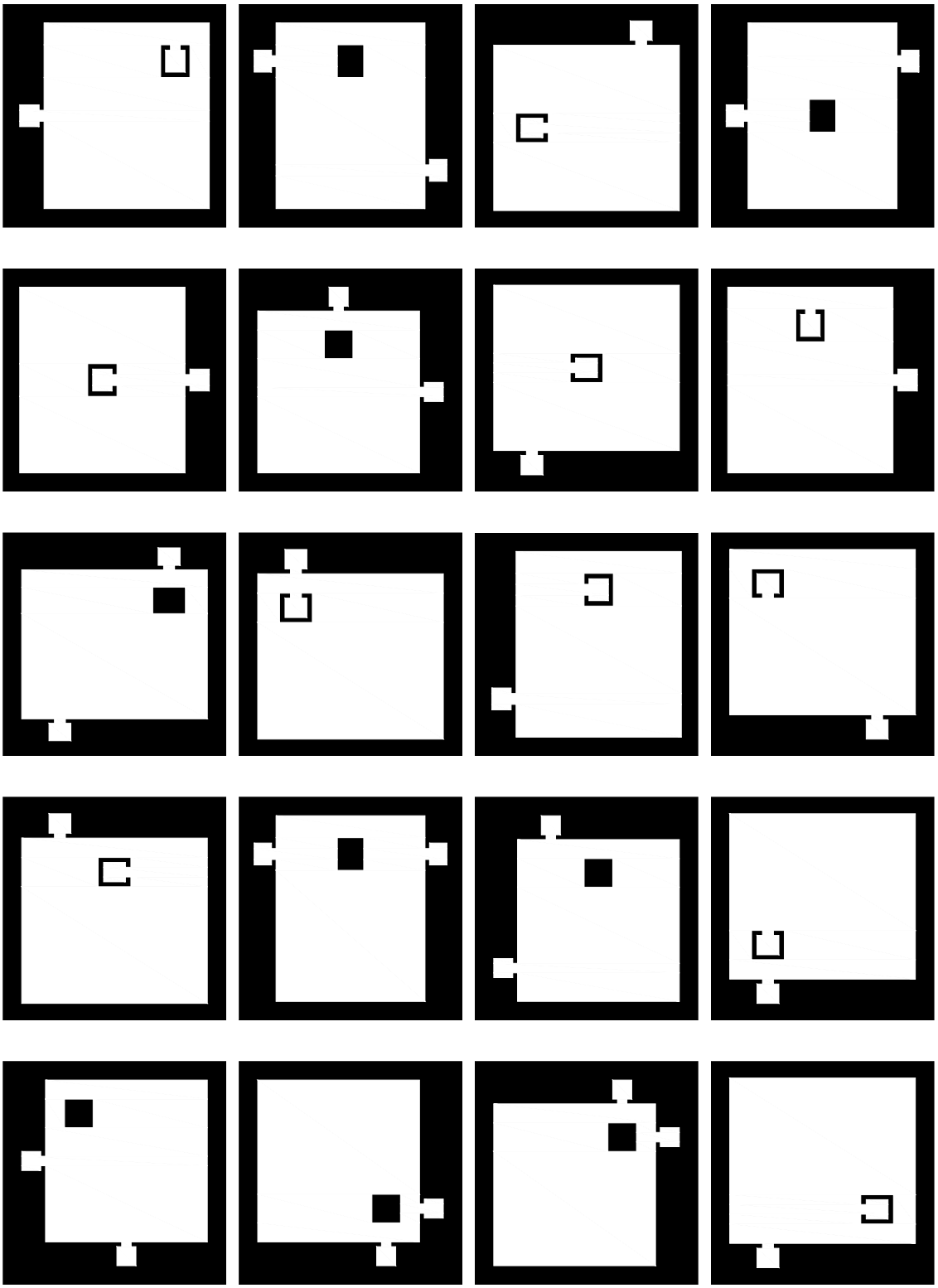


Figure A.8: Randomly generated environments containing 1 block and 2 rooms.

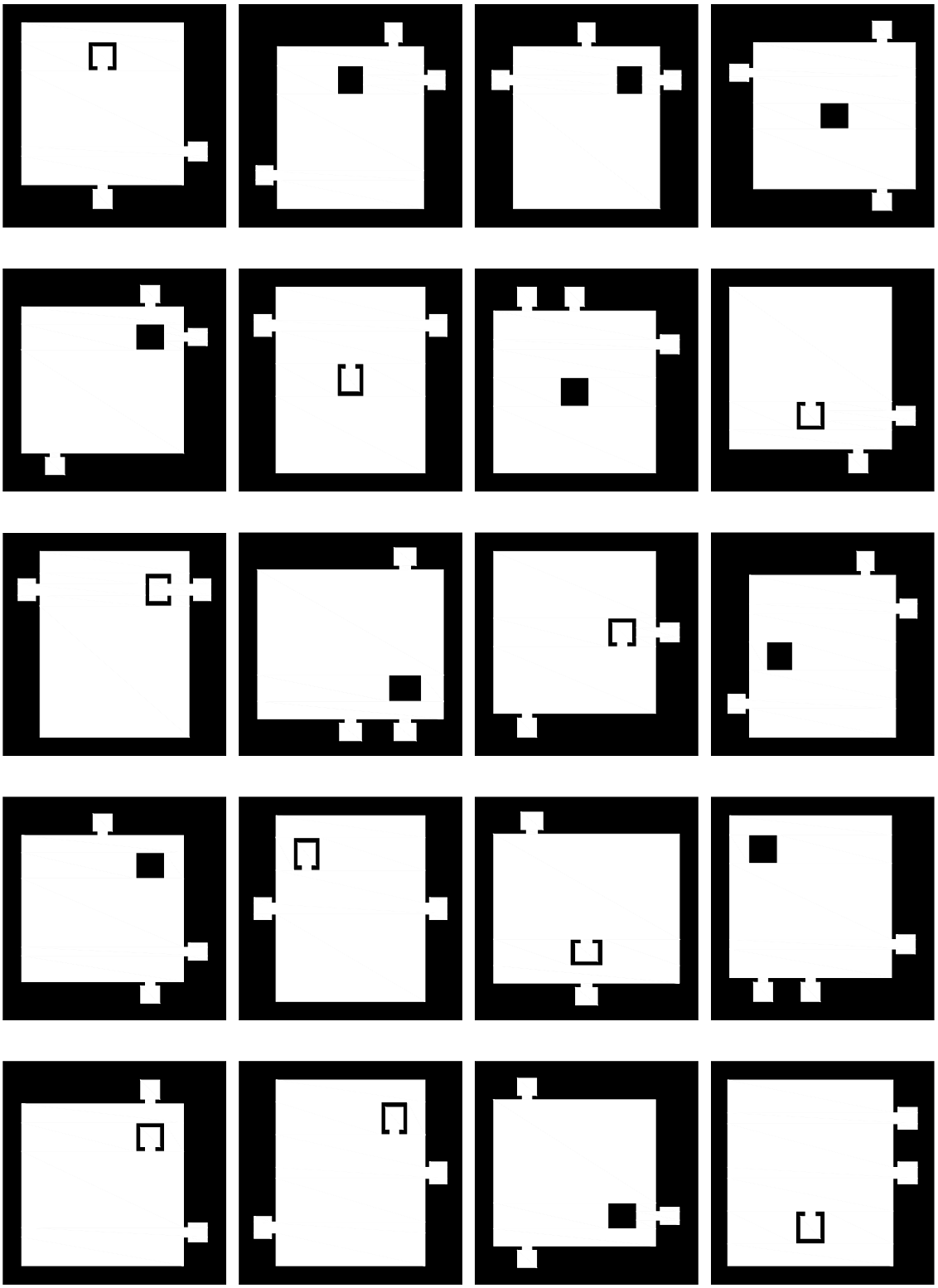


Figure A.9: Randomly generated environments containing 1 block and 3 rooms.

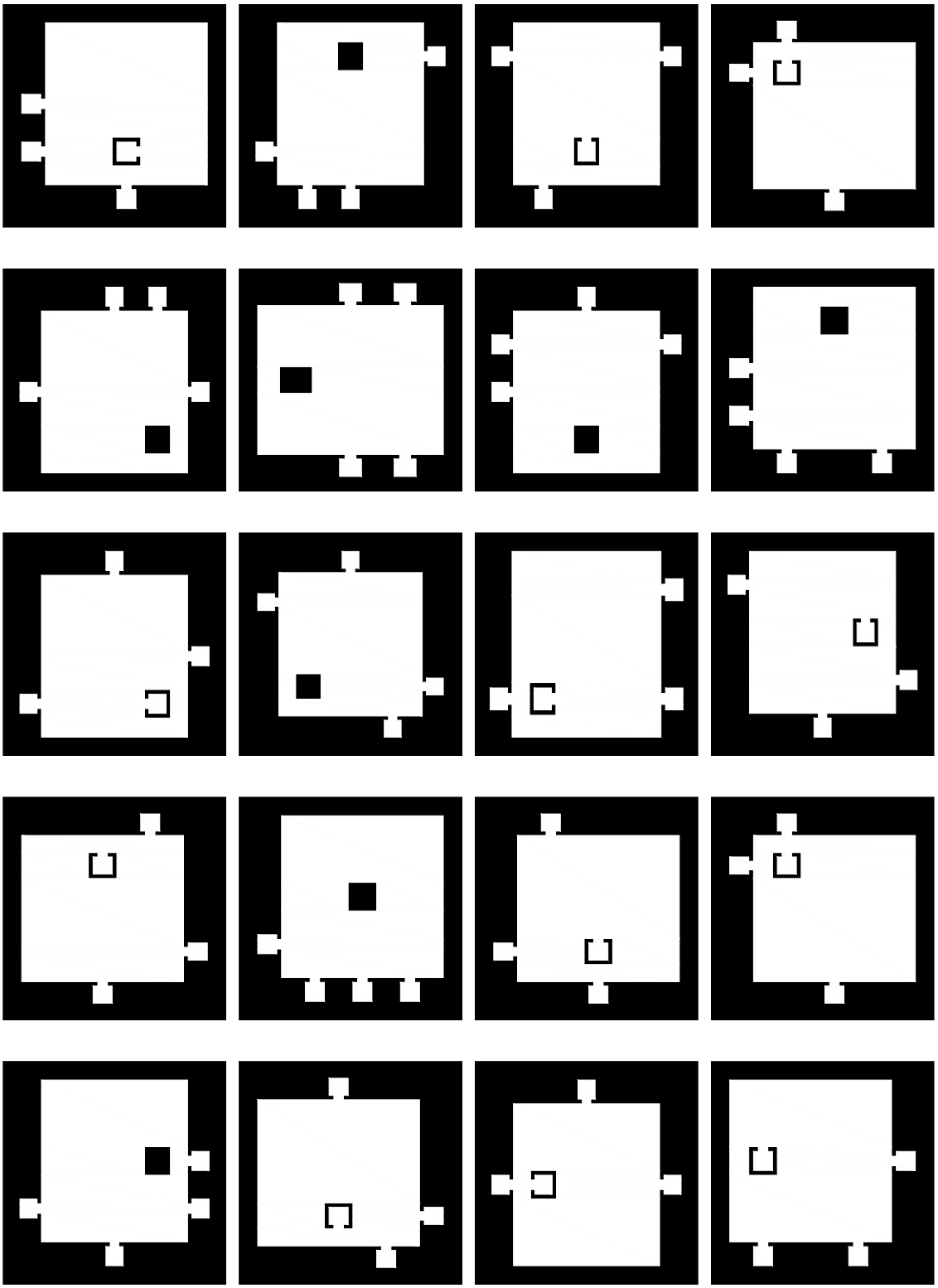


Figure A.10: Randomly generated environments containing 1 block and 4 rooms.

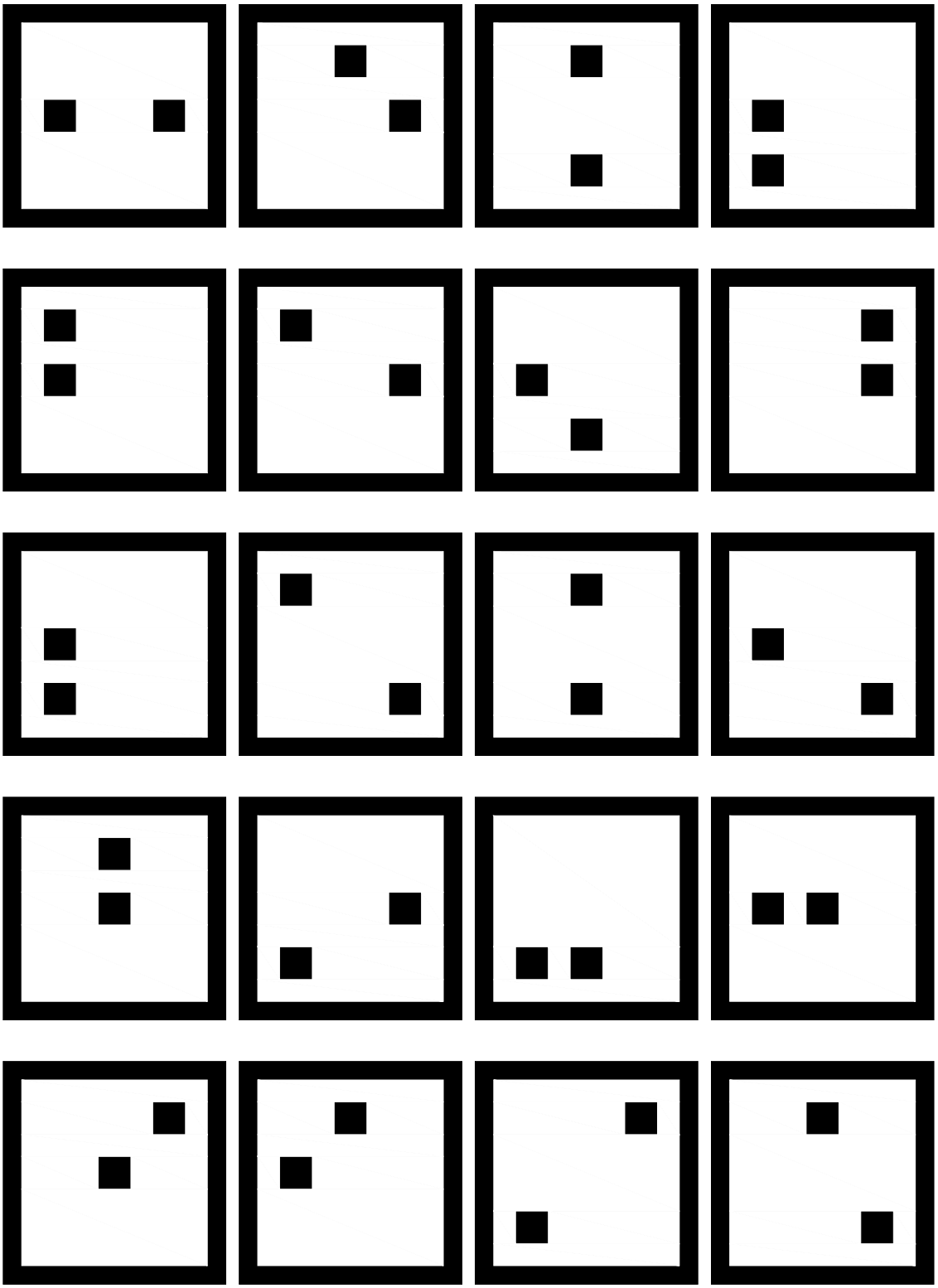


Figure A.11: Randomly generated environments containing 2 blocks and 0 rooms.

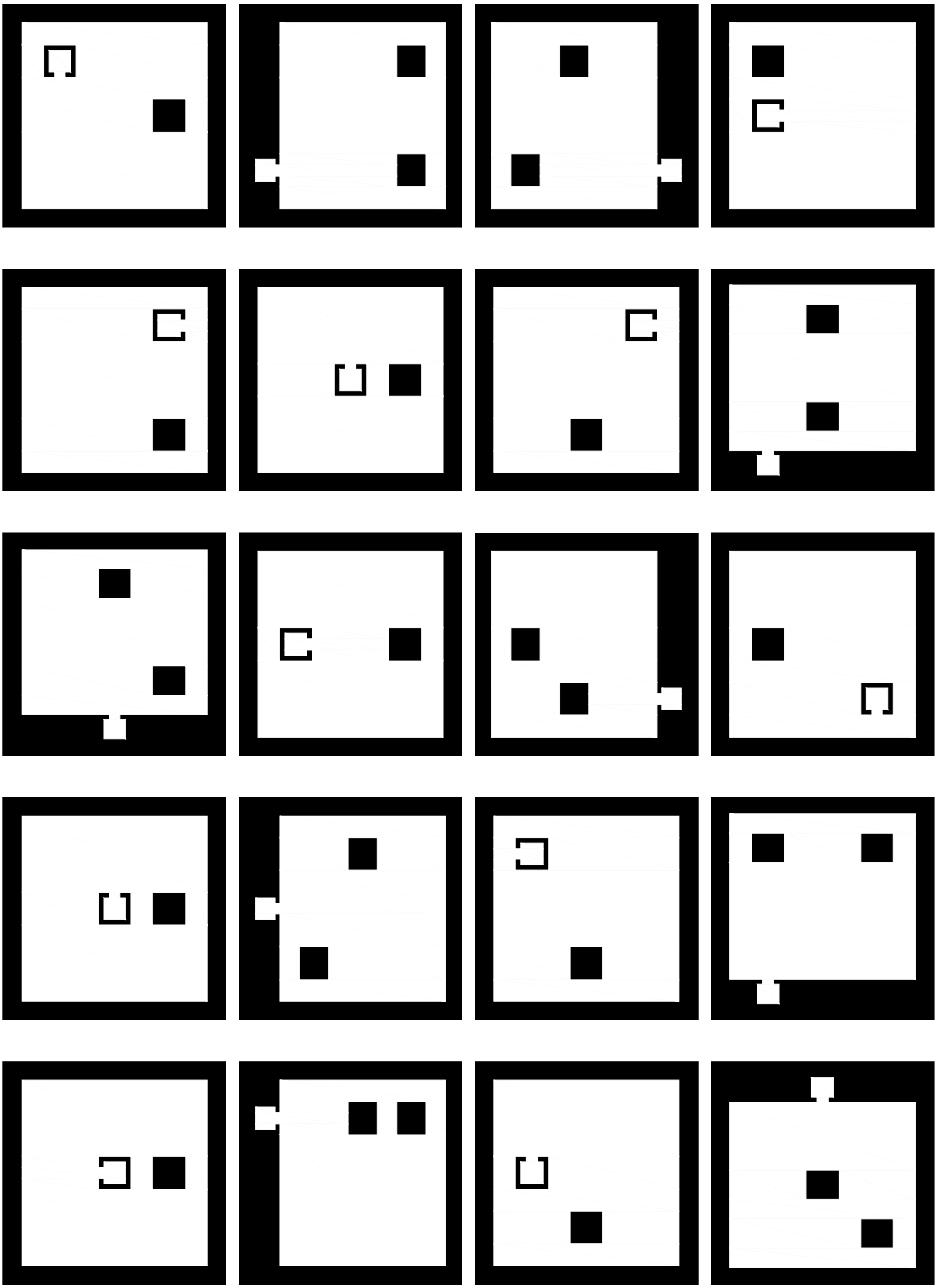


Figure A.12: Randomly generated environments containing 2 blocks and 1 room.

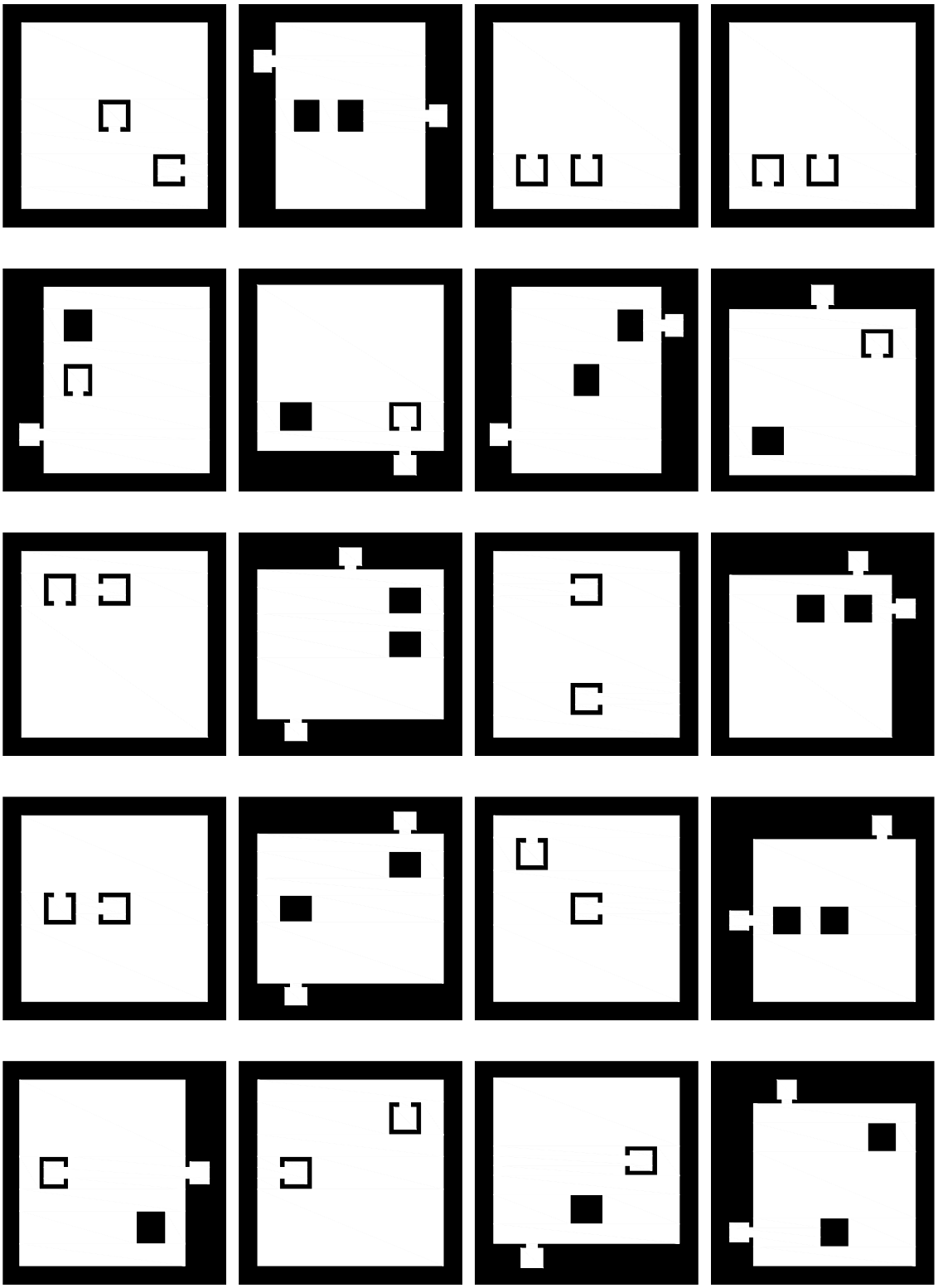


Figure A.13: Randomly generated environments containing 2 blocks and 2 rooms.

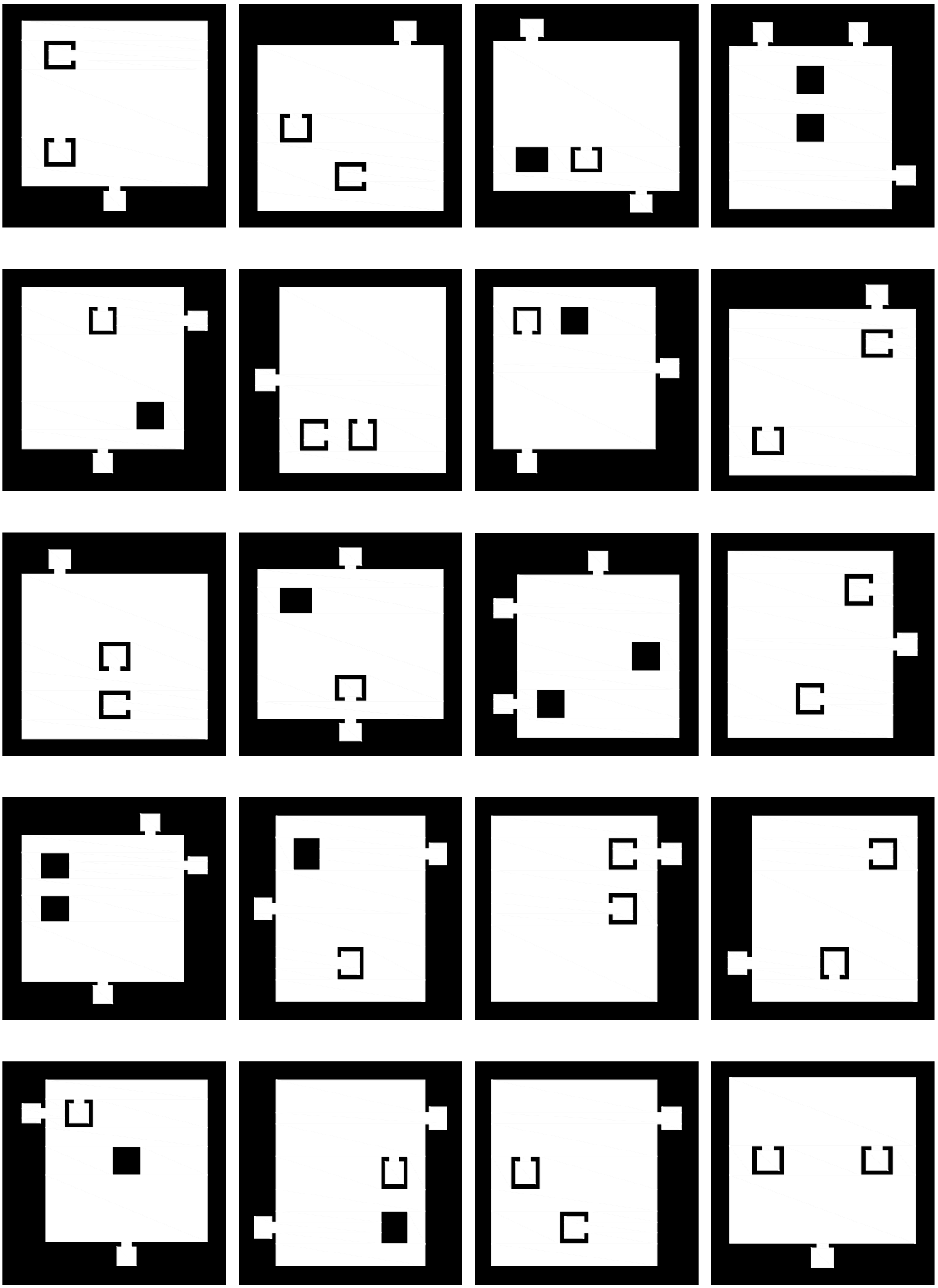


Figure A.14: Randomly generated environments containing 2 blocks and 3 rooms.

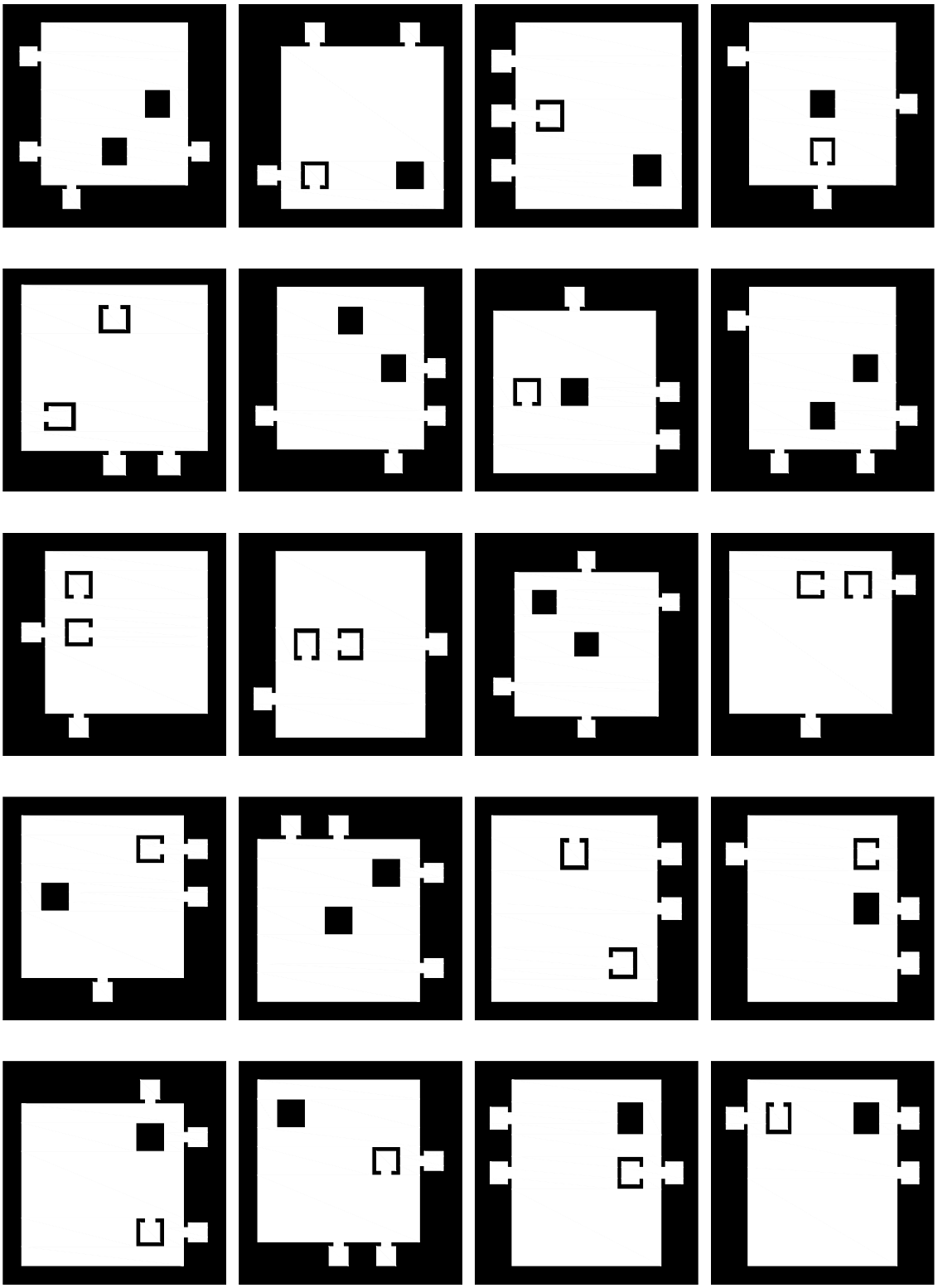


Figure A.15: Randomly generated environments containing 2 blocks and 4 rooms.

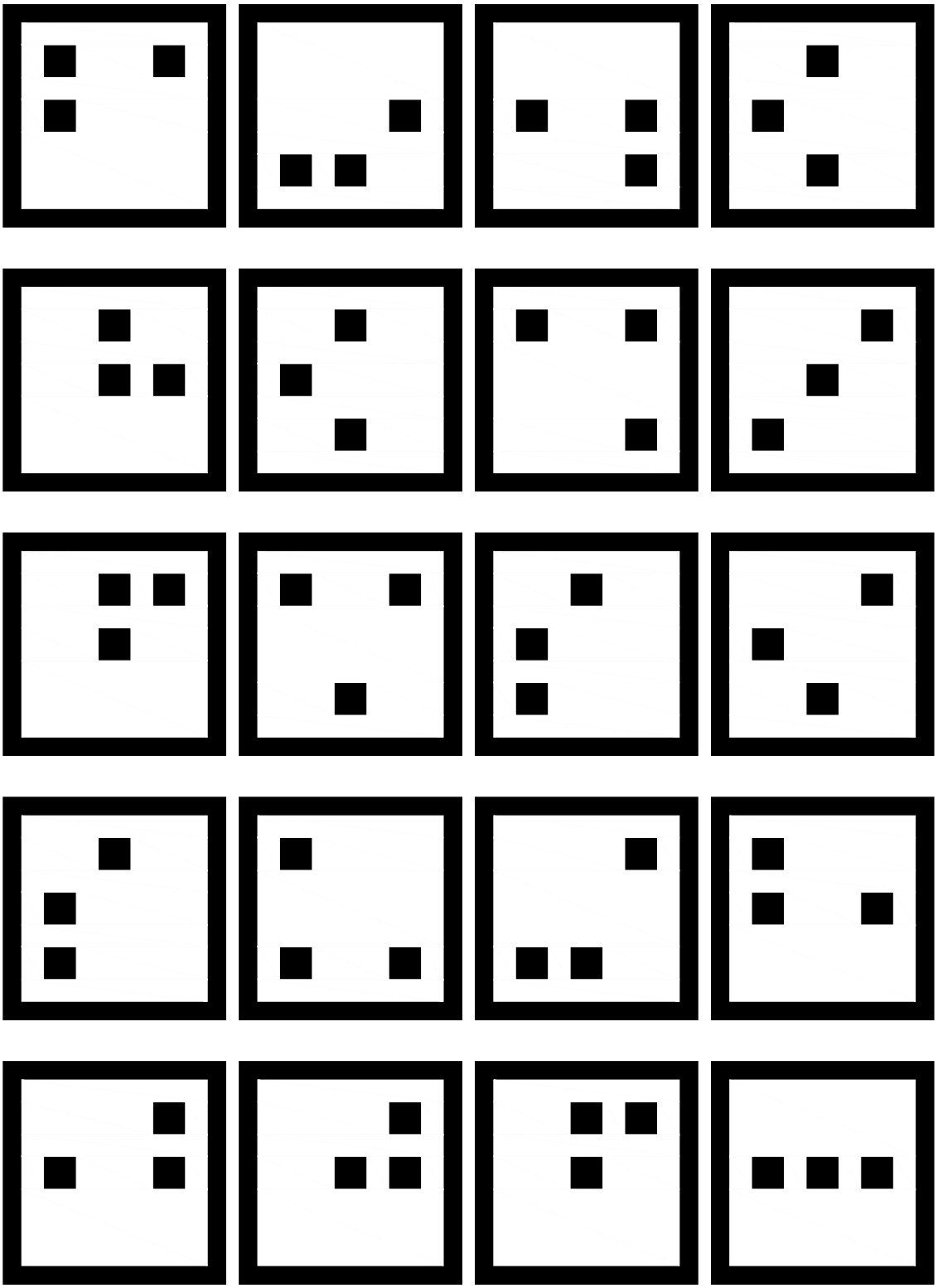


Figure A.16: Randomly generated environments containing 3 blocks and 0 rooms.

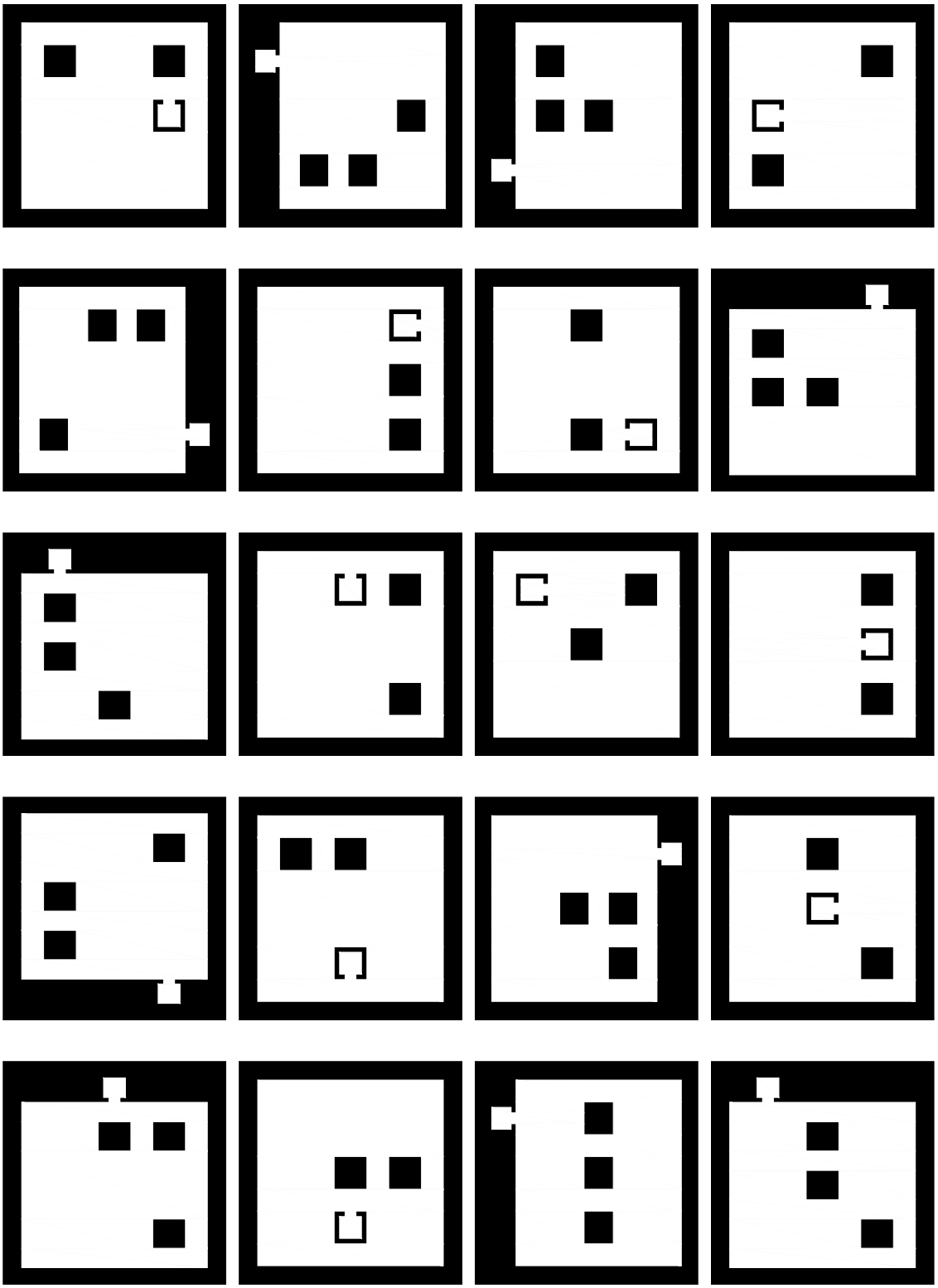


Figure A.17: Randomly generated environments containing 3 blocks and 1 room.

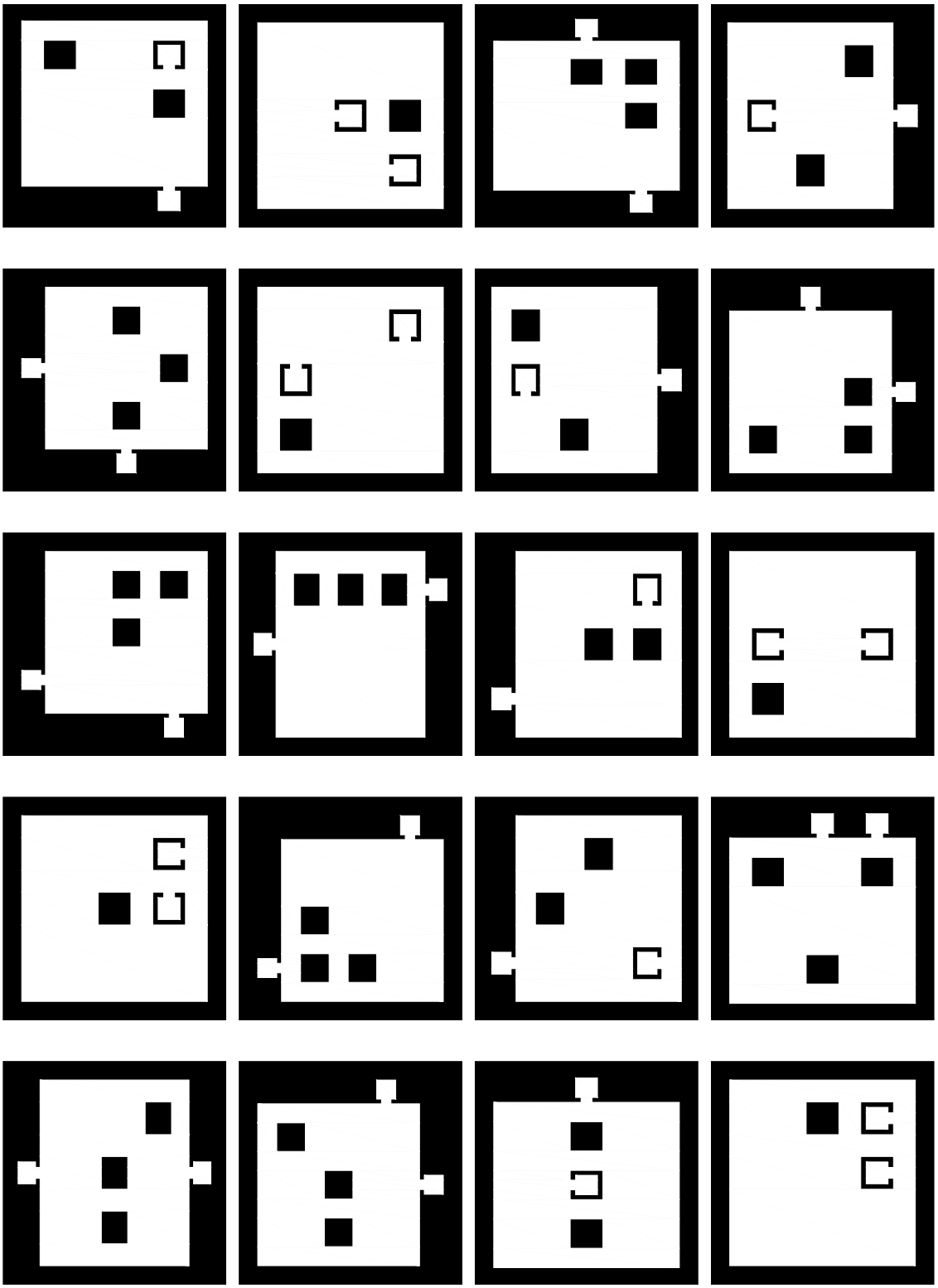


Figure A.18: Randomly generated environments containing 3 blocks and 2 rooms.

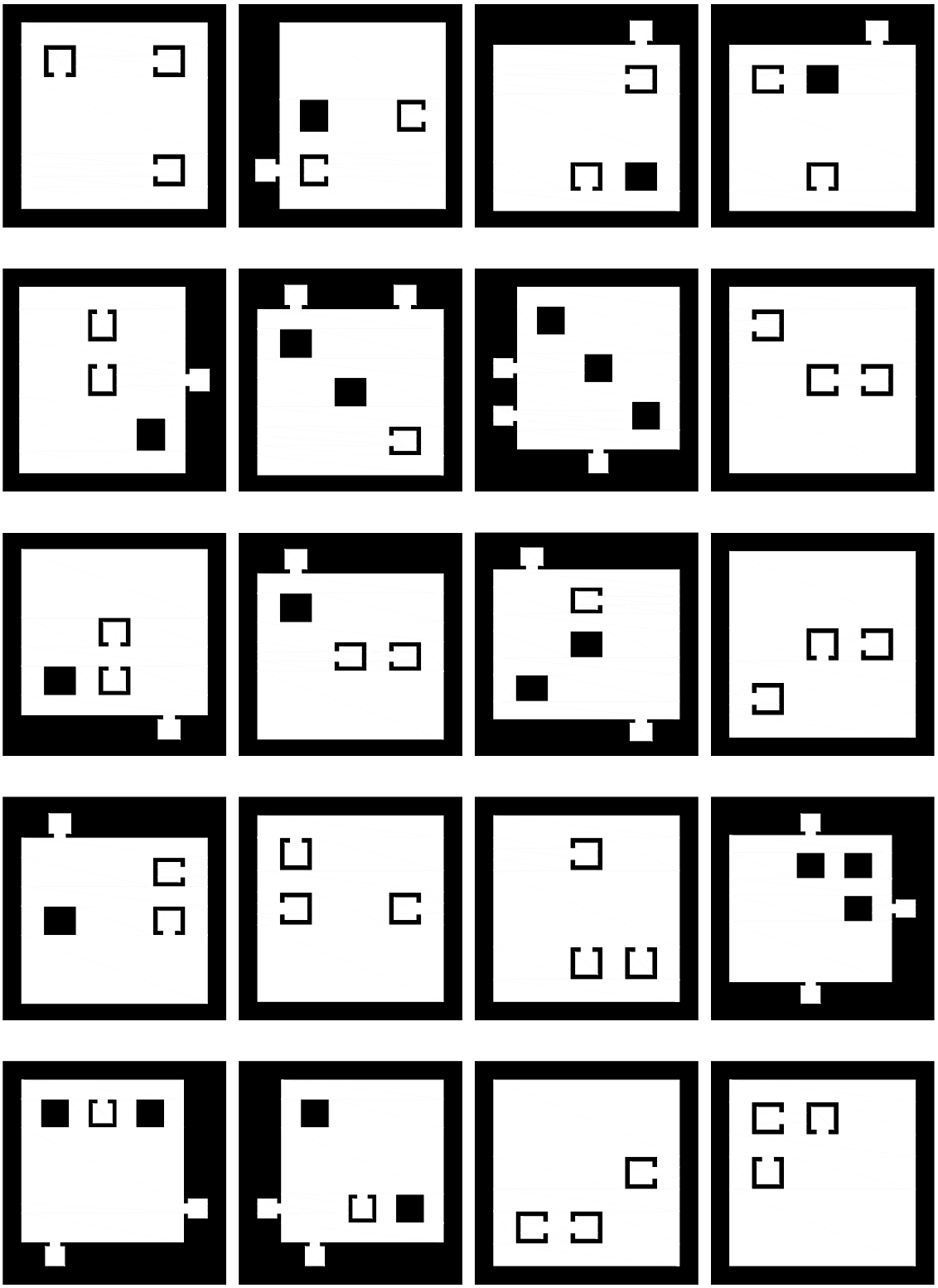


Figure A.19: Randomly generated environments containing 3 blocks and 3 rooms.

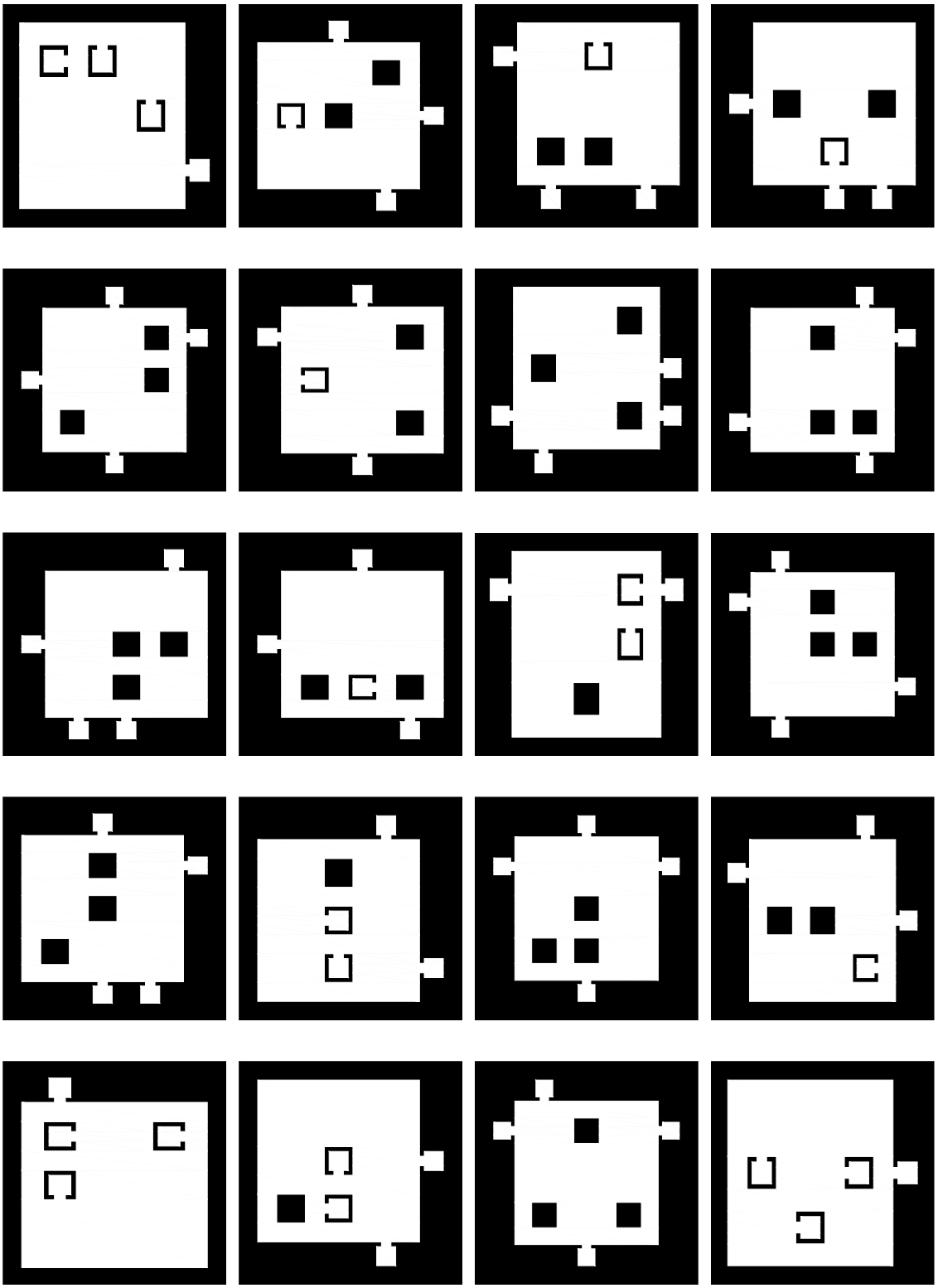


Figure A.20: Randomly generated environments containing 3 blocks and 4 rooms.

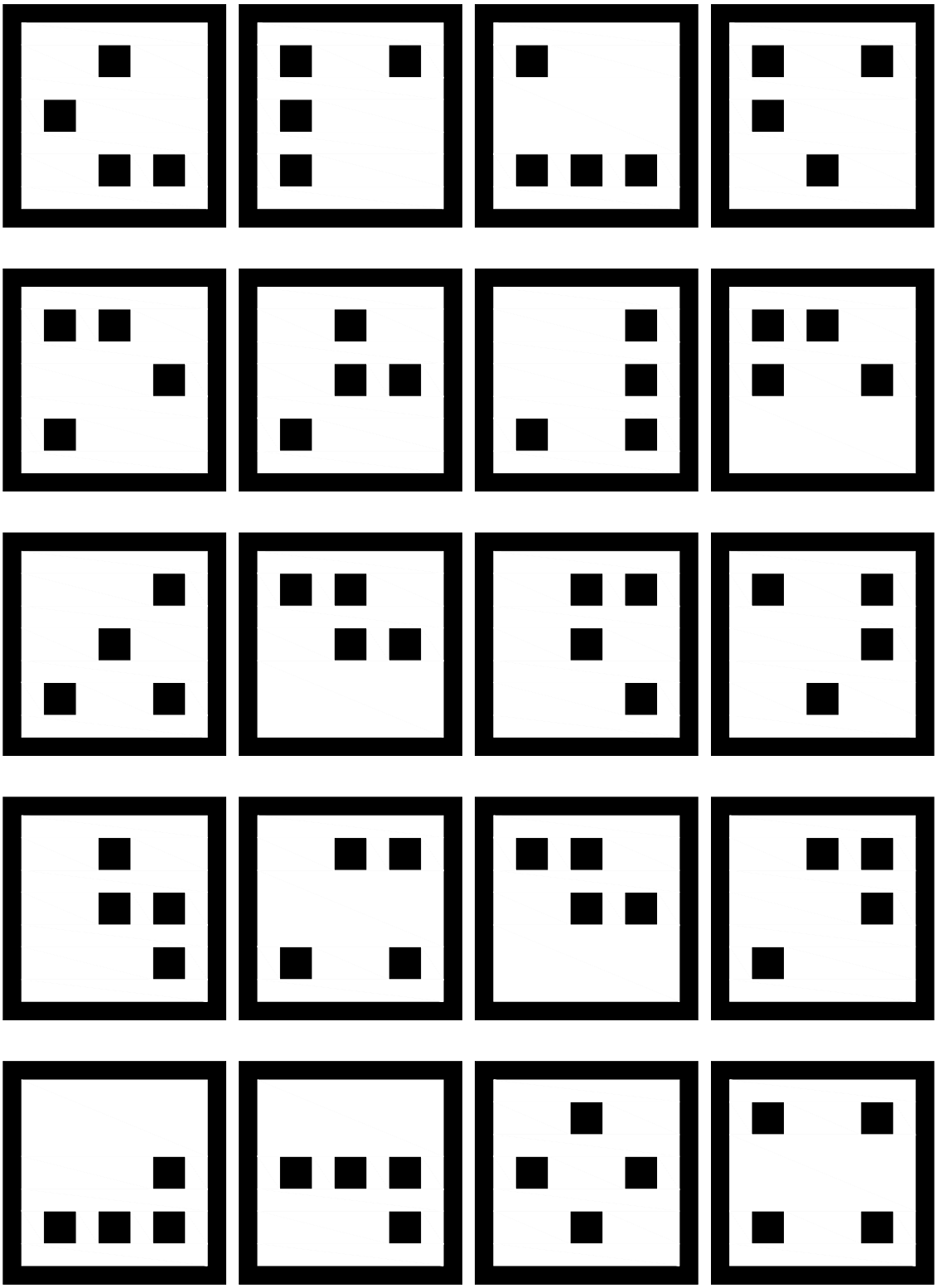


Figure A.21: Randomly generated environments containing 4 blocks and 0 rooms.

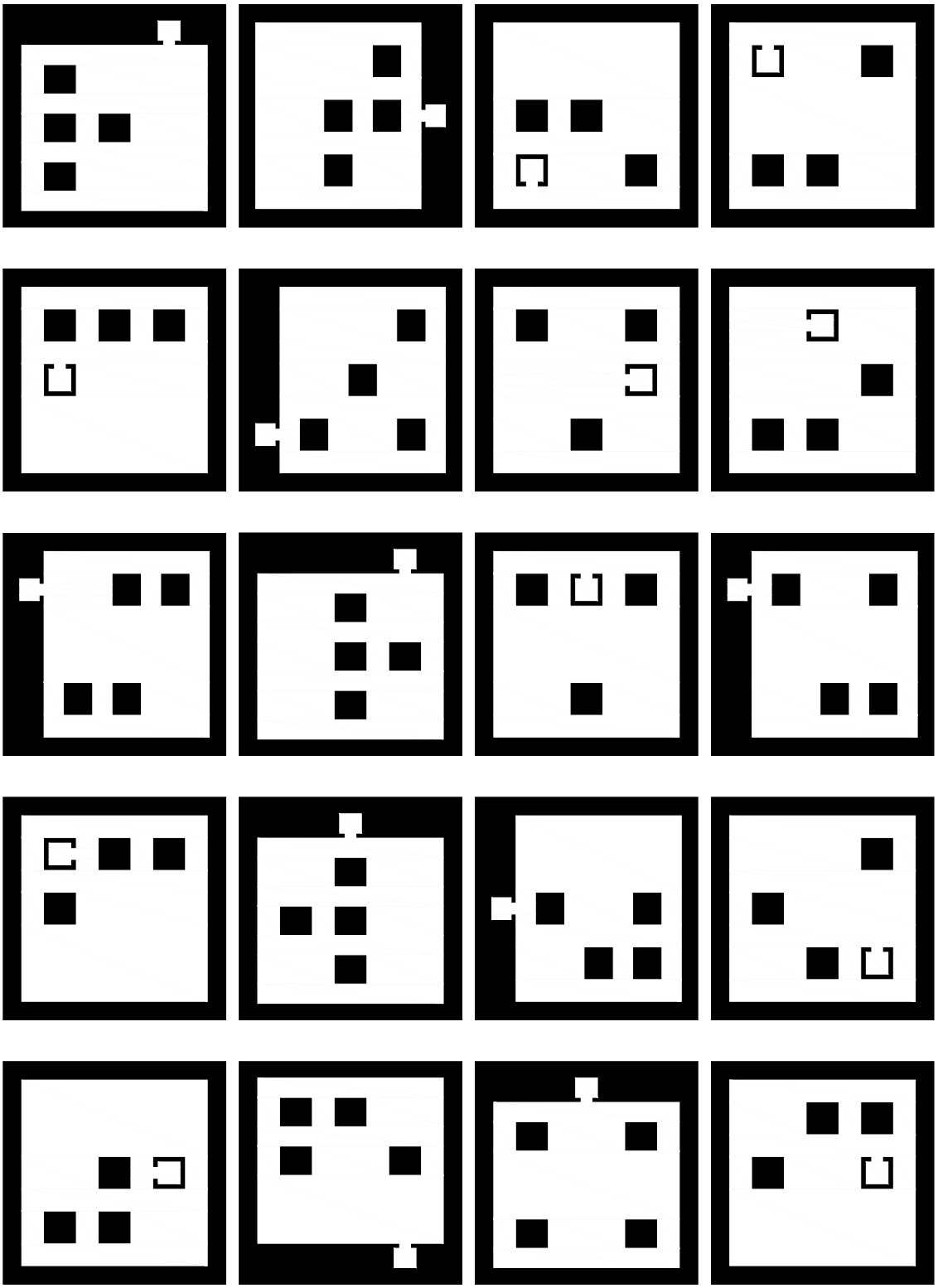


Figure A.22: Randomly generated environments containing 4 blocks and 1 room.

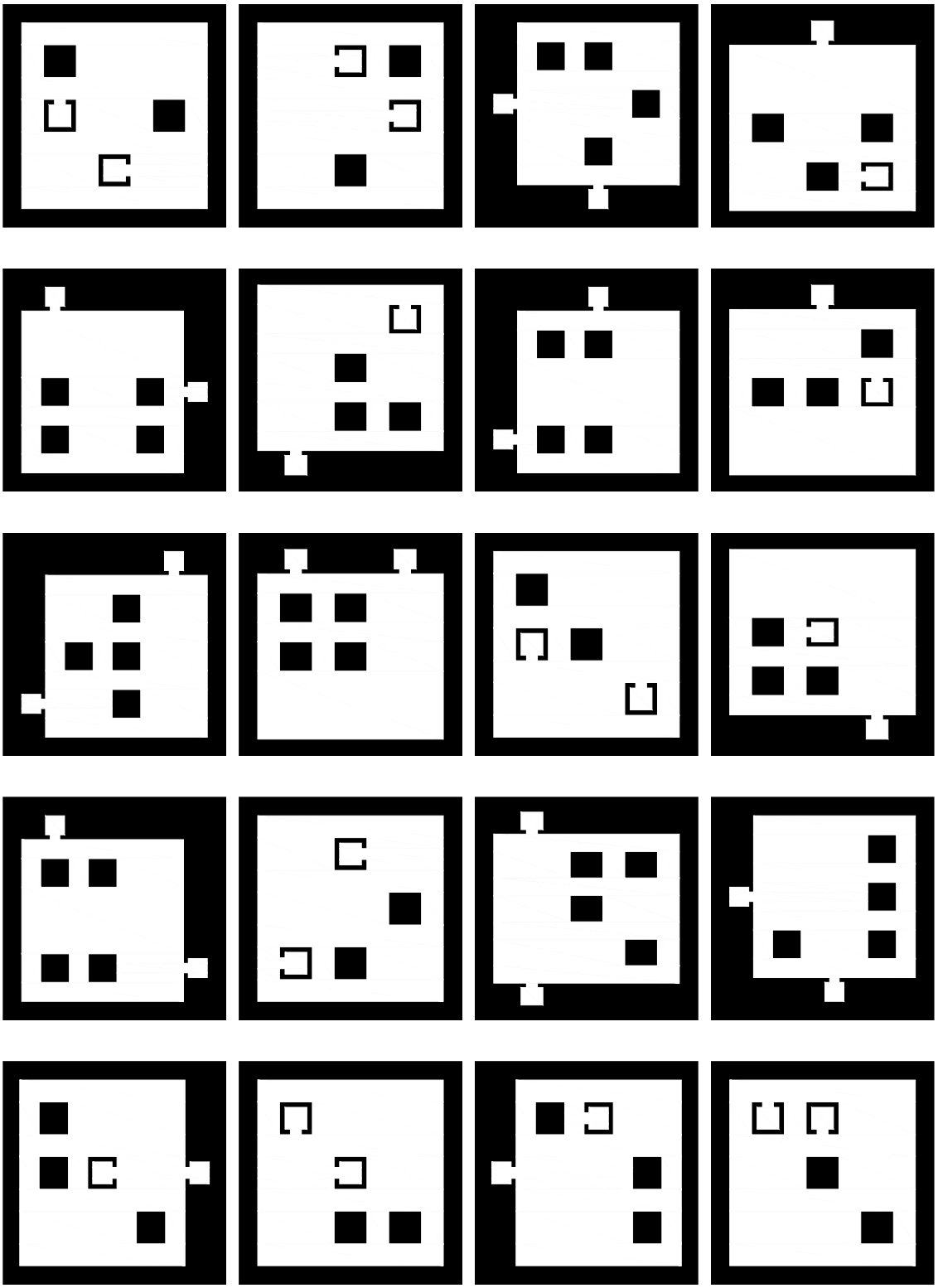


Figure A.23: Randomly generated environments containing 4 blocks and 2 rooms.

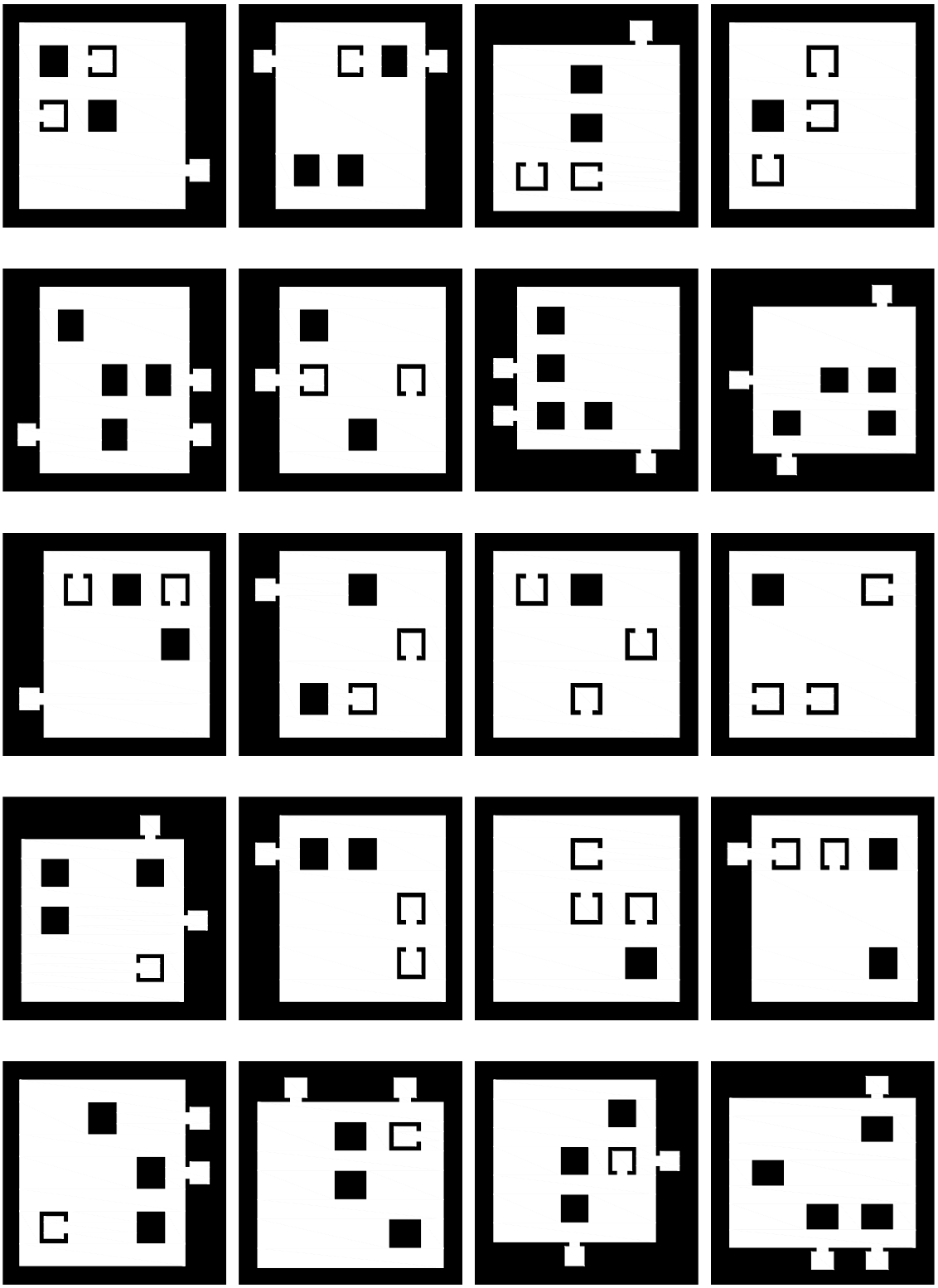


Figure A.24: Randomly generated environments containing 4 blocks and 3 rooms.

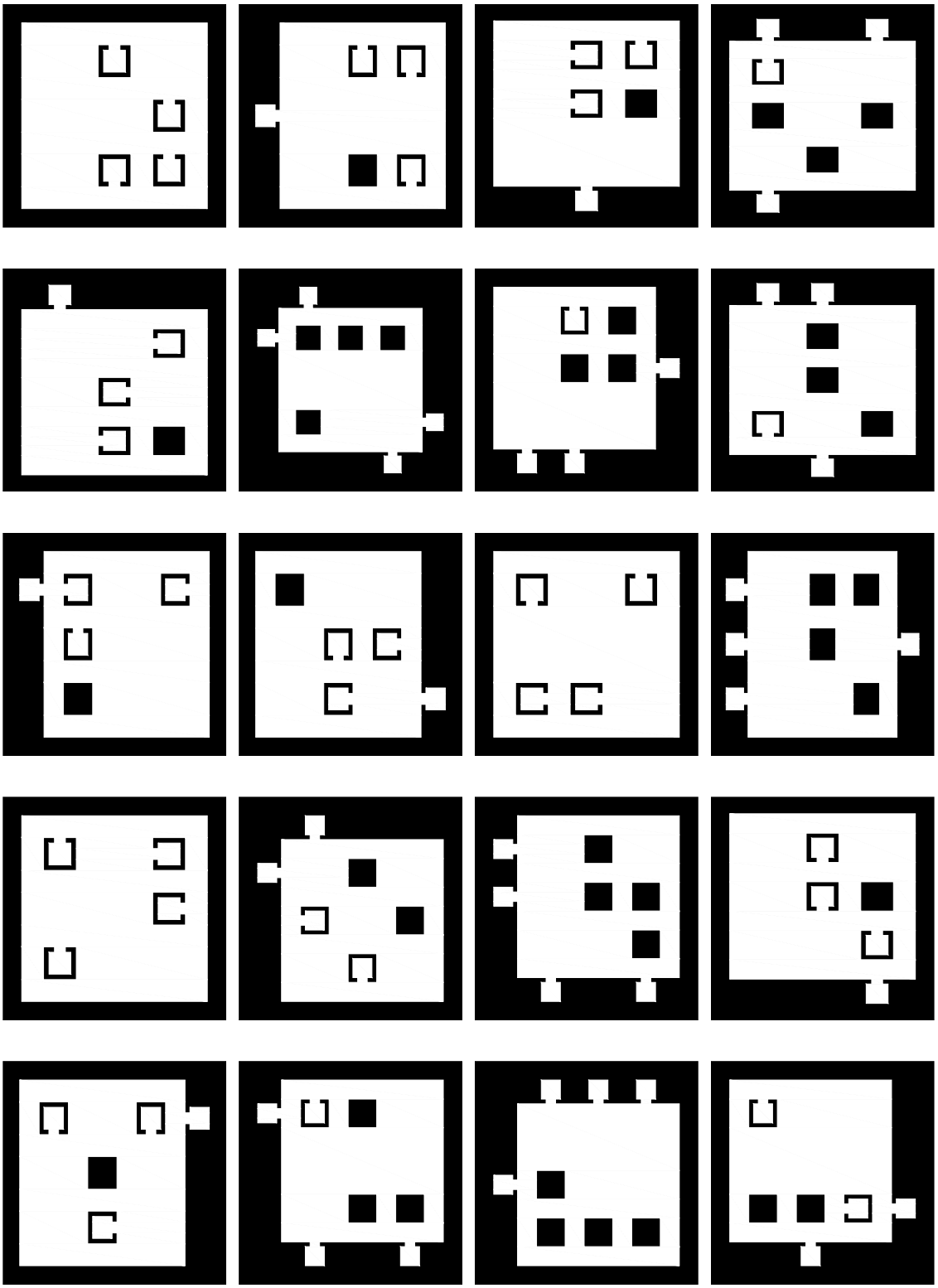


Figure A.25: Randomly generated environments containing 4 blocks and 4 rooms.