PROCTD: A PROCEDURAL TOWER DEFENSE BENCHMARK FOR SPATIAL REASONING AND LONG-HORIZON PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Tower Defense (TD) games combine long-horizon economic planning with short-horizon spatial reasoning, yet remain largely absent from reinforcement learning benchmarks. We introduce ProcTD, a procedurally generated, Gymnasium-compatible TD environment with partial observability, configurable spatial scale, heterogeneous tower–enemy matchups, technology progression, and an optional tower-selling mechanic. The raw action space of ProcTD grows quadratically with map size, from 1,540 discrete actions on 16×16 to 6,148 on 32×32, yet only a small subset is valid at any step. Centrally, ProcTD is optimized for large-scale experimentation on commodity hardware, as its JIT-compiled implementation exceeds 10,000 environment steps per second on a single CPU core. We benchmark general-purpose RL methods and find that performance degrades substantially as spatial scale increases. As a feasibility baseline, we introduce Macro-Action Policy with Symbolic Placement Proximal Policy Optimization (MAPS-PPO), which combines a learned high-level policy with handcrafted spatial placement that reduces the action space by more than 80%. MAPS-PPO reaches the maximum 50 waves in the median on 16×16 maps with an IQR of 0, and again a median of 50 on 32×32 but with an IQR of 9. PPO reaches 42 and 27 waves, respectively. ProcTD therefore provides a lightweight benchmark for studying large, structured, spatially grounded action spaces and is currently not solved by general-purpose methods. We release the environment open-source with baseline implementations and an asymmetric attacker–defender extension that allows for competitive benchmarking.

1 INTRODUCTION

Reinforcement learning (RL) benchmarks have historically driven methodological progress by providing controlled environments in which increasingly capable agents can be compared. Successful benchmarks include visually simple environments with compact action spaces, such as Atari (Bellemare et al., 2013), complex strategic environments, such as StarCraft II (Vinyals et al., 2017; 2019) and NetHack (Küttler et al., 2020), and procedurally generated environments, such as Procgen (Cobbe et al., 2020). However, existing benchmarks emphasize different challenges, leaving room for environments that combine spatial reasoning, long-term resource allocation, state-dependent action spaces, partial observability, and procedural variation within a single task, while remaining computationally inexpensive enough for large-scale experimentation.

Tower Defense (TD) is a strategy-game genre centered on resource allocation and spatial placement of defensive structures (Avery et al., 2011), which provides a natural setting in which these challenges arise jointly. An agent must decide not only *what* action to take but also *where* to take it: towers must be placed at suitable locations, upgraded over time, and selected according to their complementary strengths and weaknesses. Defensive structures are committed before the full composition of future waves is known, so the agent must make spatial and economic investments under uncertainty. In addition to short-horizon geometric optimization over unit placement, TD utilizes a long-horizon economic process involving capital accumulation, technology unlocks, and deferred returns that may only materialize many waves later. Increasing the size of the map enlarges both

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

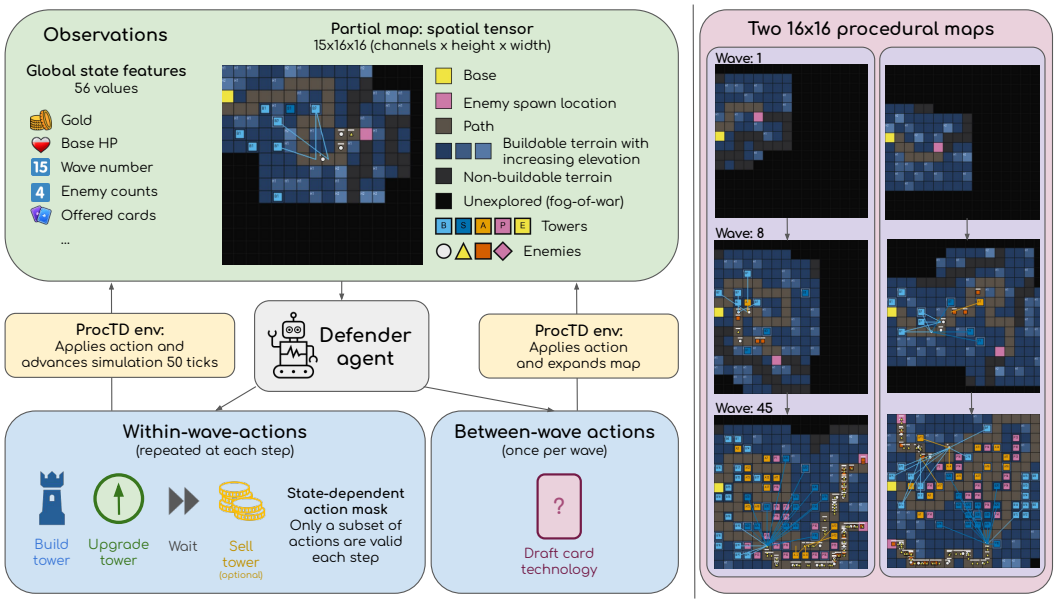


Figure 1: **Overview of ProcTD**. (Left) The defender acts on spatial and global observations through state-dependent within-wave actions and between-wave technology drafts. (Right) Two procedurally generated maps are progressively revealed over the course of an episode.

the spatial observation and the set of possible placement actions. The spatial action space grows quadratically with map size, while only a state-dependent subset of actions is valid at any given step. Mechanics such as selling introduce additional long-horizon trade-offs by allowing earlier investments to be reversed and resources reallocated. At the same time, Tower Defense is amenable to fast simulation because the environment does not require photorealistic rendering or continuous physical control.

We introduce **ProcTD** (procedural Tower Defense), an open-source and configurable Tower Defense benchmark designed to expose these challenges in a controlled manner[1]. ProcTD generates maps procedurally and provides explicit controls over the environment complexity, allowing difficulty to be increased through spatial scaling and strategic mechanics. The environment combines spatial observations with global game information and exposes a large, state-dependent action space (Dulac-Arnold et al., 2015) that covers tower placement and management. Randomized technology availability and heterogeneous tower-enemy matchups further require agents to adapt their strategy across episodes rather than rely on a fixed build order. It additionally uses fog-of-war partial observability and provides an asymmetric attacker–defender multi-agent extension. The implementation is compatible with the Gymnasium (Towers et al., 2025) and PettingZoo (Terry et al., 2021) APIs and is optimized for high-throughput simulation, exceeding $10,000$ steps per second on consumer CPUs.

Our main contributions are:

- We introduce **ProcTD**, a fast, procedurally generated Tower Defense benchmark combining spatial decision-making, resource management, large state-dependent action spaces, and configurable strategic complexity, with explicit and interpretable **difficulty controls** over spatial scale and mechanics such as tower selling.
- We **benchmark several well-established general-purpose RL algorithms** on ProcTD to provide reference performance across benchmark configurations and to identify regimes in which standard end-to-end approaches struggle.
- We provide Macro-Action Policy with Symbolic Placement Proximal Policy Optimization (MAPS-PPO), a **neuro-symbolic feasibility baseline** that combines learned high-level

[1]Some mechanics are inspired by the commercial roguelike Tower Defense game *Rogue Tower* (Die of Death Games, 2022)

decision-making with handcrafted spatial structure, testing whether difficult benchmark configurations remain tractable when appropriate domain structure is available.

- We release ProcTD as an **extensible Gymnasium-compatible environment**[2] with procedural generation and partial-observability options.
- We introduce an **asymmetric multi-agent attacker–defender mode** built on the PettingZoo API, enabling evaluation against adaptive opponents while remaining directly comparable to the single-agent setting.

Our experiments show that PPO (Schulman et al., 2017) achieves strong performance on 16×16 maps but degrades substantially on 32×32. Replacing learned placement with a handcrafted spatial prior (MAPS-PPO) more consistently retains performance on larger scales. In the attacker–defender mode, defenders trained against the fixed wave generator generalize poorly to alternative attacker behaviors.

2 Related Work

Strategy-game RL environments. Games have long served as benchmarks for sequential decision-making, ranging from Atari (Bellemare et al., 2013) to large-scale commercial strategy games such as StarCraft II (Vinyals et al., 2017; 2019) and Dota 2 (Berner et al., 2019). While Atari is largely saturated as a research testbed, StarCraft II and Dota 2 require computational scales that are prohibitive for many experiments. Purpose-built research environments are more attainable and provide more controlled settings for studying specific aspects of strategic decision-making: microRTS (Ontañón et al., 2018; Huang et al., 2021) and Tribes (Perez-Liebana et al., 2020) emphasize spatial control, strategic planning, and resource management in lightweight strategy-game settings, the StarCraft Multi-Agent Challenge (Samvelyan et al., 2019) isolates cooperative multi-agent control in micromanagement scenarios, and Neural MMO (Suarez et al., 2023) focuses on long-horizon multi-agent learning and generalization in procedurally generated worlds with resource gathering, exploration, and competition. ProcTD differs from these environments in the temporal structure of its core decision: rather than issuing dynamic commands to mobile units, the agent commits to static tower placements that, in the default configuration, persist for the remaining episode. A single early placement must remain load-bearing across dozens of subsequent waves rather than being correctable at the next timestep, shifting the credit-assignment problem from short-horizon tactical control toward long-horizon spatial commitment.

Tower Defense environments. Prior work on Tower Defense games is comparatively limited. Bergdahl et al. (2024) studied high-level strategic control in *Plants vs. Zombies*, combining learned decisions with scripted lower-level placement logic on handcrafted levels. Wang et al. (2026) later introduced a fixed-map Tower Defense benchmark for evaluating language-model agents. A separate line of work uses procedural content generation for offline Tower Defense level authoring rather than per-episode variation during learning (Liu et al., 2019; Dias et al., 2022). ProcTD instead generates new task instances procedurally across episodes, allowing benchmark difficulty to vary across maps, spatial scales, and strategic mechanics rather than being tied to a fixed set of handcrafted scenarios.

Procedural generation and generalization. Procedural variation is widely used in RL benchmarks to reduce the potential for memorizing fixed environments and to study performance across distributions of task instances. Procgen (Cobbe et al., 2020), Obstacle Tower (Juliani et al., 2019), NetHack (Küttler et al., 2020), and Crafter (Hafner, 2022) expose agents to procedurally generated levels or worlds with substantial variation between episodes. ProcTD follows this benchmark philosophy of per-episode variability, while additionally combining procedural maps with fog-of-war partial observability, long-horizon resource management, and configurable spatial scale.

[2] ProcTD Repository

3 THE PROCTD BENCHMARK

3.1 GAME AND ENVIRONMENT OVERVIEW

Tower Defense is a strategy game genre in which enemies traverse predefined routes toward a defended objective, while the player places and upgrades static defensive structures to stop them. The game progresses in waves, with each wave consisting of a group of enemies that enter the map and attempt to reach the defended objective.

ProcTD instantiates this setting as a partially observable Markov decision process (POMDP) (Kaelbling et al., 1998) on procedurally generated maps. The environment state comprises the map topology, constructed towers and their upgrades, active enemies, available resources, base health, wave progression, and current technology-card offers. Partial observability is introduced through a fog-of-war: only an initial segment of the map is revealed, with additional routes and spawn locations becoming visible as the episode progresses (Appendix A.4). This forces early defensive investments to be made under uncertainty.

3.2 OBSERVATION AND ACTION SPACES

The observations combine a spatial tensor with global game information. Channel-wide components like the map topology and tower positions scale with the size of the map; global components, like the current resources and offered technologies, are flat vectors. If selling is enabled, an extra channel carries the sell-value of each tile. For a default 16×16 map, the complete observation contains 3,896 values (Appendix A.1).

For a 16×16 map, the corresponding raw action space contains 1,540 discrete actions, of which only a state-dependent subset is valid at any given step. ProcTD exposes an action mask indicating currently valid actions, including affordability, visibility, occupancy, and technology-unlock constraints (Huang & Ontañón, 2022). Locked tower types retain fixed action indices but remain masked until their corresponding technology is unlocked. Similarly, sell actions are valid only for tiles currently occupied by towers. The environment does not apply this mask internally, allowing both masked and unmasked agents to be evaluated on the same underlying interface.

3.3 TOWERS

The agent constructs five tower classes with complementary roles: **Basic**, a cheap generalist; **Sniper**, a long-range specialist against armored and boss enemies; **AOE**, which counters swarms through splash damage; **Support**, which slows enemies without dealing damage; and **Economy**, which generates passive gold without participating in combat. Enemies differ in health, movement speed, and type, while towers differ in cost, damage, range, cooldown, and type-specific effectiveness. No single class is sufficient on its own, so the agent must compose a portfolio under a shared budget rather than scale one class.

Construction costs increase with the number of towers of the same class, while towers gain experience through combat and can receive additional manual upgrades. Tower–enemy damage multipliers create a rock-paper-scissors-style matchup structure that promotes compositional diversity. The full progression and list of mechanics and available technologies are given in Appendices A.2 and A.5. Together, these mechanics couple short-horizon spatial placement decisions with longer-horizon resource allocation and technology progression.

3.4 PROCEDURAL MAP GENERATION

Every episode samples a new map through a constrained multi-step generation procedure: starting from the defended base near the left edge, the generator constructs a self-avoiding route network with 2-4 branches terminating at the boundary enemy-spawn locations. Buildable terrain is restricted to randomized regions around the routes, while elevation, which increases the damage of towers, is biased toward path corners. This produces variation in route curvature, branch structure, chokepoints, elevation, and buildable terrain across episodes, requiring the agent to learn a *policy over topologies* rather than a fixed placement pattern. Full generation details and parameters are given in Appendix B.

3.5 Configurable Difficulty and Complexity

ProcTD exposes several independent controls for scaling task difficulty:

Enemy waves: ProcTD contains a sequence of fifty increasingly difficult enemy waves. Waves are generated from a growing point budget and consist of progressively unlocked Standard, Runner, and Tank enemies. Each wave has a primary enemy type that is sampled more frequently, giving waves distinct compositions, and every tenth wave is replaced by a single boss enemy. The sequence of wave compositions and spawn cadence is fixed across episodes for a given seed. The release order of the budgeted units and their revealed entry point are sampled per episode. A global health multiplier uniformly scales enemy and boss health, providing controllable difficulty that is independent of map size and wave composition. Full wave-generation details are given in Appendix A.3.

Map scale: The complexity of the environment is dependent on the used map size. ProcTD defines 16×16, 32×32, and 64×64 map tiers, but the scale is theoretically unbounded. As map dimensions increase, both the spatial observation size and the number of spatial actions grow quadratically with map size, but the underlying game mechanics remain unchanged. The 16×16 map should be regarded as a screening tier for fast experimentation, the 32×32 map is the primary research size, and the 64×64 map is provided to future-proof the benchmark.

Tower selling: If tower selling is enabled, the agent gains an additional sell action for each map tile: selling a tower returns a fraction of the total gold invested in it (0.9 as default), including both its construction cost and all manual upgrade payments, while forfeiting accumulated XP and levels and freeing the tile for rebuilding. This makes earlier placement decisions partially reversible and allows the agent to reallocate resources and adapt its defensive composition as enemy matchups change.

3.6 Implementation and Computational Efficiency

ProcTD is designed for high-throughput simulation. Game state is stored in flat, fixed-capacity NumPy arrays containing up to 256 towers and 2,048 enemies, following a data-oriented design (Fabian, 2018; Harris et al., 2020). Core simulation operations, including enemy movement, targeting, cooldowns, damage, status effects, rewards, and wave progression, are executed inside a Numba JIT-compiled kernel (Lam et al., 2015) that updates the state in place.

ProcTD offers fast throughput because training does not require graphical rendering that is common in many game-based benchmarks. On a single CPU core, it averages approximately 10,000 environment steps per second, with throughput remaining largely stable across map sizes. Furthermore, ProcTD is fully compatible with the Gymnasium API (Towers et al., 2025) and is distributed as a pip-installable Python package, making it straightforward to integrate with existing RL tooling. Further engineering details are given in Appendix C.

3.7 Multi-Agent Extension

ProcTD provides an asymmetric competitive attacker–defender extension, formulated as a general-sum Markov game (Littman, 1994) and implemented through the PettingZoo API (Terry et al., 2021). In this setting, the fixed enemy-wave schedule of the single-agent environment is replaced by an attacker agent. The defender's observation, action space, masking, and reward remain identical to the single-agent setting; only the source of the enemy schedule changes. The game is asymmetric in both observations and actions. The attacker controls wave construction by specifying a sequence of enemy type and spawn-point choices, bounded by a per-wave cap on the number of units. It cannot directly observe the defender's towers and instead receives a persistent spatial "death grid" summarizing where its units were destroyed, which the adaptive and learned attackers use to recompose and reroute later waves. Together with the defender's fog-of-war observation, this creates asymmetric partial information on both sides and allows the benchmark to study adaptive offense and defense rather than only performance against a fixed enemy schedule. Full details of the attacker interface, observations, rewards, and scripted baseline attackers are given in Appendix D.

4 Experimental Setup

4.1 Benchmark Configuration and Metrics

We evaluate ProcTD at map sizes 16×16 and 32×32; an additional 64×64 tier is supplied in the environment. Unless otherwise stated, experiments use the default configuration with 200 starting gold and tower selling disabled (Appendix A.7). Models are trained with a shaped dense reward combining combat and base health, wave progression, and terminal signals (Appendix A.6). We additionally evaluate the selling mechanism in Appendix F.3 and other reward formulations, including the standard reward function, in Appendix F.2.

Agents are evaluated over 100 unseen seeded episodes, and we report the mean ($\pm$ standard deviation) and median (IQR) of the episode reward and the number of waves survived. We define an environment as *solved* when the agent reaches a median of 50 waves survived with an IQR of 0, i.e., when it consistently clears all waves. A validation of the sensitivity to the training seed is reported in Appendix F.4.

4.2 Macro-Action Policy with Symbolic Placement

In ProcTD, end-to-end learning must jointly discover strategic decisions and spatial placement over a large, geometrically structured action space. In practice, an agent may spend millions of interaction steps discovering simple spatial structure that human players typically handle through placement heuristics, such as placing defensive towers near enemy paths, instead of focusing on higher-level decisions such as resource allocation and tower composition. We therefore implement Macro-Action Policy with Symbolic Placement (MAPS), a two-stage action interface that separates high-level strategic decisions from low-level spatial placement. This decomposition follows the broader idea of hierarchical and macro-action abstraction in reinforcement learning (Sutton et al., 1999; Vezhnevets et al., 2017; Wu, 2019; Hausknecht & Stone, 2016).

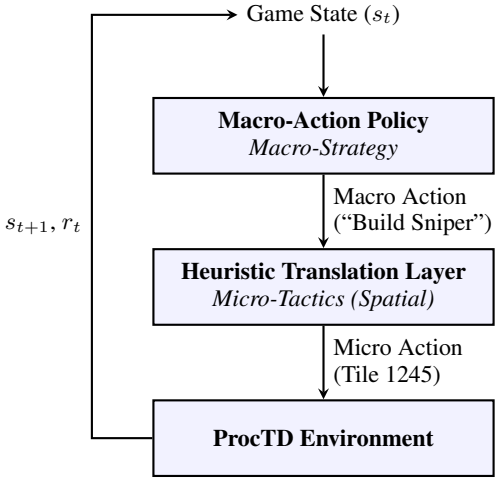


Figure 2: **The MAPS action interface.** A macro-action policy selects a high-level action, while a deterministic heuristic translates build actions into spatially grounded micro-actions. The macro-action policy can be learned, as in MAPS-PPO, or sampled uniformly, as in MAPS-RND.

The macro-action space consists of `PICK CARD 1–3`, `WAIT`, `BUILD` $\times$ 5 tower types, and one `UPGRADE` action for each map tile. For `BUILD` actions, the agent selects only the tower type, while a deterministic heuristic chooses the placement location; `UPGRADE` actions remain spatially indexed. On a 16×16 map, this reduces the action space from 1,540 to 265 actions, a reduction of more than 80%. The placement heuristic scores candidate tiles based on expected damage-per-second (DPS) for the selected tower type, 5×5 Chebyshev chokepoint coverage, elevation, distance, Economy dead-zone placement, and early-game survival rules. The exact scoring procedure is given in Appendix E.1.

4.3 Baseline Agents

We compare seven baselines spanning random, heuristic, and learned policies. **Random** uniformly samples from valid raw actions, while **Simple Heuristic** places Basic towers near the base using a simple spatial rule and **Advanced Heuristic** uses hand-crafted rules for tower composition, placement, and upgrades. **PPO** trains PPO directly on the raw action space with invalid-action masking, while **Unmasked PPO** does not use action masking. **MAPS-RND** and **MAPS-PPO** instead use the MAPS interface introduced in Section 4.2: MAPS-RND samples uniformly from valid macro actions, whereas MAPS-PPO learns a PPO policy over the same macro-action space. Full baseline and architecture details are provided in Appendix E.2 and additional results with value-based **DQN**

(Mnih et al., 2015) and model-based **Dreamerv3** (Hafner et al., 2025) are given in Appendix F.1. Models are trained with one seed; an ablation of seed-sensitivity is given in Appendix F.4.

The PPO-based agents share the same encoder architecture, with a convolutional stack over the spatial observation and small MLPs over the global features, followed by separate actor and critic heads. The convolutional capacity is increased for larger maps, and agents are trained for 100M environment steps. Because the agents differ primarily in their action interface, this shared architecture provides a controlled comparison of the effect of action-space abstraction. The model architecture and training protocols of PPO-based agents are specified in Appendix E.3.

4.4 Competitive Mode Setup

In the competitive mode (Section 3.7), we evaluate the trained, frozen defenders against three attacker baselines: a random scripted attacker, an adaptive heuristic attacker, and a learned PPO attacker. Note that the learned attacker is trained specifically against a frozen defender, thus we characterize defender-specific vulnerability rather than equilibrium performance. The random and heuristic attackers differ only in how they choose wave composition, while the learned attacker selects wave composition, spawn point, and release order through a factored action space. All competitive evaluations use 100 seeded 16×16 episodes. Full details of the attacker action space, observations, scripted policies, and PPO training setup are given in Appendix D.

5 Results and Discussion

Spatial structure substantially improves performance. Table 1 shows a clear separation between unstructured, heuristic, and learned baselines. Random and Simple Heuristic perform similarly, whereas Advanced Heuristic and MAPS-RND improve substantially when structured placement is introduced. PPO further improves over the non-learned baselines on the 16×16 environment, while MAPS-PPO achieves the strongest overall performance and consistently reaches the full episode horizon.

Table 2 separates the contributions of the learned macro-policy and the spatial placement heuristic. Replacing heuristic placement with random valid placement causes the largest drop in performance; replacing the learned macro-policy with random valid macro-actions also substantially reduces survival. The placement heuristic therefore provides the larger contribution, while the learned macro-policy adds a substantial complementary gain.

Table 1: Single-agent baseline performance on one training seed. Values report mean ($\pm$ std) and median (IQR) for reward and waves survived; † denotes agents without invalid-action masking.

Size	Agent	Mean Reward	Median Reward	Mean Waves	Median Waves
16×16	Random	$-51.8^{\pm 55.2}$	$-62.1^{(36.3)}$	$9.3^{\pm 6.7}$	$7.0^{(6.3)}$
	Simple Heuristic	$-60.9^{\pm 22.9}$	$-54.3^{(15.8)}$	$7.9^{\pm 1.2}$	$8.0^{(2.0)}$
	Advanced Heuristic	$41.8^{\pm 50.2}$	$45.0^{(68.7)}$	$20.6^{\pm 2.5}$	$21.0^{(1.0)}$
	MAPS-RND	$116.4^{\pm 163.1}$	$96.7^{(169.7)}$	$24.8^{\pm 9.5}$	$27.0^{(10.0)}$
	Unmasked PPO†	$-84.7^{\pm 22.9}$	$-80.4^{(17.8)}$	$5.7^{\pm 3.1}$	$5.0^{(2.0)}$
	PPO	$622.5^{\pm 459.7}$	$541.3^{(919.8)}$	$38.1^{\pm 12.5}$	$42.0^{(23.0)}$
	MAPS-PPO	$\mathbf{964.9^{\pm 347.6}}$	$\mathbf{1116.9^{(36.4)}}$	$\mathbf{46.1^{\pm 9.7}}$	$\mathbf{50.0^{(0.0)}}$
32×32	Random	$-58.1^{\pm 42.6}$	$-69.0^{(35.9)}$	$8.3^{\pm 4.9}$	$7.0^{(5.0)}$
	Simple Heuristic	$-59.9^{\pm 25.1}$	$-54.1^{(32.2)}$	$8.5^{\pm 1.1}$	$9.0^{(1.0)}$
	Advanced Heuristic	$32.7^{\pm 49.9}$	$41.9^{(55.7)}$	$19.7^{\pm 3.0}$	$20.0^{(4.0)}$
	MAPS-RND	$116.4^{\pm 130.6}$	$87.7^{(115.5)}$	$25.4^{\pm 7.8}$	$27.0^{(9.0)}$
	Unmasked PPO†	–	–	–	–
	PPO	$255.7^{\pm 257.8}$	$195.5^{(187.4)}$	$28.3^{\pm 10.2}$	$27.0^{(11.0)}$
	MAPS-PPO	$\mathbf{844.6^{\pm 440.2}}$	$\mathbf{1116.9^{(644.4)}}$	$\mathbf{42.7^{\pm 13.0}}$	$\mathbf{50.0^{(9.0)}}$

Raw-action PPO degrades sharply with spatial scale. Increasing the map size from 16×16 to 32×32 substantially reduces PPO's performance, while the heuristic and MAPS-based baselines are affected much less strongly. MAPS-PPO retains a median survival of the full 50-wave horizon, although its increased dispersion does not solve the larger environment under our criterion. The substantially larger degradation of PPO is consistent with spatial decision-making becoming a more difficult learning problem as map size grows. In particular, the raw action space increases from 1,540 actions on 16×16 maps to 6,148 on 32×32, whereas the MAPS interface grows from only 265 to 1,033 actions (Table A2). On 16×16, MAPS-PPO improves rapidly and plateaus close to the maximum performance, while PPO improves more gradually (Figure 3). With larger 32×32 maps, MAPS-PPO still reaches high survival relatively early, while PPO learns more slowly , but neither method converge after 100M steps.

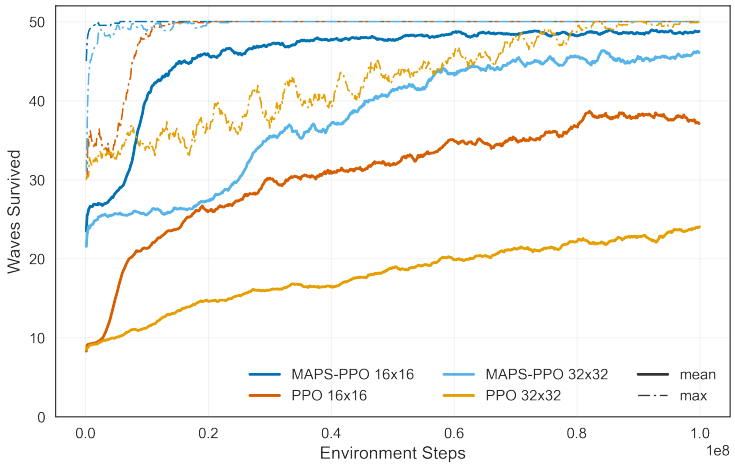


Figure 3: Learning curves on 16×16 vs. 32×32 for MAPS-PPO and PPO. PPO degrades sharply under the quadrupled placement space; MAPS-PPO remains stable.

ProcTD leaves substantial headroom for general-purpose RL. Together, the two map sizes place ProcTD in a useful benchmark regime. General-purpose PPO performs well on the smaller environment but degrades markedly as spatial scale increases, while MAPS-PPO shows that near-maximal performance remains attainable when substantial spatial structure is provided. This further motivates the spatial variability of ProcTD, i.e., increasing the map size to 64×64 and beyond allows future-proofing for more advanced methods. The remaining gap motivates methods that can learn effective spatial and strategic abstractions from the raw interface rather than relying on handcrafted placement rules.

Table 2: MAPS component ablation on 16×16, crossing the macro-policy with spatial placement. Values report mean waves survived, with median in parentheses.

	Heuristic Placement	**Random Placement**
MAPS-PPO	46.1 (50)	10.3 (6)
MAPS-RND	25.5 (27)	7.4 (6)

Table 3: Competitive-mode evaluation of single-agent defenders against scripted and learned attackers. "Att. Win" denotes the fraction of episodes in which the attacker destroys the base before Wave 50, and "Min" the minimum number of waves survived across the evaluation episodes.

Defender	**Attacker**	**Waves**	**Att. Win**	**Min**
MAPS-PPO	Random	$44.5^{\pm 14.1}$	14%	1
	Adaptive	$28.3^{\pm 15.7}$	72%	1
	Learned	$15.0^{\pm 9.5}$	96%	1
PPO	Random	$44.9^{\pm 9.8}$	26%	12
	Adaptive	$19.5^{\pm 7.9}$	98%	7
	Learned	$16.2^{\pm 4.7}$	100%	6

Invalid-action masking is critical on the raw interface. Unmasked PPO makes little progress even on the 16×16 environment, showing that learning action validity simultaneously with the task policy substantially increases the difficulty of the raw interface. Additional experiments with unmasked DQN and DreamerV3 show similarly poor performance and are reported in Appendix F.1.

Competitive attackers expose brittle defenders. Table 3 shows that both defenders transfer relatively well to a random attacker but degrade substantially against adaptive and learned ones. The learned attacker wins >95% of episodes against both agents, so neither fully-learned nor heuristic placement yields a robust defense. MAPS-PPO suffers from extremely poor worst-case performance, with all attackers achieving at least one win in the first wave. This indicates that fixed placement heuristics are prone to exploitation. PPO, despite achieving a higher performance floor, is unable to learn a transferable policy still.

6 Conclusion

ProcTD is a lightweight, high-throughput Tower Defense benchmark that combines long-horizon resource management with spatial decision-making under partial observability and procedural variation. Our experiments show that general-purpose PPO can achieve strong performance on the smaller 16×16 setting, but degrades substantially as spatial scale and action-space size increase. MAPS-PPO, which incorporates a handcrafted spatial placement prior, demonstrates that high performance remains attainable on the larger setting when appropriate structure is provided. This gap leaves substantial headroom for methods that can learn effective spatial and strategic abstractions directly from the raw interface. ProcTD additionally supports masked and unmasked evaluation, configurable game mechanics, and an asymmetric attacker–defender mode in which adaptive opponents expose vulnerabilities not apparent against the fixed single-agent wave generator. We release the environment and baseline implementations as a testbed for reinforcement learning in large, structured, spatially grounded action spaces.

7 Limitations and Future Work

The work in this paper has several limitations. First, MAPS-PPO relies on a handcrafted placement heuristic, so its performance is bounded by the quality of that prior. Second, our main experiments focus on 16×16 and 32×32 maps; the provided 64×64 tier remains largely unexplored. Third, procedural generation can occasionally produce unusually difficult or possibly unwinnable layouts. Finally, the competitive experiments train attackers against frozen defenders, so they measure defender-specific vulnerability rather than equilibrium play.

Several extensions follow naturally from these limitations. A central algorithmic direction is to replace the handcrafted placement heuristic with a learned or differentiable spatial module. Future benchmark extensions could include larger and more varied maps, randomized spawn and base locations, richer path generation, and an endless mode. Additional game mechanics such as targeting priorities, richer tower upgrades, damage-over-time effects, adjacency synergies, and path-blocking towers could further increase strategic depth. For the competitive setting, attacker populations, alternating or simultaneous training of attackers and defenders, and denser attacker reward signals would allow study of co-adaptation rather than fixed-opponent exploitation.

AI Use Statement

Large Language Models (LLMs) were used in the preparation of this manuscript for writing and structural assistance, retrieval and discovery, drafting, and minor implementation support. They assisted with LaTeX formatting of tables and figures, suggesting alternative structures for individual sections, identifying potential related work, and producing preliminary drafts of some passages. All identified references were retrieved and verified by the authors before citation, and all AI-generated code was checked by the authors. The benchmark design, experimental design, analysis, and interpretation of results are the authors' own. The authors verified all text, code, and results and take full responsibility for the content of this paper.

Ethics Statement

ProcTD is a simulated reinforcement-learning benchmark and does not involve human participants, personal data, or deployment in real-world decision-making. The environment is intended for research on sequential decision-making, spatial reasoning, and multi-agent learning. We do not identify specific foreseeable societal harms beyond those associated with general reinforcement-learning research.

Reproducibility Statement

We provide the full ProcTD source code, baseline models, experiment configurations, and evaluation scripts required to reproduce the results reported in this paper in an anonymized repository: https://anonymous.4open.science/r/ProcTD/.

References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A., and Bellemare, M. G. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL.

Avery, P., Togelius, J., Alistar, E., and van Leeuwen, R. P. Computational intelligence and tower defence games. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pp. 1084–1091, 2011. 10.1109/CEC.2011.5949738.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. 10.1613/jair.3912.

Bergdahl, J., Sestini, A., and Gisslén, L. Reinforcement learning for high-level strategic control in tower defense games. In *2024 IEEE Conference on Games (CoG)*, pp. 343–350, 2024. 10.1109/CoG60054.2024.10645621.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pinto, H. P. d. O., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. 10.48550/arXiv.1912.06680.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2048–2056. PMLR, 2020. URL.

Colas, C., Sigaud, O., and Oudeyer, P.-Y. How many random seeds? statistical power analysis in deep reinforcement learning experiments. *arXiv preprint arXiv:1806.08295*, 2018. 10.48550/arXiv.1806.08295.

Demšar, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. URL.

Dias, D. M. P. L., Sousa, J. P. P., Barroso, B. C. V. B., and Magalhães, I. M. B. A novel procedural content generation algorithm for tower defense games. In *Proceedings of the 6th International Conference on Algorithms, Computing and Systems*, pp. 1–7, 2022. 10.1145/3564982.3564993.

Die of Death Games. Rogue tower. Video game, 2022. URL. Released 28 January 2022.

Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., and Coppin, B. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015. 10.48550/arXiv.1512.07679.

Fabian, R. *Data-Oriented Design: Software Engineering for Limited Resources and Short Schedules*. Independently published, 2018. ISBN 978-1916478701. URL.

Hafner, D. Benchmarking the spectrum of agent capabilities. In *International Conference on Learning Representations*, 2022. URL.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025. 10.1038/s41586-025-08744-2.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature*, 585:357–362, 2020. 10.1038/s41586-020-2649-2.

Hausknecht, M. and Stone, P. Deep reinforcement learning in parameterized action space. In *International Conference on Learning Representations*, 2016. URL.

Huang, S. and Ontañón, S. A closer look at invalid action masking in policy gradient algorithms. In *Proceedings of the International Florida Artificial Intelligence Research Society Conference*, volume 35, 2022. 10.32473/flairs.v35i.130584.

Huang, S., Ontañón, S., Bamford, C., and Grela, L. Gym-μrts: Toward affordable full game real-time strategy games research with deep reinforcement learning. In *2021 IEEE Conference on Games (CoG)*, pp. 1–8. IEEE, 2021. 10.1109/CoG52621.2021.9619076.

Juliani, A., Khalifa, A., Berges, V.-P., Harper, J., Teng, E., Henry, H., Crespi, A., Togelius, J., and Lange, D. Obstacle tower: A generalization challenge in vision, control, and planning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pp. 2684–2691, 2019. 10.24963/ijcai.2019/373.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998. 10.1016/S0004-3702(98)00023-X.

Küttler, H., Nardelli, N., Miller, A., Raileanu, R., Selvatici, M., Grefenstette, E., and Rocktäschel, T. The NetHack learning environment. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL.

Lam, S. K., Pitrou, A., and Seibert, S. Numba: A LLVM-based python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 1–6, 2015. 10.1145/2833157.2833162.

Littman, M. L. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pp. 157–163, 1994. 10.1016/B978-1-55860-335-6.50027-1.

Liu, S., Li, C., Li, Y., Ma, H., Hou, X., Shen, Y., Wang, L., Chen, Z., Guo, X., Lu, H., Du, Y., and Tang, Q. Automatic generation of tower defense levels using PCG. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, pp. 1–9, 2019. 10.1145/3337722.3337723.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 10.1038/nature14236.

Ng, A. Y., Harada, D., and Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning*, pp. 278–287, 1999.

Ontañón, S., Barriga, N. A., Silva, C. R., Moraes, R. O., and Lelis, L. H. S. The first microRTS artificial intelligence competition. *AI Magazine*, 39(1), 2018. 10.1609/aimag.v39i1.2777.

Perez-Liebana, D., Hsu, Y.-J., Emmanouilidis, S., Khaleque, B. D. A., and Gaina, R. D. Tribes: A new turn-based strategy game for AI research. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pp. 252–258, 2020. 10.1609/aiide.v16i1.7438.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021. URL.

Samvelyan, M., Rashid, T., Schroeder de Witt, C., Farquhar, G., Nardelli, N., Rudner, T. G. J., Hung, C.-M., Torr, P. H. S., Foerster, J., and Whiteson, S. The starcraft multi-agent challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2186–2188, 2019. URL.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 10.48550/arXiv.1707.06347.

Suarez, J., Bloomin, D., Choe, K. W., Li, H. X., Sullivan, R., Kanna, N., Scott, D., Shuman, R., Bradley, H., Castricato, L., Isola, P., Yu, C., Jiang, Y., Li, Q., Chen, J., and Zhu, X. Neural mmo 2.0: A massively multi-task addition to massively multi-agent learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 50094–50104, 2023. 10.52202/075280-2178.

Sutton, R. S., Precup, D., and Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999. 10.1016/S0004-3702(99)00052-1.

Terry, J. K., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., Santos, L. S., Dieffendahl, C., Horsch, C., Perez-Vicente, R., Williams, N. L., Lokesh, Y., and Ravi, P. PettingZoo: Gym for multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 15032–15043, 2021. URL.

Towers, M., Kwiatkowski, A., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., Perez-Vicente, R., Terry, J. K., Pierré, A., Schulhoff, S., Tai, J. J., Tan, H., and Younis, O. G. Gymnasium: A standard interface for reinforcement learning environments. In *Advances in Neural Information Processing Systems*, volume 38, pp. 163114–163129, 2025. 10.52202/085713-4916.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. FeUdal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3540–3549. PMLR, 2017. URL.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J., and Tsing, R. StarCraft II: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017. 10.48550/arXiv.1708.04782.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575:350–354, 2019. 10.1038/s41586-019-1724-z.

Wang, D., Zhou, C., Zhao, D., Liu, X., Ma, M. C., Ushaw, G., and Davison, R. TowerMind: A tower defence game learning environment and benchmark for LLM as agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 26151–26159, 2026. 10.1609/aaai.v40i31.39818.

Wu, B. Hierarchical macro strategy model for MOBA game AI. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1206–1213, 2019. 10.1609/aaai.v33i01.33011206.

APPENDIX

A Environment Specification **15**
A.1 Observation and Action Spaces 15
A.2 Tower Mechanics and Progression 15
A.3 Enemy Waves and Spawn Routing 17
A.4 Fog-of-War 18
A.5 Technology Drafting 18
A.6 Reward Design 19
A.7 Default Environment Configuration 20

B Procedural Map Generation **21**
B.1 Map Construction 21

C Implementation and Computational Details **22**
C.1 Simulation Engine 22
C.2 Gymnasium Interface and Execution Details 22
C.3 Throughput 22

D Competitive Multi-Agent Extension **24**
D.1 Scripted attackers 24
D.2 Learned attackers 25

E Baselines and Training Details **27**
E.1 Spatial Placement Heuristic 27
E.2 Baseline Agents 28
E.3 Learned-Agent Details 29

F Extended Results and Analysis **31**
F.1 Unmasked Models' Results 31
F.2 Reward Shaping 31
F.3 Tower Selling 32
F.4 Training Seed Sensitivity 33

G Statistical and Distributional Analysis **34**

A Environment Specification

A.1 Observation and Action Spaces

Table A1 defines the structure of ProcTD's observation and action spaces for a map of height H and width W, while Table A2 reports the resulting total observation and action-space sizes for the 16×16, 32×32, and 64×64 tiers, with and without tower selling and under the MAPS interface.

Table A1: Observation and raw action spaces for a map of height H and width W. Spatial channels are grouped by role. Selling adds one additional spatial channel and HW sell actions.

Component	Description
Observation – spatial tensor($15HW$ w/o sell and $16HW$ w/ sell)	
Topology (ch. 0–5)	path, buildable, height, fog-reveal, base, active spawns
Towers (ch. 6–8)	type (/5), level (/20), manual-upgrade tier (/5)
Enemies (ch. 9–14)	presence, accumulated HP (/100), per-type Std/Runner/Tank/Boss
Sell (optional, ch. 15)	current cell value per tile (log (1 + value)/8)
Observation – global vectors: 56 values	
Resources	11 values: gold, base HP, wave, enemy count, flags, 5 tower costs
Upgrades	21 drafted-card stack counts
Offered cards	21-value multi-hot offer mask
Card slots	3 raw slot IDs
Raw action space ($6HW + 4$ w/o sell and $7HW + 4$ w/ sell)	
Build	$5HW$ actions
Upgrade	HW actions
Wait	1 action
Draft	3 actions
Sell (optional)	HW additional actions

Table A2: Total observation and action-space sizes for the three ProcTD map tiers, with and without tower selling.

	Observation		Raw actions		Macro actions	
Map size	No sell	Sell	No sell	Sell	No sell	Sell
16×16	3,896	4,152	1,540	1,796	265	521
32×32	15,416	16,440	6,148	7,172	1,033	2,057
64×64	61,496	65,592	24,580	28,676	4,105	8,201

A.2 Tower Mechanics and Progression

The five tower classes (Table A3) have complementary roles:

- **Basic**: generalist tower but penalized against Tanks and Bosses.
- **Sniper**: specializes against Tanks and Bosses but is weak against Runners.
- **AOE**: anti-swarm tower with splash damage within 1.5 tiles but is weak against Tanks and Bosses.
- **Support**: slows enemies without dealing damage.
- **Economy**: generates passive gold without participating in combat.

The construction cost of a tower type increases geometrically with the number N of already constructed towers of that type as

$$\text{Cost} = \max\left(5, \left\lceil 1.25^N \, \text{Cost}_{\text{base}} - 5D \right\rceil\right), \tag{1}$$

where D is the number of drafted Discount cards (Appendix A.5).

Towers gain XP only for effective HP removed, excluding overkill: 0.50 XP per point of single-target damage, 0.25 XP per point of splash damage, and 1.5 XP per slowed enemy for Support, capped at 12 XP per tick. For a tower at level L, where $L \in \{1, \ldots, 20\}$, the XP required to reach the next level is

$$\mathrm{XP}_{\mathrm{req}} = \mathrm{XP}_{\mathrm{base}}\, M_{\mathrm{FL}}\, 1.5^{L-1}, \tag{2}$$

where $\mathrm{XP}_{\mathrm{base}}$ is the tower-specific base XP requirement from Table A3, and M_{FL} is the multiplier introduced by the Fast Learner technology (Appendix A.5). The damage multiplier associated with tower level is

$$M_{\mathrm{level}} = 1 + 0.15(L - 1). \tag{3}$$

Each tower can additionally receive up to five manual upgrades. At upgrade level U, the upgrade cost is $\mathrm{Cost}_{\mathrm{upgrade}} = \mathrm{Cost}_{\mathrm{tower}}\, 1.5^{U}$, and the resulting damage multiplier is $1 + 0.75U$.

For damaging towers, the final damage is

$$\mathrm{Dmg} = \mathrm{Dmg}_{\mathrm{base}}\, M_{\mathrm{level}}\, (1 + 0.75U)\, M_{\mathrm{height}}\, M_{\mathrm{matchup}}, \tag{4}$$

where $M_{\mathrm{height}} = 1.25^{h}$; and $\mathrm{Dmg}_{\mathrm{base}}$ and M_{matchup} are given in Table A3. Support towers deal no damage and instead halve the movement speed of every enemy in range for $15 + 5S$ ticks, where S is the number of drafted Support Duration cards, scaled by $(1 + 0.75U)$ and M_{level}; the level multiplier additionally scales the Support tower's range. Economy income scales as $17.5 + 3.0U$ gold per cleared wave.

Targeting is deterministic: damage towers hit the in-range enemy with the greatest path progress. AOE additionally splashes damage and Support slows all enemies in range.

Table A3: Base statistics for the five tower classes and tower–enemy matchup multipliers. Statistics are shown before level, manual-upgrade, terrain, and matchup modifiers. Multipliers above 1 indicate favorable matchups and values below 1 unfavorable matchups. Support applies a slowing effect instead of damage, while Economy generates passive gold.

	Base statistics					**Matchup multiplier**			
Tower	Cost	Dmg	Range	Cooldn	XP req	Std	Runner	Tank	Boss
Basic	20	2.5	3.0	10	300	1.00	1.00	0.25	0.50
Sniper	40	15.0	7.0	30	200	1.00	0.25	3.00	2.50
AOE (Area of Effect)	50	1.5	2.5	20	600	1.00	3.00	0.10	0.25
Support	30	0.0	3.0	15	250		slow (no damage)		
Economy	150	0.0	–	–	–		gold (no damage)		

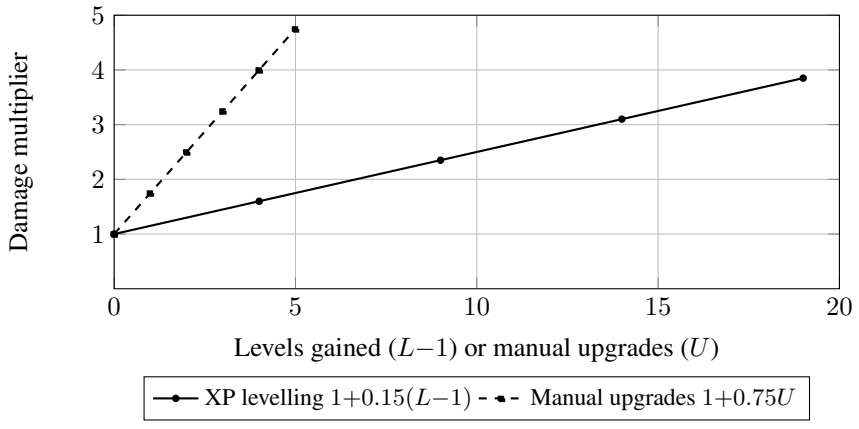


Figure A1: Tower damage multipliers. Levels are earned automatically through combat XP and are capped at 20; manual upgrades are bought with gold and are capped at $U = 5$, so they reach a comparable multiplier in far fewer steps.



Figure A2: Construction cost scales geometrically as 1.25^N in the number of towers of the same type already on the map, so the ninth tower of a type costs roughly six times the first.

Tower selling. When tower selling is enabled, a constructed tower can be removed in exchange for a fraction of the gold invested in it. By default, the refund is 90% of the combined construction cost and all manual-upgrade costs. Selling forfeits the tower's accumulated XP and levels and frees the tile for subsequent construction. The effect of tower selling is studied in Appendix F.3.

A.3 Enemy Waves and Spawn Routing

The environment contains 50 waves indexed by $w \in \{1, \ldots, 50\}$, with every 10th wave being a boss-wave.

Normal waves. Non-boss wave composition is generated from the point budget

$$\text{Budget}_w = \left\lfloor 20 + 3.5w + 0.35w^2 \right\rfloor . \tag{5}$$

The available enemy archetypes, their unlock waves, budget costs, and stat multipliers are summarized in Table A4. For each wave, one currently available enemy type is selected as the primary type and receives sampling weight 3 with weight 1 for the other available types. This produces waves with distinct compositions while retaining variation within each wave. The spawn interval decreases with wave number as $\max(5, 30 - w)$ ticks.

Table A4: Enemy characteristics.

Archetype	Available from	Budget cost	HP ×	Speed ×	Bounty ×	Reward×
Standard	Wave 1	3	1.0	1.0	1.0	1.0
Runner	Wave 5	2	0.5	1.5	0.5	0.5
Tank	Wave 8	5	3.0	0.5	3.0	2.5
Boss	Every 10th wave	–	Eq. (8)	0.35	Eq. (8)	25.0

Before applying archetype-specific modifiers, enemy health at wave w is

$$\text{HP}_w = \left(20 + 3.5w + 0.2w^3\right) M_{\text{HP}}, \tag{6}$$

where M_{HP} is the global enemy-health multiplier, with default value 1. The base movement speed is 0.05 path-units per tick, and the base kill bounty is

$$\max\left(1, \left\lfloor 1 + \frac{w}{3.5} \right\rfloor\right) \tag{7}$$

gold.

The enemy archetype-specific health, speed, and bounty multipliers are summarized in Table A4.

Boss waves. Every tenth wave ($w \bmod 10 = 0$) replaces the regular budget-generated wave with a single Boss enemy. Its health and bounty are

$$\mathrm{HP}_w^{\mathrm{boss}} = \mathrm{HP}_w \cdot 7.25 \left\lfloor \frac{w}{10} \right\rfloor, \qquad \mathrm{Bounty}_w^{\mathrm{boss}} = 40 + 25 \left\lfloor \frac{w}{10} \right\rfloor, \tag{8}$$

with a spawn interval of 10 ticks.

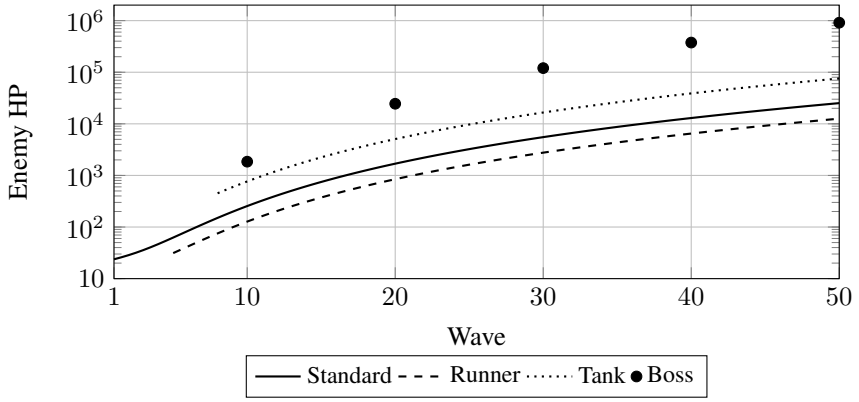


Figure A3: Enemy health across the 50-wave horizon with marks starting from the first possible occurrence. Boss waves introduce discrete high-HP spikes every tenth wave.

A.4 Fog-of-War

Partial observability is enforced through incremental revelation of the generated route network, with the reveal schedule scaled to map size. At generation, the first

$$\min\big(6(W+H)/32,\ |\mathrm{path}|\big)$$

path tiles are marked visible, and `reset` immediately reveals one additional segment of

$$\max\big(3,\ 3(W+H)/32\big)$$

tiles. The agent therefore begins each episode with a short visible frontier of 9, 18, and 36 path tiles on the 16×16, 32×32, and 64×64 tiers, respectively. Surrounding terrain is revealed within a fixed Chebyshev radius of 3 at all map sizes.

After each successful draft transition, the next segment of $\max\big(3,\ 3(W+H)/32\big)$ path tiles is revealed and the active spawn frontier is recalculated farther from the base. This progressively pushes enemy pressure outward toward the true branch endpoints as the episode advances. Because both the initial visible window and the per-draft reveal increment scale linearly with $W + H$, the fraction of the route visible at a given stage remains approximately consistent across map tiers rather than making larger maps proportionally more hidden.

Spawn routing. Enemies are spawned from the currently active visible frontier of the route network, rather than necessarily from the final map-boundary endpoints. For each spawn, an active location is sampled uniformly, while the enemy archetype is drawn from the remaining wave composition according to the remaining counts. As fog-of-war recedes and additional route segments are revealed, the active spawn frontier is recalculated farther from the base. This progressively exposes additional branches and requires the agent to defend multiple overlapping routes and chokepoints.

A.5 Technology Drafting

Between waves, the simulation pauses and the agent enters a technology-drafting phase. At each draft, three cards are sampled from a pool of 21 available technologies, and the agent selects one of the three offered cards. Table A5 summarizes the implemented technology cards and their effects. Defeating a boss grants additional draft opportunities, with the number of extra drafts increasing with the boss index: one additional draft after Wave 10, two after Wave 20, etc.

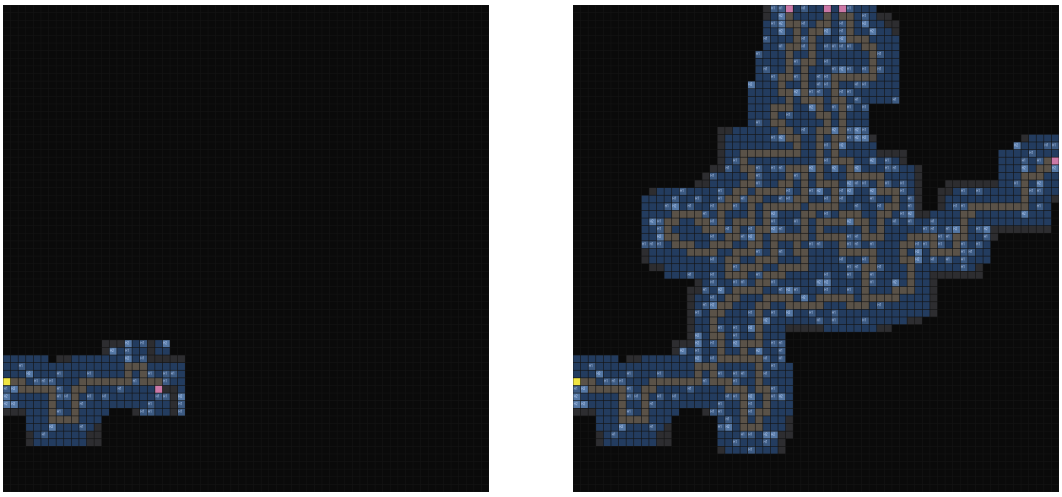


Figure A4: ProcTD fog-of-war. **Left:** the initially revealed segment at Wave 1. **Right:** the fully revealed topology with branched spawn routes, elevation, and constrained buildable terrain. The agent commits early defenses without seeing the full map.

Several tower classes are gated by one-time unlock cards. AOE is guaranteed to become available after Wave 1, Sniper after Wave 4, and Support after Wave 7, while Economy remains dependent on the sampled card offers. Once a one-time unlock has been acquired, it is removed from the pool of future eligible offers. Remaining offer slots are sampled from the currently valid card pool.

Table A5: Technology cards available in ProcTD and their effects. Stackable cards apply once per drafted copy.

Card	**Effect**
Unlock Sniper/AOE/Support/Economy	Marks that tower's build actions valid in the mask
Basic/Sniper Damage	$+20\%$ damage per stack
Basic/Sniper Range	$+1$ range per stack
AOE Splash	$+0.5$ splash radius per stack
Support Duration	$+5$ slow ticks per stack
Height Advantage	Height multiplier $\rightarrow (1.25 + 0.10S)^h$
Fast Learner	XP requirements $\times 0.8$ per stack
Overclock	Tower cooldowns $\times 0.9$ per stack
Armor Piercing	$+50\%$ vs. Tanks per stack
Crowd Control	$+50\%$ vs. Runners per stack
Bounty Hunter	$+20\%$ kill bounty per stack
Discount	-5 gold build cost per stack (min 5)
Gold Mine	$+25$ gold after each cleared wave
Base Repair	Restore 25 base HP (capped)
Base Health	$+10$ to both maximum and current base HP
Stimulus Package	$+150$ gold immediately

A.6 Reward Design

The dense reward, used in our experiments, combines combat performance, base-health preservation, wave progression, terminal outcomes, and action-validity signals. At environment step t, the reward is

$$\begin{aligned} r_t = {} & 0.05\, r_{\text{combat}} - 4.0\, \text{HP}_{\text{lost}} + (5.0 + 0.2w)\, \mathbb{I}_{\text{wave_cleared}} \\ & + 250.0\, \mathbb{I}_{\text{victory}} - 40.0\, \mathbb{I}_{\text{game_over}} + 0.1\, \mathbb{I}_{\text{valid_draft}} - 100.0\, \mathbb{I}_{\text{invalid_draft}} \\ & - 0.1\, \mathbb{I}_{\text{invalid_action}} + \lambda_{\text{idle}}\, \mathbb{I}_{\text{idle}} \end{aligned} \tag{9}$$

where w is the current wave index; r_{combat} is the sum of the combat rewards associated with enemies defeated during the current step, with the per-enemy values given in Table A4; HP_{lost} is the amount of base health lost during the step; and $\mathbb{I}_{(\cdot)}$ denotes an indicator function that equals 1 when the corresponding event occurs during the current environment step and 0 otherwise:

- $\mathbb{I}_{\text{wave_cleared}}$: the current wave was successfully cleared.
- $\mathbb{I}_{\text{victory}}$: all 50 waves were completed.
- $\mathbb{I}_{\text{game_over}}$: the base was destroyed (0 base HP).
- $\mathbb{I}_{\text{valid_draft}}$: a valid technology-draft action was selected.
- $\mathbb{I}_{\text{invalid_draft}}$: an invalid action was selected during the paused draft phase.
- $\mathbb{I}_{\text{invalid_action}}$: an invalid non-drafting action was selected.
- $\mathbb{I}_{\text{idle}}$: an idle action was selected.

The base-health and terminal terms penalize leaks and episode failure, while the wave-clear bonus increases with progression. Each non-Boss unit that reaches the base costs exactly one point of base health regardless of its archetype, whereas a Boss that reaches the base destroys it outright, so a single Boss leak incurs the full -40 terminal penalty rather than a -4 leak penalty. Valid draft actions receive a small positive reward, whereas drafting violations incur a large penalty of -100 to enforce compliance with the paused draft menu; ordinary invalid actions incur only a small penalty of -0.1. The coefficient λ_{idle} controls the idle-action penalty and is set to 0 in all reported experiments.

In addition to the densely shaped reward, ProcTD provides other reward variants for controlled ablations: `standard` retains only the wave-completion bonus, `linear` gives $+1$ per wave cleared, and `sparse` gives $+1$ only for clearing the full 50-wave horizon. Their empirical effects are evaluated in Appendix F.2.

A.7 Default Environment Configuration

Table A6: Environment configuration used across all evaluations.

Parameter	Value
Starting gold	200
Base HP	15
Max enemies	2048
Enemy HP multiplier	1.0
Curriculum scale	1.0
Anti-idle wait penalty	0.0
Max wave	50

B PROCEDURAL MAP GENERATION

Each episode generates a unique grid through a constrained multi-step algorithm.

B.1 MAP CONSTRUCTION

Opening path. The base is placed at a random row on the left edge, excluding the corner tiles. Up to 100 candidate opening paths are generated by a constrained random walk with target length 6–9, restricted to the first eight columns. Straight moves have a sampling weight 2.0 and turns weight 1.0. Candidates are scored as

$$\text{Score} = 2.0 \times \text{turns} + 0.05 \times \text{adjacent_buildable_slots},$$

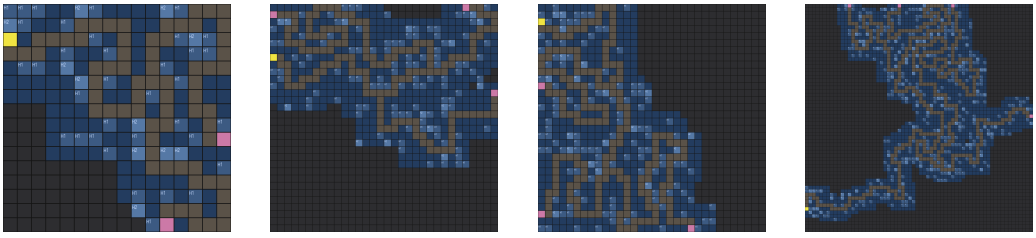
and the first candidate with score ≥ 4.0 is selected; if none satisfies this criterion, a short straight path is used as a fallback.

Main trunk. From the endpoint of the opening path, a greedy self-avoiding walk is steered toward a jittered near-center target with an offset of up to $\pm \max(2, W/8)$. The maximum allowed straight-line length is tiered by map width at 4, 6, and 8 tiles for $W \leq 16$, $W \leq 32$, and $W \leq 64$, respectively, encouraging the formation of chokepoints. Moves that reduce distance to the target receive weight 2.5, moves that increase distance receive sampling weight 0.8, and continuing in the same direction receives an additional factor of 1.4. Dead ends trigger backtracking rather than terminating generation, improving reliability on larger maps.

Branching spawns. The generator creates 2, 3, or 4 branches for $W \leq 16$, $W \leq 32$, and $W \leq 64$, respectively; on three- and four-branch maps, one or two are secondary branches splitting off an existing branch 30–45% along its length, producing a forked route network. The total branch-length budget scales with the linear map extent $W + H$ and the number of branches, and is divided evenly across them. Each branch is steered toward one of seven predefined boundary anchors, must terminate on a boundary tile, and is accepted only after reaching at least 75% of its target length; the resulting endpoints define the enemy spawn locations. Generation is retried up to 1000 times per seed, after which the seed is deterministically replaced with a warning, for at most 16,000 attempts in total; across 25,000 maps at each of 16×16, 32×32 and 64×64, no seed required a reseed.

Buildable terrain. For each path tile, the buildable Chebyshev radius is sampled as 1, 2, or 3 tiles with probabilities 20%, 50%, and 30%, respectively. This restricts tower placement to localized regions around the generated routes while still producing variation in the available defensive geometry. The resulting buildable areas therefore shape the tactical placement options available on each map.

Elevation. Buildable tiles adjacent to path corners and splits are elevated with probability 35%, compared to 14% on straight segments; elevated tiles receive level $h = 1$ with probability 0.8 and $h = 2$ with probability 0.2, and tiles near the opening path are always elevated to guarantee early high ground. A tower at elevation $h \in \{0, 1, 2\}$ receives a damage multiplier of 1.25^h and $+h$ tiles of range.



16×16 32×32 32×32 64×64

Figure B1: Procedurally generated maps across four seeds, varying path curvature, branch structure, chokepoints, elevation, and buildable terrain.

C Implementation and Computational Details

C.1 Simulation Engine

ProcTD uses a data-oriented implementation (Fabian, 2018): instead of per-object Python instances, game state is stored in flat, fixed-capacity NumPy arrays (Harris et al., 2020), supporting up to 2,048 concurrent enemies and 256 towers. Core simulation operations, including enemy movement and targeting, tower cooldowns, damage, splash and slow effects, bounty allocation, tower XP, and wave progression, are executed inside a Numba JIT-compiled kernel that mutates the state in place. This design avoids repeated Python object dispatch and garbage-collection overhead during environment interaction.

C.2 Gymnasium Interface and Execution Details

ProcTD is implemented as a standard `gymnasium.Env` subclass (`TowerDefenseEnv`) and is instantiated directly from a configuration dictionary and not through `gym.register`. The `reset(seed, options)` method returns an `(observation, info)` pair, and `step(action)` returns the standard five-tuple `(observation, reward, terminated, truncated, info)`. The action space is a single `spaces.Discrete` covering build, upgrade, wait, draft, and, when enabled, sell actions (Appendix A.1). The observation space is a `spaces.Dict` containing the spatial map tensor (`spaces.Box`), a normalized resource vector (`spaces.Box`), a multi-hot offered-card mask (`spaces.MultiBinary`), offered-card slot indices (`spaces.Box`), and an active-upgrade count vector (`spaces.Box`).

Each non-drafting action advances the simulator by 50 internal ticks, including build, upgrade, wait, and invalid normal actions. During technology drafting, the simulation is fully paused and advances only after a valid draft choice has been made. Consequently, one environment step corresponds to a fixed block of simulated game time during normal play, while draft decisions occur between waves without advancing the underlying simulation.

The environment exposes an `action_masks()` method returning a boolean array over the complete raw action space, following the `sb3-contrib` (Raffin et al., 2021) `MaskablePPO` convention without requiring a separate `ActionMasker` wrapper. The mask encodes state-dependent validity constraints such as affordability, visibility, occupancy, upgrade eligibility, and technology unlocks, while the underlying action indices remain fixed across states. The MAPS interface (Sec. 4.2) is implemented as a `gym.Wrapper` around the base environment that maps compressed macro-actions to the corresponding raw actions and constructs the corresponding aggregated action mask over the macro action space.

For training, environments are parallelized using Stable-Baselines3's (Raffin et al., 2021) `SubprocVecEnv` and wrapped in `VecMonitor` for episode-level statistics. `VecNormalize` is used with observation normalization disabled (`norm_obs=False`), since the spatial and global observation components are already scaled to comparable ranges, while reward normalization is enabled during training and disabled during evaluation.

ProcTD additionally provides optional visual rendering for inspecting generated maps and agent behavior. Rendering is intended for visualization and debugging and is not required for training or evaluation. All throughput measurements and reported RL experiments are performed without visual rendering enabled.

C.3 Throughput

Environment speed is measured with a single-threaded benchmark: one environment instance stepped in a plain Python loop with uniformly random actions, after a warm-up episode that triggers Numba compilation, so no JIT cost enters the timing. Episode resets, and therefore procedural map generation, are included. One agent step advances the simulation by 50 ticks. All figures are single-core; the environment is neither vectorized nor internally parallel, so throughput scales approximately linearly with the number of worker processes. Each figure is the mean of three runs. These numbers characterize the simulator in isolation; end-to-end training throughput is lower, as it additionally incurs policy inference, rollout storage, and gradient updates.

Table C1: Single-core environment throughput (steps per second, random policy).

CPU	16×16	32×32	64×64
AMD Ryzen 5 5600X	10,324	14,371	14,765
Intel Core i9-14900HX	13,632	20,986	23,519
Intel Core i7-1355U	14,280	19,567	22,152

Throughput increases with map size because the benchmark uses a random policy. On larger maps enemies traverse far longer paths, so a wave occupies many more steps, and the run spends nearly all of its time inside the first few waves: the mean number of live enemies per step falls from 0.41 at 16×16 to 0.02 at 64×64, and the mean tower count from 6.1 to 3.0. Per-step cost scales with the product of these, so larger maps are cheaper to simulate under a random policy, up to a floor set by the fixed per-step overhead. The measurement therefore characterizes simulator overhead rather than difficulty, and should not be read as larger maps being cheaper to train on.

D COMPETITIVE MULTI-AGENT EXTENSION

Attacker action space. On each attacker turn, the agent emits an ordered spawn queue of shape $(Q_{\max}, 2)$. Each row (τ, s) specifies an enemy type $\tau \in \{0, 1, 2\}$, corresponding to Standard, Runner, and Tank, and an active spawn index s. The row capacity is $Q_{\max} = \max(64, \lfloor \text{Budget}_W/2 \rfloor)$, where Budget_W is the budget of the final wave W and 2 is the cost of the cheapest unit, so the queue can always hold the largest budget-respecting wave; at $W = 50$ this gives $Q_{\max} = 535$. An out-of-range enemy-type value acts as an end-of-queue sentinel, allowing variable-length waves within the fixed-size action representation. Our attackers follow the same wave budget and enemy unlock progression as the single-agent generator described in Appendix A.3. For balance, the environment enforces a per-wave cap on how many enemies may spawn, equal to the enemy count the single-agent generator would produce for that wave. An attacker may allocate its queue however it likes, including beyond the nominal budget, but spawning stops once the cap is reached and the remainder of the queue is discarded. The cap bounds unit count rather than point cost, so cheap-unit strategies are the ones it truncates. On Boss waves, the submitted attacker queue is ignored and the Boss is generated by the environment according to the standard wave generator.

Asymmetric observation and death grid. The attacker receives the same spatial tensor as the defender, but with the tower channels zeroed, preventing direct observation of tower type, level, or presence. Enemy channels and shared static map information remain visible. To provide indirect information about the defense, the attacker additionally receives a persistent death grid of shape $(3, H, W)$, with one channel for each non-Boss enemy archetype. Whenever a tower destroys an attacker unit, its location is added to the corresponding channel through a 3×3 spatial blur that contributes 1 to each of the nine surrounding cells; units that survive to reach the base are not recorded, so the grid measures where the defense is lethal rather than where attacks fail. The grid accumulates across waves and is exposed to the attacker as $\log(1 + x)/\log(1 + 10^5)$, clipped to $[0, 1]$, which keeps late-episode structure visible after heavily defended cells have absorbed many kills. The death grid is therefore the attacker's only observation that indirectly reflects the defender's tower placement and effectiveness.

D.1 SCRIPTED ATTACKERS

Two scripted attackers anchor evaluation: a *random* attacker (uniform among affordable types) and a *heuristic* attacker that reads the death grid to shift among standard-flood, runner-heavy, and tank-heavy strategies.

Both spend the wave budget at unit costs 3, 2, and 5 for Standard, Runner, and Tank, respect the unlock schedule (Runners from wave 5, Tanks from wave 8), and draw a single active spawn point uniformly at random for the whole wave, so that composition rather than routing is what separates them. Neither acts on Boss waves.

The random attacker repeatedly samples uniformly among the currently affordable types until the remaining budget cannot cover any unit. Its expected composition is therefore fixed by the cost structure alone, and it carries no information about the defense from one wave to the next.

The heuristic attacker scores each unlocked type by kills per point, k_τ/c_τ, where k_τ is the total accumulated in channel τ of the death grid, and commits the whole budget to the type with the lowest score: the archetype the defense has so far been least cost-effective at stopping. Three queue builders realize that choice. A *flood* fills the budget with a single type. A *runner-heavy* mix places a Tank in every third slot, so that the Tank absorbs fire while Runners advance. A *tank-heavy* mix spends a quarter of the budget on Runners before filling the remainder with Tanks. A pivot rule overrides the score when Tanks appear to be dying cheaply: the attacker tracks the increase in the summed Tank channel of the death grid across a wave, divided by the number of Tanks it sent, and counts a wave as easy when this ratio exceeds 0.5. After three consecutive easy waves it switches to Runners for two waves.

D.2 LEARNED ATTACKERS

Beyond the scripted baselines, we train a PPO attacker directly against each frozen defender, selecting a wave composition (the split of the wave budget among standard, runner, and tank units), a spawn point, and a release order at every wave.

Factored action space. The raw $(Q_{\max}, 2)$ queue is a poor fit for a policy-gradient learner: at 16×16 it is a 1070-dimensional action whose entries past the sentinel are ignored. The learned attacker therefore acts in a factored discrete space, MultiDiscrete($[15,\ S{+}1,\ 2]$) with $S = 4$ the maximum number of spawn points, and a wrapper compiles each action into a queue. The first factor selects the budget split (f_0, f_1, f_2) among Standard, Runner, and Tank from the 15 points of the simplex grid with step $1/4$. Unit counts are $n_\tau = \lfloor f_\tau \,\text{Budget}_w / c_\tau \rfloor$ with $(c_0, c_1, c_2) = (3, 2, 5)$; residual budget is spent on Standard units, and on one further Runner if at least 2 points remain and Runners are unlocked. A share assigned to a type that has not yet unlocked is added to the Standard share, so the action space stays fixed while the effective choice set grows with the wave number. The second factor selects one of the S active spawn points, taken modulo the number currently active, or a round-robin that assigns consecutive units to successive spawn points. The third selects the release order: grouped, with Tanks first, then Standards, then Runners; or interleaved, cycling Tank–Standard–Runner while units remain. On Boss waves the action is ignored and an empty queue is submitted.

Observation. In addition to the asymmetric tensor and death grid described above, the attacker receives nine scalars summarizing the previous wave: base damage inflicted (normalized by maximum base HP), the share of units sent of each of the three types, an estimated per-type kill rate, and two flags indicating whether Runners and Tanks are unlocked for the coming wave. Kill rates are estimated from the per-channel increase in the raw, unscaled death grid divided by nine, the number of cells each kill writes to under the 3×3 blur, then divided by the number of units sent of that type and clipped to $[0, 1]$; kills at the map border write to fewer than nine cells, so the estimate is a slight underestimate. These counts are read from the simulator state rather than from the observation, since the grid the policy sees is log-scaled by the environment and is therefore not linear in the number of kills.

Table D1: Attacker PPO hyperparameters.

Parameter	**Value**
Total attacker steps	10,000,000
Parallel environments	42
Rollout length per environment	256
Batch size	512
Epochs per update	10
Optimizer	Adam, lr $= 3 \times 10^{-4}$
Discount γ	0.99
GAE λ	0.95
Clip range	0.2
Entropy coefficient	0.01
Value coefficient	0.5
Gradient norm clip	0.5
Target KL	0.03
Reward normalization	running estimate, clipped to $[-10, 10]$
Map encoder	Conv 32–64–64, strides 1–2–2
Death-grid encoder	Conv 16–32–32, strides 1–2–2
Scalar encoder	Linear $\to$ 32
Policy / value heads	2×256 each
Activation	ReLU

Training. The attacker is PPO with a convolutional trunk over the map and death-grid tensors and a two-layer MLP over the scalar features, followed by separate policy and value heads. The opponent is a defender checkpoint loaded once and frozen; it acts deterministically and is never updated, so

from the attacker's side the game reduces to a stationary MDP. One attacker step spans a full wave, during which the defender acts every 50 simulation ticks, so an episode is at most 50 attacker steps and 10 million attacker steps correspond to as many simulated waves. Hyperparameters are listed in Table D1. Evaluation is deterministic over 100 seeded episodes, matching the protocol used for the scripted attackers.

Attacker reward and termination. The defender retains the reward function from Eq. (9), while the attacker receives

$$r_t^{\text{att}} = 1.0\,\text{HP}_{\text{leaked}} + 10.0\,\mathbb{I}_{\text{boss_leak}},$$

where $\text{HP}_{\text{leaked}}$ is the amount of defender's base HP lost during the current environment step, and $\mathbb{I}_{\text{boss_leak}}$ is an indicator that equals 1 if a Boss reaches the base and 0 otherwise. The two reward streams are tracked independently, yielding a general-sum game with strongly opposed objectives. An episode terminates when the base is destroyed, resulting in an attacker victory, or when the defender survives all 50 waves, resulting in a defender victory.

E BASELINES AND TRAINING DETAILS

E.1 SPATIAL PLACEMENT HEURISTIC

The Advanced Heuristic and the two MAPS-based agents share the same spatial placement scoring procedure. The following section summarizes this common placement logic. The three agents differ in how the resulting spatial scores are integrated into their decision-making.

For a candidate tile (y, x) and tower type t, the placement score is

$$S_t(y,x) = E_t(y,x)\, M_t(y,x)\, D_t(y,x) + B_t(y,x),$$

where E_t is a tower-specific effectiveness score, M_t is an optional early-game modifier, D_t is a distance-based preference, and B_t is a fork-proximity bonus.

Let $h(y,x)$ denote the terrain height of a candidate tile and w the current wave. Path coverage is the soft route multiplicity of the tile's radius-2 neighbourhood,

$$p(y,x) = \sum_{|dy|\le 2,\ |dx|\le 2} \mathbb{I}_{\text{visible path}}(y{+}dy,\ x{+}dx)\ \big[\,1 + 0.4\,(\text{rc}(y{+}dy,\ x{+}dx) - 1)\,\big],$$

where rc counts how many enemy routes pass through a tile, so a trunk tile shared by two routes contributes 1.4 rather than 2. Only revealed path tiles contribute, so the score uses no information hidden from the agent. Candidate tiles with $p = 0$ are discarded for Basic, AOE and Support towers and are never scored.

The effective damage term is

$$E_t(y,x) = \begin{cases} 0.25(1+0.2h), & t = \text{Basic},\\ 0.50(1+0.2h), & t = \text{Sniper},\\ 0.075\,p(1+0.2h), & t = \text{AOE},\\ 0.05\,p(1+0.2h), & t = \text{Support},\\ 1.5, & t = \text{Economy},\ h = 0,\ p = 0,\\ -1, & t = \text{Economy},\ \text{otherwise}. \end{cases}$$

The early-game modifier is

$$M_t(y,x) = \begin{cases} 5.0, & t = \text{AOE},\ N_{\text{AOE}} = 0,\ w < 5,\ p \ge 3,\\ 2.5, & t = \text{AOE},\ N_{\text{AOE}} = 0,\ w < 5,\ p < 3,\\ 2.0, & t = \text{Sniper},\ N_{\text{Sniper}} = 0,\ 5 \le w < 8,\\ 1.0, & \text{otherwise}, \end{cases}$$

where N_t is the number of currently constructed towers of type t. The distance factor is

$$D_t(y,x) = \max\!\left(0.1,\ 1 - 0.5\,\frac{d_t(y,x)}{d_{\max}}\right), \qquad d_{\max} = \begin{cases} \sqrt{H^2+W^2}/2, & t = \text{Sniper},\\ \sqrt{H^2+W^2}, & t \ne \text{Sniper}, \end{cases}$$

where d_t is the Euclidean distance to the map center for Sniper towers and to the base for all other tower types, and H, W are the map height and width.

The three overrides encode the fact that early waves are won or lost by a single placement rather than by marginal damage efficiency. A defense that misplaces its first few towers loses before wave 5 and never observes the mid- or late-game dynamics the benchmark is meant to probe, so the overrides buy the agent passage through roughly the first 10% of an episode: an AOE tower against the opening Standard clusters, a Sniper once Runners unlock at wave 5, and opening Basic placements chosen for raw defensive value rather than by a score calibrated for a built-up defense. Each deactivates as soon as its condition is met, leaving the steady-state policy to S_t. The constants were tuned on 16×16 maps and held fixed at all other map sizes.

A fork bonus rewards tiles near the point where the routes diverge. Let $\mathcal{F}$ be the set of revealed trunk tiles of maximal route multiplicity that neighbour a tile of lower multiplicity; $\mathcal{F} = \emptyset$ when the map has a single route or the fork has not yet been revealed. Then

$$B_t(y,x) = \begin{cases} \dfrac{0.5}{1 + \min_{(f_y,f_x)\in\mathcal{F}} \max(|y-f_y|,\ |x-f_x|)}, & t \ne \text{Economy},\ \mathcal{F} \ne \emptyset,\\ 0, & \text{otherwise}, \end{cases}$$

using Chebyshev distance. The placement score is

$$S_t(y,x) = E_t(y,x)\, M_t(y,x)\, D_t(y,x) + B_t(y,x),$$

and the heuristic builds on $\arg\max_{(y,x)} S_t(y,x)$ over the legal tiles for type t.

One exception overrides the score entirely. While fewer than three Basic towers stand, Basic placements are scored by

$$S_{\text{Basic}}(y,x) = 25\,h + 10\,p + \max(0,\ 15 - d_{\text{base}}(y,x))\,,$$

which prioritises high ground on heavily travelled tiles close to the base, and tiles with $p = 0$ remain excluded. This rule determines the opening placements and reverts to the general score once three Basic towers exist.

E.2 Baseline Agents

We compare nine baseline agents: four non-learned and five learned agents. Unless stated otherwise with †, all baselines operate on the action space masking invalid options.

- **Random.** Samples uniformly from the currently valid actions.
- **Simple Heuristic.** Builds only Basic towers, ignoring drafting and upgrades. Placement maximizes $S_{\text{basic}}(y,x) = 20.0\,h(y,x) - D_{\text{base}}(y,x)$, favoring elevated tiles close to the base while ignoring chokepoints and specialized tower classes.
- **Advanced Heuristic.** A fully scripted policy using the spatial placement heuristic described in Sec. E.1, together with hand-designed rules for drafting, resource management, and manual upgrades, detailed later.
- **PPO.** PPO operating directly on the full raw action space with invalid-action masking.
- **Unmasked PPO (†).** PPO operating directly on the full raw action space without invalid-action masking. This baseline isolates the effect of action masking relative to PPO.
- **MAPS-PPO.** PPO operating on the macro-action space. For build actions, the policy selects the tower class and the shared spatial heuristic selects the placement tile; non-placement decisions, including drafting and manual upgrades, are selected directly by the learned policy.
- **MAPS-RND.** Uses the same macro-action space and spatial placement heuristic as MAPS-PPO, but samples uniformly from the currently valid macro-actions. This baseline isolates the contribution of the learned policy from the spatial placement prior and MAPS action-space structure.
- **Unmasked DQN (†).** A value-based learned baseline operating directly on the full raw action space without invalid-action masking. This baseline tests whether the failure of unmasked PPO is specific to policy-gradient methods or is a property of the raw action space itself.
- **DreamerV3 (†).** A model-based learned baseline operating on the full raw action space without action masking. Because DreamerV3 trains its actor through imagined latent-state rollouts, simulator-derived action masks are not directly available for imagined future states.

Advanced Heuristic-specific rules:

- **Drafting.** Among the three offered cards, the policy selects the unlock card whose tower type achieves the best projected value-per-gold at its best available placement, evaluated with the same scoring function as when selecting the tower to build; non-unlock cards are treated as neutral and ties are broken uniformly at random.
- **Resource management.** The policy maintains a savings buffer equal to the scaled cost of the next intended tower and acts only when gold covers both the action and the buffer, so that spending never leaves it unable to place the tower it has selected.
- **Manual upgrades.** Upgrades are considered only once at least four towers stand and only for towers below two manual upgrades, scored by elevation and current tower level less half the upgrade cost, and are preferred over new construction when affordable. When neither an upgrade nor a build passes the buffer test, the policy waits.

E.3 Learned-Agent Details

PPO Network architectures. The learned PPO-based agents are implemented in Stable-Baselines3 (Raffin et al., 2021) using `MaskablePPO` from `sb3-contrib` (Huang & Ontañón, 2022). Unmasked PPO, PPO and MAPS-PPO use the same multi-input feature extractor at a given map size, with the convolutional architecture scaled to the spatial resolution. The spatial encoder configurations are summarized in Table E1. Non-spatial observation components are encoded separately and concatenated with the resulting map embedding. The concatenated representation is passed to separate actor and critic MLP heads.

Table E1: Network architecture used for Unmasked PPO, PPO and MAPS-PPO (w/o selling) for 16×16 and 32×32 maps.

Component	16×16	32×32
Spatial input	$15 \times 16 \times 16$	$15 \times 32 \times 32$
CNN filters	$32, 64, 64$	$32, 64, 64, 128$
Kernel size	3×3	3×3
Strides	$1, 2, 2$	$1, 2, 2, 1$
Map embedding	1,024	8,192
Resources encoder	32	32
Offered-card encoder	32	32
Active-upgrade encoder	32	32
Card-slot encoder	8	8
Actor MLP	$[512, 512]$	$[512, 512]$
Critic MLP	$[512, 512]$	$[512, 512]$
Activation	ReLU	ReLU

PPO hyperparameters. Unmasked PPO, PPO and MAPS-PPO use identical PPO hyperparameters, summarized in Table E2. The only map-size-dependent setting is the discount factor, which is $\gamma = 0.9975$ on 16×16 maps and $\gamma = 0.99795$ on 32×32 maps.

Table E2: PPO hyperparameters used for Unmasked PPO, PPO and MAPS-PPO.

Hyperparameter	Value
Learning rate α	1×10^{-4}
Rollout length (`n_steps`)	2048
Batch size	1024
Epochs	4
Discount γ (16×16)	0.9975
Discount γ (32×32)	0.99795
GAE λ	0.95
Clip range ϵ	0.20
Entropy coefficient	0.01
Value coefficient	0.5
Maximum gradient norm	0.5
Target KL	0.02

Training and on-the-fly evaluation protocol. PPO and MAPS-PPO are trained with 64 parallel `SubprocVecEnv` workers. Training is capped at 100 million environment steps, with checkpoints saved every 250,000 steps. After a minimum of 5 million steps, training may terminate early when the rolling standard deviations of policy entropy, policy-gradient loss, and value loss over the previous 65 rollouts fall below fixed relative thresholds of 0.01, 0.001, and 0.03, respectively; otherwise training continues to the full step budget: this never happens during our training. During training, learned policies are evaluated every 1 million environment steps on 20 held-out episodes using deterministic action selection. These evaluation episodes use seeds independent of the training en-

vironment seed and are used only for monitoring training progress and convergence. They are not included in the final reported results.

DQN and Dreamer. Both unmasked baselines act directly on the raw action space with no legality filtering, so every configuration in Tables E3 and E4 is otherwise standard for the respective algorithm: the DQN settings follow the Stable-Baselines3 defaults apart from the longer discount horizon ($\gamma = 0.9975$), the wider $[512, 512]$ Q-head needed for the dictionary observation, and a replay buffer sized to fit the map tensors in memory, while DreamerV3 uses the published large-scale configuration with a one-hot actor over the flat action index. The two budgets are deliberately unequal. DQN was run for the full 100M environment steps used by the masked agents, whereas DreamerV3 was capped at 1M steps because its world-model updates dominate wall-clock time at roughly 40 hours per million environment steps, making a matched budget infeasible on our hardware.

Table E3: Hyperparameters for the unmasked DQN baseline, which operates on the full raw action space without invalid-action masking.

Parameter	**Value**
Total environment steps	100,000,000
Parallel environments	4
Optimizer	Adam, lr $= 1 \times 10^{-4}$
Replay buffer size	100,000
Learning starts	10,000
Batch size	256
Discount γ	0.9975
Train frequency	every 4 steps, 1 gradient step
Target update interval	1,000 steps
Soft update τ	0.005
Exploration fraction	0.2 (final $\epsilon = 0.05$)
Gradient norm clip	10.0
Feature extractor	Conv 32–64–64, strides 1–2–2
Scalar encoders	32 / 32 / 32 / 8
Q head	$[512, 512]$, ReLU

Table E4: Hyperparameters for the DreamerV3 baseline, which operates on the full raw action space without invalid-action masking.

Parameter	**Value**
Total environment steps	1,000,000
Parallel environments	8
Episode time limit	2,000 steps
Action repeat	1
Replay capacity	200,000 transitions
Prefill	2,500 steps
Train ratio	512
Batch size $\times$ length	16×64
World-model optimizer	Adam, lr $= 1 \times 10^{-4}$, clip 1000
Actor / critic optimizer	Adam, lr $= 3 \times 10^{-5}$, clip 100
Deterministic state	512
Stochastic state	32×32 discrete
Hidden units	512
CNN depth / kernel / minres	32 / 4 / 4
Imagination horizon	15
Discount / λ	0.997 / 0.95
Actor distribution	one-hot, unimix 0.01
Actor gradient	REINFORCE, entropy 3×10^{-4}
Critic	symlog discrete, slow target 0.02
Reward EMA	enabled

F Extended Results and Analysis

Table F1: Extended single-agent baseline performance on one training seed. Values report mean ($\pm$ std) and median (IQR) for reward and waves survived; † denotes agents without invalid-action masking.

Size	Agent	Mean Reward	Median Reward	Mean Waves	Median Waves
16×16	Random	$-51.8^{\pm 55.2}$	$-62.1^{(36.3)}$	$9.3^{\pm 6.7}$	$7.0^{(6.3)}$
	Simple Heuristic	$-60.9^{\pm 22.9}$	$-54.3^{(15.8)}$	$7.9^{\pm 1.2}$	$8.0^{(2.0)}$
	Advanced Heuristic	$41.8^{\pm 50.2}$	$45.0^{(68.7)}$	$20.6^{\pm 2.5}$	$21.0^{(1.0)}$
	MAPS-RND	$116.4^{\pm 163.1}$	$96.7^{(169.7)}$	$24.8^{\pm 9.5}$	$27.0^{(10.0)}$
	Unmasked PPO†	$-84.7^{\pm 22.9}$	$-80.4^{(17.8)}$	$5.7^{\pm 3.1}$	$5.0^{(2.0)}$
	PPO	$622.5^{\pm 459.7}$	$541.3^{(919.8)}$	$38.1^{\pm 12.5}$	$42.0^{(23.0)}$
	MAPS-PPO	$\mathbf{964.9^{\pm 347.6}}$	$\mathbf{1116.9^{(36.4)}}$	$\mathbf{46.1^{\pm 9.7}}$	$\mathbf{50.0^{(0.0)}}$
	DreamerV3†	$255.7^{\pm 257.8}$	$195.5^{(187.4)}$	$28.3^{\pm 10.2}$	$27.0^{(11.0)}$
	Unmasked DQN†	$-23.6^{\pm 35.0}$	$-25.0^{(41.9)}$	$14.3^{\pm 5.5}$	$14.0^{(10.0)}$
32×32	Random	$-58.1^{\pm 42.6}$	$-69.0^{(35.9)}$	$8.3^{\pm 4.9}$	$7.0^{(5.0)}$
	Simple Heuristic	$-59.9^{\pm 25.1}$	$-54.1^{(32.2)}$	$8.5^{\pm 1.1}$	$9.0^{(1.0)}$
	Advanced Heuristic	$32.7^{\pm 49.9}$	$41.9^{(55.7)}$	$19.7^{\pm 3.0}$	$20.0^{(4.0)}$
	MAPS-RND	$116.4^{\pm 130.6}$	$87.7^{(115.5)}$	$25.4^{\pm 7.8}$	$27.0^{(9.0)}$
	Unmasked PPO†	–	–	–	–
	PPO	$255.7^{\pm 257.8}$	$195.5^{(187.4)}$	$28.3^{\pm 10.2}$	$27.0^{(11.0)}$
	MAPS-PPO	$\mathbf{844.6^{\pm 440.2}}$	$\mathbf{1116.9^{(644.4)}}$	$\mathbf{42.7^{\pm 13.0}}$	$\mathbf{50.0^{(9.0)}}$
	DreamerV3†	–	–	–	–
	Unmasked DQN†	–	–	–	–

Table F2 additionally reports the wall-clock training time for all models.

Table F2: Wall-clock training time and step counts for each run.

Run	Steps	Wall-clock
MAPS-PPO, 16×16	100M	9.024 hr
MAPS-PPO, 16×16, selling	150M	16.48 hr
MAPS-PPO, 32×32	100M	12.46 hr
PPO, 16×16	100M	8.42 hr
PPO, 16×16, selling	150M	23.91 hr
PPO, 32×32	100M	8.73 hr
Unmasked PPO†, 16×16	100M	5.56 hr
Unmasked DQN†, 16×16	100M	18.28 hr
DreamerV3†	1M	40.00 hr

F.1 Unmasked Models' Results

Table F1 shows the extended results with DreamerV3 and DQN included. Like unmasked PPO, unmasked DQN unmasked DQN also performs poorly on the raw action space. Similarly, DreamerV3, which cannot naturally handle action masks in imagined trajectories, is not able to achieve substantial survival. An unmasked action space leads to worse performance and sample efficiency, as critical training effort is spent on learning which actions are legal rather than which actions are optimal. We omit the 32×32 unmasked PPO and DQN results, as these already did not perform well on 16×16.

F.2 Reward Shaping

We study reward density (Ng et al., 1999) with four modes sharing identical dynamics: `dense` (Equation (9)), `standard` (survival bonus only), `linear` ($+1$ per wave), and `sparse` ($+1$ on full

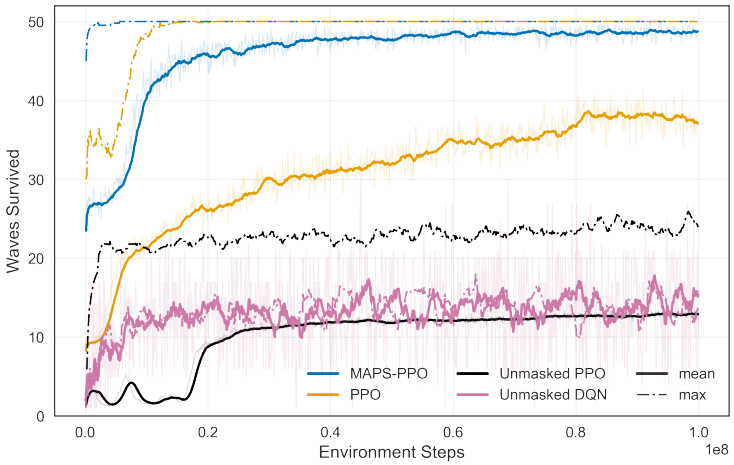
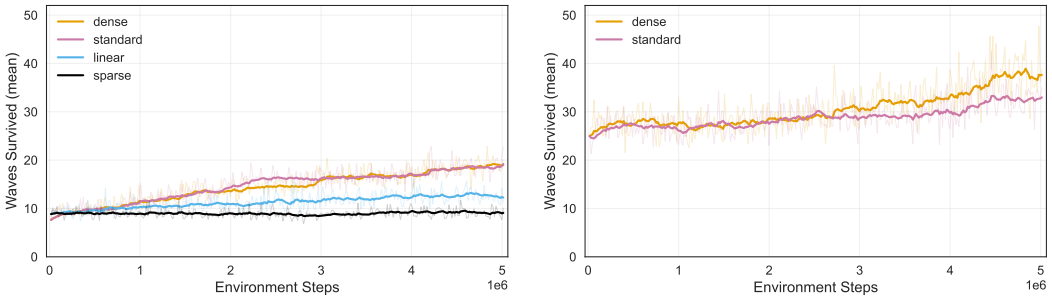


Figure F1: PPO vs. Unmasked PPO vs. Unmasked DQN on 16×16; DQN's max waves appears lower than mean waves at times - this is a result of smoothing.

victory), trained for 5M steps each. Under PPO (Figure F2a) `linear` and `sparse` fail to provide a usable signal while `dense` and `standard` perform similarly. On MAPS-PPO (Figure F2b), the dense reward asserts a clear, sustained advantage that is attributable to late-game credit assignment, where steeply scaling enemy health rewards fine-grained feedback. We therefore adopt the `dense` reward throughout.



(a) PPO with all four modes.

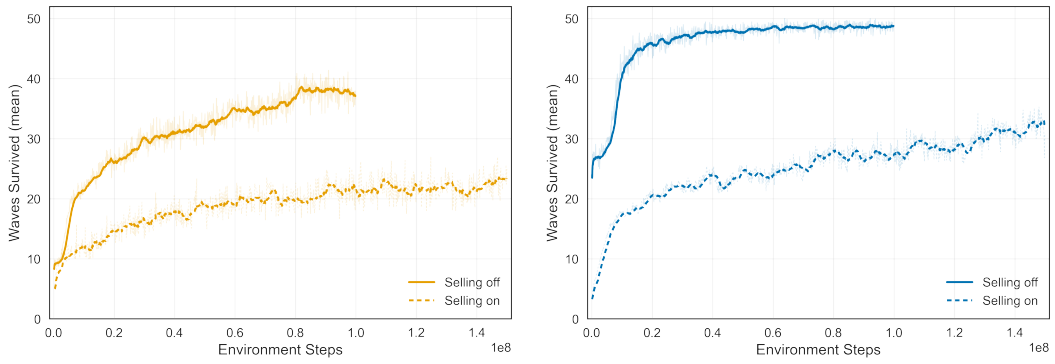
(b) MAPS-PPO, `standard` vs. `dense`.

Figure F2: Reward-shaping ablation, measuring mean waves survived over 5M steps. For PPO, sparse and linear rewards do not perform well, while dense and the standard reward lead to improved training efficiency. For MAPS-PPO, the dense rewards improves upon the standard reward.

F.3 Tower Selling

As discussed in Sec. 3, the inclusion of the selling mechanic results in 256 new actions on the 16×16 map. Models are trained for 150M steps to account for this (Figure F3).

The larger action space with enabled selling slows training. Enabling the option to sell towers suppresses the efficiency and performance of PPO and MAPS-PPO. This again hints at a close connection between the size and complexity of the action space and the performance. Although having the option to sell should theoretically not be worse than not having it, since the agent can opt to simply not use it, selling load-bearing towers can majorly impact survival. Thus, potentially negative actions seem to have large impact.



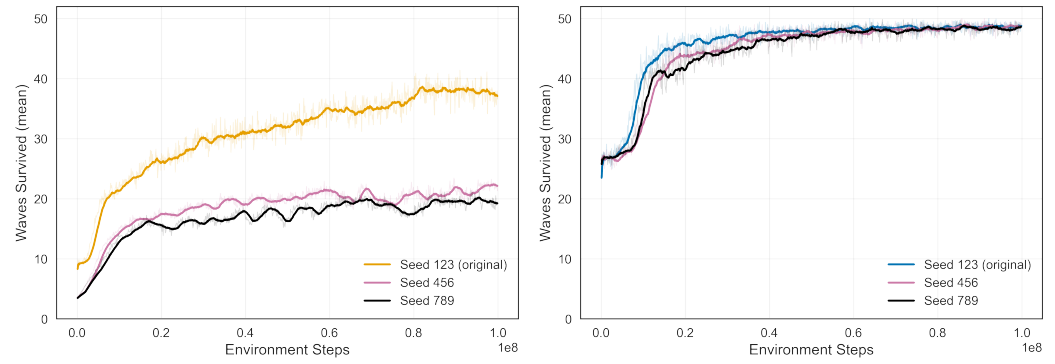
(a) PPO. (b) MAPS-PPO.

Figure F3: Comparison of PPO and MAPS-PPO with and without being able to sell towers. For both methods, having the option to sell towers diminishes performance, showing that neither is able to handle potentially detrimental actions.

F.4 Training Seed Sensitivity

Figure F4 reports three training-seed replicates, with the environment seed fixed at 123 so that map, card, and route generation is identical across runs and any spread is attributable to the algorithm's own random initialization and rollout stochasticity. MAPS-PPO converges tightly across all three seeds, but PPO diverges sharply: the initial seed, on which PPO hyperparameters were tuned on, performs much better. MAPS-PPO also exhibits more stable learning across training seeds.

The failure mode is not merely slower convergence: two of three PPO seeds plateau at roughly half the terminal performance of the third with no late-training recovery. Since the hyperparameters were tuned on the seed that reaches 37-38 waves, PPO's performance on the raw interface appears to depend on seed-specific tuning and does not transfer to others.



(a) PPO (b) MAPS-PPO

Figure F4: Comparing the training seed sensitivity of PPO and MAPS-PPO. PPO is extremely sensitive to the seed it was tuned for and performs much worse on two other seeds. MAPS-PPO is largely robust to the training seed.

G Statistical and Distributional Analysis

Because RL performance distributions are high-variance and non-Gaussian, parametric tests are often invalid (Colas et al., 2018; Agarwal et al., 2021). For each fixed trained checkpoint, we evaluate policies on the same 100 seeded environment instances and use the non-parametric Wilcoxon signed-rank test for pairwise comparisons (Demšar, 2006). The Friedman omnibus test yields 392.88 ($p = 1.02 \times 10^{-82}$) on 16×16, indicating highly significant differences. Critical-difference diagrams (Figure G1) and superiority heatmaps (Figure G2) report the post-hoc pairwise results, and violin plots (Figure G3) expose the strong distributional skew. MAPS-PPO stands strictly alone at the highest rank on both map sizes (avg. rank 1.42 and 1.43); on 32×32 PPO and MAPS-RND converge to the same 27.00 median yet PPO retains a statistically superior ranking profile.

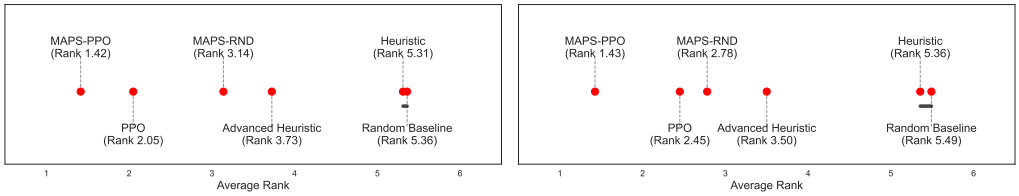


Figure G1: Critical-difference diagrams (Friedman + pairwise Wilcoxon) on (**left**) 16×16 and (**right**) 32×32. Models joined by a bar are not significantly different.

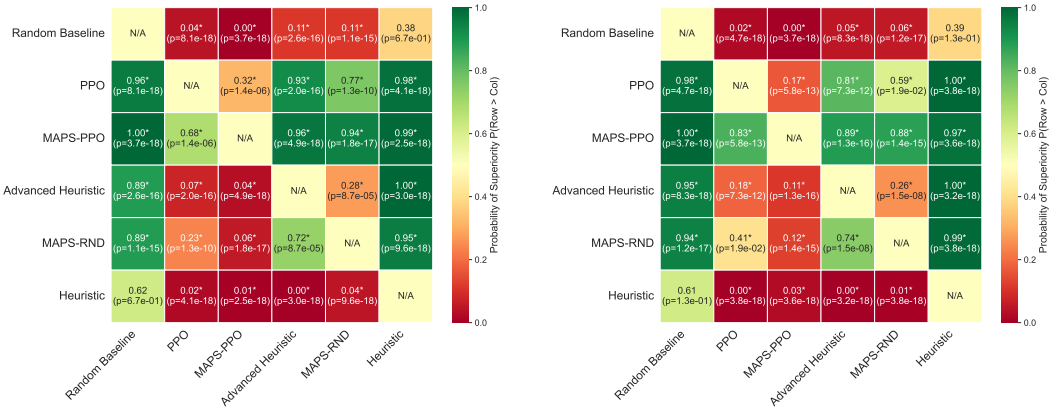


Figure G2: Probability-of-superiority heatmaps for all model pairs on 16×16 (**left**) and 32×32 (**right**).

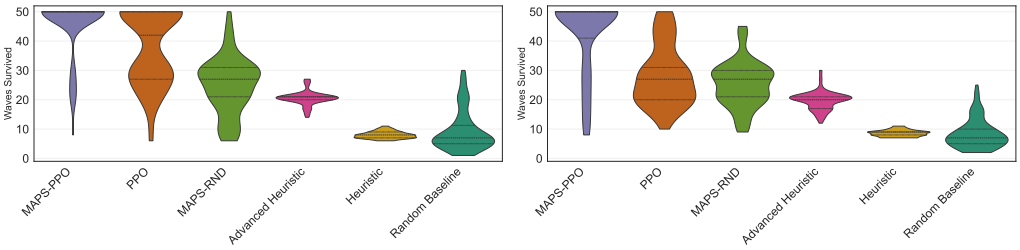


Figure G3: Distribution of outcomes across 100 evaluation episodes on (**left**) 16×16 and (**right**) 32×32.