MINDFUL Seminar Series

9/17/2021

Open-Ended Learning Leads to Generally Capable Agents

Open-Ended Learning Team, DeepMind, London, UK

https://arxiv.org/abs/2107.12808

Presented by Drew Buck

University of Missouri

Department of Electrical Engineering and Computer Science

Results Showreel

Outline

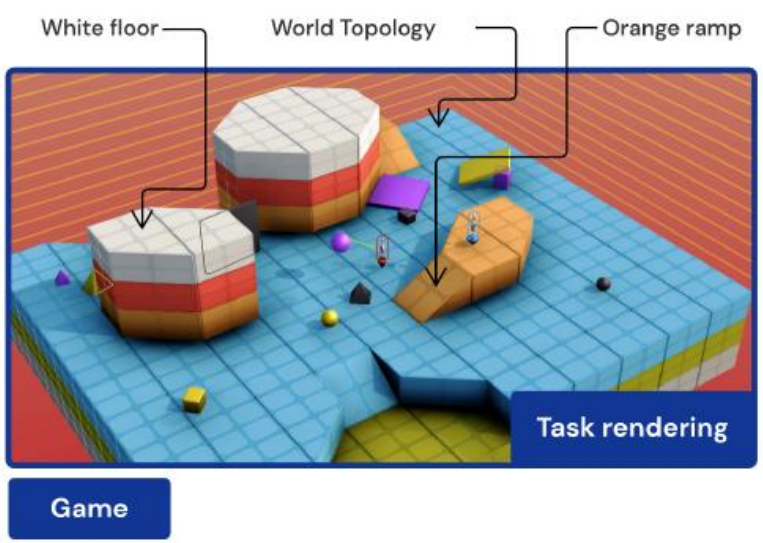
Outline (continued)

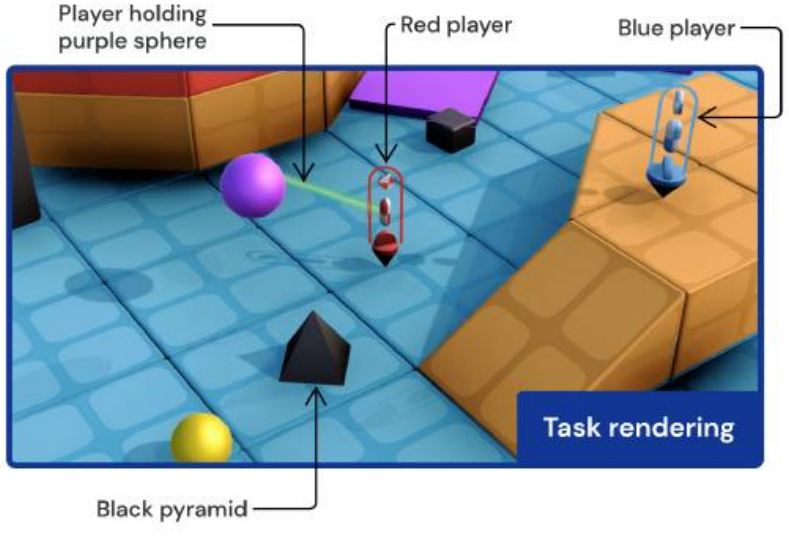
Additional Resources

- **Blog post (good summary):** https://deepmind.com/blog/article/generally-capable-agents-emerge-from-open-ended-play

- **Results showreel (15 min):** https://www.youtube.com/watch?v=lTmL7jwFfdw

XLand Environment Space





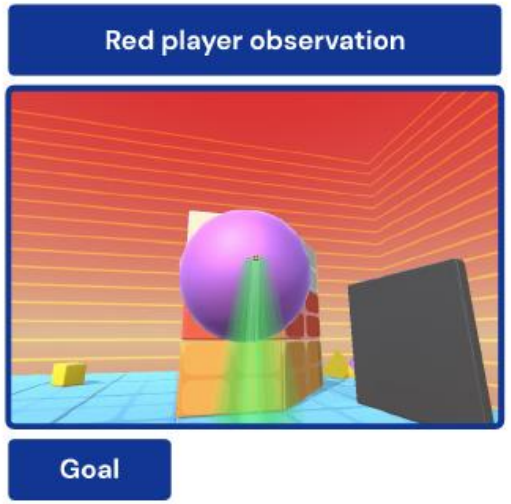


Figure 2 | **(Left & Center)** An instance of a *task* within the XLand environment space, composed of the *world* – the layout of the topology, initial object and player positions, and player gadgets – as well as the *game* – the specification of rewarding states for each player in this task. **(Right)** The observation of the red player consisting of the first-person view and the goal of the player.

The XLand Universe

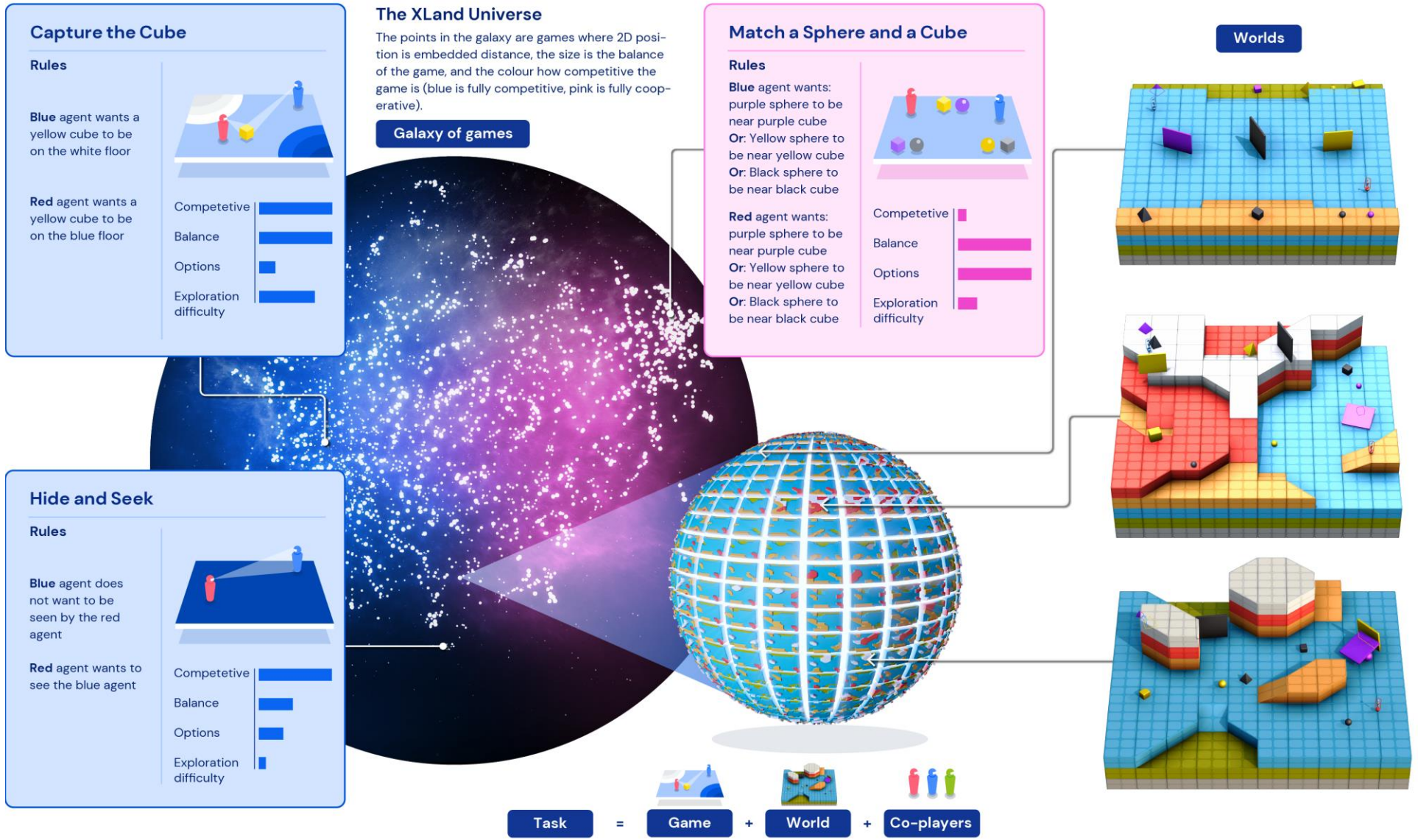


Figure 3 | Visualisation of the XLand environment space. **(Left)** Each dot corresponds to a single *game* and is positioned by a 2D UMAP embedding of distance between games, with the size of the dot corresponding to the *balance* of the game, and the colour representing competitiveness of the game (from blue – completely competitive, to purple – completely cooperative). **(Right)** Each game can be played on a myriad of *worlds*, which we can smoothly mutate to traverse a diverse set of physical challenges. **(Bottom)** An XLand *task* consists of combining a game with a world and co-players.

World Space

Figure 6 | An example array of worlds from the XLand environment space.

Worlds contain:

- Static topology
  - Up to 5 layers
  - Ramps, walls, … constrained
- Movable objects
  - 3 colors
  - 4 shapes
- Players
  - Up to 3 players
  - Each player has one gadget:
    - Freeze gadget
      - Makes object temporarily unmovable
    - Tag gadget
      - Temporarily removes another player
      - Player is returned to its initial position

Procedural World Generation

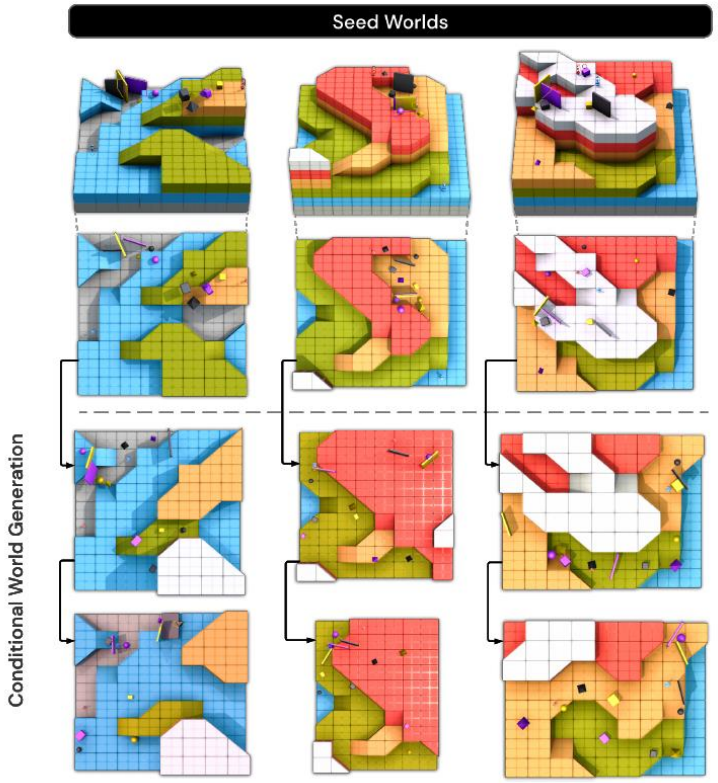


Figure 31 | Worlds can be generated conditioned on an existing world. This allows smooth variation of worlds. This figures shows three examples of this process, with each column showing an initial seed world and the rows showing two steps of conditional world generation.

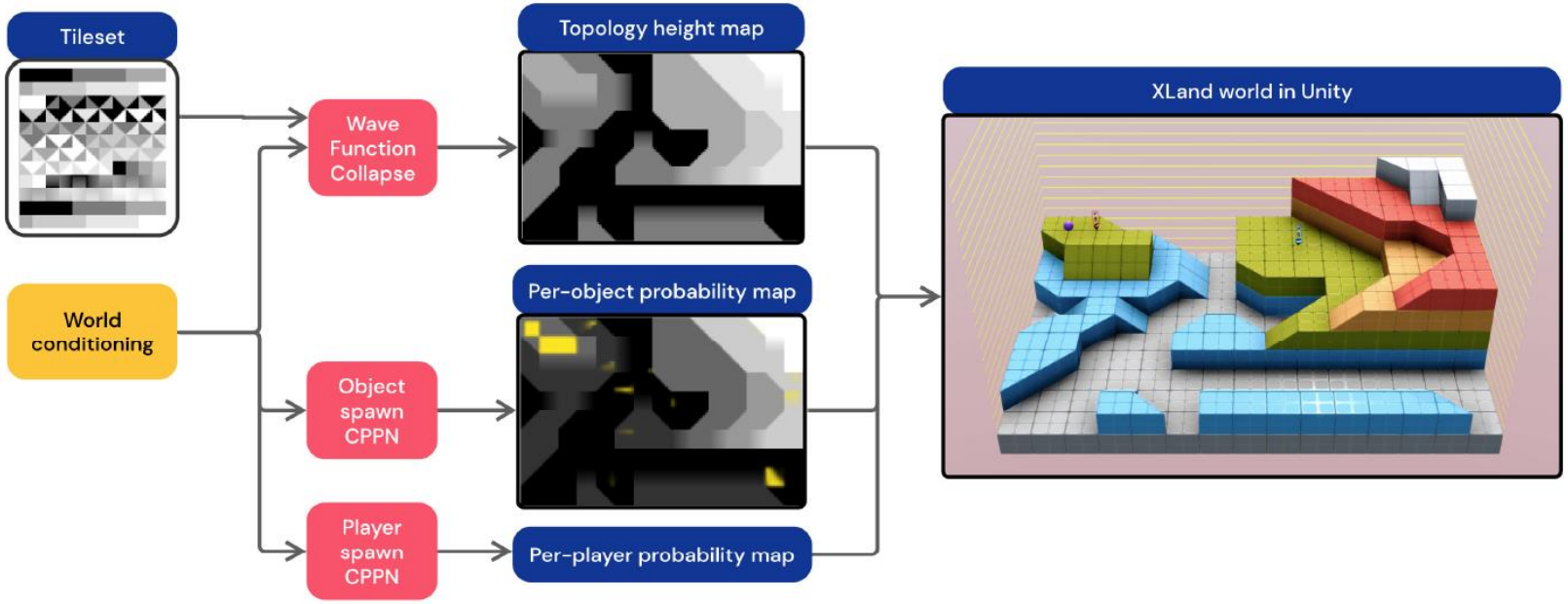


Figure 32 | The steps of procedural world generation. The process is conditioned on a seed and optionally an existing world to be similar to. Wave Function Collapse (Gumin, 2016) acting on a tileset of primitive building blocks creates a height map of the topology. Additionally, for each object and player, a CPPN (Ha, 2016; Stanley, 2007b) creates a probability map for the entity's initial spawn location. These elements are combined in the Unity game engine to produce the playable world.

World-Agent Co-Evolution

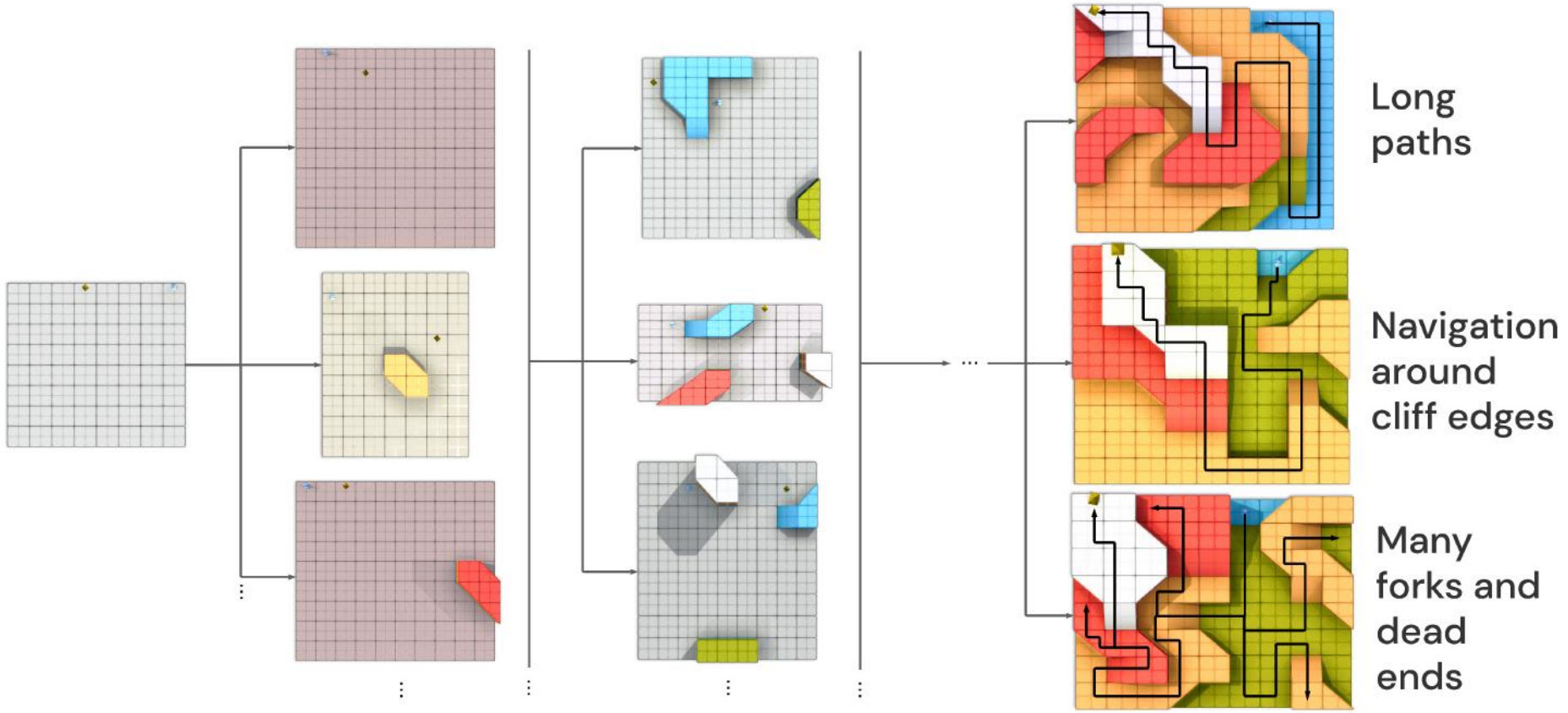


Figure 33 | The process of world-agent co-evolution resulting in complex worlds. **(Left)** The initial seed world to the train world set. The agent is trained to maximise the reward of its goal *"Be near a yellow pyramid"*. **(Middle)** The progression of the worlds in the train set as training progresses. Worlds undergo evolution with a minimum criterion fitness such that a world must be solved sometimes but not too often by the agent. **(Right)** The resulting world set is more diverse in terms of the navigational feature space and exhibit interesting topological elements.

"To evaluate a world in the evaluate set, we make the agent play in the world for 100 episodes. … If the agent scores a reward in at least one episode but less than half of the 100 episodes, the corresponding world is added to the train set."

Game Space

A game $\boldsymbol{G}$ consists of a goal $\boldsymbol{g}_i \in \mathcal{G}$ for each of the n players, $\boldsymbol{G} = (\boldsymbol{g}_1, \ldots, \boldsymbol{g}_n)$.

A binary atomic predicate $\phi_j : \boldsymbol{S} \rightarrow \{0, 1\}$ indicates if a physical relation is true in a particular state of the simulated environment $\boldsymbol{s} \in \boldsymbol{S}$.

Ex: `near(purple sphere, opponent)`

A goal could look like

$$\boldsymbol{g} = \underbrace{(\phi_{j_1} \wedge \phi_{j_2})}_{\text{option 1}} \vee \underbrace{(\phi_{j_2} \wedge \phi_{j_3} \wedge \phi_{j_4})}_{\text{option 2}}$$

meaning, "*Hold a purple sphere* (ϕ_{j_1}) *while being near a yellow sphere* (ϕ_{j_2}) ***or*** *be near a yellow sphere* (ϕ_{j_2}) *while seeing an opponent* (ϕ_{j_3}) *who is not holding the yellow sphere* (ϕ_{j_4})".

The reward function $r_{\boldsymbol{g}}(\boldsymbol{s})$ for a goal $\boldsymbol{g} \coloneqq \bigvee_{i=1}^{k} \left[\bigwedge_{j=1}^{n_i} \phi_{ij} \right]$ is

$$r_{\boldsymbol{g}}(\boldsymbol{s}) = \min\left(\sum_{i=1}^{k} \prod_{j=1}^{n_i} \phi_{ij}(\boldsymbol{s}), 1 \right) \in \{0, 1\}$$

Ex: The game hide and seek consists of two goals $(\boldsymbol{g}_{\text{seek}}, \boldsymbol{g}_{\text{hide}})$

$$\boldsymbol{g}_{\text{seek}} = \phi_{\text{seek}} = \texttt{see(me, opponent)}$$

$$\boldsymbol{g}_{\text{hide}} = \phi_{\text{hide}} = \texttt{not(see(opponent, me))}$$

Game Properties

To define the more complex game properties, recall that every goal is a Boolean expression over a set of d predicates ϕ_j. Let us define $\phi : \mathcal{S} \to \{0,1\}^d$, a mapping that assigns each simulation state $\mathbf{s}$ to a binary vector of d predicate truth values. A goal is simply a mapping from $\phi(\mathcal{S})$ to $\{0,1\}$, labelling which predicate states are rewarding. We denote by $N_\phi := \#\{\phi(\mathbf{s}) : \mathbf{s} \in \mathcal{S}\}$ the size of the predicate state space. We define a distance metric between two goals $\mathbf{g}_i$ and $\mathbf{g}_j$ as

$$\|\mathbf{g}_i - \mathbf{g}_j\|_{\mathcal{G}} := \frac{\#\{\phi(\mathbf{s}) : r_{\mathbf{g}_i}(\mathbf{s}) \neq r_{\mathbf{g}_j}(\mathbf{s})\}}{N_\phi} \in [0,1].$$

Definition 3.4. *Exploration difficulty of a game is the fraction of predicate states in which no player is being rewarded.*

$$\kappa(\mathbf{G}) = \kappa((\mathbf{g}_1, \ldots, \mathbf{g}_n)) = \frac{\#\{\phi(\mathbf{s}) : \forall_k r_{\mathbf{g}_k}(\mathbf{s}) = 0\}}{N_\phi}$$

we will also call the unnormalised exploration difficulty the quantity

$$\hat{\kappa}(\mathbf{G}) := N_\phi \kappa(\mathbf{G}).$$

Definition 3.5. *Cooperativeness is the fraction of predicate states in which all the players are being rewarded compared to the number of predicate states in which at least one of them is.*

$$\mathrm{coop}(\mathbf{G}) = \mathrm{coop}((\mathbf{g}_1, \ldots, \mathbf{g}_n)) = \frac{\#\{\phi(\mathbf{s}) : \forall_k r_{\mathbf{g}_k}(\mathbf{s}) = 1\}}{N_\phi - \hat{\kappa}(\mathbf{G})}$$

Definition 3.6. *Competitiveness is the fraction of predicate states in which some but not all players are being rewarded compared to the number of predicate states in which at least one of them is.*

$$\mathrm{comp}((\mathbf{g}_1, \ldots, \mathbf{g}_n)) = \frac{\#\{\phi(\mathbf{s}) : \max_k r_{\mathbf{g}_k}(\mathbf{s}) \neq \min_k r_{\mathbf{g}_k}(\mathbf{s})\}}{N_\phi - \hat{\kappa}(\mathbf{G})}$$

Definition 3.7. *Balance with respect to game transformations $\Xi \supset \{\text{identity}\}$ is the maximal cooperativeness of the game when goals are transformed with elements of Ξ:*

$$\mathrm{bal}(\mathbf{G}) = \max_{\xi \in \Xi} \mathrm{coop}(\xi(\mathbf{G})).$$

Game Examples

Simple navigation task XLand games include simple challenges such as a player being tasked with finding an object of interest and grabbing it. Tasks like this challenge navigational skills, perception, and basic manipulation.

$$g_1 := \texttt{hold(me, yellow sphere)}$$
$$g_2 := \texttt{near(me, yellow pyramid)}$$
$$\kappa(G) = \tfrac{1}{4} \quad \text{comp}(G) = \tfrac{2}{3} \quad \text{bal}(G) = \tfrac{7}{15}$$

Simple cooperation game Setting the goal of both players to be identical gives a fully cooperative, balanced game, which challenges a player's ability to navigate and manipulate objects, but also to synchronise and work together.

$$g_1 := \texttt{near(yellow pyramid, yellow sphere)}$$
$$g_2 := \texttt{near(yellow pyramid, yellow sphere)}$$
$$\kappa(G) = \tfrac{1}{2} \quad \text{comp}(G) = 0 \quad \text{bal}(G) = 1$$

Hide and Seek A well known game of hiding and seeking, that has been used in the past as a source of potentially complex behaviours (Baker et al., 2020). This is an example of a simple, fully competitive, imbalanced game in XLand.

$$g_1 := \texttt{see(me, opponent)}$$
$$g_2 := \texttt{not(see(opponent, me))}$$
$$\kappa(G) = 0 \quad \text{comp}(G) = 1 \quad \text{bal}(G) = \tfrac{1}{3}$$

Capture the Cube The competitive game of Capture the Flag has been shown to be as a rich environment for agents to learn to interact with a complex 3d world, coordinate and compete (Jaderberg et al., 2019). Each player must get the flag (for example represented as a cube) to their base floor to score reward. An example one-flag instantiation of this game in XLand (with a supporting world) is

$$g_1 := \texttt{on(black cube, blue floor)} \wedge \texttt{not(on(black cube, red floor))}$$
$$g_2 := \texttt{on(black cube, red floor)} \wedge \texttt{not(on(black cube, blue floor))}$$
$$\kappa(G) = \tfrac{1}{4} \quad \text{comp}(G) = 1 \quad \text{bal}(G) = 1$$

Game Examples

XRPS A final example is that of XRPS games, inspired by the study of non-transitivites in games leading to strategic depth (Czarnecki et al., 2020; Vinyals et al., 2019). We give each player three options to choose from, each one being explicitly countered by exactly one other option. A player can choose to pick up a yellow sphere, but it will get a reward if and only if an opponent is not holding a purple sphere; if it picks up a purple sphere the reward will be given if and only if the opponent does not pick up a black sphere, and so on. With these cyclic rules, players are encouraged not only to navigate and perceive their environment, but also to be aware of opponent actions and strategies, and to try to actively counter potential future behaviours, leading to potentially complex, time-extended dynamics.

$$
\begin{aligned}
\widehat{g}_{\text{rock}} := \;& \texttt{hold(me,yellow sphere)} \wedge \\
& \texttt{not(hold(opponent,yellow sphere))} \wedge \\
& \texttt{not(hold(opponent,purple sphere))} \\
\widehat{g}_{\text{paper}} := \;& \texttt{hold(me,purple sphere)} \wedge \\
& \texttt{not(hold(opponent,purple sphere))} \wedge \\
& \texttt{not(hold(opponent,black sphere))} \\
\widehat{g}_{\text{scissors}} := \;& \texttt{hold(me,black sphere)} \wedge \\
& \texttt{not(hold(opponent,black sphere))} \wedge \\
& \texttt{not(hold(opponent,yellow sphere))} \\
g_1 := \;& \widehat{g}_{\text{rock}} \vee \widehat{g}_{\text{paper}} \vee \widehat{g}_{\text{scissors}} \\
g_2 := \;& \widehat{g}_{\text{rock}} \vee \widehat{g}_{\text{paper}} \vee \widehat{g}_{\text{scissors}} \\
\kappa(\mathbf{G}) = \;& \tfrac{1}{4} \quad \text{comp}(\mathbf{G}) = 1 \quad \text{bal}(\mathbf{G}) = 1
\end{aligned}
$$

Generating Games

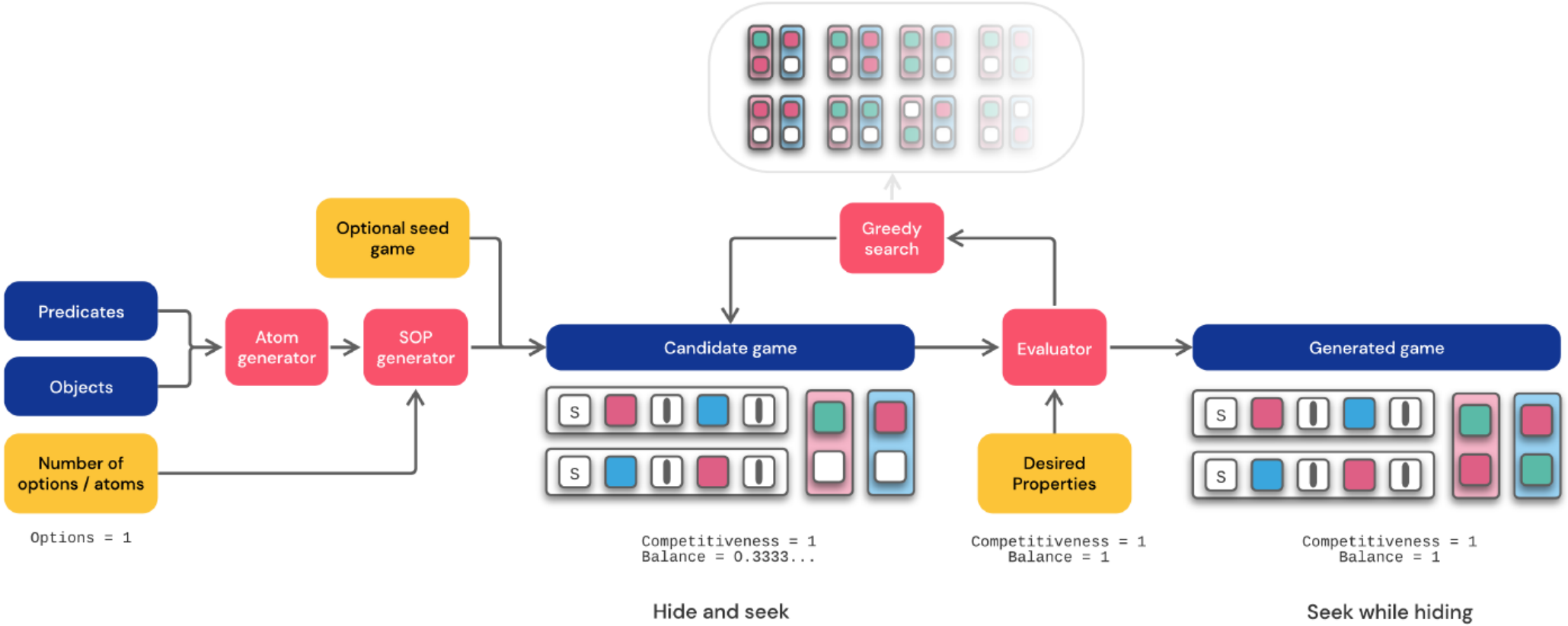


Figure 37 | The process of generating of a game with target properties. We see the matrix representation, as well as relations. Greedy local search performs simple modifications to the game matrix, as described in Section A.2.

Normalized Percentiles

Goal when training agents is to maximize expected total return (reward) over all tasks.
But, each task can be of completely different complexity, so how to aggregate scores over many tasks?

- Agents should catastrophically fail on a few tasks as possible
- Agents should be competent on as many tasks as possible
- Broad ability is preferred over narrow competency

A Game Theoretic solution would be to focus on the worst-case scenario.

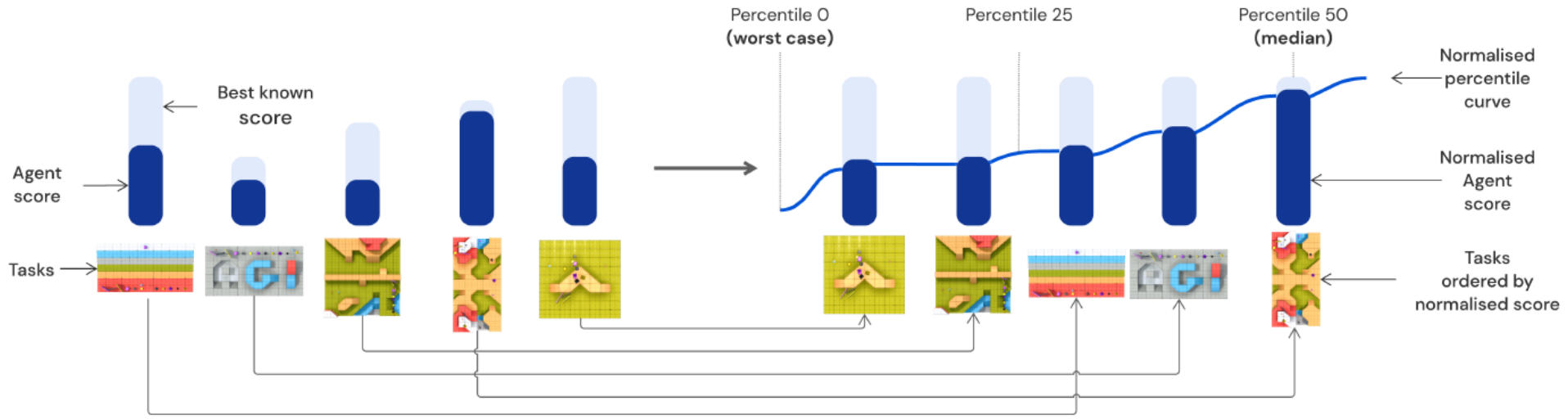


Figure 10 | The process of computing normalised percentiles. Tasks vary significantly in terms of their complexity, some have much higher values of optimal policies than others. We normalise the performance of the agent by an estimate of an optimal policy score – using the Nash equilibrium of trained agents – providing a normalised score, which after ordering creates a normalised percentile curve. This can be iteratively updated as new trained agents are created.

Deep Reinforcement Learning

Three main components for training:

1. Deep RL to update the policy for a single agent
2. Dynamic task generation with population based training (PBT)
3. Generational training to bootstrap behavioral learning

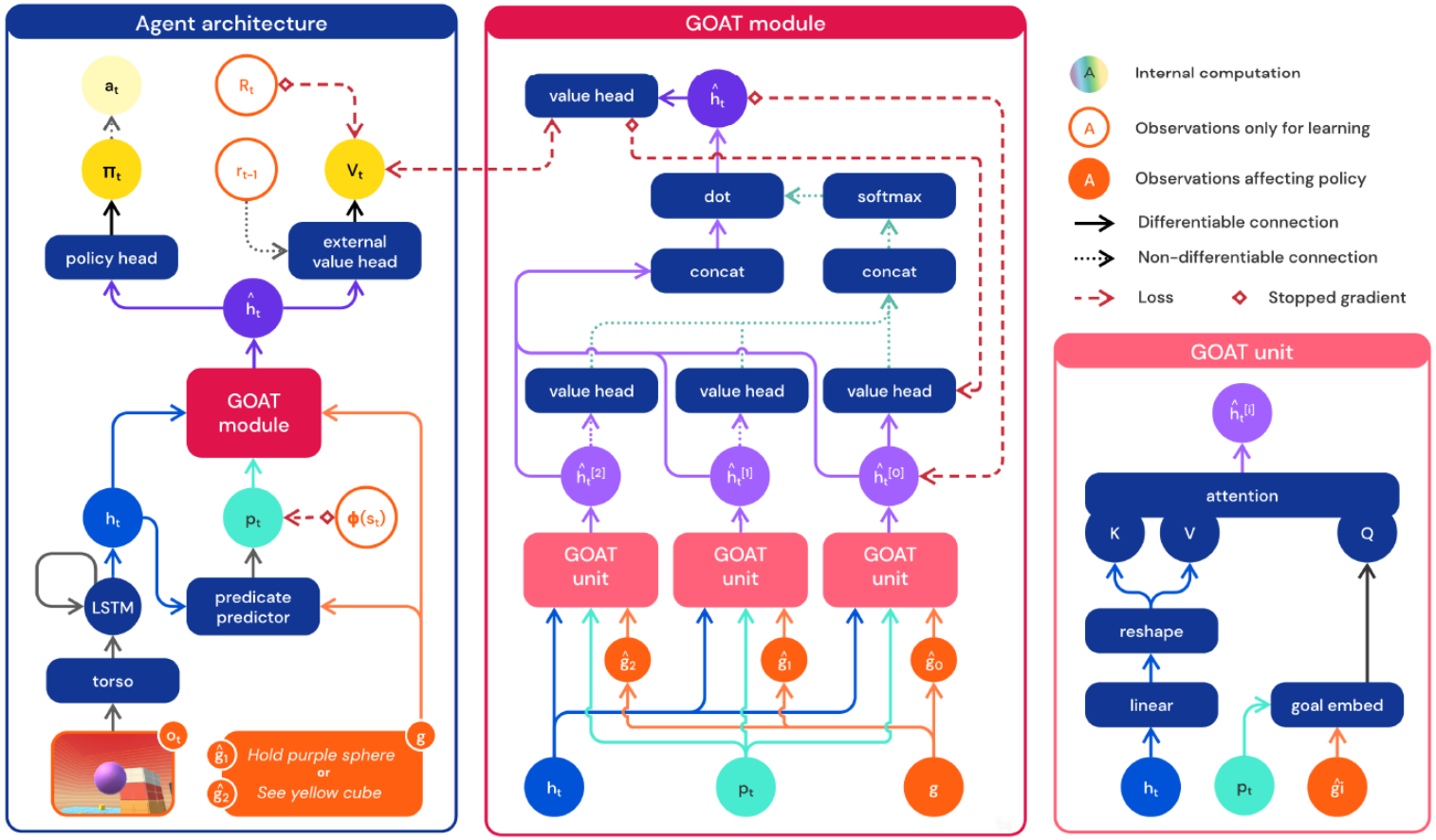


Figure 12 | A schematic of the neural network used to parameterise the agent's policy. The input observations $\mathbf{o}_t$ consist of RGB images and the proprioception, and the agent also receives its goal $\mathbf{g}$. The agent processes the observations through the torso and a recurrent core to produce $\mathbf{h}_t$, which is used for the predicate predictor, producing $\mathbf{p}_t$. The recurrent core output, the predicate predictor output, and the goal is passed to the GOAT module. The GOAT module (see Section 5.1) attends to a specific part of the recurrent representation based on the current goal of the agent, and performs logical analysis of the goal using *value consistency* (see Theorem 5.1). The goal embedding and predicate predictor architectures are provided in Figure 38. Modules with the same names share weights (*i.e.* each value head, as well as each GOAT unit).

Goal Embedding

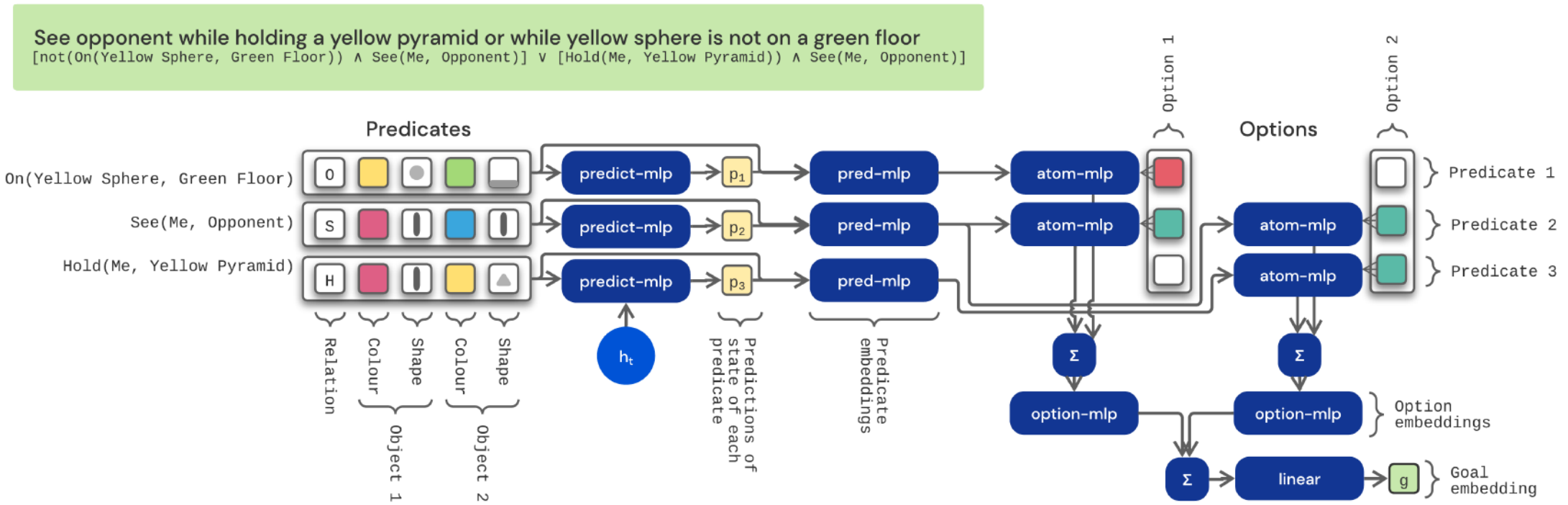
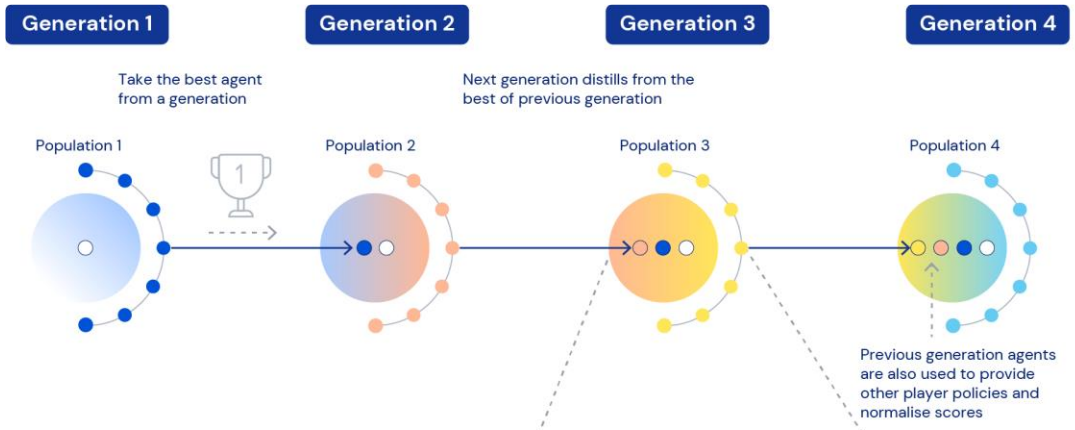
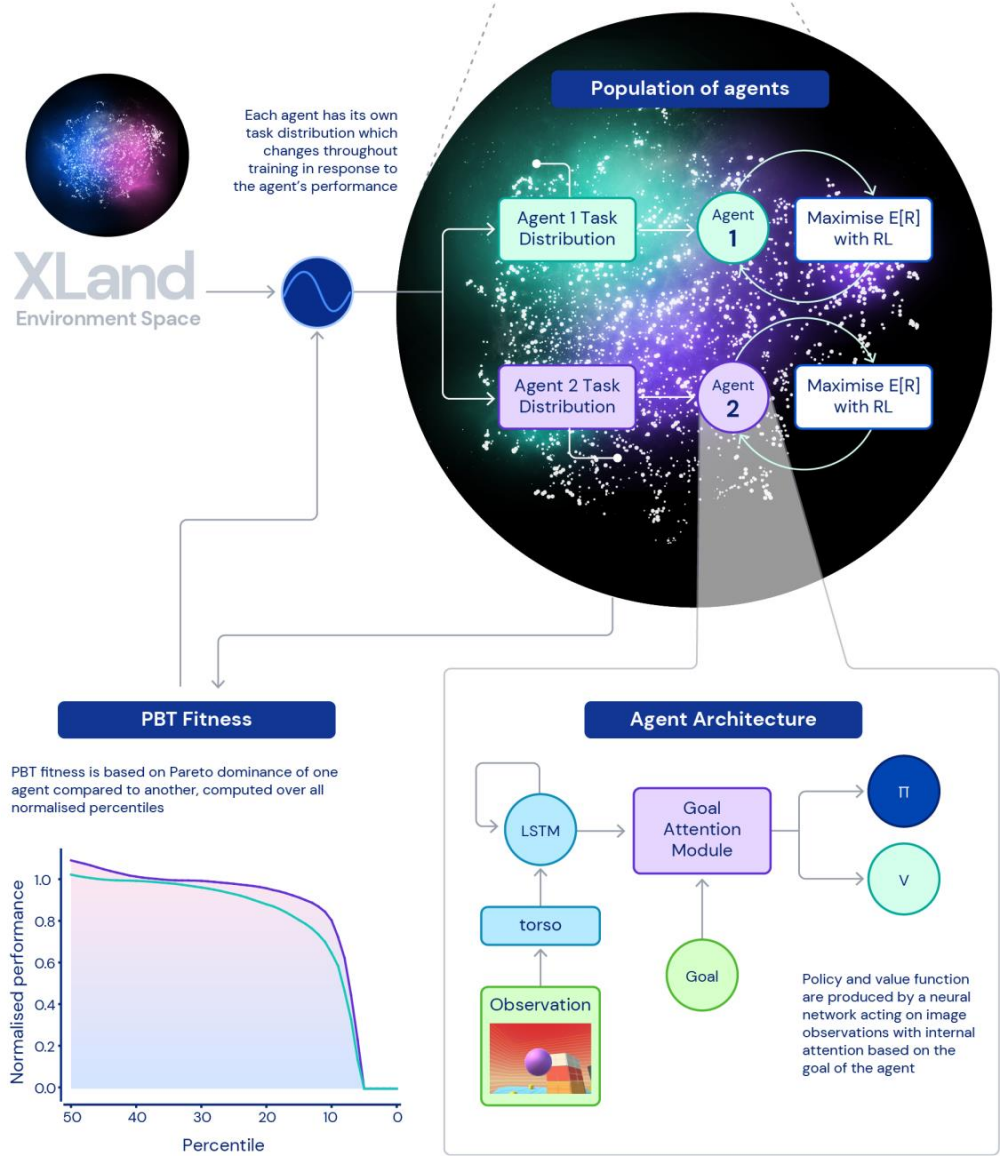


Figure 38 | The architecture of the goal embedding/prediction modules. Atomic predicates are provided in a 5-hot encoded fashion, since all the relations used take two arguments, each of which can be decomposed to a colour and shape. For player object we simply have a special colour "me" and "opponent". Details of the architecture are provided in Section A.6.

Learning Process

Figure 13 | The combined learning process. **(Top)** Generations of agents are trained, composed of populations of agents where the best performing agents become distillation teachers of the next generation as well as co-players to train against. **(Middle)** Inside each population, agents are trained with dynamic task generation that continuously adapts the distribution of training tasks Pπk(ℵ) for each agent πk, and population based training (PBT) modulates the generation process by trying to Pareto dominate other agents with respect to the normalised percentiles metric. **(Bottom)** Each agent trains with deep reinforcement learning and consists of a neural network producing the policy π and value function **v**.





Generational Training

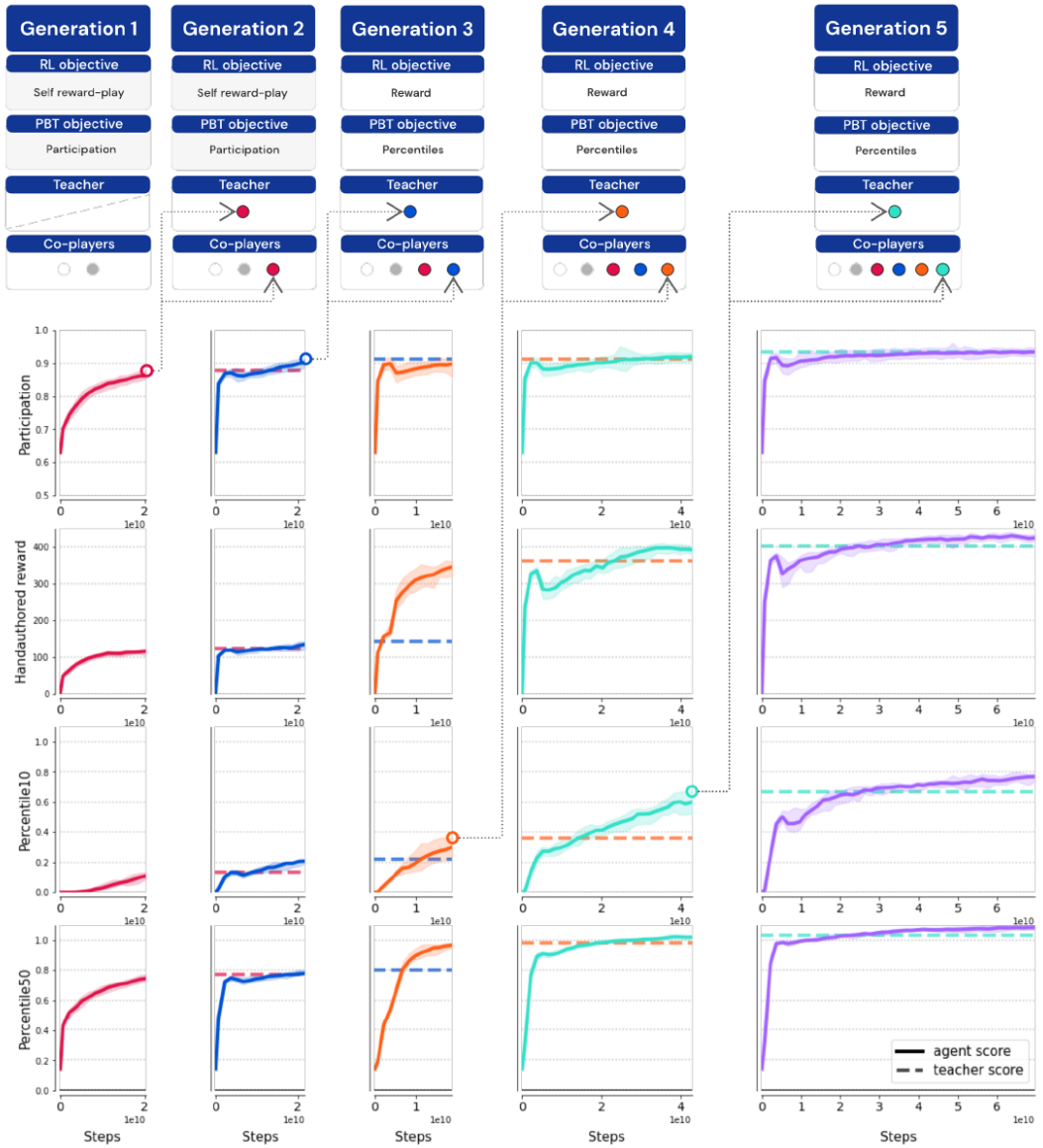


Figure 14 | Generations of performance as measured on the held out test task set. The first two generations focus on the maximisation of participation using the self reward-play RL objective (Section 5.3). In between generations, the best agent wrt. the objective is selected and used as a teacher and additional co-player to play against in further generations. Generations 3-5 focus on the improvement of normalized percentiles, and use the raw reward for the RL algorithm. The dashed line in each plot corresponds to the performance of the teacher from the previous generation. The co-players are the set of policies that populate the co-players in these multiplayer tasks, with this set initialised to just the trivially created noop-action and random-action agents (white and grey circles).

Evaluation

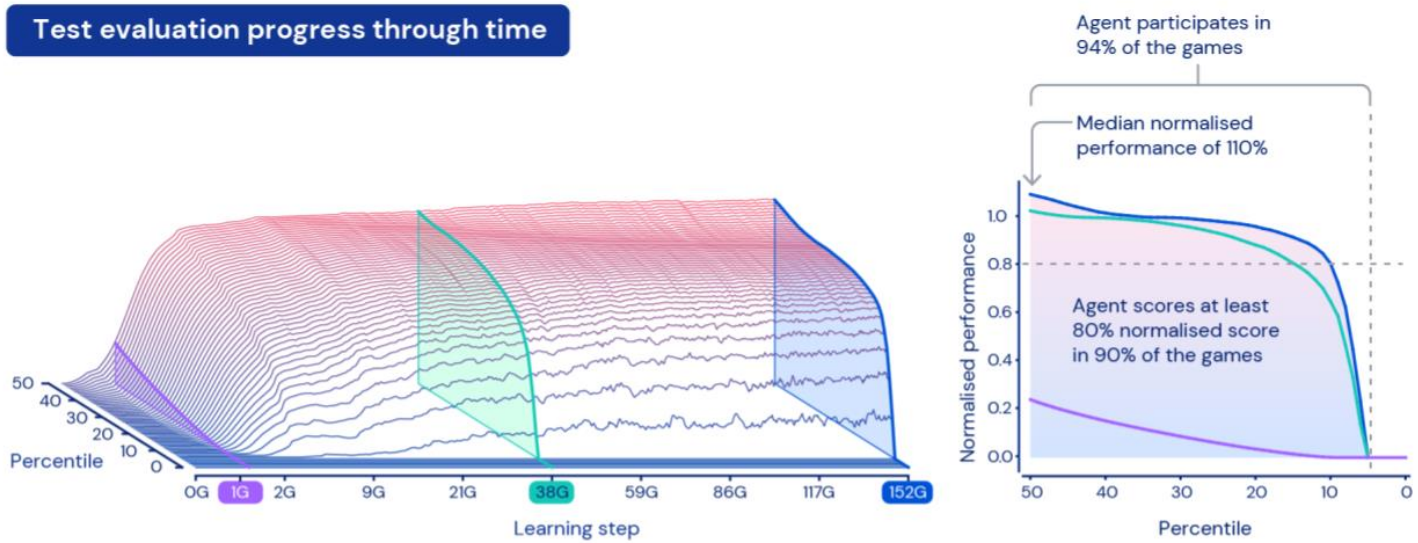
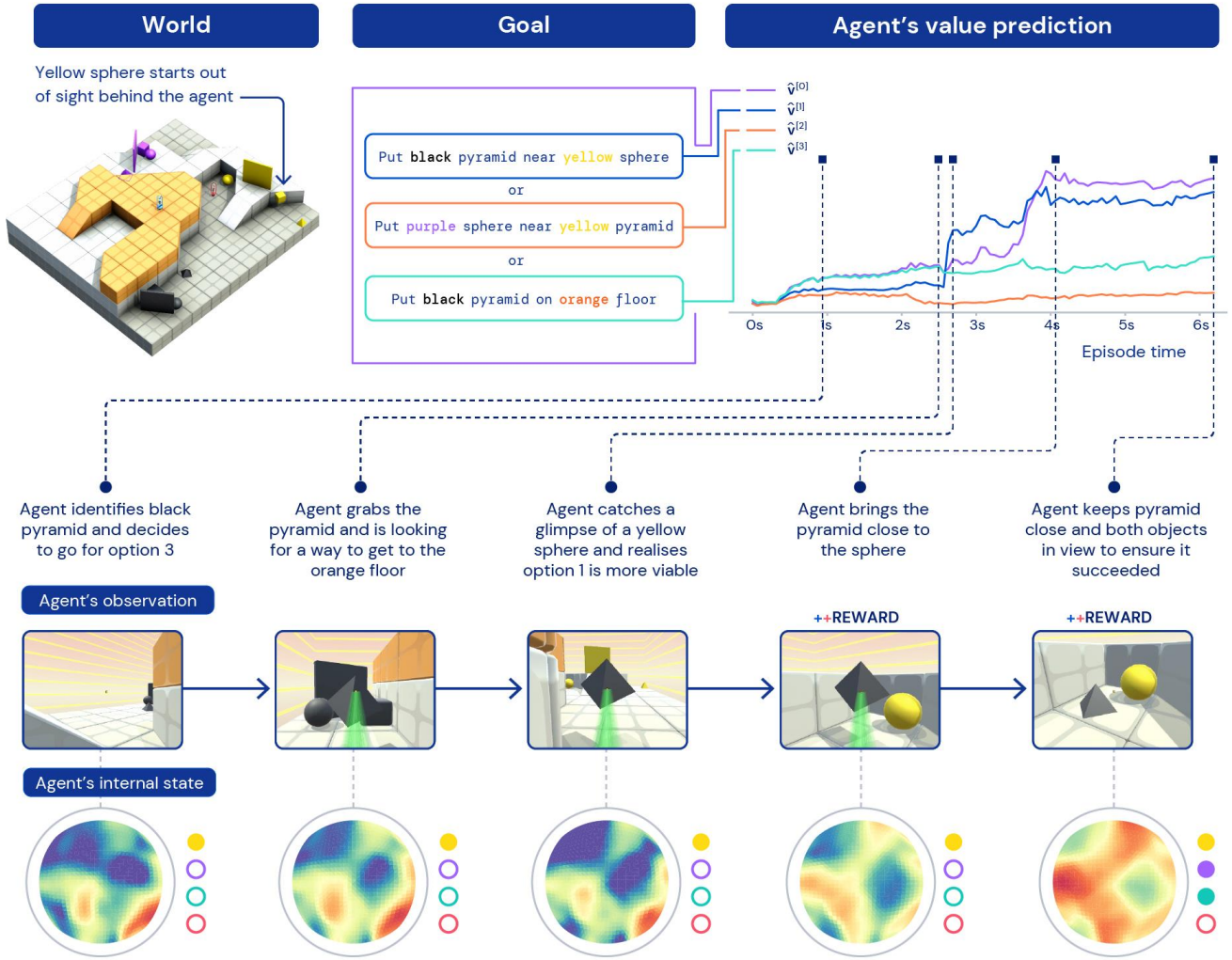


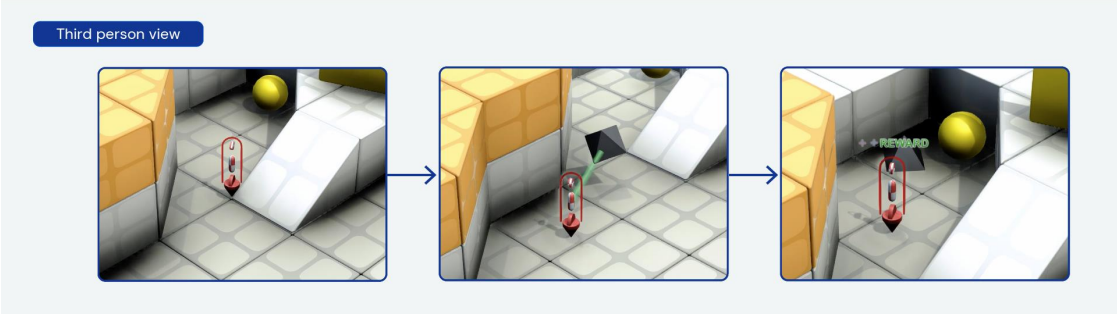
Figure 15 | **(Top)** On the left we see the *learning surface*, showing the progress of a generation 5 agent through time with respect to each of the normalised percentiles. The surface shows the normalised score (height) for each percentile (depth) through training (x-axis). Therefore, the flat bottom of the surface (zero height) is the part of the space where the agent is not participating. On the right, we see an orthogonal projection onto the surface at the end of training. **(Bottom**) We highlight the performance on 6 hand-authored tasks at three points in training, showing how improvements in the normalised percentiles correspond to improvement in these hand-authored tasks.

Example: Changing Goals

Test Example 1: Agent evaluating on the fly and changing its mind

A goal composed of 3 options, each composed of a single predicate

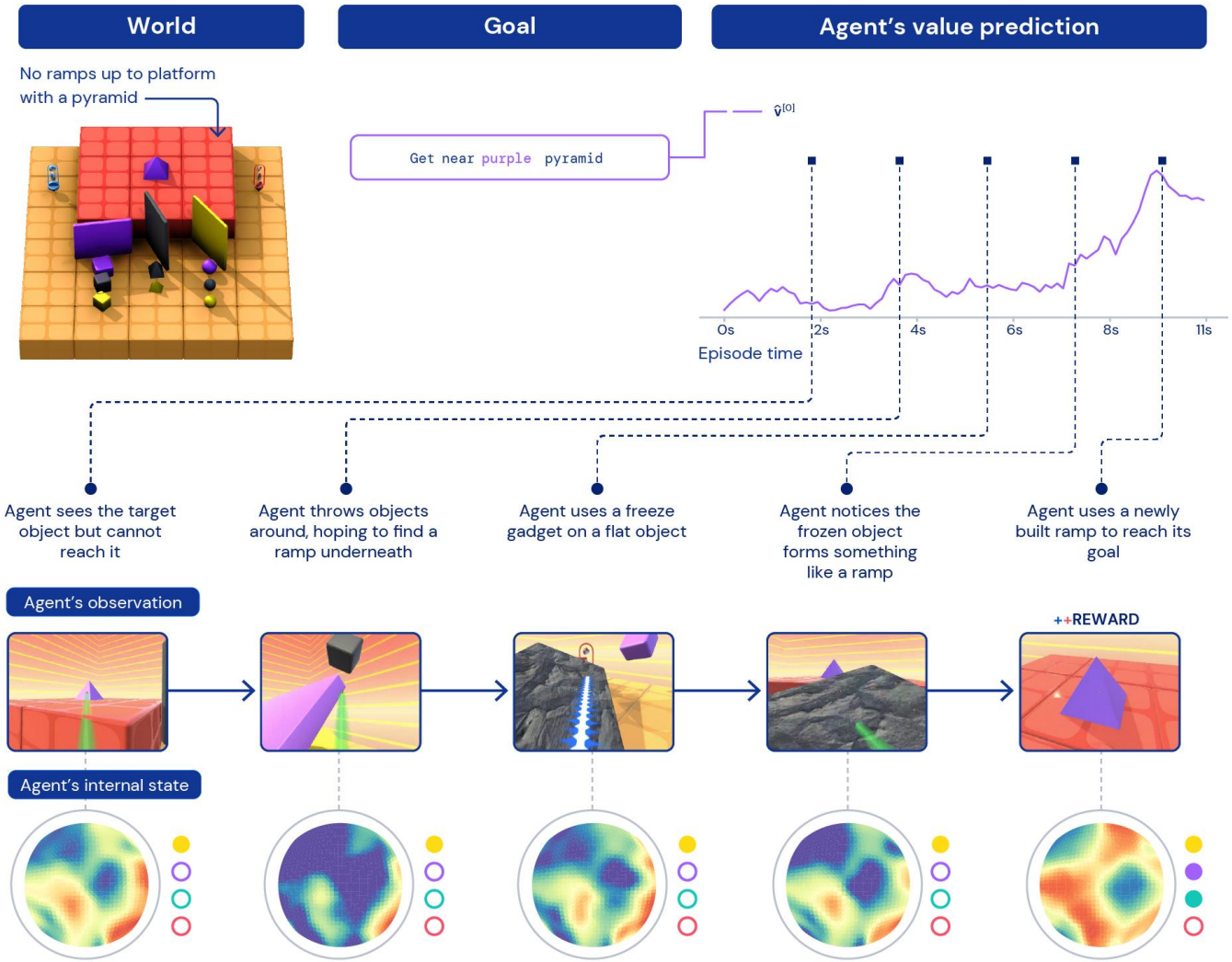


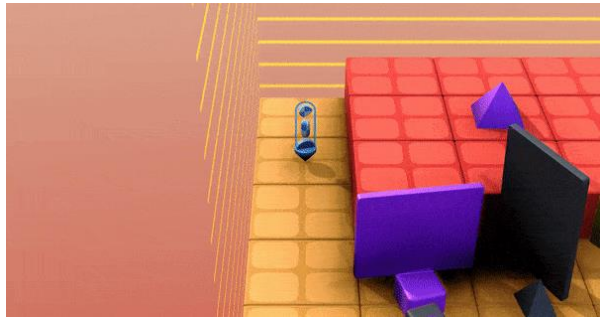


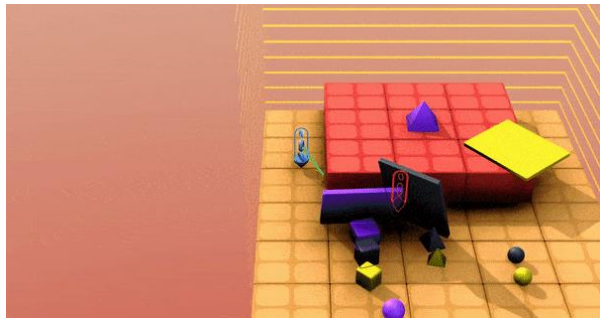
Example: Using Tools

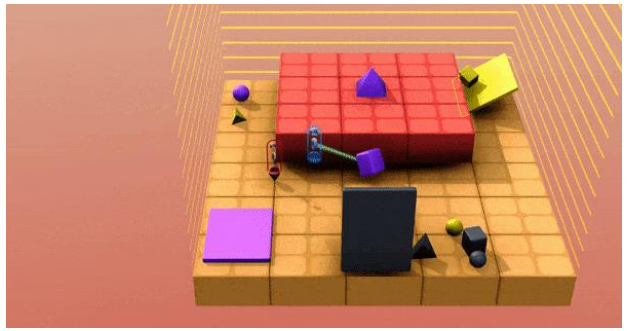
Test Example 2: Agent facing a new challenge and shows tool use

A goal composed of a single option with a single predicate, requiring getting to the purple pyramid but without any static ramps to navigate up the world topology





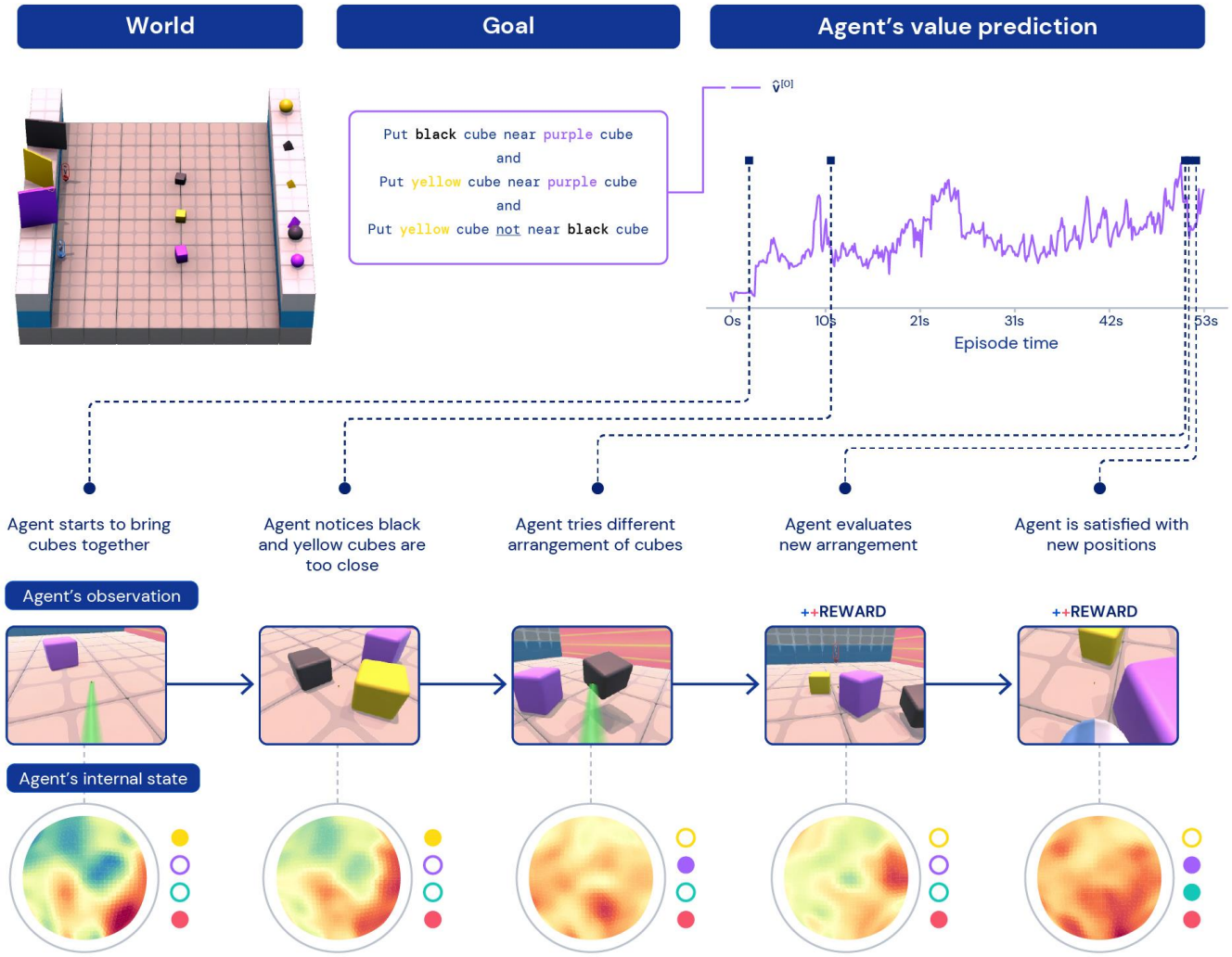


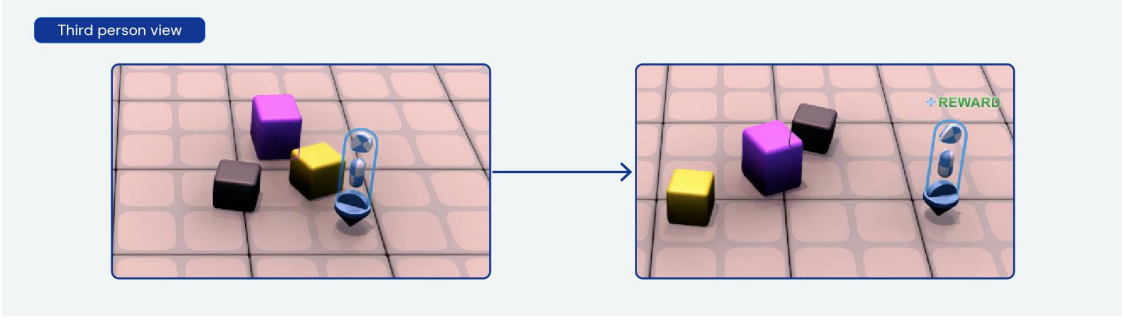


Example: Experimentation

Test Example 3: Agent faces logical puzzle and shows experimentation

A goal composed of a single option with a conjunction of 3 predicates requiring finding the correct physical layout of cubes





Internal Representation

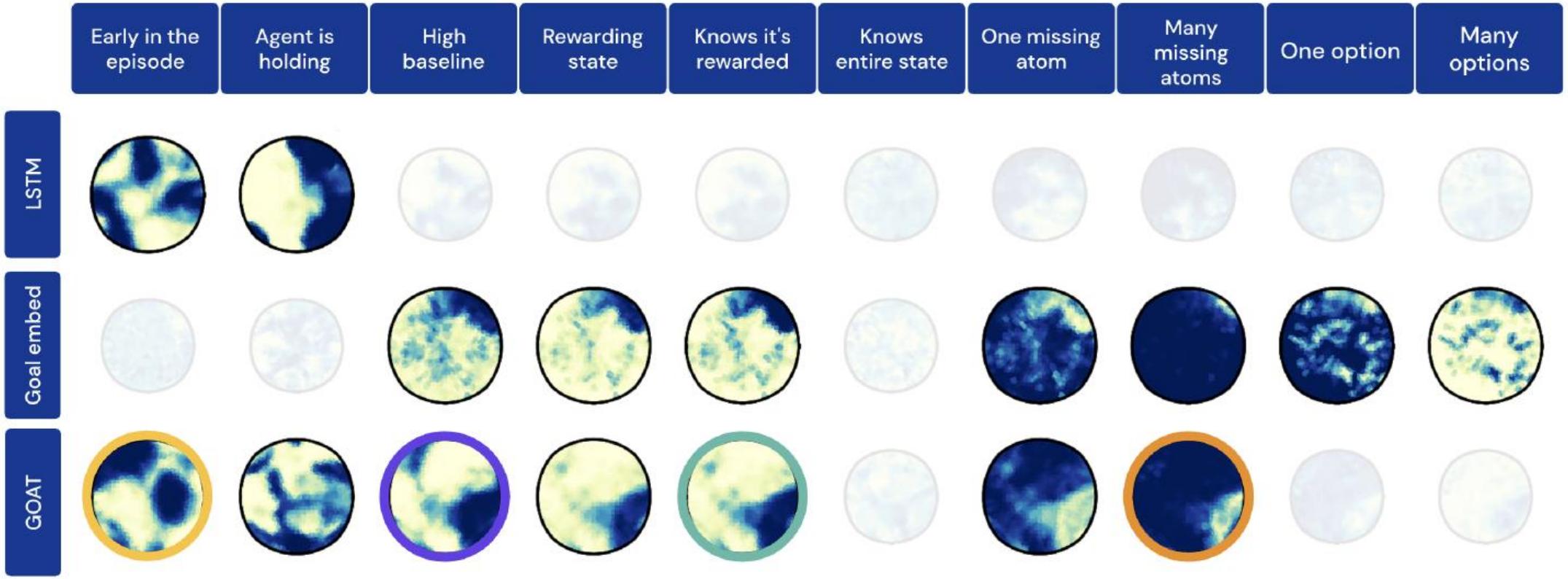


Figure 28 | Internal representation analysis of the agent. We use Kohonen Network representations of various properties for three different modules of the agent (LSTM, goal embedding, GOAT). Within a Kohonen Network, the bright yellow colour denotes states where the property is true, and blue where it is false. We shade out plots which represent combinations of properties and modules where the given property is not represented in a statistically significant manner by the output of the module (see Section 6.6).

Kohonen Networks

Sampled 180,000 states from 30k trajectories.
Mapped onto 900 Kohonen Neurons. (Self-Organizing Map)
Color based on fraction of states that map to a neuron that satisfy a given probe property.

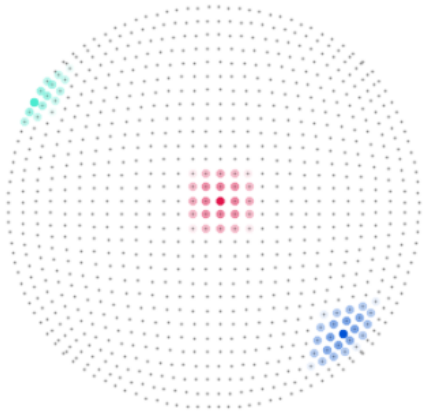


Figure 39 | Visualisation of the Kohonen Network used in our analysis, composed of 900 Kohonen Neurons. Three neurons are called out in colour, with their receptive field (neurons that have non zero update weight) colour-coded with the colour intensity representing the weight.

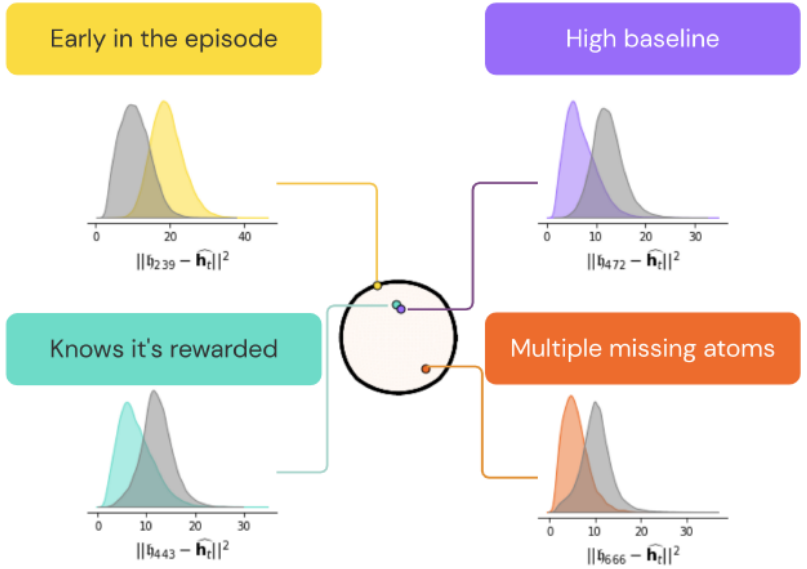


Figure 29 | Internal representation analysis of the agent. The Kohonen Neurons encode four well represented concepts from Figure 28. The kernel density estimation plots represent the density of the activity of the neuron when the concept is true (in colour) or false (in gray).

Failure Cases

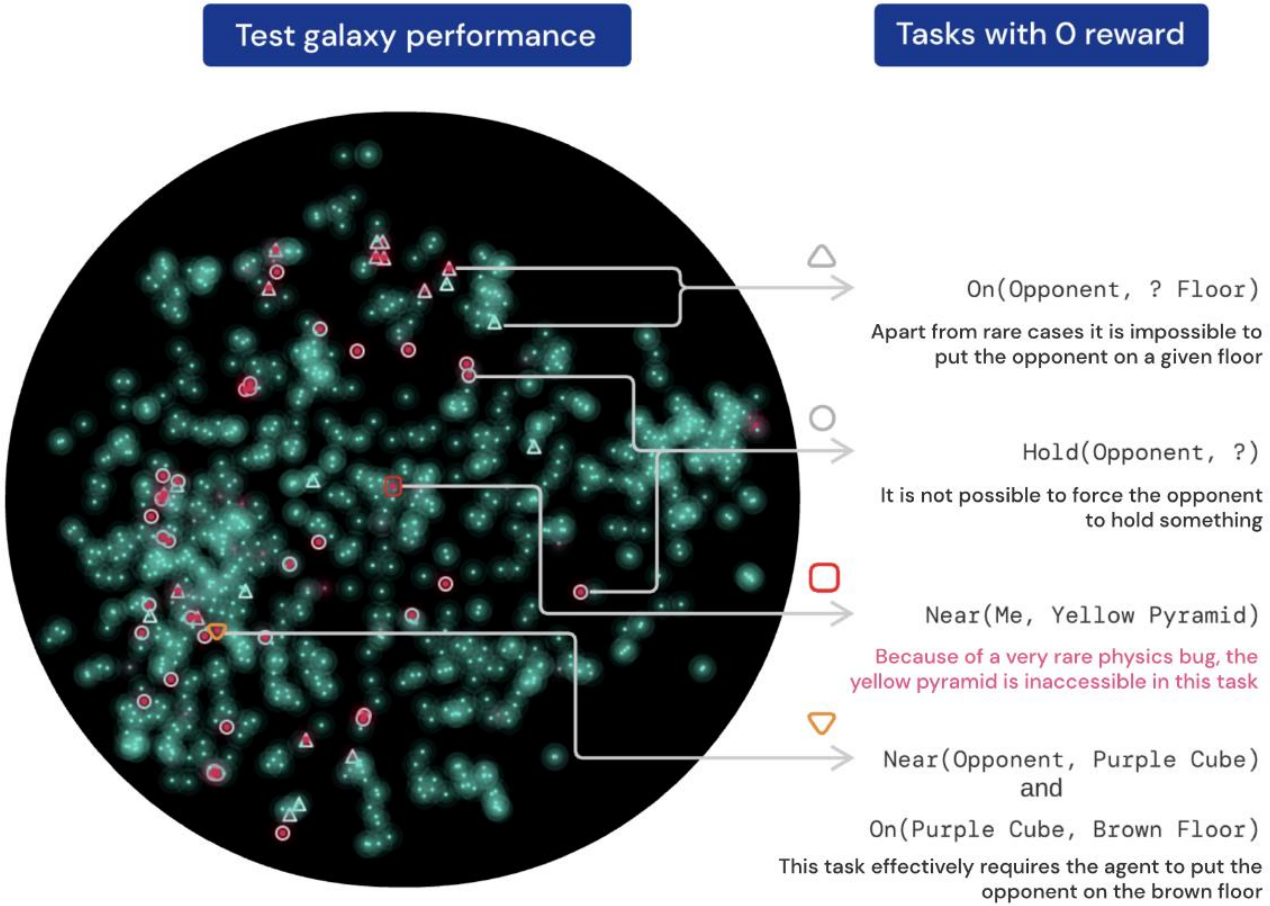


Figure 18 | A visualisation of the test set of tasks, with the corresponding agent performance. The red colour corresponds to a low normalised score and green to a high one. We identify four sources of games the agent scores 0 reward on (listed on the right): 1) tasks that require the agent to put the opponent on a specific floor (marked as triangles in the galaxy); 2) tasks that require the agent to make the co-player hold an object (marked as circles in the galaxy); 3) a single task (in red in the galaxy) which is impossible due to a very rare physics simulation bug; 4) a single task (in orange in the galaxy) that requires the agent to put the co-player on a given floor by a composition of two predicates. After removing these four types of tasks, which cannot be solved even by a human, our agents participate in *every test task*.

Fine Tuning

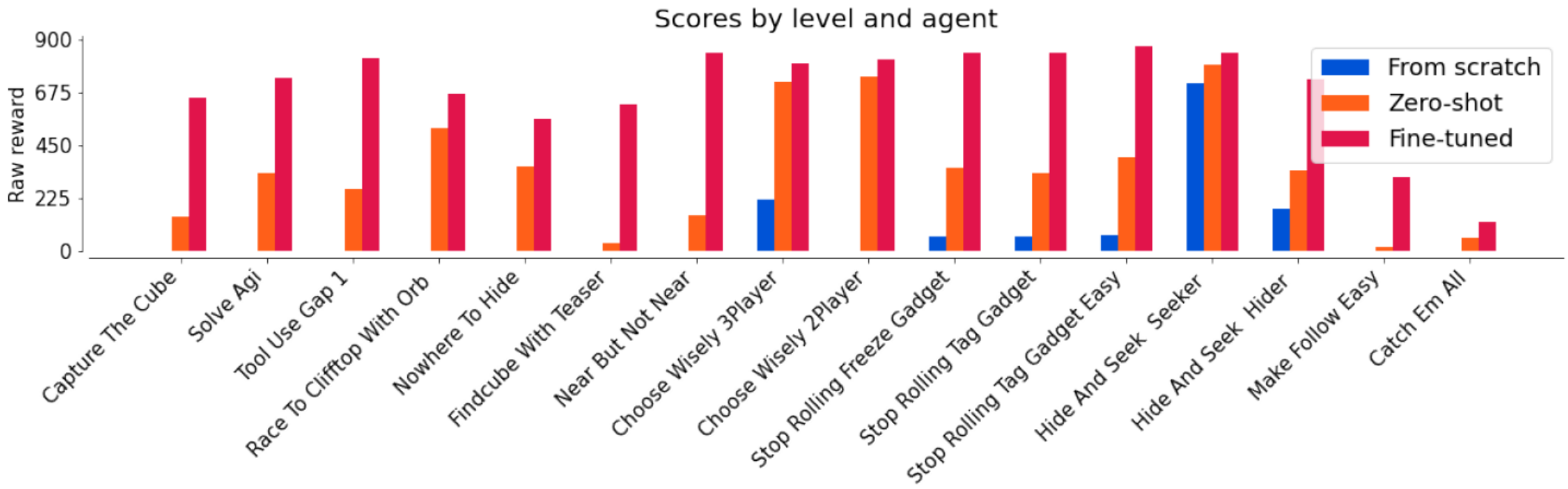


Figure 27 | Comparison of three agents from different training regimes on a range of hand-authored levels. Scratch: An agent trained from scratch for 100 million steps. Zero-shot: the agent trained using our methodology and evaluated on these held out levels zero-shot. Fine-tuned: the same agent but trained for an additional 100 million steps on the level. 100 million steps is equivalent to 30 minutes of wall-clock time in our setup. This rapid finetuning improves the agent score significantly compared to zero-shot, and in the majority of cases training from scratch does not achieve any reward.