Open-Ended Learning Leads to Generally Capable Agents

Open-Ended Learning Team*, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, Nat McAleese, Nathalie Bradley-Schmieg, Nathaniel Wong, Nicolas Porcel, Roberta Raileanu, Steph Hughes-Fitt, Valentin Dalibard and Wojciech Marian Czarnecki
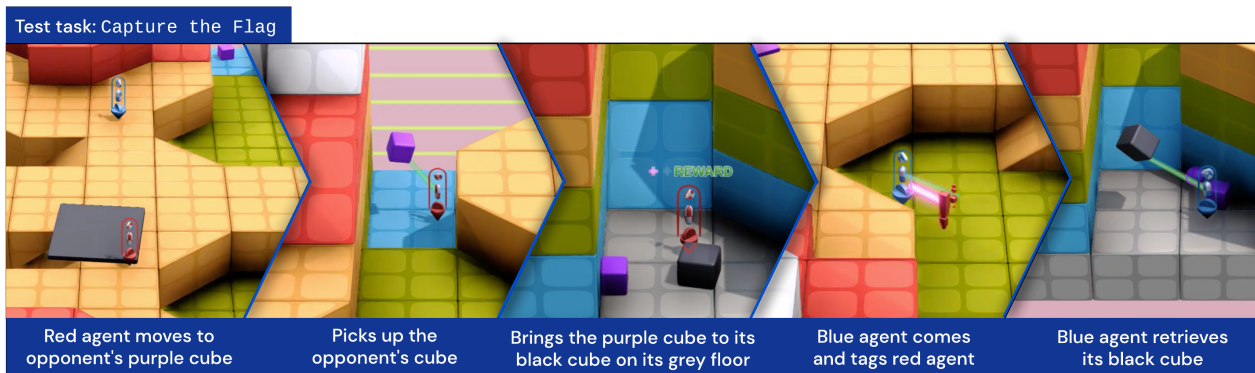
DeepMind, London, UK

Figure 1 | Example zero-shot behaviour of an agent playing a Capture the Flag task at test time. The agent has trained on 700k games, but has never experienced any Capture the Flag games before in training. The red player's goal is to put both the purple cube (the opponent's cube) and the black cube (its own cube) onto its base (the grey floor), while the blue player tries to put them on the blue floor – the cubes are used as flags. The red player finds the opponent's cube, brings it back to its cube at its base, at which point reward is given to the agent. Shortly after, the opponent, played by another agent, tags the red player and takes the cube back.

Artificial agents have achieved great success in individual challenging simulated environments, mastering the particular tasks they were trained for, with their behaviour even generalising to maps and opponents that were never encountered in training. In this work we create agents that can perform well beyond a single, individual task, that exhibit much wider generalisation of behaviour to a massive, rich space of challenges. We define a universe of tasks within an environment domain and demonstrate the ability to train agents that are generally capable across this vast space and beyond. The environment is natively multi-agent, spanning the continuum of competitive, cooperative, and independent games, which are situated within procedurally generated physical 3D worlds. The resulting space is exceptionally diverse in terms of the challenges posed to agents, and as such, even measuring the learning progress of an agent is an open research problem. We propose an iterative notion of improvement between successive generations of agents, rather than seeking to maximise a singular objective, allowing us to quantify progress despite tasks being incomparable in terms of achievable rewards. Training an agent that is performant across such a vast space of tasks is a central challenge, one we find that pure reinforcement learning on a fixed distribution of training tasks does not succeed in. We show that through constructing an open-ended learning process, which dynamically changes the training task distributions and training objectives such that the agent never stops learning, we achieve consistent learning of new behaviours. The resulting agent is able to score reward in every one of our humanly solvable evaluation levels, with behaviour generalising to many held-out points in the universe of tasks. Examples of this zero-shot generalisation include good performance on Hide and Seek, Capture the Flag, and Tag. Through analysis and hand-authored probe tasks we characterise the behaviour of our agent, and find interesting emergent heuristic behaviours such as trial-and-error experimentation, simple tool use, option switching, and cooperation. Finally, we demonstrate that the general capabilities of this agent could unlock larger scale transfer of behaviour through cheap finetuning. A summary blog post can be found here and a video catalogue of results here.

1| Introduction

Over recent years, deep reinforcement learning (deep RL) has repeatedly yielded highly performant artificial agents across a range of training domains (Mirhoseini et al., 2021; OpenAI et al., 2019; Silver et al., 2017). The marriage of expressive neural network architectures, together with scalable and general reinforcement learning algorithms to train these networks, has resulted in agents that can outperform humans on the complex simulated games they were trained on (Mnih et al., 2015). In addition, through *multi-agent* deep RL, agents have also demonstrated impressive robustness to held-out opponents – opponents that were never encountered during training (Jaderberg et al., 2019). Some of the most salient examples include robustness to the top human professional players (Berner et al., 2019; Silver et al.,

---

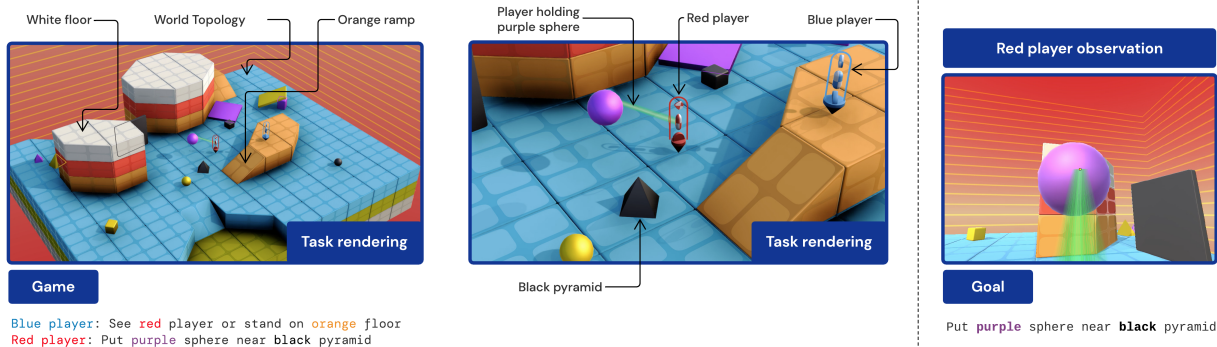*Authors ordered alphabetically by first name. More details in

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.

2016; Vinyals et al., 2019). However, these agents are often constrained to play only the games they were trained for – whilst the exact instantiation of the game may vary (*e.g.* the layout, initial conditions, opponents) the goals the agents must satisfy remain the same between training and testing. Deviation from this can lead to catastrophic failure of the agent.

In this work we move towards creating an artificial agent whose behaviour generalises beyond the set of games it was trained on, an agent which is robust and generally capable across a vast evaluation space of games. By training an agent effectively across a massively multi-task continuum we obtain a neural network policy that exhibits general heuristic behaviours, allowing it to score reward in all humanly solvable tasks in our held-out evaluation task set. In addition, we see the agent being capable in tasks that not only are explicitly held-out from training, but lie far outside of its training distributions, including versions of hide and seek (Baker et al., 2020) and capture the flag (Jaderberg et al., 2019).

To produce a vast and diverse continuum of training and evaluation tasks we develop an environment space, dubbed *XLand*, that permits procedural generation of rich 3D worlds and multiplayer games (described by the goals of the players). These span both two- and three-player tasks, highly competitive and completely cooperative as well as mixtures of both, balanced and imbalanced games, and strategically deep games (*e.g.* Capture the Flag or XRPS, see Section 3.2.3). The capabilities asked of players include visual scene understanding, navigation, physical manipulation, memory, logical reasoning, and theory of mind.

To train agents in this environment space, we first define a multi-dimensional measure of performance, *normalised score percentiles*, which characterises agent performance and robustness across the evaluation task space. We create an open-ended training process to iteratively improve the spectrum of normalised score percentiles. The training process uses deep RL at its core with an attention-based neural network architecture allowing implicit modelling of goals of the game which are provided to the agent. The training

tasks consumed by the agent are dynamically generated in response to the agent's performance, with the generating function constantly changing to keep a population of agents improving across all percentiles of normalised score. This population training is repeated multiple times sequentially, each generation of agents bootstrapping their performance from previous generations with policy distillation, each generation of agents contributing new policies to train against in this multiplayer environment, and each generation redefining the normalised score percentiles as the frontier of performance across task space is advanced. From experimental results we demonstrate the clear benefit of each component of this learning process, with the dynamic task generation being particularly important for learning compared to uniform sampling from task space.

The result of this training process is an agent that is generally capable across the held-out evaluation space. Qualitatively, we observe the agent exhibiting behaviours that are generally applicable, rather than optimal for any specific task. Examples of such behaviours include: experimentation through directed exploration until the agent recognises a rewarding state has been achieved; seeking another player out to gather information of its state irrespective of its goal; and tagging another player if it is holding an object that is related to the agent's goal irrespective of that player's intention. We also probe quantitatively the behaviour of agents in test-time multi-agent situations and see evidence of cooperation emerging with training. In addition to the agent exhibiting zero-shot capabilities across a wide evaluation space, we show that finetuning on a new task for just 100 million steps (around 30 minutes of compute in our setup) can lead to drastic increases in performance relative to zero-shot, and relative to training from scratch which often fails completely.

The paper is organised as follows: first we introduce the XLand environment space in Section 2 followed by an exploration of the quantitative properties of this environment space in Section 3. In Section 4 we introduce the goal, metric, and evaluation space we use to measure progress in the open-ended environment. In Section 5 we detail the
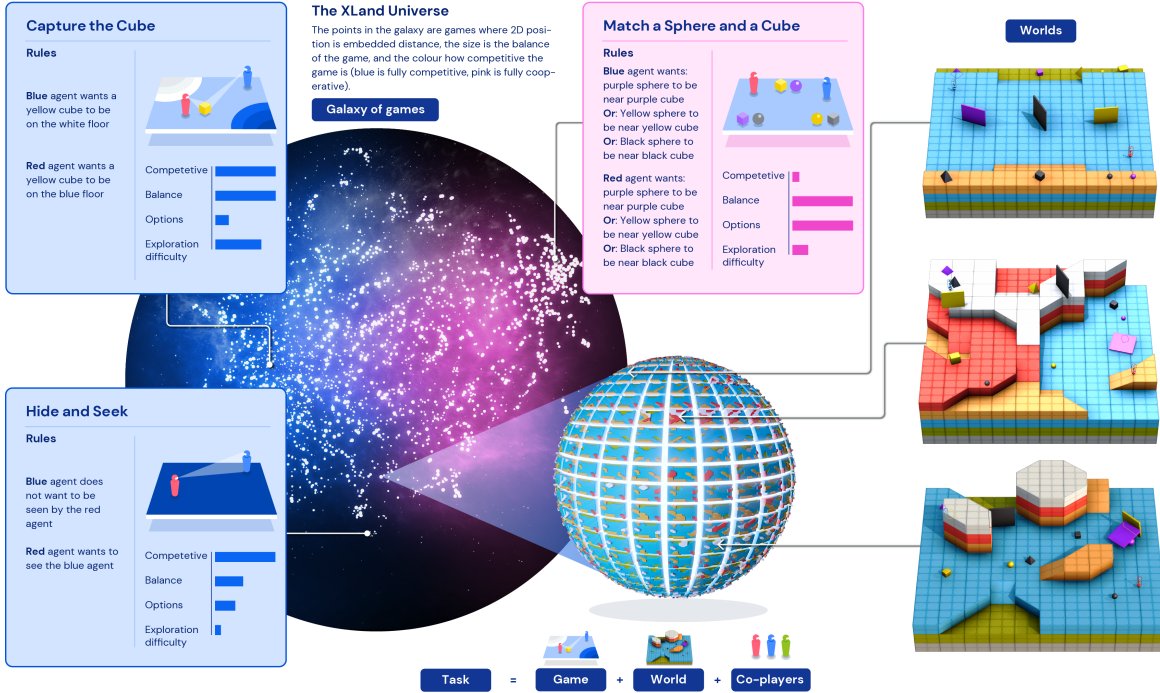
Figure 3 | Visualisation of the XLand environment space. **(Left)** Each dot corresponds to a single *game* and is positioned by a 2D UMAP embedding of distance between games, with the size of the dot corresponding to the *balance* of the game, and the colour representing competitiveness of the game (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.

different components of our learning system and how these work together. Section 6 describes the experimental results, dynamics, and analysis of the produced agent. Finally, Section 7 gives an overview of some related works, followed by the conclusions of this work in Section 8. All proofs and experimental details can be found in the Appendices.

2 | XLand Environment Space

To promote the emergence of general behaviour in reinforcement learning, we seek an environment that exhibits dimensions of consistency across tasks, as well as dimensions of smooth variation.

The development of an environment exhibiting smooth vastness with consistency is central to this work, and as such, we introduce the *XLand* environment space. XLand is a 3D environment consisting of static topology together with dynamic objects simulated by rigid-body physics, with multiple players (controllable by both humans or agents) perceiving first person observations and egocentric movement akin to DM-Lab (Beattie et al., 2016) and Quake III: Arena (Jaderberg et al., 2019). Players have different world-affecting gadgets at their disposal, are able to carry and hold dynamic objects, and receive reward at each timestep based

on the state of the environment: relationships between players, objects, and topology. The environment is developed and simulated using the Unity framework from Ward et al. (2020), with an example task seen in Figure 2.

Consistency in this environment comes from: players always having the same control interface, observation specification, gadget dynamics, and movement dynamics; objects being simulated with similar physical properties; and a limited set of topological building blocks. However, the remainder of the environment properties are vastly but also smoothly variable: the layout and structure of topological building blocks, the positions of objects, the lighting, and crucially the specification of rewarding states for each player. Finally, from a single player's perspective, the policies of the co-players can be vastly but smoothly variable.

The XLand task space, from the perspective of the target player (*e.g.* an agent), denoted as $\aleph$, is a Cartesian product of all possible worlds $\mathbf{w} \in \mathcal{W}$, games $\mathbf{G} \in \mathfrak{G}$ (defined as one goal $\mathbf{g}_i \in \mathcal{G}$ for each of the n players), and the policies $\pi_i \in \Pi$ of each of the remaining $n-1$ players (the players of the game not including the target player). Formally

$$\aleph := \mathcal{W} \times \bigcup_{n=1}^{\infty} \left[\mathcal{G}^n \times \Pi^{n-1} \right].$$

Under this definition, each XLand task

$$\mathbf{x} = (\mathbf{w}, (\mathbf{g}_1, \ldots, \mathbf{g}_n), (\pi_2, \ldots, \pi_n)) \in \aleph$$

can be seen as a regular POMDP over a simulation state space $\mathcal{S}$. For notational simplicity we often refer to the policy of the target player as either π or π_1. At each timestep t, each player π_i receives its player-centric observations $\mathbf{o}_t^i := (f_i(\mathbf{s}_t), \mathbf{g}_i)$, where f_i extracts a pixel-based render of the state of the world from the perspective of player i and also provides the prioperception readings (*e.g.* whether a player is holding something). Note, the reward from the environment is not included in player observations. Based on these observations, an action $\mathbf{a}_t^i$ of each player is sampled from its corresponding policy $\mathbf{a}_t^i \sim \pi_i(\mathbf{h}_t^i)$, where $\mathbf{h}_t^i = (\mathbf{o}_1^i, \ldots, \mathbf{o}_t^i)$ is a sequence of observations perceived so far. The initial state of the simulation is uniquely identified by $\mathbf{w}$. The simulation is terminated after a fixed number of $T = 900$ iterations (two minutes when simulated in real-time). The transition function comes from the simulation's physics engine that calculates the new state $\mathbf{s}_{t+1}$ from its current state $\mathbf{s}_t$ given the simultaneous actions of all the players involved in a specific task $(\mathbf{a}_t^i)_{i=1}^n$, analogously to other multi-agent real-time environments (Berner et al., 2019; Jaderberg et al., 2019; Vinyals et al., 2019). From the perspective of a single player (such as a learning agent), actions of all the co-players can be seen as part of the transition function, and thus the whole process relies only on $\mathbf{a}_t^1$, the action of the target player. The reward function $r_t : \mathcal{S} \to \{0, 1\}$ returns 1 if and only if a player's goal is satisfied in the current simulation state. Consequently, on a given task, a player's goal is to maximise the expected future discounted number of timesteps in which its goal is satisfied

$$\mathbf{V}_\pi(\mathbf{x}) := \mathbb{E}\left[R_\pi(\mathbf{x})\right] = \mathbb{E}\left[\sum_{t=1}^T \gamma^t r_t\right].$$

We will now describe in more detail the makeup of the XLand environment, separating out the initial conditions of the physical environment space, *worlds*, from the specification of rewarding states for each player, *games*. We will highlight the vastness and smoothness of these components of XLand, and finally how these components combine and interact to form a vast and complex space of *tasks*, Figure 3.

2.1 | World Space

Tasks in XLand are embedded within 3D physically simulated worlds, an example of which shown in Figure 2. The layout of the topology, the initial locations of the objects, the initial locations of the players, and the gadgets at each players' disposal are central to the behaviour being asked of a capable player in this task. For example, consider the simple game consisting of a single player, which receives reward when the player is near a purple sphere. If the player is initially located next to the purple sphere, the player needs to simply stand still. If the purple sphere is initially located out of sight of the player, the player must search for the object.

The topology could provide navigational challenges to this search, requiring analysis of connected paths and memory to quickly find the object. The physical interaction between the initial location of the sphere and the topology or other objects could cause the sphere to roll, requiring the player to intercept the sphere once it is found, and if the player has a freeze gadget this would allow the player to stop the sphere rolling by freezing its motion.

The initial condition of a simulated world defines the possible challenges faced by a player somewhat independently of the game, the goals of the players. As such, we define the world $\mathbf{w}$ as the initial state of the simulated 3D world and its constituents, the state at the beginning of each episode of play. The three main components of a world are the topology, objects, and players. Worlds are procedurally generated (Shaker et al., 2016).

Topology A world in XLand contains a static topology which defines the unmovable landscape that is navigated by the players, surrounded by four walls which enclose the rectangular playable area, with variable lighting conditions. The topology is generated by first selecting a rectangular size of the world which encloses a grid, and subsequently placing a number of predefined 3D topological tiles. These tiles can be placed in any arrangement but cannot violate local neighbourhood connectivity constraints, ensuring that the arrangement of 3D tiles forms congruent and connected playable regions.

Objects Objects are elements of XLand worlds that are dynamic – they undergo physics simulation and can be manipulated by players. Each world defines a specified initial location for each movable object as well as its orientation, shape, colour and size. Object instances vary in size, colour, and shape. There are three colours – black, purple, yellow – and four shapes – cube, sphere, pyramid, slab.

Players The players of the game, which can be controlled by agents, are given initial positions in the same manner as objects. Players are coloured, and in this work we consider up to three players, each being assigned a unique colour of either blue, red, or green. In addition, each player is assigned a gadget: either the freeze gadget or the tagging gadget. The freeze gadget can be used by a player only on an object and has the effect of freezing the dynamics of the object so that it remains static and unmovable for 5 seconds, before becoming dynamic again and undergoing physics simulation as normal. The tagging gadget can be used by a player on an object or another player and has the effect of removing the object or player from the world for 3 seconds, before the object or player is returned to the world at its initial location, rather than the location at which it was removed.

An instance of a world $\mathbf{w}$ is therefore a particular topology, combined with a set of objects with locations, and a particular set of players with locations and gadgets. An agent playing in a world $\mathbf{w}$ will always experience identical initial conditions.

For notation simplicity we will omit the dependence of all returns/values on the discount factor value γ.

Our process of generating worlds leads to a vast and smooth space of worlds, with these properties explored further in Section 3.1. More details of this process can be found in Section A.1.1 and Figure 32.

2.2| Game Space

Whilst a world defines the initial state of the simulated physical space for the players to act in, a task requires a *game* for these players to act towards. A game **G** consists of a goal $\mathbf{g}_i \in \mathcal{G}$ for each of the n players, $\mathbf{G} = (\mathbf{g}_1, \dots, \mathbf{g}_n)$. A goal defines the reward function for the associated player, and each player is tasked with acting in a way to maximise their total reward, while perceiving only their own goal (and not seeing goals of the co-players).

The state of our simulated environment $\mathbf{s} \in \mathcal{S}$ describes the physical world the players interact with. **s** consists of the positions of all the objects, players, their orientations, velocities, *etc*. We define a set of atomic predicates $\phi_j : \mathcal{S} \to \{0, 1\}$ in the form of a physical relation applied to some of the entities present in the state. These relations include: being near, on, seeing, and holding, as well as their negations, with the entities being objects, players, and floors of the topology. An example predicate could be `near(purple sphere, opponent)`, which is going to return 1 if and only if one of the co-players is currently close to a purple sphere. With the set of possible predicates fixed, a goal of a player can be represented by a set of options (disjunctions) over sets of necessary predicates for this option (conjunctions). Consequently, an example goal could look like

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

which, for some example predicates, could mean *"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})"*. This is a canonical representation of Boolean formulas, the *disjunctive normal form* (DNF), which can express any Boolean formula (Davey and Priestley, 2002). The corresponding reward function $r_{\mathbf{g}}(\mathbf{s})$ follows the transformation of disjunctions becoming sums, and conjunctions becoming products, *i.e.* for a goal $\mathbf{g} := \bigvee_{i=1}^{k}[\bigwedge_{j=1}^{n_i}\phi_{ij}]$:

$$r_{\mathbf{g}}(\mathbf{s}) = \max_{i=1}^{k}\left[\min_{j=1}^{n_i}\phi_{ij}(\mathbf{s})\right] = \min\left(\sum_{i=1}^{k}\prod_{j=1}^{n_i}\phi_{ij}(\mathbf{s}), 1\right).$$

A simple example in our game space would be the game of hide and seek. The two-player version of the game consists of two goals ($\mathbf{g}_{\text{seek}}, \mathbf{g}_{\text{hide}}$) where the goal of one player consists of just one option of one predicate, $\mathbf{g}_{\text{seek}} = \phi_{\text{seek}} = $ `see(me, opponent)`, and the goal of the co-player is $\mathbf{g}_{\text{hide}} = \phi_{\text{hide}} = $ `not(see(opponent, me))`.

This general construction of games allows us to represent a vast number of highly diverse games, ranging from simple games of finding an object to complex, strategically deep games. Importantly, the space of games is also smooth, allowing for gradual transition between games. These properties are explored in Section 3.2.

2.3| Task Space

A task in XLand **x** is the combination of a world **w**, a game **G** and the policies of the *co-players* ($\pi_2, \dots, \pi_n$). With this view, despite its clearly multi-agent nature, we can view each task as a standard single-player problem for π_1.

The combination of a world, a game, and co-players can interact in complex ways to shape the space of optimal behaviours required of the player. Consider the example game where the player has a goal consisting of two options *"Hold a purple sphere **or** hold a yellow sphere"* and there is one co-player with the identical goal. If the game is played in a fully open world where initially both rewarding objects are visible, the challenge to obtain the optimal behaviour is to choose to navigate to the closest object. If the paths to each object are occluded along the route, the optimal behaviour might require memory to reach its goal object, remembering the path to take. If the world is such that only one of the objects is initially visible but out of reach on a higher floor, the optimal behaviour may be to manipulate another object to reach the goal object. Now consider the variation of co-player policies. If the co-player picks up the purple sphere and moves away quickly, the optimal behaviour of the player may be to ignore the purple sphere and navigate to hold the yellow sphere. However, if the co-player seeks out the player and uses its tagging gadget on sight, hindering the player's ability to navigate to the goal object, the optimal behaviour of the player may be to avoid being seen or to tag the co-player itself, before navigating to a goal object.

A result of this complex interaction is that the cross product of a set of worlds, games, and co-player policies creates a set of tasks with challenges – optimal behaviours of the player – which is larger than the sum of the number of worlds, games, and co-player policies.

3| Environment Properties

The previous section introduced the XLand environment space and its tasks' construction from worlds, games, and co-players. In this section we analytically and empirically explore some of the properties of this space, focusing on world and game properties independently. In both cases we explain how these components give rise to the properties of vastness, diversity, and smoothness.

3.1| World Properties

The worlds are high dimensional objects consisting of topology, object locations, and player locations. To highlight the characteristics of worlds, we can describe a world in terms of the navigational challenges it poses due to the topology and the objects.

Our worlds are all grid-aligned, with varied dimensions of each single *tile*, some of which (ramps) one can use to navigate to a higher level. We consider two world representations: first, the height map, $\tau(\mathbf{w}) : \mathcal{W} \to [0, 1]^{w \times h}$ where w, h are the width and height of the world respectively, and each element in $\tau(\mathbf{w})$ is the height of the top of the tile at the location of the element. The second representation is a
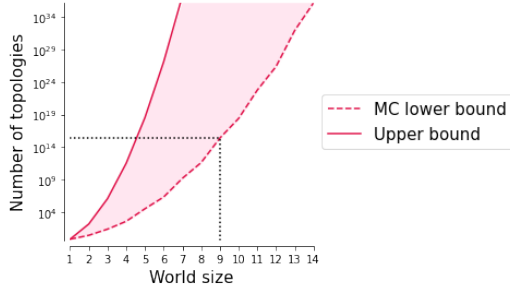
Figure 4 | Visualisation of the bounds of the number of possible world topologies of shape (n, n) as a function of a world size n. See Section A.1.2 for details.

world topology graph, representing navigation paths.

Definition 3.1 (World topology graph). *For a given world* **w**, *we define a directed graph* $G_{\mathbf{w}} = (V_{\mathbf{w}}, E_{\mathbf{w}})$ *where each tile of a world is represented as a vertex, and an edge exists between two vertices* v_i *and* v_j *if and only if it is possible for a player to travel between the two neighbouring tiles in a straight line (they are on the same level, the height of* v_j *is lower so the agent can fall to it, or* v_j *is an accessible ramp leading to a higher level).*

Given this graph, we can define various proxy measures of navigational complexity by looking at the distribution of paths between every pair of vertices.

Definition 3.2 (Shortest paths distribution). *For a given* **w** *we define* $\rho_{\mathrm{sp}}(\mathbf{w})$ *as a distribution of lengths of shortest paths between every pair of vertices in* $G_{\mathbf{w}}$.

Definition 3.3 (Resistance distances distribution). *For a given* **w** *we define* $\rho(\mathbf{w})$ *as a distribution of resistance distances (Klein and Randić, 1993) between every pair of vertices in* $G_{\mathbf{w}}$, *where a resistance distance between* v_i *and* v_j *is given by* $\Gamma_{ii} + \Gamma_{jj} - \Gamma_{ij} - \Gamma_{ji}$ *for* $\Gamma = (L + \frac{1}{w \cdot h} \mathbf{1}_{w \cdot h \times w \cdot h})^{\dagger}$, *L being the Laplacian matrix of* $G_{\mathbf{w}}$ *and* $\dagger$ *being the Moore-Penrose pseudoinverse (Penrose, 1955).*

3.1.1 | World Vastness

Let us start by discussing the vastness of worlds by looking at how many topographies are possible in XLand. In principle, every combination of tiles could be utilised, creating $N_{\mathrm{floors}}^{w \cdot h} \cdot N_{\mathrm{tiles}}$ possibilities. However, as discussed previously, constraints on tile placements exist to ensure ramps connect levels and there are accessible playable regions. Consequently, it is reasonable to count the number of world topologies where all ramps are properly connected, and that have at least 50% of the world fully accessible (there exists a way to go from every point to any other point within the accessible area). We estimate a lower bound to this quantity with Monte Carlo sampling, and present results in Figure 4 (see Section A.1.2 for details). For worlds of size 9 by 9 tiles, $(9, 9)$, we have more that 10^{16} unique topologies (corrected for 8 possible symmetries) – a vast space of worlds.

3.1.2 | World Smoothness

We hypothesise that small changes in the world topography lead to small changes in the overall navigational complexity.
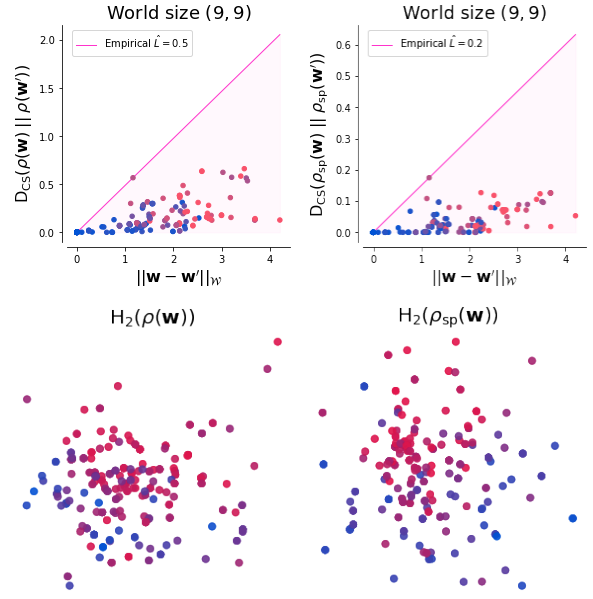

Figure 5 | An empirical visualisation of the world space smoothness. We take a set of worlds of size $(9, 9)$ and then apply local mutations up to 30 times. **(Top)** Each dot on the plot represents one pair of mutated worlds, with the x-axis showing the L_2 distance in tile space, and the y-axis showing the Cauchy-Schwarz Divergence between distributions of reachability graph resistances $\rho(\mathbf{w})$ (left) and shortest path distances $\rho_{\mathrm{sp}}(\mathbf{w})$ (right). The pink line represents the empirical smoothness coefficient. The colour of each dot encodes the number of mutations between the pair of worlds, from 1 (blue) to 30 (red). **(Bottom)** We linearly embed each of the worlds, trying to find a linear projection where the entropy of the corresponding distribution (in colour) can be well described by a distance from the center of the projection. One can see how small changes in the world space (position) lead to small deviations of the entropy (colour).

To formalise this claim we take a set of 9 by 9 worlds, and then apply local changes to each of them, in the form of moving tiles around, changing floors, *etc.* Given this set of mutated worlds, we plot the relation between the change in the topography

$$\|\mathbf{w} - \mathbf{w}'\|_{\mathcal{W}} := \sqrt{\sum_{i,j=1}^{w,h} [\tau(\mathbf{w})_{ij} - \tau(\mathbf{w}')_{ij}]^2}$$

and the Cauchy-Schwarz Divergence (Nielsen, 2012)

$$D_{\mathrm{CS}}(p, q) := -H_2(p) - H_2(q) + 2H_2^{\times}(p, q)$$
$$:= \log \int p^2(x)dx + \log \int q^2(x)dx$$
$$- 2 \log \int p(x)q(x)dx,$$

between the corresponding shortest paths distributions ρ_{sp} and resistance distances distributions ρ. The top row of Figure 5 shows that there is a visible linear bound in the change in the paths distributions, suggesting L-Lipschitzness.

To further confirm this claim, we take the same set of worlds and find a linear projection (Section A.1.3) that embeds our worlds in a 2-dimensional space, with each point coloured by its corresponding Renyi's quadratic entropy

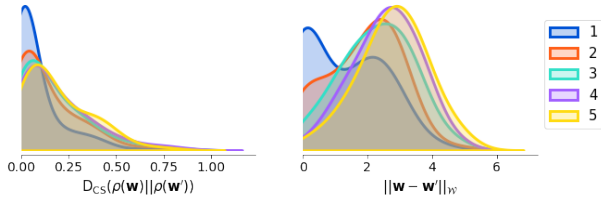Figure 6 | An example array of worlds from the XLand environment space.

Figure 7 | The distributions of distances between two worlds, $\mathbf{w}$ and $\mathbf{w}'$, with different number of local tile mutations between them (colour). The distances are the navigational resistance (left) and topology distance (right). With only a few mutations the characteristics of the world can change a lot.

H_2 of the distribution of paths over its navigation graph, Figure 5 (bottom). We can see that the world space appears smooth.

3.1.3 | World Diversity

The world topology, jointly with object and player positions, allow one to express arbitrary navigation challenges, including various types of mazes, but also complex maps with difficult to access regions, and occluded visibility similar to the maps used in competitive first-person video games (Jaderberg et al., 2019), see Figure 6.

To illustrate diversity, one can see that the Cauchy-Schwarz Divergence between resistance distances distributions ρ as well as topology distances can be increased with relatively few local tile mutations (see Figure 7). This confirms that, despite being a relatively smooth space, our world space spans diverse worlds, which can be found with local search methods (*e.g.* evolutionary algorithms).

3.2 | Game Properties

Once multiple goals are combined to form a game, new complexity emerges – the ways in which the objectives of players interact, compete, and affect each other. This complex interaction is central to the fields of Game Theory and multi-agent

RL (Balduzzi et al., 2019; Shoham et al., 2007). To characterise the properties of games, we focus our analysis on three dimensions of games: the number of options, exploration difficulty, and their cooperative/competitive/balance aspects.

The first property is the number of options in a given goal (and a game). Having multiple options for getting a reward in each timestep encourages players to be constantly evaluating the environment state, assessing which option is the more rewarding one.

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

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

This distance between two goals is the fraction of different predicate evaluations where one goals is rewarding, and the other goal is not. Analogously, between two games

$$\|\mathbf{G}_i - \mathbf{G}_j\|_{\mathfrak{G}} := \frac{1}{n} \sum_{k=1}^{n} \|(\mathbf{G}_i)_k - (\mathbf{G}_j)_k\|_{\mathcal{G}} \in [0, 1].$$

This leads to the following observation.

Observation 3.1. $(\mathcal{G}, \|\cdot\|_{\mathcal{G}})$ *and* $(\mathfrak{G}, \|\cdot\|_{\mathfrak{G}})$ *are metric spaces.*

In particular we have

$$\mathbf{g}_i \equiv \mathbf{g}_j \iff r_{\mathbf{g}_i} = r_{\mathbf{g}_j} \iff \|\mathbf{g}_i - \mathbf{g}_j\|_{\mathcal{G}} = 0$$

$$\mathbf{G}_i \equiv \mathbf{G}_j \iff \|\mathbf{G}_i - \mathbf{G}_j\|_{\mathfrak{G}} = 0.$$

This allows us to define the next game property: exploration difficulty.

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

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

One simple interpretation of this quantity is: assuming each of the predicates is independent and equally probable to be (dis-)satisfied at a given timestep, then $1 - \kappa(\mathbf{g})$ describes the probability of at least one player getting a reward. Consequently, we will refer to goals as trivial if $\kappa(\mathbf{g}) \in \{0, 1\}$, since these are goals where every policy is an optimal policy (similarly we say a game is trivial from the perspective of the main agent if $\kappa(\mathbf{g}_1) \in \{0, 1\}$).

Proposition 3.1. *For every goal $\mathbf{g}$ where $\kappa(\mathbf{g}) = 0$ or $\kappa(\mathbf{g}) = 1$ every policy is optimal.*

Given exploration difficulty, we now define a new property – the notion of cooperativeness – that will assign a number between 0 and 1 to each game, where a game of cooperativeness 1 is going to be one where all players always get rewards jointly, and cooperativeness 0 when they can never both get a reward at the same timestep.

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.*

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

Symmetrically, competitiveness can be expressed as $\text{comp}(\mathbf{G}) = 1 - \text{coop}(\mathbf{G})$ or more explicitly with the following definition.

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.*

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

Finally, let us introduce the property of balance of a game. In game design, the issue of one player of the game having a constant advantage is a common one, referred to as an imbalance. Whilst fully symmetric, simultaneous moves games are fully balanced by construction, it is a complex problem to assess the degree of balance when the game is not symmetric, *i.e.* when the goals of each player are different.

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

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

With the above definition it is easy to note that when $\Xi = \{\text{identity}\}$ then balance is equivalent to cooperativeness. Consequently, balance can be seen as a relaxation of the notion of cooperation, under the assumption that some aspects of game rules are equivalent (equally hard). For XLand we note that colours of objects should have negligible effect on the complexity of a task, meaning that satisfying a predicate `hold(me,yellow sphere)` should be equally hard as `hold(me,purple sphere)`. Consequently, we use Ξ to be the set of all bijective recolourings of objects in goals that are consistent across the entire goal.

3.2.1 | Game Vastness

Let us denote the number of unique atomic predicates as n_a, the number of options a goal consists of as n_o and the number of predicates in each option as n_c. There are exactly $n_a^{n_c \cdot n_o}$ goals that differ in terms of their string representation, however many goals are equivalent such that

$$\mathbf{g}_i \equiv \mathbf{g}_j \iff r_{\mathbf{g}_i} = r_{\mathbf{g}_j}.$$

For example, the goal of *seeing a purple sphere or not seeing a purple sphere* is equivalent to the goal of *holding a yellow cube or not holding a yellow cube*, both corresponding to $r(\mathbf{s}) = 1$. Counting the exact number of unique r functions that emerge from n_i options each being a conjunction of n_c out of n_a predicates is a hard combinatorial problem, but under the assumption that each atomic predicate (apart from their negations) is independently solvable we can provide a lower bound of the number of unique goals.

Theorem 3.1. *Under the assumption that each atomic predicate that does not involve negation is independently solvable, the number of unique n-player games $N_{\mathfrak{G}}$ with respect to the reward functions they define satisfies:*

$$\frac{1}{n!} \left[\frac{1}{n_o!} \prod_{i=1}^{n_o} \left(\binom{n_a/2 - i \cdot n_c}{n_c} 2^{n_c} \right) \right]^n \leq N_{\mathfrak{G}} \leq \binom{n_a}{n_c}^{n \cdot n_o}.$$

Figure 8 shows these bounds as functions of the number of options, atoms and conjunctions. As an example we see that with 3 options, each a conjunction of 3 predicates, using a set of 200 atomic predicates (the approximate number available in practice) gives us more than 10^{37} unique 2-player games (composed of more than 10^{18} goals) – a vast space of games.

3.2.2 | Game Smoothness

For our XLand task space to exhibit smoothness, the game space itself must be smooth: if we change our games by a small amount, the game properties of interest should only change by a small amount. We show that the interesting properties of our games are L-Lipschitz functions.

Proposition 3.2. *Exploration difficulty is a 1-Lipschitz function, meaning that for any $\mathbf{G}_i, \mathbf{G}_j$ we have*

$$\|\kappa(\mathbf{G}_i) - \kappa(\mathbf{G}_j)\| \leq \|\mathbf{G}_i - \mathbf{G}_j\|_{\mathfrak{G}}.$$

Theorem 3.2. *$\text{coop}((\cdot, \mathbf{g}'))$ is a $\frac{1}{1-k}$-Lipschitz function wrt. $\|\cdot\|_{\mathcal{G}}$ for any $\mathbf{g}$ such that $\kappa((\mathbf{g}, \mathbf{g}')) = k$.*
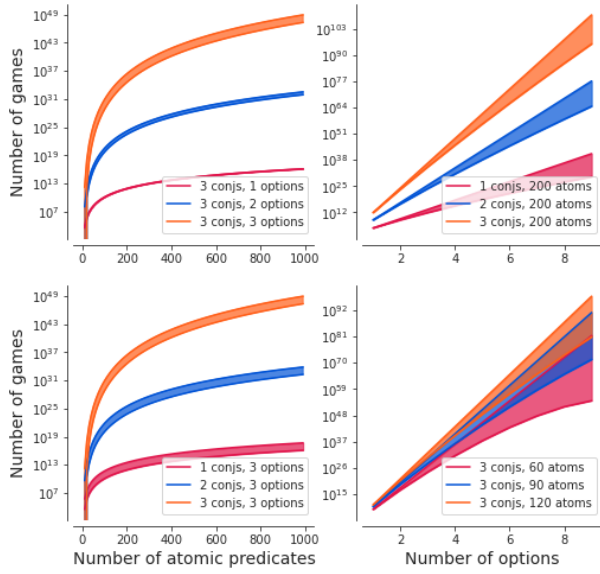
Figure 8 | Bounds on the number of 2-player games provided by Theorem 3.1 as functions of the number of options and atomic predicates. Our evaluation space (Section 4.2) spans approximately 200 atomic predicates and up to 3 options.

In a natural way the same is true for competitiveness.

Observation 3.2. $\text{comp}((\cdot, \mathbf{g}'))$ *is a* $\frac{1}{1-k}$-*Lipschitz function wrt.* $\| \cdot \|_{\mathcal{G}}$ *for any* $\mathbf{g}$ *such that* $\kappa((\mathbf{g}, \mathbf{g}')) = k$.

Therefore, if we change one of the goals by a small amount, we have an upper bound on the change in exploration difficulty, cooperativeness, and competitiveness of the whole game.

Figure 9 verifies these properties empirically by showing the relation between the distance in game space compared to the change in competitiveness and exploration difficulty. We also provide a 2D projection of sample games using PCA, showing that these two properties are visible, suggesting they explain a lot of variance in game space. These examples show analytically and empirically the smoothness of game space.

3.2.3 | Game Diversity

We have shown that the game space consists of vastly many games, and that small changes in their definitions lead to small changes in properties of interest. One missing aspect is to show how diverse this game space is, that eventually, after taking many small steps, one can change a game into a wildly different one.

Theorem 3.3. *For every two player game* $\mathbf{G}$ *such that* $\hat{\kappa}(\mathbf{G}) = k$ *and a desired change in competitiveness* $m \in (-\text{comp}(\mathbf{G}), 1 - \text{comp}(\mathbf{G}))$ *such that* $k|m| \in \mathbb{N}$ *there exists a* $\mathbf{G}'$ *such that* $\text{comp}(\mathbf{G}') = \text{comp}(\mathbf{G}) + m$ *and* $\|\mathbf{G} - \mathbf{G}'\|_{\mathfrak{G}} \leq \frac{k|m|}{2}$.

To see qualitatively the diversity of games, we present a few examples of games showcasing a range of challenges imposed on players.
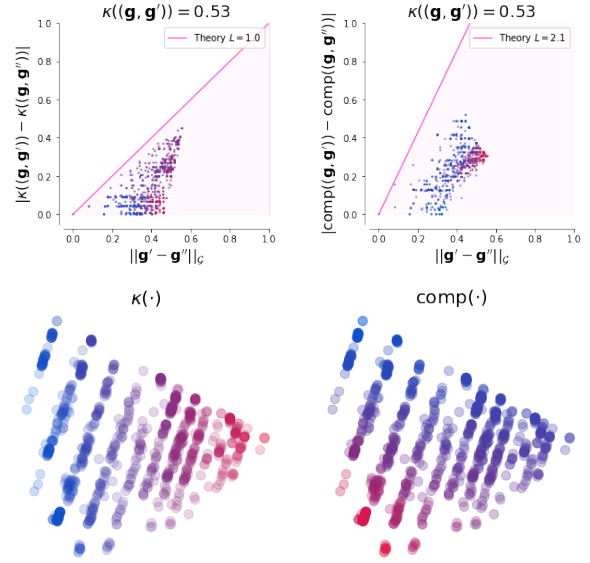
Figure 9 | Empirical confirmation of game space smoothness with respect to exploration difficulty (κ, left) and competitiveness (comp, right). We took a single game, then created 1000 similar games by performing simple edits on one of its goals ($\mathbf{g}'$) creating a new one ($\mathbf{g}''$)– removal of an option/relation, adding a new option/relation, substitution of a relation, *etc*. **(Top)** Each of these games corresponds to a point, with the x-axis being its distance from one randomly selected anchor game ($\mathbf{G}, \mathbf{G}'$) (with the exploration difficulty reported in the title), and on the y-axis the difference in its comp or κ (colour corresponds symmetrically to κ on comp plot and vice versa). The pink line is the upper bound from Proposition 3.2 and Theorem 3.2. **(Bottom)** The matrix of pairwise distances between these games is computed, and PCA used to embed them on a plane, followed by representing comp and κ with a point colour. In both cases one can see very smooth transitions.

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.

$$\mathbf{g}_1 := \texttt{hold(me, yellow sphere)}$$
$$\mathbf{g}_2 := \texttt{near(me, yellow pyramid)}$$
$$\kappa(\mathbf{G}) = \tfrac{1}{4} \quad \text{comp}(\mathbf{G}) = \tfrac{2}{3} \quad \text{bal}(\mathbf{G}) = \tfrac{7}{15}$$

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.

$$\mathbf{g}_1 := \texttt{near(yellow pyramid, yellow sphere)}$$
$$\mathbf{g}_2 := \texttt{near(yellow pyramid, yellow sphere)}$$
$$\kappa(\mathbf{G}) = \tfrac{1}{2} \quad \text{comp}(\mathbf{G}) = 0 \quad \text{bal}(\mathbf{G}) = 1$$

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.

$$\mathbf{g}_1 := \texttt{see(me, opponent)}$$
$$\mathbf{g}_2 := \texttt{not(see(opponent, me))}$$
$$\kappa(\mathbf{G}) = 0 \quad \text{comp}(\mathbf{G}) = 1 \quad \text{bal}(\mathbf{G}) = \frac{1}{3}$$

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

$$\mathbf{g}_1 := \quad \texttt{on(black cube, blue floor)}\wedge$$
$$\texttt{not(on(black cube, red floor))}$$
$$\mathbf{g}_2 := \quad \texttt{on(black cube, red floor)}\wedge$$
$$\texttt{not(on(black cube, blue floor))}$$
$$\kappa(\mathbf{G}) = \quad \tfrac{1}{4} \quad \text{comp}(\mathbf{G}) = 1 \quad \text{bal}(\mathbf{G}) = 1$$

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.

$$\widehat{\mathbf{g}}_{\text{rock}} := \quad \texttt{hold(me,yellow sphere)}\wedge$$
$$\texttt{not(hold(opponent,yellow sphere))}\wedge$$
$$\texttt{not(hold(opponent,purple sphere))}$$
$$\widehat{\mathbf{g}}_{\text{paper}} := \quad \texttt{hold(me,purple sphere)}\wedge$$
$$\texttt{not(hold(opponent,purple sphere))}\wedge$$
$$\texttt{not(hold(opponent,black sphere))}$$
$$\widehat{\mathbf{g}}_{\text{scissors}} := \quad \texttt{hold(me,black sphere)}\wedge$$
$$\texttt{not(hold(opponent,black sphere))}\wedge$$
$$\texttt{not(hold(opponent,yellow sphere))}$$
$$\mathbf{g}_1 := \quad \widehat{\mathbf{g}}_{\text{rock}} \vee \widehat{\mathbf{g}}_{\text{paper}} \vee \widehat{\mathbf{g}}_{\text{scissors}}$$
$$\mathbf{g}_2 := \quad \widehat{\mathbf{g}}_{\text{rock}} \vee \widehat{\mathbf{g}}_{\text{paper}} \vee \widehat{\mathbf{g}}_{\text{scissors}}$$
$$\kappa(\mathbf{G}) = \quad \tfrac{1}{4} \quad \text{comp}(\mathbf{G}) = 1 \quad \text{bal}(\mathbf{G}) = 1$$

4| Goal and Metric

In Section 2 we introduced the XLand environment and explored some of the properties of this space in Section 3 such as the vastness, diversity, and smoothness across tasks. We now turn our attention to training an agent on XLand.

To train an agent π in an episodic environment such as XLand, one generally aims to maximise the expected return of the agent

$$\mathbf{V}_\pi(\mathrm{P}_\pi) := \mathbb{E}_{\mathrm{P}_\pi(\aleph)} \left[R_\pi(\mathbf{x}) \right].$$

where P_π is an agent-specific distribution over tasks.

A challenge in evaluating the performance of an agent in this massively multitask environment comes from the fact that each task can be of completely different complexity. The optimal value

$$\mathbf{V}^*(\mathbf{x}) := \max_\pi \mathbf{V}_\pi(\mathbf{x})$$

of one task can be of a different order of magnitude than the optimal value of another task $\mathbf{V}^*(\mathbf{x}')$, *i.e.* $\mathbf{V}^*(\mathbf{x}) \gg \mathbf{V}^*(\mathbf{x}')$. Consequently, simply averaging the agent's value across all tasks to form a single score will overemphasise tasks with large optimal values. Even if one was able to sensibly normalise value per-task, with a big enough set of tasks, averaging will remove relevant information regarding agent failures. For example, averaging will not surface an agent's failure modes on some tasks if these tasks do not occupy a big part of the task space (Balduzzi et al., 2018). This becomes an even bigger issue if there is no particular ground truth test distribution of interest, but rather our goal is to find a policy that is generally capable.

A Game Theoretic solution would be to focus on the infimum performance (the worst-case scenario (Nash et al., 1950)), since performance on this task will always lower bound any expectation over a distribution defined over the same set. Unfortunately, the infimum suffers from not providing any notion of progress or learning signal if there are any tasks that are simply impossible or extremely hard.

4.1| Normalised Percentiles

In this work we seek to create *generally capable* agents in the whole XLand task space. General capability is not strictly defined but has some desiderata:

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

These desiderata cannot be encapsulated by a single number describing an agent's performance, as they do not define a total order (Balduzzi et al., 2019). We move away from characterising agents purely by expected return, and instead consider the distribution of returns over a countable task space. However, for large task spaces this is a very high-dimensional object. In addition, due to the drastically different return scales of tasks, returns cannot be compared and one needs knowledge of each individual task to interpret the significance of reward. Naturally, one could normalise the return per task by the return of the optimal policy on each specific task. However, in practice:

$\mathbf{V}_\pi(\mathbf{x})$ is interpreted as an expectation over the Dirac delta distribution around $\mathbf{x}$.
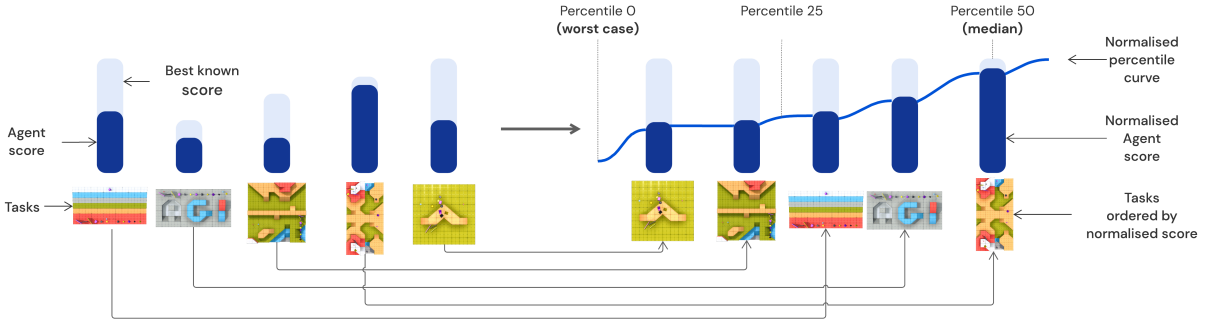
Figure 10 | The process of computing normalised percentiles. Tasks vary significantly in terms of their complexity, some have much higher values of optimal policies than others. We normalise the performance of the agent by an estimate of an optimal policy score – using the Nash equilibrium of trained agents – providing a normalised score, which after ordering creates a normalised percentile curve. This can be iteratively updated as new trained agents are created.

- an optimal policy is not known a priori,
- we want to use these normalisers over the entire environment space, which means that we need to know a single optimal policy for the entire space, and then normalise by its score on each task.

In order to address these issues we follow two practical simplifications.

First, to address the need of having one optimal policy for the entire space, we compute the normalisation factor for each game independently, and then combine them into a global normaliser.

Second, even with the above simplification we do not have access to an optimal policy per game. However, we can take ideas from multi-agent training algorithms that eventually converge to a Nash equilibrium (Heinrich and Silver, 2016; Marris et al., 2021; McMahan et al., 2003). We iteratively build a set of agents that are capable of solving a specific goal, and use the best mixture (Nash equilibrium) of them as a normalising constant. As training progresses and our agents become better at satisfying the goal, they will beat the existing Nash equilibrium and improve the normalising constant. This dynamic provides us with an iterative notion of improvement for a multi-task environment, rather than a fixed numerical quantity to describe progress. It is akin to theoretical results showing that in multi-agent problems it is impossible to have a fixed objective, because finding better agents and improving the quality of evaluation are the same problem (Garnelo et al., 2021). These normalisers give us a normalised score per task.

Finally, to mitigate the problem of having a high-dimensional normalised score distribution, we characterise the distribution in terms of the percentiles of normalised score, up to a maximum of the 50th percentile (median normalised score):

$$\text{perf}(\pi|\mathbf{g}, \mathbf{\Pi}_t) := \min_{(\pi_j, \mathbf{g}_j)} \mathbb{E}_{\mathbf{w}}\left[R_\pi(\mathbf{w}, (\mathbf{g}, \mathbf{g}_2, \dots \mathbf{g}_n), (\pi_2, \dots \pi_n))\right]$$

$$\text{norm}(\mathbf{g}|\mathbf{\Pi}_t) := \max_\pi \text{perf}(\pi|\mathbf{g}, \mathbf{\Pi}_t) = \text{NashValue}(\mathbf{g}|\mathbf{\Pi}_t)$$

$$\widehat{\text{perf}}(\pi|\mathbf{g}, \mathbf{\Pi}_t) := \frac{\text{perf}(\pi|\mathbf{g}, \mathbf{\Pi}_t)}{\text{norm}(\mathbf{g}|\mathbf{\Pi}_t)} \in [0, 1]$$

$$\text{perc}(\pi|\mathbf{\Pi}_t)[k] := \mathcal{P}_k(\widehat{\text{perf}}(\pi|\mathbf{g}, \mathbf{\Pi}_t)), \quad \text{for } k \in \{0, \dots 50\}$$

where $\mathcal{P}_k$ is the kth percentile and both min and max operations over policies operate over convex combinations of policies from a corresponding population $\mathbf{\Pi}_t$. Figure 10 illustrates this process. Each agent's performance is described as 51 numbers between 0 and 1, with each number being the normalised score at each integer percentile in the range of 0 to 50 (inclusive), which forms a non-decreasing sequence

$$\text{perc}(\pi)[k+1] \geq \text{perc}(\pi)[k].$$

One can read out various human interpretable quantities from this representation, e.g. $\text{perf}(\pi)[0]$ is the infimum – the normalised score an agent obtains on the hardest game; $\text{perf}(\pi)[50]$ is the median normalised performance; the smallest k such that $\text{perf}(\pi)[k] > 0$ informs us that an agent scores any reward in at least $(100 - k)\%$ of games (and thus provides a notion of coverage/participation).

We say an agent π is better than agent π' if and only if it achieves at least as good a score for every percentile, and on at least one percentile it achieves a strictly better score, formally:

$$\pi \succeq_{\mathbf{\Pi}_t} \pi' \iff \forall_k \text{perc}(\pi|\mathbf{\Pi}_t)[k] \geq \text{perc}(\pi'|\mathbf{\Pi}_t)[k]$$

$$\pi \succ_{\mathbf{\Pi}_t} \pi' \iff \exists_k \text{perc}(\pi|\mathbf{\Pi}_t)[k] > \text{perc}(\pi'|\mathbf{\Pi}_t)[k]$$

$$\wedge \pi \succeq_{\mathbf{\Pi}_t} \pi'.$$

Let us refer to our desiderata – if agent π fails catastrophically (never achieves any reward) on fewer tasks than π' then it will have non-zero values on a larger number of percentiles, and thus captured in our notion of being better. Conversely, if catastrophic failures are more common, then π will not be considered better (it can be non-comparable or worse). The notion of competency refers to the fraction of the score obtained by the Nash equilibrium over known policies, and thus similarly by being competent on more tasks, π will increase its values on smaller percentiles. Finally, a narrow competency will be visible in low scores over low percentiles, and despite high scores being obtained on high percentiles – such an agent will not be considered better. In addition, cutting our percentiles at 50 means that an agent that is an expert on less than half of the games, but does

---

$R_{\alpha\pi+(1-\alpha)\pi'}(\mathbf{x}) := \alpha R_\pi(\mathbf{x}) + (1-\alpha) R_{\pi'}(\mathbf{x})$

not score any points on remaining ones, will be considered worse than an agent of broader ability.

To summarise, we propose to use the following tools to measure and drive progress of general capabilities of agents:

- to normalise performance by the estimated highest obtainable score,
- to iteratively improve the estimate of the highest obtainable score,
- to evaluate agents across normalised score percentiles, creating a multi-dimensional performance descriptor,
- to require Pareto dominance over said descriptor to guarantee improvements with respect to our desiderata.

4.2 | Evaluation Task Set

The normalised percentile metric described in the previous section provides a way to compare agents and drive learning with a lens towards general capability. However, this metric is still evaluated with respect to a distribution of tasks $P_\aleph$. The XLand task space as defined in Section 2 is prohibitively large, and as such we need to create a manageable evaluation task set against which to assess agents' general capability.

Given a budget number of evaluation tasks (*e.g.* on the order of thousands), arbitrarily sampling tasks from $\aleph$ could risk critically underrepresenting the vastness and diversity of the underlying task space, with aliasing also hiding the smoothness property. As such, we define an evaluation task space that samples tasks spanning a smaller but representative subspace of XLand tasks, and skew sampling to ensure uniform coverage of interesting world and game features. Finally, we combine these evaluation worlds and games with pretrained evaluation policies to give us an evaluation task set.

Evaluation worlds   For evaluation, we want a set of worlds that expose agents to a range of topological challenges. We use a world-agent co-evolution process (Section A.1.1, Figure 33), saving the training distribution of worlds created at each point in training of this process. This gives a collection of worlds where the earlier-created worlds are generally topologically simpler than those created later in training. Uniformly sampling this collection of worlds with respect to the creation time gives a set of worlds spanning the range of topological complexity (as defined by an agent learning to find an object). We also randomly apply reflections and resampling of object positions to this set of worlds. Finally, we add additional Wave Function Collapse (Gumin, 2016) generated worlds, biased towards specific topological elements that we observe rarely: ones containing single central islands and door-like bottlenecks separating play areas. The gadget of each player is uniformly sampled and the colour ordering of each player randomly permuted. Exactly 12 objects are placed into each evaluation world, one of each colour-shape combination.

Evaluation games   In the game space, we look to create a set of evaluation games that span a large range of complexity and expressivity, but are still logically simple enough for quick human understanding. Therefore, representing the goals of the game in their *disjunctive normal form*, we restrict the evaluation games to have at most three options per goal, with each option composed of at most three predicates, and a maximum of six unique predicates used across all goals. Only two- and three-player games are considered in the evaluation set. Additionally, we ensure the evaluation set of games spans the range of competitiveness and balance (defined in Section 3) – we create discrete buckets in competitiveness-balance space, with some buckets corresponding to the extreme values of these measures. Evaluation games are sampled such that there is an equal number of games per competitiveness-balance bucket, and per competitiveness-balance bucket an equal number of games across the different number of options and predicates in the game. We also remove trivial games (*i.e.* where $\kappa(\mathbf{g}_1) \in \{0, 1\}$). The result is an evaluation set of games which is uniform across balance buckets, competitiveness buckets, number of options, and number of predicates.

Evaluation co-players   Each evaluation task must include policies to act as the co-players of the task, leaving a single player slot available for evaluation of an agent in the task. For the purposes of this work, we use a collection of pretrained agents. These include a `noop` agent that always emits the noop action (corresponding to not moving) and a `random` agent that emits an action uniformly sampled from the whole action space. In addition, we use agents trained on simpler incarnations of the evaluation space, as well as sub-spaces of evaluation space (*e.g.* an agent trained only on single predicate games). These agents were generated during earlier phases of the research project.

We combine the evaluation worlds, games, and co-players to get test and validation sets. We first generate the test set of evaluation tasks. Next the validation set of evaluation tasks is generated in an identical manner, however explicitly holding out all games and worlds within a certain distance from the test set (Section A.3) and likewise holding out all test set co-players except for the trivially generated `noop` and `random` policies. In addition, all hand-authored tasks (Section 4.3) are held out from all evaluation task sets. The test task set consists 1678 world-game pairs played with all 7 co-players for a total of 11746 tasks. The validation task set consists of 2900 world-game pairs played with a growing number of co-players: `noop`, `random` and an extra player per previous generation of training.

4.3 | Hand-authored Task Set

The evaluation set of tasks described previously covers a diverse subspace of XLand, however the automatic generation of these tasks can make interpretation of successful policies difficult – it can be hard to know what challenges an individual task poses. We created a hand-authored set of tasks which act as interpretable evaluation tasks. In addition, many of these hand-authored evaluation tasks are out-of-distribution or represent challenges that are extremely rare

Capture the Cube	Make Follow	Stop Rolling	Tag Fiesta 3p	Solve AGI

The agent (blue player) is rewarded when the purple cube and the black cube are both touching the blue floor. The co-player is rewarded when the purple cube and black cube are both touching the grey floor.

The agent (blue player) is rewarded when the red player touches the red floor. The red player is controlled by a policy designed to follow the blue player.

The agent (blue player) is rewarded when the purple sphere does not touch the red floor and the purple sphere is not held by the agent. The purple sphere starts at the top of the slope.

The agent (green player) is rewarded when any one of the co-players is not touching any of the floors. All players are equipped with the tagging gadget. The co-players have an equivalent goal.

The agent (blue player) is rewarded when the black sphere is placed on the red floor. The co-player (red player) is a policy that attempts to take any object the blue player holds.
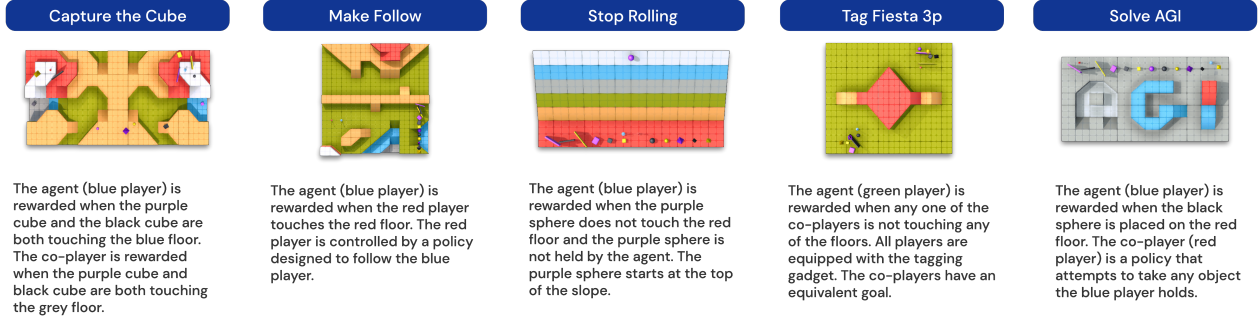
Figure 11 | Five examples of some of the 42 tasks in the hand-authored evaluation task set. A full list of hand-authored evaluation tasks is given in Table 8.

to be seen in a sample from the evaluation set, and thus further test ability of an agent to generalise. Examples of the 42 tasks in the hand-authored task set can be found in Figure 11 (full list is provided in Table 7), and include well known tasks such as *Capture the Flag*, *Hide and Seek*, and *King of the Hill* which have been projected into XLand space. Other examples include physical challenges such as *Stop Rolling* and *Tool Use*. The hand-authored task set is also held out from all training.

5 | Learning Process

We now turn our attention to the learning process. We seek agents that are generally capable in the XLand space. As one of the proxies to this goal, we want agents that can zero-shot generalise to tasks from the test set, and use normalised percentiles computed on the test set as the performance metric to encapsulate this.

Our training process consists of three main components:

1. Deep RL to update the neural network of a single agent. Deep RL optimises an agent to maximise expected return across a distribution of training tasks given.
2. Dynamic task generation with population based training to provide the distribution of training tasks for a population of agents. The task distributions are changed throughout training and are themselves optimised to improve the population's normalised percentiles on the validation set.
3. Generational training of populations of agents to chain together multiple learning processes with different objectives. Agents are trained with different learning objectives per generation, with each subsequent population bootstrapping behaviour off the previous generation of agents, to improve validation normalised percentiles with each subsequent generation.

We will now describe these three components in more detail.

5.1 | Deep Reinforcement Learning

An agent playing on an XLand task $\mathbf{x}$ takes in high-dimensional observations $\mathbf{o}_t$ at each timestep, and produce a policy from which actions are sampled $\mathbf{a}_t \sim \pi_t$, allowing the agent to maximise the collected reward on the task. We

use a neural network to parameterise the policy, and train this network using the V-MPO RL algorithm (Song et al., 2020). Similarly to the the original V-MPO implementation, we use single-task PopArt normalisation of the value function (Hessel et al., 2019; van Hasselt et al., 2016). At each weight update, the network parameterising π is updated in the direction to maximise the expected discounted return on the instantaneous task distribution $\mathbf{V}_\pi(P_\pi)$.

The per-timestep observation $\mathbf{o}_t$ the neural network takes as input consists of an RGB image from the agent player's point-of-view $\mathbf{o}_t^{\text{RGB}}$, proprioception values corresponding to the forces relating to the agent's player holding an object $\mathbf{o}_t^{\text{prio}}$, as well as the goal $\mathbf{g}$ of the agent's player in the task $\mathbf{x}$. A recurrent neural network processes this information to produce a value prediction $\mathbf{v}_t$ and policy π_t, from which a single action $\mathbf{a}_t$ is sampled.

Goal attention network The recurrent neural network incorporates an architecture that is tailored towards the structure of $\mathbf{V}^*$ the value of an optimal policy for a given task. For simplicity let us write

$$\mathbf{V}^*(\mathbf{g}) := \max_\pi \mathbf{V}_\pi(\mathbf{g})$$

to denote the value of the optimal policy when we hold the world, other goals, and co-players fixed.

Theorem 5.1 (Value Consistency). *For a goal* $\mathbf{g} := \bigvee_{o=1}^k \left[\bigwedge_{c=1}^{n_o} \phi_{oc}\right]$ *we have*

$$\mathbf{V}^*(\mathbf{g}_l) \leq \mathbf{V}^*(\mathbf{g}) \leq \mathbf{V}^*(\mathbf{g}_u)$$

for $\mathbf{g}_l := \bigvee_{o=1}^{k-1} \left[\bigwedge_{c=1}^{n_o} \phi_{oc}\right], \mathbf{g}_u := \bigvee_{o=1}^k \left[\bigwedge_{c=1}^{n_o'} \phi_{oc}\right]$ *where* $n_o' \geq n_o$.

This property says that for each game we can easily construct another game providing an upper or lower bound of the optimal value, by either selecting a subset of options or a superset of conjunctions. Therefore, with $\mathbf{g} := \widehat{\mathbf{g}}_1 \vee \cdots \vee \widehat{\mathbf{g}}_{n_o}$ we have

$$\mathbf{V}^*(\mathbf{g}) \geq \max_{i=1}^{n_o} \mathbf{V}^*(\widehat{\mathbf{g}}_i).$$

By putting $\widehat{\mathbf{g}}_0 := \mathbf{g}$ we can consequently say that

$$\mathbf{V}^*(\mathbf{g}) = \max_{i=0}^{n_o} \mathbf{V}^*(\widehat{\mathbf{g}}_i),$$
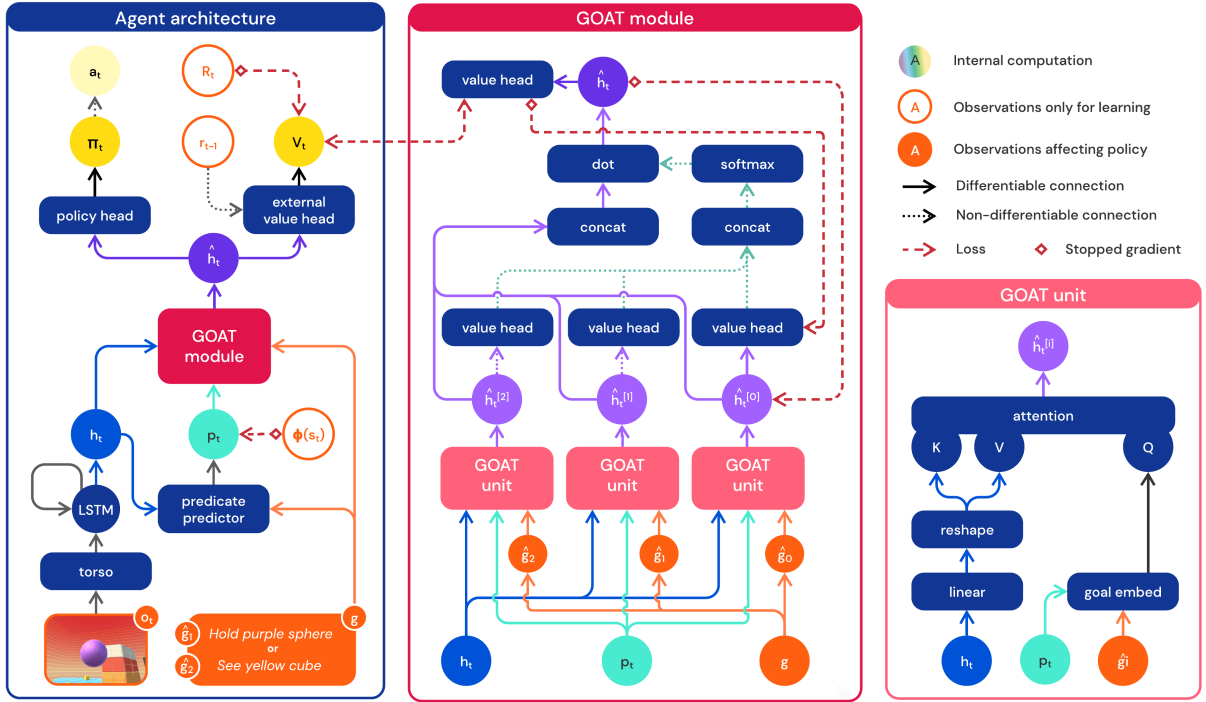
Figure 12 | A schematic of the neural network used to parameterise the agent's policy. The input observations $\mathbf{o}_t$ consist of RGB images and the proprioception, and the agent also receives its goal $\mathbf{g}$. The agent processes the observations through the torso and a recurrent core to 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).

the optimal value for $\mathbf{g}$ is the maximum of values of the subgames consisting of each of the options $\widehat{\mathbf{g}}_i$ and the full goal $\mathbf{g}$ itself.

We encode this property explicitly in the neural network architecture. At each timestamp, the agent produces an internal hidden state embedding using the history of its observations but no knowledge of the goal. Separately, the goal is embedded and used as a query for an attention mechanism that produces a goal-attention hidden state $\widehat{\mathbf{h}}_t^{[0]}$. In parallel, the agent produces analogous embeddings for each option $\widehat{\mathbf{h}}_t^{[i]}$, and estimates the current value of each $\widehat{\mathbf{v}}_t^{[i]}$. This asks the agent to predict *what would be its expected return if it was to focus on option i until the end of the episode*, $\widehat{\mathbf{v}}_t^{[i]}$. With the attention mechanism, the agent switches its hidden state to another option's hidden state if and only if the value of said option was higher than its current estimate of the value of the whole goal. This way the agent is internally encouraged to be consistent with the *Value consistency* property of the game space.

More specifically, $\mathbf{h}_t$ is the hidden state of an LSTM (Hochreiter and Schmidhuber, 1997) that takes as input the processed pixel and proprioception observations. We attach an atomic predicate state predictor to the output of the LSTM: this predictor is a simple multi-task binary classifier, outputting $\mathbf{p}_t$ which predicts the dimensions of $\phi(\mathbf{s}_t)$ relevant for $\mathbf{g}$, and is trained as an auxiliary classification

loss (*i.e.* only shaping the internal representations, without explicitly affecting the policy (Jaderberg et al., 2017b)). The *goal attention (GOAT)* module then follows:

$$\mathbf{GOAT}(\mathbf{h}_t, \mathbf{p}_t, \mathbf{g}) := \sum_i \left[\alpha(\widehat{\mathbf{v}}^{[i]}, \{\widehat{\mathbf{v}}^{[j]}\}_{j=0}^o) \right] \widehat{\mathbf{h}}_t^{[i]}$$

$$\widehat{\mathbf{v}}_t^{[i]} := f_{\mathbf{V}}(\widehat{\mathbf{h}}_t^{[i]}) \qquad \forall_{i=0}^o$$

$$\widehat{\mathbf{h}}_t^{[i]} := \mathbf{GOAT}_{\text{unit}}(\mathbf{h}_t, \mathbf{p}_t, \widehat{\mathbf{g}}_i) \qquad \forall_{i=0}^o$$

$$\mathbf{GOAT}_{\text{unit}}(\mathbf{h}, \mathbf{p}, \mathbf{g}) := \text{att}(\mathbf{h}, [\mathbf{p}; f_{\mathcal{G}}(\mathbf{g})]),$$

where $\text{att}(\cdot, \cdot)$ is an attention module (Bahdanau et al., 2015), $[\![\cdot]\!]$ denotes the stop gradient operation, meaning $\nabla_x [\![x]\!] = 0$, and $\alpha(a, A)$ is a weighting scheme, *e.g.*: $\alpha_{\text{argmax}}(a, A) = 1 \iff a = \max\{A\}$, or $\alpha_{\text{softmax,Z}}(a, A) := \frac{\exp(a/Z)}{\sum_{b \in A} \exp(b/Z)}$.

Given this parameterisation, we add corresponding *consistency losses*:

$$\ell_t^{\mathbf{V}} := \left([\![\widehat{\mathbf{v}}_t]\!] - \widehat{\mathbf{v}}_t^{[0]} \right)^2 \qquad \ell_t^{\mathbf{h}} := \left([\![\widehat{\mathbf{h}}_t]\!] - \widehat{\mathbf{h}}_t^{[0]} \right)^2,$$

where $\widehat{\mathbf{h}}_t := \mathbf{GOAT}(\mathbf{h}_t, \mathbf{p}_t, \mathbf{g})$, $\widehat{\mathbf{v}}_t := f_{\mathbf{V}}(\widehat{\mathbf{h}}_t)$, and $f_{\mathfrak{G}}$ is the goal embedding function (see Figure 38). These losses encourage the value predicted for the full goal $\mathbf{g}$ to not be smaller than the value of any of the options $\widehat{\mathbf{g}}_i$. A schematic view of this process is provided in Figure 12 with the details of

the goal embedding $f_{\mathcal{G}}$ and atomic predicate predictions provided in Figure 38. Note, that these are all internal value functions predictions that do not use any privileged information. We observed that faster learning can be achieved if the value function $\mathbf{v}$ used for RL itself does get access to extra information (Vinyals et al., 2019) in the form of the reward from the last step r_{t-1}. We add a simple L2 loss to align these two value heads in a co-distillation manner (Zhang et al., 2018):

$$\ell^{\text{align}} := \|\mathbf{v}_t - \widehat{\mathbf{v}}_t\|^2.$$

We do not stop gradients through $\mathbf{v}$ meaning that the privileged information value head $\mathbf{v}$ is penalised for expressing quantities that the internal $\widehat{\mathbf{v}}$ cannot model as well.

5.2| Dynamic Task Generation

Due to the vastness of task space, for any given agent, many tasks will either be too easy or too hard to generate good training signal. To tackle this, we allow the train task distribution to change throughout the learning process in response to the policy of the agent itself. The agent's neural network is trained with RL on the instantaneous training task distribution $P_\pi(\aleph)$.

We operationalise $P_\pi(\aleph)$ by using a filtering of a proposal distribution using a simple set of tests evaluating tasks usefulness for the current stage of learning.

Proposal train tasks are generated in a similar manner to the evaluation validation task set: worlds, games, and co-players are generated as described in Section 4.2 ensuring no collisions with the validation and test sets (Section A.3). We establish a task's usefulness by comparing the performance of the agent to the performance of a control policy π_{cont}.

The intuition of using a control policy is that the agent will only train on a task if the agent's returns are significantly better than those of the control policy. This guarantees that the agent is performing meaningful actions in the task that affect the return. In practice, we set the control policy to be a uniform random action policy. However, an interesting alternative would be to set the policy to be the agent's past policy – this would let us determine whether the agent's policy has recently improved or worsened on this task.

A proposal task is accepted (used for training) if and only if the following three criteria are met:

1. The agent has a low probability of scoring high on a given task

$$\Pr[R_\pi(\mathbf{x}) > m_s] < m_{\text{solved}}.$$

2. The agent has a high probability of performing better than the control policy

$$\Pr[R_\pi(\mathbf{x}) > R_{\pi_{\text{cont}}}(\mathbf{x}) + m_>] > m_{>\text{cont}}.$$

3. The control policy is not performing well

$$\mathbf{V}_{\pi_{\text{cont}}}(\mathbf{x}) < m_{\text{cont}}.$$

At a high level, the filtering of proposal tasks gives a mechanism for removing tasks that are too-easy (criterion 1), tasks that are too-hard (criterion 2), and tasks in

which the control policy is sufficient to achieve a satisfactory score (criterion 3), based on the agent's current behaviour at each point in training. All the above parameters $\mathbf{m} = \{m_>, m_s, m_{\text{cont}}, m_{>\text{cont}}, m_{\text{solved}}\}$ form agent-specific hyperparameters that define $P_\pi(\aleph)$. We estimate the criteria using Monte Carlo with 10 episode samples for each policy involved.

For example, a control policy return threshold $m_{\text{cont}} = 5$ would disallow any training tasks where a control policy is able to get a return of at least 5 (the reward is on a scale of 0 to 900). When using a uniform-random policy over actions as the control policy, this could be used to ensure the training task distribution doesn't contain tasks that are very easy to get reward. The combination, for example, of $m_> = 2$ and $m_{>\text{cont}} = 0.9$ would only allow training tasks where the agent achieves a return in all ten episode samples of at least 2 reward more than the return achieved by the control policy – this could ensure that the agent only trains on tasks where its behaviour is already better than that of the control policy. As a final example, the combination of $m_s = 450$ and $m_{\text{solved}} = 0.1$ would disallow training on any task where the agent is able to achieve more than 450 reward on any of its episode samples – this could filter out tasks where the agent is already performing well.

Whilst this filtering mechanism provides a way to supply the agent with a dynamic training task distribution, the filtering criterion itself may benefit from being dynamic. What is considered too-hard or too-easy at the beginning of training may encourage early learning, but cause saturation or stalling of learning later in training. Due to the vastness of the XLand task space we seek learning processes that do not saturate, but rather dynamically shift to ensure the agent never stops learning.

To address this, we incorporate population based training (PBT) (Jaderberg et al., 2017a) which provides a mechanism to dynamically change hyperparameters of the learning process (Jaderberg et al., 2019). Rather than training a single agent, we train a population of agents, each agent training on its own task distribution $P_{\pi_k}(\aleph)$ that is controlled by its own hyperparameters $\mathbf{m}_k$. Additionally, the learning rate and V-MPO hyperparameter ϵ_α are added to the set of hyperparameters modified by PBT.

PBT requires a fitness function to compare two agents and propagate the preferred agent. We use the normalised percentiles on the validation set. Periodically during training, agents are compared, and only if an agent Pareto dominates another agent in normalised score across percentiles it undergoes evolution – the dominant agent's weights are copied, its instantaneous task distribution copied, and the hyperparameters copied and mutated, taking the place in training of the non-dominant agent. More details can be found in Section A.7.

This process constantly modifies the dynamic task generation process and agent population to drive iterative improvement in normalised percentiles.

5.3| Generational training

With this combination of deep RL and dynamic task distributions we hope to provide a training process to continually improve agents in terms of their normalised percentiles as measured on the validation task set. However, in practice, the limitations of RL and neural network training dynamics still pose a challenge in training agents on the XLand task space from scratch.

It has been observed that higher performance and faster training can be achieved in deep RL by first training an agent, then subsequently training a new agent on the identical task whilst performing policy distillation from the first agent (Czarnecki et al., 2019; Furlanello et al., 2018; Schmitt et al., 2018). We employ this technique multiple times on populations of agents: a population of agents is trained, then a new generation of agents is trained distilling from the best agent of the previous generation's population, with this process repeated multiple times. Each generation bootstraps its behaviour from the previous generation. Furthermore, these previous generations also give us an opportunity to increase our pool of co-player policies and increase the diversity of our training experience, similarly to the AlphaStar league (Vinyals et al., 2019). At each generation, our training procedure includes the best player from each previous generation in this pool of players.

A final advantage of generational training of populations is that the learning objectives and agent architecture can vary generation-to-generation. We take advantage of this by using self reward-play: an RL objective which encourages exploration. In our training methodology, self reward-play is utilised for the first few generations followed by the regular RL objective in the later generations.

Self reward-play   One of the central desires of a generally capable agent is that the agent should catastrophically fail on as few tasks as possible. To target this objective we seek agents that minimise the smallest non-zero normalised percentile – to obtain at least one timestep of reward in as many tasks as possible, the problem of exploration. We define *participation* as the percentage of tasks the agent obtains a non-zero reward in.

To aid learning participation, we present challenges to the agent that it is capable of satisfying by asking the agent to revert a changes in the environment that the agent itself previously created. Self reward-play rewards the agent for satisfying a goal $\mathbf{g}$, and after succeeding the agent is rewarded for fulfilling $\texttt{not}(\mathbf{g})$ without resetting the environment, with this flip in goal repeating after each satisfaction. This can be seen as an agent playing in a self-play competitive manner against itself, where one player must satisfy $\mathbf{g}$ and the other player must satisfy $\texttt{not}(\mathbf{g})$, however the players act sequentially, and are played by the same agent.

In practice, we implement this by using the reward $r_t^{\text{srp}} := |r_t - r_{t-1}|$ and setting the discount $\gamma_t = 0$ if $r_t^{\text{srp}} > 0$ (which rewards the agent for minimising the time until the next goal flip).

Empirically, we find that optimising for self reward-play

drastically improves exploration. The agent is encouraged to interact with the world and to change its reward state, after which it must change the state back again, and so on. In comparison, when optimising the discounted sum of environment reward, changing the environment yields the risk of changing the (unobserved) environment reward from 1 to 0 which discourages the agent from interacting with the environment. As a result, agents that optimise with self reward-play achieve significantly higher participation in the same amount of training time (see Section 6.2.2). However, by construction, self reward-play does not optimise agents to be competent (*i.e.* whilst the smallest non-zero normalised score percentile is minimised, the normalised percentiles remain low). We discuss in detail how self reward-play is leveraged in Section 6.2.

Iterative normalised percentiles   As discussed in Section 4.2, the test set contains a fixed set of co-player policies (used also to evaluate against). However, the validation set does not contain these, but only the trivially generated noop-action and random-action policies. For evaluation, co-player policies are required to play validation tasks with, and the normaliser score used by the normalised percentile metric also uses this fixed set of co-player policies. The generational training process allows us to start only with the trivially generated noop-action and random-action policies and to iteratively refine the validation normalised percentiles metric: each generation creates agents which are added to the validation set and used to update the normalised percentile metric, with the next generation incorporating the previous generation's policies in its training, with this process repeating, iteratively refining the normalised percentile metric and expanding the set of co-player policies. This means that the normalised percentiles metric on the validation set used to guide training changes each generation as more policies are added to the validation co-player set. Note that for all results reported in Section 6, we report the normalised percentiles on the test set which is fixed, with the same fixed set of co-player policies, for all generations.

5.4| Combined Learning Process

These three components of the training process – deep reinforcement learning, dynamic task generation, and generational training – are combined to create a single learning process. The three pieces are hierarchically related. On the smallest wall-clock timescale (seconds), deep RL provides weight updates for the agents' neural networks, iteratively improving their performance on their task distributions. On a larger timescale (hours), dynamic task generation and population based training modulate the agents' task distributions to iteratively improve the Pareto front of validation normalised percentile scores. Finally, on the largest timescale (days), generational training iteratively improves population performance by bootstrapping off previous generations, whilst also iteratively updating the validation normalised percentile metric itself.

From the opposite perspective, the overall system continuously creates generations of agents seeking to improve the validation normalised percentile metric – to gradually
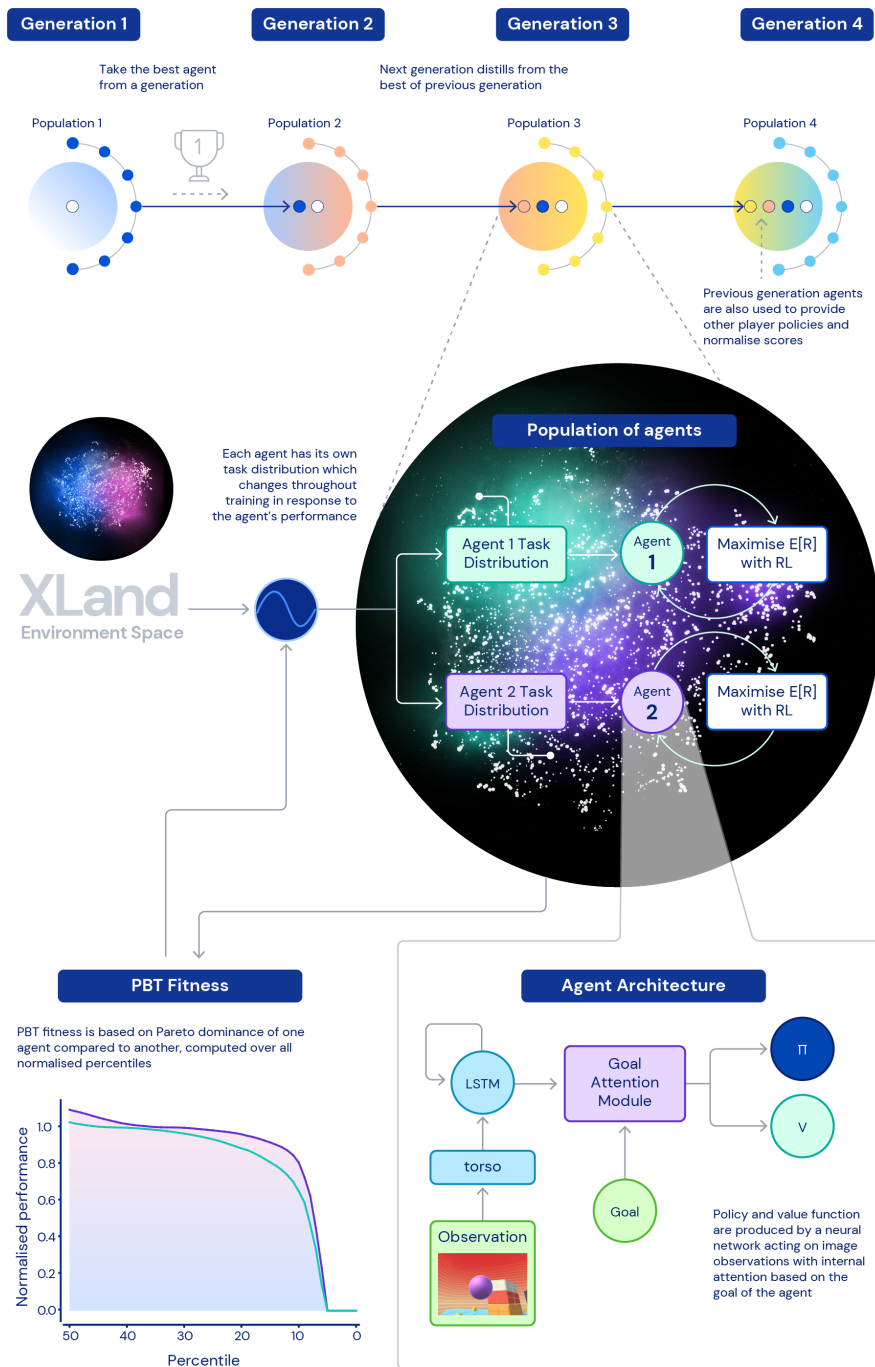
Figure 13 | The combined learning process. **(Top)** Generations of agents are trained, composed of populations of agents where the best performing agents become distillation teachers of the next generation as well as co-players to train against. **(Middle)** Inside each population, agents are trained with dynamic task generation that continuously adapts the distribution of training tasks $P_{\pi_k}(\aleph)$ for each agent π_k, 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**.
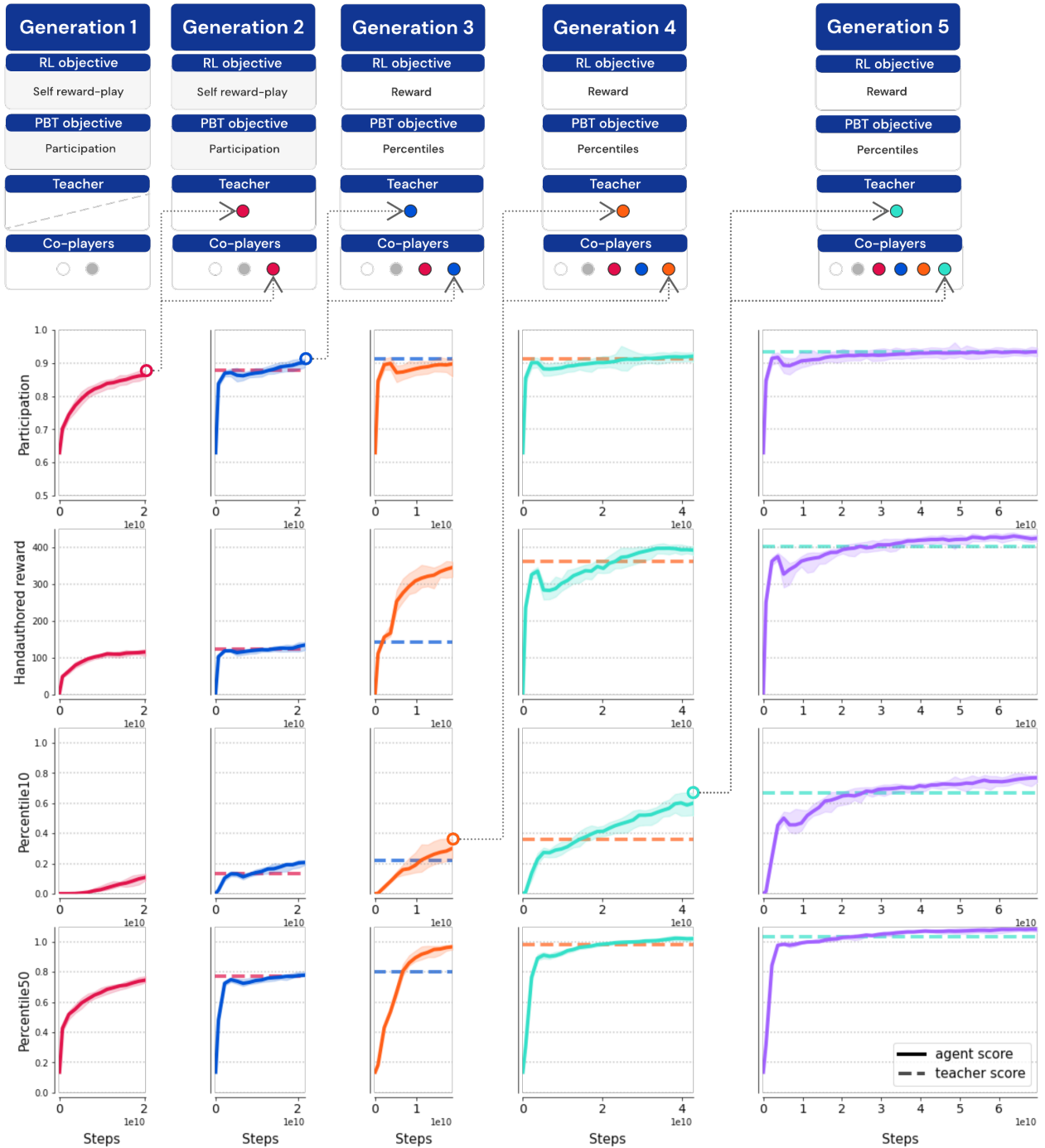
Figure 14 | Generations of performance as measured on the held out test task set. The first two generations focus on the maximisation of participation using the self reward-play RL objective (Section 5.3). In between generations, the best agent wrt. the objective is selected and used as a teacher and additional co-player to play against in further generations. Generations 3-5 focus on the improvement of normalised 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).
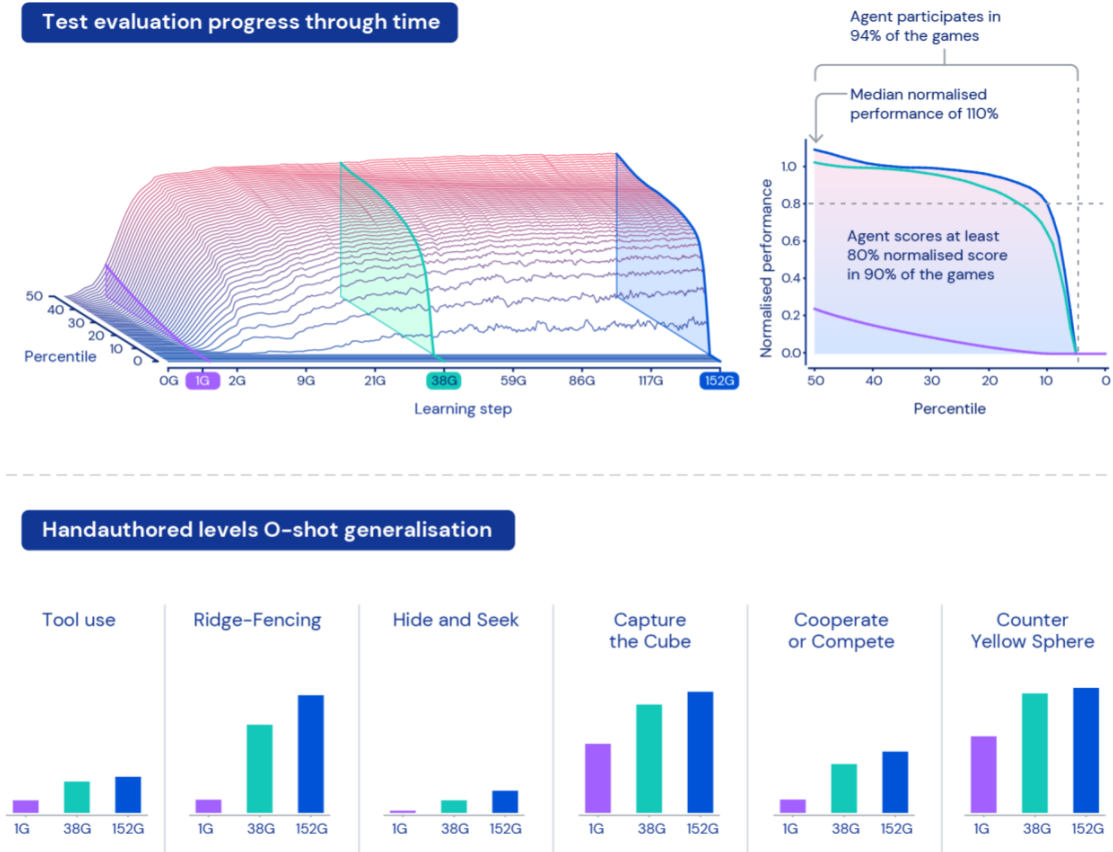
Figure 15 | **(Top)** On the left we see the *learning surface*, showing the progress of a generation 5 agent through time with respect to each of the normalised percentiles. The surface shows the normalised score (height) for each percentile (depth) through training (x-axis). Therefore, the flat bottom 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.

improve the coverage and competence on tasks. In order to do so, a generation's population is changing the distribution of training tasks for each agent such that the agents keep improving the Pareto front of validation normalised percentile scores. The agents themselves are updating their neural network weights with reinforcement learning based on the stream of experience they generate from their training task distributions, gradually improving their performance on this shifting distribution. The whole process is summarised in Figure 13.

The iterative nature of the combined learning system, with the absence of a bounded metric being optimised, leads to a potentially open-ended learning process for agents, limited only by the expressivity of the environment space and the agent's neural network.

6 | Results and Analysis

In this section, we show the results of training agents with the learning process introduced in Section 5, with the specific experimental setup described in Section 6.1. The learning dynamics are explored in Section 6.2 with respect to the evaluation metric defined in Section 4. In Section 6.3,

we analyse the zero-shot generalisation performance of the trained agent across the test set. Section 6.4 delves into some emergent agent behaviour that is observed on hand-authored probe tasks. Moving beyond zero-shot behaviour, in Section 6.5 we show the results of finetuning the trained agents for wider transfer. Finally, in Section 6.6 we analyse the representations formed by the agent's neural network.

All the results reported in this section are computed on tasks that were held-out of training.

6.1 | Experimental Setup

More details on the architecture, hyperparameters, other elements of the experimental setup are provided in Section A.4, A.5, A.6, and A.7. Each agent is trained using 8 TPUv3s and consumes approximately 50,000 agent steps (observations) per second.

6.2 | Agent Training

We trained five generations of agents, varying the learning setup with each generation. The results of this process is shown in Figure 14. The learning process per generation is described below.
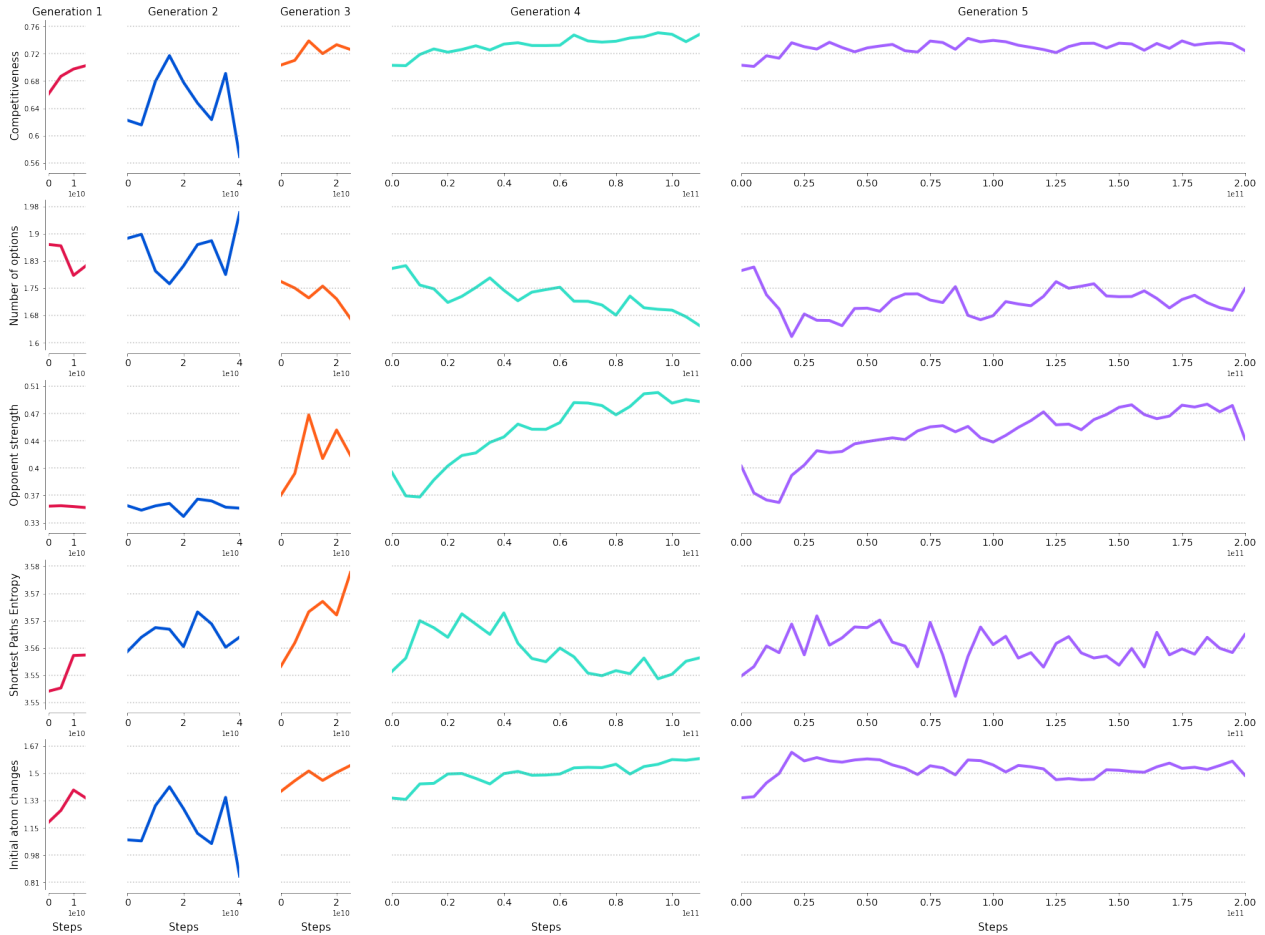
Figure 16 | Evolution of the training distribution of tasks due to dynamic task generation throughout 5 generations of agents (showing values from one agent per population only). We characterise the the training distribution by five measures (from the top): competitiveness (a property of games), number of options (a property of games), opponent strength (how performant the co-player in the task is), shortest paths entropy (a property of the worlds), initial atom changes (a property of the tasks, how many atomic predicates must be changed to satisfy an option). These change a lot throughout training, for example the strength of the opponents grows over time, generation 3 focuses more on worlds with larger shortest path entropy, and later generations focus on more competitive games.

The co-player set of policies were initialised with a noop and a random policy. We used the generational mechanisms described in Section 5.3. At the end of each generation, we selected the best agent that was produced throughout the generation. This agent was then used in three ways by subsequent generations: 1) as a policy to use for distillation in the next generation, 2) as an additional policy in the co-player set of policies, and 3) as an additional player as part of the computation of the validation normalised percentile metric.

We varied the learning setup in the following way across generations. In the first two generations, the agent was trained with self reward-play to encourage exploration. In these generations, the fitness used for PBT was the average participation as measured on the validation task set. Subsequent generations were trained without self reward-play and used Pareto dominance over 10th, 20th and 50th percentiles of normalised score on the validation task set as PBT fitness. When selecting the best agent for the next generation, the agent with the highest participation was chosen in the first two generations, and the agent with the highest

10th percentile normalised score in subsequent generations.

After two generations of training, we obtained an agent trained with self reward-play with a high test participation (91%) but low test 10th percentile and 50th percentile normalised scores – 23% and 79% respectively. The generation 3 agents quickly outperformed these scores as they did not use self reward-play and instead maximised true reward. Our final agent in generation 5 reached 95% participation (however it participates in 100% of tasks that humans can, see details in Section 6.3.1), 82% 10th percentile, 112% 50th percentile (median normalised score) on the test set, and 585 average return on the hand-authored task set (which is provably at least 65% of the optimal policy value), Figure 14 (right). The learning surface for the final 5th generation is shown in Figure 15.

6.2.1 | Dynamic Task Generation Evolution

Figure 16 shows how various properties of our tasks change throughout training as a result of the dynamic task generation (DTG).
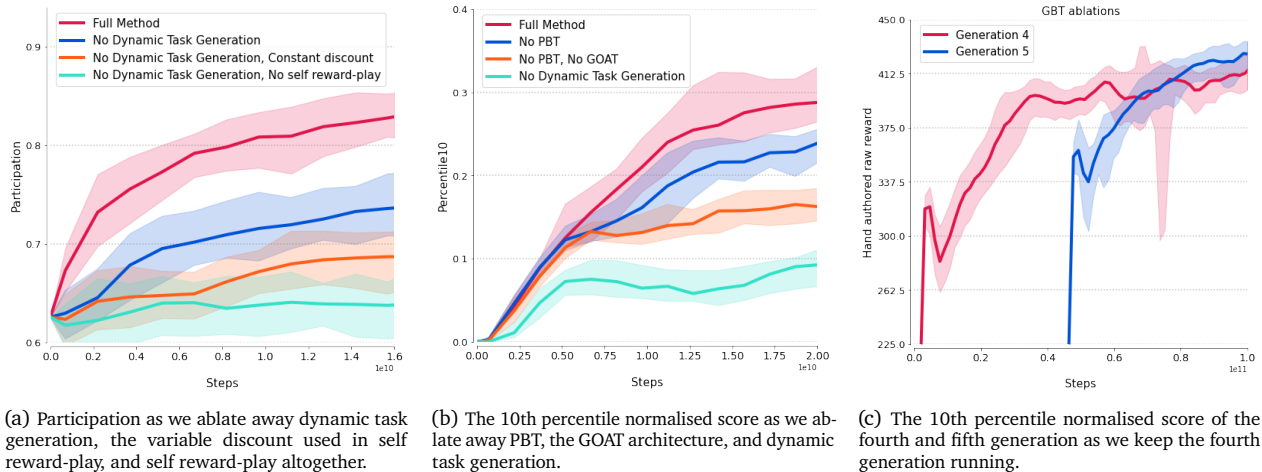
(a) Participation as we ablate away dynamic task generation, the variable discount used in self reward-play, and self reward-play altogether.

(b) The 10th percentile normalised score as we ablate away PBT, the GOAT architecture, and dynamic task generation.

(c) The 10th percentile normalised score of the fourth and fifth generation as we keep the fourth generation running.

Figure 17 | Ablations of the training methods. In all plots, the curve designates the median agent performance in the population while the shaded area shows the spread between the best and the worst agent.

We can see that for generation 3 and onward, DTG significantly increases the average strength of the co-players that the agent trains against. Similarly, there is an increase in the competitiveness as well as a decrease in the number of options of games presented to the agent for training. The composition of these 3 factors – stronger opponents, more competitive scenarios, and less options, creates a training distribution of hard problems (since agents are forced to compete with capable opponents). Similarly, the number of initial atomic predicate changes needed gradually increases, meaning that agents are increasingly placed in scenarios where multiple predicate states must be changed from their initial state to obtain reward.

All these changes are driven by the agent's performance; there is no direct control given to the agent to focus on any of the above properties, and thus these dynamics are purely emergent.

6.2.2 | Ablation Studies

Our ablation studies evaluate the impact of different aspects of our training methodology.

Early generations: self reward-play and dynamic task generation. As discussed in Section 5.3, early training in our environment is difficult. We use self reward-play to encourage the agent to explore changing the environment state, and dynamic task generation to avoid training on tasks that are initially too hard for the agent and would not provide any useful training signal. In this ablation, we trained multiple agents from scratch with a diverse pool of co-player policies. We show the participation of the different trained agents in Figure 17a. Our full method, which used both dynamic task generation and self reward-play, reached a participation of 84% after 16 billion steps. We see that removing in turn dynamic task generation, the use of zero discounts on step changes (part of our self reward-play procedure), and self reward-play resulted in significant reductions in performance. When none of these methods are used, the agent fails to learn any meaningful policy.

Later generations: Population based training, the GOAT architecture and dynamic task generation. In our next ablation, we consider a setup similar to the third generation in our main experiments. The agents were not trained with self reward-play, but during the first 4 billion steps have a distillation loss towards the teacher policy of an agent that was trained with self reward-play. The agents were trained with a diverse pool of co-player policies. The results are shown in Figure 17b. We trained each agent for 20 billion steps. Similarly to our main experiments, our full method uses PBT, the GOAT architecture and dynamic task generation. Our first ablation removes PBT from our method, replacing it by a simple sweep across 8 agents, which leads to a ~ 20% reduction in performance of the best agent. Additionally removing the GOAT architecture from our method and replacing it with a simpler architecture similar to the one used in Hessel et al. (2019) yields another ~ 30% reduction in performance. Finally, removing dynamic task generation from our method whilst keeping other aspects constant leads to a ~ 65% reduction in performance.

Generation based training. In our final ablation, we consider the benefits of generation based training. We kept the fourth generation of main experiments from Section 6.2 running in order to compare its performance to the fifth generation. The results are shown in Figure 17c. We offset the fifth generation's curve to the point the best agent from the fourth generation was selected. We can see that as training progresses the fifth generation outperforms the previous generation (both in terms of comparing best agents from corresponding populations, as well as comparing the averages), even when generation 4 was trained for the same amount of time.

6.3 | Performance Analysis

Due to the vastness of task space, with unknown maximum scores, there is no single notion of performance to report. Consequently, we rely on relative performance analysis and other qualitative notions of progress described below.
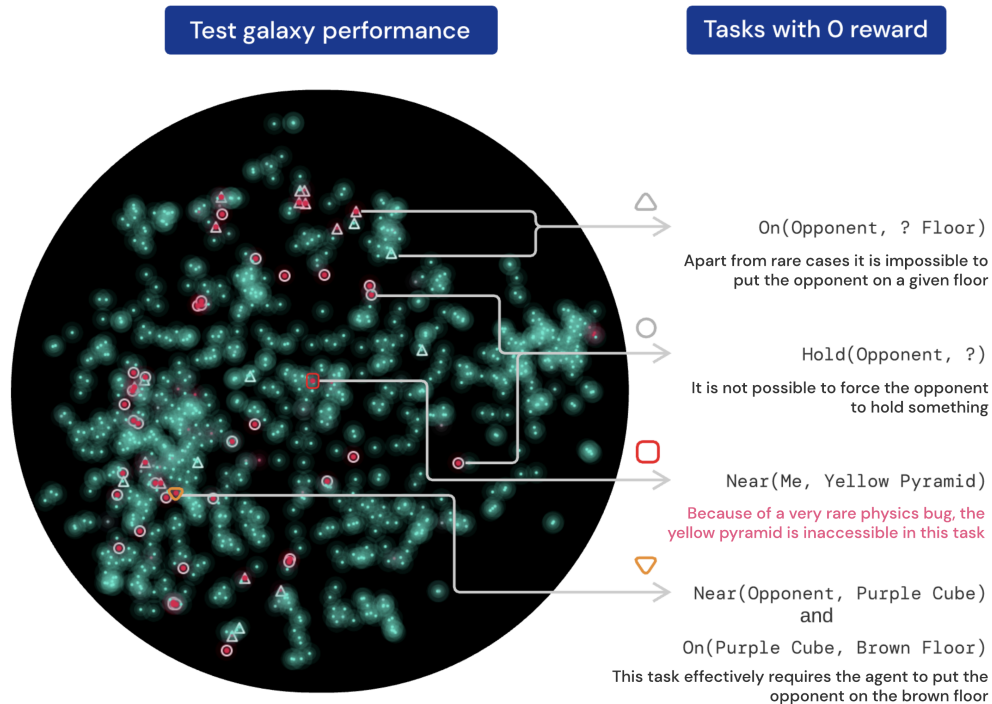
Figure 18 | A visualisation of the test set of tasks, with the corresponding agent performance. The red colour corresponds to a low normalised score and green to a high one. We identify four sources of games the agent scores 0 reward on (listed on the right): 1) tasks that require the agent to put the 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*.

6.3.1 | Coverage

First, we focus our attention on answering the question *are there any test tasks, where the agent never reaches a rewarding state?* We identify that there are indeed a few percent of this space where none of the agents ever score any points. Further investigation shows that all these failed tasks involve impossible challenges, requiring an agent to *make the co-player hold something* (which, without the cooperation of the opponent is impossible) or to *place the co-player on a specific floor* (which, can also be impossible to achieve given the physical simulation of the game). Furthermore, we identify a single task, which, due to a very rare physics bug is impossible to solve because the object of interest spawns outside the reachable region. Figure 18 shows these games in the XLand galaxy. After removing the impossible tasks listed above our agent participates in every single test task, thus suggesting they are indeed widely capable.

6.3.2 | Relative Performance

We investigate various properties of the games, and how they translate to the relative improvement of our agents (using test normalised scores to measure this quantity). In Figure 19 we can see that the overall normalised score of our agent is higher on games which are more competitive, suggesting that it is in these challenging competitive scenarios our proposed learning process brings the biggest improvement relative to the pretrained evaluation policies in the

test set. Similarly, high normalised score is correlated with a large number of goal predicates (and thus a need to reason about many relations at the same time) as well as high initial atom changes (the number of relations that need to be changed, before an agent can get to a rewarding state). We also observe the biggest improvements with fewer options – games where there is just one option are much harder on a purely navigational level, as an agent cannot *choose* what to do, but rather is forced to satisfy a single option. Finally, we also see a big improvement relative to the evaluation policies when the agent is tasked with goals involving object-object interactions, such as *make the yellow sphere be near the purple pyramid*, as opposed to tasks related to the players themselves, e.g. *hold a purple sphere*. Overall, we see a general trend of agents showing the greatest improvements in the most challenging parts of our game space.

6.4 | General Capabilities

We now provide an overview of some of the general capabilities of the agent observed, allowing them to participate in a variety of tasks, execute various behaviours, and show satisfactory handling of new, unexpected situations.

Whilst the current instantiation of XLand is extremely vast, one can easily hand-author tasks that could only extremely rarely, or cannot at all, be generated during training due to the constraints of our training task generation process. For example we can place agents in worlds that lack ramps to
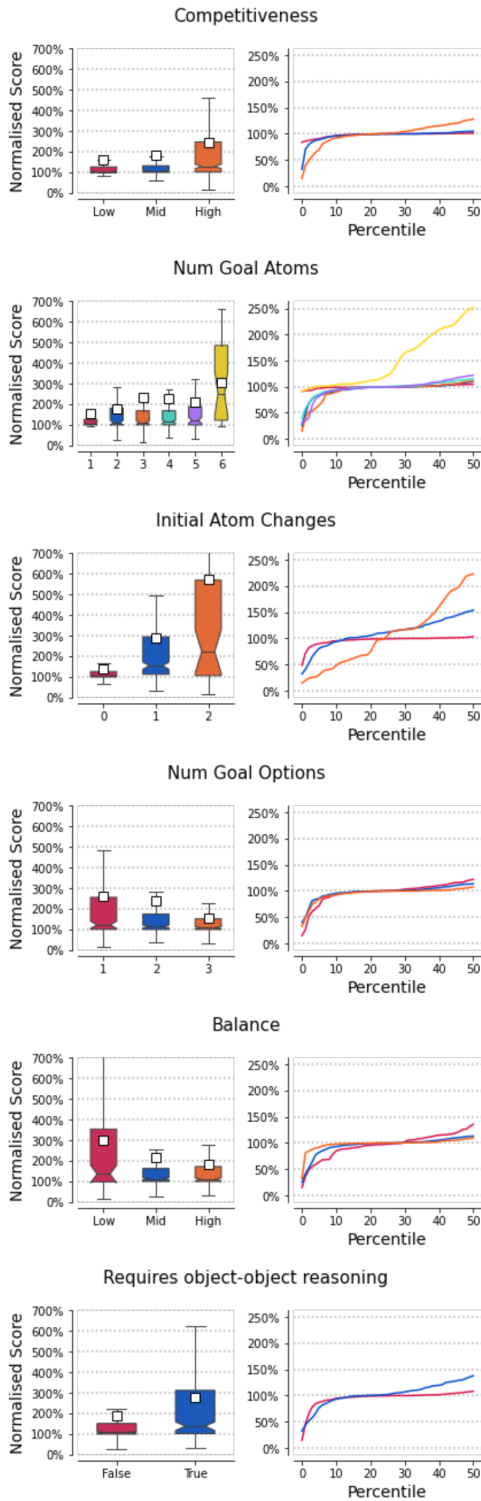
Figure 19 | **(Left)** Box plots showing the distribution of normalised scores for the final agent across various types of validation tasks. Whiskers denote the minimum and maximum value, the notch denotes the median, and the box area is between the 25th and 75th percentiles. The white square denotes average performance. **(Right)** Normalised test percentiles for the corresponding subsets of tasks.

challenge their ability to navigate, we can make them face unseen co-players, and we can execute interventions mid-episode. These probe tasks allow us to better understand and clarify the limits of generality of our agents.

6.4.1 | Hand-authored tasks

We now consider the qualitative behavioural properties of our agents at different points throughout training on hand-authored tasks (see Figure 11 for some some examples and Table 7 & 8 for a list of all). We compare two agents on a selection of the hand-authored task set: the final generation 4 (π_{G_4}) agent and the final generation 5 (π_{G_5}) agent. A selection of videos of the generation 5 (π_{G_5}) agent can be found in the supplementary results video here.

Capture the flag. In this two-player task, the agents' goal is to capture the cube in the opponent's base and bring it back to their own base. An agent gets a reward if the opponent's cube touches the floor of their own base while their own cube also touches the floor of their own base, with the opponent having an equivalent goal with respect to its base floor. Both agents are able to navigate to their opponent's base to capture their cube. However, π_{G_4} often finds it difficult to find the way back to its own base. Furthermore, it often gets tagged by the opponent, making it respawn at its initial spawn location. π_{G_5} on the other hand shows better navigational skills and usually finds its way back to its base after capturing the cube.

Hide and seek: hider. π_{G_4} moves somewhat randomly with abrupt changes in direction. This can make it hard for the opponent to keep seeing it. π_{G_5} on the other hand moves very specifically away from the co-player and often up the ramp and onto the side of the platform opposite the co-player. This forces the co-player to go around the ramp.

Hide and seek: seeker. π_{G_4} searches for the co-player throughout the world and then stands still once the co-player is in its vision. It does not anticipate the co-player's movement as it is about to come out of its vision. π_{G_5} prefers to continuously follow the co-player in order to be right next to it. In this way, it rarely lets the co-player out of its vision.

King of the hill. In this two-player task, the agent gets a reward if it is the only player at the top of the hill (touching the white floor). Once they get to the top of the hill, both π_{G_4} and π_{G_5} stay there and are able to push away the co-player whenever it comes near. However, π_{G_4} sometimes fails to navigate to the top of the hill, getting stuck in a loop. π_{G_5} is more consistent in its navigational abilities to get to the top of the hill.

XRPS Counter Yellow Sphere. In XRPS (Section 3.2.3), the agent can get points for holding any sphere, as long as its colour is not countered by the colour of the sphere the opponent is holding. However, the opponent player is goal-conditioned to hold the yellow sphere only. π_{G_4} tends

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

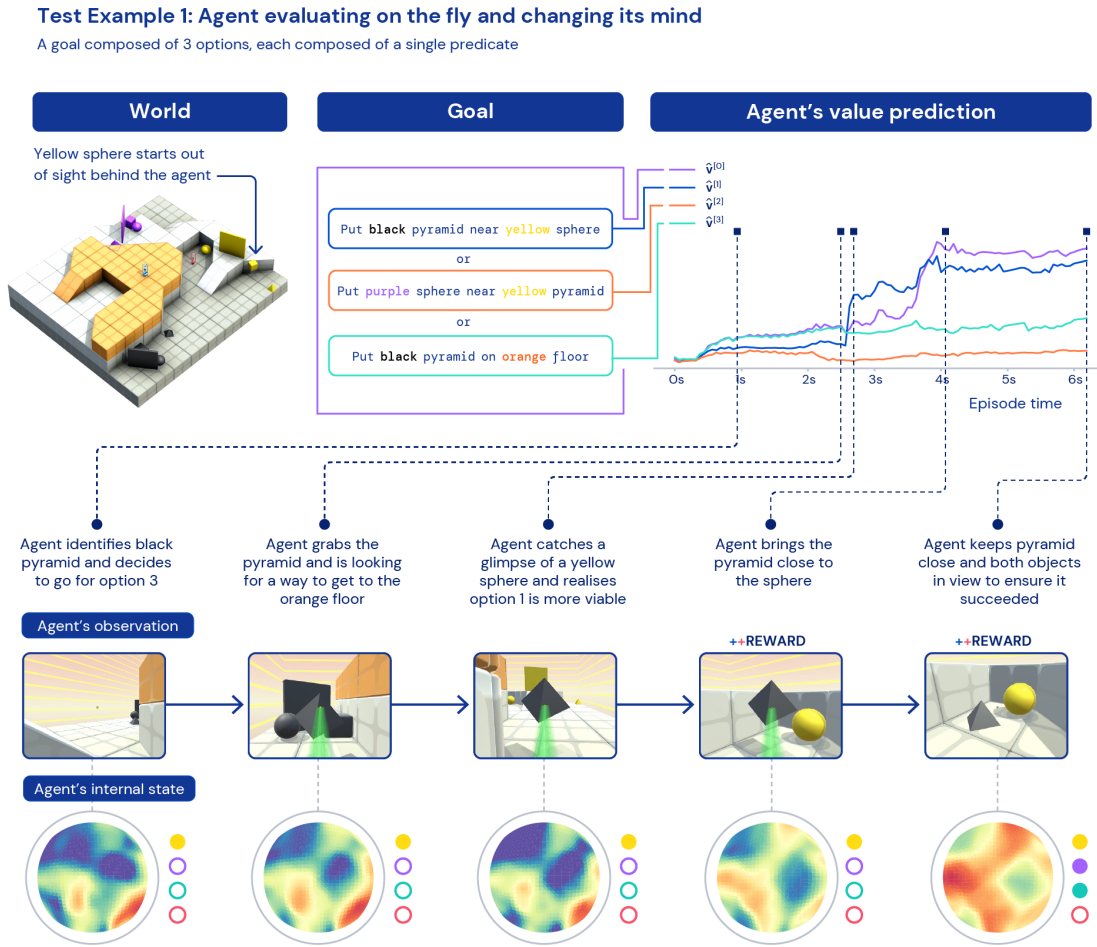A goal composed of 3 options, each composed of a single predicate

Figure 20 | **(Top)** From the left: rendering of the world; a goal composed of 3 options, each represented as a single predicate; Plots of the internal value function predictions of the GOAT module, with the colours corresponding to specific options. **(Middle)** Call-outs of 5 situations, from the perspective of the agent. **(Bottom)** A Kohonen Network representing the activity of the GOAT module (Section 6.6). The four coloured circles represent the Kohonen Neurons activity (from top): whether the agent is early in the episode (yellow), if it is optimistic about future rewards (purple), if it thinks it is in a rewarding state (cyan), if it thinks multiple atoms are missing (orange). See Figure 29 for more details.

to hold a sphere at random from the ones available. When this happens to be the black sphere, it gets no reward due to the co-player countering it with the yellow sphere. π_{G_5} on the other hand notices the co-player holding the yellow sphere and counters it by stealing the yellow sphere and holding it itself. It succeeds at holding it while the co-player tries to get it back. However, neither agent explicitly seeks to hold the purple sphere which would counter the opponent holding the yellow sphere.

Stop rolling. In this task, the agents have to keep a sphere from rolling to the bottom of a slope. The agents only get a reward if the sphere is not touching the bottom floor and is not being held. π_{G_4} simply lifts the sphere up in the air and lets it drop, gaining rewards for the brief moments when the sphere is dropping. π_{G_5} throws the sphere up the slope and then tries to block it from rolling down with its body. Often, π_{G_5} manages to corner the sphere between its body and the wall as the sphere is on the slope and scores rewards for the remainder of the episode without moving.

6.4.2 | Behavioural case studies

Let us now focus on 3 specific case studies showing interesting emergent behaviours.

On-the-fly option evaluation In Figure 20 we see an agent trying to solve a task with a goal consisting of 3 possible options. Initially, the agent does not see a yellow sphere, but it does see a black pyramid and the orange floor. Its third option rewards the agent for placing the black pyramid on the orange floor, and looking at the agent's internal option-values prediction, we see that indeed the value of the whole goal $\hat{v}_t^{[0]}$ (violet curve) is closest to the third option value $\hat{v}_t^{[3]}$ (green curve). Around 2.5s into the episode, the agent sees a yellow sphere, which leads to a dramatic increase in its internal prediction of what would happen if it was to satisfy option 1 instead ($\hat{v}_t^{[1]}$, blue curve), which rewards the agent for placing the black pyramid near the yellow sphere. As a result, the internal value function of the
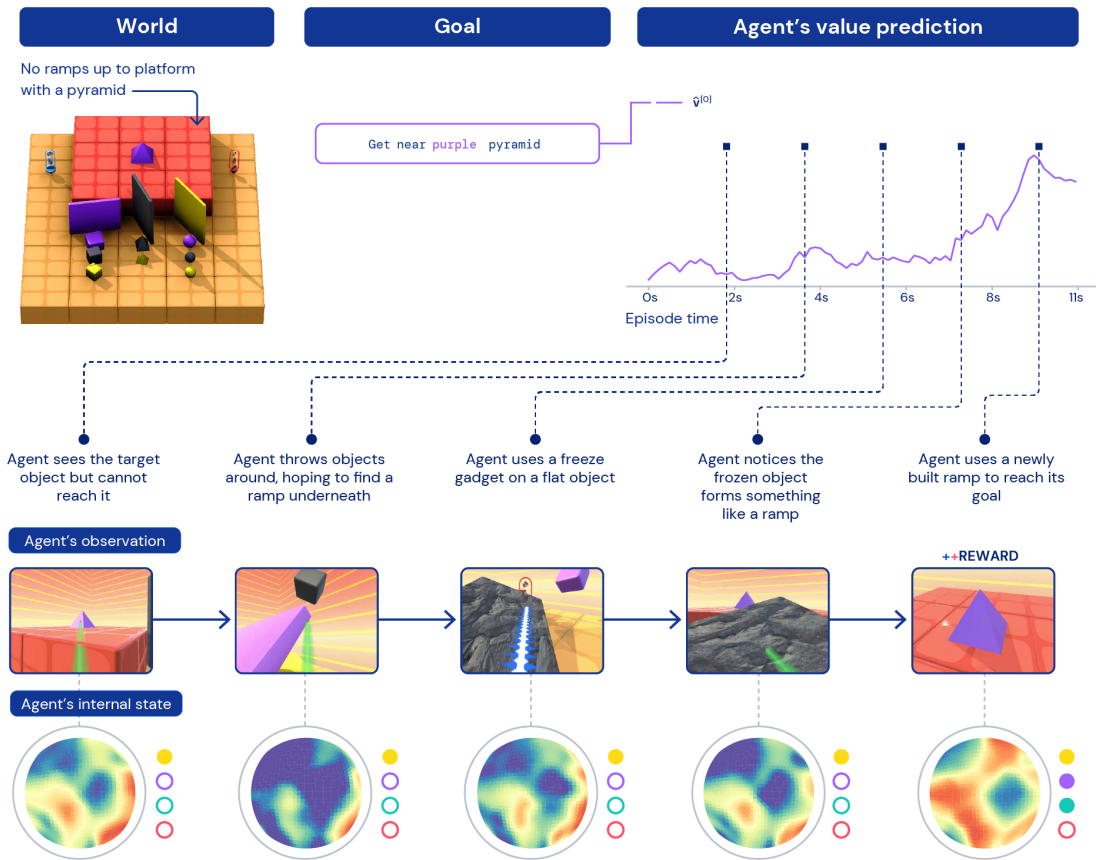
Figure 21 | **(Top)** From the left: rendering of the world; a goal composed of one option; Plots of the internal value function prediction of the agent. **(Middle)** Call-outs of 5 situations, from the perspective of the agent. **(Bottom)** A Kohonen Network representing the activity of the GOAT module (Section 6.6). The four coloured circles represent the Kohonen Neurons activity (from top): whether the agent is early in the episode (yellow), if it is optimistic about future rewards (purple), if it thinks it is in a rewarding state (cyan), if it thinks multiple atoms are missing (orange). See Figure 29 for more details.

whole game switches to upper bound the first option, and rather than navigating to the orange floor, the agent brings the black pyramid next to the sphere. This case study exemplifies the internal reasoning of the GOAT module, hinting at intentional decisions about which options to satisfy based on the current state of the environment.

Tool use In Figure 21 we see an agent placed in a world, where it needs to get near to a purple pyramid placed on a higher floor. However, in this world there is no ramp leading to the upper floor – this initial lack of accessibility is impossible to occur during training due to the procedural world generation process constraints. We observe the agent initially trying to move around the red block, looking for a ramp. It starts to throw various objects around, which can either be interpreted as looking for a ramp hidden underneath, or simply an emergent heuristic behaviour of trying to increase the entropy of the environment in a situation when the agent does not know what to do. Around 5 seconds into the episode a slab thrown by an agent lands in

the position partially supported by the upper floor, and the agent uses a freezing gadget to keep it in place. A moment later the agent can see a target purple pyramid in front of it with a frozen object looking like a ramp leading to the purple pyramid's floor, and its internal value estimate rapidly increases, suggesting that the agent understands that it has found a solution to the task. The agent navigates onto the frozen object and reaches its goal. We can see that the internal representation activity (described in Section 6.6) at 10 seconds is very similar to the final internal activity from the previous case study – we recognise this visual pattern as emerging when an agent is in a *content* state.

Experimentation Figure 22 is a final case study, where an agent is placed in a big open room, with most of the objects removed from the reachable space, and only 3 cubes left. The task the agent is facing is to put the black cube near the purple cube, the yellow cube near the purple cube, without putting the black and yellow cubes near each other. This simple logical puzzle requires an agent to figure out

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
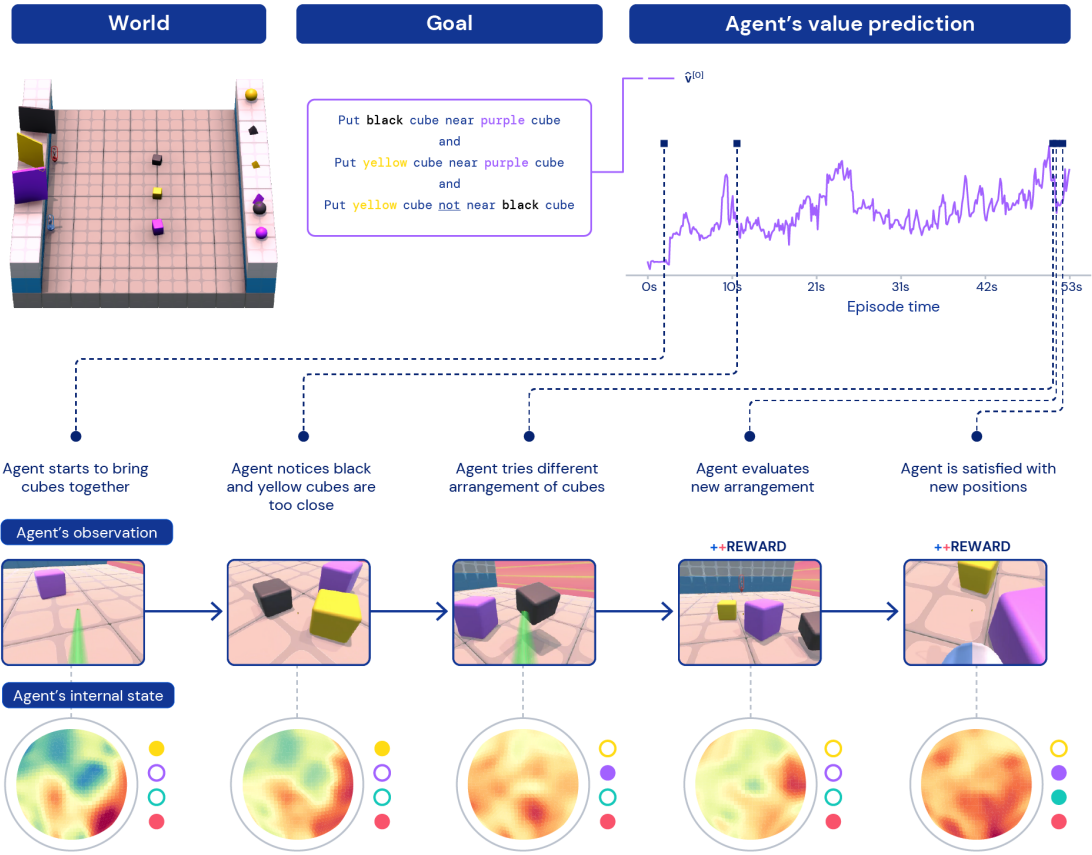

Figure 22 | **(Top)** From the left: rendering of the world; a goal composed of one options; Plots of the internal value function prediction of the agent. **(Middle)** Call-outs of 5 situations, from the perspective of the agent. **(Bottom)** A Kohonen Network representing the activity of the GOAT module (Section 6.6). The four coloured circles represent the Kohonen Neurons activity (from top): whether the agent is early in the episode (yellow), if it is optimistic about future rewards (purple), if it thinks it is in a rewarding state (cyan), if it thinks multiple atoms are missing (orange). See Figure 29 for more details.

that there is a spatial ordering that satisfies these principles: a line with the yellow cube, followed by the purple cube, followed by the black cube. Note, that whilst this does not look like a complex problem on a predicate level, it is a very hard exploration problem due to the physical instantiation of XLand tasks – namely

$$\frac{\#\{\mathbf{s}:r_g(\mathbf{s})=1\}}{\#\{\mathbf{s}:\mathbf{s}\in\mathcal{S}\}} \ll \frac{\#\{\phi(\mathbf{s}):r_g(\mathbf{s})=1\}}{N_\phi}.$$

From the agent's behaviour and internal value we can hypothesise that the agent is initially confused. It starts by bringing the cubes together. Then at around 10 seconds we can see it visually inspecting the scene with the yellow and black cubes too close, after which it tries to reshuffle them. This reshuffling process is repeated multiple times for the next few dozen seconds, until eventually around 50 seconds into the episode, the agent stumbles upon a spatial arrangement of the cubes that satisfies the goal, which the agent again inspects visually. Whilst still clearly not content when it comes looking at the agent's internal state/value prediction, the agent keeps the objects in the rewarding state

and stops shuffling the cubes. This within-episode experimentation behaviour could be a general heuristic *fallback* behaviour – when it lacks the ability to 0-shot generalise through understanding, it plays with the objects, experiments, and visually verifies if it solved the task – all of this as an emergent behaviour, a potential consequence of an open-ended learning process. Note, that agent does not perceive the reward, it has to infer it purely based on the observations.

6.4.3 | Multi-agent

We now investigate some emergent multiplayer dynamics between agents playing in specific probe games. We take 13 agent checkpoints through training of the final (5th) generation of our agent (checkpoint 1 is the earliest in training through to checkpoint 13 which is the latest in training). For each of the probe scenarios described below, we play every single pair of checkpointed policies against each other. This way we obtain $13^2 = 169$ matchups, and evaluate each pair of players on 1000 different worlds (to marginalise over
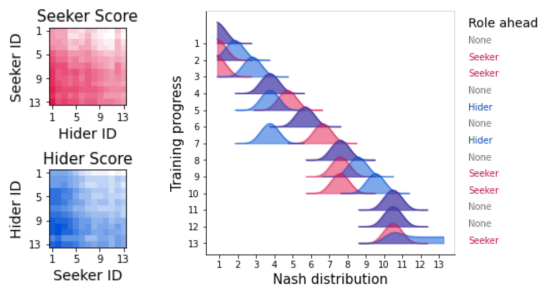
Figure 23 | **(Left)** Payoffs for the game of Hide and Seek played by checkpoints of the agent through training from start (1) to end (13), marginalised across 1000 different worlds, from the perspective of the seeker (top) and hider (bottom) player roles. (White is low, colour is high). **(Right)** The evolution of the Nash equilibrium distribution through training for each role of the player. One can note the back-and-forth dynamic of the hider and seeker improving over time. The agent never trained with these checkpoints.

physical instantiation), allowing us to study the development of strategies, social behaviours and learning dynamics of the agents in these games. Note, that the agent was never trained against these checkpoints, the only co-players it ever experienced during training were from the previous generations. More details can be found in Section A.9.

Hide and Seek   We revisit the game of hide and seek in this new evaluation context. It is an asymmetric, imbalanced, fully competitive game. Figure 23 shows the results of this experiment. With more training (later checkpoints), the results show the agents keep improving in the execution of both hider and seeker roles, showing a somewhat transitive strength progression, without exhibiting forgetting that can occur in multi-agent training (Czarnecki et al., 2020; Vinyals et al., 2019), and the Nash equilibrium is mostly centered around the newest agents. We can also see that there is a noticeable back-and-forth dynamic between the seeker and hider strategy, as initially the Nash equilibrium for the seeker stays at checkpoint 1 (early in training), whilst the hider Nash equilibrium keeps moving to the newest checkpoint. This suggests that the agent is gradually improving its hiding strategy. Later, we see the opposite – the hider Nash equilibrium stops moving, while the seeker Nash equilibrium keeps improving, with this switch happening multiple times. Note that this is being observed without the agents ever training against each other, thus we hypothesise that these developments have to be coming from agents acquiring new behaviours and strategies in other parts of XLand task space. In particular, it is worth noting that during training, whilst the agent does not play against its checkpoints, and even though the game of hide and seek itself is not a part of the training space, agents are facing games where their goal is to "see the other player", but the other player's goal will be something else. Consequently, even though they only train against a few instances of other agents, the space of behaviours they can encounter is enormous because these agents are themselves conditioned on a variety of goals.
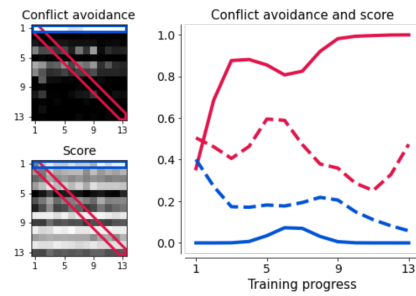

Figure 24 | **(Left)** Probabilities of each agent choosing to avoid conflict and the corresponding normalised scores, as a function of their total training time (1 meaning least trained, and 13 meaning the most trained). (White is low probability, black is high probability). **(Right)** Fraction of conflict avoiding behaviours (solid curve) and the corresponding normalised score (dashed curve). The agent becomes more conflict avoiding over time whilst preserving its performance when matched with a copy of itself (red curve), while the earlier agent playing against later agents is not avoiding conflict and its performance also keeps decreasing (blue curve).

Conflict Avoidance   We hypothesise that as training progresses agents might develop the behaviour of avoiding conflict with other agents in the situations where there is an alternative non-conflicting option to be satisfied. We create a simple game, where an agent can choose to place one of two spheres on a specific floor, while the other agent wants to put one of these spheres on a different floor. With both spheres being equidistant from a target floor, the only reason to pick the non-conflicting sphere is in order to avoid conflict with the other agent. In Figure 24 we can see that as the agent trains, it exhibits more and more conflict-avoiding behaviour, even though on average this does not necessarily lead to an increase in return on this particular task. However, empirically when early not-conflict-avoiding checkpoints play with increasingly trained checkpoints, they achieve a decreasing amount of reward. Note, that the agents are not training against each other, meaning that this development in behavioural response is purely an effect of the dynamic training distribution encountered during the open-ended learning process.

Chicken Game   In this experiment, we create an XLand version of a game-theoretic social dilemma called *Chicken*. In this setup, each agent can choose to either cooperate with its co-player or to try to dominate it. We observe two interesting trends with respect to the tendency to seek cooperative solutions in Figure 25. First, if an agent is playing with a checkpoint from very early in training, it tends to dominate it more. On the other hand, when playing with a copy of itself (self-play) its tendency to collaborate increases over training time. One simple explanation of this phenomenon is that for cooperation to work, both sides need to be capable of doing so. Consequently, it is perhaps harder to cooperate with a less capable agent. However, once facing someone of exactly same strength (self-play) the collaborative solution becomes preferred.
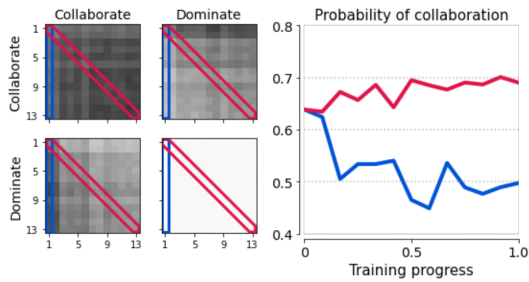
Figure 25 | **(Left)** Probabilities of each agent choosing to collaborate or dominate, as a function of their total training time (1 meaning least trained, and 13 meaning the most trained). (White is low probability, black is high probability). **(Right)** Fraction of collaborative behaviours in a Chicken-like game through agent training. The agent becomes more collaborative over time when matched with a copy of itself (red curve), and dominates more with earlier versions of itself (blue curve).
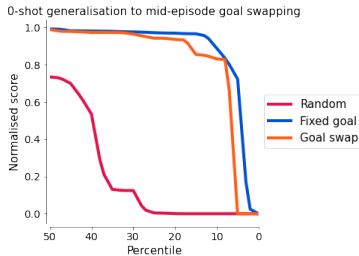

Figure 26 | Performance of the agent in 0-shot generalisation experiments where the goal of the agent is changed in the middle of an episode. Note that agents never perceived dynamically set goals during training.

6.4.4| *Goal Interventions*

During training our agents always received a single goal throughout an episode, the same goal at every timestep. We study whether the agent is able to adapt on-the-fly if this property is broken, and the goal changes mid-way through a single episode.

We sample 500 tasks all consisting of single option, one predicate games from the test set. We run the agent for an extended episode of 3/2 length of a regular episode, where in the first 1/3 the agent is given one goal (where we ignore its performance), and then we change the goal supplied to the agent to a different one. To simplify the setting, the co-players use the noop-policy, simulating a single-player game.

In Figure 26 we compare the normalised score of the agent evaluated in this scenario with the agent playing the same game but whose internal state is reset when the goal changes to simulate starting the episode from scratch with a fixed goal. We also show the performance of the agent taking random actions for reference. We notice that the performance of the agent with the changed goal is almost exactly the same as with a fixed goal, showing robustness to goal changes.

6.4.5| *Failed Hand-authored Tasks*

Whilst there are many tasks the agent participates in, there are also some hand-authored tasks the agent does not, never achieving a single reward. Some examples are:

Gap tasks Similar to the task in Figure 21, in this task there is an unreachable object which the agent is tasked with being near. The object is unreachable due to the existence of a chasm between the agent and object, with no escape route (once agent falls in the chasm, it is stuck). This task requires the agent to build a ramp to navigate over to reach the object. It is worth noting that during training no such inescapable regions exist. Our agents fall into the chasm, and as a result get trapped. It suggests that agents assume that they cannot get trapped.

Multiple ramp-building tasks Whilst some tasks do show successful ramp building (Figure 21), some hand-authored tasks require multiple ramps to be built to navigate up multiple floors which are inaccessible. In these tasks the agent fails.

Following task One hand-authored task is designed such that the co-player's goal is to be near the agent, whilst the agent's goal is to place the opponent on a specific floor. This is very similar to the test tasks that are impossible even for a human, however in this task the co-player policy acts in a way which follows the agent's player. The agent fails to lead the co-player to the target floor, lacking the theory-of-mind to manipulate the co-player's movements. Since an agent does not perceive the goal of the co-player, the only way to succeed in this task would be to experiment with the co-player's behaviour, which our agent does not do.

6.5| **Finetuning for Transfer**

Throughout this section we have so far demonstrated zero-shot generalisation to new tasks. The breadth of coverage of the agent's behaviour suggests that whilst zero-shot performance can be achieved on many out-of-distribution test tasks, there is the potential for very quick adaptation with finetuning.

Using a simple training setup – without PBT, dynamic task generation, or any other hyperparameter tuning – we finetune the weights of the generally capable agent previously analysed for 100 million steps (approximately 30 minutes of training) on a number of tasks from the hand-authored set. The results are shown in Figure 27.

The results show in all cases an increase in reward achieved by the finetuned agent compared to the zero-shot performance, with the finetuned agent showing a drastic improvement of 340% on average. By construction, the maximum reward that could ever be achieved on an XLand task of 900 timesteps is $V^*(x) \leq 900$. Using 900 as an upper bound of optimal reward per task (which is a very loose one, since even an optimal policy needs some time to reach objects of interest etc.), learning from scratch scores at least
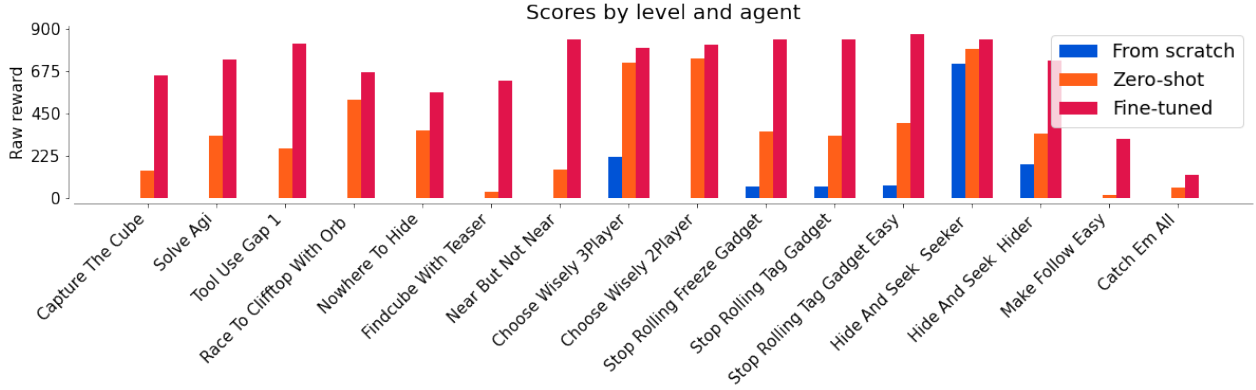
Figure 27 | Comparison of three agents from different training regimes on a range of hand-authored levels. Scratch: An agent trained from scratch for 100 million steps. Zero-shot: the agent trained using our methodology and evaluated on these held out levels zero-shot. Fine-tuned: the same agent but 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.

9% of the performance of the optimal policy, zero-shot performance is at the level of 39% and the finetuned agent achieves 77%. With the same computational budget and 30 minutes of training, learning from scratch on these tasks fails in the majority of tasks.

The task *Make Follow Easy* is described in the previous section as one of the tasks the agent fails to zero-shot generalise to. With 30 minutes of finetuning, the agent is able to achieve reward consistently in this task, learning successfully to coax the co-player to the target floor.

These experiments show the potential of massively multi-task RL pre-training, as is performed in this work, for the subsequent transfer with finetuning to many different downstream target tasks.

6.6 | Representation analysis

We now move our attention towards understanding how agents operate and the way they represent the simulated environment.

Kohonen Network   There are a multitude of methods to analyse the internal representations of agents and understand what knowledge is encoded in neuron activations (Goh et al., 2021) applicable in various situations. We utilise Kohonen Networks (also known as Self-Organising Maps) (Kohonen, 1982) to investigate the high dimensional representations learned by our agents. This technique unifies three types of analysis, allowing us to:

- visualise the space of internal representations wrt. some labelling (often done with T-SNE (Van der Maaten and Hinton, 2008)),
- visualise the current state of the agent (*i.e.* a single $\mathbf{h}_t$) (previously done, for example, with a Neural Response Map (Jaderberg et al., 2019)),
- conduct simple concept decoding tests (often done with linear classifier probes (Alain and Bengio, 2017) or single neuron activity analysis (Quiroga et al., 2005)).

A Kohonen Network is a collection of K neurons $\mathfrak{h}_i \in \mathfrak{H} := \mathbb{R}^n$ trained to represent a dataset composed of points $x_j \in \mathbb{R}^n$ under some notion of distance (here we use standard Euclidean distance), using a pre-determined structure between the neurons that prescribe the geometry one is looking for in the dataset. In our work we use neurons arranged as a grid filling a 2-dimensional circle, giving each neuron a fixed position $\mathfrak{k}_i \in \mathfrak{K} := \mathbb{R}^2$. To train the network, we iteratively minimise the following per iteration loss using gradient descent

$$\ell^{\mathfrak{H}}(\mathfrak{h}) := \sum_{i,j} \max\left\{0, \frac{\mathrm{d}_{\max}}{\mathrm{d}_{\max} - \|\mathfrak{k}_i - \mathfrak{k}_{\iota(x_j)}\|}\right\} \|x_j - \mathfrak{h}_i\|^2$$

$$\iota(x) := \arg\min_j \|x - \mathfrak{h}_j\|^2.$$

Intuitively, for each point in the dataset, the closest Kohonen Neuron is selected (the *winning* neuron) and moves the neuron a bit closer to this data point, together with other neurons that are nearby in grid $\mathfrak{K}$ space, with their adaptation downscaled proportionally to how far away from the winning neuron they are. By fitting the Kohonen Network to the data in this manner, we are asking *what 2d circle-like shape can fit into the n-dimensional dataset in such a way that its position corresponds to the density of the data?* More details can be found in Section A.11.

We gather 30k episodes of our trained agent across tasks sampled uniformly from the test set, and use activations x_j of the outputs of the LSTM, goal embedding module, and GOAT module to train three Kohonen Networks respectively. Next, we identified a collection of binary properties corresponding to state $\mathbf{s}_j$ represented in these episodes, *e.g.* whether it is early in the episode, whether the agent is holding an object, whether the agent is in a rewarding state, *etc.* For each probe property p we assign a colour to a specific Kohonen Neuron $\mathfrak{h}_i$ given by the fraction of data points containing the property relative to all the states that were mapped to this neuron:

$$c_{pi} := \frac{\#\{x_j : i = \iota(x_j) \wedge p(\mathbf{s}_j)\}}{\#\{x_j : i = \iota(x_j)\}}.$$

In Figure 28 one can see qualitatively that different properties are clearly represented in different parts of the network.
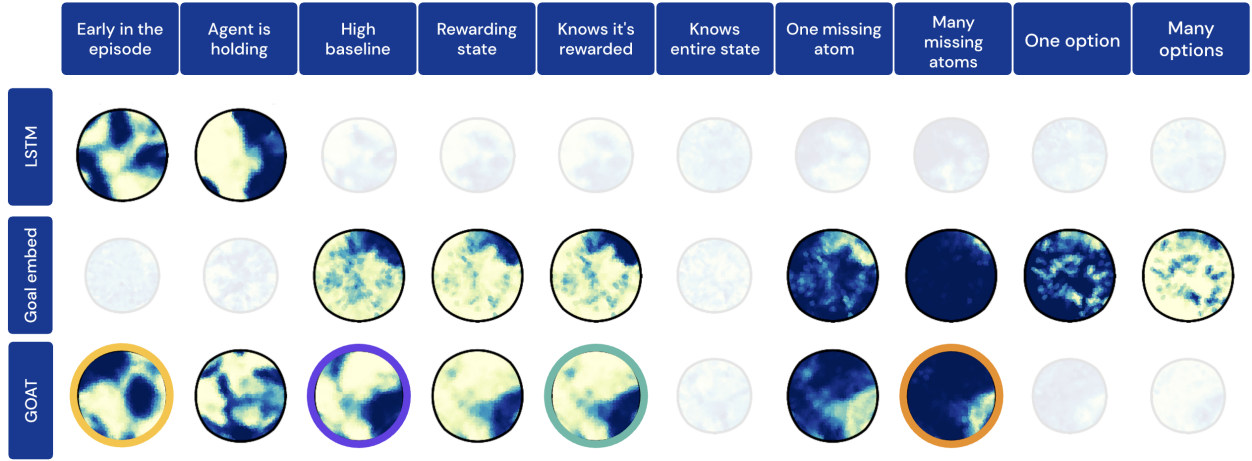
Figure 28 | Internal representation analysis of the agent. We use Kohonen Network representations of various properties for three different modules of the agent (LSTM, goal embedding, GOAT). Within a Kohonen Network, the bright yellow colour denotes states where the property is true, and blue where it 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).

To quantify this, we compute the *Balanced Accuracy* (BAC) of a classifier which assigns a label to each state by a majority vote of labels inside each cluster (*i.e.* set of points mapped onto a given neuron), formally:

$$\hat{p}_{T_{module}}(x) := c_{p\iota(x)} \geq T_{module}$$

for some threshold $T_{module} \in [0, 1]$, and we compute

$$BAC(\hat{p}, p) := \max_{T} \frac{1}{2} \left[\frac{TP[\hat{p}_T, p]}{TP[\hat{p}_T, p] + FN[\hat{p}_T, p]} + \frac{TN[\hat{p}_T, p]}{TN[\hat{p}_T, p] + FP[\hat{p}_T, p]} \right],$$

where TP, TN, FP, FN is the fraction of true positives, true negatives, false positives and false negatives from a predictor $\hat{p}$ and the ground truth p. We decide that the information is present in a specific representation if and only if $BAC(\hat{p}, p) \geq 0.8$, meaning that if we were to randomly select a state where the property is true or false, we could with at least 80% probability correctly guess this label based purely on the colour of the corresponding Kohonen Neuron.

Using this quantitative measure of information present in Figure 28, we can first see that the notion of the flow of time, and whether an agent is holding an object is clearly visible in the LSTM cell output, but is completely missing from the goal embedding module. It is however preserved at the output of the GOAT module, meaning that this information is probably useful for further policy/value predictions.

We can also see that the agent clearly internally represents that it is in a rewarding state. This is significant given that the agent does not receive its reward, nor the past rewards, as an input. The reward signal is used purely as part of RL training, so during inference the agent needs to be able to infer this information from its observations. Consequently, this implies that the agent is capable of using its RGB input to reason about the relations between objects, and their correspondence to the logical structure of the goal at hand. We further investigate whether this representation of a rewarding state is consistent with the agent's internal atomic predicate prediction (denoted in Figure 28 as *rewarding state and knows it*), where we further require all

the atomic predicate predictions that are relevant to the rewarding state (*i.e.* selected option) to be correct. We can see that this information is also very well represented. On the other hand, if we ask whether the agent represents the atomic predicates states of all relations involved in the goal (*i.e.* the atomic predicate states contributing to other options, that agent might not be pursuing right now) we see this information is not present in any of the modules we investigated. This suggests that agent has a very good, but focused, understanding of the state of the world, and attends mainly to the aspects of state that are relevant to the option it is currently following.

We can ask an analogous question of whether the agent is aware of how many atomic predicates states it needs to change before it can obtain a reward. The distinction between having to flip one atomic predicate or more is clearly encoded in the goal embedding module – with a small island of activations in the upper right corner corresponding to multiple missing atomic predicates, with the smooth big region around it corresponds to needing to flip exactly one. While this information is clearly preserved in the GOAT module output, we can see that they are mapped onto similar regions, suggesting that as the information is processed through the network and reaches the point where only policy/value needs to be produced, this distinction is potentially less relevant.

Finally, details regarding the exact game that an agent is playing (*e.g.* number of options involved) is clearly represented in its goal embedding module, but is then not propagated to the GOAT module, suggesting that whatever decision needs to be made that affects the policy/value can be done solely at the goal embedding level, and does not need to be integrated with the LSTM output.

Kohonen Neurons   An associated question that one could ask is whether there exists a single Kohonen Neuron coding for a specific property. Note that a Kohonen Neuron does not correspond to a single neuron in a neural network
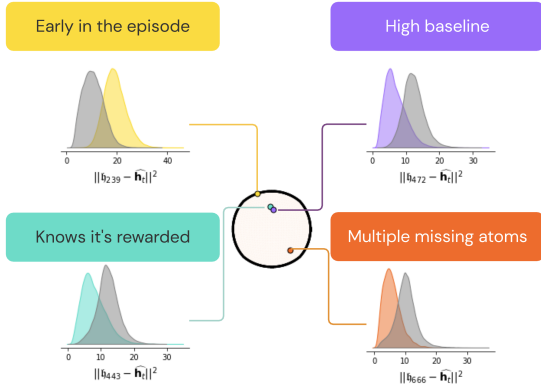
Figure 29 | Internal representation analysis of the agent. The Kohonen Neurons encode four well represented concepts from Figure 28. The kernel density estimation plots represent the density of the activity of the neuron when the concept is true (in colour) or false (in gray).
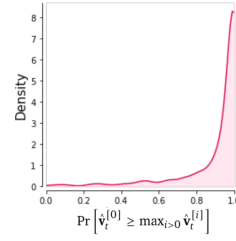

Figure 30 | The kernel density estimation of the fraction of frames inside a single episode where the agent's internal value estimation of the whole goal is lower bounded by the maximum value over options (*value consistency*, Theorem 5.1). We only consider goals with two and three options, as this property is trivially true for one option goals.

of the agent, but rather a collection of them, found using unsupervised training (and thus more related to general notions of distributed sparse representations than so called grandmother cells Connor (2005)). This can be seen more as a distributed concept, though not simply a linear classifier probe (Alain and Bengio, 2017), as the Kohonen Neuron is found without access to the corresponding labels.

$$\bar{p}_{i\mathrm{T}_{\mathrm{neuron}}}(x) := c_{pi} \geq \mathrm{T}_{\mathrm{neuron}},$$

and

$$\mathrm{BAC}(\bar{p}, p) := \max_{i,\mathrm{T}} \frac{1}{2} \left[\frac{\mathrm{TP}[\bar{p}_{i\mathrm{T}}, p]}{\mathrm{TP}[\bar{p}_{i\mathrm{T}}, p] + \mathrm{FN}[\bar{p}_{i\mathrm{T}}, p]} + \frac{\mathrm{TN}[\bar{p}_{i\mathrm{T}}, p]}{\mathrm{TN}[\bar{p}_{i\mathrm{T}}, p] + \mathrm{FP}[\bar{p}_{i\mathrm{T}}, p]} \right].$$

We note that for being early in the episode, having a high baseline, being in a rewarding state, and for multiple missing atomic predicates, we can identify corresponding Kohonen Neurons achieving BAC of over 75%, Figure 29.

Value consistency In Section 5.1 we discussed *value consistency*, the fact that an optimal policy value of the game composed of multiple alternatives is always lower bounded by the maximum value of the optimal policy for each separate option. Whilst the agent is encouraged to preserve a similar property over its current policy, it is not fully enforced. We investigate how consistent the trained agent is in this respect by looking at its internal values for each option and computing $\Pr\left[\hat{v}_t^{[0]} \geq \max_{i>0} \hat{v}_t^{[i]}\right]$. In Figure 30 we show the density estimation of episodes where a specific probability of value consistency occurs. In expectation, our agent is value consistent around 90% of the time (for the goals with more than one option, since by definition an agent is always value consistent with one option goals). Value consistency is clearly shown in a previously discussed example, Figure 20, with the value of the full game upper bounding values of the individual options, even as the individual option values fluctuate.

7 | Related Work

This work builds heavily upon the ideas of many related works. We now review some of these in the areas of multi-

agent learning and progressive learning, iterative improvement and percentiles, procedural environment generation, curriculum over tasks, curriculum over goals, and world-agent co-evolution.

Multi-agent and progressive learning. Our environment is multi-agent, and as such we face challenges of multi-agent learning systems, characterised previously as non-stationarity, exploration, and interaction modelling (Bowling, 2000; Lowe et al., 2017; Mahajan et al., 2019). Like others, we also see multi-agent reinforcement learning as a potential solution to other challenges, such as the design of autocurricula (Leibo et al., 2019) or even end-to-end learning of pixel-perception based agents (Jaderberg et al., 2019). The notion of generations of agents, forming a growing set (or *league* in Vinyals et al. (2019)) of agents lies at the core of many multi-agent learning algorithms (Balduzzi et al., 2019; Lanctot et al., 2017). The difference in this work is the utilisation of the generational split to encapsulate self-contained units of reinforcement learning such that the objective can change between generations, as well as the utilisation of a massive space of games being solved in parallel. This progressive growing of the set of agents on multi-task spaces is also related to progressive learning (Furlanello et al., 2018; Rusu et al., 2016; Schwarz et al., 2018), as well as multi-task learning with knowledge sharing (Teh et al., 2017). Sessa et al. (2020) proposes a mathematical framework of *contextual games*, which could be used to view XLand goal conditioned agents. They show an effective sampling strategy of scheduling games under an assumption of smoothness of mapping from contexts to optimal policies. From a formal standpoint the goal space of XLand forms a Boolean algebra and thus can benefit from exploitation of this structure (Nangue Tasse et al., 2020; Todorov, 2009; Van Niekerk et al., 2019). Currently, we exploit these properties in our GOAT module, as well as how we navigate game space to create games with specific properties. Vezhnevets et al. (2020) studies architectures and auxiliary losses (Jaderberg et al., 2017b) in a multi-agent setting with hindsight knowledge of agents' goals, which could be applied to our setting. Leibo et al. (2017) studies sequential social dilemma, in particular trying to identify well known social dilemma classes (Robinson and Goforth,

2005) in empirical payoffs emerging from RL simulations, which our multi-agent analysis draws upon. Automated identification of varied social dilemma in our setup is an interesting open research question.

Iterative improvement and percentiles. Iterative notions of improvements have been used, particularly in multi-agent systems, either explicitly by targeting the goal with respect to known opponents (Balduzzi et al., 2019; McMahan et al., 2003), implicitly by using internal ranking between agents (Jaderberg et al., 2019), or simply because of the reliance on self-play (Silver et al., 2016). In this work we use similar ideas but applied to worlds and games in addition to other agents (co-players), and propose a way to deal with non-comparable reward scales of the resulting tasks. When thinking about general capability and catastrophic failure of policies, the field of robust and risk sensitive reinforcement learning (Borkar and Jain, 2010; Prashanth and Ghavamzadeh, 2013; Tamar et al., 2012) has been analysing variability in obtained rewards to find safer solutions. In particular, percentile-based measures have been utilised (Delage and Mannor, 2010; Filar et al., 1995) to ensure/target specific guarantees of a probability of obtaining a reward in a given task. In this work we use similar ideas on the level of distribution over tasks, rather than on the level of individual policy outcomes. The use of curves of normalised score with respect to percentiles to visualise and characterise performance is inspired by ROC curves (Hanley and McNeil, 1982).

Procedural environment generation. Many previous works have used procedural generation and evolution to create interesting environments for players (both agents and humans). Togelius and Schmidhuber (2008) propose an evolving system to generate interesting rules for a game by selecting games in which random agents score poorly and trained agents score highly. Volz et al. (2018) use a Generative Adversarial Network (GAN, Goodfellow et al. (2014)) to generate Super Mario Bros levels. They further search the latent space of the GAN using evolutionary methods to discover levels that are difficult but achievable for a previously trained agent. Justesen et al. (2018) train an agent in a procedurally generated environment and update a difficulty parameter based on the agent's recent success rate – we make use of similar measures to influence task generation in our work. Grbic et al. (2020) evolve Minecraft levels, both via interactive and automated evolution. CPPN2GAN (Schrum et al., 2020) generates large diverse game levels by combining GANs, Content producing Compositional Pattern Producing Networks (CPPNs, (Stanley, 2007a)) and the NEAT evolutionary algorithm (Stanley and Miikkulainen, 2002). The GAN is first trained on a dataset of existing levels to reproduce individual rooms. A CPPN is then evolved to transform grid's coordinate locations into a latent representation that can be input to the GAN. The CPPN is evolved to maximise metrics such as the length of the shortest path to solve a level. In PCGRL (Khalifa et al., 2020), a deep RL agent is made to edit worlds in order to maximise a bespoke reward function, such as generating long paths for a maze.

Curriculum over tasks. Both our procedures for world-agent co-evolution (Section A.1.1) and dynamic task generation are examples of automated curriculum learning (ACL, Portelas et al. (2020b)). In ACL, the training distribution of the agent is automatically adapted throughout training. A number of methods attempt to use learning progress (Kaplan and Oudeyer, 2007; Schmidhuber, 2010) on a task as a way to decide whether the task should be trained on or not (Graves et al., 2017). In the context of reinforcement learning, this has been used to select tasks or task parameters (Kanitscheider et al., 2021; Matiisen et al., 2020; Portelas et al., 2020a). OpenAI et al. (2019) automatically adapt the parameters of their environment for solving a Rubik's cube with a robot hand. They start with an narrow domain distribution and continuously expand this distribution when the agent is seen to have good performance at its boundaries. Prioritised Experience Replay (Schaul et al., 2016) changes the distribution with which experience is replayed by prioritising those with high Temporal Difference (TD) error. Similarly, Jiang et al. (2020) propose Prioritised Level Replay which samples new levels to play on based on the observed TD error in recent experience on those levels. In CARML, Jabri et al. (2019) adapt the task distribution to form a curriculum for meta-RL by maximising the information between a latent task variable and their corresponding trajectories. In PowerPlay, Schmidhuber (2013) propose a framework to continuously seek the simplest unsolved challenge to train on. The adaptation of curricula for many of these works use hand-crafted heuristics, as we do with dynamic task generation, however in our case the parameters of the heuristic itself are adapted with PBT.

Curriculum over goals. A large body of work is concerned with the training of goal-conditioned agents (Schaul et al., 2015) in a single environment. In these past works, the goal usually consists of the position of the agent or a target observation to reach, however some previous work uses text goals (Colas et al., 2020) for the agent similarly to this work. When the goal is a target observation, most methods acquire new goals by sampling observations previously generated in the environment: Nair et al. (2018) generate visual goals by training a Variational Auto-Encoder (Kingma and Welling, 2014) over the generated experience. Hindsight Experience Replay (HER, Andrychowicz et al. (2017)) trains a goal-conditioned agent by replaying trajectories with the agent conditioned on the goal that was achieved in the trajectory. Fang et al. (2019) add a curriculum to Hindsight Experience Replay by dynamically changing the selection of trajectories for replay. Pong et al. (2020) propose a method to increase the importance of rarely sampled observation as goals. Warde-Farley et al. (2019) propose a variety of goal achievement reward functions which measure how similar a state is to the goal state. Racanière et al. (2020) perform a curriculum over environment goals in randomly initialised 2D and 3D worlds. A setter generates goals for a solver agent. The setter minimises a few different losses which aim to yield a wide variety of tasks of various difficulties for the current agent policy. CURIOUS (Colas et al., 2019) sets a curriculum over environment goals by prioritising goal spaces which have shown recent learning progress and then sampling goals uniformly over goal spaces.

Florensa et al. (2018) propose an adversarial goal generation procedure in which a goal-GAN generates goals for locomotion tasks that the agent must solve. The objective of the goal setter is similar to that used in our world-agent co-evolution procedure: guarantee that the success probability is within a preset range. Zhang et al. (2020) choose goals where there is high epistemic uncertainty on the Q-function. AMIGo (Campero et al., 2021) also generates a curriculum of goals for the agent but does so by looking at the current number of steps needed by the agent to reach the goal.

In Asymmetric self-play (OpenAI et al., 2021; Sukhbaatar et al., 2018), two agents interact in turn in the environment: Alice and Bob. Alice first plays in the environment and generates a trajectory. From there, Bob can either be tasked with returning the player to its original location, or, in a new episode, reaching the same state that Alice achieved. The self reward-play modification (Section 5.3) can be seen as a sequential version of this within a single episode and the same agent playing both Alice and Bob.

World-agent co-evolution. Our procedure for world-agent co-evolution (Section A.1.1) shares similarity with POET (Wang et al., 2019, 2020) and PAIRED (Dennis et al., 2020). In all cases, the procedure generates a dynamic high-dimensional world distribution for agents. In POET, a population of environment-agent pairs is evolved through time. Agents are continuously trained on their paired environment. Occasionally, agents are transferred to other environments in the population. In PAIRED, two agents are coevolved: a protagonist agent and an antagonist agent. The protagonist agent attempts to solve tasks generated by the antagonist agent. The antagonist also plays in the generated environments. The difference between the average score of the protagonist and the best score of the antagonist across multiple trials is defined as the regret. The protagonist is trained to minimise this regret while the antagonist is trained to maximise it. Compared with both these methods, our proposed procedure is simpler: it only requires a single agent to be trained to solve tasks. We filter levels only based on the agent's estimated probability of success. Finally, the use of the world-agent co-evolution process to create the base distribution for training and evaluation for the remainder of our learning process is an example of AI-generating algorithms (Clune, 2019).

8 | Conclusions

In this work, we introduced an open-ended 3D simulated environment space for training and evaluating artificial agents. We showed that this environment space, XLand, spans a vast, diverse, and smooth task space, being composed of procedurally generated worlds and multiplayer games. We looked to create agents that are generally capable in this environment space – agents which do not catastrophically fail, are competent on many tasks, and exhibit broad ability rather than narrow expertise. An iteratively revised metric of normalised score percentiles on an evaluation set of tasks was used to characterise general capability, and a learning process to drive iterative improvement created. This learning process is composed of agents training with deep RL, on

training task distributions that are dynamically generated in response to the agent's behaviour. Populations of agents are trained sequentially, with each generation of agents distilling from the best agent in the previous generation, iteratively improving the frontier of normalised score percentiles, whilst redefining the metric itself – an open-ended learning process.

Combining this environment space with such a learning process resulted in agents that appear to have broad ability across our held-out evaluation space, catastrophically failing on only a small percentage of tasks that are humanly impossible. We qualitatively and quantitatively characterised some of the emergent behaviours of this agent and saw general behavioural heuristics such as experimentation and success recognition, and the tendency to cooperate more with other competent agents, behaviours which appear to generalise to many out-of-distribution probe tasks. These behaviours are driven by rich internal representations that we analysed, showing clear representations of the structure and state of the goals they are tasked to follow.

These results hint at the ability to train agents, without human demonstrations, which exhibit general capabilities across vast task spaces. Beyond zero-shot generalisation, the ability to quickly finetune these pretrained agents on complex out-of-distribution tasks was demonstrated clearly. We hope the presented methods and results pave the way for future work on creating ever more adaptive agents that are able to transfer to ever more complex tasks.

Author Contributions

The following lists the main contributions of the authors to the work presented.

Adam Stooke: Learning process development and research investigations.
Anuj Mahajan: Agent analysis.
Catarina Barros: Environment development and visuals.
Charlie Deck: Environment development.
Jakob Bauer: Infrastructure development, learning process development, research investigations, and technical management.
Jakub Sygnowski: Infrastructure development, agent analysis, and research investigations.
Maja Trebacz: Research investigations.
Max Jaderberg: Learning process development, research investigations, manuscript, visuals, XLand concept, and team management.
Michael Mathieu: Learning process development and research investigations.
Nat McAleese: Infrastructure development and research investigations.
Nathalie Bradley-Schmieg: Program management.
Nathaniel Wong: Environment development and visuals.
Nicolas Porcel: Environment development.
Roberta Raileanu: Research investigations.
Steph Hughes-Fitt: Program management.
Valentin Dalibard: Learning process development, infrastructure development, research investigations, agent analysis, and manuscript.

References

G. Alain and Y. Bengio. Understanding intermediate layers using linear classifier probes. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=HJ4-rAVtl.

M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf.

D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 9780898716245.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

B. Baker, I. Kanitscheider, T. M. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch. Emergent tool use from multi-agent autocurricula. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SkxpxJBKwS.

D. Balduzzi, K. Tuyls, J. Perolat, and T. Graepel. Re-evaluating evaluation. In S. Bengio,

H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/cdf1035c34ec380218a8cc9a43d438f9-Paper.pdf.

D. Balduzzi, M. Garnelo, Y. Bachrach, W. Czarnecki, J. Perolat, M. Jaderberg, and T. Graepel. Open-ended learning in symmetric zero-sum games. In *International Conference on Machine Learning*, pages 434–443. PMLR, 2019.

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, et al. DeepMind Lab. *arXiv preprint arXiv:1612.03801*, 2016.

C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

V. Borkar and R. Jain. Risk-constrained Markov decision processes. *49th IEEE Conference on Decision and Control (CDC)*, pages 2664–2669, 2010.

M. Bowling. Convergence problems of general-sum multiagent reinforcement learning. In *ICML*, pages 89–94, 2000.

J. C. Brant and K. O. Stanley. Minimal criterion coevolution: a new approach to open-ended search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 67–74, 2017.

A. Campero, R. Raileanu, H. Küttler, J. B. Tenenbaum, T. Rocktäschel, and E. Grefenstette. Learning with amigo: Adversarially motivated intrinsic goals. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=ETBc_MIMgoX.

J. Clune. AI-GAs: AI-generating algorithms, an alternate paradigm for producing general artificial intelligence. *arXiv preprint arXiv:1905.10985*, 2019.

C. Colas, P. Oudeyer, O. Sigaud, P. Fournier, and M. Chetouani. CURIOUS: intrinsically motivated modular multi-goal reinforcement learning. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1331–1340. PMLR, 2019. URL http://proceedings.mlr.press/v97/colas19a.html.

C. Colas, A. Akakzia, P.-Y. Oudeyer, M. Chetouani, and O. Sigaud. Language-conditioned goal generation: a new approach to language grounding for RL. *arXiv preprint arXiv:2006.07043*, 2020.

C. E. Connor. Friends and grandmothers. *Nature*, 435(7045): 1036–1037, 2005.

W. M. Czarnecki, R. Pascanu, S. Osindero, S. Jayakumar, G. Swirszcz, and M. Jaderberg. Distilling policy distillation. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1331–1340. PMLR, 2019.

W. M. Czarnecki, G. Gidel, B. Tracey, K. Tuyls, S. Omidshafiei, D. Balduzzi, and M. Jaderberg. Real world games look like spinning tops. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 17443–17454. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/ca172e964907a97d5ebd876bfdd4adbd-Paper.pdf.

B. A. Davey and H. A. Priestley. *Introduction to lattices and order*. Cambridge university press, 2002.

DeepMind. Sonnet. https://github.com/deepmind/sonnet, 2020.

E. Delage and S. Mannor. Percentile optimization for Markov decision processes with parameter uncertainty. *Oper. Res.*, 58(1):203–213, Jan. 2010. ISSN 0030-364X. doi: 10.1287/opre.1080.0685. URL https://doi.org/10.1287/opre.1080.0685.

M. Dennis, N. Jaques, E. Vinitsky, A. Bayen, S. Russell, A. Critch, and S. Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 13049–13061. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/985e9a46e10005356bbaf194249f6856-Paper.pdf.

M. Fang, T. Zhou, Y. Du, L. Han, and Z. Zhang. Curriculum-guided hindsight experience replay. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/83715fd4755b33f9c3958e1a9ee221e1-Paper.pdf.

J. A. Filar, D. Krass, and K. W. Ross. Percentile performance criteria for limiting average Markov decision processes. *IEEE Transactions on Automatic Control*, 40(1): 2–10, 1995.

C. Florensa, D. Held, X. Geng, and P. Abbeel. Automatic goal generation for reinforcement learning agents. In J. Dy

and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1515–1528. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/florensa18a.html.

T. Furlanello, Z. Lipton, M. Tschannen, L. Itti, and A. Anandkumar. Born again neural networks. In *International Conference on Machine Learning*, pages 1607–1616. PMLR, 2018.

M. Garnelo, W. M. Czarnecki, S. Liu, D. Tirumala, J. Oh, G. Gidel, H. van Hasselt, and D. Balduzzi. Pick your battles: Interaction graphs as population-level objectives for strategic diversity. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1501–1503, 2021.

G. Goh, N. C. †, C. V. †, S. Carter, M. Petrov, L. Schubert, A. Radford, and C. Olah. Multimodal neurons in artificial neural networks. *Distill*, 2021. doi: 10.23915/distill.00030. https://distill.pub/2021/multimodal-neurons.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu. Automated curriculum learning for neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1311–1320. JMLR.org, 2017.

D. Grbic, R. B. Palm, E. Najarro, C. Glanois, and S. Risi. Evocraft: A new challenge for open-endedness. *CoRR*, abs/2012.04751, 2020. URL https://arxiv.org/abs/2012.04751.

M. Gumin. Wave Function Collapse Algorithm, 2016. URL https://github.com/mxgmn/WaveFunctionCollapse.

D. Ha. Generating abstract patterns with TensorFlow. *blog.otoro.net*, 2016. URL https://blog.otoro.net/2016/03/25/generating-abstract-patterns-with-tensorflow/.

J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36, 1982.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

J. Heinrich and D. Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3796–3803, 2019.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

A. Jabri, K. Hsu, A. Gupta, B. Eysenbach, S. Levine, and C. Finn. Unsupervised curricula for visual meta-reinforcement learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/d5a28f81834b6df2b6db6d3e5e2635c7-Paper.pdf.

M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017a.

M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017b. URL https://openreview.net/forum?id=SJ6yPD5xg.

M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

M. Jiang, E. Grefenstette, and T. Rocktäschel. Prioritized level replay. *CoRR*, abs/2010.03934, 2020. URL https://arxiv.org/abs/2010.03934.

N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. *NeurIPS 2018 Workshop on Deep Reinforcement Learning*, 2018.

I. Kanitscheider, J. Huizinga, D. Farhi, W. H. Guss, B. Houghton, R. Sampedro, P. Zhokhov, B. Baker, A. Ecoffet, J. Tang, O. Klimov, and J. Clune. Multi-task curriculum learning in a complex, visual, hard-exploration domain: Minecraft. *CoRR*, abs/2106.14876, 2021. URL https://arxiv.org/abs/2106.14876.

F. Kaplan and P.-Y. Oudeyer. In search of the neural circuits of intrinsic motivation. *Frontiers in Neuroscience*, 1:17, 2007. ISSN 1662-453X. doi: 10.3389/neuro.01.1.1.017.2007. URL https://www.frontiersin.org/article/10.3389/neuro.01.1.1.017.2007.

A. Khalifa, P. Bontrager, S. Earle, and J. Togelius. PCGRL: procedural content generation via reinforcement learning. *CoRR*, abs/2001.09212, 2020. URL https://arxiv.org/abs/2001.09212.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In Y. Bengio and Y. LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6114.

D. J. Klein and M. Randić. Resistance distance. *Journal of mathematical chemistry*, 12(1):81–95, 1993.

T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69, 1982.

M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Pérolat, D. Silver, and T. Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4193–4206, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

J. Z. Leibo, V. F. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In K. Larson, M. Winikoff, S. Das, and E. H. Durfee, editors, *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, pages 464–473. ACM, 2017. URL http://dl.acm.org/citation.cfm?id=3091194.

J. Z. Leibo, E. Hughes, M. Lanctot, and T. Graepel. Autocurricula and the emergence of innovation from social interaction: A manifesto for multi-agent intelligence research. *CoRR*, abs/1903.00742, 2019. URL http://arxiv.org/abs/1903.00742.

S. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theor.*, 28(2):129–137, Sept. 2006. ISSN 0018-9448. doi: 10.1109/TIT.1982.1056489. URL https://doi.org/10.1109/TIT.1982.1056489.

R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6382–6393, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

A. Mahajan, T. Rashid, M. Samvelyan, and S. Whiteson. MAVEN: Multi-agent variational exploration. In *Advances in Neural Information Processing Systems*, pages 7613–7624, 2019.

L. Marris, P. Muller, M. Lanctot, K. Tuyls, and T. Graepel. Multi-agent training beyond zero-sum with correlated equilibrium meta-solvers. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 7480–7491. PMLR, 2021. URL http://proceedings.mlr.press/v139/marris21a.html.

T. Matiisen, A. Oliver, T. Cohen, and J. Schulman. Teacher–student curriculum learning. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9):3732–3740, 2020. doi: 10.1109/TNNLS.2019.2934906.

H. B. McMahan, G. J. Gordon, and A. Blum. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 536–543, 2003.

A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. Visual reinforcement learning with imagined goals. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/7ec69dd44416c46745f6edd947b470cd-Paper.pdf.

G. Nangue Tasse, S. James, and B. Rosman. A boolean task algebra for reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9497–9507. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/6ba3af5d7b2790e73f0de32e5c8c1798-Paper.pdf.

J. F. Nash et al. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

F. Nielsen. Closed-form information-theoretic divergences for statistical mixtures. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 1723–1726. IEEE, 2012.

OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019. URL http://arxiv.org/abs/1910.07113.

OpenAI, M. Plappert, R. Sampedro, T. Xu, I. Akkaya, V. Kosaraju, P. Welinder, R. D'Sa, A. Petron, H. P. de Oliveira Pinto, A. Paino, H. Noh, L. Weng, Q. Yuan, C. Chu, and W. Zaremba. Asymmetric self-play for automatic goal discovery in robotic manipulation. *CoRR*, abs/2101.04882, 2021. URL https://arxiv.org/abs/2101.04882.

R. Penrose. A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, volume 51, pages 406–413. Cambridge University Press, 1955.

V. Pong, M. Dalal, S. Lin, A. Nair, S. Bahl, and S. Levine. Skew-fit: State-covering self-supervised reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 7783–7792. PMLR, 2020. URL http://proceedings.mlr.press/v119/pong20a.html.

R. Portelas, C. Colas, K. Hofmann, and P.-Y. Oudeyer. Teacher algorithms for curriculum learning of deep RL in continuously parameterized environments. In *Conference on Robot Learning*, pages 835–853. PMLR, 2020a.

R. Portelas, C. Colas, L. Weng, K. Hofmann, and P.-Y. Oudeyer. Automatic curriculum learning for deep RL: A short survey. In C. Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4819–4825. International Joint Conferences on Artificial Intelligence Organization, 7 2020b. doi: 10.24963/ijcai.2020/671. URL https://doi.org/10.24963/ijcai.2020/671. Survey track.

L. Prashanth and M. Ghavamzadeh. Actor-critic algorithms for risk-sensitive MDPs. In *Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 1*, pages 252–260, 2013.

R. Q. Quiroga, L. Reddy, G. Kreiman, C. Koch, and I. Fried. Invariant visual representation by single neurons in the human brain. *Nature*, 435(7045):1102–1107, 2005.

S. Racanière, A. K. Lampinen, A. Santoro, D. P. Reichert, V. Firoiu, and T. P. Lillicrap. Automated curriculum generation through setter-solver interactions. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=H1e0Wp4KvH.

D. Robinson and D. Goforth. *The topology of the 2x2 games: a new periodic table*, volume 3. Psychology Press, 2005.

A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/schaul15.html.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *International Conference on Learning Representations*, Puerto Rico, 2016.

J. Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

J. Schmidhuber. Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in psychology*, 4:313, 2013.

S. Schmitt, J. J. Hudson, A. Zidek, S. Osindero, C. Doersch, W. M. Czarnecki, J. Z. Leibo, H. Kuttler, A. Zisserman, K. Simonyan, et al. Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835*, 2018.

B. Scholkopf. The kernel trick for distances. *Advances in neural information processing systems*, pages 301–307, 2001.

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

J. Schrum, V. Volz, and S. Risi. CPPN2GAN: Combining compositional pattern producing networks and gans for large-scale pattern generation. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, page 139–147, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371285. doi: 10.1145/3377930.3389822. URL https://doi.org/10.1145/3377930.3389822.

J. Schwarz, W. Czarnecki, J. Luketina, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pages 4528–4537. PMLR, 2018.

P. G. Sessa, I. Bogunovic, A. Krause, and M. Kamgarpour. Contextual games: Multi-agent learning with side information. *Advances in Neural Information Processing Systems*, 33, 2020.

N. Shaker, J. Togelius, and M. J. Nelson. *Procedural content generation in games*. Springer, 2016.

Y. Shoham, R. Powers, and T. Grenager. If multi-agent learning is the answer, what is the question? *Artificial intelligence*, 171(7):365–377, 2007.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529 (7587):484–489, 2016.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

B. W. Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

H. F. Song, A. Abdolmaleki, J. T. Springenberg, A. Clark, H. Soyer, J. W. Rae, S. Noury, A. Ahuja, S. Liu, D. Tirumala, N. Heess, D. Belov, M. A. Riedmiller, and M. M. Botvinick. V-MPO: on-policy maximum a posteriori policy optimization for discrete and continuous control. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SylOlp4FvH.

K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162, June 2007a. ISSN 1389-2576. doi: 10.1007/s10710-007-9028-8. URL https://doi.org/10.1007/s10710-007-9028-8.

K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007b.

K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, June 2002. ISSN 1063-6560. doi: 10.1162/106365602320169811. URL https://doi.org/10.1162/106365602320169811.

S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=SkT5Yg-RZ.

A. Tamar, D. Di Castro, and S. Mannor. Policy gradients with variance related risk criteria. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 1651–1658, Madison, WI, USA, 2012. Omnipress. ISBN 9781450312851.

Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu. Distral: Robust multitask reinforcement learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/0abdc563a06105aee3c6136871c9f4d1-Paper.pdf.

E. Todorov. Compositionality of optimal control laws. *Advances in neural information processing systems*, 22:1856–1864, 2009.

J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *2008 IEEE Symposium on Computational Intelligence and Games, CIG 2008*, 2008 IEEE Symposium on Computational Intelligence and Games, CIG 2008, pages 111–118, 2008. ISBN 9781424429745. doi: 10.1109/CIG.2008.5035629. Copyright: Copyright 2009 Elsevier B.V., All rights reserved.; 2008 IEEE Symposium on Computational Intelligence and Games, CIG 2008 ; Conference date: 15-12-2008 Through 18-12-2008.

L. Van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

H. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver. Learning values across many orders of magnitude. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4294–4302, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

B. Van Niekerk, S. James, A. Earle, and B. Rosman. Composing value functions in reinforcement learning. In *International Conference on Machine Learning*, pages 6401–6409. PMLR, 2019.

A. Vezhnevets, Y. Wu, M. Eckstein, R. Leblond, and J. Z. Leibo. Options as responses: Grounding behavioural hierarchies in multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 9733–9742. PMLR, 2020.

O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354, 2019.

V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, page 221–228, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356183. doi: 10.1145/3205455.3205517. URL https://doi.org/10.1145/3205455.3205517.

R. Wang, J. Lehman, J. Clune, and K. O. Stanley. POET: open-ended coevolution of environments and their optimized solutions. In A. Auger and T. Stützle, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, pages 142–151. ACM, 2019. doi: 10.1145/3321707.3321799. URL https://doi.org/10.1145/3321707.3321799.

R. Wang, J. Lehman, A. Rawal, J. Zhi, Y. Li, J. Clune, and K. O. Stanley. Enhanced POET: open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 9940–9951. PMLR, 2020. URL http://proceedings.mlr.press/v119/wang20l.html.

T. Ward, A. Bolt, N. Hemmings, S. Carter, M. Sanchez, R. Barreira, S. Noury, K. Anderson, J. Lemmon, J. Coe, P. Trochim, T. Handley, and A. Bolton. Using Unity to help solve intelligence, 2020. URL https://arxiv.org/abs/2011.09294.

D. Warde-Farley, T. V. de Wiele, T. D. Kulkarni, C. Ionescu, S. Hansen, and V. Mnih. Unsupervised control through non-parametric discriminative rewards. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=r1eVMnA9K7.

Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu. Deep mutual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4320–4328, 2018.

Y. Zhang, P. Abbeel, and L. Pinto. Automatic curriculum learning through value disagreement. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/566f0ea4f6c2e947f36795c8f58ba901-Abstract.html.
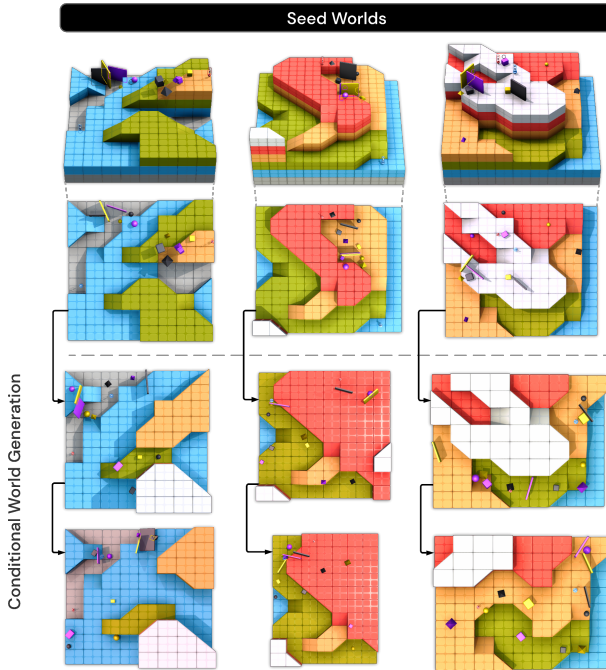
Figure 31 | Worlds can be generated conditioned on an existing world. This allows smooth variation of worlds. This figures shows three examples of this process, with each column showing an initial seed world and the rows showing two steps of conditional world generation.

A| Appendix

A.1| Worlds

A.1.1| Procedural World Generation

Figure 32 gives an overview of the generation process which we now describe in more detail for each of the components.

Topology Tile assignments are procedurally generated using the Wave Function Collapse algorithm (WFC) (Gumin, 2016). WFC acts as a constraint satisfaction algorithm, acting on a set of tiles which correspond to 3D geometry such as floor elements of different heights, ramps, and diagonal floor pieces with associated connectivity rules. WFC iteratively samples a grid location, samples a tile to be assigned to that location, and then updates each remaining grid locations' probability distribution over what tiles can be sampled given the constraints defined by tile connectivity rules. This process repeats, and ends when all grid locations have an assigned tile. The largest connected component of exposed floor is defined as the playable area of the world. Finally, a random scaling of the three dimensions of the tile elements is performed to create non-cuboidal tile elements, and random lighting applied (location, direction, intensity, hue). We additionally randomly apply reflections of topology to sometimes create symmetric worlds. The result is the ability to procedurally generate a wide variety of convex topologies composed of varied yet coherent structures.

Objects An object's initial position in the world is determined by sampling from a 2D probability map corresponding to the top-down map of the world topology, with non-zero values in the playable area of the world, and subsequently positioning the object in 3D at the floor level at the sampled 2D point. The probability map is given by a fixed parameter Compositional Pattern-Producing Network (CPPN) (Ha, 2016; Stanley, 2007b) which takes in the 2D position, height, and tile identity at each position in the map, and a position-independent latent variable associated with the instance of the object. This allows the probability of object placement to be associated with certain floors, elements of the topology, absolute or relative locations, in a highly non-linear manner determined by the latent variable. Object instances have a randomly sampled size, a colour, and a shape. There are three colours – black, purple, yellow – and four shapes – cube, sphere, pyramid, slab. Object locations can be sampled independently as per the process previously described, or in spatial groups clustered by shape or colour.

Conditional world generation The mechanisms described so far allow us to create a world generating function, where each world sample is drawn independently $\mathbf{w} \sim P_{\mathcal{W}}(\cdot)$. However, it is also possible to condition the world sampling such that a new world $\hat{\mathbf{w}} \sim P_{\mathcal{W}}(\mathbf{w})$ is similar to a given world $\mathbf{w}$. To achieve this for the topology, we can bias the initial probability over each grid location by the delta function of the conditioned worlds topology. For the object and player locations we add Gaussian noise to the latent variable associated with each object and player, and for all other categorically sampled quantities we resample. Some examples of this process is shown in Figure 31. We show in Section 3.1 that this results in the ability to smoothly vary worlds and can be used to generate worlds via an evolutionary process.

Game conditioned worlds We can also condition the generation of worlds such that a particular game $\mathbf{G}$ is achievable given the topology and layout of objects. We define a function $\mathbf{w_G} = f(\mathbf{w}, \mathbf{G})$ which takes an existing world $\mathbf{w}$ and a game $\mathbf{G}$ and returns a new world $\mathbf{w_G}$ such that all players, objects, and topological elements (*e.g.* floors) referenced in the game will be present and exposed in the playable area.

World-agent co-evolution Whilst our procedural world generation function $P_{\mathcal{W}}(\cdot)$ has a vast and diverse support, it is interesting to consider how to shift the distribution towards more interesting worlds, particularly those that may pose navigational challenges to players. To explore this, we created an RL training process to train an agent to maximise reward on a dynamic set of worlds, but with a static game which always consist of a single player (the agent) with the goal *"Be near a yellow cube"*. Our procedure maintains two sets of worlds which dynamically change as the agent trains: the train set and the evaluate set. When the agent requests a world to train on, we sample one uniformly from the train set.

The train set initially contains only a world consisting of an open room (Figure 33 (left)) and the evaluate set
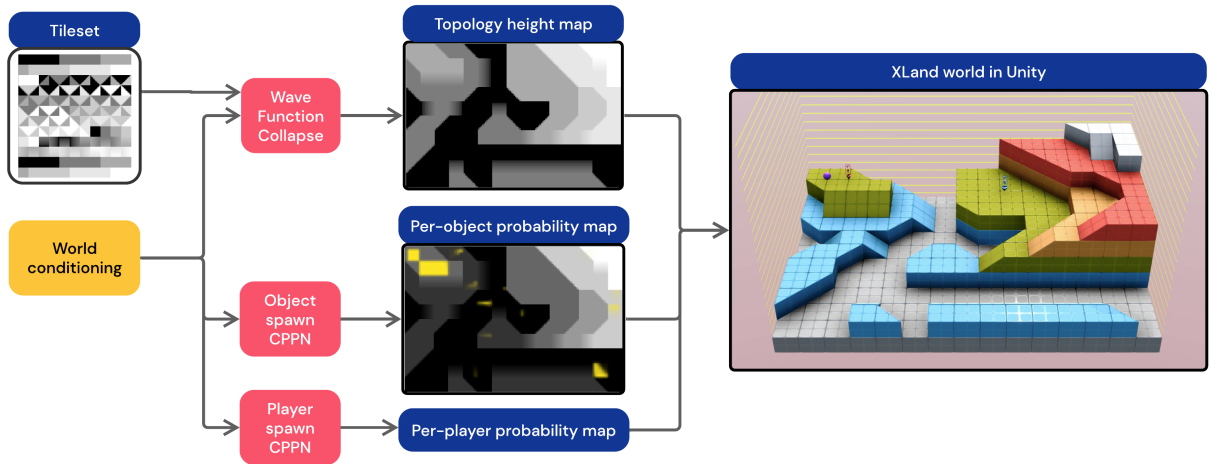
Figure 32 | The steps of procedural world generation. The process is conditioned on a seed and optionally an existing world to be similar to. Wave Function Collapse (Gumin, 2016) acting on a tileset of primitive building blocks creates a height map of the topology. Additionally, for each object and player, 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.

is set to be empty. An evolutionary algorithm governs the progression of the train worlds similarly to criterion co-evolution (Brant and Stanley, 2017). At regular intervals, our procedure attempts to generate a new world to add to the train set. It selects a random world from the existing train set. It then mutates this world using the conditional world generation process described previously, $\mathbf{w}_{\text{child}} \sim P_{\mathcal{W}}(\mathbf{w}_{\text{parent}})$. This new world is added into the evaluate set. To evaluate a world in the evaluate set, we make the agent play in the world for 100 episodes. None of these episodes are used for RL training. 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. For each world in the train set, we also monitor the agent scores across the last 100 episodes and discard the world if the agent does not still meet this criterion.

The agent is therefore continually training on worlds which pose some navigational challenge, though not too challenging. Since the fitness of worlds is related to the agent's behaviour, as the agent trains and improves in its navigational behaviour, we observe the complexity of the worlds in the train set continually increases. Figure 33 (middle) illustrates the train set of this process as both the agent and the train world distribution co-evolve. We can see in Figure 33 (right) how these worlds exhibit some interesting features such as long navigational paths, forks in paths that can be taken, thin paths such that the agent easily fall off, and hidden goal objects.

A.1.2 | Counting worlds

We would like to count how many $w \times h$ worlds are there, such that there exists a region A with the following properties:

- its size is at least $\frac{wh}{2}$.
- for every two points in A there exists a path between

them.
- there is no path that leads from A outside (and thus there are no irreversible decisions of leaving the region).

Due to the complexity of this task, we provide a lower and upper bound rather than the exact number of worlds. The upper bound is trivial: we have 6 possible flat tiles (one per level), 4 possible orientations of ramps and 4 possible orientations of "diagonal tile". Consequently, we have at most $6^{w \cdot h} \cdot (1 + 4 + 4)^{w \cdot h}$ such worlds. We now turn our attention to a lower bound. Let us take a world and focus on every other tile (and thus operate on the $\lceil \frac{w}{2} \rceil \times \lceil \frac{h}{2} \rceil$ subgrid, $\mathbf{w}'$, see Figure 34). We argue that if after assigning floor levels to each point in this subgrid the resulting graph $G_{\mathbf{w}'}$ has a single strongly connected component (SCC), then there exists at least one world in the full grid that satisfies the desiderata. Because the graph is strongly connected (has one SCC) this means that there is a path between every two points, and naturally there is no path leaving this world. However this world is at most 1/4th of the size of the $\mathbf{w}$, thus we embed it in our bigger world, by filling in the missing tiles. For every edge of $G_{\mathbf{w}'}$ that is bidirectional we put a corresponding ramp (purple in Figure 34), if the edge is one directional we fill it with a flat tile at the height of maximum of neighbouring heights (teal in Figure 34). We fill the remaining 1/4th of tiles with the highest floor (red in Figure 34). We treat $\mathbf{w}'$, together with tiles that we added in place of edges (as well as potentially some of the highest floors if they are accessible) as our region A. This A is at least of size 75% of $\mathbf{w}$. Every pair of points has a path between them, since by construction there is a path in $\mathbf{w}'$, and the tiles we added do not form any dead ends. In particular if any of the highest floors become accessible, the player can always jump down from it to a neighbouring tile of lower height. Consequently there are no paths leaving A.

To compute the exact number we take number of all possible $\mathbf{w}'$ which is $6^{\frac{wh}{4}}$ and then estimate the probability
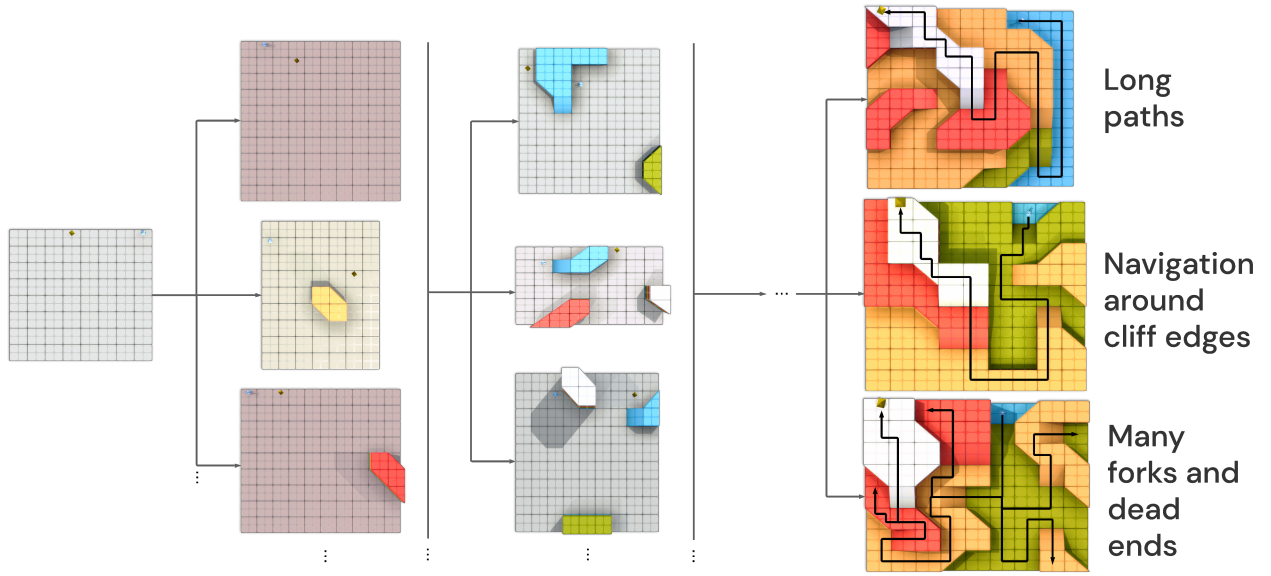
Figure 33 | The process of world-agent co-evolution resulting in complex worlds. **(Left)** The initial seed world to the train world set. The agent is trained to maximise the reward of its goal *"Be near a yellow pyramid"*. **(Middle)** The progression of the worlds in the train set as training progresses. 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.
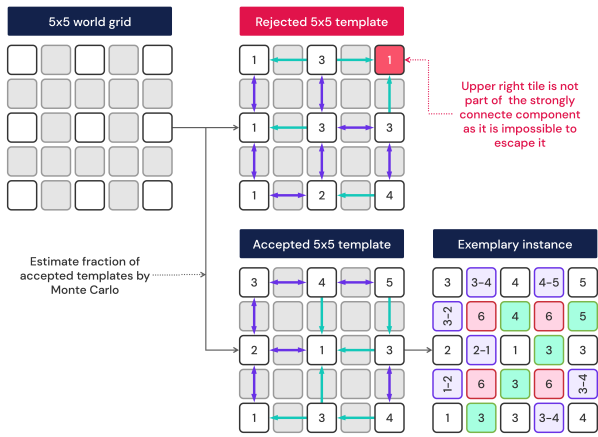

Figure 34 | Visualisation of the process used to estimate the number of correct worlds of a given size. Cyan lines represent one directional edges, while purple ones bidirectional. In the exemplary instance, filled tiles have colours corresponding to the edges they replaced, and in red we we see "pillars" used to fill in missing pieces.

n	$\lceil \frac{n}{2} \rceil$	Prob[#SCC($\mathbf{w}'$) = 1]
1	1	= 100%
2	1	= 100%
3	2	≈ 16%
4	2	≈ 16%
5	3	≈ 3%
6	3	≈ 3%
7	4	≈ 0.5%
8	4	≈ 0.5%
9	5	≈ 0.07%
10	5	≈ 0.07%
11	6	≈ 0.005%
12	6	≈ 0.005%
13	7	≈ 0.0006%
14	7	≈ 0.0006%

Table 1 | Monte Carlo estimations of the fraction of worlds with a single connected component as a function of the world size n (so the world is $n \times n$ tiles). We use 200,000 samples.

of such a world forming exactly one SCC by Monte Carlo sampling with 200,000 worlds (Table 1). For simplicity we also put $n = w = h$. Additionally, if w (h) is even, then there is a final column (row) at the edge of the world that is less constrained with respect to our desiderata: for each of $\frac{h}{2}$ ($\frac{w}{2}$) tiles we can fill in the neighbouring tiles by either the neighbour value, or a 1-off version. This leads to an additional factor of 2^{w+h}.

A.1.3| Worlds linear projection

In order to find a linear projection of the world space that visualises some property of interest $h.(\mathbf{w}) : \mathcal{W} \to \mathbb{R}_+$ we define a simple objective:

$$\ell_{\text{smooth}}(\theta) := \sum_{\mathbf{w}} \left\| \| \underbrace{\langle \widehat{\tau}(\mathbf{w}), \theta_{\mathbf{W}} \rangle - \theta_{\mathbf{b}}}_{\text{projection}} \|^2 - \underbrace{(h.(\mathbf{w}) - \theta_c)}_{\text{target}} \right\|^2.$$

and find a projection $\theta = (\theta_{\mathbf{W}}, \theta_{\mathbf{b}}, \theta_c)$ through gradient descent with learning rate 0.1 trained for 30,000 iterations. We denote by $\widehat{\tau}$ a vectorised version of the topology projection τ. For flat tiles, τ simply assigns the normalised floor level (e.g. for a tile t_k at level $k \in \{0, \ldots, 5\}$ we have $\tau(t_k) = \frac{1}{5}k$), and for simplicity ramps are assigned a half a floor value, i.e. for a ramp $t_{1 \leftrightarrow 2}$ between floor 1 and 2 $\tau(t_{1 \leftrightarrow 2}) = \frac{1}{5} \cdot 1.5$.
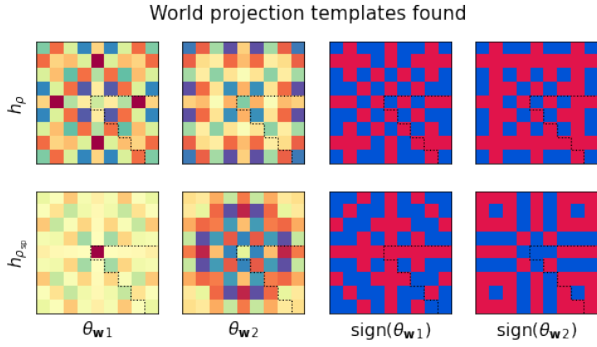
Figure 35 | Linear projections $\theta_{\mathbf{w}}$ found from the process of linearly embedding world topologies. The first two columns represent the 2 projection dimensions, and the following two columns simply show only the sign of each entry to emphasise the pattern. We can see a checkerboard like structure being discovered which naturally translates to navigational complexity/distribution of shortest paths. Dotted lines show the effective size of the learnable parameters, with remaining ones emerging from learning in the space invariant to symmetries.

Intuitively, we seek a linear projection $\theta_{\mathbf{W}} \in \mathbb{R}^{w \cdot h \times 2}$ in the space $\widehat{\tau}(\mathcal{W}) \subset \mathbb{R}^{w \cdot h}$ such that the distance from some point $\theta_{\mathbf{b}} \in \mathbb{R}^2$ in the embedding space corresponds to the distance between the target property and some arbitrary learned bias $\theta_c \in \mathbb{R}$. This can naturally lead to finding projections where the property of interest shows circular like placement (since we use Euclidean distance from a point as a predictor), but of course can also look more linear, if $\theta_{\mathbf{b}}$ is placed very far away from the projection of the data.

For our analysis we used $h_\rho(\mathbf{w}) := \mathrm{H}_2(\rho(\mathbf{w}))$ and $h_{\rho_{sp}}(\mathbf{w}) := \mathrm{H}_2(\rho_{sp}(\mathbf{w}))$ (note that entropy is always positive and thus satisfies the assumptions).

In order to avoid spurious results coming from a rich parameter space (81) compared to number of samples (around 250) we exploit the fact that each of $h.$ is invariant to rotations and symmetries, and learn a projection that is invariant to them too. To achieve this we simply parametrise $\theta_{\mathbf{w}}$ (which is itself a 9 by 9 world) to be an average of all 8 rotations/symmetries. Thanks to that, we reduce the number of actual trainable weights in the linear projection from 81 to 15, significantly reducing overfitting.

We can see templates found in Figure 35, where one can notice checkerboard like patterns being learned. To some extent this is a world-topology navigational complexity analogue to edge detectors from computer vision.

A.2 | Games

The following technical details of game space describe our current instance of XLand. Nothing in the system is constrained to using only the following relations, predicates, number of players, options etc. and can easily be expanded to cover an even richer space.

A.2.1 | Relations

near(a,b). Is true if object a is at most 1 meter away from object b (for reference a player's avatar is 1.65m tall).
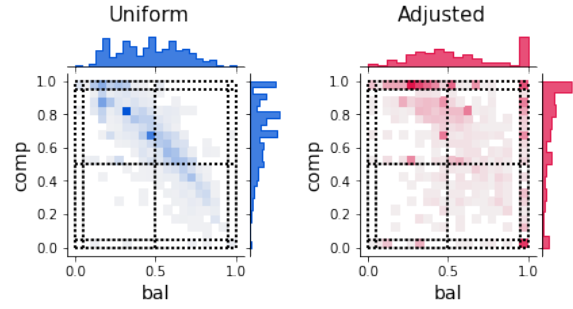

Figure 36 | The distributions of competitiveness and balance of games created by different sampling mechanisms. **(left)** We uniformly sample matrices and attach random relations for 1000 games with 1, 2 and 3 options, and see that extreme values are never sampled, and there is a huge correlation between balance and competitiveness, e.g. leading to no fully balanced and fully competitive games. **(right)** After using our local search procedure to seek values of b^*, c^* in various ranges (rectangular areas with dotted lines) we easily populate each entry including the extreme values.

In order to measure this we first calculate the center of mass of each object, and find the point on an object that is closest to the center of mass of the other. We find a midpoint between these two points on objects surfaces. We compute closest points on the surfaces of the objects to this midpoint, and measure the distance between them. This construction is relatively cheap to compute, and takes into consideration the size of the object (as in we do not merely measure distance between centers of mass, but actual body of the object).

see(a,b). If a is not a player, then it is evaluated by drawing a line connecting centers of masses of both objects and checking if it is unobstructed. If a is a player then it evaluates to true if and only if b would render in the field of view of a.

on(a,b). This relation can only be used on a floor and a player/object. It evaluates to true if and only if the given player/object a is in contact with the upper surface of the given floor colour b.

hold(a,b). This relation can only be used on a player and object. It evaluates to true if a player is holding a specific object with its beam. Note that it is technically possible for two agents to hold the same object at the same time.

A.2.2 | Atomic predicates

XLand currently consists of every possible instantiation of the above 4 relations between 9 objects (3 colours – purple, black and yellow – and 3 shapes – pyramid, cube and sphere), 2 player references (me, opponent) and 5 floor colours. This leads to 212 unique atomic predicates listed in Table 2 after taking into account logical symmetry, such as $\mathbf{g} := \mathtt{near(a,b)} \equiv \mathtt{near(b,a)} =: \mathbf{g}'$ in the sense that $r_{\mathbf{g}} = r_{\mathbf{g}'}$. Similarly see is symmetric if both arguments are objects (since objects have no orientation), but is not symmetric if at least one player is involved (since they have a

directional vision).

A.2.3 | Generating games

A naive way of generating a game could involve sampling a set of 6 atomic predicates, and then sampling a random matrix $\{-1, 0, 1\}^{3 \times 6}$ where rows are options, each made up of conjunctions of predicates, with at most 3 non zero entries per row. However, as seen in Figure 36 this creates an extremely non-uniform distribution with respect to balance/competitiveness. In fact it is almost impossible to randomly sample a fully competitive game.

Consequently we rely on a local search methodology of game generation (see Figure 37). In order to generate a game with a given number of options, conjunctions and target balance b^*, and competitiveness c^*:

- We sample a random game matrix with a specified number of options and conjunctions
- We sample 3 unique atomic predicates ϕ_1, ϕ_2, ϕ_3 , and then sample recolouring ξ and put $\phi_4 := \xi(\phi_1), \phi_5 := \xi(\phi_2), \phi_6 := \xi(\phi_3)$.
- If some predicates are identical then we keep resampling ξ. This always terminates as there always exists a recolouring that creates 3 new predicates.
- We combine matrix and predicates to get $\mathbf{G}$
- We compute $b := \text{bal}(\mathbf{G})$ $c := \text{comp}(\mathbf{G})$
- For a fixed number of iterations, or until $b = b^* \wedge c = c^*$ we:
    - Define improvement $\text{imp}(\mathbf{g}') := \min\{|b - b^*| - |\text{bal}(\mathbf{g}') - b^*|, |c - c^*| - |\text{comp}(\mathbf{g}') - c^*|\}$
    - We randomly select a goal $\mathbf{g}$ from $\mathbf{G} = (\mathbf{g}, \mathbf{g}')$
    - We perform a local modification of the goal by trying to change matrix representation of this goal:
        · Flip one of the 1s to -1s or vice verse
        · Add 1 or -1 in a random option (if it would not invalidate the limit of max 3 conjunctions)
        · Remove 1 or -1 in a random option (if it would not zero out the option)
        · Copy an option from $\mathbf{g}'$
        · Copy an option from $\mathbf{g}'$ and negate it (multiply by -1)
    - For each of such modifications $\mathbf{g}''$ we first verify that the game is not trivial, and that no 2 options are repeated, and then compute $\text{imp}(\mathbf{g}'')$.
    - If at least one modification led to non-negative improvement ($\mathbf{g}^*$), we pick the highest one and construct corresponding $\mathbf{G} = (\mathbf{g}', \mathbf{g}^*)$, recompute b and c and go to the next iteration.
    - If all improvement were negative we terminate.

The whole process is repeated 10 times and the best game selected (in terms of distance to b^*, c^*). Of course this process does not guarantee convergence to the exact value of b^* and c^*, in particular some are impossible – for example, there is no 1 option, 1 predicate game of competitiveness 0.25 and balance 0.75. Since generating a game with given properties is expensive, we also utilise the ability to create multiple games with the same characteristics described in the next section.

A.2.4 | Creating alike games

Given a game $\mathbf{G}$ it is very easy to create another game of exactly the same number of options, conjunctions, and same balance and competitiveness. We randomly choose one of the bijective recolourings ξ of objects, e.g.

$$\xi(\texttt{black sphere}) := \texttt{black sphere}$$
$$\xi(\texttt{purple sphere}) := \texttt{yellow sphere}$$
$$\xi(\texttt{yellow sphere}) := \texttt{purple sphere}$$
$$\xi(\texttt{black pyramid}) := \texttt{black pyramid}$$
$$\xi(\texttt{purple pyramid}) := \texttt{purple pyramid}$$
$$\xi(\texttt{yellow pyramid}) := \texttt{yellow pyramid}$$
$$\xi(\texttt{black cube}) := \texttt{yellow cube}$$
$$\xi(\texttt{purple cube}) := \texttt{purple cube}$$
$$\xi(\texttt{yellow cube}) := \texttt{black cube}$$

and return $\xi(\mathbf{G})$. It is easy to verify that $\text{comp}(\mathbf{G}) = \text{comp}(\xi(\mathbf{G}))$ (since competitiveness does not depend on semantincs of predicates) and also $\text{bal}(\mathbf{G}) = \text{bal}(\xi(\mathbf{G}))$, since because $\xi \in \Xi$ is a bijection, there also exists $\xi^{-1} \in \Xi$ thus

$$\text{bal}(\xi(\mathbf{G})) = \max_{\xi' \in \Xi} \text{coop}(\xi'(\xi(\mathbf{G}))) \geq \text{coop}(\xi^{-1}(\xi(\mathbf{G}))) = \text{bal}(\mathbf{G}).$$

And at the same time

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

consequently

$$\text{bal}(\mathbf{G}) = \text{bal}(\xi(\mathbf{G})).$$

Unfortunately, this process does not guarantee that $\xi(\mathbf{G}) \neq \mathbf{G}$ as the recolouring might not affect the game, or might just happen to recolour symmetric parts of the game. In practise, we repeat this process until a new game is found, and terminate it after 100 unsuccessful tries (*e.g.* note that the game of hide and seek will never produce any new games with this method as it does not contain any objects).

A.2.5 | Generation of a 3 player game

For simplicity, 3 player games are generated in the following way:

- We take a 2 player game $(\mathbf{g}_1, \mathbf{g}_2)$
- We create $\mathbf{g}_3$ by randomly mixing options from $\mathbf{g}_1$ and $\mathbf{g}_2$, and negate every predicate in a given option with 50% too.
- We randomly permute agents (and corresponding goals).

A.2.6 | PCA projection

In order to obtain a PCA over the game space, we note that

$$\|\mathbf{G}_i - \mathbf{G}_j\|_{\mathfrak{G}}^2 = \frac{1}{4n^2 \cdot N_\phi^2} \|\beth(\mathbf{G}_i) - \beth(\mathbf{G}_j)\|^2$$
$$= \frac{1}{4n^2 \cdot N_\phi^2} \left[\|\beth(\mathbf{G}_i)\|^2 + \|\beth(\mathbf{G}_j)\|^2 - 2\langle\beth(\mathbf{G}_i), \beth(\mathbf{G}_j)\rangle \right],$$

where $\beth(\mathbf{g})$ is a mapping that outputs a vector of length N_ϕ, where ith dimension equals 1 if ith valuation of predicates

```
hold(me,black cube)             hold(me,black pyramid)          hold(me,black sphere)           hold(me,purple cube)
hold(me,purple pyramid)         hold(me,purple sphere)          hold(me,yellow cube)            hold(me,yellow pyramid)
hold(me,yellow sphere)          hold(opponent,black cube)       hold(opponent,black pyramid)    hold(opponent,black sphere)
hold(opponent,purple cube)      hold(opponent,purple pyramid)   hold(opponent,purple sphere)    hold(opponent,yellow cube)
hold(opponent,yellow pyramid)   hold(opponent,yellow sphere)    near(black cube,black pyramid)  near(black cube,black sphere)
near(black cube,me)             near(black cube,opponent)       near(black cube,purple cube)    near(black cube,purple pyramid)
near(black cube,purple sphere)  near(black cube,yellow cube)    near(black cube,yellow pyramid) near(black cube,yellow sphere)
near(black pyramid,black sphere) near(black pyramid,me)         near(black pyramid,opponent)    near(black pyramid,purple cube)
near(black pyramid,purple pyramid) near(black pyramid,purple sphere) near(black pyramid,yellow cube) near(black pyramid,yellow pyramid)
near(black pyramid,yellow sphere) near(black sphere,me)         near(black sphere,opponent)     near(black sphere,purple cube)
near(black sphere,purple pyramid) near(black sphere,purple sphere) near(black sphere,yellow cube) near(black sphere,yellow pyramid)
near(black sphere,yellow sphere) near(me,purple cube)           near(me,purple pyramid)         near(me,purple sphere)
near(me,yellow cube)            near(me,yellow pyramid)         near(me,yellow sphere)          near(opponent,purple cube)
near(opponent,purple pyramid)   near(opponent,purple sphere)    near(opponent,yellow cube)      near(opponent,yellow pyramid)
near(opponent,yellow sphere)    near(purple cube,purple pyramid) near(purple cube,purple sphere) near(purple cube,yellow cube)
near(purple cube,yellow pyramid) near(purple cube,yellow sphere) near(purple pyramid,purple sphere) near(purple pyramid,yellow cube)
near(purple pyramid,yellow pyramid) near(purple pyramid,yellow sphere) near(purple sphere,yellow cube) near(purple sphere,yellow pyramid)
near(purple sphere,yellow sphere) near(yellow cube,yellow pyramid) near(yellow cube,yellow sphere) near(yellow pyramid,yellow sphere)
see(black cube,black pyramid)   see(black cube,black sphere)    see(black cube,me)              see(black cube,opponent)
see(black cube,purple cube)     see(black cube,purple pyramid)  see(black cube,purple sphere)   see(black cube,yellow cube)
see(black cube,yellow pyramid)  see(black cube,yellow sphere)   see(black pyramid,black sphere) see(black pyramid,me)
see(black pyramid,opponent)     see(black pyramid,purple cube)  see(black pyramid,purple pyramid) see(black pyramid,purple sphere)
see(black pyramid,yellow cube)  see(black pyramid,yellow pyramid) see(black pyramid,yellow sphere) see(black sphere,me)
see(black sphere,opponent)      see(black sphere,purple cube)   see(black sphere,purple pyramid) see(black sphere,purple sphere)
see(black sphere,yellow cube)   see(black sphere,yellow pyramid) see(black sphere,yellow sphere) see(me,black cube)
see(me,black pyramid)           see(me,black sphere)            see(me,opponent)                see(me,purple cube)
see(me,purple pyramid)          see(me,purple sphere)           see(me,yellow cube)             see(me,yellow pyramid)
see(me,yellow sphere)           see(opponent,black cube)        see(opponent,black pyramid)     see(opponent,black sphere)
see(opponent,me)                see(opponent,purple cube)       see(opponent,purple pyramid)    see(opponent,purple sphere)
see(opponent,yellow cube)       see(opponent,yellow pyramid)    see(opponent,yellow sphere)     see(purple cube,me)
see(purple cube,opponent)       see(purple cube,purple pyramid) see(purple cube,purple sphere)  see(purple cube,yellow cube)
see(purple cube,yellow pyramid) see(purple cube,yellow sphere)  see(purple pyramid,me)          see(purple pyramid,opponent)
see(purple pyramid,purple sphere) see(purple pyramid,yellow cube) see(purple pyramid,yellow pyramid) see(purple pyramid,yellow sphere)
see(purple sphere,me)           see(purple sphere,opponent)     see(purple sphere,yellow cube)  see(purple sphere,yellow pyramid)
see(purple sphere,yellow sphere) see(yellow cube,me)            see(yellow cube,opponent)       see(yellow cube,yellow pyramid)
see(yellow cube,yellow sphere)  see(yellow pyramid,me)          see(yellow pyramid,opponent)    see(yellow pyramid,yellow sphere)
see(yellow sphere,me)           see(yellow sphere,opponent)     on(black cube,brown floor)      on(black cube,olive floor)
on(black cube,orange floor)     on(black cube,blue floor)       on(black cube,grey floor)       on(black cube,white floor)
on(black pyramid,brown floor)   on(black pyramid,olive floor)   on(black pyramid,orange floor)  on(black pyramid,blue floor)
on(black pyramid,grey floor)    on(black pyramid,white floor)   on(black sphere,brown floor)    on(black sphere,olive floor)
on(black sphere,orange floor)   on(black sphere,blue floor)     on(black sphere,grey floor)     on(black sphere,white floor)
on(me,brown floor)              on(me,olive floor)              on(me,orange floor)             on(me,blue floor)
on(me,grey floor)               on(me,white floor)             on(brown floor,opponent)        on(brown floor,purple cube)
on(brown floor,purple pyramid)  on(brown floor,purple sphere)   on(brown floor,yellow cube)     on(brown floor,yellow pyramid)
on(brown floor,yellow sphere)   on(olive floor,opponent)        on(olive floor,purple cube)     on(olive floor,purple pyramid)
on(olive floor,purple sphere)   on(olive floor,yellow cube)     on(olive floor,yellow pyramid)  on(olive floor,yellow sphere)
on(opponent,orange floor)       on(opponent,blue floor)         on(opponent,grey floor)         on(opponent,white floor)
on(orange floor,purple cube)    on(orange floor,purple pyramid) on(orange floor,purple sphere)  on(orange floor,yellow cube)
on(orange floor,yellow pyramid) on(orange floor,yellow sphere)  on(purple cube,blue floor)      on(purple cube,grey floor)
on(purple cube,white floor)     on(purple pyramid,blue floor)   on(purple pyramid,grey floor)   on(purple pyramid,white floor)
on(purple sphere,blue floor)    on(purple sphere,grey floor)    on(purple sphere,white floor)   on(blue floor,yellow cube)
on(blue floor,yellow pyramid)   on(blue floor,yellow sphere)    on(grey floor,yellow cube)      on(grey floor,yellow pyramid)
on(grey floor,yellow sphere)    on(white floor,yellow cube)     on(white floor,yellow pyramid)  on(white floor,yellow sphere)
```

Table 2 | List of all atomic predicates used in the current iteration of XLand. It consists of 212 elements that relate 2 players, 9 movable objects and 5 floors through use of 4 relations. The slab object is never used in atomic predicates.
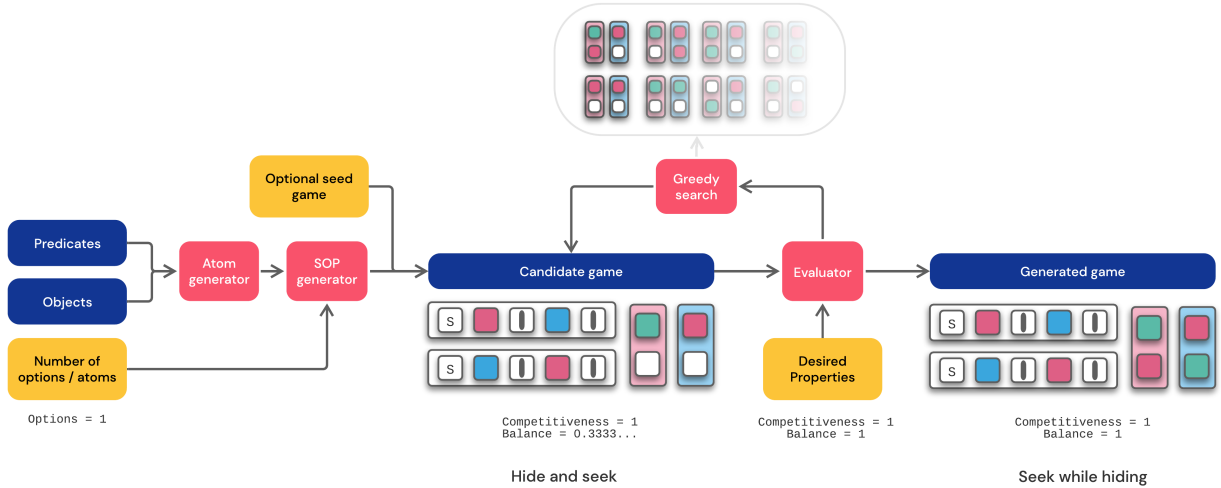
Figure 37 | The process of generating of a game with target properties. We see the matrix representation, as well as relations. Greedy local search performs simple modifications to the game matrix, as described in Section A.2.

is rewarding under **g**, and -1 otherwise, and $\beth(G) : \mathfrak{G} \to \{-1, 1\}^{n \cdot N_\phi}$ is simply concatenation of $\beth(g)$ over goals of the game. With this construction, the norm of each vector is constant $\|\beth(\cdot)\|^2 = n \cdot N_\phi$ and thus we can compute a $\beth$-kernel (Scholkopf, 2001):

$$K_\beth(G_i, G_j) \propto -\|G_i - G_j\|_\mathfrak{G}^2$$

This way we can utilise any kernelised method (such as Kernel PCA (Schölkopf et al., 1998)) to analyse the Game Space without explicitly forming exponentially large $\beth(\mathfrak{G})$ space (which in our case has over $6 \cdot 10^{65}$ dimensions) and see linear structures in the space in which $\| \cdot \|_\mathfrak{G}$ is just the Euclidean distance.

A.3| Holding out tasks from training

In order to create hold-out games we simply create a single set of unique games (in the sense that no two games $G \equiv G'$ even under any recolouring ξ), split them between test and validation. Training games generated are online rejected if there is any collision (including under recolouring) with test or validation.

For hold-out worlds similarly a set of unique worlds was created (Section 4.2) and split between test and validation. Training worlds generated are online rejected if there is any collision with test or validation.

Finally, validation and test task sets are composed by random matching of corresponding world and games. This means that there is no game, nor world (and consequently no task) shared between the two sets or encountered during dynamic training task generation.

A.4| Reinforcement Learning

We use V-MPO (Song et al., 2020) with hyperparameters provided in Table 3. We use a batch size of 64 and unroll length 80.

Name	Value	Evolved with PBT
discount (γ)	0.99	no
learning rate	1e-4	yes
baseline cost	1	no
target update period	50	no
ϵ_α	0.01252	yes
ϵ_{temp}	0.1	no
init α	5	no
init temp	1	no
top k fraction	0.5	no
$m_>$	7.5	yes
m_s	375	yes
m_{cont}	0	yes
$m_{>\text{cont}}$	0	yes
m_{solved}	1	yes

Table 3 | V-MPO and DTG hyperparameters. The hyperparameters that have "yes" in the last column are adjusted using PBT (see Section A.7).

A.5| Distillation

We follow on-policy distill (Czarnecki et al., 2020), which for a teacher π^{teacher} and student π defines the per timestep t auxiliary loss

$$\ell_t^{\text{distill}} := \text{KL} \left[\left\| \pi_t^{\text{teacher}} \right\| \middle\| \pi_t \right].$$

In order to further exploit the fact that our goal is not to replicate the teacher but rather use to bootstrap from and provide assistance with exploration, we mask this loss over states when the reward is obtained. In other words, we distill on a timestep if and only if an agent is coming from a non-rewarding state, thus defining *exploration distillation* loss

$$\ell_t^{\text{exp-distill}} := (1 - r_{t-1}) \cdot \ell_t^{\text{distill}}.$$

Name	Type	Description
ACCELERATION	$\mathbb{R}^3$	Acceleration in each of 3 axes.
RGB	$[0, 255]^{72 \times 96 \times 3}$	Pixel observations in RGB space with 72 by 96 pixel resolution.
HAND IS HOLDING	$\{0, 1\}$	Flag whether an item is held.
HAND DISTANCE	$\mathbb{R}$	A distance to a held object if one is held.
HAND FORCE	$\mathbb{R}$	A force applied to an item held by an agent.
LAST ACTION	$(\mathbf{a}_{t-1,i})_{i=1}^{10}$	A 10-hot encoding of the previously executed action.
GOAL MATRIX	$\{-1, 0, 1\}^{6 \times 6}$	a matrix encoding a goal in DNF form.
GOAL ATOMS	$\mathbb{N}^{6 \times 6}$	a matrix encoding corresponding atoms from the goal, provided as categoricals.

Table 4 | List of all observations an agent receives as part of $\mathbf{o}_t^i := (f_i(\mathbf{s}_t), \mathbf{g}_i)$. Note, that this does not contain a reward, as the agent policy π does not have access to this information, only the value head does during training.

Name	Possible values	Description
MOVE FORWARD BACK	$\{-1, 0, 1\}$	Whether to apply forward/backward or no movement.
MOVE LEFT RIGHT	$\{-1, -0.05, 0, 0.05, 1\}$	Strafing.
LOOK LEFT RIGHT	$\{-1, -0.2, -0.05, 0, 0.05, 0.2, 1\}$	Left/right rotation.
LOOK UP DOWN	$\{-1, -0.03, 0, 0.03, 1\}$	Up/down rotation.
GRAB	$\{0, 1\}$	Grab an object.
USE GADGET	$\{0, 1\}$	Use currently equipped gadget.

Table 5 | The structure of the decomposed action space, consisting of 6 dimensions of discrete actions. Every combination of the values is feasible, leading to 2100 possible actions.

Note we have binary 0 or 1 rewards r_{t-1} only in this work. An agent uses a weight of 4 of this loss for the first 4 billion steps in every generation apart from the first generation (since there is no initial teacher), and the weight changes to 0 afterwards.

A.6| Network architecture

Unless otherwise specified, every MLP uses the ReLU activation function. We use Sonnet (DeepMind, 2020) implementations of all neural network modules, and consequently follow Sonnet default initialisation schemes as well.

Torso. An RGB observation (see Table 4 for details) is passed to a ResNet (He et al., 2016) torso with $[16, 32, 32]$ channels, each consisting of 2 blocks and output size of 256. Max pooling is used, as well as scalar residual multiplier. Torso produces $\hat{\mathbf{o}}_t$.

Goal embedding. The goal embedding network is presented in Figure 38. The predicate embedding network uses a linear layer with 256 outputs. The atom-mlp is a 2-hidden layer MLP with 256 hidden units. The option-mlp is a 2-hidden layer MLP with 256 hidden units. The final goal embedding is 256 dimensional. We use summation to aggregate across conjunctions in the option, as well as options in the goal. This processing is akin to using a Graph Neural Network (Battaglia et al., 2018), where there is one vertex for a goal, connected to one vertex per option, which has edges to predicate vertices, labeled with either 1 or -1 (to denote negation), and each predicate vertex is labeled with its value.

LSTM. We use a 2 layer LSTM core with 256 neurons each, and skip connections. The LSTM takes as input $\hat{\mathbf{o}}_t$ and produces $\mathbf{h}_t$.

Atom predictor. The atom predictor uses inputs $\mathbf{h}_t$ and atom embedding $\mathbf{e}_{\mathbf{a}_t}$, concatenated and tiled to form a tensor of size [num_atoms, atom_features + hidden_size]. We apply a small MLP with 2 hidden layers of sizes 64 and 32 and one output neuron per atom predicted, forming $\mathbf{p}_t$ of size num_atoms.

GOAT module. The GOAT module takes $\mathbf{h}_t$, $\mathbf{p}_t$ and $\mathbf{g}$ as inputs. In order to apply attention to a flat output of an LSTM, we first linearly upscale it to 1024 dimensions, and then reshape to a 32 by 32 matrix (and analogously the goal embedding is reshaped to have 32 features) and then it is passed to an attention module with a single head, and key and value sizes set to 128. We use a temperature of 0.1 and softmax mixing in the GOAT module. The internal value heads of GOAT use an MLP with 256 hidden units. Note, that these do not accept last reward as input (as opposed to the external value head), since these value heads are used during inference and affect the policy output – our agent's policy is not conditioned on reward. The GOAT module produces $\widehat{\mathbf{h}}_t$ as an output.

PopArt. The PopArt value head (Hessel et al., 2019; van Hasselt et al., 2016) uses 256 hidden units.

Policy head. The policy head is a simple MLP, applied to $\widehat{\mathbf{h}}_t$, with 256 hidden units, and 6 linear, softmaxed heads $(\pi_t)_k$, one per action group (see Table 5 for details). We
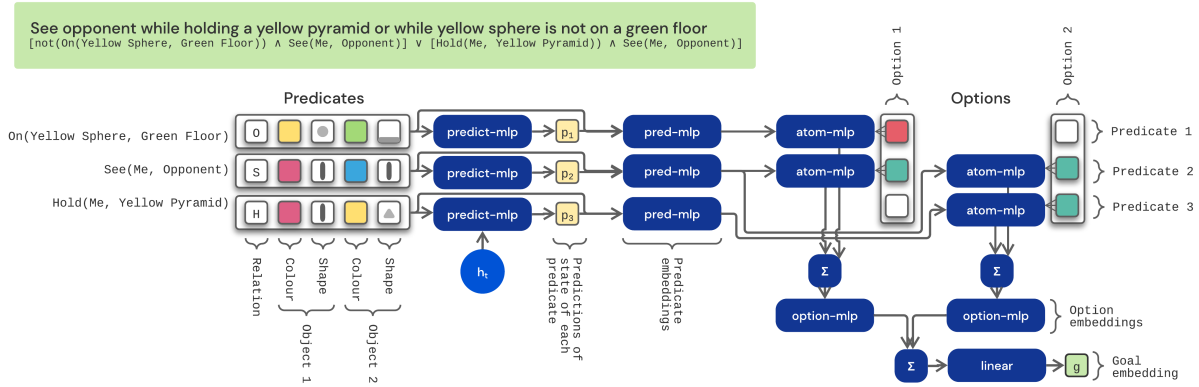
Figure 38 | The architecture of the goal embedding/prediction modules. Atomic predicates are provided in a 5-hot encoded fashion, since all the relations used take two arguments, each of which can be decomposed to a colour and shape. For player object we simply have a special colour "me" and "opponent". Details of the architecture are provided in Section A.6.

follow a decomposed action space, meaning that we assume the value of each action group is conditionally independent given the current state:

$$\pi_t[\mathbf{a}_t] := \Pr[\mathbf{a}_t = (\mathbf{a}_t^1, \ldots, \mathbf{a}_t^6)] := \prod_{k=1}^{6} (\pi_t)_k[\mathbf{a}_t^k],$$

which allows us to efficiently parameterise the joint distribution over the entire space of 2100 actions (Jaderberg et al., 2019).

External Value Head. The value head used for RL algorithms, takes, in addition to $\widehat{\mathbf{h}}_t$ the last reward r_{t-1} (concatenated), and produces the value prediction by an MLP with 256 units.

A.6.1 | Auxiliary losses

All auxiliary losses are computed by treating a trajectory as if it was a sequence-to-sequence problem, without taking into consideration the effects it has on the control policy (Czarnecki et al., 2019; Jaderberg et al., 2017b).

We do not weight nor tune auxiliary losses weights, they simply added up together, jointly with the updates coming from the RL system. Merging is done on the update level, meaning that RL updates do not take into consideration effect they have on auxiliary losses either.

GOAT. There are two consistency losses $\ell^{\mathbf{V}}, \ell^{\mathbf{h}}$ coming from the GOAT architecture.

Atom prediction. We use standard cross-entropy loss for multi-label classification.

External value function matching. Internal value functions of GOAT have an additional alignment loss in the form of the L2 distance between the final GOAT value and the external value function. We do not stop gradients on external value function, meaning that they co-distill.

A.7 | Population Based Training

After the initial period of 5e8 steps of guaranteed no evolution events, every 1e8 steps we check whether some agents should be evolved. A pair of agents π_i and π_j are considered eligible for evolution from π_j (parent) to π_i (child) if and only if:

- π_i did not undergo evolution in last 2.5e8 steps,
- $\forall_{k \in \{10,20,50\}} \text{perc}(\pi_j | \mathbf{\Pi}_t)[k] \geq \text{perc}(\pi_i | \mathbf{\Pi}_t)[k]$,
- $\text{score}_{ij} := \frac{\sum_{k \in \{10,20,50\}} \text{perc}(\pi_j | \mathbf{\Pi}_t)[k]}{\sum_{k \in \{10,20,50\}} \text{perc}(\pi_i | \mathbf{\Pi}_t)[k]} > 1.01$.

Next, we take a pair where score_{ij} is maximum and perform the evolution operation. This means that weights and hyperparameters of π_i are copied from π_j, and for each of the hyperparameters, independently with 50% chance we apply following mutation:

- V-MPO hyperparameter ϵ_α is multiplied or divided by 1.2 (with 50% chance).
- learning rate is multiplied or divided by 1.2 (again with 50% chance)
- m_{cont} is increased or decreased by 7.5 (again with 50% chance), and clipped to $(0, 900)$ (range of the return).
- m_{cont} is multiplied or divided by 1.1 (again with 50% chance), and clipped to $(0, 900)$ (range of the return).
- $m_>$ is multiplied or divided by 1.2 (again with 50% chance), and clipped to $(0, 900)$ (range of the return).
- m_{solved} and $m_{>\text{cont}}$ is increased or decreased by 0.1 (again with 50% chance), and clipped to $(0, 1)$ (corresponding to possible values of our MC estimator with 10 samples).

For efficiency reasons, the child agent also inherits the parent's DTG training task set.

A.8 | GOAT

Placement of the GOAT module. The GOAT module is placed after the recurrent core, and lacks recurrence itself. The reason is to be able to query "*what would happen if the*

policy's goal was different" without having to unroll entire trajectory with a separate goal. In principle one could have $n_o + 1$ copies of the agent unrolled over the experience, one conditioned on the full goal, and the remaining n_o on corresponding options. The full goal conditioned unroll would generate the actions and experience, whilst the remaining unrolls would be off-policy. Placing the GOAT module post recurrence avoids this redundancy, and actually forces the agent to explicitly split it's reasoning into goal conditioned and goal invariant (recurrent) parts. Arguably this is also responsible for creating an agent that is capable of easily reacting to a new goal/change of goals mid-episode, despite not being trained to do so.

Feedback loop that affects the policy. Let us imagine that $\mathbf{v}_t^{[1]} > \mathbf{v}_t^{[0]}$, meaning that option 1 has a higher value than the whole game. If this is correct, meaning that $\mathbf{v}_t \approx \mathbf{V}_\pi(\mathbf{s}_t)$ this means that an agent could improve its policy if it was to act as if its goal was changed purely to option 1. Consequently the loss $\ell^{\mathbf{h}}$, which aligns the internal state with the internal state corresponding to following just option 1, can be seen as a step of policy improvement. We are working under an assumption that f_π is a smooth L-Lipschitz function, meaning that for some (hopefully small $L > 0$) $\|f_\pi(\widehat{\mathbf{h}}) - f_\pi(\widehat{\mathbf{h}'})\| \le L\|\widehat{\mathbf{h}} - \widehat{\mathbf{h}'}|$. One could also add an explicit loss aligning policies themselves, however this would require balancing a loss over distributions over actions (e.g. KL) with the loss over hidden activations (L2). For simplicity we are not doing this in this work.

A.9| Multi-agent analysis

For multi-agent analysis we took the final generation of the agent (generation 5) and created equally spaced checkpoints (copies of the neural network parameters) every 10 billion steps, creating a collection of 13 checkpoints.

A.9.1| Hide and seek

We the following definition of hide and seek

$$\mathbf{g}_1 := \text{see(me,opponent)}$$
$$\mathbf{g}_2 := \text{not(see(opponent,me))}$$

We collected 1000 test worlds and ran 1 episode per matchup (agent pair) per world.

A.9.2| Conflict Avoidance

The game is defined as

```
g₁ :=  on(purple sphere, orange floor)∨
       on(yellow sphere, orange floor)
g₂ :=  on(yellow sphere, grey floor)
```

Experiments were conducted on a single world, where agents spawn equally distant to both spheres, and thus their choice of which one to use is not based on distance heuristics they might have learned. They can also see all the target floors initially thus there is no exploration needed. We collected 1000 episodes per matchup (agent pair).

Chicken	Cooperate	Defect
Cooperate	1,1	**1,2**
Defect	**2,1**	0,0

$\mathbf{g}_1$		$\mathbf{g}_2$
$\phi_1 \wedge \neg\phi_5 \wedge \phi_4$		$\phi_1 \wedge \phi_5 \wedge \neg\phi_4$
$\phi_2 \wedge \neg\phi_5 \wedge \phi_4$		$\phi_2 \wedge \phi_5 \wedge \neg\phi_4$
$\phi_3 \wedge \phi_5 \wedge \neg\phi_6$		$\phi_3 \wedge \phi_4 \wedge \neg\phi_6$

Table 6 | **(Top)** The Chicken social dilemma with Nash Equilibria highlighted in bold font. **(Bottom)** The corresponding option based encoding using task predicates. ϕ_1, ϕ_2, ϕ_3 are unique predicates and ϕ_4, ϕ_5, ϕ_6 represent conjunctions with non overlapping predicates.

A.9.3| Encoding Chicken in Xland

We encode social dilemmas in Xland using task predicates such that the game options can be grouped into *meta-actions* and the social dilemma rewards correspond to option *multiplicity* that the player has given the opponent meta-action. Table 6 represents the encoding for the social dilemma game of *Chicken*. Here, the payoff for the meta-actions *Cooperate* and *Defect* are given. The choice of the predicates makes it impossible for both the players to satisfy options corresponding to the Defect meta-action simultaneously (hence the meta-action joint reward is $(0, 0)$ for the Defect-Defect joint meta-action), whereas the options corresponding to Cooperate are compatible with any option. In our experiments we used options of same the predicate length and type so that they are similar in complexity to satisfy. The exact game is given as follows:

```
ĝc :=see(opponent,yellow pyramid)∧
      see(yellow cube,yellow sphere)∧
      not(see(black pyramid,purple pyramid))
ĝd₁ :=see(me,yellow pyramid)∧
      see(black cube,black sphere)∧
      not(see(opponent,yellow pyramid))
ĝd₂ :=see(me,yellow pyramid)∧
      see(purple cube,purple sphere)∧
      not(see(opponent,yellow pyramid))
g₁ :=ĝc ∨ ĝd₁ ∨ ĝd₂
g₂ :=ĝc ∨ ĝd₁ ∨ ĝd₂
```

We collected 1000 test worlds and run 1 episode per matchup (agent pair) per world.

A.10| Handauthored levels

We created a fixed set of hand-authored tasks as an additional static evaluation set. These are described in Table 7 and Table 8.

A.11| Representation analysis

To conduct the representation analysis we gathered 3000 trajectories coming from test tasks. Next, we randomly

Name	Description	Agent return > 0
Capture The Cube	A competitive game where both players must bring the opponent's cube to their cube and base floor in a symmetrical world.	✓
Catch Em All	A cooperative game where both players must make 5 particular objects near each other, gathering them from across the world.	✓
Choose Wisely 2p	Each player has the same 3 options to choose from: one of 3 cubes to hold without the other player holding that cube.	✓
Choose Wisely 3p	The same as above with 3 players.	✓
Coop or Not	Both players have the one cooperative option and one competitive option to choose from.	✓
Find Cube	The player must find the cube to be near in a complex world.	✓
Find Cube With Teaser	The same as above, however the world allows the agent to see the cube but the world does not allow direct navigation to the cube.	✗
Hide And Seek: Hider	Asymmetric hide and seek in a simple world with the agent playing the hider.	✓
Hide And Seek: Seeker	Same as above but the agent playing the seeker.	✓
Hold Up High	Both players must hold the yellow pyramid on the highest floor.	✓
King of the Simplest Hill	Both players must be on the top floor of the world without the other player being on that floor.	✓
King of The Hill	Same as above but with a more complex world.	✓
Keep Tagging	The player must stop the other player holding objects or touching a floor.	✓
Make Follow Easy	The agent must lead the other player (whose policy is to follow) to a particular floor colour.	✗
Make Follow Hard	Same as above however the target floor is higher and has a smaller area.	✗
Mount Doom	In a world with a large central mountain, the agent must get to the top without the other player getting to the top.	✓
Mount Doom 2	Same as above but the other player starts at the top.	✓
Navigation With Teaser	Similar to Find Cube With Teaser, however the agent must navigate to a target floor rather than the cube.	✓
Nowhere To Hide	The players start on opposite towers, and the agent must stop the other player (noop policy) from touching the tower floor.	✓
Object Permanence Black Cube	The agent starts and can see a yellow cube on the left and a black cube on the right. The agent must choose which path to take (which means the agent loses sight of the cubes) to reach the target cube (black in this case).	✓
Object Permanence Yellow Cube	Same as above but the target cube is yellow.	✓
One Pyramid Capture The Pyramid	Same world as Capture the Cube, however both players must take the single yellow pyramid to their base floor.	✓
Race To Clifftop With Orb	Both players must stand on the top of a cliff edge holding the yellow sphere, without the other player standing on the cliff.	✓

Table 7 | List of hand authored tasks. The last column shows if the agent participates in the specific task (whether it ever reaches a rewarding state). Continues in Table 8.

Name	Description	Agent return > 0
Ridge Fencing	Similar to King of the Hill, however the target floor is a narrow ridge.	✓
Sheep Herder	The agent must make the other player stand near a set of target objects.	✗
Solve AGI	The agent must take the black sphere and put it on the target floor against another player that is trying to oppose this.	✓
Stay Where You Spawn	The player doesn't have to do anything, it is rewarded if it stays in its initial position.	✓
Stop Rolling Freeze Gadget	In world composed of one big slope, the agent must stop the purple sphere from touching the bottom floor without holding it.	✓
Stop Rolling Tag Gadget	Same as above, except the agent has a tag gadget rather than freeze gadget.	✓
Stop Rolling Tag Gadget Easy	Same as above, except the slope is less steep.	✓
Tag Fiesta 2p	Both players have a goal to make the other player not touch any floors. All players have the tag gadget.	✓
Tag Fiesta 3p	Same as above but a 3-player version.	✓
Tool Use Climb 1	The agent must use the objects to reach a higher floor with the target object.	✗
Tool Use Gap 1	The agent must reach an object but there is a gap in the floor.	✓
Tool Use Gap 2	Same as above but the gap is bigger.	✗
Who Gets The Block	Both players want to be holding the same cube but on different coloured floors.	✓
XFootball	Both players want the black sphere to be touching the floor at opposite ends of the world.	✓
XRPS Counter Black	A version of XRPS Section 3.2.3 in which the agent can choose from three rock-paper-scissor like options. The other player always chooses to hold the black sphere.	✓
XRPS Counter Purple	Same as above but where the other player always chooses to hold the purple sphere.	✓
XRPS Counter Yellow	Same as above but where the other player always chooses to hold the yellow sphere.	✓
XRPS With Tag Gadget	Full rock-paper-scissor like game.	✓
XRPS 2 With Tag Gadget	Same as above but with different predicates representing the options.	✓

Table 8 | List of all hand authored tasks continued. The last column shows if the agent participates in the specific task (whether it ever reaches a rewarding state).
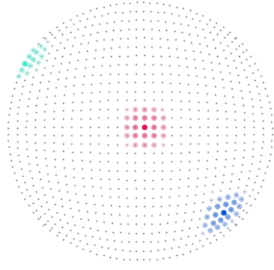
Figure 39 | Visualisation of the Kohonen Network used in our analysis, composed of 900 Kohonen Neurons. Three neurons are called out in colour, with their receptive field (neurons that have non zero update weight) colour-coded with the colour intensity representing the weight.

When plotting a Kohonen map for a specific hidden state we subtract the average activity in our dataset and use colour using a diverging colour map. We do this for visual clarity to avoid displaying a pattern (bias) that is constantly there.

Kohonen Neurons   Once the Kohonen Neuron $\mathfrak{h}$ for a property p has been identified, we define the following classifier:

$$\hat{p}(x) := \text{KDE}[\mathfrak{h}_i : p(\mathbf{s}_i)](x) > \text{KDE}[\mathfrak{h}_i : \neg p(\mathbf{s}_i)](x)$$

where $\text{KDE}[A](x)$ is the kernel density estimation density of a set A evaluated at x. We use a Gaussian kernel and Silverman's rule to estimate its bandwidth (Silverman, 2018).

selected 60 timestamps $\hat{t}_i \in (1, 900)$ and used them to subsample the trajectories, resulting in 180,000 states $\mathbf{s}_i$ with corresponding recorded activations of $\widehat{h}_i$ (GOAT module), $\mathbf{h}_i$ (LSTM state) and the goal embedding.

We train a Kohonen Network composed of 900 neurons, arranged in a 30×30 grid covering a unit circle through a transformation of the lattice over $[-1, 1]^2$

$$\mathfrak{k}(x, y) := \left(x \cdot \sqrt{1 - \tfrac{1}{2}y^2}, y \cdot \sqrt{1 - \tfrac{1}{2}x^2} \right),$$

for 5000 iterations, using stochastic gradient descent with batches of size 100 and an exponentially decaying learning rate $\text{lr}_k := 0.1 \cdot \exp(1 - \tfrac{k}{5000})$. The initial neurons positions are given by the minibatch k-means clustering (Lloyd, 2006) using `k-means++` heuristic initialisation (Arthur and Vassilvitskii, 2007), batch size of 100 and 100 iterations.

We use $\text{d}_{\max} = 3$, with the visualisation of the emerging Kohonen Network and local receptive fields shown in Figure 39.

We use the following definitions of properties of state $\mathbf{s}_t$ when playing with a goal $\mathbf{g}$:

- **Early in the episode**. True if and only if $t < \hat{t}_{30}$ (approximately half of the episode).
- **Agent is holding**. True if and only if $\mathbf{s}_t$ represents a situation in which the agent is holding an object.
- **High baseline**. True if and only if $\mathbf{v}_t > 7.5$ (after Popart normalisation).
- **Rewarding state**. True if and only if $r_{\mathbf{g}}(\mathbf{s}_t) = 1$.
- **Knows it's rewarded**. True if and only if $r_{\mathbf{g}}(\mathbf{s}_t) = 1$ and the active option has all its atoms predicted correctly.
- **Knows entire state**. True if and only if every atom prediction is correct.
- **One missing atom**. True if and only if $r_{\mathbf{g}}(\mathbf{s}_t) = 0$ and there exists $\mathbf{s}'$ such that $\|\phi(\mathbf{s}_t) - \phi(\mathbf{s}')\| = 1$ and $r_{\mathbf{g}}(\mathbf{s}') = 1$.
- **Many missing atoms**. True if and only if $r_{\mathbf{g}}(\mathbf{s}_t) = 0$ and for every $\mathbf{s}'$ such that $r_{\mathbf{g}}(\mathbf{s}') = 1$ we have $\|\phi(\mathbf{s}_t) - \phi(\mathbf{s}')\| > 1$.
- **One option**. Whether $\mathbf{g}$ consists of a single option.
- **Many options**. Whether $\mathbf{g}$ consists of more than one option.

B| Proofs for Section 3 (Environment Properties)

Proposition 3.1. *For every goal $\mathbf{g}$ where $\kappa(\mathbf{g}) = 0$ or $\kappa(\mathbf{g}) = 1$ every policy is optimal.*

Proof. Lets assume that $\kappa(\mathbf{g}_1) = z \in \{0, 1\}$ this means that for every state $\mathbf{s}$ we have $r_{\mathbf{g}}(\mathbf{s}) = z$. Consequently, for every policy π we have $\mathbf{V}_\pi(\mathbf{x}) = \mathbf{V}_\pi((\mathbf{w}, (\mathbf{g}_1, \ldots, \mathbf{g}_n), (\pi_1, \ldots, \pi_n))) = Tz$, and so in particular $\forall_\pi \mathbf{V}_\pi(\mathbf{x}) = Tz = \max_\pi \mathbf{V}_\pi(\mathbf{x}) = \mathbf{V}^*(\mathbf{x})$. □

Theorem 3.1. *Under the assumption that each atomic predicate that does not involve negation is independently solvable, the number of unique n-player games $N_{\mathfrak{G}}$ with respect to the reward functions they define satisfies:*

$$\frac{1}{n!}\left[\frac{1}{n_o!}\prod_{i=1}^{n_o}\left(\binom{n_a/2 - i \cdot n_c}{n_c}2^{n_c}\right)\right]^n \leq N_{\mathfrak{G}} \leq \binom{n_a}{n_c}^{n \cdot n_o}.$$

Proof. The high level idea for the lower bound is to count the number of games where inside each goal every predicate is unique (and can only repeat across players).

First, let us prove the statement for two goals, $\mathbf{g}_i$ and $\mathbf{g}_j$, where each has a corresponding set of predicates used $(\phi_{i o_k c_l})_{k,l=1}^{n_o,n_k}$ and $(\phi_{j o_k c_l})_{k,l=1}^{n_o,n_c}$, each being lexicographically sorted over options (indexed by o_k, and over predicates inside each option (indexed by c_l), so that the option and alternatives orderings are irrelevant.

If the two goals are different, this means that there exists k^*, l^* such that $\phi_{i k^* l^*} \neq \phi_{j k^* l^*}$. Let's take the smallest such k^* and a corresponding smallest l^*. This means that there exists an option in one the goals, that the other goal does not possess. Without loss of generality, let us assume it is an option of $\mathbf{g}_i$, meaning that $\neg \exists_{k'} \phi_{j k'} = \phi_{i k^*}$. Since this option uses unique predicates across the goal, let us define s^* as a simulation state such that all the predicates of this option are true, while all the other predicates are false. Then we have

$$r_{\mathbf{g}_i}(s^*) = 1 \neq 0 = r_{\mathbf{g}_j}(s^*)$$

proving that $r_{\mathbf{g}_i} \neq r_{\mathbf{g}_j}$ and thus $\mathbf{g}_i \not\equiv \mathbf{g}_j$.

The only thing left is to count such goals. For that, let us note that this is an iterative process, where for each ith of n_o options we can pick n_c out of $n_a/2 - i \cdot n_c$ predicates to be used (since we already picked $i \cdot n_c$ before, and we are not picking negations). Once we picked the predicates, each of them can be either itself or its negations, which introduces the 2^{n_c} factor. And since the process is order variant, we need to simply divide by the number of permutations of length n_o, leading to

$$\frac{1}{n_o!}\prod_{i=1}^{n_o}\left(\binom{n_a/2 - i c n_c}{n_c}2^{n_c}\right)$$

and completing the lower bound proof.

The upper bound comes from simply noting that every reward function that comes from a Boolean expression with n_o alternatives, each being a conjunction of n_c out of n_a

predicates has a corresponding Boolean expression of this form, and thus we can just count how many such expressions are there:

$$\binom{n_a}{n_c}^{n \cdot n_o},$$

completing the upper bound proof. □

Proposition 3.2. *Exploration difficulty is a 1-Lipschitz function, meaning that for any $\mathbf{G}_i, \mathbf{G}_j$ we have*

$$\|\kappa(\mathbf{G}_i) - \kappa(\mathbf{G}_j)\| \leq \|\mathbf{G}_i - \mathbf{G}_j\|_{\mathfrak{G}}.$$

Proof. We will show this with a proof by contradiction. Let us assume that the negation holds, meaning that there are two such games that

$$\|\kappa(\mathbf{G}_i) - \kappa(\mathbf{G}_j)\| > \|\mathbf{G}_i - \mathbf{G}_j\|_{\mathfrak{G}},$$

This would mean that

$$\|\#\{\phi(\mathbf{s}) : \forall_k r_{(\mathbf{G}_i)_k}(\mathbf{s}) = 0\} - \#\{\phi(\mathbf{s}) : \forall_k r_{(\mathbf{G}_j)_k}(\mathbf{s}) = 0\}\|$$
$$> \|\mathbf{G}_i - \mathbf{G}_j\|_{\mathfrak{G}} N_\phi.$$

The left hand side of the inequality measures the difference in the number of non-rewarding states. The right hand side measures the difference in the number of states that simply have a different reward (and thus already includes those counted on the left hand side). Clearly the left hand side cannot be strictly bigger than the right. Contradiction. □

Theorem 3.2. $\mathrm{coop}((\cdot, \mathbf{g}'))$ *is a $\frac{1}{1-k}$-Lipschitz function wrt. $\| \cdot \|_{\mathcal{G}}$ for any $\mathbf{g}$ such that $\kappa((\mathbf{g}, \mathbf{g}')) = k$.*

Proof. Let as assume that

$$\|\mathbf{g}_i - \mathbf{g}_j\|_{\mathcal{G}} = \frac{z}{N_\phi}.$$

From the definition of the metric this means there are exactly z predicate states where one of them is rewarding and the other is not.

Let us denote by y number of predicate states where both $\mathbf{g}_i$ and $\mathbf{g}'$ are rewarded. Then the number of predicate states where $\mathbf{g}_j$ and $\mathbf{g}'$ are rewarded has to belong to $(y - z, y + z)$. Now by denoting $\hat{k} = k N_\phi$ we have

$$\|\mathrm{coop}(\mathbf{g}_i, \mathbf{g}') - \mathrm{coop}(\mathbf{g}_j, \mathbf{g}')\|(N_\phi - \hat{k}) \leq z$$
$$= \|\mathbf{g}_i - \mathbf{g}_j\|_{\mathcal{G}} N_\phi,$$

and thus

$$\|\mathrm{coop}(\mathbf{g}_i, \mathbf{g}') - \mathrm{coop}(\mathbf{g}_j, \mathbf{g}')\| \leq \frac{N_\phi}{N_\phi - \hat{k}}\|\mathbf{g}_i - \mathbf{g}_j\|_{\mathcal{G}}$$
$$= \frac{N_\phi}{N_\phi - k N_\phi}\|\mathbf{g}_i - \mathbf{g}_j\|_{\mathcal{G}} = \frac{1}{1 - k}\|\mathbf{g}_i - \mathbf{g}_j\|_{\mathcal{G}}$$

It is natural to ask if the restriction imposed is not empty, but it is easy to prove that in the vicinity of any game there is another one satisfying said restriction.

Proposition B.1. *For any game $\mathbf{G} = (\mathbf{g}, \mathbf{g}')$ where $\kappa(\mathbf{G}) = k > 0$ there exists a goal $\mathbf{g}''$ such that $\kappa((\mathbf{g}, \mathbf{g}'')) = k$ and it is in vicinity of the previous game in the sense that*

$$\|(\mathbf{g}, \mathbf{g}'') - (\mathbf{g}, \mathbf{g}')\|_{\mathfrak{G}} = \frac{1}{2 N_\phi}.$$

Without loss of generality let us assume that $\mathbf{g}$ has at least one rewarding predicate state $\phi(\mathbf{s}^*)$. If $\phi(\mathbf{s}^*)$ is also rewarding for $\mathbf{g}'$ then we define $\mathbf{g}''$ as an exact copy of $\mathbf{g}'$, but set $\phi(\mathbf{s}^*)$ to be non rewarding thus the distance between the two is 1. If it was not rewarding in $\mathbf{g}'$ we symmetrically make it rewarding in $\mathbf{g}''$, again moving by 1 in the game space. The resulting game $(\mathbf{g}, \mathbf{g}'')$ has $\kappa((\mathbf{g}, \mathbf{g}'')) = k$ since we did not add any new rewarding predicate states.

$\square$

Theorem 3.3. *For every two player game $\mathbf{G}$ such that $\hat{\kappa}(\mathbf{G}) = k$ and a desired change in competitiveness $m \in (-\text{comp}(\mathbf{G}), 1 - \text{comp}(\mathbf{G}))$ such that $k|m| \in \mathbb{N}$ there exists a $\mathbf{G}'$ such that $\text{comp}(\mathbf{G}') = \text{comp}(\mathbf{G}) + m$ and $\|\mathbf{G} - \mathbf{G}'\|_{\mathfrak{G}} \leq \frac{k|m|}{2}$.*

Proof. Let us first assume that $m > 0$, consequently $\text{comp}(\mathbf{G})$ is smaller than 1, which means that if we look at $\mathbf{G} = (\mathbf{g}_1, \mathbf{g}_2)$ we can find at least $k \cdot (1 - m)$ predicate states, where $r_{\mathbf{g}_1}(\phi(\mathbf{s})_i) = r_{\mathbf{g}_2}(\phi(\mathbf{s})_i)$. Let us define

$$\mathbf{g}_2'(\phi(\mathbf{s})) := \begin{cases} 1 - \mathbf{g}_2(\phi(\mathbf{s})) \text{ if } \phi(\mathbf{s}) \in \{\phi(\mathbf{s})_i\}_{i=1}^{mk} \\ \mathbf{g}_2(\phi(\mathbf{s})) \text{ otherwise} \end{cases}$$

By construction $\kappa((\mathbf{g}_1, \mathbf{g}_2)) = \kappa((\mathbf{g}_1, \mathbf{g}_2'))$ and $\text{comp}((\mathbf{g}_1, \mathbf{g}_2)) + m = \text{comp}((\mathbf{g}_1, \mathbf{g}_2'))$ and $\|(\mathbf{g}_1, \mathbf{g}_2) - (\mathbf{g}_1, \mathbf{g}_2')\| = \frac{km}{2}$. Proof for $m < 0$ is analogous. $\square$

C| Proofs for Section 5 (Learning Process)

Theorem 5.1 (Value Consistency)**.** *For a goal $\mathbf{g} := \bigvee_{o=1}^{k}[\bigwedge_{c=1}^{n_o} \phi_{oc}]$ we have*

$$\mathbf{V}^*(\mathbf{g}_l) \leq \mathbf{V}^*(\mathbf{g}) \leq \mathbf{V}^*(\mathbf{g}_u)$$

for $\mathbf{g}_l := \bigvee_{o=1}^{k-1}\left[\bigwedge_{c=1}^{n_o} \phi_{oc}\right], \mathbf{g}_u := \bigvee_{o=1}^{k}\left[\bigwedge_{c=1}^{n_o'} \phi_{oc}\right]$ where $n_o' \geq n_o$.

Proof. Since $\mathbf{g}_l$ differs from $\mathbf{g}$ by simply missing the kth option, this means that the corresponding reward function

$$r_{\mathbf{g}_l}(\mathbf{s}) = \max_{o=1}^{k-1}\left[\min_{c=1}^{n_o} \phi_{oc}(\mathbf{s})\right] \leq \max_{o=1}^{k}\left[\min_{c=1}^{n_o} \phi_{oc}(\mathbf{s})\right] = r_{\mathbf{g}}(\mathbf{s}).$$

Consequently $\mathbf{V}^*(\mathbf{g}_l) \leq \mathbf{V}^*(\mathbf{g})$. Analogously, $\mathbf{g}_u$ differs from $\mathbf{g}$ by potentially having additional predicates in each options, this means that the corresponding reward function

$$r_{\mathbf{g}}(\mathbf{s}) = \max_{o=1}^{k}\left[\min_{c=1}^{n_o} \phi_{oc}(\mathbf{s})\right] \leq \max_{o=1}^{k}\left[\min_{c=1}^{n_o'} \phi_{oc}(\mathbf{s})\right] = r_{\mathbf{g}_u}(\mathbf{s}).$$

Consequently, $\mathbf{V}^*(\mathbf{g}) \leq \mathbf{V}^*(\mathbf{g}_u)$. $\square$