Supplementary Materials: Content Overview

- Appendix A contains details about each game element.
- Appendix B contains details about the reward function and starting energy levels.
- Appendix C contains a more detailed specification of the observation space.
- Appendix D contains a more detailed specification of the action space.
- Appendix E contains an overview of the different types of factors of variation in the generated worlds, with pointers to the full documentation in the code.
- Appendix F contains a brief description of the mechanics of each of the 16 basic tasks and four compositional tasks.
- Appendix G contains more details about the exact procedure for generating and selecting the worlds used for evaluation.
- Appendix H contains a description of the information that was provided to the human participants as well as other details about the human data collection.
- Appendix I contains more details about our exact curriculum learning algorithm.
- Appendix J contains the hyperparameters that were used for each network.
- Appendix K contains more details on the exact compute hardware used for training.
- Appendix L contains additional experimental result tables and figures, including optimality gap scores, scores for all training conditions, performance breakdown by task, and learning curves.
- Appendix M contains more details on how scores were calculated.
- Appendix N contains more details about the exact simulator performance under different conditions.

Our code is is publicly available, and links to the code and data can be found at at `https://generallyintelligent.ai/avalon`.

A Game details

A.1 Terrain, biomes, scenery, objects

The base terrain is randomly generated via an iterative process of subdivision, bicubic interpolation, and adding various types of random noise at progressively smaller scales (see the `build_outdoor_world_map` function in the code for an exact specification). This heuristic approach is designed to mimic the rugged surfaces of natural landscapes and mountains without using a significant amount of compute. A plane of water intersects the island at a specified height to form the surrounding body of water as well as pools within.

The terrain slope, elevation, and distance from water map every point to a set of 7 visually distinct biomes: *tropical rain forest, tropical seasonal forest, temperate rain forest, temperate deciduous forest, grassland, temperate desert,* and *subtropical desert*. Additional logic shapes regions around water into more specialized biomes such as beaches and swamps (*water, fresh water, coastal, swamp, dark shore*) and adds a few bigger mountains and hills (whose surfaces are either *bare*, in which case they can be climbed, or *unclimbable*, in which case they cannot) to the landscape. We have tuned the resulting distributions to get worlds that feel both natural and high-variance. The island is then populated with foliage in a biome-dependent manner, with different levels of noise on the borders and density variations that create clearings for the player or agent to walk through.

Scenery elements include four types of trees, a bush, a flower, and a mushroom. Their arrangement and density are biome-specific — for example, tropical biomes have mostly palm trees. Each piece of scenery populates specific biomes at specific spatial densities, with additional logic at biome boundaries and near coasts. All scenery objects have noise applied to their colors, scales, and orientations. Trees are climbable so that players can use them to escape predators or gain a better vantage point, while other smaller scenery items are non-colliding (agents can simply walk through them). In addition to scenery objects, a series of interactable items are scattered about in different locations and densities depending on the task. These include logs, sticks, large heavy boulders, medium-sized stackable stones, and fist-sized rocks. Many tasks require that these items be either maneuvered into new positions or used as tools or weapons.

Figure 3: **Scenery objects.** Left to right: maple trees, acacia trees, fir trees, palm trees, and a composite shot that includes bushes, flowers, and a mushroom.

Figure 4: **Interactable items.** Left to right: log, boulders, sticks, and rocks.

A.2 Obstacles

To procedurally enforce obstacles between the player and food, the terrain is shaped into irregular rings encircling either the player spawn point or the food location. These rings can consist of cliffs, chasms, ridges, or sudden drops that make it impossible for the player to get to the food without accomplishing the relevant task. Compositional tasks use concentric rings to impose a series of obstacles between the player and food. Noise is added to these rings to make them resemble more natural formations.

A.3 Fruit trees and fruits

For most individual tasks, the available food item is a fruit that can be found either on or under a large fruit tree. Fruit trees are taller and more visually striking than other foliage (see Figure 5) so that identifying distant food is not akin to spotting a needle in a haystack.

The canonical fruit is an apple, but there are nine others that differ from it in some key aspect. These were selected not only for visual variety but to challenge the agent in different ways; for example, some require special preparation to become edible, while others are delicate and can become inedible with the wrong treatment. The full list of fruits and their properties is provided in Table 2 (the word "fruit" is used loosely here, as it also includes honeycombs and avocadso). There are also carrots, which are the only food that can grow anywhere i.e. not restricted to be found under a tree or in a building.

The only tasks that feature a fruit other than the canonical apple are `eat`, which has a single fruit that can be any of the 10, and the compositional task `survive`, which has multiple different types of fruits scattered around the island.

A.4 Buildings and doors

In addition to the natural features of the island, Avalon also includes procedurally-generated buildings which can be entered and explored. Buildings are used as an alternative site for placing fruits and can serve as a site for some tasks such as `open`, `push`, `navigate` and `stack`. Buildings may have one or more stories and parts of the buildings may be climbable. They are also guaranteed to contain at least

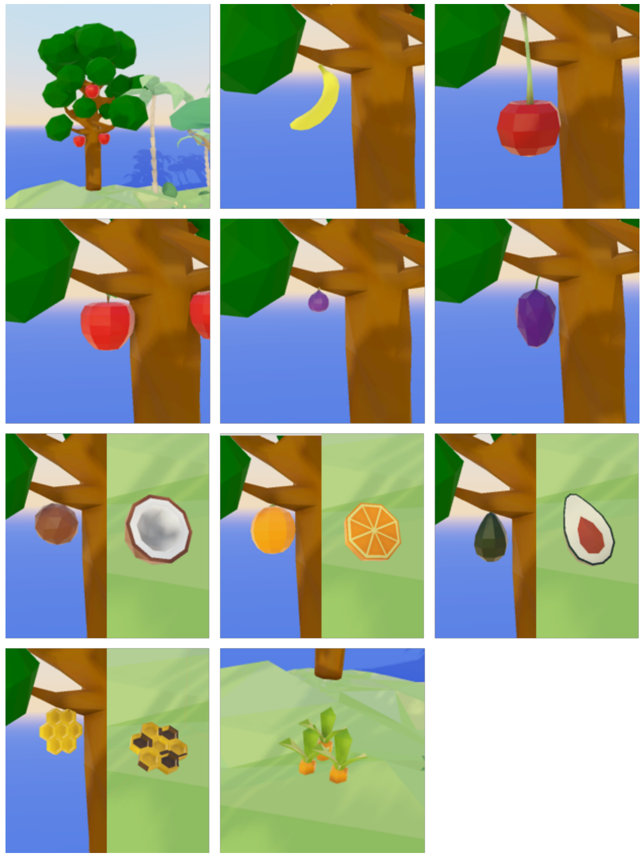

Figure 5: **Fruit tree and fruits**. Top left to bottom right: fruit tree, banana, cherry, apple, mulberry, fig, coconut (left: on tree, right: opened), orange (left: on tree, right: opened), avocado (left: on tree, right: opened), honeycomb (left: on tree, right: dirty), carrots.

Table 2: Fruits and their properties.

Fruit	Description
Apple	Canonical fruit.
Banana	Provides less energy.
Fig	Inedible (splattered) if it experiences too much force.
Orange	Must be opened by making it experience a force.
Avocado	Must be opened by hitting with a rock.
Coconut	Must be opened by dropping from a height.
Honeycomb	Inedible (dirty) if it contacts the ground.
Cherry	Dark apple with long stem, always multiple.
Mulberry	Smaller than other fruit.
Carrot	Must be pulled from the ground by its leaves.

one piece of food. The `open` task takes place exclusively in buildings, with a range of door types and locks corresponding to different difficulties of the task. Doors can be unlocked or locked, and rotating (so the agent can push through it), or sliding (so the agent must drag it to the side). Doors can have between zero and three locks. There are three types of locks: a deadbolt that must be slid to the side, a rotating bolt that must be rotated to the side, and a timed button switch that toggles the ability to move the locks or open the door.

A.5 Predators and prey

There are 17 animals (nine predators, eight prey) that appear in certain tasks. Animals were selected to span a wide range of behaviors and game mechanics, described in detail below. Depending on the task, animals are either populated randomly or clustered near the food. The full list of animals and their properties is provided in Tables 3 and 4 and screenshots are provided in Figures 6 and 7. In the remainder of this section, we briefly describe the axes of variation.

Table 3: Predators (ordered from easiest to hardest) and their properties.

Animal	Domain	Activation	Deactivation	Idle Behavior	Active Speed	Attack	Defense
Bee	Air	Within Radius (Small)	Outside Radius, KO	Territory Bound, Fast Random	Fast	-0.25 Perm. Respite	$+1$
Snake	Ground	Within Radius (Small)	KO	Static Avoidant	Fast	-5.0	$+1$
Hawk	Air	Within Radius (Wide) Within Territory	Outside Radius Outside Territory, KO	Territory Bound, Periodic Fixed	Fast	-0.25 Temp. Respite	$+1$
Hippo	Ground, No Inside	Within Radius (Wide)	Outside Radius, Outside Domain	Static Avoidant	Fast	-0.5 Persistent	$+\infty$
Alligator	Ground	Within Radius (Wide)	Outside Radius, Outside Domain, KO	Slow Random	Slow	-0.5 Persistent	$+1$
Eagle	Air	Within Radius (Wide), You Don't See It	You See It, KO	Fast Random	Fast	-0.5 Persistent	$+1$
Wolf	Ground	Within Radius (Wide), Within Territory	Outside Radius, Outside Domain, KO	Territory Bound Slow Random	Fast	-0.5 Persistent	$+2$
Jaguar	Ground, Climb	Within Radius (Wide), Must See You	Outside Radius, You Stop Moving, KO	Periodic Fixed	Fast	-0.5 Persistent	$+2$
Bear	Ground, Climb, No Inside	Within Radius (Wide)	Outside Radius, Outside Domain	Slow Random	Fast	-0.5 Persistent	$+\infty$

Table 4: Prey (ordered from easiest to hardest) and their properties.

Animal	Domain	Activation	Deactivation	Idle Behavior	Active Speed	Defense
Frog	Ground	n/a	KO	Slow Random	n/a	$+1$
Turtle	Ground	Within Radius (Small)	Outside Radius, KO	Periodic Fixed	Slow	$+1$
Mouse	Ground, Climb	Within Radius (Small), Must See You	Outside Radius, Can't See You, KO	Fast Random	Fast	$+1$
Rabbit	Ground	Within Radius (Wide)	Outside Radius, KO	Fast Random	Fast	$+1$
Pigeon	Air	Within Radius (Small)	Outside Radius, KO	Territory Bound, Fast Random	Fast	$+1$
Squirrel	Ground, Climb	Within Radius (Small)	Outside Radius, KO	Fast Random	Fast	$+1$
Crow	Air	Within Radius (Wide)	Outside Radius, KO	Slow Random	Fast	$+1$
Deer	Ground	Within Radius (Wide)	Outside Radius, KO	Static Avoidant	Fast	$+2$

Domain. Animals are either confined to the ground or exclusively fly. Some ground animals can climb. All animals can enter buildings except bears and hippos, which are too big to fit through doors.

Activation and deactivation. At any given moment, an animal is either *active* or *inactive*, with inactivity the default. Different predators and prey have different activation (deactivation) conditions for when they start (stop) chasing or fleeing the player, respectively. For example, some only activate when the player is nearly on top of them while others activate from a wider radius. Activation conditions are often mirrored in deactivation conditions, though this is not always the case; for example, hawks and wolves both activate when the player enters their territory, but wolves continue the pursuit even after the player exits their territory. Most predators and all prey can be deactivated by killing, but two predators are indestructible (hippo, bear). Another way to deactivate some predators is by triggering the Outside Domain condition. For example, predators that can neither fly nor climb can be deactivated by climbing a tree. Hippos and bears are too big to fit through doors so going inside a building triggers this condition for them. Some special activation conditions: "Must See You" means the player must be within the predator's field of view and "You Don't See It" means it sneakily attacks when the player is not looking. For deactivation conditions, "You See It" is the mirror of "You Don't See It" and "You Stop Moving" is akin to the freeze defense strategy for not getting eaten.

Idle behavior. While inactive, animals display different default idle behaviors. Some move randomly (fast or slow) while others move along fixed periodic trajectories. Some only prowl within their territory. Static Avoidant animals are static until the player approaches, then slowly move away to increase their distance from the player, then finally activate if the player get within their activation threshold.

Active speed, attack and defense stats. When active, some animals are slower than the player, while others are faster. Some predators attack repeatedly (Persistent) while others attack once and then leave for a bit (Temporary Respite). Bees attack once and then die (Permanent Respite). Attacks can be high damage or low damage. A single snake attack is fatal. Some animals can be disabled with one hit, others need multiple hits, and some are indestructible.

Figure 6: **Predators.** Top left to bottom right: wolf, bear, jaguar, crocodile, hippo, snake, hawk, eagle, bee.

Figure 7: **Prey.** Top left to bottom right: turtle, squirrel, rabbit, deer, mouse, frog, pigeon, crow.

B Reward function

In all tasks, the agent's reward is given by the same reward function of its energy level over time. Agents begin each episode with a certain initial energy and must inevitably expend energy in moving to complete the task. The only way to gain energy is by eating. In addition to energy expenditure from motion, energy can be lost by taking damage from enemies or falls. Episodes terminate for one of three reasons:

i) The agent energy goes to zero.

ii) The agent eats all the food in the episode (the episode ends 10 frames later).

iii) The episode times out.

The determination of the initial energy, definition of energy expenditure, and implementation of termination conditions vary slightly between training and evaluation settings, described in Section B.1. The formulas for each energy term and values of associated constants are provided in Section B.2.

B.1 Training and evaluation

During training, the agent receives dense reward equal to its framewise change in energy level, together with framewise penalties. This reward R is the sum of any energy gained from eating E_{food} minus any energy lost from damage due to falling E_{fall} and predators $E_{\text{predators}}$, minus the movement penalty P_{move} and frame penalty P_{frame},

$$R = E_{\text{food}} - E_{\text{fall}} - E_{\text{predators}} - P_{\text{move}} - P_{\text{frame}} \tag{1}$$

Each energy term is discussed in detail in Section B.2. During training and evaluation, the initial energy level $E_0 = 1$. The time limit during training depends on the difficulty level and task, tuned such that 98% of humans trials lie within the time limit. These time limits improve compute efficiency during training, terminating episodes in which the agent cannot perform the task.

B.2 Energy terms

Each of the energy terms in Equation 1 are described below, with all constants listed in Table 5.

Food. Food is eaten once it is held and within 0.5m of the head. All food (fruits and prey) provides $E_{\text{food}} = 1$ upon being eaten, except for bananas which provide $E_{\text{food}} = 0.5$ (see Table 2).

Falling. Fall damage occurs if the vertical speed of the agent v_{vertical} is above a certain threshold $v_{\text{threshold}}$ when colliding with the ground, with the energy penalty from a fall given by

$$E_{\text{fall}} = C_{\text{fall}} m_{\text{body}} \max\left(0, v_{\text{vertical}}^2 - v_{\text{threshold}}^2\right) \tag{2}$$

The energy coefficient C_{fall} and threshold $v_{\text{threshold}}$ are set so the agent of body mass m_{body} would not take damage from any fall smaller than 12 meters and take lethal damage when falling from anything greater than 28 meters. See Table 5 for details.

Predators. The energy cost of predator attacks depends on the predator as specified in Table 3.

B.3 Penalty terms

The penalty terms differ from the energy terms in that they do not modify the total agent energy: they do not modify the ability of the agent to survive or act in its environment. Rather, these terms are designed to assist the agent in choosing more appropriate actions which could improve learning and generalization. As such, they contain a number of parameters that may be tuned to optimize agent performance.

Frame. Our scoring criteria (Section M) adds a bonus if all the food on a level is consumed before the end of the episode. This bonus is proportional to the number of frames remaining before the timer would terminate the episode. We use the same proportionality constant used for scoring (1e-4) as our frame penalty P_{frame}.

Movement. As stated in Section 3, the agent controls its head, body, and two hands. The head and body are generally controlled simultaneously. Translating the head in any direction except vertically translates the entire agent. Similarly, rotating the head about the vertical axis (i.e. looking from side to side) reorients the entire agent. However, when it comes to the vertical axis, the agent's head and body are treated as distinct. When translating the head vertically, only the head moves; this is to enable crouching, during which the agent is still resting on the ground but its vantage point is lower. To vertically translate the agent's body as well, such that its distance from the ground changes, the agent must jump, climb, or fall. Similarly, rotating the head about axes other than the vertical axis only reorients the head, not the body. This is so that the agent can look up and down or tilt its head to look at things without doing somersaults.

With this formalism in place, the energy cost of movement can be stated as the sum of kinetic ΔK and gravitational potential ΔP energy changes for each of the agent's head, body, and two hands,

$$P_{\text{move}} = C_{\text{move}} \left(\sum_{\substack{i \in [\text{body}, \\ \text{hands}]}} (\Delta K_i + \Delta P_i) + \Delta P_{\text{head}} \right), \tag{3}$$

where the energy costs of the body capture all global translations and rotations of the full agent and the energy costs of the head and hands are calculated relative to the body. For the head, there is only a potential energy cost to penalize keeping the head tilted; there is no kinetic energy cost for crouching. The overall movement coefficient C_{move} and the kinetic and potential energy terms are determined by the physics-based formulas,

$$\Delta K_i = C_{\text{K}i} m_i \max(0,\, \bar{\mathbf{v}} \cdot \Delta \mathbf{v}), \tag{4}$$

$$\Delta P_i = C_{\text{P}i} m_i g \max(0,\, \Delta h), \tag{5}$$

where $C_{\text{K}i}$ and $C_{\text{P}i}$ are coefficients that can be tuned, $\bar{\mathbf{v}}$ is the average velocity over the last step, $\Delta \mathbf{v}$ is the change in velocity, m_i is the mass of the part i, g is the gravitational acceleration, and Δh is the change in height over the last step. The constants used to calculate these energy terms are listed in Table 5. During training, C_{move} was tuned with other hyperparameters. The body part masses were set to be close to the average human equivalents.

Table 5: Parameters used to calculate fall and movement energy values. C_{move} is a hyperparameter that can be adjusted for training. During evaluation, $C_{\text{move}} = 0$

Parameter	Value
C_{fall}	0.000026
$v_{\text{threshold}}$ (m/s)	10
C_{move}	1e-8
P_{frame}	1e-4
m_{body} (kg)	60
m_{head} (kg)	4.8
m_{hand} (kg)	3.0
g (m/s^2)	10
C_{K}	0
C_{P}	1

C Observation space

The agent's observation space is made up of $96 \times 96 \times 4$ egocentric RGB + depth images and the following 23 proprioceptive inputs:

- 4×2 variables (3D vectors) corresponding to the framewise change in position and rotation of the agent's body, head, and two hands.
- 3×2 variables (3D vectors) corresponding to the position and rotation of the agent's head and two hands relative to its body.

- 2×1 variables (booleans) corresponding to whether each hand is colliding with anything.
- 2×1 variables (booleans) corresponding to whether each hand is grasping something.
- 4 variables (floats) corresponding to the agent's energy expenditure from movement, energy lost from enemy attacks, energy gained from eating, and current energy level.
- 1 variable (float) corresponding to the frames remaining before the episode is terminated by the timer.

D Action space

The full 21-dimensional action space includes $3 \times (3 + 3)$ continuous degrees of freedom for the 3D translation and rotation of the agent's three components (head and two hands), as well as 2×1 binary grasp actions for the hands and 1 binary jump action. This action space roughly corresponds to the controls of a VR headset and was used for all experiments in the paper. Human players were also allowed to use the controller joystick for movement and rotation, but these inputs were carefully mixed into the simpler action space in which the agent acts so that the two could be exactly equivalent.

We also provide a reduced 9-dimensional action space which maps to mouse and keyboard controls. In this reduced action space, there is a single translational degree of freedom for forward/backward motion, with the agent always moving in the direction it is facing, which is controlled by two angular degrees of freedom (pitch and yaw). In addition to the 2×1 binary grasp actions and 1 continuous jump action, this setting also has 1 discrete action for eat and 2×1 discrete actions for throw.

E Factors of variation

The procedure that generates Avalon's levels is parameterized by many *factors of variation* which determine the structure and appearance of the generated world. These factors of variation can be broken down into settings that affect terrain (shape and color), scenery (trees, bushes, flowers, etc), environment (sky, lighting and graphics options), buildings, items (animals, tools, and food), and tasks (distributions over factors like how far to jump, how many enemies to include, etc). Two levels generated from the same factors of variation will share these high-level features but will differ in various low-level details like object positions and the exact topography of the island.

Due to the focus on variety and diversity in Avalon, the factors of variation are mostly defined in code rather than in external configuration files. This also allows for better specification of allowed values (via type signatures).

E.1 Terrain

The terrain (base world geometry) is generated via repeated subdivision and addition of various types of noise. The `WorldConfig` object defines 34 properties that control this procedure, including `fractal_iteration_count`, `noise_scale_decay`, and `size_in_meters`. See the code for a complete list and documentation for each value.

The `generate_world_config` function gives an example of how to dynamically generate interesting, varied `WorldConfigs`. The `build_outdoor_world_map` converts a `WorldConfig` into a `HeightMap`. It can easily be swapped out for any other approach to generating a `HeightMap` (grid of heights).

After generating the base terrain, the world is broken into different "biomes" that control which scenery objects will be placed in that region, as well as the colors used for the terrain in that region. Some of these biomes affect the height of the world (ex: the beach biome controls the erosion of shores near the ocean). This entire process of biome assignment and calculation is controlled by the `generate_biome_config` function, which generates a `BiomeConfig`. This object has 43 properties such as `beach_slope_cutoff`, `swamp_elevation_max`, and `rock_color_noise`. See the code for a fully documented list.

E.2 Scenery

In order to strike a good balance between visual complexity and performance, biomes in Avalon are populated with instanced scenery models. These models are given per-instance and per-vertex color noise, as well as per instance scaling and rotation variation without really incurring any significant performance overhead. These attributes are controlled by the `FloraConfig` and `SceneryConfig` objects, which together define 10 attributes like `density`, `border_mode`, and `correlated_scale_range`.

E.3 Environment

All aspects of the Godot `WorldEnvironment` and `Sky` objects are procedurally generated, as well as all aspects of the `Sun` light that is created in each scene. This enables varying factors such as `sun_latitude`, `fog_color` and `tonemap_mode`. We give examples of setting 46 of these, though all supported properties of these objects in Godot can be set directly. See the Godot documentation for more details on each setting.

E.4 Buildings

Buildings are used both as components of compositional worlds and as variants of the basic tasks (ex: `explore` tasks are set either inside of a building or in an outdoor, natural world). The appearance of buildings can be changed via the `BuildingAestheticsConfig`, which contains 20 attributes like `desired_story_count`, `window_width` and `trim_color`

E.5 Items

All items, including animals, have a variety of attributes which can be set directly and differ per-item. See `items.py` for a complete definition of all attributes. Each item also has `safe_scale` and `base_color` attributes, which can be used to explicitly set the size and color of each object. Animals each have a variety of behavior-dependent attributes like speed, activation radius, etc that can be altered for each instance.

E.6 Tasks

Tasks are each generated by a single function, parameterized only by difficulty. This function converts that difficulty value into all other lower-level factors of variation (e.g. jump distance for `jump`, path width for `move`, etc). See the corresponding `.py` files in the `tasks` folder for a complete specification of each task's parameters. Tasks which can be composed also contain a single function to create the given type of obstacle (ex: a gap for jumping over, a path for walking along, etc), and these functions are used by `compositional.py` to implement the compositional tasks.

F Task definitions

Each task in Avalon maps to a set of generated worlds. Each world is set up such that the agent must usually complete the task in order to successfully reach the food. Each task has a variety of individual parameters, controlled in aggregate by the difficulty parameter, which ranges from 0 to 1. See Figure 9 for examples of how tasks vary by difficulty.

Almost every task world has only one fruit or prey; after the agent eats the fruit or prey, the episode ends. The two exceptions are `gather`, which has multiple fruits, and `survive`, which has multiple prey and fruits. To enable visibility for the agent in rugged terrain, the fruit in each world is always on a fruit tree or inside a building.

Each tasks is set up with the goal of isolating the skill being learned. Thus, all tasks only have apples as canonical fruit at all difficulty levels, with the exception of more difficult `eat` worlds, which contain harder-to-eat fruits; `hunt` and `throw`, which contain only prey and no fruit; and `survive`, which can contain all fruit and prey types. This is intended to isolate the skills needed for each task from the skills needed to eat more complex fruits. Additionally, in most tasks, with the exception of `explore`, `find`, `gather`, and `survive`, the agent and the fruit spawn in locations such that the agent can see where the fruit is at spawn. In this case, the agent may not be facing the fruit at spawn, but upon turning in place will be able to see the fruit. This is intended to isolate the skills required for each task from the skills required for exploring the world, which is tested in isolation in `explore`, as well as in the compositional tasks.

F.1 Basic tasks

There are 16 "basic" tasks. Each basic task generates a world in which the agent usually must complete that task in order to reach and eat the food.

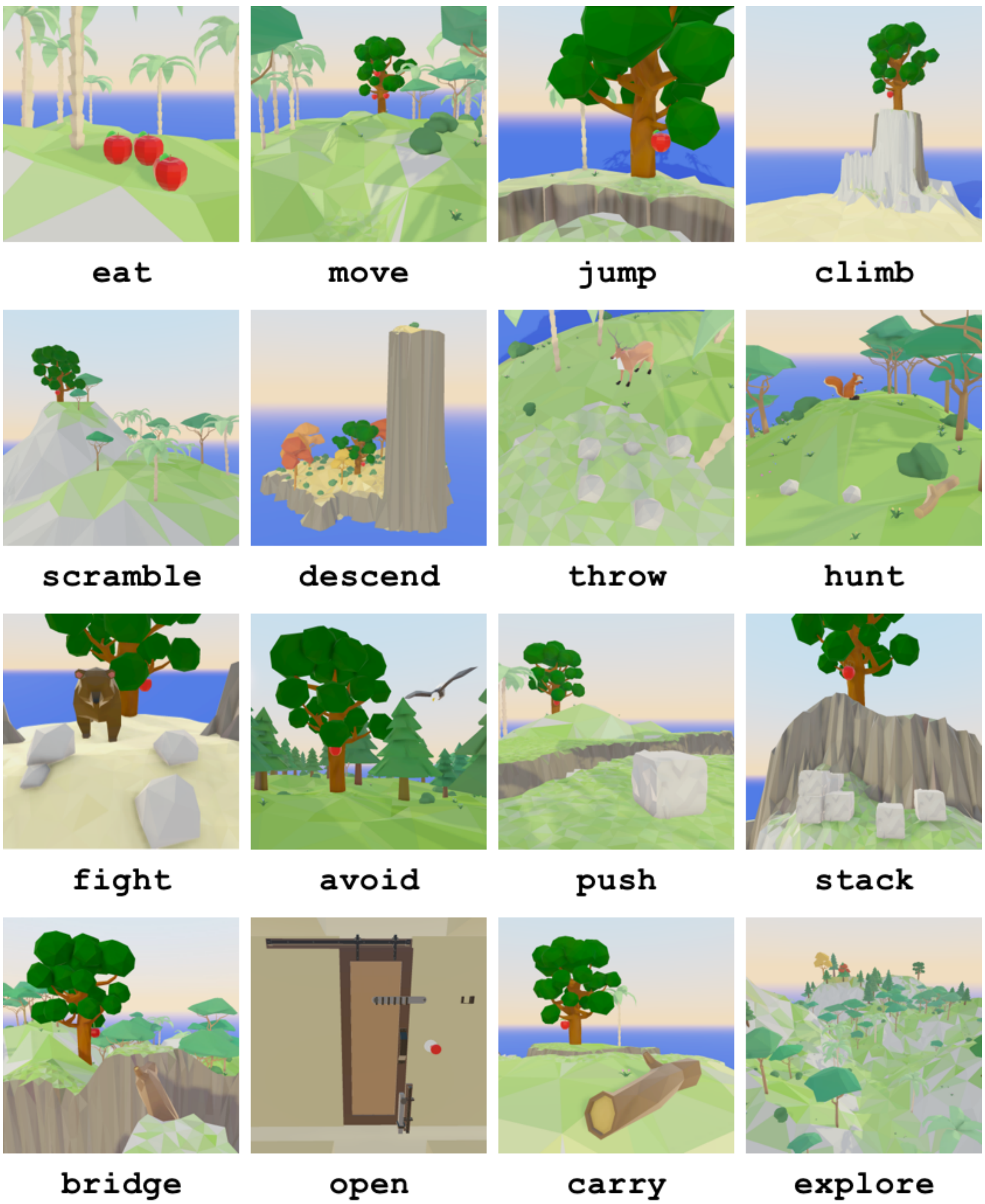

Figure 8: **Basic tasks.** All 16 basic tasks. For each task, the world is generated such that the agent must complete the task in order reach and eat the food.

Eat. In `eat`, the agent starts in front of the fruit, and must grab the fruit and bring the fruit to within a radius of its head in order to succeed. As difficulty increases, worlds contain different types of fruit that must be opened in more complex ways (see Appendix A for details on fruit types).

Move. In `move`, the agent starts a distance away from the fruit, and must move its body to reach the fruit. As difficulty increases, the agent spawns farther away from the fruit, and the paths leading to the fruit become narrower. At the highest difficulties, the agent must traverse thin, multi-segment paths across chasms in order to reach the fruit.

Jump. In `jump`, the agent must jump over a chasm to reach the fruit. As difficulty increases, the chasm gets wider, and the area that can be jumped across gets narrower, such that the agent must figure out where it can successfully land a jump. If agents fall into the chasm while jumping, they can climb back out to the original side and retry the jump.

Climb. In `climb`, the agent must climb up a cliff wall in order to reach the fruit. The climbing motion requires repeatedly bringing one hand up, grabbing the cliff, pulling the hand downward, and then doing the same with the other hand, without letting go of the cliff. As difficulty increases, the cliff gets taller so the agent must climb further, and the climbable path gets narrower so the agent has less area to grab onto. The climbable path also contains multiple, angled segments at higher difficulties, requiring more than just climbing in a single straight line.

Scramble. In `scramble`, the agent must combine walking, jumping and climbing to move over terrain to reach the fruit. As difficulty increases, the terrain becomes more mountainous, requiring more climbing and less jumping.

Descend. In `descend`, the agent must descend down a cliff. This can be accomplished by climbing down, or falling while grabbing the cliff wall periodically. On easier difficulties, there is also a platform partway down that can be used to safely descend by first falling on the platform, then falling the rest of the way. If the agent just jumps off the cliff randomly (without falling on this platform), it will take damage. As difficulty increases, the cliff gets taller so the agent will take more damage (and, at the highest difficulties, die). The path that can be grabbed also gets narrower, so the agent must figure out where to descend from.

Throw. In `throw`, the agent must throw a rock at prey in order to kill it, and then eat it in the same way fruit is eaten. `throw` worlds only have prey, and no fruit. As difficulty increases, the agent starts farther away from the prey, and the world gets larger, so the prey can run away. Additionally, the types of prey that spawn are harder to hit (e.g. pigeons that fly, instead of frogs that sit on the ground and hop slowly).

Hunt. In `hunt`, the agent must find the prey and kill it either by throwing a rock, or hitting it with a stick. As difficulty increases, the world gets larger, the types of prey that spawn are harder to hit, and fewer tools spawn for hunting. The agent may only get one rock, or one stick.

Fight. In `fight`, the agent must use sticks or stones to fight a predator that is guarding the food in order to reach and eat the food. While it is not strictly necessary to defeat the predators in order to reach the food in some levels, as difficulty increases, the predator types that spawn are more aggressive and more difficult to hit, and there are more of them, effectively forcing the agent to fight.

Avoid. In `avoid`, the agent must avoid predators that are near the food in order to reach and eat it. Unlike in `fight`, the agent is not provided with sticks or stones for fighting the predator. As difficulty increases, the number of predators increase, and more aggressive types of predators become more likely to spawn.

Push. In `push`, the agent must push a heavy boulder into a position where it can be used as a stepping stone for jumping onto the cliff ledge where the fruit is. As difficulty increases, the boulder gets heavier and more difficult to control, and must be pushed farther. This task has both indoor and outdoor variants.

Stack. In `stack`, the agent must stack stones and jump on top of them in order to reach the cliff ledge where the fruit is. At low difficulties, the agent only needs to use one stone to reach the top. As difficulty increases, the cliff ledge gets taller, so more layers of stones need to be stacked to reach the top. At the highest difficulty, the agent must stack a pyramid with four layers of stones in order to reach the top. This task has both indoor and outdoor variants.

Bridge. In `bridge`, the agent must pick up a log and lay it across a chasm in order to create a bridge to walk over the chasm. At low difficulties, the log is sometimes already in the solved position. As difficulty increases, the log starts out farther away, and the width of the chasm section that is narrow enough to be bridged shrinks, so the agent must find the right place to bridge the chasm.

Open. In `open`, the agent starts inside a building and must open a door to get into the room where the fruit is. See Appendix A for details on types of doors and locks. At the easiest levels, doors are rotating and can be walked through. As difficulty increases, more difficult variants of doors (such as sliding doors and doors that must be pulled rather than pushed) with more difficult locks (such as the timed switch) appear more often. At the highest difficulty, doors can all three locks.

Carry. `carry` spawns a task world that requires objects from one of `throw`, `fight`, `stack`, or `bridge`, and then moves the objects a distance away from where they would normally spawn. Thus, the agent must carry the object to the location where it will be used. As difficulty increases, objects spawn farther away and must be found and carried a longer distance.

Explore. In all other basic tasks, fruit or prey is visible from where the agent spawns. In `explore`, fruit cannot be seen from where the agent spawns. Thus, the agent must explore the terrain in order to find the fruit. This task has both indoor and outdoor variants. In the outdoor variant, as difficulty increases, the world gets larger and the terrain rockier, making it more difficult to spot and reach the fruit. In the indoor variant, the building to be explored becomes progressively larger with higher difficulties.

F.2 Compositional tasks

There are four "compositional" tasks. Compositional tasks generate worlds in which the agent must complete a sequence of multiple basic tasks in order to get food. To prevent the agent from bypassing any task, compositional tasks use concentric rings of terrain obstacles to impose a sequence of task obstacles, or spawn buildings that require one of the basic tasks.

Navigate. In `navigate`, the fruit is visible from where the agent spawns, and the agent must navigate through a variety of basic tasks to reach the fruit. As difficulty increases, the world gets larger, with more difficult terrain, and the number and difficulty level of basic tasks that must be solved also increases (up to a maximum of four basic tasks).

Find. `find` is like `navigate`, but the fruit is not visible from where the agent spawns. In order to achieve maximal score on the task, the agent must find the fruit and solve a variety of basic tasks to reach the fruit before time runs out. As difficulty increases, the world gets larger and more mountainous, and the number and difficulty of basic tasks to be solved along the way also increases.

Gather. `gather` is like `find`, but with multiple pieces of fruit in the world. None of the fruit is guaranteed to be visible from where the agent spawns. Thus, the agent must find and reach all pieces of fruit before time runs out, solving obstacles along the way, in order to achieve max score on the task. As difficulty increases, the world gets larger and more mountainous, the distance between fruits increases, and the number and difficulty of basic tasks to be solved increases.

Survive. `survive` is like `gather`, but with prey and predators scattered about the world, in addition to fruit and some basic task obstacles. Unlike in `navigate`, `find`, and `gather`, obstacles don't necessarily appear at all or need to be solved in sequence. Instead, the agent's goal is to survive as long as possible by eating prey or fruit, avoiding predators, and solving obstacles in order to find more food. As difficulty increases, the world gets larger and more mountainous, and there are more predators of higher difficulty and fewer prey and fruit. `survive` at the high difficulty levels is in some ways the pinnacle task: the agent usually needs to have learned a wide range of skills from all other tasks in order to achieve maximum scores on the most difficult survive levels.

G Evaluation worlds

G.1 World generation

A fixed set of 50 worlds for each task was generated to evaluate human and agent performance. Each world was generated with a unique seed while difficulties ranged between 0.0 and 1.0 and the exact distribution of difficulties was task dependent. For each task type, 20% of the worlds were set to

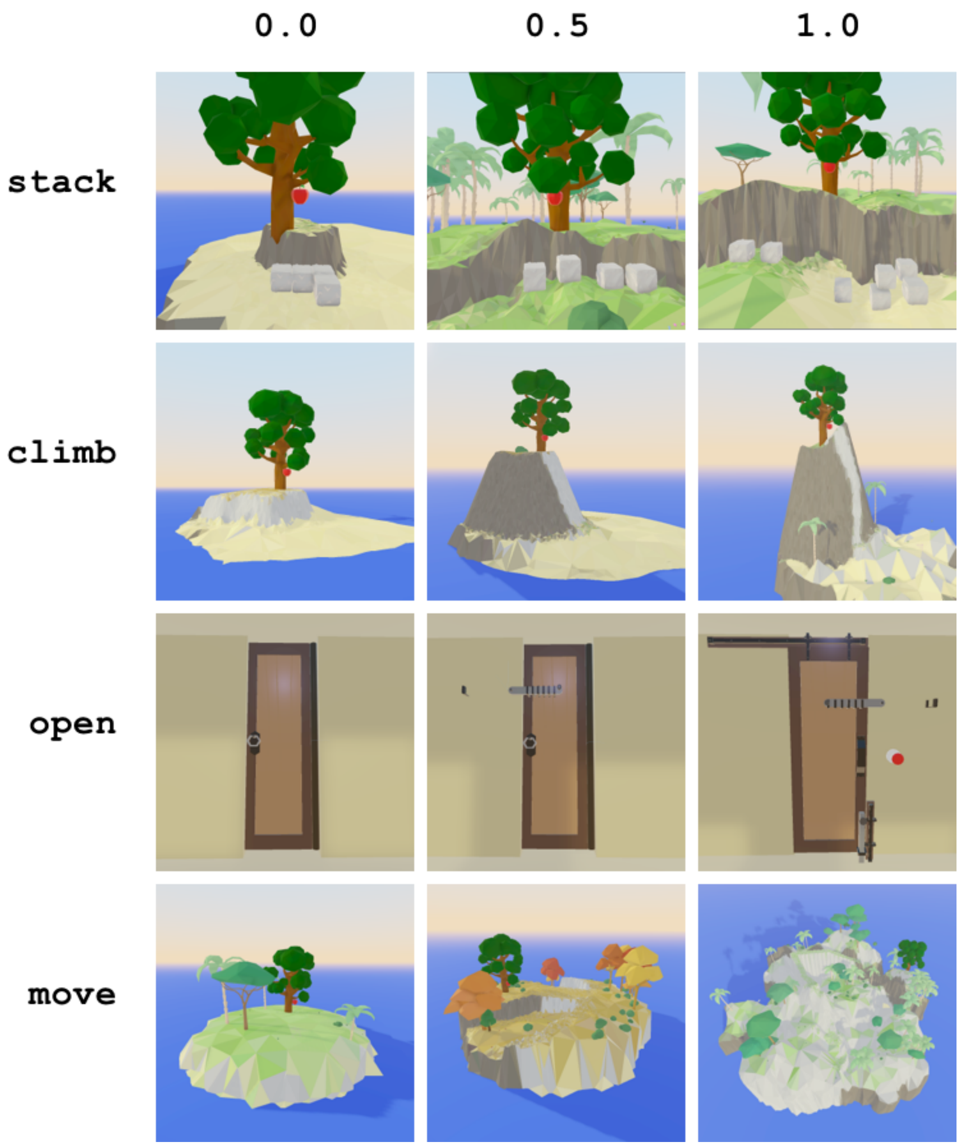

Figure 9: **Task variation with difficulty.** Examples of how tasks vary as difficulty increases. `stack` 0.0 is already solved and the agent just needs to jump on the blocks to reach the landing; `stack` 1.0 requires the agent to create a pyramid of blocks that is three blocks high in order to reach the landing. `climb` 1.0 requires climbing a higher cliff, on a narrower path than `climb` 0.0. In `open` 0.0 the agent can directly walk through the door; `open` 1.0 requires the agent to unlock three locks, including a timed switch, to open the door. `move` 0.0 requires moving a short distance on flat land to get food, whereas `move` 1.0 requires moving a longer distance on a narrow path surrounded by cliffs.

have difficulty 1.0 and the remaining worlds had difficulties that were evenly spaced between 0.0 and 1.0. For `hunt`, `avoid` and `eat`, a world for each type of food, prey and predator were generated to guarantee that each type of entity was present in an evaluation world. For `avoid` and `hunt`, the difficulties of these forced worlds were set to 1.0, while for eat the difficulty was set to 0.5. The forced levels were created first and then the same procedure as the other tasks was used to generate the remaining worlds.

Figure 10: **Compositional tasks.** All four compositional tasks. Compositional tasks generate worlds in which the agent must complete a sequence of multiple basic tasks in order to get food.

G.2 Replaced worlds

During data collection, participants reported worlds that were impossible or very difficult to complete. Of the original 1,000 levels generated (50 levels $\times$ 20 tasks), there were a total of 30 levels (3%) that had any issues at all. Ultimately, 10 levels (1%) were replaced.

Four levels were impacted by 2 bugs which have since been fixed. When these issues were encountered during data collection, we simply generated new worlds with an increased seed (and the same difficulty level) to replace them in the evaluation set. Twenty-six levels did not have successful playthroughs in the initial round of data collection. These levels tended to be on the highest difficulty setting and most difficult tasks, and are listed in Table 6. Of those 26 levels, we were able to eventually solve 20 by simply having more players try them. Of those 20 levels, 9 required multiple attempts from the same individual before they were ultimately solved; however, multiple attempts were needed only due to the difficulty of execution, not because they required advance knowledge of the level, and thus it seems likely to us that the level could have been solved on the first attempt given enough skilled players (but unfortunately we did not have enough study participants to verify). The remaining 6 did not get successful playthroughs even after additional attempts, and thus were replaced. However, only 1 of these 6 was truly impossible (the terrain is extremely jagged, leading to a situation where the fruit spawned in an area that is unreachable) whereas the others were merely very difficult.

In summary, 10 of the generated worlds (1%) were replaced, but of those only 1 (0.1%) was due to a level actually being impossible. We reiterate that the evaluation set does not contain any unsolvable worlds as they were all replaced, but we report these issues here for the sake of transparency.

G.3 Time Limit

To limit the amount of time a participant could spend on a world, the following time limits were used:

- 15 minutes for `navigate` and `find`
- 10 minutes for `survive`, `gather`, `stack`, `carry` and `explore`
- 5 minutes for all remaining tasks

When evaluating the agent, the maximum roll-out length was limited to be the same as human participants. See Appendix M for more information on evaluation and scoring.

H Human evaluation

H.1 Selection and compensation

Thirty participants were drawn from a pool of volunteers who indicated interest and an ability to commit 10 hours to the study during a week-long time-frame. Participants were asked to sign a participant consent form that included information regarding the purpose, procedure, risks and discomforts, potential benefits, costs, payment, confidentiality, and the subject's rights during and after the study (see AdultConsentForm in the supplemental materials).

Table 6: List of 30 generated evaluation worlds for which there were initially no successful playthroughs. After inspection and additional playthroughs, 10 worlds were replaced in the evaluation set (see comments column), although only 1 was actually unsolvable after fixing bugs.

Task	Seed	Difficulty	Comment
`eat`	542140	0.5	Replaced (due to bug, now fixed).
`eat`	542145	0.5	Replaced (due to bug, now fixed).
`avoid`	542771	0.85	
`avoid`	542787	1	
`descend`	542387	1	
`fight`	542731	1	Replaced (unlucky world, impossible).
`fight`	542694	1	
`find`	543024	0.92	
`find`	543031	1	Replaced (very difficult, no successful plays).
`find`	543033	1	Replaced (very difficult, no successful plays).
`find`	543034	1	Replaced (very difficult, no successful plays).
`find`	543017	0.74	
`gather`	543125	0.95	
`gather`	543127	1	
`gather`	543131	1	
`gather`	543132	1	
`gather`	543134	1	
`gather`	543137	1	Replaced (very difficult, no successful plays).
`jump`	542276	0.97	
`throw`	542612	0.62	Replaced (due to bug, now fixed).
`navigate`	542971	0.85	
`navigate`	542973	0.9	Replaced (very difficult, no successful plays).
`navigate`	542979	1	
`navigate`	542984	1	
`survive`	543077	1	
`survive`	543080	1	
`survive`	543086	1	
`bridge`	542527	1	
`carry`	542936	1	Replaced (due to bug, now fixed).

Risks and discomforts included cybersickness and short-term effects following VR use. The participants were also warned that there is a small chance that their identities could be inferred from their anonymized motion data. Participants were given an option to either be thanked or to remain entirely anonymous. All personally identifying information (i.e. names) has been removed from the dataset of human motion data (replaced with a random unique identifier).

Participants received payment in one of two forms:

1. Any participant who did not already own an Oculus Quest 2 headset received a new one along with a supplemental battery pack, and any hours spent over the required 10 hours were reimbursed at $30/hour.

2. Any participant who already had an Oculus headset received a supplemental battery pack and was reimbursed at $30/hour for the full 10 hours and any time they spent over that.

The rate of $30/hour was set to be fair to all participants regardless of whether they had an Oculus headset to start: an Oculus Quest 2 headset costs $300, which is commensurate with a $30/hour reimbursement over a 10-hour study period. Participants were also offered prizes of $200, $100, and $50 for the top three performers to incentivize focus and maximum effort during the evaluation worlds, since the gameplay can become somewhat monotonous. These amounts were selected as a balance between incentivizing focus while avoiding causing any undue stress to participants. If a participant didn't finish the required 10 hours, they were given the option to send back their headset (if applicable) and get reimbursed for the time spent or pay us back for the difference in hours.

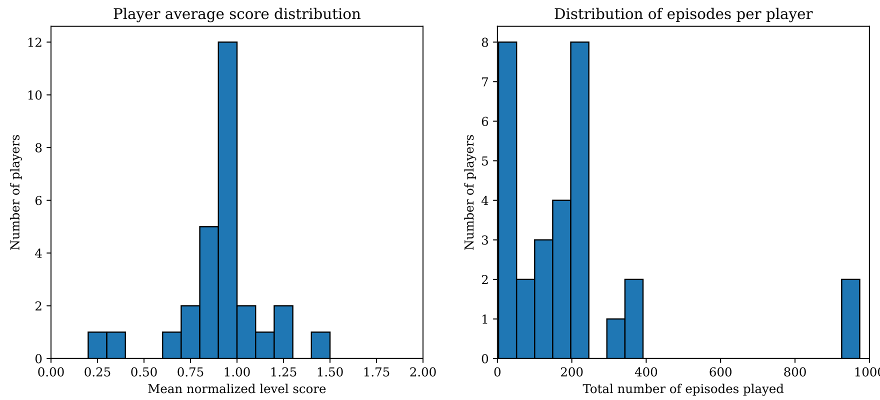

Figure 11: **Left**: Distribution of aggregated scores for each player. **Right**: Distribution of number of episodes played. Two users completed almost every evaluation level, while most completed around 200 levels.

In total, approximately $12,000 was spent on participant compensation between Oculus headsets, supplemental battery packs, and cash prizes/reimbursements.

H.2 Instruction and feedback

For the first 3 hours of the study, participants were asked to set up their Oculus headsets by installing our Avalon APK. They received a document with setup instructions as well as information about basic game mechanics such as the game controls and the tools and animals they would encounter in the environment (also included in the supplemental materials, see "Oculus Setup Practice Instructions"). Participants were then asked to complete at least 2 practice worlds for each of the 20 tasks. This data was not included in the evaluation set, and is not included in the recorded human data.

For the next 7 hours of the study, participants were asked to complete a series of evaluation worlds chosen at random from the pool of 1,000 evaluation worlds. No participant ever saw the same level twice. Two participants played almost 1,000 levels, and most players completed around 200 levels (Figure 11), for a total of 6,145 played episodes. Participant motion data was captured during these evaluation worlds. Participants were allowed to "reset" if they ended up failing or getting stuck in a world. These resets were not counted towards the 5 playthroughs per world. This ability to reset was added to reduce stress on the human participants (whether due to perceived failure or real-life interruptions).

In a feedback survey after the study, participants reported an average overall satisfaction with the study of 4.57 out of 5 and 30 out of 32 of the participants said they would like to be considered for future studies.

H.3 Analysis of performance

Most levels were quite simple, leading to final episode scores clustered tightly around one (Figure 12). Aggregated player performance was quite similar as well (Figure 11), with low performing users playing on just a few high difficulty levels.

In 962 episodes, the player either died during the episode or reached the time limit without eating food, getting a final score of zero. The remaining 5,183 episodes had a score greater than zero, a condition we refer to as a "success." The distribution of these successes across tasks is shown in Figure 13. Human performance is fairly consistent across most tasks, with some of the compositional tasks presenting more of a challenge.

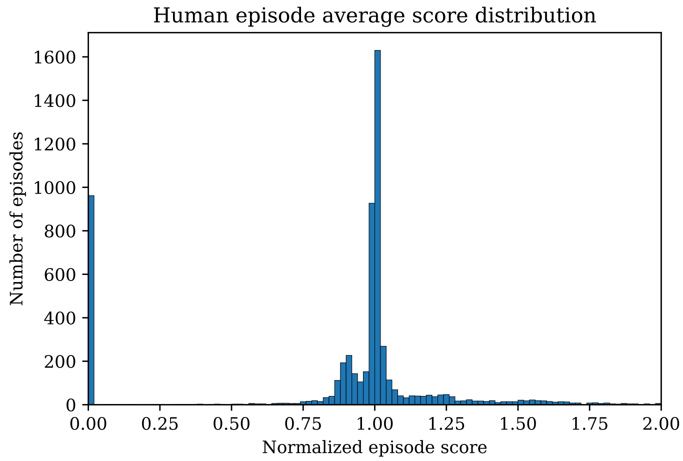

Figure 12: Distribution of scores for each episode. Most players were able to eat all the available food in about the same time for most episodes, leading to a narrow distribution of scores.

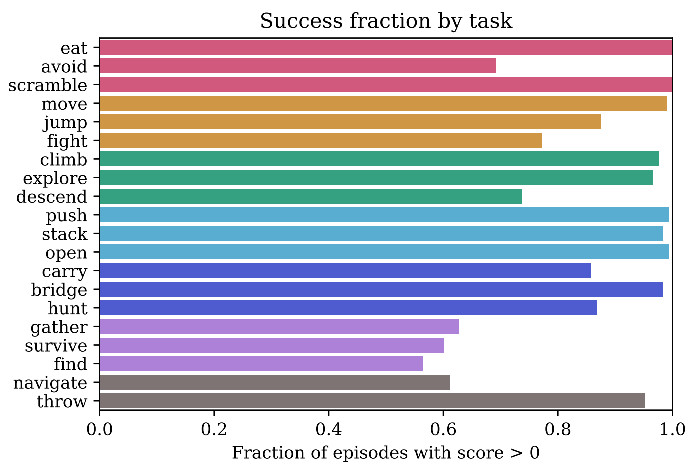

Figure 13: Success rate of players on each task. Success is defined as completing an episode with a score greater than 0.

I Curriculum

All task generators are parameterized, at the highest level, by the task id and the "difficulty" d_t (a float in the range [0, 1]). Each environment generation worker process maintains a mapping from task id to the current maximum difficulty d_t (all initialized at 0), and generates a level from a uniform distribution $U[0, d_t]$. When an agent succeeds (or fails) at a generated environment, the d_t is increased (or decreased) by H_t, a hyper-parameter that controls how quickly the task curriculum adjusts. An agent succeeds at its environment if it eats all of the available food before the episode timer is finished.

J Hyperparameters

See table 7 for hyperparameters used for training PPO, IMPALA, and Dreamer.

Table 7: Hyperparameters used for training

Parameter	PPO	IMPALA	Dreamer
Optimizer	Adam	RMSProp	Adam
Learning rate	2.5e-4	1.5e-4	model=1e-4, value=1e-4, actor=1e-5
Other optimizer params	ϵ =1e-5	$\alpha = 0.95$, $\epsilon = 0.005$	ϵ =1e-5, weight decay=1e-6
Grad norm clipping	0.5	200	100
Baseline cost	1	2	1
Entropy cost	1.5e-4	5e-3	2e-3
Discount factor	0.99	0.998	.99
GAE Lambda	0.83	-	.95
IS clip range	0.03	-	-
Batch size	16	32	116
Unroll length	200	100	30
Task difficulty update (H_t)	3e-4	5e-3	6e-4
Movement penalty coefficient (C_T)	1e-8	1e-8	1e-8
Frame penalty (P_{frame})	1e-4	1e-4	1e-4
Actor gradients	-	-	REINFORCE

K Training compute

Our training took place in containers, with 8 containers to a machine. One GPU was assigned per container. The whole machine provides 8x Nvidia GeForce RTX 3090 (with 24GB RAM), and is powered by 2x AMD EPYC 7313 CPUs (with 16 cores/32 threads each) with 256GB RAM total. Training time was approximately 30 hours for a single run.

L Results for other training runs

Table 8 shows the optimality gap scores corresponding to our main experimental runs (whereas Table 1 reports average scores). The same trends are apparent in both tables: the compositional tasks are much more difficult than most of the basic tasks, and the gap between human scores and even the best-performing networks are emphasized here.

Tables 9 and 10 show the average scores and optimality gap scores respectively for IMPALA under each of the four training conditions, averaged over three training runs with different seeds. A few salient conclusions may be drawn from this data. First, comparing MT-TB with the single task baselines ST-B, one can see that the similarity of environments has enabled significant transfer learning. Several tasks like `jump` and `climb` are too challenging to learn on their own, but may be achieved by training on other tasks. While the performance of basic tasks such `eat` and `move` somewhat lower in the multi-task setting than the single task setting, the difference is small enough that it is more likely due to fewer exposures to each task in the multi task case, rather than being a case of catastrophic forgetting.

Second, we note that training on only the basic tasks seems to be the most effective use of training time. While an agent training on all tasks or even just the compositional tasks does learn a broad variety of basic skills (for example, despite never seeing these levels, MT-TC has significant performance on `eat` and `open`), these agents do not outperform an agent trained on just the basic tasks when measured on either the basic or compositional aggregate scores. We suspect that these tasks may be too difficult for the agent to make much progress at this stage in training.

In Figure 14, we show a performance profile by task for each network (IMPALA and PPO) on the MT-TB training setting (for which results are shown in Table 1 in the main paper). For many tasks, humans had quite consistent performance, due to the simplicity of the tasks and sparse reward. This leads to the relatively flat horizontal sections of the graph extending from 0 to 1, where the y-axis indicates the fraction of these levels in which the agent got the single available food. For many tasks the scores drop off as they approach $\tau = 1$, as the tasks get harder and there is some more nuance to performance beyond the binary "found food or not" (e.g., from tasks that take longer and or where there is some risk to humans of dying). From the tasks with any amount of super-human performance,

Task	PPO	Dreamer	IMPALA		
	50m steps With curr.	50m steps With curr.	50m steps With curr.	500m steps With curr.	50m steps No curr.
eat	0.402 ± 0.067	0.445 ± 0.065	0.357 ± 0.062	0.375 ± 0.097	0.999 ± 0.001
move	0.741 ± 0.062	0.721 ± 0.071	0.640 ± 0.062	0.603 ± 0.076	1.000 ± 0.000
jump	0.798 ± 0.050	0.806 ± 0.058	0.709 ± 0.056	0.725 ± 0.072	1.000 ± 0.000
climb	0.816 ± 0.043	0.809 ± 0.051	0.782 ± 0.049	0.675 ± 0.074	1.000 ± 0.000
descend	0.834 ± 0.043	0.751 ± 0.058	0.843 ± 0.044	0.784 ± 0.059	1.000 ± 0.000
scramble	0.719 ± 0.054	0.627 ± 0.058	0.573 ± 0.062	0.444 ± 0.070	1.000 ± 0.000
stack	0.921 ± 0.036	0.890 ± 0.043	0.874 ± 0.036	0.891 ± 0.055	1.000 ± 0.000
bridge	0.956 ± 0.027	0.907 ± 0.045	0.924 ± 0.029	0.912 ± 0.049	1.000 ± 0.000
push	0.899 ± 0.039	0.874 ± 0.053	0.867 ± 0.043	0.872 ± 0.054	1.000 ± 0.000
throw	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
hunt	0.961 ± 0.024	0.942 ± 0.028	0.933 ± 0.029	0.880 ± 0.051	1.000 ± 0.000
fight	0.831 ± 0.052	0.746 ± 0.076	0.789 ± 0.052	0.724 ± 0.075	1.000 ± 0.000
avoid	0.697 ± 0.170	0.634 ± 0.118	0.551 ± 0.159	0.510 ± 0.102	1.000 ± 0.000
explore	0.827 ± 0.047	0.831 ± 0.048	0.803 ± 0.048	0.760 ± 0.069	1.000 ± 0.000
open	0.948 ± 0.024	0.892 ± 0.041	0.907 ± 0.034	0.899 ± 0.046	1.000 ± 0.000
carry	0.932 ± 0.031	0.940 ± 0.028	0.913 ± 0.032	0.891 ± 0.057	1.000 ± 0.000
navigate	1.000 ± 0.000	1.000 ± 0.000	0.988 ± 0.010	0.963 ± 0.032	1.000 ± 0.000
find	0.998 ± 0.003	1.000 ± 0.000	0.987 ± 0.014	0.987 ± 0.017	1.000 ± 0.000
survive	0.957 ± 0.013	0.956 ± 0.014	0.950 ± 0.015	0.915 ± 0.028	1.000 ± 0.000
gather	0.979 ± 0.010	0.980 ± 0.012	0.970 ± 0.010	0.968 ± 0.014	1.000 ± 0.000
all basic	0.830 ± 0.017	0.801 ± 0.016	0.779 ± 0.016	0.747 ± 0.019	1.000 ± 0.000
all comp.	0.983 ± 0.004	0.984 ± 0.005	0.974 ± 0.007	0.958 ± 0.013	1.000 ± 0.000
all	0.861 ± 0.014	0.838 ± 0.012	0.818 ± 0.013	0.789 ± 0.015	1.000 ± 0.000

Table 8: Optimality gap results for Table 1. Lower scores are better.

these seem likely to be cases where the episode was extremely short and the agent moved faster and oriented itself more quickly than the average human.

From the graphs in Figure 14, we can also see that most of the advantage for IMPALA over PPO comes from better performance on the basic tasks. Both networks perform quite poorly on the compositional tasks, mostly only succeeding at `survive`, which often has plentiful food, making nonzero scores easier to achieve. One can get a sense for the relative difficulties of the tasks. Some, like `open` and `avoid`, seem surprisingly easy, though this is likely due to relatively forgiving levels at low difficulties rather than due to highly effective agents.

We visualize the generalization performance of single task training in Figure 15. One can see that training on other tasks readily transfers to `eat` or `move`, but that the agents are not able to make much progress on either the trained task or the other tasks.

We finally show learning curves for IMPALA using MT-TB in Figure 16. One can see that performance on some tasks continues to improve throughout training, and so the agent may continue to benefit from more steps in the environment.

M Scoring

For both human players and RL agents, the score of each run is given by

$$S = \max\left(0, E_f - E_0 + T \times P_{\text{frame}} \cdot\right), \tag{6}$$

where E_0 is the starting energy level, E_f is the final energy level (when the last food is consumed, the agent dies or the episode times out), T is the total number of episode frames remaining until timeout, and P_{frame} is a constant penalty per frame. For effective comparison with humans, we do not impose energy penalties for movement during evaluation (i.e. the C_{move} of Appendix B is zero),

Task	MT-TB	MT-TA	MT-TC	ST-B
eat	0.716 ± 0.062	0.657 ± 0.070	0.236 ± 0.051	0.751 ± 0.060
move	0.399 ± 0.062	0.370 ± 0.064	0.018 ± 0.014	0.397 ± 0.059
jump	0.309 ± 0.056	0.288 ± 0.056	0.008 ± 0.011	0.112 ± 0.033
climb	0.229 ± 0.047	0.144 ± 0.043	0.000 ± 0.000	0.000 ± 0.000
descend	0.173 ± 0.045	0.128 ± 0.037	0.002 ± 0.003	0.000 ± 0.000
scramble	0.467 ± 0.064	0.252 ± 0.053	0.006 ± 0.008	0.630 ± 0.050
stack	0.130 ± 0.038	0.115 ± 0.037	0.000 ± 0.000	0.000 ± 0.000
bridge	0.076 ± 0.028	0.044 ± 0.026	0.000 ± 0.000	0.000 ± 0.000
push	0.150 ± 0.045	0.092 ± 0.035	0.000 ± 0.000	0.000 ± 0.000
throw	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
hunt	0.071 ± 0.029	0.030 ± 0.020	0.003 ± 0.005	0.250 ± 0.051
fight	0.235 ± 0.053	0.212 ± 0.050	0.002 ± 0.004	0.000 ± 0.000
avoid	0.603 ± 0.144	0.326 ± 0.055	0.000 ± 0.000	0.000 ± 0.000
explore	0.213 ± 0.050	0.190 ± 0.044	0.044 ± 0.025	0.247 ± 0.054
open	0.097 ± 0.034	0.063 ± 0.025	0.009 ± 0.010	0.083 ± 0.029
carry	0.089 ± 0.033	0.090 ± 0.037	0.005 ± 0.006	0.000 ± 0.000
navigate	0.012 ± 0.011	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
find	0.015 ± 0.015	0.008 ± 0.011	0.000 ± 0.000	0.000 ± 0.000
survive	0.050 ± 0.014	0.037 ± 0.011	0.010 ± 0.005	0.031 ± 0.013
gather	0.030 ± 0.010	0.019 ± 0.008	0.000 ± 0.000	0.000 ± 0.000
all basic	0.247 ± 0.017	0.188 ± 0.012	0.021 ± 0.004	-
all comp.	0.027 ± 0.006	0.016 ± 0.005	0.002 ± 0.001	-
all	0.203 ± 0.012	0.153 ± 0.010	0.017 ± 0.003	-

Table 9: Mean-human normalized performance for RL agents trained with protocols defined in Section 5.2. MT-TB is trained on the 16 simple tasks, MT-TA is trained on all 20 tasks, and MT-TC is trained on the four compositional tasks. All agents are trained for 50M steps using IMPALA. ST-B results are trained only on a single task for 50M steps, and are evaluated only on that same task.

Task	MT-TB	MT-TA	MT-TC	ST-B
eat	0.357 ± 0.062	0.451 ± 0.070	0.787 ± 0.051	0.316 ± 0.060
move	0.640 ± 0.062	0.673 ± 0.064	0.982 ± 0.014	0.616 ± 0.059
jump	0.709 ± 0.056	0.736 ± 0.056	0.994 ± 0.011	0.889 ± 0.033
climb	0.782 ± 0.047	0.867 ± 0.043	1.000 ± 0.000	1.000 ± 0.000
descend	0.843 ± 0.045	0.883 ± 0.037	0.998 ± 0.003	1.000 ± 0.000
scramble	0.573 ± 0.064	0.767 ± 0.053	0.994 ± 0.008	0.386 ± 0.050
stack	0.874 ± 0.038	0.896 ± 0.037	1.000 ± 0.000	1.000 ± 0.000
bridge	0.924 ± 0.028	0.961 ± 0.026	1.000 ± 0.000	1.000 ± 0.000
push	0.867 ± 0.045	0.915 ± 0.035	1.000 ± 0.000	1.000 ± 0.000
throw	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
hunt	0.933 ± 0.029	0.974 ± 0.020	0.997 ± 0.005	0.767 ± 0.051
fight	0.789 ± 0.053	0.812 ± 0.050	0.998 ± 0.004	1.000 ± 0.000
avoid	0.551 ± 0.144	0.704 ± 0.055	1.000 ± 0.000	1.000 ± 0.000
explore	0.803 ± 0.050	0.822 ± 0.044	0.961 ± 0.025	0.773 ± 0.054
open	0.907 ± 0.034	0.938 ± 0.025	0.991 ± 0.010	0.918 ± 0.029
carry	0.913 ± 0.033	0.921 ± 0.037	0.995 ± 0.006	1.000 ± 0.000
navigate	0.988 ± 0.011	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
find	0.987 ± 0.015	0.994 ± 0.011	1.000 ± 0.000	1.000 ± 0.000
survive	0.950 ± 0.014	0.963 ± 0.011	0.990 ± 0.005	0.969 ± 0.013
gather	0.970 ± 0.010	0.981 ± 0.008	1.000 ± 0.000	1.000 ± 0.000
all basic	0.779 ± 0.017	0.833 ± 0.012	0.981 ± 0.004	-
all comp.	0.974 ± 0.006	0.984 ± 0.005	0.998 ± 0.001	-
all	0.818 ± 0.012	0.863 ± 0.010	0.984 ± 0.003	-

Table 10: Optimality gap results for Table 9. Lower scores are better.

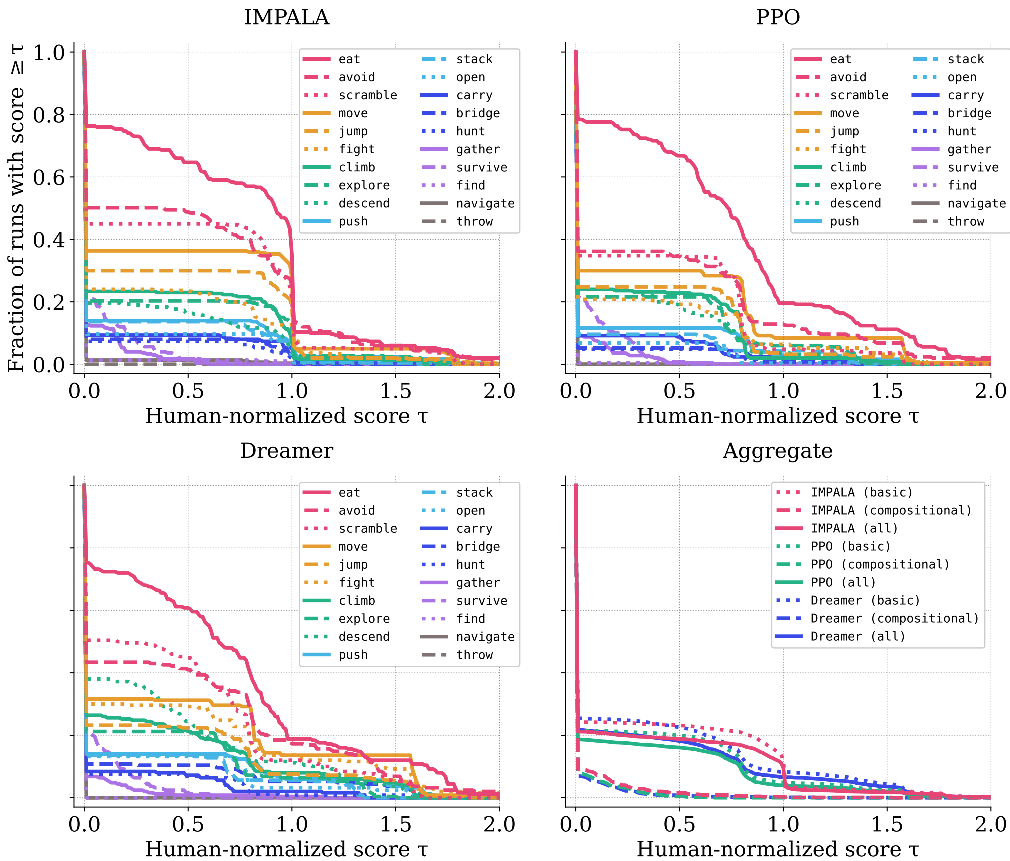

Figure 14: Results for RL agents trained on all 16 basic tasks and evaluated on each of Avalon's 20 tasks (MT-TB). Scores shown are average scores, normalized against mean human performance. Top row and bottom left: scores for IMPALA, PPO and Dreamer on each task. Bottom right: aggregate scores averaged over the 16 basic tasks, the four compositional tasks, and all 20 tasks.

opting instead for the constant per-frame energy penalty. We use the same P_{frame} during training of 1e-4 when scoring runs and for each task, E_0 is always set to 1.0.

For most tasks, success is simply whether or not the player ate all of the food in the level (and there is usually just a single piece). For tasks which always have multiple pieces of food, namely `gather` and `survive`, a run is considered successful if the player ate all food, or ate at least one food and ended the task with more health than they started with.

All reported scores are normalized post hoc such that the average human performance on a given task is 1.0 and the performance of a random agent is 0.0.

N Simulator Performance

One of the major contributions of Avalon is the simulator, which was specifically designed to be high-performance. Table 11 gives the performance for multiple processes on a single 2080Ti GPU, and Table 12 gives the performance for a single process for the same configuration. Each table gives a breakdown of performance, in terms of simulated steps per second, by both the size and complexity of the worlds being simulated, as well as by the graphical options that were enabled. The worlds range in size from Small (64m × 64m) to Huge (almost a square half-kilometer). See Figure 17 for images of the levels that were used for profiling). The graphical options include which renderer to use (GLES2 vs GLES3) and which options (fancy vs basic). For the "basic" condition, shadows were disabled, lighting was done per-vertex, and all settings were configured for the Godot defaults for mobile rendering. For the "fancy" condition, the opposite was true–lighting was per pixel, shadows

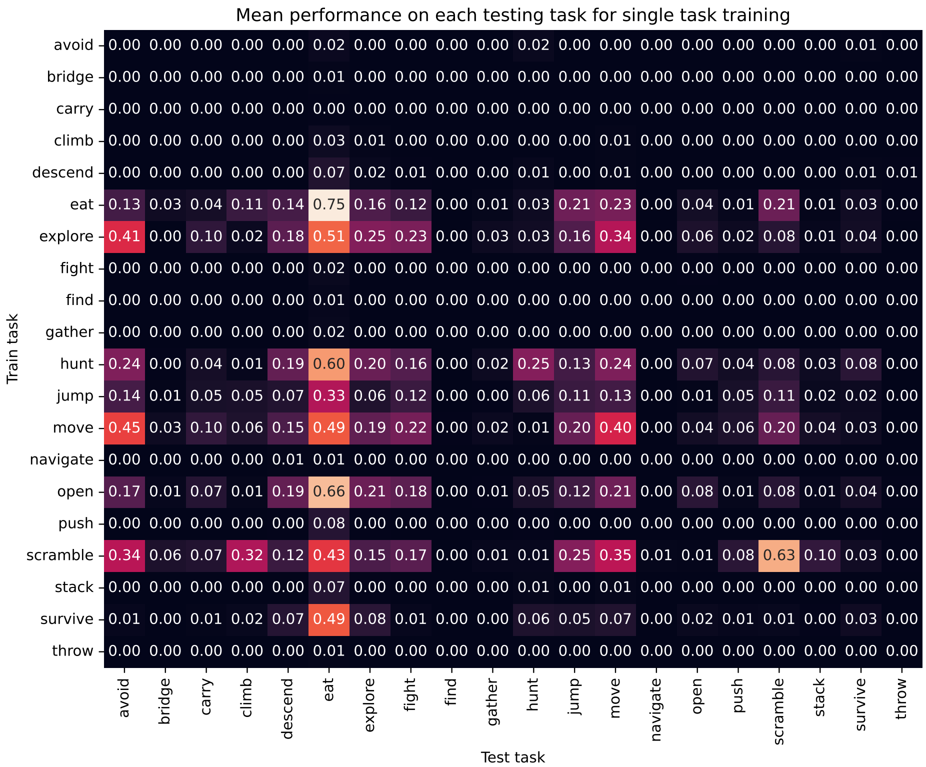

Figure 15: Mean human-normalized performance for IMPALA trained for 50M environment steps on a single task and tested on another task.

were enabled, and all Godot defaults for desktop computers were used. Additionally, GLES3 has MSAA 4x enabled for both conditions, since we use this for our agent because it makes it easier to detect small objects from farther away without incurring much of a performance penalty.

Our performance numbers are meant to get as close as possible to the performance numbers reported for Habitat. We used a 2080Ti GPU in order to make our numbers more directly comparable. We also show results on a 3090 GPU (which was used for all of our other experiments) in Tables 13 and 14. The 2080Ti was paired with a Intel i9-10980XE CPU, while the 3090 had an AMD Ryzen 9 5950X. We also rendered at 128 x 128, a higher resolution than used in the rest of our paper, in order to be more directly comparable. Like Habitat, our benchmarking setting consisted of reading actions from a pipe and writing outputs to a file (no networks were being trained or run on the same GPU). Unlike Habitat, our simulator works in a straightforward loop of reading actions, stepping physics and gameplay logic, and rendering the resulting frame, while their fastest numbers came from an interleaved setting that delayed observation of the effect of an action by an extra frame. We also do not require any frame pointer passing or other complex integrations, making it easier to integrate our simulator into any network training setup, including distributed settings.

It should be noted that this performance is close to the maximum possible, given the hardware and software available today–a non-trivial amount of the time spent in rendering our Small world is spent simply clearing the screen via the glClear call (a necessary component for rendering). The single-process numbers represent the maximum possible speed-up over real-time for a single agent in our simulator (e.g., since we run the simulation at 10 steps per simulated second and a single process on the 3090 can do 4,114 steps per second, we can run an agent at up to 411.4x real time). It should be noted that this speed is not reflective of performance real world–practical training and evaluation is dominated by the network forward and backward passes. The multi-process numbers are more reflective of training throughput and efficiency per-GPU.

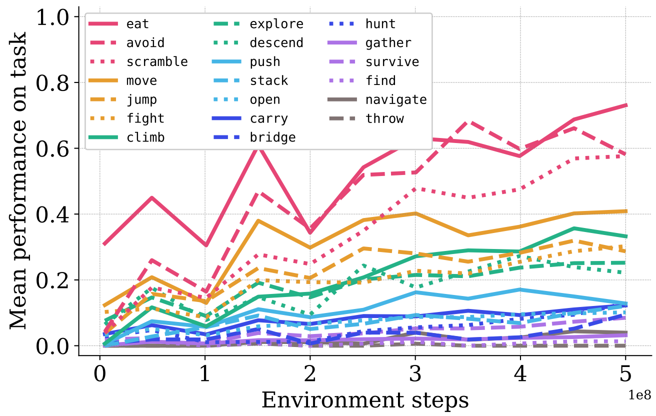

Figure 16: Learning curves for mean human normalized performance on each task, using IMPALA and MT-TB for 500m training runs with the curriculum. Each point is the average over 3 different training runs with different seeds.

	Small	**Medium**	**Large**	**Huge**
GLES2 (basic)	10,157	9,854	5,886	1,218
GLES2 (fancy)	8,452	7,811	3,545	562
GLES3 (basic)	6,853	6,706	5,321	1,757
GLES3 (fancy)	5,901	5,685	3,627	848

Table 11: Multi-process performance (steps per second) on a single 2080TI GPU on various sizes of worlds (from Small to Huge).

	Small	**Medium**	**Large**	**Huge**
GLES2 (basic)	2,880	2,747	2,144	623
GLES2 (fancy)	2,574	2,409	1,583	415
GLES3 (basic)	2,557	2,385	2,066	735
GLES3 (fancy)	2,203	2,033	1,506	519

Table 12: Single-process performance (steps per second) on a single 2080TI GPU on various sizes of worlds (from Small to Huge).

	Small	**Medium**	**Large**	**Huge**
GLES2 (basic)	11,730	10,963	5,999	1,120
GLES2 (fancy)	9,445	8,614	3,611	529
GLES3 (basic)	7,393	7,290	6,025	2,175
GLES3 (fancy)	6,336	6,182	4,056	1,081

Table 13: Multi-process performance (steps per second) on a single 3090 GPU on various sizes of worlds (from Small to Huge).

	Small	**Medium**	**Large**	**Huge**
GLES2 (basic)	4,114	3,953	3,349	1,198
GLES2 (fancy)	3,624	3,511	2,450	570
GLES3 (basic)	3,679	3,529	3,181	1,381
GLES3 (fancy)	3,216	3,036	2,278	795

Table 14: Single-process performance (steps per second) on a single 3090 GPU on various sizes of worlds (from Small to Huge).

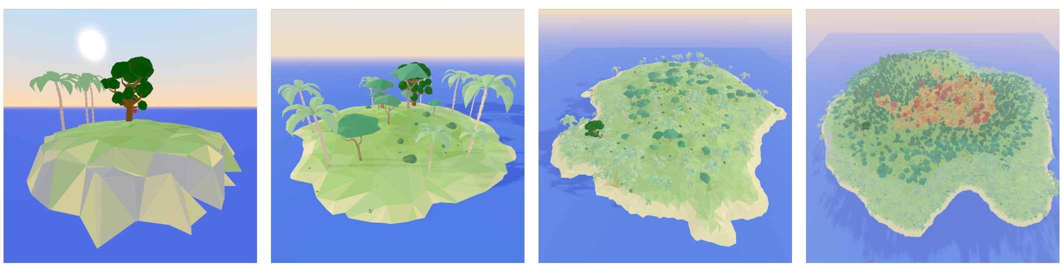

Figure 17: Each of the worlds used for profiling and their sizes, from left to right: Small (32m $\times$ 32m), Medium (64m $\times$ 64m), Large (220m $\times$ 220m), and Huge (440m $\times$ 440m)