EVAAA: A Virtual Environment Platform for Essential Variables in Autonomous and Adaptive Agents

Sungwoo Lee[1,2,8*] **Jungmin Lee**[1,2,8*] **Sohee Kim**[1,3,8] **Hyebhin Yoon**[9] **Shinwon Park**[5]

Junhyeok Park[6] **Jaehyuk Bae**[7] **Seok-Jun Hong**[1,2,3,4,8†] **Choong-Wan Woo**[1,2,3,8†]

[1]Center for Neuroscience Imaging Research, Institute for Basic Science (IBS), South Korea, [2]Department of Biomedical Engineering, Sungkyunkwan University, South Korea, [3]Department of Intelligent Precision Healthcare Convergence, Sungkyunkwan University, South Korea, [4]Center for the Developing Brain, Child Mind Institute, NY, USA, [5]Autism Center, Child Mind Institute, NY, USA, [6]Graduate School of Metaverse, KAIST, South Korea, [7]Department of Brain and Cognitive Sciences, KAIST, South Korea, [8]Life-inspired Neural Network for Prediction and Optimization Research Group, South Korea, [9]Neurocore Co., Ltd., Seoul, South Korea

Content Overview

- **Appendix A** describes the Unity-based interface implemented in EVAAA, including an environment setup, prefab structures, and object instantiation.
- **Appendix B** provides a comprehensive introduction to Essential Variables (EVs), including their design, dynamics, and role in internal state regulation.
- **Appendix C** explains the implementation of the reward system and its connection to the balance of internal states.
- **Appendix D** details the agent's observation system, covering both interoceptive inputs (EVs) and exteroceptive sensors (i.e., vision, olfaction, thermoception, and collision detection).
- **Appendix E** outlines the modular configuration to generate EVAAA environments, along with the instructions for environment customization.
- **Appendix F** presents the structure and progression of naturalistic training environments.
- **Appendix G** describes the design of unseen experimental testbeds for evaluation.
- **Appendix H** details the computational resources, evaluation metrics, and human experiment.
- **Appendix I** provides analyses of agent behavior across training and test environments, including emergent behavioral patterns.
- **Appendix J** discusses broader impacts, including potential applications and ethical considerations of EVAAA.

All code and data are publicly available at: `https://github.com/cocoanlab/evaaa`

A Unity Interface

A.1 Prefabs

Environmental elements such as terrain, resources, obstacles, and predators are implemented as reusable and configurable Unity prefabs. These assets are sourced from the Unity Asset Store, and we

[*] Equal contribution: sungwoo320@gmail.com, jungmin.lee@g.skku.edu

[†] Corresponding authors: hongseokjun@skku.edu, waniwoo@skku.edu

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Track on Datasets and Benchmarks.

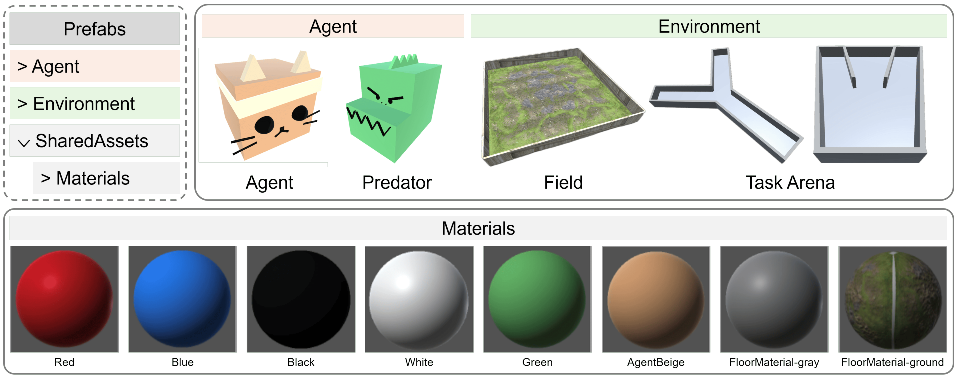


Figure 5: **Overview of prefab categories and object types used in EVAAA.** Prefabs are grouped into Agents, Environment, and Materials. Each category includes reusable components for constructing and customizing interactive scenes: Agents (main agent and predators), Environment (terrain and containers), and Materials (varied textures and colors for visual distinction).

selected only freely available assets typically used in both research and applied projects. This modular system enables rapid prototyping, task generation, condition randomization, and reproducible scene setup. Prefabs can be customized through the Unity Editor or programmatically at runtime, and reused across scenes without manual rebuilding. Figure 5 provides a visual overview of the following available prefab types:

- **Agent** prefabs include both the main agent and predators.
- **Environment** prefabs define the simulation area based on terrain fields.
- **Material** prefabs offer color customization for visual or functional distinction and can be applied to any object type. Available colors include black, red, light green, yellow, orange, and more.

B Essential Variables

Essential Variables (EVs) represent the agent's internal state and are central to the design of EVAAA. Four scalar EVs are defined: satiation, hydration, body temperature, and tissue integrity (the level of damage). These variables evolve over time based on the agent's actions and environmental conditions, and are continuously updated during simulation. EVs influence the agent's observation vector, reward computation, and termination conditions. Over time, even in the absence of activity, levels of satiation and hydration gradually decline, reflecting natural metabolic processes that lead to hunger and dehydration. To continue an episode, all EVs must remain within their viable ranges; deviations lead to negative rewards or early termination. This framework anchors the learning process in survival-relevant dynamics without hard-coded objectives.

C Reward system

The reward system in EVAAA is driven by the agent's internal state (EVs), rather than relying on task-specific external objectives. At each timestep, the agent receives a reward signal reflecting how closely its internal state matches predefined homeostatic setpoints. The reward r_t is calculated as the negative normalized Euclidean distance between the current EVs and their target values:

$$r_t = -\sqrt{\sum_{i=1}^{4} \left(\frac{EV_i(t) - \mu_i}{\sigma_i} \right)^2}$$

Where:

- $EV_i(t)$ is the value of the i-th essential variable at time t.

- μ_i is the setpoint (target value) (e.g., 0 for satiation/hydration/temperature/damage)
- σ_i is the allowed deviation, derived from researcher-defined valid ranges for each EV

The negative reward increases with greater deviations from the setpoints, encouraging the agent to get back to the balanced internal states (i.e., homeostasis). For instance, if the agent becomes dehydrated, overheated, or injured, it receives increasingly negative rewards.

D EVAAA Observation details

D.1 Interoception

I. EV update dynamics Each EV is modeled as a scalar internal state $\mathrm{EV}_i \in \mathbb{R}$ and updated at every timestep using a linear function: $\mathrm{EV}_i(t+1) = \mathrm{EV}_i(t) + \beta_0 \cdot D_i + \sum_{j=1}^{3} \beta_j \cdot S_{ij}(t) + \beta_{i,e} \cdot \Delta_i(t)$

Term Definitions:

- $\beta_0 \cdot D_i$: constant passive decay scaled to the EV's range
- $S_{ij}(t)$: EVs input
- β_j: coefficients for inter-EV influences (e.g., foodCoefficient.change$_j$)
- $\beta_{i,e} \cdot \Delta_i(t)$: discrete event-driven changes from resource interaction (e.g., eating food, drinking water)

Although the current setting has no interdependency among EVs, non-linear interactions among EVs are implemented as one of the potential configurations for future use.

II. EV ranges and setpoints Each EV operates within a researcher-defined viable range and is anchored to a homeostatic set point representing the agent's optimal internal state.

- Satiation, hydration, and temperature use symmetric ranges from -15 to 15, with a setpoint at 0. Deviations in either direction indicate homeostatic imbalance (e.g., overheating or starvation).
- Damage operates on a 0 to 100 scale, with a setpoint at 0 representing no damage. Higher values indicate increased damage resulting from environmental hazards.

An episode is terminated if any EV falls outside its defined range.

D.2 Vision

At each timestep, the agent receives egocentric visual input rendered from a first-person monocular view using Unity ML-agents' built-in CameraSensor function. Observations are encoded as 64x64 RGB images by default, represented in a 3D-tensor format (height, width, channels). Resolution and color depth are configurable.

D.3 Olfaction

Each time a resource object is instantiated—whether at the beginning of an episode or after being consumed and respawned—it is assigned a fixed 10-dimensional feature vector that encodes its identity (e.g., food vs water), with additional variation introduced through uniform random sampling (**Fig. 6**). These vectors remain constant throughout the episode. The agent uses a spherical olfactory sensor based on Unity's Physics OverlapSphere to detect nearby resources within a configurable radius. For each detected object, its feature vector is divided by its distance to the agent. If multiple objects are detected (e.g., several food sources at distances d_1, d_2, d_3, and d_4), their distance-weighted vectors are summed to make a single 10-dimensional observation at each timestep. This process informs the agent of the existence of nearby resources without revealing their exact positions.

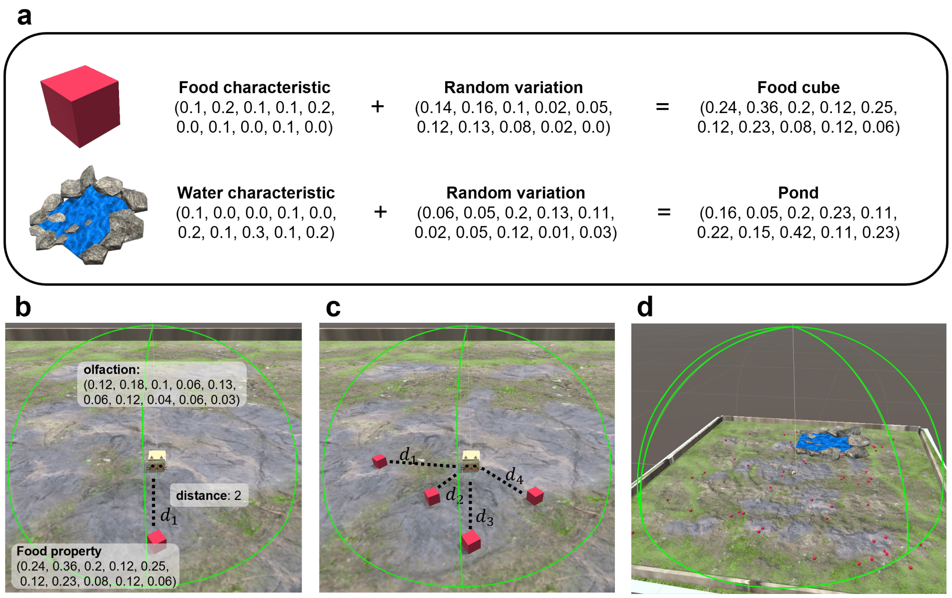


Figure 6: **Overview of the olfactory mechanism.** (a) Each resource object emits a unique 10-dimensional chemical feature vector. (b) The agent senses nearby objects using a spherical detection field. (c) Multiple resources are encoded through the distance-dependent aggregation of their feature vectors. (d) The olfactory radius can be extended to encompass larger areas and heterogeneous resource types.

D.4 Thermoception

The agent perceives ambient temperature through a 3x3 grid thermal sensor, with the agent located at the center (**Fig. 7**). These sensors sample values from an invisible two-dimensional 100x100 temperature grid that overlays the environment. Each cell encodes a scalar temperature value but is not directly observable. At each timestep, if a sensor intersects with a grid cell, it records the corresponding temperature value. The nine sensors together produce a 9-dimensional observation vector that captures the spatial temperature distribution around the agent. This enables detection of heat gradients and environmental temperature changes without relying on explicit spatial coordinates. Although the temperature field remains static during an episode, it can be reconfigured to simulate dynamic conditions such as day-night cycles or local heat sources (e.g., bonfires).

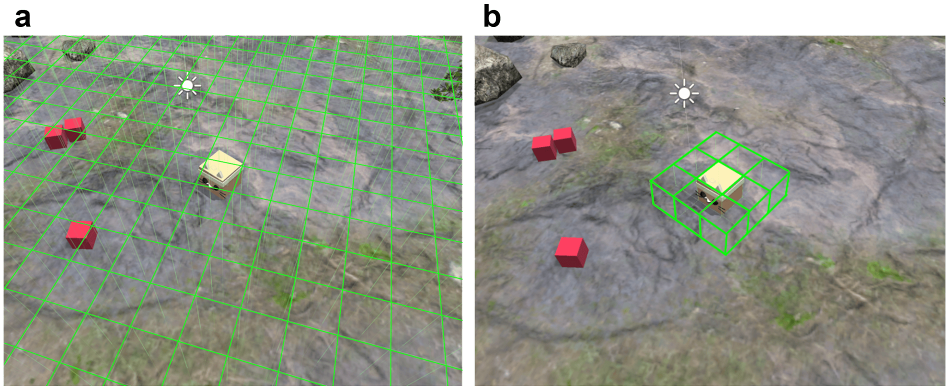


Figure 7: **Overview of the thermoception mechanism.** (a) A 100x100 invisible temperature grid is placed over the terrain. (b) The agent samples local temperatures via a 3×3 array of thermal sensors centered on its position. Each sensor maps to overlapping grid cells, producing a 9-D temperature observation at each timestep.

D.5 Collision Setting

The agent is equipped with a radial raycasting system that emits 100 evenly spaced rays across a full 360-degree field around its body (**Fig. 8**). These rays are divided into 10 angular segments, with each segment aggregating binary collision feedback. At every step, rays are cast outward from the agent's position and are checked for intersections with nearby objects, excluding the agent itself. If any ray within a segment collides with an obstacle or predator, the corresponding entry in a 10-dimensional 'collisionObservation' vector is set to a non-zero value. This setup allows the agent to detect nearby hazards and avoid collisions during navigation. Upon contact, the agent receives a scalar damage value proportional to the impulse magnitude of the collision. This value is capped at a predefined maximum to prevent extreme updates and then subtracted from the agent's internal damage EV.

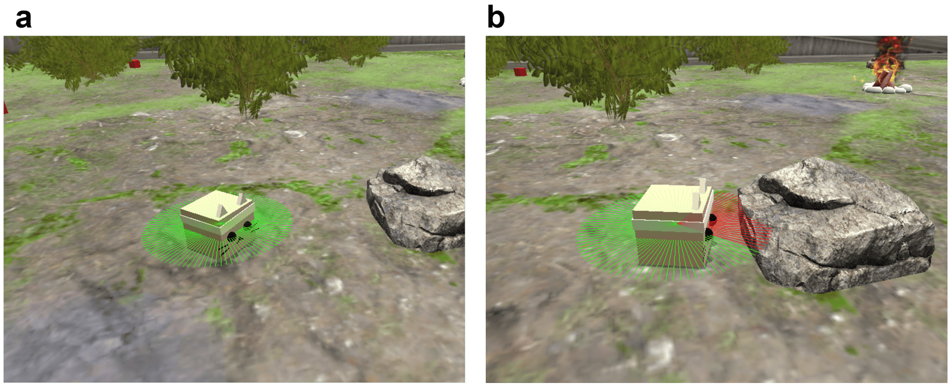


Figure 8: **Collision detection using radial raycasting.** (a) In obstacle-free conditions, rays (green) return no contact. (b) When rays intersect an object (e.g., a rock), contact is detected (red), updating the agent's 10-D collision observation vector.

E EVAAA implementation details

EVAAA provides a modular configuration system using structured JSON files, allowing researchers to flexibly adjust agent properties, environmental layout, resource generation, and interaction parameters. All configuration files are grouped under the Config directory and organized into reusable components, supporting both training and evaluation through a unified structure (**Fig. 9**). Core configuration categories include:

- E.1 **Agent config** defines action settings (i.e., movement speed, eating action), internal state initialization, and sensor modalities.
- E.2 **Court config** controls terrain size, boundary settings, and floor materials.
- E.3 **Resource config** specifies food/water generation, spawn intervals, and resource placement behaviors (e.g., random distribution, static positioning, respawn strategy).
- E.4 **ThermoGrid config** sets the resolution and values of the ambient temperature field.
- E.5 **Obstacle config** determines the type and placement of physical obstacles.
- E.6 **Landmark config** defines the valid boundary for predators to navigate.
- E.7 **Predator config** configures predator behavior, aggression levels, and spawn patterns.
- E.8 **Day/Night cycle config** manages time-of-day cycles and lighting conditions.

Each subsection below details the available parameters, default values, and example use cases for configuring EVAAA environments.

E.1 Agent config

The agent configuration defines how the agent moves, senses, and maintains internal homeostasis (**Fig. 10**). Each EV has a bounded range and an initial value, adjustable through parameters like

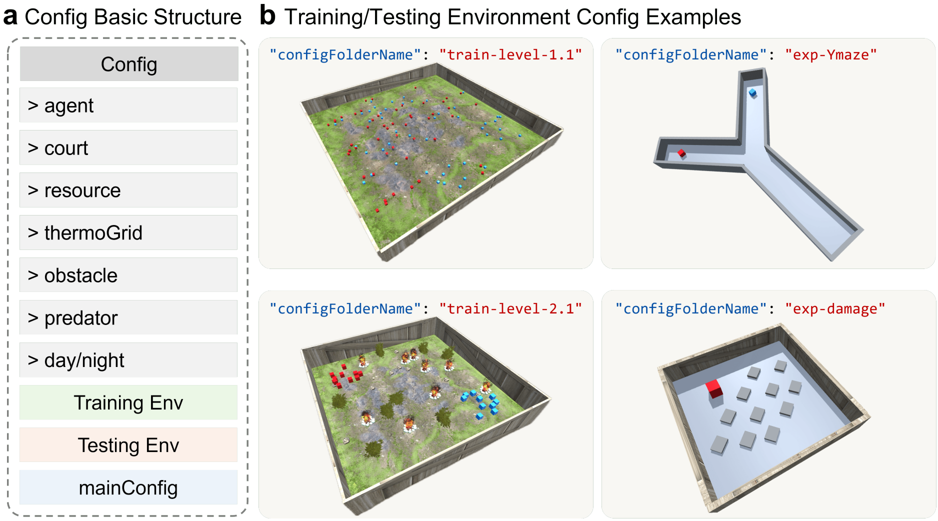

a Config Basic Structure
Config
> agent
> court
> resource
> thermoGrid
> obstacle
> predator
> day/night
Training Env
Testing Env
mainConfig
b Training/Testing Environment Config Examples
"configFolderName": "train-level-1.1"
"configFolderName": "exp-Ymaze"
"configFolderName": "train-level-2.1"
"configFolderName": "exp-damage"

Figure 9: **Overview of the configuration system of EVAAA.** (a) Modular config hierarchy, organized by components such as agent, court, resource, and obstacle. (b) Example instantiations for training (e.g., train-level-1.1, train-level-2.1) and testing (e.g., exp-Ymaze, exp-damage) via the configFolderName parameter.

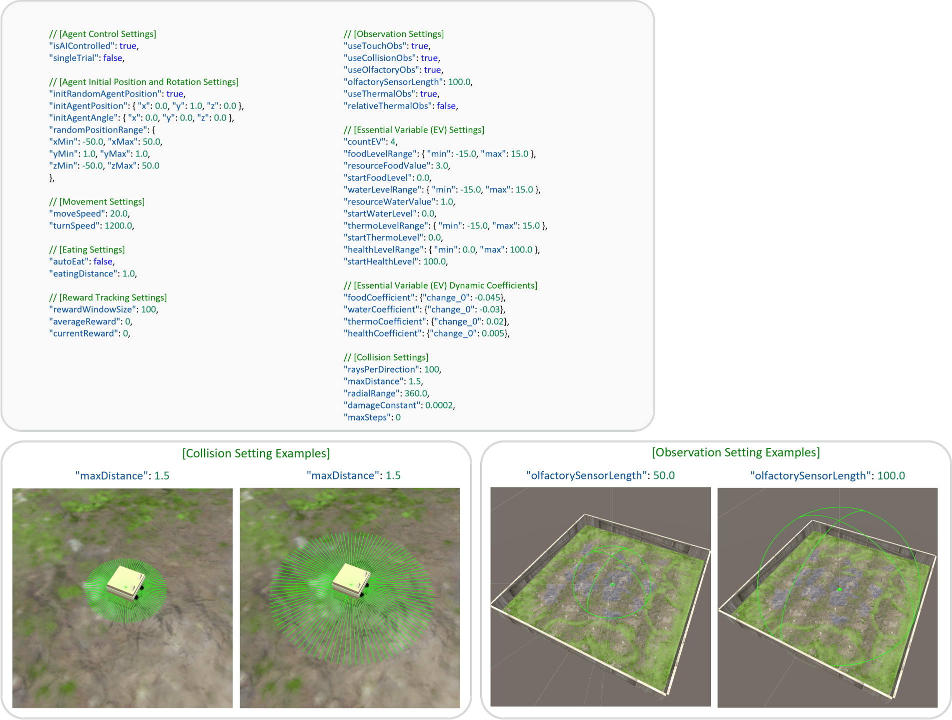
'startFoodLevel'. Consuming resources increases the corresponding EV based on values such as 'resourceFoodValue', 'resourceWaterValue', and so on. Movement is configured using 'moveSpeed' and 'turnSpeed', and the agent's initial position and orientation are set using 'randomPositionRange' and 'initAgentAngle'. The 'autoEat' true/false flags control whether the agent eats automatically when near a resource, and 'eatingDistance' specifies the required proximity. Sensory modalities—touch, collision, olfactory, and thermal—can be toggled individually, with 'olfactorySensorLength' controlling olfactory range. Collision-related settings include sensor range and 'damageConstant', which affects how damage is applied upon impact. EV decay rates are determined by coefficients such as 'foodCoefficient', which define how quickly each EV decreases over time.

E.2 Court config

The court configuration defines the size, layout, and visual appearance of the simulation field (**Fig. 11**). By default, the field is a 100x100 unit square centered at the origin, with coordinates ranging from -50 to 50 along the x and z axes. It acts as the base area for placing all environmental items, including resources, thermal grids, obstacles (e.g., rocks, trees, bonfires), and predators. Wall height ("wallHeight") and surface materials (e.g., "floorMaterialName", "wallMaterialName") are configurable, allowing environments to range from naturalistic (e.g., grassy textures) to controlled lab-like settings.

E.3 Resource config

The resource configuration file defines the placement, quantity, and appearance of interactive resources such as food and water (**Fig. 12**). Each resource group is assigned a 'prefabName', a spawn count, and randomized value ranges for position, rotation ('rotationRange'), and scale ('scaleRange'). In early training stages, resources are instantiated as simple red or blue cubes placed randomly. As the environment becomes more complex, resource shapes diversify into more realistic forms like apples and ponds, and their placement becomes more structured—for example, grouped apples or fixed-location ponds. Resource types are categorized as 'Random', 'Static', or 'GroupedRandom', which control spatial behavior and respawn logic.

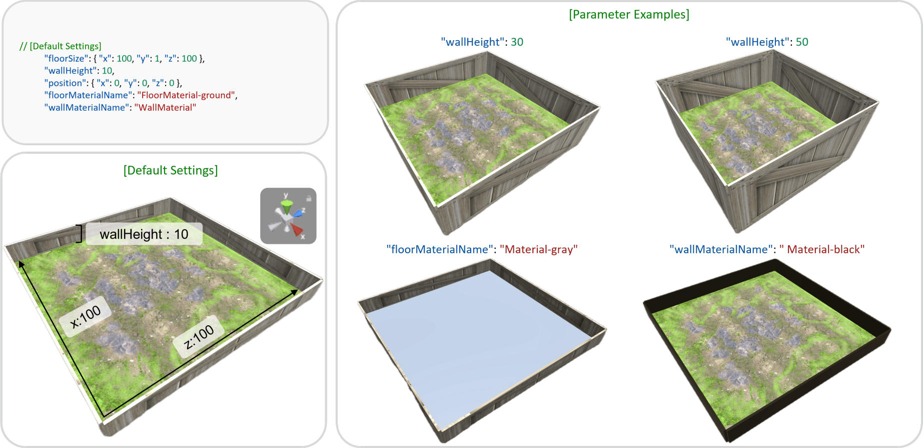


```
// [Agent Control Settings]
"isAIControlled": true,
"singleTrial": false,

// [Agent Initial Position and Rotation Settings]
"initRandomAgentPosition": true,
"initAgentPosition": { "x": 0.0, "y": 1.0, "z": 0.0 },
"initAgentAngle": { "x": 0.0, "y": 0.0, "z": 0.0 },
"randomPositionRange": {
"xMin": -50.0, "xMax": 50.0,
"yMin": 1.0, "yMax": 1.0,
"zMin": -50.0, "zMax": 50.0
},

// [Movement Settings]
"moveSpeed": 20.0,
"turnSpeed": 1200.0,

// [Eating Settings]
"autoEat": false,
"eatingDistance": 1.0,

// [Reward Tracking Settings]
"rewardWindowSize": 100,
"averageReward": 0,
"currentReward": 0,
```

```
// [Observation Settings]
"useTouchObs": true,
"useCollisionObs": true,
"useOlfactoryObs": true,
"olfactorySensorLength": 100.0,
"useThermalObs": true,
"relativeThermalObs": false,

// [Essential Variable (EV) Settings]
"countEV": 4,
"foodLevelRange": { "min": -15.0, "max": 15.0 },
"resourceFoodValue": 3.0,
"startFoodLevel": 0.0,
"waterLevelRange": { "min": -15.0, "max": 15.0 },
"resourceWaterValue": 1.0,
"startWaterLevel": 0.0,
"thermoLevelRange": { "min": -15.0, "max": 15.0 },
"startThermoLevel": 0.0,
"healthLevelRange": { "min": 0.0, "max": 100.0 },
"startHealthLevel": 100.0,

// [Essential Variable (EV) Dynamic Coefficients]
"foodCoefficient": {"change_0": -0.045},
"waterCoefficient": {"change_0": -0.03},
"thermoCoefficient": {"change_0": 0.02},
"healthCoefficient": {"change_0": 0.005},

// [Collision Settings]
"raysPerDirection": 100,
"maxDistance": 1.5,
"radialRange": 360.0,
"damageConstant": 0.0002,
"maxSteps": 0
```

[Collision Setting Examples]
"maxDistance": 1.5
"maxDistance": 1.5
[Observation Setting Examples]
"olfactorySensorLength": 50.0
"olfactorySensorLength": 100.0

Figure 10: **Agent configuration parameters in EVAAA.** Top: JSON snippets illustrating control, movement, eating, essential variable (EV) decay, observation, and collision settings. Bottom: Visualization of selected parameters, including collision sensing range (maxDistance) and olfactory sensor range (olfactorySensorLength). These settings govern how agents perceive the environment and manage internal state variables.



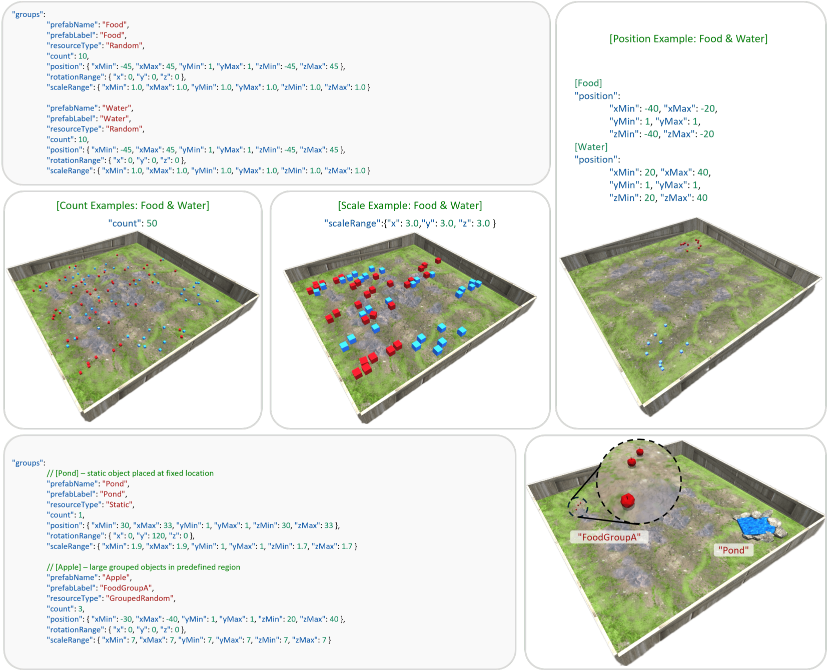
```
// [Default Settings]
        "floorSize": { "x": 100, "y": 1, "z": 100 },
        "wallHeight": 10,
        "position": { "x": 0, "y": 0, "z": 0 },
        "floorMaterialName": "FloorMaterial-ground",
        "wallMaterialName": "WallMaterial"
```

Figure 11: **Court configuration option in EVAAA.** Left: Default court settings define a 100×100 unit area with 10-unit high walls and a ground-like floor texture. Right: Examples illustrating how modifying wallHeight, floorMaterialName, and wallMaterialName changes the spatial and visual characteristics of the court. These parameters enable customization of court geometry and appearance for controlled or naturalistic experimental setups.

E.4 ThermoGrid config

The thermal environment in EVAAA is modeled using a dynamic, grid-based temperature system. The grid consists of a configurable number of square cells that span the full simulation area (**Fig. 13**), with each cell to which a scalar temperature value is assigned. A global baseline temperature (‘fieldDefaultTemp’) is applied across the grid, while localized variations are introduced via "hotspots". Each hotspot has a configurable temperature (hotSpotTemp), size (hotSpotSize), and count (hotSpotCount), allowing precise control over how much heat is applied, where it is placed, and how widely it spreads. To simulate realistic heat diffusion, a 2D smoothing filter (e.g., Gaussian blur) is applied using the smoothingSigma parameter, generating continuous temperature gradients across the field. Two types of hotspot placement are supported:

- **Random hotspot placement:** Enabled via useRandomHotSpot = true, this method distributes temperature sources randomly across the grid using the predefined hotspot parameters.
- **Object-linked hotspot placement:** Enabled via useObjectHotSpot = true, this mode generates thermal gradients around specific obstacles (e.g., bonfires) that act as heat sources. In this case, the temperature field is informed by the positions and values of these thermal objects rather than using random placement.



```
"groups":
          "prefabName": "Food",
          "prefabLabel": "Food",
          "resourceType": "Random",
          "count": 10,
          "position": { "xMin": -45, "xMax": 45, "yMin": 1, "yMax": 1, "zMin": -45, "zMax": 45 },
          "rotationRange": { "x": 0, "y": 0, "z": 0 },
          "scaleRange": { "xMin": 1.0, "xMax": 1.0, "yMin": 1.0, "yMax": 1.0, "zMin": 1.0, "zMax": 1.0 }

          "prefabName": "Water",
          "prefabLabel": "Water",
          "resourceType": "Random",
          "count": 10,
          "position": { "xMin": -45, "xMax": 45, "yMin": 1, "yMax": 1, "zMin": -45, "zMax": 45 },
          "rotationRange": { "x": 0, "y": 0, "z": 0 },
          "scaleRange": { "xMin": 1.0, "xMax": 1.0, "yMin": 1.0, "yMax": 1.0, "zMin": 1.0, "zMax": 1.0 }
```

[Count Examples: Food & Water]
"count": 50
[Scale Example: Food & Water]
"scaleRange":{"x": 3.0,"y": 3.0, "z": 3.0 }
[Position Example: Food & Water]

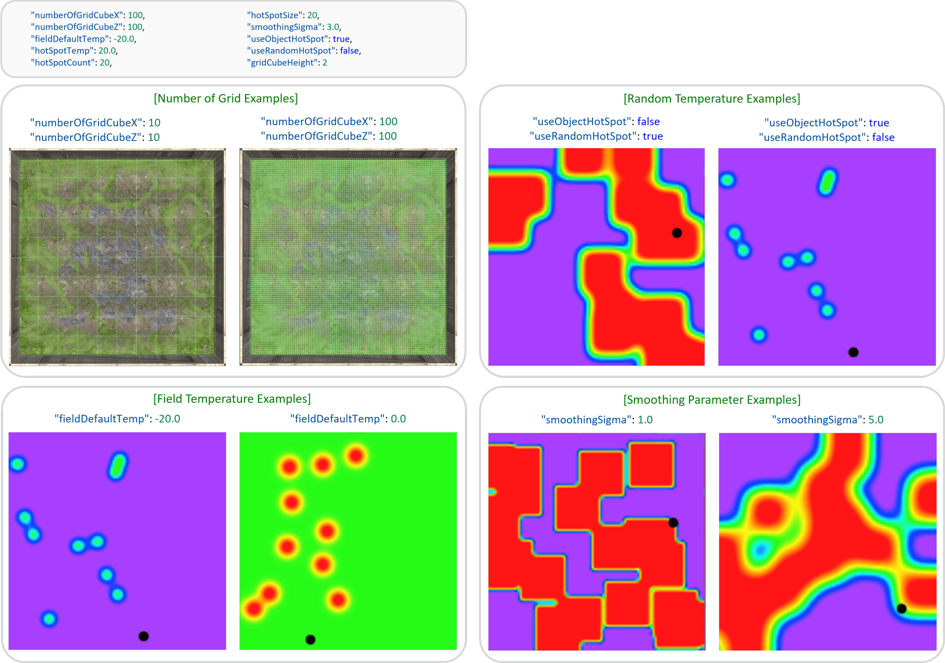
```
[Food]
"position":
          "xMin": -40, "xMax": -20,
          "yMin": 1, "yMax": 1,
          "zMin": -40, "zMax": -20
[Water]
"position":
          "xMin": 20, "xMax": 40,
          "yMin": 1, "yMax": 1,
          "zMin": 20, "zMax": 40
```

```
"groups":
          // [Pond] – static object placed at fixed location
          "prefabName": "Pond",
          "prefabLabel": "Pond",
          "resourceType": "Static",
          "count": 1,
          "position": { "xMin": 30, "xMax": 33, "yMin": 1, "yMax": 1, "zMin": 30, "zMax": 33 },
          "rotationRange": { "x": 0, "y": 120, "z": 0 },
          "scaleRange": { "xMin": 1.9, "xMax": 1.9, "yMin": 1, "yMax": 1, "zMin": 1.7, "zMax": 1.7 }

          // [Apple] – large grouped objects in predefined region
          "prefabName": "Apple",
          "prefabLabel": "FoodGroupA",
          "resourceType": "GroupedRandom",
          "count": 3,
          "position": { "xMin": -30, "xMax": -40, "yMin": 1, "yMax": 1, "zMin": 20, "zMax": 40 },
          "rotationRange": { "x": 0, "y": 0, "z": 0 },
          "scaleRange": { "xMin": 7, "xMax": 7, "yMin": 7, "yMax": 7, "zMin": 7, "zMax": 7 }
```

Figure 12: **Resource configuration options in EVAAA.** Top: JSON examples illustrating how to control resource type, quantity, spatial distribution, and visual scale using parameters like count, position, and scaleRange. Middle: Visualizations showing effects of varying object count (left), object size (center), and placement region for different resource types (right). Bottom: Advanced configuration examples including fixed-position static objects (e.g., “Pond”) and grouped dynamic objects (e.g., clustered apples in “FoodGroupA”) for structured environmental design.

E.5 Obstacle config

Obstacles are categorized as either randomized or fixed (**Fig. 14**). When the minimum and maximum values of the position range are the same, the obstacle is placed at a fixed location. Otherwise, it is positioned randomly within the specified range. For example, in Figure 14, Trees and RockWalls are placed at fixed positions. Rocks, Bushes, and Bonfires, by contrast, are located in random locations that change at the start of each episode.



```
"numberOfGridCubeX": 100,
"numberOfGridCubeZ": 100,
"fieldDefaultTemp": -20.0,
"hotSpotTemp": 20.0,
"hotSpotCount": 20,
"hotSpotSize": 20,
"smoothingSigma": 3.0,
"useObjectHotSpot": true,
"useRandomHotSpot": false,
"gridCubeHeight": 2
```

[Number of Grid Examples]

"numberOfGridCubeX": 10
"numberOfGridCubeZ": 10

"numberOfGridCubeX": 100
"numberOfGridCubeZ": 100

[Random Temperature Examples]

"useObjectHotSpot": false
"useRandomHotSpot": true

"useObjectHotSpot": true
"useRandomHotSpot": false

[Field Temperature Examples]

"fieldDefaultTemp": -20.0

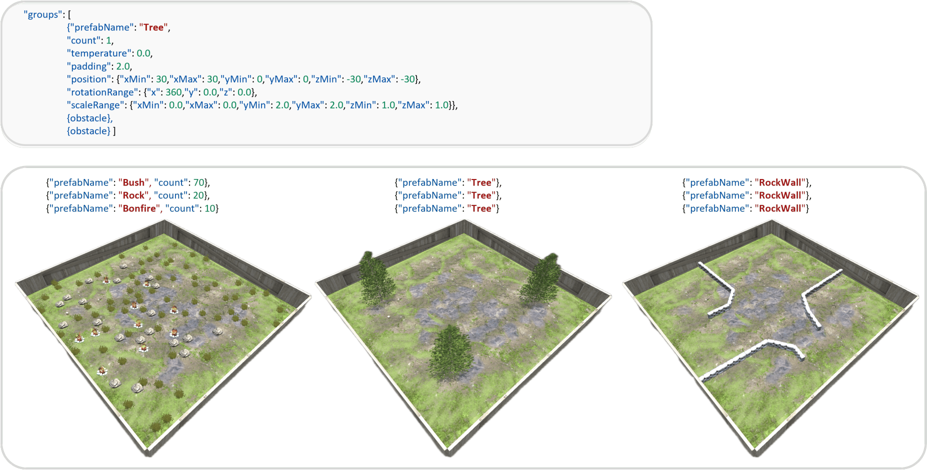
"fieldDefaultTemp": 0.0

[Smoothing Parameter Examples]

"smoothingSigma": 1.0

"smoothingSigma": 5.0

Figure 13: **Thermal field options in EVAAA.** Top left: Grid resolution comparison showing coarse (10x10) and fine (100x100) temperature grids. Top right: Hotspot generation using random placement (‘useRandomHotSpot’) vs. object-linked placement (‘useObjectHotSpot’). Bottom left: Impact of changing default baseline temperature (‘fieldDefaultTemp’) on the ambient environment. Bottom right: Effect of smoothing parameter (‘smoothingSigma’) on thermal diffusion, with higher values producing more continuous gradients. Temperature is color-coded from cool (blue) to hot (red).



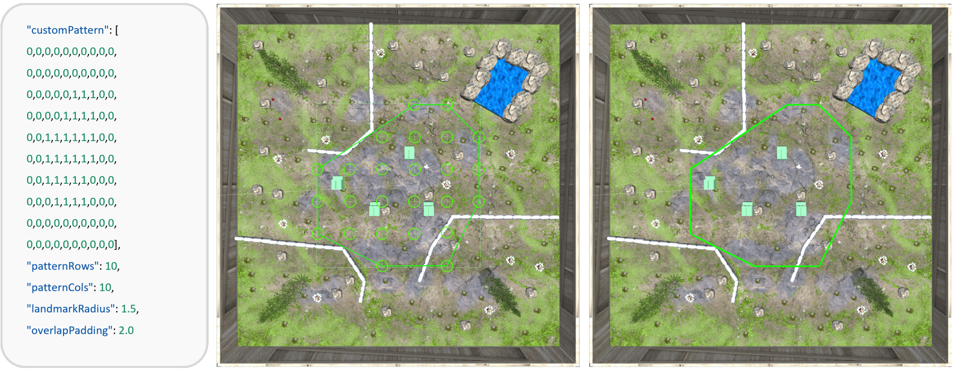
```
"groups": [
          {"prefabName": "Tree",
          "count": 1,
          "temperature": 0.0,
          "padding": 2.0,
          "position": {"xMin": 30,"xMax": 30,"yMin": 0,"yMax": 0,"zMin": -30,"zMax": -30},
          "rotationRange": {"x": 360,"y": 0.0,"z": 0.0},
          "scaleRange": {"xMin": 0.0,"xMax": 0.0,"yMin": 2.0,"yMax": 2.0,"zMin": 1.0,"zMax": 1.0}},
          {obstacle},
          {obstacle} ]
```

```
{"prefabName": "Bush", "count": 70},
{"prefabName": "Rock", "count": 20},
{"prefabName": "Bonfire", "count": 10}
```

```
{"prefabName": "Tree"},
{"prefabName": "Tree"},
{"prefabName": "Tree"}
```

```
{"prefabName": "RockWall"},
{"prefabName": "RockWall"},
{"prefabName": "RockWall"}
```

Figure 14: **Obstacle configuration in EVAAA.** Top: JSON configuration specifying parameters for static obstacles such as trees, including position, padding, and scale. Bottom left: Randomized placement of bushes, rocks, and bonfires to create unorganized terrain. Bottom center: Organized (or structured) placement of tree obstacles with consistent spacing. Bottom right: Placement of walls to obstruct navigation or visual perception.



```
"customPattern": [
0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,1,1,1,0,0,
0,0,0,0,1,1,1,1,0,0,
0,0,1,1,1,1,1,1,0,0,
0,0,1,1,1,1,1,1,0,0,
0,0,1,1,1,1,1,0,0,0,
0,0,0,1,1,1,1,0,0,0,
0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0],
"patternRows": 10,
"patternCols": 10,
"landmarkRadius": 1.5,
"overlapPadding": 2.0
```

Figure 15: **Landmark-based predator navigation boundary.** Left: A 10x10 array is defined (i.e., 'customPattern'). Middle: Landmarks (depicted in green circles) are placed according to the number specified per grid cell. However, if an obstacle is present that would block predator navigation, the landmark is omitted from that location. A convex hull (represented in green line) is computed around the placed landmarks to define the predator's navigable area. Right: The final convex mesh collider (in green outline) restricts the predator's range of motion.

E.6 Landmark config

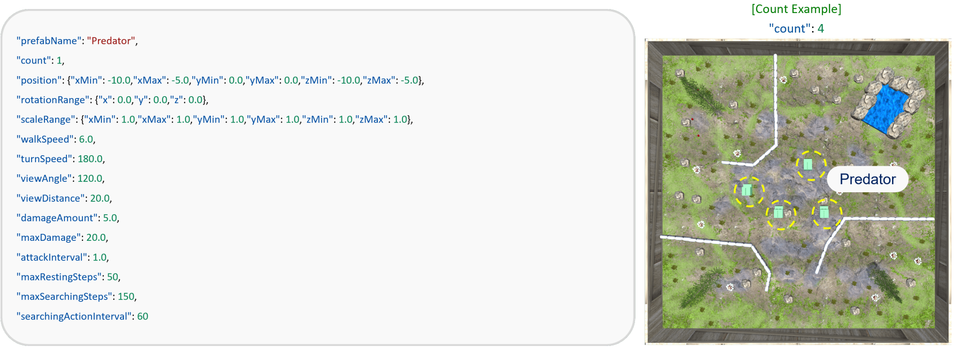
Landmarks are invisible reference points used to guide predators in setting their destinations. They are generated based on a grid layout defined in a landmark JSON configuration file (i.e., 'landmarkConfig.json'). This configuration specifies landmark positions using a 'customPattern' matrix, where a value of 1 denotes the presence of a landmark at a corresponding grid cell. It also defines parameters such as the size of each landmark (e.g., 'landmarkRadius'), the spacing between landmarks to avoid overlap (e.g., 'overlapPadding'), and a flattened array (e.g., 'customPattern') indicating the number of landmarks to place in each grid cell. To define the predator's moving area, the system computes a convex hull around all placed landmarks (**Fig. 15**). This hull is then converted into a mesh collider, representing the spatial boundary within which the predator is allowed to move. This method restricts predators to specific areas of the environment while preserving flexibility in design layout.

E.7 Predator config

The predator in EVAAA uses Unity's built-in AI navigation system and operates across four discrete behavioral states: Resting, Searching, Chasing, and Attacking. In the Searching state, the predator moves stochastically using Unity's built-in NavMesh—a navigation system that enables pathfinding over walkable surfaces— and the predator's state changes to the Resting state after a configurable number of steps (**Fig. 16**). When the agent enters the predator's vision cone, defined by an adjustable angle and distance, the predator switches to the Chasing state, targeting the agent's position. On physical contact, it enters the Attacking state and increases the agent's internal damage EV. If the agent escapes or breaks contact, the predator returns to Searching.

E.8 Day/Night cycle config

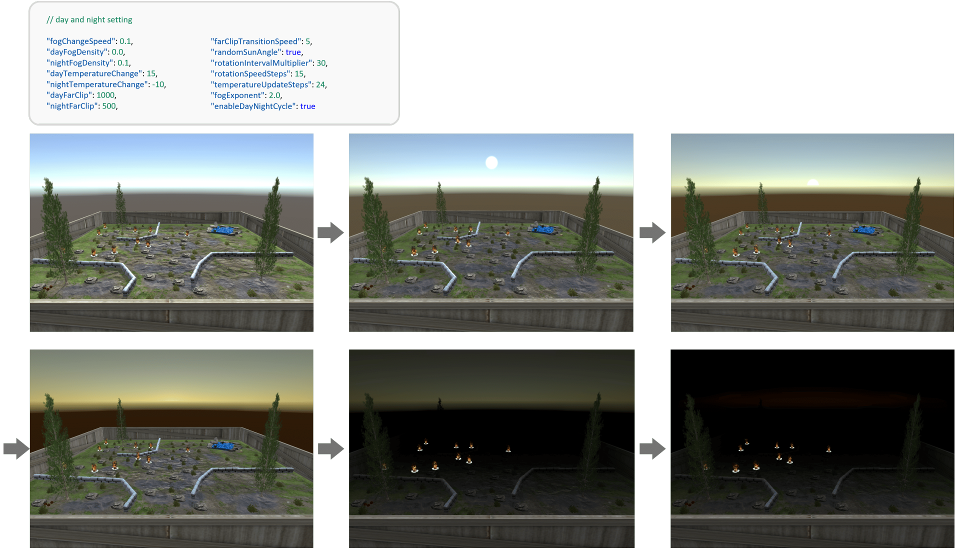
EVAAA implements a configurable day-night cycle that introduces time-dependent environmental changes. The cycle is implemented by rotating a directional light source in the Unity scene (**Fig. 17**). Each phase adjusts environmental parameters such as lighting intensity, fog density, ambient color, and scene exposure. All relevant settings are defined in the daynightConfig.json, including phase durations, light rotation speed ('dayChangeSpeed'), and temperature effects like 'nightTemperatureChange', which lowers the overall thermal field during night phases. For example, temperature gradually rises during daytime and decreases at night, requiring agents to adjust their thermal regulation behavior over time.



```
"prefabName": "Predator",
"count": 1,
"position": {"xMin": -10.0,"xMax": -5.0,"yMin": 0.0,"yMax": 0.0,"zMin": -10.0,"zMax": -5.0},
"rotationRange": {"x": 0.0,"y": 0.0,"z": 0.0},
"scaleRange": {"xMin": 1.0,"xMax": 1.0,"yMin": 1.0,"yMax": 1.0,"zMin": 1.0,"zMax": 1.0},
"walkSpeed": 6.0,
"turnSpeed": 180.0,
"viewAngle": 120.0,
"viewDistance": 20.0,
"damageAmount": 5.0,
"maxDamage": 20.0,
"attackInterval": 1.0,
"maxRestingSteps": 50,
"maxSearchingSteps": 150,
"searchingActionInterval": 60
```

```
[Count Example]
"count": 4
```

Figure 16: **Predator deployment configuration in EVAAA.** Left: JSON configuration for a predator agent, including key parameters such as position range, view angle (120 degrees), walk speed, and state transition limits (e.g., ‘maxRestingSteps’, ‘maxSearchingSteps’). Right: Visualization of four predators navigating using Unity’s NavMesh system.



```
// day and night setting

"fogChangeSpeed": 0.1,                    "farClipTransitionSpeed": 5,
"dayFogDensity": 0.0,                     "randomSunAngle": true,
"nightFogDensity": 0.1,                   "rotationIntervalMultiplier": 30,
"dayTemperatureChange": 15,               "rotationSpeedSteps": 15,
"nightTemperatureChange": -10,            "temperatureUpdateSteps": 24,
"dayFarClip": 1000,                       "fogExponent": 2.0,
"nightFarClip": 500,                      "enableDayNightCycle": true
```

Figure 17: **Day/night cycle configuration and visual transition examples.** Top: Configuration parameters controlling the day-night cycle, including fog density, temperature shift, sun angle, and clipping distance. Bottom: Five key visual phases of the cycle: day, sunset, night, deepnight, and dawn. These phases dynamically modulate ambient lighting, visibility, and temperature, influencing agent behavior and sensory inputs.

F Naturalistic training level design

EVAAA implements a two-tiered environment structure: (1) Naturalistic training environments and (2) unseen experimental testbeds. The naturalistic training environment is organized as a curriculum that gradually increases environmental complexity, enabling agents to learn homeostatic regulation across various challenges–from basic resource foraging to predator avoidance. To evaluate whether agents have learned homeostatic behaviors and can generalize beyond these basic survival behaviors, we introduce a set of unseen test environments. The unseen test environments are described in the next section G.

Level 1. Basic resource foraging Level 1 focuses on basic survival by training agents to regulate internal states through food and water (random locations) foraging and consumption. In Level 1-1, 50 red food cubes and 50 blue water cubes are randomly distributed across the field with a variable ambient temperature. This setup promotes learning the direct link between resource consumption and increases in EV level. In Level 1-2, resources are relocated to the two opposite diagonal corners of the field, requiring more spatially directed foraging.

Level 2. Obstacle-resource mapping Level 2 increases task complexity by introducing environmental hazards and spatial cues. In Level 2-1, bonfires act as localized heat sources, requiring agents to actively regulate body temperature, while rocks and bushes obstruct movement. The agent must avoid collisions to keep the damage level within a viable range. Level 2-2 introduces consistent spatial layouts: water cubes now appear as a stationary pond providing a reliable water source, and food, now depicted as apples, spawns only under trees. These spatial cues encourage the agent to learn predictable environment-resource associations and support more strategic, non-random foraging behavior.

Level 3. Terrain exploration and navigation Level 3 supports flexible navigation and adaptive exploration within partially structured environments. In Level 3-1, three tree regions are positioned in separate corners of the field, each partially enclosed by walls and bushes. Apples are placed beneath these trees, requiring the agent to navigate around obstacles while managing path efficiency and thermal regulation. In Level 3-2, food location varies across time: at the beginning of each episode, apples spawn in only one randomly selected tree region (Fig. 18). After the agent consumes all the food in that region, the apples reappear in a randomly selected tree. This shifting food placement encourages agents to explore, identify the current resource location, and adjust their foraging strategies using spatial navigation skills and rapid decision-making.

Level 4. Dynamic threats Level 4 integrates predator threats with a day/night cycle, challenging agents to coordinate survival behavior with internal regulation. In Level 4-1, predators patrol the field and inflict damage upon contact, but revert to a passive search state once the agent escapes their line of sight. To prevent unavoidable deaths during resource consumption, predators are restricted to pre-defined navigation zones (Fig. 15). In Level 4-2, predator behaviors and ambient temperature shift over time: predators pursue during the daytime but remain inactive at night, while temperature rises and falls accordingly. These conditions require agents to balance safety, thermoregulation, and foraging, promoting time-sensitive and adaptive strategies.

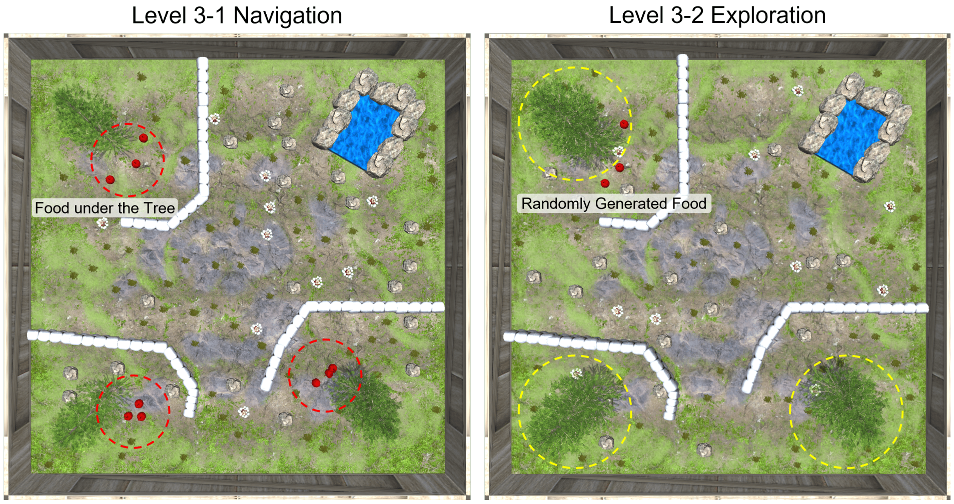


Figure 18: **Level 3 food resources random generation example.** In Level 3-1, the apples are placed at all three tree regions. In Level 3-2, on the other hand, at the start of each episode, food spawns in a randomly selected region (e.g., yellow dotted regions) and relocates to another once consumed.

G Experimental testbed design

G.1 Basic Homeostatic Regulation

Two-resource choice task To assess whether agents can prioritize among competing internal needs, this task presents two distinct resources (e.g., food vs water or food vs bonfire to warm the body temperature) separated by a wall. Once the agent commits to one side of the resource area, access to the other side becomes temporarily obstructed. Successful performance requires selecting the resource most urgently needed to maintain current homeostasis.

Collision avoidance task This task evaluates the agent's ability to anticipate and avoid damage. A visible resource (e.g., a food red cube) is placed behind narrow passages or closely spaced obstacles that induce damage upon contact. Although the resource is visible from the start, the agent must navigate safely through constrained space, demonstrating precise motor control and the capacity to inhibit impulsive actions in favor of safe, state-preserving paths.

G.2 Advanced Adaptive Skills

Risk-taking task In this setting, the agent must reach a needed resource (e.g., food or water) located behind a bonfire that rapidly raises the agent's internal temperature level. The agent must decide whether to endure the rising temperature to preserve other essential variables, which would otherwise decline over time and ultimately lead to failure in survival. This task tests the agent's ability to manage the conflicting needs.

Y-maze (spatial navigation) The Y-maze evaluates agents' ability to make spatial decisions based on internal state comparisons. Two resource types (e.g., food and water) are placed at the ends of a Y maze. At the start of each episode, the agent begins at the base and must choose which arm to explore first. If the satiation level is lower than hydration, the optimal strategy is to visit the food cube located arm first. Once the satiation level is restored and hydration becomes more depleted, the agent must redirect toward the water arm that was previously seen but not initially visited. This task evaluates the agent's ability to monitor internal needs, recall previously encountered spatial locations, and flexibly sequence goal-directed actions in response to evolving interoceptive demands.

Goal manipulation under volatile internal-state conditions This task introduces sudden, unobservable changes to the agent's internal state during ongoing action execution. The change is triggered when the agent crosses a transparent boundary (**Fig. 19**). For example, the agent may begin moving toward a food resource due to low satiation levels, but after crossing the boundary, hydration becomes the most depleted EVs. The agent must recognize this internal shift, revise its plan, and re-prioritize its goals accordingly. This setting tests the agent's ability to adapt goal-directed actions based on internal state changes and serves as a virtual analogue to real-world behavioral experiments where manipulating internal physiological states is challenging, such as in animal studies.

Multi-goal planning The agent must coordinate actions across three EVs: satiation, hydration, and temperature (e.g., warmth from a bonfire). The agent's behavior is guided by an internal priority that reflects the current levels of EVs. For example, if the current internal state is as temperature < hydration < satiation, the optimal policy is to first seek warmth to restore body temperature, then hydrate, and finally consume food. Success depends on the agent's ability to plan and execute actions according to the relative urgency of each EV while maintaining overall homeostasis. While multiple action sequences may be valid, the agent must keep all EVs within a stable range throughout the test period.

Predators with day/night In this task, the resource is visible, but predators actively move around and attempt to attack the agent during the day, making direct access risky. At night, predators become inactive, providing a safer opportunity to obtain the resources. The optimal strategy is to avoid predators during the daytime and wait until nightfall to approach and take the resource. This setup is designed to assess the agent's ability to balance immediate needs and survival by demonstrating risk-aware, time-sensitive decision-making.

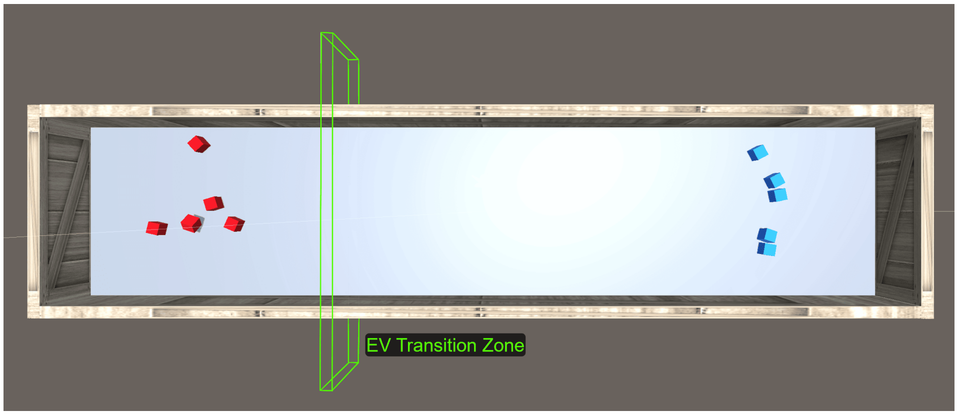


Figure 19: **Goal manipulation under volatile internal-state conditions.** The agent initiates movement in one direction based on its initial EV level. Upon crossing an invisible EV transition zone (indicated by the green box), the EV level is altered, prompting the agent to reverse direction.

H Evaluation protocol

H.1 Computational resources

All experiments were conducted using a single NVIDIA RTX 3090 GPU. Training times varied depending on task complexity, with most environments requiring approximately 1.5 to 2 days to reach convergence under default curriculum schedules and environment configurations. The system ran under a standard PyTorch + Unity ML-Agents setup with GPU acceleration.

H.2 Architectures and hyperparameters

We implemented three reinforcement learning agents: DQN, PPO, and DreamerV3. For PPO and DreamerV3, we adopted the Sheeprl framework `https://github.com/Eclectic-Sheep/sheeprl`, which provides modular and reproducible implementations aligned with recent benchmarks. In contrast, the DQN agent was developed from scratch to suit our environment's specific requirements. The following sections describe the model architectures for each agent.

DreamerV3 DreamerV3 was implemented using the Sheeprl API with the $dreamer_v3_S.yaml$ configuration. The agent consists of a world model, an actor, and a critic. The world model encodes RGB observations using a four-layer convolutional network with progressively increasing channels (32 → 64 → 128 → 256), kernel size 4×4, stride 2, and LayerNorm applied after each layer. Vector observations—including essential variables, olfactory, thermal, and collision features—are processed via a two-layer MLP with 512 units per layer, SiLU activations, and LayerNorm. Latent dynamics are modeled through a recurrent state-space model (RSSM) comprising a feedforward MLP (1029 → 512), a LayerNorm-GRU cell (1024 → 1536), and two additional MLPs representing the transition and representation models (512 → 1024 and 5120 → 1024, respectively). The decoder reconstructs visual input using a transposed convolutional network (256 → 128 → 64 → 32 → 3), and reconstructs vector modalities using an MLP decoder with multiple linear heads for different variables. The actor and critic are two-layer MLPs (1536 → 512 → 512), both with LayerNorm and SiLU activations. The actor outputs the actions (5), while the critic predicts value distributions across 255 bins. All components follow Hafner's initialization scheme and use consistent activation and normalization settings.

PPO PPO was implemented using the Sheeprl API with a dual-branch encoder and separate actor and critic heads. Visual inputs are constructed by stacking four consecutive 64×64 RGB frames, resulting in a 12-channel input image. This consists of three convolutional layers with ReLU activations (8×8, 4×4, and 3×3 kernels; strides 4, 2, and 1), followed by a fully connected projection to a 512-dimensional visual embedding. Vector inputs—including essential variables, olfactory, thermal, and collision signals—are passed through a two-layer MLP with ReLU activation and a final

Table 2: **Hyperparameters used for training**

Parameter	**DQN**	**PPO**	**DreamerV3-S**
Optimizer	Adam	Adam	Adam
Learning rate	0.0001	0.00025	0.0001
Other optimizer params	ϵ-greedy: decays 1.0→0.1 (209k)	$\epsilon = 10^{-5}$ decay=0	$\epsilon = 10^{-6}$ decay=0

Grad norm clipping	–	0.5	`actor/critic=100,`
			`world_model=1000`

Baseline cost	–	0.5	1
Entropy cost	–	0.01	0.0003
Discount factor (γ)	0.95	0.99	0.997
GAE Lambda (λ)	–	0.95	0.95
IS clip range (ϵ_{clip})	–	0.1	–
Batch size	12	256	16
Unroll length	–	1024	64
Actor gradients	TD (Q-learning)	Analytical (GAE)	REINFORCE

projection to 64 dimensions. The concatenated feature vector (576 dimensions) is shared across the actor and critic networks. The actor is implemented as a one-layer MLP that produces a latent feature vector, which the agent subsequently interprets as the parameters of a diagonal Gaussian distribution over actions. The critic is a two-layer MLP that outputs scalar state values. Both heads use ReLU activations and follow a consistent architectural template defined in the configuration. This design ensures efficient encoding of multimodal state information while maintaining architectural symmetry across policy and value functions.

DQN Visual input is represented as a stack of two consecutive 64×64 RGB frames (6 channels total) and passed through a four-layer convolutional encoder with increasing channel widths (16 → 32 → 64 → 128), each followed by BatchNorm and ReLU activations. The resulting visual embedding is projected to 1000 dimensions. Vector observations—including essential variables, olfactory, thermal, and collision signals—are each passed through independent linear encoders that project to 50–100 dimensions depending on the modality. These embeddings are concatenated with the visual feature and passed through a fully connected fusion layer (1450 → 400) before producing action-value estimates via a final output layer (400 → 5). The network is trained using the standard DQN objective with experience replay and a target network. Reward shaping is enabled to enhance learning performance in the presence of sparse or delayed feedback. The design ensures tight integration between perceptual and internal state features through modality-specific encoders and joint feature fusion.

See **table 2** for hyperparameters used for training DQN, PPO, and Dreamer V3.

H.3 Success rate metrics

Success rates were computed independently for each experimental test condition using task-specific criteria aligned with the behavioral objectives of each task. Evaluation strategies fall into two categories: single-trial success and survival-duration success. For survival-based tasks, a fixed maximum episode length (maxStep) is assigned per task, defining the duration the agent must survive to be considered successful.

Single-Trial Success Criteria. These tasks focus on discrete resource acquisition. An episode is marked as successful if the agent completes the designated objective before reaching the maxStep:

- **Two-resource choice tasks (Food, Water):** Success is defined as EpisodeEndType = ResourceConsumed and positive consumption of the relevant resource (FoodConsumed > 0, WaterConsumed > 0). (maxStep = 500)
- **Goal manipulation under volatile internal-state conditions:** Success requires adapting to an internal state switch and reaching the newly prioritized resource.(maxStep = 300)

- **Risk-taking task:** Success is defined as EpisodeEndType = ResourceConsumed, reflecting successful resource acquisition under threat. (maxStep = 1000)
- **Collision avoidance task:** Success is defined as EpisodeEndType = ResourceConsumed, requiring the agent to navigate hazard zones to reach the target. (maxStep = 500)
- **Predators with day/night:** Success is defined as EpisodeEndType = ResourceConsumed, requiring the agent to avoid moving predators to reach the target. (maxStep = 300)

Survival-Duration Success Criteria. These tasks emphasize sustained internal regulation and long-horizon planning. An episode is marked as successful if the agent survives for the entire duration, defined by the maxStep limit:

- **Two-resource choice tasks (Temperature):** Success = EpisodeEndType = MaxStepReached, indicating full survival under thermal constraints. (maxStep = 300)
- **Y-Maze (spatial navigation):** Success = EpisodeEndType = MaxStepReached, requiring the agent to navigate and switch goals adaptively. (maxStep = 350)
- **Multi-goal planning:** Success = EpisodeEndType = MaxStepReached, reflecting correct prioritization across multiple EVs over time.(maxStep = 350)

For each agent model and experiment type, the total number of episodes and the number of successful episodes are recorded. The success rate is computed as Success Rate $= \frac{\text{Successful Episodes}}{\text{Total Episodes}} \times 100$

H.4 Human evaluation protocol

To assess human performance on the same tasks as EVAAA agents, we conducted a controlled behavioral study with eight participants. The study was approved by the Institutional Review Board (IRB), and informed consent was obtained from all participants. The whole experiment lasted approximately 2–3 hours and consisted of two phases: a curriculum-based training phase and a test phase. During training, participants experienced the same naturalistic training environments to understand the relationship between environment and EV, and reward feedback. Most participants spend considerable time in Level 1-1, first learning how to interact with the environment—such as using keyboard inputs (e.g., arrow keys, spacebar)—and then exploring how different actions influence EV levels. After each level, experimenters pose diagnostic questions to evaluate participants' understanding of EV regulation (e.g., what actions led to successful survival). These questions serve as a practical substitute for the full training experience, as it is unrealistic to ask human participants to complete the same number of steps as agents (e.g.,~100,000steps is 400 minutes of continuous play for each level of naturalistic tasks). We advanced to the next level when participants demonstrated a sufficient level of understanding.After completing the curriculum, participants received a brief explanation of the evaluation protocol and completed the same test environments. Participants were compensated at a rate of $11/hour Participation was voluntary, with the option to withdraw at any time without penalty. No personally identifiable or biometric data was collected.

I Results

I.1 Key behavioral-based analysis of agent and human behavior across unseen test tasks

As noted in the main results, we also implemented a recurrent baseline (PPO+LSTM). However, this model failed to demonstrate successful learning (**Fig. 20**). Therefore, the following behavioral-event analysis focuses on the agents trained with DQN, PPO, and DreamerV3. To complement the main result, we present an analysis based on selected key behavioral events across tasks, highlighting key environmental interactions that influence success rates. **Figure 21-24** visualizes normalized event counts (mean ± SE) for each algorithm (e.g., DQN, PPO, DreamerV3), human participants, and training level, offering insights into behavioral patterns such as resource consumption and causes of failure. Across testbeds, we observe distinct strategies in how different agents and human participants manage EV levels and respond to task-specific hazards. In tasks such as *Two-resource choice tasks (Food, Water)*, *Risk-taking task*, *Collision avoidance task*, *Goal manipulation under volatile internal-state conditions*, and *Predators with day/night*, higher success rates can be achieved when agents and humans engage in food or water consumption actions that are aligned with their current essential variable (EV) levels, often through task-specific strategies such as avoiding collision while approaching the resource. In contrast, tasks such as *Two-resource choice tasks (Temperature)*, *Y-Maze (spatial navigation)*, and *Multi-goal planning*, emphasize long-term survival, where success depends on maintaining internal-state balance until the task's maximum step rather than reaching a discrete goal. For instance, in *Two-Resource Choice (Temp)*, successful agents and humans tend to remain near heat sources (e.g., bonfire) as internal temperature drops, maintaining this positioning strategy over time instead of over-consuming food or water. Overall, these analyses provide a detailed framework for dissecting the behavioral dynamics underlying agent success or failure. By analyzing key event patterns, we can identify which action patterns facilitate goal completion as well as maladaptive behaviors that correlate with failure, offering mechanistic insights into agent-environment interactions and internal state regulation.

I.2 Agent emergent behavior

Modality ablation study. As discussed in the main text, we conducted a systematic ablation study to assess the relative importance of each sensory component. We trained DreamerV3 agents with one of five components removed: (1) Essential Variables (EVs), (2) Vision, (3) Olfaction, (4) Thermoception, and (5) Collision. Agents in the 'No EV' and 'No Vision' conditions fail to learn, confirming these features are critical. Agents in the 'No Olfaction', 'No Thermoception', and 'No Collision' conditions demonstrate varying degrees of learning, allowing for an analysis of their differential impact (**Fig. 25**). Following this, the **Figure 26** presents the generalization performance of these agents on the unseen test tasks.

Detouring behavior after misjudgment. In the two-resource choice task (e.g., food vs. water choice), agents trained on Level 1-2 or higher consistently exhibited detouring behavior. After initially approaching the incorrect resource—for instance, heading toward the food cube while dehydrated—they corrected their trajectory by detouring around the wall to reach the appropriate resource. In contrast, agents trained exclusively on Level 1-1, where resources are randomly placed and easily accessible, tended to persist along suboptimal paths without corrective action. The emergence of detouring behavior underscores the role of stable environment-resource associations in facilitating spatial re-planning and policy adjustment.

Self-Termination. Agents trained on Level 3-2 frequently exhibit deliberate episode termination behavior in the Y-maze (spatial navigation) task. Unlike earlier training levels, such as Level 3-1, where food consistently appears in predictable locations, Level 3-2 introduces uncertainty by spawning apples in only one randomly selected tree region. When unable to locate the food area, some agents adopt a strategy of repeatedly colliding with allies to deplete their damage EV and terminate the episode. This behavior reflects a learned policy that favors resetting the environment over persisting in low-reward or difficult-to-recover states.

Videos demonstrating these emergent behaviors are available in the GitHub repository.

J Broader impacts

EVAAA is a simulation platform for training and evaluating agents based on internal survival needs such as satiation, hydration, body temperature, and tissue integrity (the level of damage). By focusing on intrinsic regulation rather than external rewards, EVAAA enables research on adaptive behavior, goal prioritization, and autonomous decision-making under internal pressure. This line of research may inform the design of more robust agents for applications like healthcare support, education, or uncertain environments. However, internal-state-driven learning raises challenges in transparency and control. For example, agents may exhibit opaque strategies when internal signals are not externally visible, or misaligned behaviors if deployed without careful oversight. To support responsible use, all code and tasks are released as open-source. We encourage users to ensure interpretability, avoid overfitting to abstract survival dynamics, and evaluate downstream impacts before applying such systems in real-world contexts.

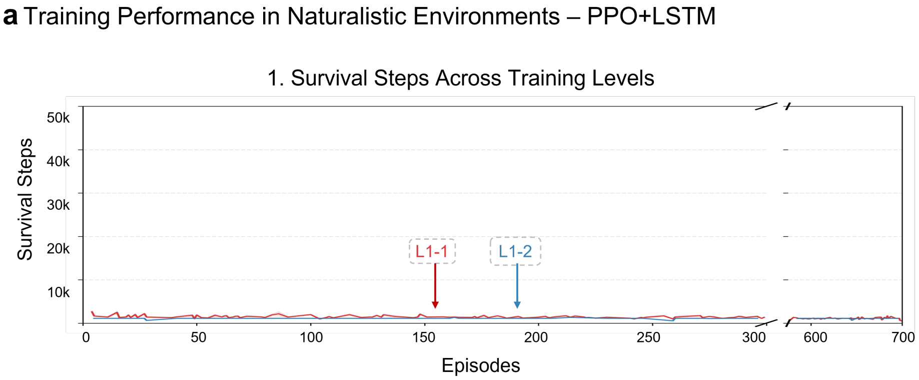


Figure 20: **Training performance of the recurrent baseline (PPO+LSTM).** PPO+LSTM shows survival steps remained minimal throughout training in both Level 1-1 (red) and Level 1-2 (blue). This indicates a failure to learn, justifying the model's exclusion from the detailed behavioral-event analysis.

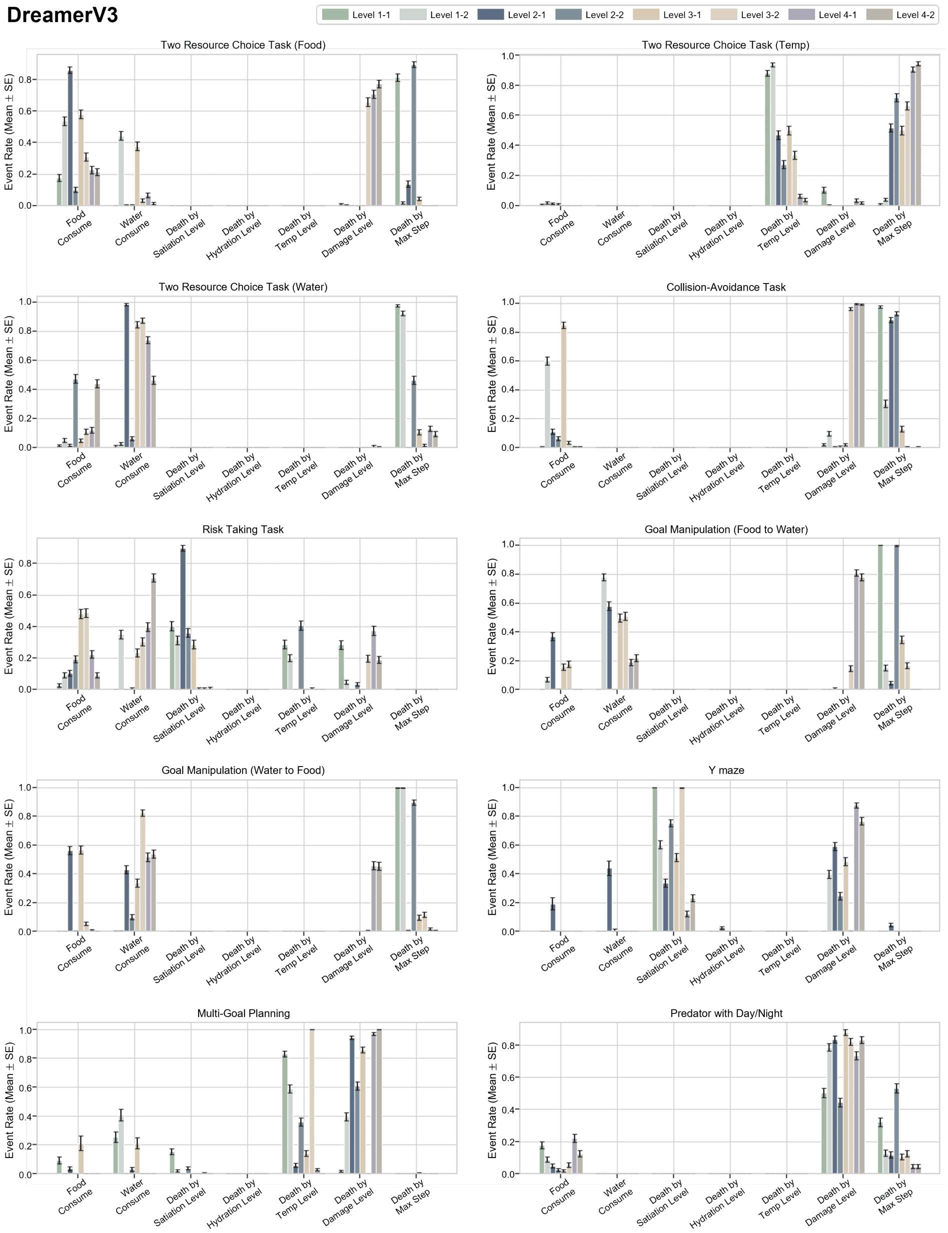


Figure 21: **DreamerV3 event-based behavior across test tasks.** DreamerV3 demonstrates flexible and adaptive strategies across ten test environments. It actively manages internal states by adjusting resource use and positioning, leading to higher success in both simple and complex tasks.

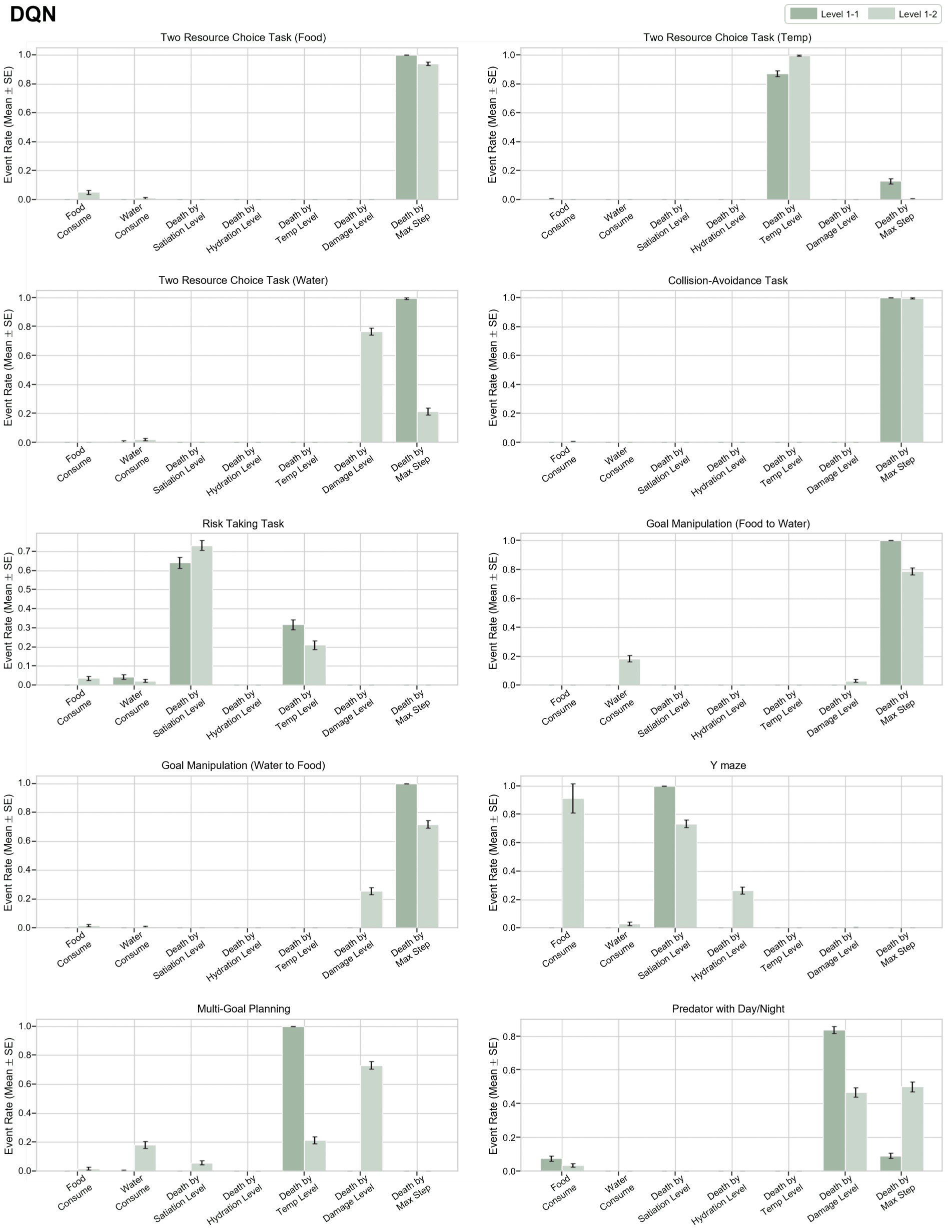


Figure 22: **DQN event-based behavior across test tasks.** DQN exhibits limited adaptability when evaluated in testbed environments. Event patterns reveal frequent failures in appropriate resource consumption, leading to low success rates—primarily due to reaching the maximum allowed steps—indicating the agent's difficulty in regulating internal states and adapting behavior beyond its training experience.

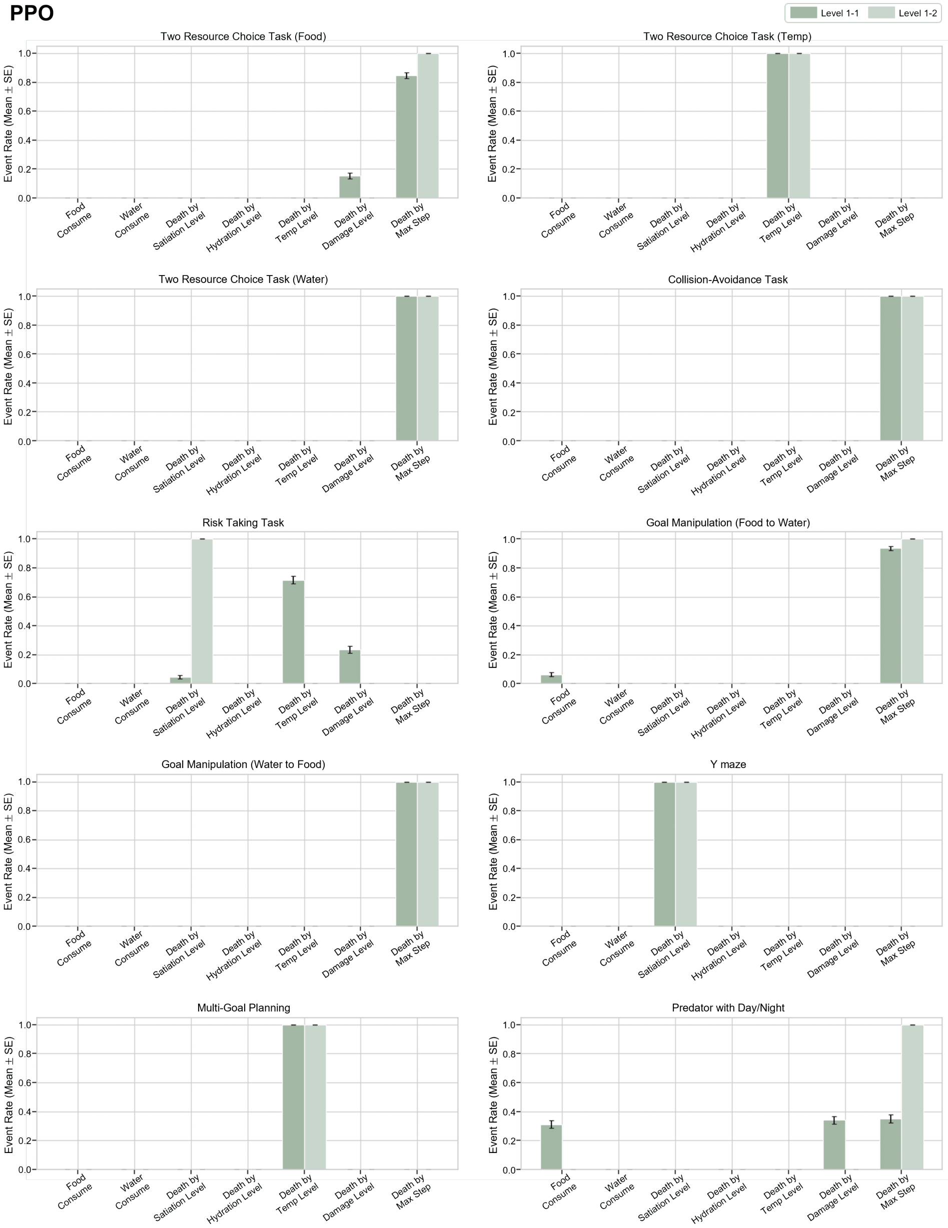


Figure 23: **PPO event-based behavior across test tasks.** PPO demonstrates slightly better adaptability than DQN, yet still fails to regulate internal states effectively. Agent deaths frequently result from insufficient resource consumption, with failures often linked to temperature regulation.

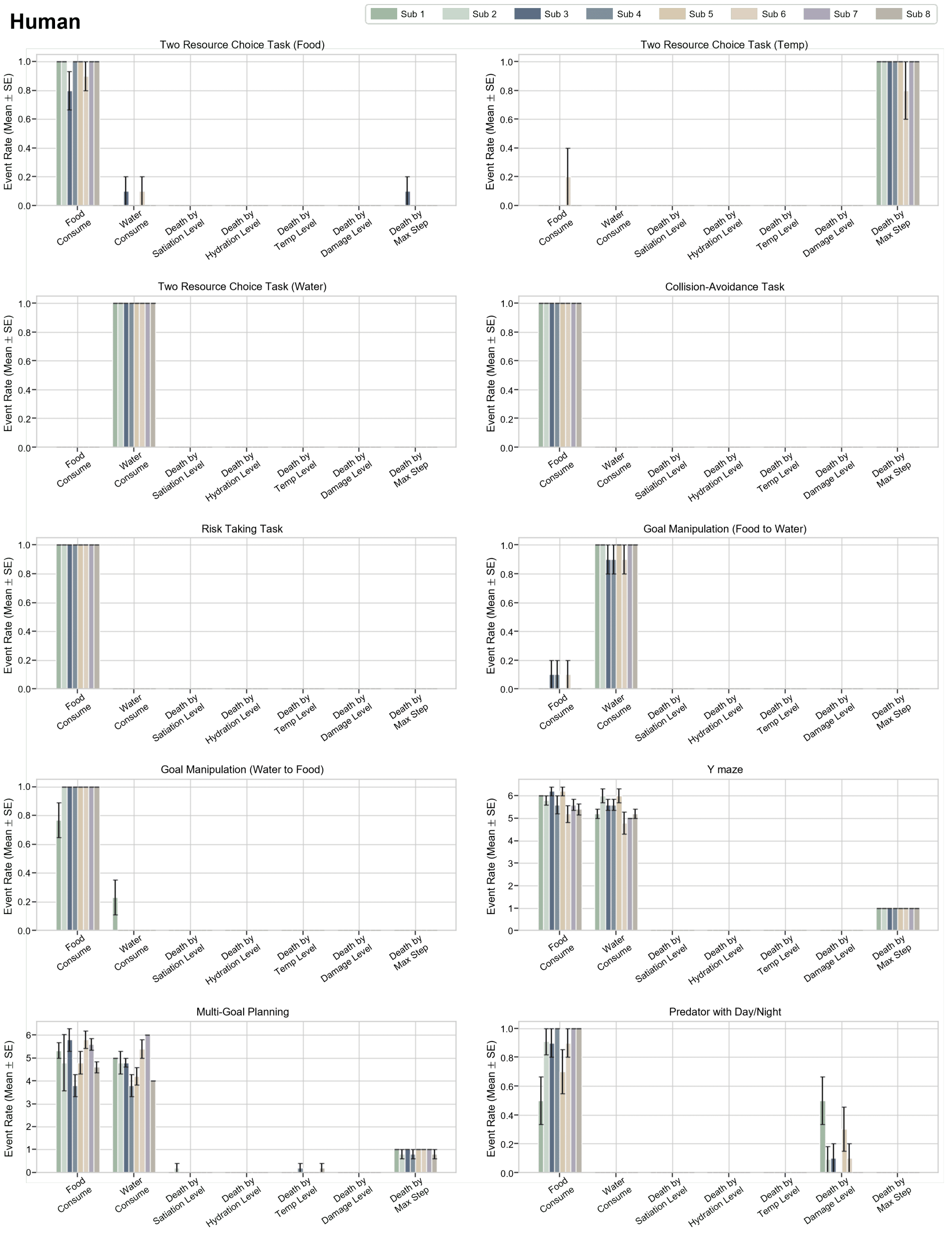


Figure 24: **Human event-based behavior across test tasks.** Human participants consistently show high levels of appropriate resource consumption and low internal failure rates across tasks. Their event patterns indicate proactive management of internal states, such as consistently consuming food and water before critical depletion, which supports high success and flexible adaptation in diverse environments.

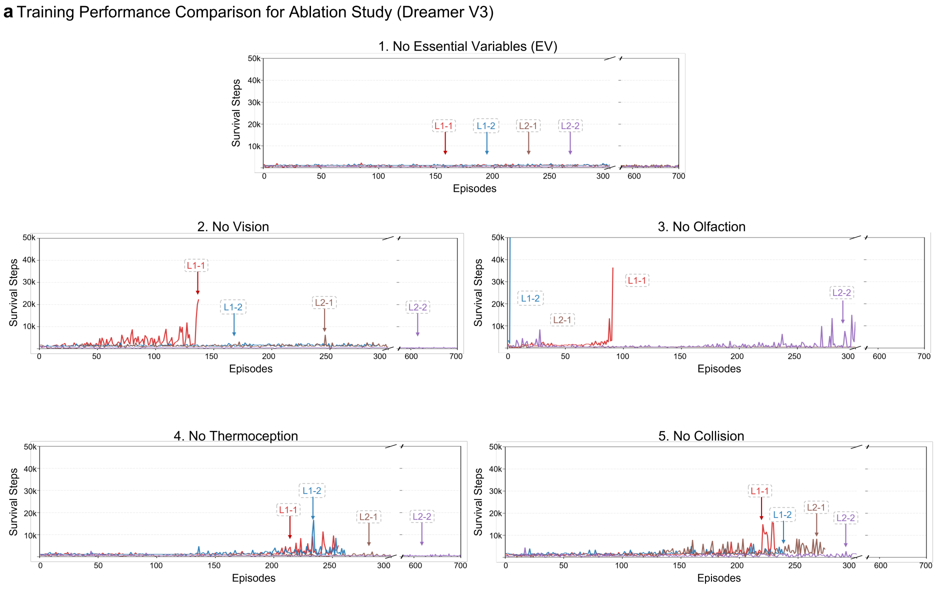


Figure 25: **Training performance comparison for the modality ablation study.** Training performance for agents trained with one sensory component removed. (1) 'No Essential Variables' and (2) 'No Vision' conditions result in minimal successful learning, confirming their critical role. The remaining conditions illustrate the differential impact of non-visual modalities: agents in the (3) 'No Olfaction' condition perform notably better than those in the (4) 'No Thermoception' and (5) 'No Collision' conditions, suggesting thermoception and collision are more impactful for successful training.

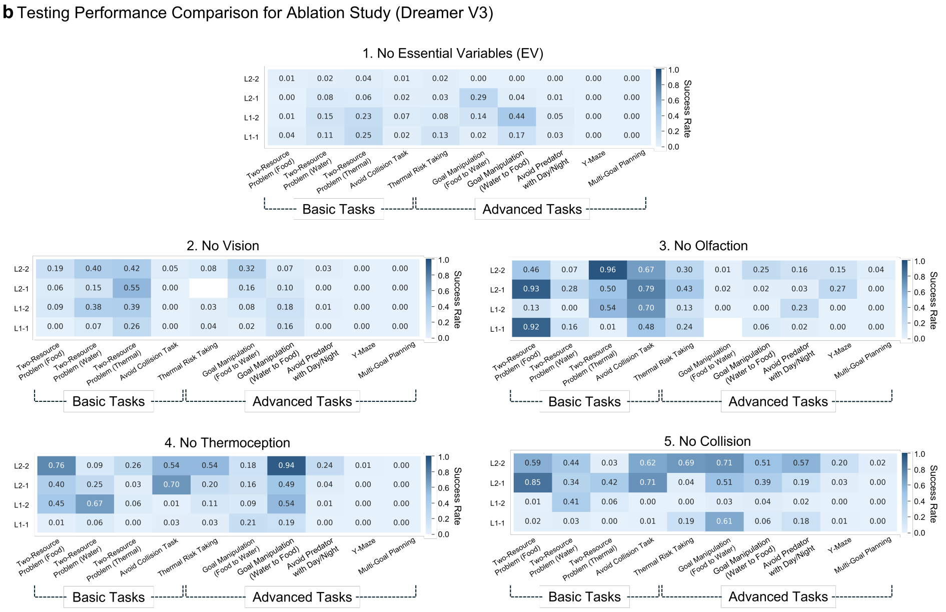


Figure 26: **Testing performance comparison for the modality ablation study.** Testing performance (success rate) on unseen test tasks for the five ablation conditions. Agents in the (1) 'No EV' and (2) 'No Vision' conditions fail to generalize. In contrast, agents in the (3) 'No Olfaction' condition generalize well, especially on Basic Tasks. The (4) 'No Thermoception' and (5) 'No Collision' conditions show markedly poorer generalization than the 'No Olfaction' group, confirming that thermoception and collision are more important for generalization than olfaction.