AMaze: a benchmark generator for sighted maze-navigating agents

Kevin Godin-Dubois [1¶], **Karine Miras** [1], **and Anna V. Kononova** [2]

1 Vrije Universiteit Amsterdam, The Netherlands **2** Leiden University, The Netherlands ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Yasmin Mzayek

Reviewers:

- @stheid
- @abhiramm7
- @inpefess

Submitted: 01 July 2024
Published: unpublished

Summary

The need to provide fair comparisons between agents, especially in the field of Reinforcement Learning, has led to a plethora of benchmarks. While these are devoted to tailor-made problems, they offer with very little degrees of freedom for the experimenter. AMaze is instead a benchmark *generator* capable of producing human-intelligible environments of arbitrarily high complexity. By using visual cues in a maze-navigation task, the library empowers researchers across a large range of fields.

Statement of need

AMaze is a pure-Python package with an emphasis on the easy and intuitive generation, evaluation and analysis of mazes. Its primary goal is to provide a way to quickly generate mazes of targeted difficulty, e.g., to test a Reinforcement Learning algorithm. By modeling loosely embodied robots with three distinct input/output spaces, AMaze makes it possible to prototype agent-centric scenarios of decision making, pattern recognition and general behavior through exposition to a wide array of contexts.

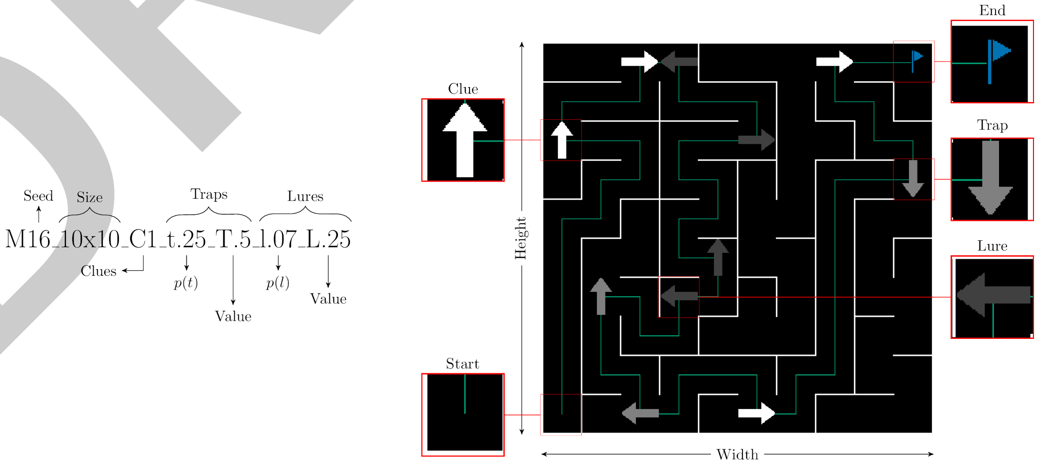

Figure 1: A sample maze from the AMaze library. In the API, every maze can be converted to and from a human-readable string where each underscore-separated component describes one of its facets. The *seed* seeds the random number generator used for the paths and stochastic placement of *lures* and *traps*. These have a specific probability, shape and/or value and may be specified multiple times to increase the complexity, as described in the documentation[1]

[1] https://amaze.readthedocs.io/en/latest/

Features

Users of AMaze have two main components to take into consideration: mazes and agents. These are introduced below with more details available in the documentation.

Mazes

Mazes can be described by human-readable string as illustrated in Figure 1, where every component is optional. The *seed* is used in the random number generator responsible for: a) the depth-first search that creates the paths and b) the stochastic placement of the *lures* and *traps*. As will be detailed below, agents only see a single cell at a time making intersections impossible to handle without additional information. *Clues* provide such an information by helpfully pointing towards the correct direction. However, users may additionally specify the presence of *traps*, at a given frequency, to replace a clue at an intersection. Traps always point towards the wrong direction thereby forcing agents to discriminate between the two. Furthermore, there is a lighter class of negative sign, namely *lures*, which occur outside of intersection and unhelpfully point towards an obviously bad direction (e.g. a wall).

Mazes can broadly be grouped into classes according to the features they exhibit. The most *trivial* cases correspond to mazes with a single path (enforced by removing intersections). When intersections are labeled with appropriate clues, mazes are considered as *simple*. Additionally, exhibiting either lures or traps form the corresponding classes while the more general case with all types of signs is labeled as *complex*. To accurately compare between different types of mazes across multiple categories, the library provides, for any given maze M, two dedicated metrics, the surprisingness S_M and deceptiveness D_M defined as follows:

$$
\begin{aligned}
S_M &= -\sum_{i \in I_M} p(i) * log_2(p(i)) \\
D_M &= \sum_{c \in \text{cells}(M)} \sum_{\substack{s \in \text{traps}(M) \\ s[0:3]=c}} -p(s|c) log_2(p(s|c))
\end{aligned}
$$

which, informally, account for the likelihood of encountering different states (walls, signs) and different *variations* of a given cell (same walls, different signs). Through these metrics, experimenters can make an informed decision about the level of complexity of the mazes they use. As illustrated by the distributions of S_M and D_M, sampled from 500'000 mazes across all five classes (Figure 2), the space of all possible mazes is both diverse and arbitrarily complex.

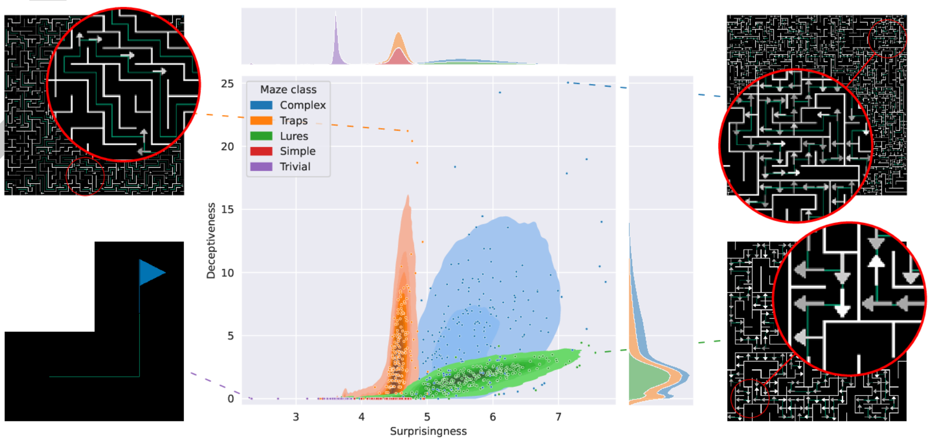

Figure 2: Distribution of Surprisingness S_M versus Deceptiveness D_M across 500'000 unique mazes from all five different classes. Outlier mazes are depicted in the borders to illustrate the underlying Surprisingness (right column) or lack thereof (left column).

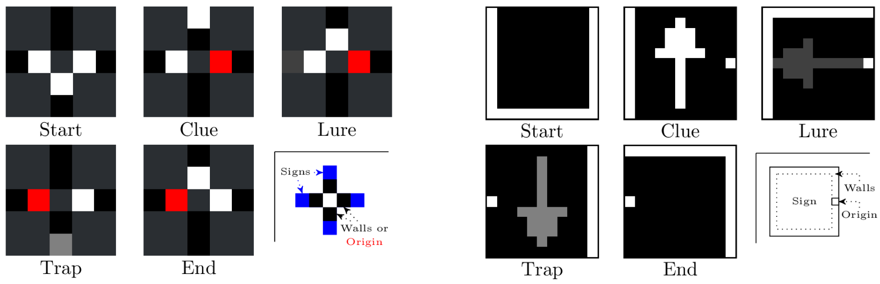

Figure 3: Discrete (left) and continuous (right) inputs for the examples shown in Figure 1. The former is solely used for the fully discrete case while the latter covers both hybrid and fully continuous cases.

Agents

Agents in AMaze are loosely embodied robots that wander around mazes perceiving only local information (the cell they are in) and a one-item memory (the direction they come from, if any). To accommodate various use cases, these agents come in three different forms: fully discrete, fully continuous and hybrid. In the former case, an agent has access to something akin to a pre-processed input, as in Figure 3, where the first four fields describes the wall configuration and the remainder provide information about signs, if any. These can be distinguished through their luminosity as agents only perceive grayscale values. These observations are used to deduce the correct action out of the four cardinal directions.

In the hybrid case, actions are identical while observations are coarse-grained images, of configurable size (e.g., 11x11 in Figure 3), where walls are indicated by pixels on the perimeter. The temporal information of the previous direction is still provided, as a single white pixel centered on the appropriate side. More importantly, the center of the image is used to display an arbitrary shape as a sign (clue, lure or trap). Finally, the fully continuous case is characterized by having the robot control its acceleration. Thus, the agent must also infer and take into consideration its position and inertia.

Comparison to existing literature

AMaze differs from existing benchmarks on two important aspects:

- *Computational efficiency* when compared to alternative vision-based tasks
- *Extensive control* over the environment and *intuitive understanding* of an agent's behavior

The former relates to the underlying LUT-based generation of visual information which alleviates the need for expensive rendering techniques. Through having only array pointers moving around, AMaze was designed to have fast-running simulations while still being directly usable with traditional architectures such as CNNs. On the latter point, the API allows precise tuning of many of a maze's characteristics, in addition to random exploration. Additionally, as an agent behavior is a 2D trajectory in a maze, it is very straightforward for a human observer to interpret its behavior and determine what went right or wrong, and when.

To illustrate the initial statements, we compare AMaze to a sample of benchmark suites (Figure 4). This includes gymnasium (Towers et al., 2023), an ubiquitous benchmark suite in the Python ecosystem; Lab2D (Beattie et al., 2020), a grid-world environment with both text and script parametrization; and Maze Explorer (Harries et al., 2019), a customizable 3D maze platform based on the DOOM video-game. Indeed, while mazes are commonly used as evaluation environments in Machine Learning (Lehman & Stanley, 2008; Miconi et al., 2018) they are often ad-hock solutions, deeply tied to a specific framework as in Beattie et al. (2016).

The test uses 81 variations of AMaze with different input image sizes (11x11, 15x15, 21x21), maze sizes (5, 10, 20), lure frequencies (0, 0.5, 1), and observation and action spaces (discrete,

JOSS
The Journal of Open Source Software

hybrid and continuous). This diversity of environment types was generated to give sufficient data for a fair comparison while also showcasing the ease with which AMaze can create feature-specific sets of mazes e.g. for benchmarking purposes. In the figure, N is the number of unique environments used/provided by the library and Time is measured on 1000 time steps averaged over 10 replicates on an i7-1185G7 (3GHz). Discrete inputs are enumerable and finite while Continuous uses decimal values. Images can fall in either categories, but are characterized by a high number of inputs.

Family	N	Inputs	Outputs	Control	Time (s) Median
Toy Text	5	Discrete	Discrete	None	0.009
Classic Control	5	Continuous	Both	None	0.023
AMaze	**192**	**Both**	**Both**	**Extensive**	**0.025**
Lab2D	11	Both	Discrete	Lua	0.056
Mujoco	11	Continuous	Continuous	None	0.085
Box2D	5	Continuous	Both	None	0.151
ALE	104	Image	Discrete	Modes	0.400
MazeExplorer	81	Image	Discrete	Extensive	0.553

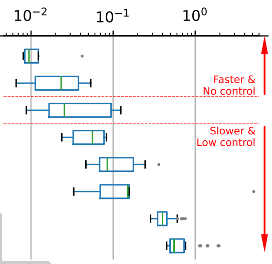

Figure 4: Comparison of AMaze with gymnasium's environments suite. Inputs, Outputs and amount of human Control are taken from the documentation while Time is measured on 1000 timesteps averaged over 10 replicates. AMaze is more computationally efficient than all but the simplest environments while also being the highly parametrizable.

Control describes how a human experimenter can specify, or at least influence, environmental features to suit their needs. Thus None implies fixed environments (most common) while various libraries use different methods to allow for customization such as the Lua scripting language (Lab2D), built-in Modes (ALE) or hand-made maps (Toy Text, Frozen Lake only). Extensive control requires a streamlined way to generate feature-specific custom environments with dense visual information.

In terms of computational speed, while taking more time than Classical Control tasks (Barto et al., 1983) or Toy Text environments (Sutton & Barto, 2018), AMaze is demonstrably faster than those based on 2D (Box2d) or 3D (MuJoCo, Todorov et al. (2012)) simulators or the Arcade Learning Environment (Bellemare et al., 2013).

Given the broad range of generated environments, this comparison demonstrates how competitive the library is compared to existing alternatives with respect to its execution speed and customizability.

Acknowledgements

This research was funded by the Hybrid Intelligence Center, a 10-year programme funded by the Dutch Ministry of Education, Culture and Science through the Netherlands Organisation for Scientific Research, https://hybrid-intelligence-centre.nl, grant number 024.004.022.

References

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC-13*(5), 834–846. https://doi.org/10.1109/TSMC.1983.6313077

Beattie, C., Köppe, T., Duéñez-Guzmán, E. A., & Leibo, J. Z. (2020). *DeepMind Lab2D*. https://github.com/deepmind/lab2d

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., … Petersen, S. (2016). *DeepMind Lab*. https://arxiv.org/abs/1612.03801v2

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, *47*, 253–279. https://doi.org/10.1613/jair.3912

Harries, L., Lee, S., Rzepecki, J., Hofmann, K., & Devlin, S. (2019). MazeExplorer: A Customisable 3D Benchmark for Assessing Generalisation in Reinforcement Learning. *2019 IEEE Conference on Games (CoG)*, *2019-Augus*, 1–4. https://doi.org/10.1109/CIG.2019.8848048

Lehman, J., & Stanley, K. O. (2008). Exploiting Open-Endedness to Solve Problems Through the Search for Novelty. *Artificial Life XI*, *Alife Xi*, 329–336.

Miconi, T., Clune, J., & Stanley, K. O. (2018). Differentiable plasticity: Training plastic neural networks with backpropagation. *35th International Conference on Machine Learning, ICML 2018*, *8*, 5728–5739. ISBN: 9781510867963

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press. ISBN: 9780262039246

Todorov, E., Erez, T., & Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. https://doi.org/10.1109/IROS.2012.6386109

Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G. de, Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Shen, A. T. J., & Younis, O. G. (2023). *Gymnasium*. https://doi.org/10.5281/zenodo.8269265