MazeExplorer: A Customisable 3D Benchmark for Assessing Generalisation in Reinforcement Learning

Luke Harries*, Sebastian Lee*, Jaroslaw Rzepecki, Katja Hofmann, Sam Devlin
Microsoft Research, Cambridge, UK
*Joint first author

***Abstract*—This paper presents a customisable 3D benchmark for assessing generalisability of reinforcement learning agents based on the 3D first-person game Doom and open source environment VizDoom. As a sample use-case we show that different domain randomisation techniques during training in a key-collection navigation task can help to improve agent performance on unseen evaluation maps.**

***Index Terms*—Artificial intelligence, Machine learning, Machine learning algorithms**

I. Introduction

In reinforcement learning (RL), it is common to evaluate agent performance in the same environments as those in which the agent is trained. The lack of separation between training and test environments, as is otherwise the norm in machine learning, can lead to brittle policies that do not generalise to new or even slightly modified environments (i.e. overfitting to training experiences). This shortcoming of RL research practices has been increasingly acknowledged and assessed in recent work [1]–[4]. Its solution will entail – among other components – environment frameworks in which it is easy to perform reproducible, statistically significant, customisable, and principled experimentation.

As both emphasis on theoretical justification of different experimentation configurations heightens (across aspects such as reproducibility and statistical best practices) and RL algorithms are brought to bear in real-world applications, such as robotics, where overfitting the training environment will not be feasible (e.g. due to safety or cost), we anticipate that generalisation will be a key factor in future RL research directions.

For these reasons, we introduce in this paper a new RL benchmark environment, MazeExplorer, based around procedural generation of 3D navigation tasks in the first person game Doom. Installation via pip, a gym environment interface, and a high degree of customisation make this a flexible and easy to use package for researchers to experiment with. Additionally, we present results of domain randomisation of maps during training as an example use-case of our package.

II. Related Work

The concept of benchmarks is integral to RL research. In the modern deep RL literature, widely used benchmarks include the arcade learning environment [5] and MuJoCo [6]. Although the decision to train and evaluate on the same maps ultimately belongs to the experimenter and is not an inherent feature of these environments, in practice they have inflexible APIs that are clearly not designed to be used to generate split train/test configurations; as such their use often leads to research that suffers from the overfitting problems outlined above.

More recently, DeepMind Lab, CoinRunner, and the Sonic Benchmark are all examples of environments designed for use in generalisable RL research [7]–[9]. They all employ randomisation in the form of procedural generation. Our work differs from those of CoinRunner and Sonic in that it is built on Doom - a 3D rather than 2D environment. 3D environments provide more challenging control problems and will be more applicable to real world applications (e.g. robotics) than 2D environments. Doom can also operate at up to 7000 fps and permits extensive customisation of maps (including map dimensions and specification of extent of randomisation), allowing for faster and more configurable experimentation than would be possible with the higher fidelity DeepMind Lab [10]. The benchmark tasks proposed here can also be seen to complement those in the minecraft-based Project Malmö [11], as they instead emphasise multi-agent learning [12].

Our domain randomisation investigation is influenced by previous work [13], and is applied here to tasks in our environment as a demonstration of what we envision typical use-cases of our package to be. Our experimentation approach - and the general philosophy of the package - is based on principles outlined in Henderson et al. [1].

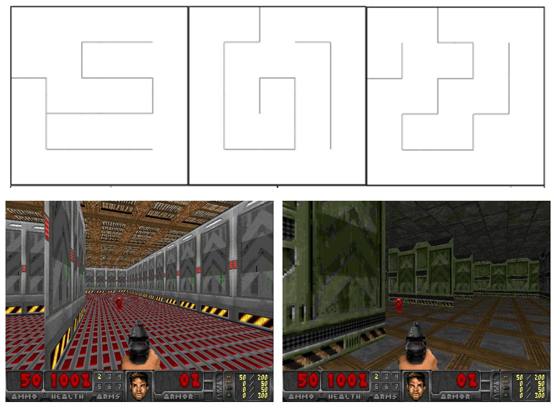
Fig. 1: *Top*: Example 10×10 mazes generated using NavDoom from a top-down view. *Bottom*: Example POVs of the agent in the MazeExplorer navigation task with randomised textures.

III. MazeExplorer

The primary motivation of the MazeExplorer environment is to evaluate the ability of trained agents to generalise. In each MazeExplorer mission, the goal is to navigate a procedurally generated maze and collect a set number of keys. MazeExplorer follows the OpenAI Gym interface and allows for a high level of customisation. An example of the Python interface is shown in Appendix A. The following subsections will outline some package details and the high-level API.

A. VizDoom

MazeExplorer generates navigation missions for VizDoom, a Doom-based platform for developing AI agents that learn purely from visual information [14]. VizDoom is well suited for RL research as it is multi-platform, fast (up to 7000 fps) and memory efficient (a few MBs).

Researchers typically create custom maps (WAD format) for their agents using GUIs such as DoomBuilder [15] or Slade [16], or by scripting with the python OMGIFOL library [17], and add custom tasks via *game events* using the domain specific language ACS [18]. Our aim with MazeExplorer is to abstract this time-intensive process without compromising customisability.

B. Procedural Maze Generation

We use NavDoom [19] to procedurally generate a user-defined number of distinct training and test mazes. The mazes are guaranteed to have a solution though will often have more. Fig. 1 shows top-down schematics of example generated mazes and agent points of view (POV) for random textured levels.

C. Experiment Configuration Options

The primary configurable aspects of MazeExplorer are map dimension and complexity, number of maps, randomisation of spawn and key positions, and randomisation of textures. A full list of configuration options can be found in Appendix B.

IV. Benchmarks

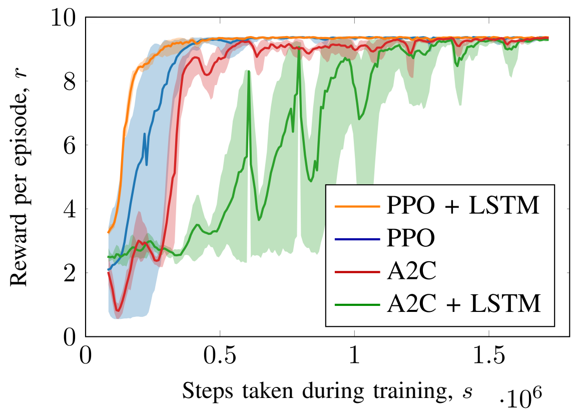

Fig. 2: Learning curves for PPO, A2C, PPO+LSTM and A2C+LSTM, smoothed over 8000 steps, shown with min and max curves to outline robustness. All four algorithms converge on all three repeats in fewer than 2×10^6 steps (Max r is 9).

To illustrate how our package could be used in research, we present the performance of A2C [20] and PPO [21] (with and without LSTMs [22]) on an example navigation task from MazeExplorer and go on to show the effects of domain randomisation of maps during training on generalisation. All experiments were done using the open source OpenAI stable-baselines to facilitate reproducibility [23]. In all of the experiments below, every map (training and evaluation) contains 9 keys and the agent receives a reward of +1 for picking up each key. We use an episode timeout of 2100 steps, action frame repeat of 4, and frame stack of 4.

A. Overfitting

We begin by assessing A2C and PPO on the simplest MazeExplorer configuration - a single (10×10) map with fixed spawn and key positions, and fixed wall, ceiling and roof textures. Fig. 2 shows the training curves of A2C, PPO, A2C + LSTM and PPO + LSTM. All 4 policies clearly converge on this simple task.

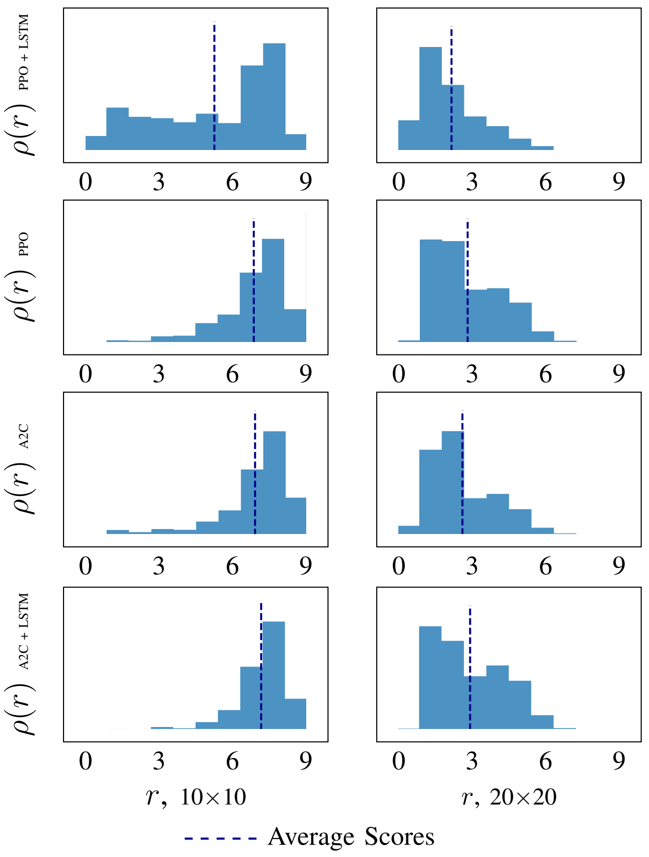

Fig. 3: Probability distribution of reward scores over 1000 rollouts on six different evaluation maps, $\rho(r)$, vs. reward score, r, for the learned policies taken from Fig. 2. Scores on the 10×10 maps are better than those on 20×20 maps but performance on all is noticeably worse than in training.

However, if we port these learned policies onto a different map (an evaluation map), they do not perform well. This is a simple yet illustrative example of the overfitting problem outlined at the start of this paper and can be seen graphically in Fig. 3, which shows the distributions of 6 evaluation map scores (3 at 10×10 and 3 at 20×20) over multiple rollouts of the trained policies from Fig. 2.

Fig. 2 and Fig. 3 show that convergence in a training task in RL is not indicative of having learned a generally applicable policy. Interestingly, on average the least robust policy in terms of learning (A2C + LSTM), generalises best to new maps, and the most robust (PPO + LSTM), generalises the poorest. This suggests that what one would typically associate with 'efficient' (i.e. fast) learning of a task, can lead to greater over-fitting, perhaps because the agent sees a less diverse dataset before having converged on an optimal policy for the specific task configuration on which it is being trained.

We can show more explicitly how the evaluation performance compares to the training performance by doing periodic rollouts of the policy during training. These results for PPO + LSTM are shown in Fig. 4.

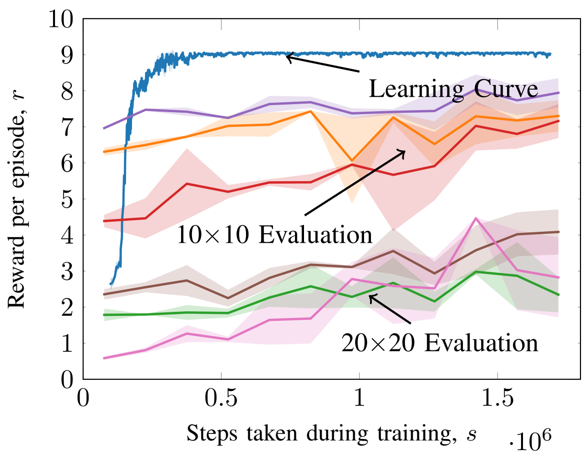

Fig. 4: For PPO + LSTM: The top curve is the learning curve (rewards on the training map) the rest are evaluation curves (reward on unseen evaluation maps) - the middle three (purple, orange, red) are on 10×10 maps and the bottom three (green, pink, brown) are on 20×20 maps. Evaluation performance is markedly worse than training performance.

B. *Domain Randomisation*

In the experiment above, the agent has learned a mapping from a specific path through a static map to the reward signal. What we want the agent to learn is a mapping from seeking and collecting red keys to the reward signal. One way to encourage a better mapping is to add randomisation to the map configurations seen by the agent during training. This can be done in multiple ways, all of which MazeExplorer supports:

- **Multiple maps**: rather than using a single map during training, each episode randomly selects one of an ensemble of training maps with different layouts.
- **Random spawn**: rather than having the agent start in the same position in a given map, the spawn position and angle can be randomised. This forces exploration while making it harder for the agent to memorise a route through the map.
- **Random textures**: rather than having constant wall colours on a given map we can randomise the texture of the walls, ceiling and floor in each new episode.
- **Random keys**: rather than fixing key positions we can randomise their positions in each new episode.

Sec. IV-A suggests that the overfitting problem is independent of algorithm choice, so from this point we perform our experiments with PPO + LSTM (empirically the most robust for this task) only. We train policies using every combination of the configurations outlined below, which makes $2 \times 2 \times 2 = 8$ different configurations. (We keep the textures constant in the first set of experiments).

Number of Maps	Random Spawn	Random Keys
1, 10	Yes/No	Yes/No

Table I below summarises the results of these experiments, which were each run with 3 different random seeds and evaluated on the same six maps as those used in Sec. IV-A. Plots in the format of Fig. 4 for each of these configurations are omitted for brevity but will be available in the README along with the code release. All variants follow the pattern that 10×10 evaluation performance is worse than training performance, and 20×20 evaluation performance is worse still. Increased randomisation narrows this gap.

Experiment Configuration			Mean Evaluation Scores	
Maps	Spawn	Keys	10×10	20×20
Untrained (Random) Agent			2.32 ±1.20	0.60 ±0.64
1	Fixed	Fixed	5.26 ±2.59	1.89 ±0.84
1	Fixed	Random	7.49 ±0.24	2.76 ±0.17
1	Random	Fixed	7.68 ±0.37	3.50 ±0.42
1	Random	Random	8.16 ±0.21	4.05 ±0.45
10	Fixed	Fixed	8.18 ±0.43	4.20 ±0.45
10	Fixed	Random	8.01 ±0.43	4.07 ±0.41
10	Random	Fixed	8.31 ±0.21	4.34 ±0.66
10	Random	Random	**8.34** ±0.26	**4.29** ±0.43

TABLE I: Domain Randomisation Ablation Results

The learned agent generalises better to the previously unseen evaluation maps when aspects of the training are randomised and it is no longer possible simply to memorise a route through the map. The agent must now learn more about the general task we want it to learn. In particular, training on multiple maps appears to give the most significant boost in performance. On average the best performing agent with randomisation collects 3.08 and 2.45 more keys than the agent trained in completely fixed maps on the 10×10 and 20×20 evaluation maps respectively.

V. Conclusion

MazeExplorer provides a highly customisable benchmark for assessing generalisation in RL research. As part of our domain randomisation experiments, we demonstrate some of that customisability (random spawn, random keys, multi-map training). In doing so we have exposed the poor generalisability of state-of-the-art RL algorithms alone (i.e. without specific methods such as domain randomsiation), particular when test maps are larger than those on which agents are trained.

Outside of those highlighted in the above sections, there are two further features of the package that we envision being particularly useful for the community. The first is the ability to randomise textures. This adds a layer of difficulty to the learning task, our experiments with the vanilla stable baseline implementations of PPO and A2C were unable to learn even static configurations of the navigation task with randomised textures. We thus propose this itself as a challenge to the community, especially with emphasis on generalisability. The second is simply the idea that by limiting the number of keys to 1, MazeExplorer provides a very good platform for testing the exploration capabilities of algorithms in sparse reward settings.

We hope that in future, together with contributions from the open source community, further features can be added to MazeExplorer. These may include: multiple map training over varying size/complexity mazes to facilitate curriculum style learning, more sophisticated map structures (e.g. multiple floors), and flickering observations or sticky actions for use in research on stochastic environments.

References

[1] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[2] C. P. . K. Gao. https://bair.berkeley.edu/blog/2019/03/18/rl-generalization/, 2019. [Online; access 28-03-19].

[3] N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi, "Procedural level generation improves generality of deep reinforcement learning," *arXiv preprint arXiv:1806.10729*, 2018.

[4] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "General video game ai: Competition, challenges and opportunities," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[5] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[6] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.

[7] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, *et al.*, "Deepmind lab," *arXiv:1612.03801*, 2016.

[8] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," *arXiv:1812.02341*, 2018.

[9] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman, "Gotta learn fast: A new benchmark for generalization in rl," *arXiv:1804.03720*, 2018.

[10] E. Beeching, C. Wolf, J. Dibangoye, and O. Simonin, "Deep reinforcement learning on a budget: 3d control and reasoning without a supercomputer," *arXiv:1904.01806*, 2019.

[11] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, "The malmo platform for artificial intelligence experimentation.," in *IJCAI*, pp. 4246–4247, 2016.

[12] D. Perez-Liebana, K. Hofmann, S. P. Mohanty, N. Kuno, A. Kramer, S. Devlin, R. D. Gaina, and D. Ionita, "The multi-agent reinforcement learning in malmö (marlö) competition," *arXiv:1901.08129*, 2019.

[13] D. S. Chaplot, G. Lample, K. M. Sathyendra, and R. Salakhutdinov, "Transfer deep reinforcement learning in 3d environments: An empirical study," in *NIPS Deep Reinforcemente Leaning Workshop*, 2016.

[14] M. Wydmuch, M. Kempka, and W. Jaśkowski, "Vizdoom competitions: playing doom from pixels," *IEEE Transactions on Games*, 2018.

[15] P. vd Heiden. http://doombuilder.com/. [Online; access 28-March-2019].

[16] S. Judd. http://slade.mancubus.net/. [Online; access 13-05-19].

[17] F. Johansson. http://omgifol.sourceforge.net/. [Online; access 13-05-19].

[18] R. Software. https://zdoom.org/wiki/ACS. [Online; access 13-05-19].

[19] I. J. Lee, "Navdoom." https://github.com/agiantwhale/NavDoom, 2018.

[20] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] A. Hill, A. Raffin, M. Ernestus, A. Gleave, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines." https://github.com/hill-a/stable-baselines, 2018.

Appendix A
API Example

MazeExplorer was designed to enable seamless generation of a train/test map split. Consider for example the case in which one wants to test the ability of an agent to generalise to new random textures, MazeExplorer makes it easy to generate distinct test maps with their own configurations as shown below.

```
train_env = MazeExplorer(number_maps=1,
            size=(15,15),
            random_spawn=True,
            random_textures=False,
            keys=6)

eval_env = MazeExplorer(number_maps=10,
            size=(15,15),
            random_spawn=True,
            random_textures=True,
            keys=6)

#train
for _ in range(1000):
  obs, rewards, dones, info = train_env.step(
     training_model.action)

#evaluation
for _ in range(1000):
  obs, rewards, dones, info = eval_env.step(
     trained_model.action)
```

Appendix B
List of MazeExplorer Configurations

When generating an instance of MazeExplorer, the following arguments can be configured:

- Number of maps
- Map Size (X×Y)
- Maze complexity
- Maze density
- Random/Fixed keys
- Random/Fixed textures
- Random/Fixed spawn
- Number of keys
- Environment Seed
- Episode timeout
- Reward clipping
- Frame stack
- Resolution
- Data augmentation
- Action frame repeat
- Actions space
- Specific textures (Wall, ceiling, floor)