Obstacle Tower: A Generalization Challenge in Vision, Control, and Planning

Arthur Juliani[1], Ahmed Khalifa[2], Vincent-Pierre Berges[1], Jonathan Harper[1],

Hunter Henry[1], Adam Crespi[1], Julian Togelius[2], Danny Lange[1]

[1]Unity Technologies, 30 3rd St, San Francisco, CA 94103
[2]New York University, Brooklyn, New York 11201
{arthurj, vincentpierre, jharper, brandonh, adamc, dlange}@unity3d.com,
ahmed@akhalifa.com, julian@togelius.com

Abstract

The rapid pace of research in Deep Reinforcement Learning has been driven by the presence of fast and challenging simulation environments. These environments often take the form of games; with tasks ranging from simple board games, to classic home console games, to modern strategy games. We propose a new benchmark called Obstacle Tower: a high visual fidelity, 3D, 3rd person, procedurally generated game environment. An agent in the Obstacle Tower must learn to solve both low-level control and high-level planning problems in tandem while learning from pixels and a sparse reward signal [1]. Unlike other similar benchmarks such as the ALE, evaluation of agent performance in Obstacle Tower is based on an agent's ability to perform well on unseen instances of the environment. In this paper we outline the environment and provide a set of initial baseline results produced by current state-of-the-art Deep RL methods as well as human players. In all cases these algorithms fail to produce agents capable of performing anywhere near human level on a set of evaluations designed to test both memorization and generalization ability. As such, we believe that the Obstacle Tower has the potential to serve as a helpful Deep RL benchmark now and into the future.

Introduction

It is crucial for the development of artificial intelligence methods to have good benchmark functions, so that the performance of different methods can be fairly and easily compared. For tree search and reinforcement learning methods, the benchmarks of choice have often been based on games. Classic board games such as Checkers and Chess were prominent in AI research since its inception and spurred the development of many important techniques (Turing 2004); for example, the first reinforcement learning algorithm was developed to play Checkers (Samuel 1959). With the advent of agents capable of superhuman performance in Othello, Checkers (Schaeffer et al. 1992), Chess (Hsu et al. 1990) and Go (Silver et al. 2016), the challenges of this type of games is increasingly seen as exhausted.

In the last two decades, video games have increasingly been used as AI benchmarks. In contrast to classic board games, video games require more frequent decision making, often in real-time settings, and additionally define more complex state spaces. They may or may not also have some combination of hidden information, stochasticity, complex interaction rules, and large branching factors. A number of benchmarks focus on classic 2D arcade games, as well as first-person shooters and racing games. These games all have limited branching factors, and the benchmarks built on them either make a low-dimensional processed observation of the environment available to the agent, or a fast forward model which allows for forward planning. This includes benchmarks based on Super Mario Bros (Karakovskiy and Togelius 2012), Minecraft (Johnson et al. 2016), Quake 3 (Hingston 2009; Beattie et al. 2016) and Pac-Man (Rohlfshagen et al. 2018). The General Video Game AI competition is a special case of this, where agents are tasked with playing unseen 2D arcade-style games (Perez-Liebana et al. 2016).

A new generation of video game-based AI benchmarks do not provide agents with processed representations of the environment, but instead forces them to act based on the raw pixels, i.e. the screen output. The popularity of such benchmarks go hand-in-hand with the advent of reinforcement learning using deep neural networks as function approximators, so called deep reinforcement learning, as these deep networks are capable of processing high-dimensional input such as screen images. In particular, the Arcade Learning Environment (ALE), which is based on an emulation of the Atari 2600 video game console, became one of the more widely used reinforcement learning benchmark after it was demonstrated that Deep Q-learning could learn to play many of these games at a human-competitive level (Bellemare et al. 2013; Mnih et al. 2015).

One of the games in particular, *Montezuma's Revenge*, has proved to be very hard for reinforcement learning algorithms because of the sparse rewards and partial observability. The notorious difficulty of this environment has promoted the development of a number of novel approaches to Deep RL algorithms. These include methods which focus on providing intrinsic reward signals based on state visitation or novelty (Bellemare et al. 2016; Burda et al. 2018a), methods for hierarchical agent control (Vezhnevets et al. 2017), and novel approaches to learning from demonstrations (Aytar et al. 2018). Despite the historical lack of significant progress on

AAAI-2019 Workshop on Games and Simulations for Artificial Intelligence.

[1]https://github.com/Unity-Technologies/obstacle-tower-env

Figure 1: Examples of agent observations in the Obstacle Tower at different floor levels. **Left** Early floor is rendered in the *Ancient* theme. **Middle** Intermediate floor is rendered using the *Moorish* theme. **Right** Later floor is rendered in *Industrial* theme.

Figure 2: Examples of floor layouts in the Obstacle Tower at different floor levels. **Left** Early floor is rendered in the *Ancient* theme. **Middle** Intermediate floor is rendered using the *Moorish* theme. **Right** Later floor is rendered in *Industrial* theme.

the environment, major progress has taken place in the past year, with the first level being solved by intrinsic-reward-augmented Deep RL approaches (Burda et al. 2018b), and the entire game being solved using the recently proposed Go-Explore, a tree search algorithm which focuses on exploration (Ecoffet et al. 2018).

The Atari 2600, on which ALE is based, is a very limited machine. It has 128 bytes of RAM, no video memory and games are typically 2 or 4 kilobytes of ROM; screen output is low-resolution 2D graphics. The lack of a system clock for seeding a pseudorandom number generator means that all games are deterministic. Having variability in the challenge, ideally through some kind of procedural content generation, is important for avoiding overfitting in reinforcement learning, and being able to evaluate what many AI researchers are actually interested in - agent generalization (Cobbe et al. 2018; Zhang et al. 2018). Arguably, targeting generalization is necessary in order to make progress on artificial general intelligence, rather than just solving individual problems (Schaul, Togelius, and Schmidhuber 2011).

Recognizing these limitations, several other game-based AI environments featuring raw pixel inputs have been proposed. The VizDoom competition and benchmark, based on the classic first-person shooter Doom is a prominent example (Kempka et al. 2016). While it features a first-person perspective and complex gameplay, the age of the game means that the graphics are about as primitive as 3D graphics go; further, the only kind of randomization is in enemy movement and item spawning, as the level topologies are fixed. Other recently introduced game-based AI benchmarks, such as the OpenAI Retro Challenge (Nichol et al. 2018), CoinRun (Cobbe et al. 2018), and Pommerman (Resnick et al. 2018) all feature various kinds of environment randomization. They are however limited to providing 2D environment representation and only simple navigation challenges.

Obstacle Tower was developed specifically to overcome the limitations of previous game-based AI benchmarks, offering a broad and deep challenge, the solving of which would imply a major advancement in reinforcement learning. In brief, the features of Obstacle Tower are:

- **Physics-driven interactions**. The movement of the agent and other objects within the environment are controlled by a real-time 3D physics system.
- **High visual fidelity**. The visual fidelity of the environment is much closer to photo-realistic than other platforms such as the ALE, VizDoom, or DeepMind Lab (Beattie et al. 2016) See Figure 1 for examples of the rendered scene provided to agents to learn from.
- **Procedural generation of nontrivial levels and puzzles**. Navigating the game requires both dexterity and planning, and the levels within the environment are procedurally generated, meaning that generalization is required to perform well on all instances of the task. See Figure 2 for

examples of possible floor layouts of various levels of the Obstacle Tower.

- **Procedurally varied graphics**. There are multiple levels of variation in the environment, including the textures, lighting conditions, and object geometry. As such, agents must be able to generalize their understanding of object visual appearance.

Obstacle Tower Environment

The Obstacle Tower environment uses the Unity platform and ML-Agents Toolkit (Juliani et al. 2018). It can run on the Mac, Windows, and Linux platforms, and can be controlled via the OpenAI Gym interface for easy integration with existing DeepRL training frameworks (Brockman et al. 2016).

Episode Dynamics The Obstacle Tower environment is consists of 25 floors, with the agent starting on floor zero. All floors of the environment are treated as a single finite episode in the Reinforcement Learning context. Each floor contains at the least a starting room where the agent is spawned, and a room with stairs to the next floor. Each room can contain a puzzle to solve, enemies to defeat, obstacles to evade, or a key to open a locked door. The layout of the floors and the contents of the rooms within each floor becomes more complex at higher floors in the Obstacle Tower. This provides a natural curriculum for learning agents, allowing them to utilize information gained on earlier floors to solve tasks at later floors. For more information on the generation of the floor and room layouts, see the "Procedural Generation of Floors" section below. Within an episode, it is only possible for the agent to go to higher floors of the environment, and not to return to lower floors.

Episodes terminate when the agent encounters a hazard such as a pit or enemy, when the timer runs out, or when the agent arrives at the top floor of the environment. The timer is set at the beginning of the episode, and competing floors as well as collecting time orbs increase the time left to the agent. In this way a successful agent must learn behaviors which trades off between collecting orbs and quickly completing floors of the tower in order to arrive at the higher floors before the timer ends.

Observation Space The observation space of the environment consists of two types of information. The first type of observation is a rendered pixel image of the environment from a third person perspective. This image is rendered in 168×168 RGB, and can be downscaled to the conventional 84×84 resolution image typically used in Deep RL pixel-to-control scenarios such as the Deep Q-Network (Mnih et al. 2015). The second type of observation is a vector of auxiliary variables which describe relevant, non-visual information about the state of the environment. The elements which make up this auxiliary vector are: number of keys agent is in possession of, as well as the time left in the episode.

Action Space The action space of the environment is multi-discrete, meaning that it consists of a set of smaller discrete action spaces, of which the union corresponds to a single action in the environment. These subspaces are as follows: forward/backward/no-op movement, left/right/no-op movement, clockwise/counterclockwise rotation of the camera, and no-op/jump. We also provide a version of the environment with this action space flattened into a single choice between one of 54 possible actions.

Reward Function The Obstacle Tower is designed to be a sparse-reward environment. The environment comes with two possible configurations: a sparse and dense reward configuration. In the sparse reward configuration, a positive reward of +1 is provided only upon the agent reaching the exit stairs of a floor of the tower. In the dense reward version a positive reward of +0.1 is provided for opening doors, solving puzzles, or picking up keys. In many cases even the dense reward version of the Obstacle Tower will likely resemble the sparsity seen in previously sparse rewarding benchmarks, such as Montezuma's Revenge (Bellemare et al. 2013). Given the sparse-reward nature of this task, we encourage researchers to develop novel intrinsic reward-based systems, such as curiosity (Pathak et al. 2017), empowerment (Mohamed and Rezende 2015), or other signals to augment the external reward signal provided by the environment.

Procedural Generation of Floors

Each floor of the Obstacle Tower environment contains procedurally generated elements which impact multiple different aspects of the agent's experience. These include procedurally generated lighting, room layout, as well as overall floor plan. Together these elements ensure that it is extremely unlikely that any two instances of the Obstacle Tower generated from two different random seeds contain even one identical floor. This proceduralism also ensures that for agents to do well on new instances of the Obstacle Tower, they must have learned some general purpose representations of the task at the levels of vision, low-level control, and high-level planning.

Visual Appearance On each floor of the Obstacle Tower various aspects of the appearance of the environment are generated procedurally. This includes the selection of a visual theme which determines the textures and geometry to be used, as well as a set of generated lighting conditions. Examples of visual themes include *Ancient*, *Moorish*, *Industrial*. The lighting conditions include the direction, intensity, and color of the real-time light in the scene.

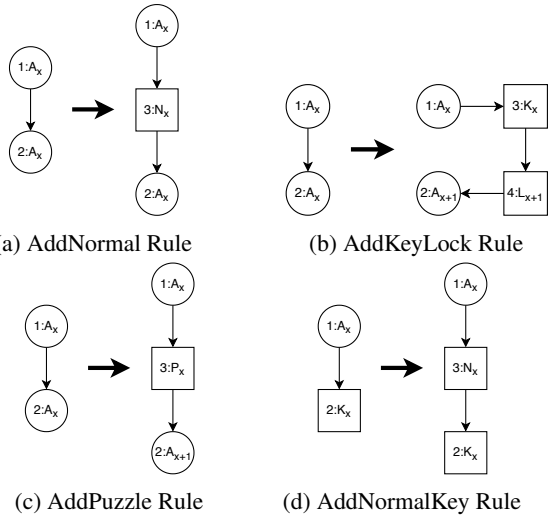
Floor layout The floor layout is generated using a procedure inspired by Dormans (Dormans 2010). The floor layout generation is divided into two parts: a mission graph and a layout grid.

The mission graph is responsible for the flow of the mission in the current level. For example: to finish the level the player needs to get a key then solve a puzzle then unlock the door to reach the stairs for the next level. Similar to Dormans, we used graph grammar which is a branch of generative grammar to generate the mission graph.

In Obstacle Tower, we constructed eight different graph grammar rules that can transform our starting graph (consists of two nodes $start$ and $exit$) to generate a range of

different floor configurations of varying difficulty levels. To understand the grammar, we must specify the node types that are being used. Also, each node is associated with an access level number. Access level determines the minimum number of locks or puzzles that need to be passed in order to reach that room. The following list all the different types of rooms:

- *Start (S):* is the starting node in the dungeon.
- *Exit (E):* is the end node in the dungeon that the player need to reach.
- *Key (K):* is a node that contains a key.
- *Lock (L):* is a node that can't be entered without a key.
- *Puzzle (P):* is a node that contain a puzzle/challenge that need to be passed to continue.
- *Lever (V):* is a node that contains a lever that need to be pressed to pass to the next room.
- *Normal (N):* any other node that doesn't follow any of the above.



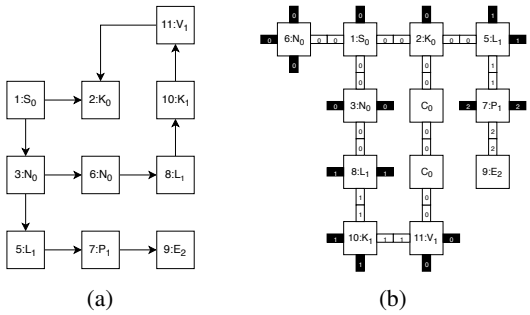
(a) AddNormal Rule (b) AddKeyLock Rule

(c) AddPuzzle Rule (d) AddNormalKey Rule

Figure 3: Four examples of Obstacle Tower mission graph rules.

Figure 3 shows four of the graph rules used during generation. The letter in the node specifies the node type, while the small subscript number is the access level. The first number is used to make a mapping between nodes in the grammar. Circular nodes are considered as wild card nodes which means it can match any node. These rules are applied on the starting graph (consists of two connected nodes with access level of zero of type $start$ and $exit$) using what is called a graph recipe. A graph recipe is a sequence of graph grammar rules that are applied after each other to generate a level. The recipe contains some randomness by allowing each rule to be applied randomly more than once. In Obstacle Tower, the system use different graph recipes for each group of levels to generate more complex floor layouts in later levels than the beginning levels.

After generating the mission graph, we need to transform that graph into 2D grid of rooms which is called layout grid.



(a) (b)

Figure 4: An example showing the generated mission graph in Obstacle Tower in figure 4a and its corresponding layout in figure 4b.

A simpler grammar called shape grammar (Stiny and Gips 1971) is used in the transformation process which is similar to Dorman's transformation process(Dormans 2010). For every node in the mission graph starting from the Start node, we allocate a room on the grid that connects to a previous room of same access level from any of the four directions (North, South, East, and West). After allocating all of the rooms and their correct connections (locks for locked connections, controlled doors for lever connections, etc), we transform any other connections to solid. At any point if mission graph or layout grid failed to generate a valid output, we just rerun the algorithm until it fits.

Figure 4 shows a mission graph generated by the system and its corresponding layout where black links means they turned to solid and C rooms are new rooms introduced to make sure that some of the connections in the level are possible (the lever node 11 back to the key node 2). As we can see the layout generation keeps track of all the possible connection with different access levels from each room as any room with the same access level can be connected. Another thing that is not shown in the figure is that the layout generator makes sure nodes have an access level parent, a node that changes the access level in the graph such as Lock, Puzzle, and Lever nodes. The algorithm make sure that if any node comes after any of these nodes, it is still connected to it through its parent.

Room layout For the generation of the layout of each room within a floor, we used a template-based system similar to that used in the popular video game Spelunky (Yu 2016). In this system each of the different room types, such as $Puzzle$ or Key have their own set of templates from which specific room configurations are drawn from. These templates consist of a grid of characters which represents the potential layout of the room. These grids can be either 3×3, 4×4, or 5×5. The specific placement of the modules and items within a room is based on these templates. The template can define the specific module or item to be placed in each position within the room, or define a category from which a specific module or item is drawn and placed probabilistically. In this way a finite number of templates can be used to generate a much larger number of possible room configurations.

Evaluation Criteria

It is essential that the evaluation of agent performance on environments such as the one described here be as reproducible and interpretable as possible. Below we provide a set of evaluation criteria to be used when benchmarking the performance of an agent within the Obstacle Tower. This criteria described here are inspired by a recent set of recommendations by Henderson and colleagues (Henderson et al. 2017).

Non-Generalization Evaluation It is possible to evaluate the performance of an agent on a single, fixed version of the Obstacle Tower. This evaluation regime may be of interest to researchers focusing on optimization and data throughput during training rather than the development of algorithms which provide agents with novel cognitive abilities. In this case we recommend explicitly reporting that the evaluation was performed on a fixed version of the Obstacle Tower, and also reporting performance on five random seeds of the dynamics of the agent. We performed training for 20 million environment time-steps, using the default hyperparameters provided for use with environments from the ALE.

Generalization Evaluation The Obstacle Tower is designed to explicitly test the generalization ability of agents. As such, we recommend evaluating the performance of agents on a held-out set of tower configurations which the agent is not exposed to during training.

- **Weak Generalization** Agents should be trained on a fixed set of 100 seeds for the environment configurations. They should then be tested on a held-out set of five randomly selected tower configuration seeds not in the training set. Each should be evaluated each five times using different random seeds for the dynamics of the agent (initial weights of the policy and/or value network(s)).
- **Strong Generalization** In addition to the requirements for weak generalization, agents should be tested on a held-out visual theme which is separate from the ones on which it was trained. In this paper we train on the *Ancient* and *Moorish* themes, and test on the *Industrial* theme.

Value as a Research Benchmark

The Obstacle Tower is designed to provide a meaningful challenge to current and future AI agents, specifically those trained using the pixels-to-control approach. There are four axes of challenge which we believe that this environment provides: Vision, Control, Planning, and Generalization. While various other environments and benchmarks have been used to provide difficult challenges for AI agents, this is to the best of our knowledge the first benchmark which combines all such axes of complexity.

Generalization As mentioned above, the Obstacle Tower relies heavily on procedural generation for the creation of each instance of the environment. This focus on proceduralism is designed with the goal of enabling the evaluation of the generalization abilities of agents. There have recently been a number of similar approaches to introducing specific challenges around generalization, such as the CoinRun environment (Cobbe et al. 2018) or the environments in the General Video Game AI challenge (Perez-Liebana et al. 2016).

Vision As described above, the main source of observation information to agents within the the Obstacle Tower is a rendered RGB image of the environment from a third person perspective. Unlike previous environments such as the ALE (Bellemare et al. 2013), VizDoom (Kempka et al. 2016), and Malmo (Johnson et al. 2016), which contain low resolution textures, simple 3D geometry, and simple lighting, the Obstacle Tower contains high-fidelity real-time lighting, complex 3D shapes, and high-resolution textures. All of this combined corresponds to a visual observation which better emulates that of the physical world than these other environments. Furthermore, the floors in the environment can be rendered in one of multiple different visual themes, such as *Ancient* or *Industrial*. The combination of high-fidelity visuals in addition to visual variation means that we expect models with much greater representational capacity than models used in previous approaches such as A3C (Mnih et al. 2016) or DQN (Mnih et al. 2015) will be needed to perform well at interpreting the visual information in the environment.

Generalization & Vision Humans can easily understand that two different doors seen under different lighting conditions are still doors. We expect that general-purpose agents should have similar abilities, however this is not the case. In many cases agents trained under one set of visual conditions, and then tested on even a slightly different visual conditions perform much worse at the same task. This can be seen in cases where slight perturbations of the RBG image observation provided to the agent result in dramatic decreases in performance (Huang et al. 2017). The procedural lighting and visual appearance of floors within the Obstacle Tower means that agents will need to be able to generalize to new visual appearances which they may never have directly experienced before.

Control In order for the agent to perform well on the Obstacle Tower environment, it must be able to navigate through multiple rooms and floors. Each of these rooms can contain multiple possible obstacles, enemies, and moving platforms, all of which require fine-tuned control over the agent's movement. Floors of the environment can also contain puzzle rooms, which involve the physical manipulation of objects within the room in order to unlock doors to other rooms on the floor. We expect that in order for agents to perform well on these sub-tasks, the ability to model and predict the results of the agents actions within the environment will be of benefit.

Generalization & Control The layout of the rooms on ever floor are different on each instance of the Obstacle Tower, as such we expect methods which are designed to exploit determinism of the training environment, such as Brute (Machado et al. 2017) and Go-Explore (Ecoffet et al. 2018) to perform poorly on the test set of environments. It is also the case that within a single instance of a Tower, there are elements of the environment which contain stochastic behavior, such as the movement of platforms and enemies.

Planning Depending on the difficulty of the floor, some floors of the Obstacle Tower require reasoning over multiple dependencies in order to arrive at the end room. For example, some rooms cannot be accessed without a key that can only be obtained in rooms sometimes very far from the door they open, thus requiring planning the path to take appropriately in order not to waste the limited time the agent has.

Generalization & Planning Due to the procedural generation of each floor layout within the Obstacle Tower, it is not possible to re-use a single high-level plan between floors. It is likewise not possible to re-use plans between environment instances, as the layout of each floor is determined by the environment's generation seed. Because of this, planning methods which require computationally expensive state discovery phases are likely not able to generalize to unseen floor layouts.

Preliminary Results

In order to analyze the difficulty, and by extension the value of the Obstacle Tower as a benchmark, we conducted a preliminary evaluation of the performance of both humans and learned agents within the environment. We evaluated performance within three distinct conditions, each designed to provide insight into the level of generalization ability that the human or agent possesses. We conducted this evaluation on a version of the Obstacle Tower which contains a maximum of 25 floors, and a limited subset of visual themes, floor configurations, and object types. We performed evaluation within the three conditions described under "Evaluation Criteria:" *No Generalization* - training and testing on the same fixed environment, *Weak Generalization* - training on testing on separate sets of environment seeds, and *Strong Generalization* - training and testing on separate sets of environment seeds with separate visual themes.

Human Performance

In order to understand the expected quality of performance from a human-level agent, we conducted a series of evaluations with human play-testers. These were drawn from a pool of Unity Technologies employees who volunteered to participate in the evaluation process. Overall fifteen participants took place in the evaluation. For human evaluation, training corresponds to the initial five minutes of playtime.

Condition	Train	Test	Test (Max)
No Gen.	15.2 (2.9)	15.2 (2.9)	22
Weak Gen.	12.3 (2.9)	15.6 (3.5)	21
Strong Gen.	12 (6.8)	9.3 (3.1)	20

Table 1: Results of human evaluation on under different conditions. Performance results under *Train* and *Test* are reported as the *mean (std)* of the number of floors solved in a single episode. Results reported under *Test (Max)* correspond to maximum floor reached by a participant in each condition.

See Table 1 for human performance results. In the *No Generalization* and *Weak Generalization* conditions humans were able to solve an average of 15 floors during the test phase, with no different in performance between the two conditions. Human participants performed slightly worse in the *Strong Generalization condition*, however were still able to solve up to 20 floors in this condition as well. As expected, these results suggest that humans are able to transfer their knowledge gained during training time to perform consistently on the same environment, as well new configurations of the environment not seen during the training phase. In fact we find that human performance increases between the training and testing phases due to the ability of humans to continue to rapidly learn from small amounts of data.

Agent Performance

In addition to evaluating the performance of human players, we also evaluated the performance of agents trained using Deep RL. In particular we utilized the OpenAI Baseline implementation of of Proximal Policy Optimization (PPO) (Schulman et al. 2017; Dhariwal et al. 2017) as well as the implementation of Rainbow provided by Google Brain's Dopamine library (Hessel et al. 2018; Castro et al. 2018)[2]. These two were chosen for being the standard implementations of current state of the art on-policy and off-policy algorithms.

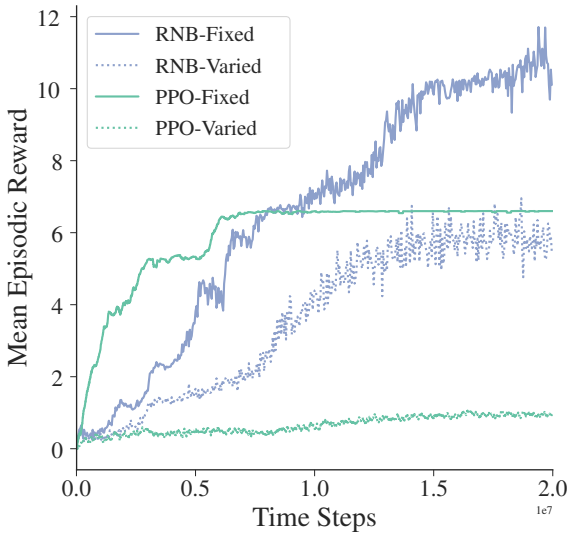


Figure 5: Mean episodic reward received during training by agent trained using OpenAI Baseline PPO (PPO) and Dopamine Rainbow (RNB) in the Fixed and Varied training conditions.

We performed training under the *No Generalization* (Fixed) and *Weak Generalization* (Varied) conditions, and

[2] https://github.com/openai/baselines; https://github.com/google/dopamine

performed evaluation on all three conditions. We utilized the default hyperparameters provided by each library for use with Atari benchmarks, in order to provide comparable results with evaluations performed on the ALE. We utilize the ability to run parallel environments within OpenAI Baselines to collect data from 50 concurrently running environments. In the case of Rainbow we collect data from a single environment running serially.

We conducted training sessions spanning 20 million and 10 million environment steps for PPO and Rainbow, respectively. See Figure 5 for graphs of the mean reward during training of the two algorithms in both the varied and fixed conditions. We find that agents trained using either algorithm are able to solve fewer than 10 floors in both the fixed and varied conditions. While both algorithms trained in the fixed condition produce similar performance scores, the discrepancy between training performance under the two conditions is much greater with PPO compared to Rainbow. In the case of PPO in the varied condition the agent achieves less than a reward of 1, whereas the Rainbow agent trained in the varied condition receives an average reward of 5 by the end of training.

Condition	PPO (F)	PPO (V)	RNB (F)	RNB (V)
No Gen.	**5.0 (0.0)**	1.0 (0.0)	**5.0 (0.0)**	**5.0 (0.0)**
Weak Gen.	1.2 (0.4)	0.8 (0.4)	0.6 (0.8)	**3.2 (1.1)**
Strong Gen.	0.6 (0.8)	0.6 (0.5)	0.0 (0.0)	**1.6 (0.5)**

Table 2: Results comparing trained models on three evaluation conditions. "F" corresponds to fixed training environment (one environment seed). "V" corresponds to varied training environment (100 environment seeds). Performance results are reported as the *mean (std)* of the number of floors solved in a single episode.

When the models described above were benchmarked in the three evaluation conditions, we find that they consistently perform poorly compared to the human results, failing to reach even an average floor completion score of 6. It is at floor 5 that the room mechanic of locked doors is introduced. We find that agents are unable to solve this subtask, and therefore are no longer able to make progress in the tower. See Table 2 for the full set of results.

Surprisingly, we find that in the case of PPO, agents trained on the fixed version of the environment are able to generalize better than agents trained in the varied version. One potential explanation for this is that while agents in the fixed environment are only exposed to a single instance of the tower, they are able to make more progress on this single instance, and therefore able to learn more useful behaviors which can be re-purposed on new instances of the tower. In comparison the agents trained using Rainbow under the varied condition outperforms all other algorithms and training conditions in terms of evaluation performance on the generalization conditions.

As expected, agents trained on either condition perform significantly worse in the *Strong Generalization* evaluation condition. Whereas the PPO agents fail to achieve an average floor completion of one, the Rainbow (Varied) agent is able to complete an average of over one floor in the *Strong Generalization* condition.

Discussion

In this paper we have described the Obstacle Tower, a new research challenge for AI agents, and provided a discussion of why we believe the environments serves as a compelling research challenge. Furthermore, our preliminary results suggest that current state of the art methods achieve far less than human level performance on all experimental conditions. While the Rainbow (varied) agent is able to display limited generalization capabilities, these are still considerably worse than humans.

We believe that in order for learned agents to better perform on the task, fundamental improvements to the state of the art in the field will be required. We expect that these improvements will be more generally applicable beyond the Obstacle Tower itself, with impacting broader domains such as robotic navigation and planning.

Potential Areas of Research

Here we briefly sketch some potential research areas we believe may be fruitful to explore in aiming towards better performance on the Obstacle Tower.

Hierarchical Control The overall goal of an agent in the Obstacle Tower is to arrive at as high a floor in the tower as possible. This goal can naturally be decomposed into at least two levels of task: solving any individual floor, and navigating between any two rooms. We believe this structure in the environment provides the opportunity to explore hierarchical control schemes (Kaelbling, Littman, and Moore 1996). One such scheme could involve the use of a high-level planner solving for each floor, and a low-level reactive controller navigating the obstacles within a given room. It is also possible that the exact nature of the hierarchical structure can be learned in an unsupervised and unstructured manner, as was done in the FuN architecture (Vezhnevets et al. 2017).

Intrinsic Motivation The Obstacle Tower environment contains very sparse rewards. One class of methods developed to deal with sparse rewards are intrinsic motivations. These included approaches to rewarding the agent based on state-novelty (Bellemare et al. 2016), curiosity (Pathak et al. 2017), or empowerment (Mohamed and Rezende 2015). Curiosity-driven methods in particular have been shown to perform well on sparse-reward tasks involving 3D environments, motivating our use of them in the baseline results described above. We believe that extending this line of work to tasks in which the test environment is different from that in which the original curiosity reward was received is a promising future area to explore.

Meta-Learning Despite great successes in learning from large fixed data sets such as ImageNet (Krizhevsky, Sutskever, and Hinton 2012), attention has been drawn to the need for machine learning models to be able to quickly adapt to new situations at test time. This includes situations which differ significantly from those which they were exposed to

during training. The general ability of a learning system to address this problem is referred to as meta-learning (Vilalta and Drissi 2002). Recent work has shown that it is possible to for models to quickly adapt to new scenarios within the context of Deep Reinforcement Learning using either a single gradient update (Finn, Abbeel, and Levine 2017), or after being trained under specific conditions using a recurrent model architecture (Wang et al. 2016). We believe that the ability for an agent to quickly adapt to the conditions of an unseen configuration of the Obstacle Tower will be needed in order for an agent to perform well on the environment.

World-Model Learning In contrast to model-free Reinforcement Learning approaches such as DQN (Mnih et al. 2015) and PPO (Schulman et al. 2017), model-based Reinforcement Learning methods such as GPS (Levine, Wagener, and Abbeel 2015) that has focused on learning the dynamics of the environment, and then exploiting those in some way to better learn a policy. Given the need for control within an environment in which there are complex, but relatively high-level movement dynamics, such as moving platforms or blocks, we expect that approaches to model the dynamics of these objects, and then learn from them can be of particular advantage (Agrawal et al. 2016; Ha and Schmidhuber 2018).

Future Extensions

What is described here corresponds to the initial version of the Obstacle Tower (v1.0). It is being released alongside the Obstacle Tower Challenge: a contest designed to encourage research using this environment [3]. This version is limited to 25 floors, and a small number of configurable options. Over the next year we plan to continue to develop the environment going forward in order to extend its functionality and usability by different groups of researchers. For example, we plan to add additional visual themes, floor layouts, and interactive elements in future releases.

We plan to release a completely open-source version of the Obstacle Tower project code in the coming months. This version will provide the ability to easily configure the environment in a number of ways such as alternating between first and third person views, adding additional state information such as a representation of the current floor layout, the freedom to modify the reward function, along with the ability to add new module and item types into the procedural generation system. We hope that these extensions will allow the Obstacle Tower to not only be useful as a high-end benchmark of agents abilities, but also as a more general customizable environment for posing novel tasks to learning agents.

Conclusion

For the past few years the Arcade Learning Environment, and the game Montezuma's Revenge in particular has been used as a benchmark to both measure and guide progress in Deep Reinforcement Learning. We hope that the Obstacle Tower environment, with the focus on unsolved problems in vision, control, planning, and generalization, can serve the community in a similar way for the coming years. We encourage researchers interested in the environment to participate in our open challenge, and look forward to seeing the results and novel approaches to algorithm design which participants develop.

[3] http://unity3d.com/OTC

Acknowledgments

The authors acknowledge the financial support from NSF grant (Award number 1717324 - "RI: Small: General Intelligence through Algorithm Invention and Selection.").

We would additionally like to thank Leon Chen, Jeff Shih, Marwan Mattar, Vilmantas Balasevicius, Ervin Teng, and Yuan Gao for helpful feedback and support during the development and evaluation of this environment, as well as all the Unity Technology employees who took part in the human performance evaluation process.

References

[Agrawal et al. 2016] Agrawal, P.; Nair, A. V.; Abbeel, P.; Malik, J.; and Levine, S. 2016. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, 5074–5082.

[Aytar et al. 2018] Aytar, Y.; Pfaff, T.; Budden, D.; Paine, T. L.; Wang, Z.; and de Freitas, N. 2018. Playing hard exploration games by watching youtube. *arXiv preprint arXiv:1805.11592*.

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

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

[Bellemare et al. 2016] Bellemare, M.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, 1471–1479.

[Brockman et al. 2016] Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

[Burda et al. 2018a] Burda, Y.; Edwards, H.; Pathak, D.; Storkey, A.; Darrell, T.; and Efros, A. A. 2018a. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*.

[Burda et al. 2018b] Burda, Y.; Edwards, H.; Storkey, A.; and Klimov, O. 2018b. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*.

[Castro et al. 2018] Castro, P. S.; Moitra, S.; Gelada, C.; Kumar, S.; and Bellemare, M. G. 2018. Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*.

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

[Dhariwal et al. 2017] Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; Wu, Y.; and Zhokhov, P. 2017. Openai baselines. `https://github.com/openai/baselines`.

[Dormans 2010] Dormans, J. 2010. Adventures in level design: Generating missions and spaces for action adventure games. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, PCGames '10. ACM.

[Ecoffet et al. 2018] Ecoffet, A.; Huizinga, J.; Lehman, J.; Stanley, K.; and Clune, J. 2018. Montezuma's revenge solved by go-explore, a new algorithm for hard-exploration problems (sets records on pitfall too).

[Finn, Abbeel, and Levine 2017] Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*.

[Ha and Schmidhuber 2018] Ha, D., and Schmidhuber, J. 2018. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, 2455–2467.

[Henderson et al. 2017] Henderson, P.; Islam, R.; Bachman, P.; Pineau, J.; Precup, D.; and Meger, D. 2017. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*.

[Hessel et al. 2018] Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; and Silver, D. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[Hingston 2009] Hingston, P. 2009. A turing test for computer game bots. *IEEE Transactions on Computational Intelligence and AI in Games* 1(3):169–186.

[Hsu et al. 1990] Hsu, F.-h.; Anantharaman, T.; Campbell, M.; and Nowatzyk, A. 1990. A grandmaster chess machine. *Scientific American* 263(4):44–51.

[Huang et al. 2017] Huang, S.; Papernot, N.; Goodfellow, I.; Duan, Y.; and Abbeel, P. 2017. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*.

[Johnson et al. 2016] Johnson, M.; Hofmann, K.; Hutton, T.; and Bignell, D. 2016. The malmo platform for artificial intelligence experimentation. In *IJCAI*, 4246–4247.

[Juliani et al. 2018] Juliani, A.; Berges, V.-P.; Vckay, E.; Gao, Y.; Henry, H.; Mattar, M.; and Lange, D. 2018. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.

[Kaelbling, Littman, and Moore 1996] Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4:237–285.

[Karakovskiy and Togelius 2012] Karakovskiy, S., and Togelius, J. 2012. The mario ai benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games* 4(1):55–67.

[Kempka et al. 2016] Kempka, M.; Wydmuch, M.; Runc, G.; Toczek, J.; and Jaśkowski, W. 2016. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, 1–8. IEEE.

[Krizhevsky, Sutskever, and Hinton 2012] Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

[Levine, Wagener, and Abbeel 2015] Levine, S.; Wagener, N.; and Abbeel, P. 2015. Learning contact-rich manipulation skills with guided policy search. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 156–163. IEEE.

[Machado et al. 2017] Machado, M. C.; Bellemare, M. G.; Talvitie, E.; Veness, J.; Hausknecht, M.; and Bowling, M. 2017. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *arXiv preprint arXiv:1709.06009*.

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

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

[Mohamed and Rezende 2015] Mohamed, S., and Rezende, D. J. 2015. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, 2125–2133.

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

[Pathak et al. 2017] Pathak, D.; Agrawal, P.; Efros, A. A.; and Darrell, T. 2017. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017.

[Perez-Liebana et al. 2016] Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Lucas, S. M.; and Schaul, T. 2016. General video game ai: Competition, challenges and opportunities. In *Thirtieth AAAI Conference on Artificial Intelligence*, 4335–4337.

[Resnick et al. 2018] Resnick, C.; Eldridge, W.; Ha, D.; Britz, D.; Foerster, J.; Togelius, J.; Cho, K.; and Bruna, J. 2018. Pommerman: A multi-agent playground. *arXiv preprint arXiv:1809.07124*.

[Rohlfshagen et al. 2018] Rohlfshagen, P.; Liu, J.; Perez-Liebana, D.; and Lucas, S. M. 2018. Pac-man conquers academia: Two decades of research using a classic arcade game. *IEEE Transactions on Games* 10(3):233–256.

[Samuel 1959] Samuel, A. L. 1959. Some studies in ma-

chine learning using the game of checkers. *IBM Journal of research and development* 3(3):210–229.

[Schaeffer et al. 1992] Schaeffer, J.; Culberson, J.; Treloar, N.; Knight, B.; Lu, P.; and Szafron, D. 1992. A world championship caliber checkers program. *Artificial Intelligence* 53(2-3):273–289.

[Schaul, Togelius, and Schmidhuber 2011] Schaul, T.; Togelius, J.; and Schmidhuber, J. 2011. Measuring intelligence through games. *arXiv preprint arXiv:1109.1314*.

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

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

[Stiny and Gips 1971] Stiny, G., and Gips, J. 1971. Shape grammars and the generative specification of painting and sculpture. In *IFIP Congress (2)*, volume 2.

[Turing 2004] Turing, A. 2004. Intelligent machinery (1948). *B. Jack Copeland* 395.

[Vezhnevets et al. 2017] Vezhnevets, A. S.; Osindero, S.; Schaul, T.; Heess, N.; Jaderberg, M.; Silver, D.; and Kavukcuoglu, K. 2017. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*.

[Vilalta and Drissi 2002] Vilalta, R., and Drissi, Y. 2002. A perspective view and survey of meta-learning. *Artificial Intelligence Review* 18(2):77–95.

[Wang et al. 2016] Wang, J. X.; Kurth-Nelson, Z.; Tirumala, D.; Soyer, H.; Leibo, J. Z.; Munos, R.; Blundell, C.; Kumaran, D.; and Botvinick, M. 2016. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*.

[Yu 2016] Yu, D. 2016. *Spelunky*. Boss Fight Books.

[Zhang et al. 2018] Zhang, C.; Vinyals, O.; Munos, R.; and Bengio, S. 2018. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*.