MDP Playground: A Design and Debug Testbed for Reinforcement Learning

Raghu Rajan¹, Jessica Lizeth Borja Diaz¹, Suresh Guttikonda¹, Fabio Ferreira¹, André Biedenkapp¹, Jan Ole von Hartz¹ & Frank Hutter¹,²
¹ University of Freiburg ² Bosch Center for Artificial Intelligence
rajanr@cs.uni-freiburg.de

Abstract

We present MDP Playground, an efficient testbed for Reinforcement Learning (RL) agents with orthogonal dimensions that can be controlled independently to challenge agents in different ways and obtain varying degrees of hardness in generated environments. We consider and allow control over a wide variety of dimensions, including delayed rewards, rewardable sequences, density of rewards, stochasticity, image representations, irrelevant features, time unit, action range and more. We define a parameterised collection of fast-to-run toy environments in OpenAI Gym by varying these dimensions and propose to use these for the initial design and development of agents. We also provide wrappers that inject these dimensions into complex environments from Atari and Mujoco to allow for evaluating agent robustness. We further provide various example use-cases and instructions on how to use MDP Playground to design and debug agents. We believe that MDP Playground is a valuable testbed for researchers designing new, adaptive and intelligent RL agents and those wanting to unit test their agents.

1 Introduction

RL has succeeded at many disparate tasks, such as helicopter aerobatics, game-playing and continuous control [2, 38, 48, 10, 14, 17]. However, a lot of the insights obtained are on very complex and in many instances blackbox environments.

There are many different types of standard environments, as many as there are different kinds of tasks in RL [e.g. 56, 6, 11]. They specialise in specific kinds of tasks. The underlying assumptions in many of these environments are that of a Markov Decision Process (MDP) [see, e.g., 44, 51] or a Partially Observable MDP (POMDP) [see, e.g., 22, 25]. However, there is a lack of simple and general MDPs which capture common difficulties seen in RL and let researchers experiment with them in a fine-grained manner. Many researchers design their own toy problems which capture the key aspect of their problem and then try to gain whitebox insights because the standard complex environments, such as Atari and Mujoco, are too expensive or too opaque for the initial design and development of their agent. To standardise this initial design and debug phase of the development pipeline, we propose a platform which distils difficulties for MDPs that can be generalised across RL problems and allows to independently inject these difficulties.

Disadvantages of complex environments when considered from a point of view of a design and debug testbed include: 1) They are very expensive to evaluate. For example, a DQN [38] run on Atari [6] took us 4 CPU days and 64GB of memory to run. 2) The environment structure itself is so complex that it leads to “lucky” agents performing better (e.g., in [18]). Furthermore, different implementations even using the same libraries can lead to very different results [18]. 3) Many difficulties are concurrently present in the environments and do not allow us to independently test

their impact on agents’ performance. During the design phase, we need environments to encapsulate, preferably orthogonally, the different difficulties present. For instance, MNIST [32] captured some key difficulties required for computer vision (CV) which made it a good testbed for designing and debugging CV algorithms, even though it cannot be used to directly learn models for much more specific CV applications such as classification of plants or medical image analysis.

The main contributions of this paper are:

- We identify and discuss dimensions of MDPs that can have a significant effect on agent performance, both for discrete and continuous environments;
- We discuss how to use MDP Playground to design and debug agents with various experiments; toy experiments can be run in as few as 30 seconds on a single core of a laptop;
- We discuss insights that can be gained with the various considered dimensions; transferring insights from toy to complex environments for some under-studied dimensions led to significant improvements in performances on complex environments.

2 Dimensions of MDPs

We try to exhaustively identify orthogonal dimensions of hardness in RL by going over the many components of a (PO)MDP. By orthogonal, we mean that these dimensions are present independent of each other in environments. This was done exhaustively to allow as many dimensions as possible for researchers to systematically study them and gain new insights.

We define an MDP as a 7-tuple $(S, A, P, R, \rho_0, \gamma, T)$, where $S$ is the set of states, $A$ is the set of actions, $P : S \times A \rightarrow S$ describes the transition dynamics, $R : S \times A \times S \rightarrow \mathbb{R}$ describes the reward dynamics, $\rho_0 : S \rightarrow \mathbb{R}^+$ is the initial state distribution, $\gamma$ is the discount factor and $T$ is the set of terminal states. We define a POMDP with two additional components - $O$ represents the set of observations and $\Omega : S \times A \times O \rightarrow \mathbb{R}^+$ describes the probability density function of an observation given a state and action. To clarify terminology, following [50] we will use information state to mean the state representation used by the agent and belief state as the posterior belief of the unobserved state given the full observation history. If the belief state were to be used as the information state by an agent, this would be sufficient to compute an optimal policy. However, since the full observation history is not tractable to store for many environments, agents in practice use the last few observations as their information state which renders it only partially observable. This is important because many of the motivated dimensions are actually due to the information state being non-Markov.

We also present here the $Q^*$-value [38] and use it as an example to argue how violations of assumptions may lead to degradation in performance. For a state $s$ and action $a$, a policy $\pi$ and $r_t$, the reward a timestep $t$, $Q^*$ is defined as: $Q^*(s, a) = \max_\pi \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_1 = a, \pi \right]$.  

2.1 Motivation for the Dimensions

An implicit assumption for many agents is that rewards are immediate depending on only the current information state and action. However, this is not true even for many simple environments. In many situations, agents receive delayed rewards [see e.g. [4]]. For example, shooting at an enemy ship in Space Invaders leads to rewards much later than the action of shooting. Any action taken after that is inconsequential to obtaining the reward for destroying that enemy ship. Regarding the $Q^*$ value, this means that if an incorrect information state is used, then updates performed for approximating $Q^*$ will tend to assign partial credit also to incomplete sequences. The agent may not realise that a whole sequence of actions is needed to be taken and not just some of them. While agents can converge asymptotically in the face of both delays and sequences, using the correct information state would lead to much better sample efficiency and more stable learning.
Environments can also be characterised by their reward density. In sparse reward settings [15], the supervisory reward signal is 0 throughout the trajectory and then a single non-zero reward is received at its end. This also holds true for the example of the tennis serve above.

Another characteristic of environments that can significantly impact performance of agents is stochasticity. The environment, i.e., dynamics $P$ and $R$, may be stochastic or may seem stochastic to the agent due to partial observability or sensor noise. A robot equipped with a rangefinder, for example, has to deal with various sources of noise in its sensors [54].

Environments also tend to have a lot of irrelevant features [45] that one need not focus on. This holds for both table-based learners and approximators like Neural Networks (NNs). NNs additionally can even fit random noise [61] and having irrelevant features is likely to degrade performance. For example, in certain racing car games, though the whole screen is visible, concentrating on only the road would be more efficient without loss in performance.

Another aspect is that of representations. The same underlying state may have many different external representations/observations, e.g., feature space vs pixel space. Mujoco tasks may be learnt in feature space vs directly from pixels, and Atari games can use the underlying RAM state or images. For images, various image transformations [shift, scale, rotate, flip and others; 19] may manifest as observations of the same underlying state and can pose a challenge to learning.

The diameter of an MDP, i.e., the maximum distance between 2 states, is another significant dimension affecting performance and reachability of states [23, 41]. If rewarding states are very far apart, then an agent would get less reward on average.

Further, several additional dimensions exist for continuous control problems. For instance, for the task of reaching a target, we have target radius [see, e.g., 28], a measure of the distance from the target within which we consider the target to have been successfully reached; action range [26], a weight penalising actions; and time unit, the discretisation of time.

We now summarise the dimensions identified above (with the (PO)MDP component they impact in brackets):

- Reward Delay ($R$)  
- Diameter ($P$)
- Sequence Length ($R$)  
- Irrelevant Features ($O$)
- Reward Density ($R$)  
- Representations ($O$)
- Stochasticity ($P$, $R$)  
- Action Range ($A$)
- Time Unit ($P$)  
- Target Radius ($T$)

Only selected dimensions are included here, to aid in understanding and to show use-cases for MDP playground. Trying to exhaustively identify dimensions has led to a very flexible platform and Appendix B lists all the dimensions of MDP Playground. We would like to point out that it largely depends on the domain which dimensions are important. For instance, in a video game domain, a practitioner may not want to inject any kind of noise into the environment, if their only aim is to obtain high scores, whereas in a domain like robotics adding such noise to a deterministic simulator could be crucial in order to obtain generalisable policies [65].

We now mathematically highlight some of our dimensions of hardness to aid understanding. The information state of an agent to compute an optimal policy would need to stack the previous $n + d$ observation and action pairs from the environment where $n$ denotes a sequence length and $d$ denotes a delay, i.e., a sequence of actions needs to be followed to obtain a reward which may be delayed by a certain number of steps. Reward density controls the fraction of elements in $S^m$ that are rewardable.

Additionally, the continuous control dimensions can mathematically be described as follows. The target radius sets $T = \{s | \|s - s_i\| < \text{target radius}\}$, where $s_i$ is the target point. The action range sets $A \subset \mathbb{R}^n$ where $a$ is the action space dimensionality. The time unit, $t$, sets $P'(s, a) = s + \int_0^t P_{\text{cont}}(s, a) \, dt$ where $P_{\text{cont}}$ is the underlying continuous dynamics function. The transition dynamics order, $n$, sets $P$ to be in $C^n$, the set of functions differentiable $n$ times.

3 MDP Playground

**Toy Environments** The toy environments are cheap and encapsulate all the identified dimensions. The components of the MDP can be automatically generated according to the dimensions or can be user-defined. Any dimension not specified is set to a vanilla default value. Further, the underlying MDP state is exposed in an augmented state variable, which allows users to design agents that may
try to identify the true underlying MDP state given the observations. We now briefly describe the auto-generated discrete and continuous environments, since we use these for the experiments section and expect that these will cover the majority of the use-cases. This is followed by implementation details of selected dimensions; details for all dimensions can be found in Algorithm 1 in Appendix C.

**Discrete Environments** In the discrete case, $S$ and $A$ contain categorical elements, and random instantiations of $P$ and $R$ are generated after the remaining dimensions have been set. The generated $P$ and $R$ are deterministic and held fixed for the environment. We keep $\rho_p$ to be uniform over the non-terminal states, and $T$ is fixed to be a subset of $S$ based on a chosen *terminal state density*.

**Continuous Environments** In the continuous case, environments correspond to the simplest real world task we could find: moving a rigid body to a target point, similar to [16] and [28]. $P$ is formulated such that each action dimension affects the corresponding space dimension - $s$ is set to be equal to the action applied for *time unit* seconds on a rigid body. This is integrated over time to yield the next state. $R$ is designed such that the reward for the current time step is the distance travelled towards the target since the last step.

**Reward Delay** The reward is delayed by a non-negative integer number of timesteps, $d$.

**Rewardable Sequence Length** For discrete environments, only specific sequences of states of positive integer length $n$ are rewardable. Sequences consist of non-repeating states allowing for $\binom{|S|-|T|}{|S|-|T|-n}$ sequences. For the continuous environment of moving to a target, $n$ is variable.

**Reward Density** For discrete environments, the reward density, $rd$, is defined as the fraction of possible sequences of length $n$ that are actually rewarded by the environment, given that $n$ is constant.

If $num_r$ sequences are rewarded, we define the reward density to be $rd = \frac{num_r}{\binom{|S|-|T|}{|S|-|T|-n}}$ and the sparsity as $1 - rd$. For continuous environments, density is controlled by having a sparse or dense environment using a `make_denser` configuration option.

**Stochasticity** For discrete environments, *transition noise* $t_n \in [0,1]$; with probability $t_n$, an environment transitions uniformly at random to a state that is not the `true` next state given by $P$.

For discrete environments, *reward noise* $r_n \in \mathbb{R}$; a normal random variable distributed according to $\mathcal{N}(0, \sigma^2_{r_n})$ is added to the `true` reward. For continuous environments, both $p_n$ and $r_n$ are normally distributed and directly added to the states and rewards.

**Irrelevant Features** For discrete environments, a new discrete dimension with its own transition function $P\textsubscript{irr}$ which is independent of $P$, is introduced. However, only the discrete dimension corresponding to $P$ is *relevant* to calculate the reward function. Similarly, in continuous environments, dimensions of $S$ and $A$ are labelled as irrelevant and not considered in the reward calculation.

**Representations** For discrete environments, when this aspect is enabled, each categorical state is associated with an image of a regular polygon which becomes the externally visible observation of the agent. This image can further be transformed by *shifting, scaling, rotating or flipping*, which are applied at random to the polygon whenever an observation is generated. For continuous environments, image observations can be rendered for 2D environments. Examples of some generated states can be seen in Figures 7, 8 in Appendix D.

**Diameter** For discrete environments, for *diameter* $= d$, the set of states is set to be a $d$-partite graph, where, if we order the $d$ sets as 1, 2, ..., $d$, states from set $n$ will have actions leading to states in set $n + 1$, with the final set $d$ having actions leading to states in set 1. The number of actions for each state will, thus, be `number of states`/$(d)$. This gives the discrete environments a grid-world like structure. For continuous environments, setting the dimension *state space max* sets the bounds of the environment to ±*state space max* and the *diameter* $= 2\sqrt{2}$ *state space max*.

**Design decisions** While many dimensions can seem challenging at first, it is also the nature of RL that different dimensions tend to be important in different specific applications. The video game domain was provided as an example of this in Section 2.1. Another example is of *reward scale*. The agents we tested here re-scale or clip rewards already and the effects of this dimension are not as important as they would be otherwise. To maintain the flexibility of having as many dimensions as possible and yet keep the platform easy to use, *default* values are set for dimensions that are not configured. This effectively turns off those dimensions. Thus, as in the code example, users only need to provide dimensions they are interested in.

Further design decisions are discussed in detail in Appendix H.
Figure 1: AUC of episodic reward at the end of training for the different agents when varying representation. 's' denotes shift (quantisation of 1), 'S' scale, 'f' flip and 'r' rotate in the labels in the first three subfigures and image_sh_quant represents quantisation of the shifts in the DQN experiment for this. Error bars represent 1 standard deviation. Note the different reward scales.

Figure 2: a and b: DDPG with time unit on toy and complex (HalfCheetah) environment at the end of training (time unit is relative to the defaults). c: DDPG with irrelevant dimensions injected on the toy environment. d: DQN on qbert. Error bars represent 1 standard deviation. Note the different y-axis scales.

**Code samples** An environment instance is created as easily as passing a Python dict:

```python
from mdp_playground.envs import RLToyEnv
cfg = {
    'state_space_type': 'discrete',
    'action_space_size': 8,
    'delay': 1,
    'sequence_length': 3,
    'reward_density': 0.25,
}
env = RLToyEnv(**cfg)
```

**Very low-cost execution** Experiments with MDP Playground are cheap, allowing academics without special hardware to perform insightful experiments. Wall-clock times depend a lot on the agent, network size (in case of NNs) and the dimensions used. Nevertheless, to give the reader an idea of the runtimes involved, DQN experiments (with a network with 2 hidden layers of 256 units each) took on average 35s for a complete run of DQN for 20 000 environment steps. In this setting, we restricted Ray RLLib [33] and the underlying Tensorflow [1] to run on one core of a laptop (core-i7-8850H CPU – the full CPU specifications for a single core can be found in Appendix [1]). This equates to roughly 30 minutes for the entire delay experiment shown in Figure 9a which was plotted using 50 runs (10 seeds × 5 settings for delay; these 50 runs could also be run in an embarrassingly parallel manner on a cluster). Even when using the more expensive continuous or representation learning environments, runs were only about 3-5 times slower.

**Complex Environment Wrappers** We further provide wrappers for Atari and Mujoco which can be used to inject some of the dimensions also into complex environments.

## 4 Using MDP Playground

We discuss in detail various experiments along with how they may be used to design new agents and to debug existing agents. For the experiments, we set |S| and |A| to 8 and the terminal state density to 0.25. The reward scale is set to 1.0 whenever a reward is given by the environment. We evaluated Rllib implementations [33] of DQN [38], Rainbow DQN [20], A3C [37] on discrete environments and DDPG [34], TD3 [14] and SAC [17] on continuous environments over grids of values for the dimensions. Hyperparameters and the tuning procedure used are available in Appendix Q. We used fully connected networks except for pixel-based representations where we used Convolutional Neural Networks (CNNs) [31].
4.1 Designing New Agents

We hope our toy environments will help identify inductive biases needed for designing new RL agents without getting confounded by other sources of "noise" in the evaluation. What is important for doing this is to be able to identify if the trends seen on the toy environments would also occur for more complex environments. We now provide empirical support for this with several experiments.

**Varying representations** We turned on image representations for discrete environments and applied various transforms (shift, scale, rotate and flip) one at a time and also all at once. We observed that the more transforms are applied to the images, the harder it is for agents to learn, as can be seen in Figures 1a-4. This was to be expected since there are many more combinations to generalise over for the agent.

It is important to note, from the point of view of a design platform, that our platform allows us to identify the inductive bias of CNNs being good for image observations without having to conduct such experiments on complex and expensive environments. This is because the toy environments capture many key features of image representations and thus the image classification capabilities of CNNs can help identify the underlying MDP state. In a similar manner, we have captured key features of other dimensions. If one were to design a new inductive bias which helps the agent identify the underlying MDP state in the presence of the other dimensions, this could easily be tested on our platform.

**Varying time unit** We observed that the time unit has an optimal value which has significant impact on performance in the toy continuous environment (Figure 2a). We decided to tune the time unit also for complex environments (Figures 2b, 7a, 14b and 14f). The insight from the toy environment clearly transferred to the complex case and there were gains of even 100% in some cases over the default value of the time units used. A further insight to be had is that for simpler environments like the toy, Pusher and Reacher, the effect of the selection of the time unit was not as pronounced as for a more complex environment like HalfCheetah. This makes intuitive sense as one can expect a narrower range of values to work for more complex environments. This shows that it is even more important to tune such dimensions for more complex environments.

The basic agent design we showed above does this once and sets its optimal time unit statically. An ideal adaptive agent design would even set the time unit in an online manner. Since the trends from the toy environment transfer to the complex environment, this would be quickly designed and tested on the toy environment. Similar comments can be made about the action range (Figures 14b and 14f in Appendix K). To the best of our knowledge, the impact of time unit and action range is under-researched while developing agents because the standard environments have been pre-configured by experts. However, it’s clear from Figure 2b that pre-configured values were not optimal and even basic tuning improves performance significantly in even known environments. In a completely unknown environment, if we want agents to perform optimally, these dimensions would need to be taken into account even more when designing agents.

**Varying irrelevant features** We observed that introducing irrelevant dimensions to the control problem, while keeping the number of relevant dimensions fixed to 2, decreased an agent’s performance (see Figures 2c & 14f). This gives us the insight that having irrelevant features interferes with the learning process. An inductive bias that learns to focus only on the relevant dimensions could easily and quickly be tested on the toy environments.

**Varying Multiple Dimensions** In MDP Playground, it is possible to vary multiple dimensions at the same time in the same base environment. For instance, Figure 3d shows the interaction effect (an inversely proportional relationship) between the action range and the time unit in the continuous toy environment. This insight allows us to design an adaptive agent which sets its action range depending on the time unit and vice versa. Since many real-world systems can be described in terms of a simple rigid body moving towards a target point, the toy continuous environment is already a useful testbed for this.

More such experiments can be found in Appendix M, including varying both P and R noises together in discrete environments and more. Further design ideas for new agents can be found in Appendix E.
4.2 Insights into Existing Agents

Apart from the insights gained for designing agents above, we discuss more insights for existing agents explicitly here.

The experiment for varying representations on toy environments discussed above (Figures 1a-c) further showed that the degradation in performance is much stronger for DQN compared to Rainbow and A3C which are known to perform better than DQN in complex environments.

This led us to another interesting insight regarding the inductive bias of CNNs. It was unexpected for us that the most problematic transform for the agents to deal with was shift. Despite the spatial invariance learned in CNNs [30], our results imply that that seems to be the hardest one to adapt to. As these trends were strongest in DQN, we evaluated further ranges for the individual transforms for DQN. Here, shifts had the most possible different combinations that could be applied to the images. Therefore, we quantised the shifts to have fewer possible values. Figure 1d shows that DQN’s performance improved with increasing quantisation (i.e., fewer possible values) of shift. We noticed similar trends for the other transforms as well, although not as strong as they do not have as many different values as shift (see Figures 2b–c in Appendix K). We emphasize that in a more complex setting, we would have easily attributed some of these results to luck but in the setting where we had individual control over the dimensions, our platform allowed us to dig deeper in a controlled manner.

Another insight we gain is from the time unit experiment (see Figures 2a and 2b), which indicates time unit should not be infinitesimally small to achieve too fine-grained control since there is an optimal time unit for which we should repeat the same action.

![Radar plots for dimensions](image)

Figure 3: Analysing and Debugging

In Figure 2d, where we varied delay on qbert, we show how a dimension induces hardness in an environment. This result is representative of the experiments on toy and complex environments which are included in Appendix K and O with the difference that results are noisier in complex environments since the dimensions are already present there in varying degrees. We, thus, studied what kinds of failure modes can occur when an agent is faced with such dimensions and even obtained noisy learning curves typically associated with RL on the toy environments as can be seen in Appendix N.

At the same time, the experiment in Figure 2d also shows how the complex environment wrappers allow researchers, who are curious, to study the robustness of their agents to these dimensions on complex environments, without having to fiddle with lower-level code. This is a typical use-case further down the agent development pipeline, i.e., close to deployment. Additional experiments aiming to gain insights are discussed in Appendix F.

**Design and Analyse Experiments** We allow the user the power to inject dimensions into toy or complex environments in a fine-grained manner. This can be used to define custom experiments with the dimensions. The results can be analysed in an accompanying Jupyter notebook using the 1D plots. There are also radar plots inspired by bsuite [42], but with more flexibility in choosing the dimensions, and these can even be applied to complex environment experiments. Since, different users might be interested in different dimensions, these are loaded dynamically from the data. For instance, radar plots for the dimensions we varied in our toy experiments can be seen as in Figures 3a and 3b.

4.3 Debugging Agents

Analysing how an agent performs under the effect of various dimensions can reveal unexpected aspects of an agent. For instance, when using bsuite agents, we noticed that when we varied our
environment’s reward density, the performance of the bsuite Sonnet DQN agent would go up in proportion to the density (see Figure 3c). This did not occur for other bsuite agents. This seemed to suggest something different for the DQN agent and when we looked at DQN’s hyperparameters we realised that it had a fixed $\epsilon$ schedule while the other agents had decaying schedules. Such insights can easily go unnoticed if the environments used are too complex. The high bias nature of our toy environments helps debug such cases.

In another example, in one of the Ray versions we used, we observed that DQN was performing well on the varying representations environment while Rainbow was performing poorly. We were quickly able to ablate additional Rainbow hyperparameters on the toy environments and found that their noisy nets [13] implementation was broken (see Figure 4 in Appendix). We then tested and observed the same on more complex environments. This shows how easily and quickly agents can be debugged to see if something major is broken. This, in combination with their low computational cost, also makes a case to use the toy environments in Continuous Integration (CI) tests on repositories.

Further, we believe the same structured nature of MDP Playground also makes it a valuable tool for theoretical research. We evaluated tabular baselines Q-learning [51], Double Q-learning [58] and SARSA [51] on the discrete non-image based environments with similar qualitative results to those for deep agents. These can be found in Appendix F. This makes our platform a bridge between theory and practice where both kinds of agents can be tested.

The experiments here are only a glimpse into the power and flexibility of MDP Playground. Users can even upload custom $P$s and $R$s and custom images for representations $O$ and our platform takes care of injecting the other dimensions for them (wherever possible). This allows users to control different dimensions in the same base environment and gain further insights.

### 5 Discussion and Related Work

The Behaviour Suite for RL [bsuite; 42] is the closest related work to MDP Playground. [42] collect known (toy) environments from the literature and use these to characterise agents based on their performance on these environments. Most environments in bsuite can be seen as an intermediate step between our MDPs and more complex environments. This is because bsuite’s environments are already more specific and complex than the toy environments in MDP Playground. This makes bsuite’s dimensions not orthogonal and atomic like ours and thus not individually controllable. Fine-grained control is a feature that sets our platform apart. bsuite has a collection of presets chosen by experts which work well but would be much harder to play around with. While MDP Playground also has good presets through default values defined for experiments, it is much easier to configure. Further, it also means that bsuite experiments are much more expensive than ours. While bsuite itself is quite cheap to run, MDP Playground experiments are an order of magnitude cheaper. In contrast to bsuite, we demonstrate how the identified trends on the toy and complex environments can be used to design and debug agents. Further, bsuite currently has no toy environment for Hierarchical RL (HRL) agents while MDP Playground’s rewardable sequences fits very well with HRL. Finally, bsuite offers no continuous control environments, whereas MDP Playground provides both discrete and continuous environments. This is important because several agents like DDPG, TD3, SAC are designed for continuous control. A more detailed comparison with bsuite and other related work can be found in Appendix F.

Toybox [57] and Minatar [60] are also cheap platforms like ours with similar goals of gaining deeper insights into RL agents. However, their games target the specific Atari domain and are, like bsuite, more specific and complementary to our approach.

We found [3] the most similar work to ours in spirit. They propose that current deep RL research has been increasing the complexity of the dynamics $P$ but has not paid much attention to the state distributions and reward distribution over which RL policies work and that this has made RL agents brittle. We agree with them in this regard. However, they only target continuous environments. We capture their dimensions in a different manner and offer many more dimensions with fine-grained control. Furthermore, their code is not open-source.

Further research includes Procgen [11], Obstacle Tower [24] and Atari [6]. Procgen adds various heterogeneous environments and tries to quantify generalisation in RL. In a similar vein, Obstacle Tower provides a generalization challenge for problems in vision, control, and planning. These
benchmarks do not capture orthogonal dimensions of difficulty and as a result, they do not have the same type of fine-grained control over their environments’ difficulty and neither can each dimension be controlled independently. We view this as a crucial aspect when testing new agents. [12] provides some overlapping dimensions with our platform but it consists of only continuous environments, and doesn’t target the toy domain.

6 Limitations of the Approach and its Ethical and Societal Implications

The toy environments are meant to be design and debug testbeds and not for engineering/tuning the final agent HPs. As such, they are extremely cheap compared to complex environments and (as one would expect), they can only be used to draw high-level insights that transfer and are likely not as discriminating as complex environments for many of the finer changes between RL agents. They also cannot be used directly to determine the values of hyperparameters (HPs) to use on complex environments. Just as complex environments require bigger NNs, they would need correspondingly different HPs, such as bigger replay buffers.

Further, high-dimensional control problems where there are interaction effects between degrees of freedom are not captured in the toy rigid body control problem as this is the domain of complex benchmarks and beyond the scope of this platform. (The platform does provide complex environment wrappers, though, which inject some of the mentioned dimensions. We couldn’t find such wrappers in the literature/on the Internet.)

Finally, Multi-Agent RL, Multi Objective RL, Time Varying MDPs (and probably some more research areas) are beyond the scope of the current work.

In terms of the broader impact on society and ethical considerations, we foresee no direct impact, only indirect consequences through RL since our work promotes standardisation and reproducibility which should accelerate RL research. An additional environmental impact would be that, at least, prototyping and testing of agents could be done cheaply, reducing carbon emissions.

7 Conclusion and Future Work

We introduced a low-cost platform to design and debug RL agents and provided instructions on how to use it with supporting experiments. The platform allows us to disentangle various factors that make RL environments hard by providing fine-grained control over various dimensions. This also lends itself to easily achievable insights and helps debug agents. We further demonstrated how the performance of the studied agents is adversely affected by the dimensions. To the best of our knowledge, we are the first to perform a principled study of how significant aspects such as non-Markov information states, irrelevant features, representations and low-level dimensions, like time discretisation, affect agent performance.

We want MDP Playground to be a community-driven effort and it is open-source for the benefit of the RL community at [https://github.com/automl/mdp-playground](https://github.com/automl/mdp-playground). While we tried to exhaustively identify dimensions of hardness, it is unlikely that we have captured all orthogonal dimensions in RL. We welcome more dimensions that readers think will help us encapsulate further challenges in RL and will add them based on the community’s thoughts.

Given the current brittleness of RL agents [18], and many claims that have been challenged [5, 57], we believe RL agents need to be tested on a lower and more basic level to gain insights into their inner workings. MDP Playground is like a programming language for regularly structured MDPs which allows delving deeper into the inner workings of RL agents.
Acknowledgements

The authors gratefully acknowledge support by BMBF grant DeToL, by the Bosch Center for Artificial Intelligence, and by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme under grant no. 716721, by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG. They would like to thank their group, especially Joerg, Steven, Samuel, for helpful feedback and discussions. Raghu would like to additionally thank Michael Littman for his feedback and encouragement and the RLSS 2019, Lille organisers and participants who he had interesting discussions with.

References


Checklist

1. For all authors...

(a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes] The orthogonal dimension that influence RL agents performances are presented in Section 2 and their role in the implemented MDPs is discussed in Section 3. We showed that varying these dimensions can provide new insights or confirm existing insights (on the toy environments that also hold on more complex ones) in Section 4.2. We discussed how our proposed benchmark can aid in designing new agents by taking the proposed dimensions into account during the design (see Section 4.1). Finally, we discuss how the benchmark can help in debugging agents and could be used for continuous integration (see Section 4.3).

(b) Did you describe the limitations of your work? [Yes] See Section 6.

(c) Did you discuss any potential negative societal impacts of your work? [Yes]

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [N/A]

(b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments (e.g. for benchmarks)...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See https://github.com/automl/mdp-playground and the link is also given in Section 7.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix R.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In Section 3, we discussed the low-cost execution of experiments on MDP Playground and we provide further details along with hardware specifications in the Appendix T.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]