Journal of Visual Languages and Computing xx (xxxx) xxxx–xxxx

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

# Interactive visualization for testing Markov Decision Processes: MDPvis

Sean McGregor[a,*], Hailey Buckingham[b], Thomas G. Dietterich[a], Rachel Houtman[b], Claire Montgomery[b], Ronald Metoyer[a,c]

[a] School of Electrical Engineering and Computer Science, Oregon State University, 1148 Kelley Engineering Center, Corvallis, OR 97331-4501, USA
[b] Department of Forest Engineering, Oregon State University, USA
[c] Computer Science and Engineering, University of Notre Dame, USA

ARTICLE INFO

*Keywords:*
Visualization
Markov decision process
Testing
Parameter space analysis
Wildfire
Optimization

ABSTRACT

Markov Decision Processes (MDPs) are a formulation for optimization problems in sequential decision making. Solving MDPs often requires implementing a simulator for optimization algorithms to invoke when updating decision making rules known as policies. The combination of simulator and optimizer are subject to failures of specification, implementation, integration, and optimization that may produce invalid policies. We present these failures as queries for a visual analytic system (MDPvis). MDPvis addresses three visualization research gaps. First, the data acquisition gap is addressed through a general simulator-visualization interface. Second, the data analysis gap is addressed through a generalized MDP information visualization. Finally, the cognition gap is addressed by exposing model components to the user. MDPvis generalizes a visualization for wildfire management. We use that problem to illustrate MDPvis and show the visualization's generality by connecting it to two reinforcement learning frameworks that implement many different MDPs of interest in the research community.

## 1. Introduction

Many challenging optimization problems in sustainability [13,18], game AI [39], and autonomous control [27] require considering the long-term impacts of actions whose outcomes are stochastic. For example, forest managers must decide whether to suppress a wildfire whose results may prevent wildfires from spreading over subsequent decades [18]. A *policy* that accounts for these temporal and stochastic effects is produced by an optimization system integrating several components that are subject to failures of specification, implementation, integration, and optimization. Since the system stochastically expresses and hides these failures (referred to as "bugs"), Testing and debugging require exploration. Visual analytics is well suited to this exploratory task. We introduce a generalized visualization (see Fig. 2), MDPvis, to support this task.

This paper builds on the work of McGregor et al. [23] with details on the users engaged in testing MDPs, additional details on the theoretical formulation of MDPs, a set of parameter space analysis examples within MDPvis, and details on integrating MDPvis with MDP research frameworks.

To address a broader class of optimization problems, we target the common optimization formulation of a Markov Decision Process (MDP). In an MDP, the state of the world evolves stochastically from one state to another depending on the action chosen at each time step. A scalar reward is received at each time step depending on the system state and the chosen action. An MDP is solved by learning a decision making rule (policy) that maximizes the long-term sum of rewards.

Some MDPs are small enough to solve with exact algorithms such as Policy Iteration and Value Iteration [6], but most MDPs of practical interest require Monte Carlo methods. In these cases, the standard approach is to implement a software simulation of the MDP and then apply a Monte Carlo optimizer, like policy gradient search [12,35] to find a near-optimal policy.

Ng [26] relays an example of a soccer playing agent whose MDP optimizer learns a policy that maximizes expected reward by exploiting a bug. The soccer agent received a reward for touching the ball under the theory that possession time is associated with scoring goals. Instead of using ball possession to advance down the field, the agent stood by the ball and began to "vibrate" to produce the maximum number of ball touches. This bug can be viewed as a problem in specification (the agent should not be rewarded for touching the ball), implementation (the agent should not be able to vibrate next to the ball), integration (the frequency of reward granted by the transition function for ball touches is too high), and optimization (a more difficult to discover policy may actually be optimal). The multitude of possible MDP bugs and causes give rise to a highly iterative development process.

* Corresponding author.
*E-mail address:* jvlc@seanbmcgregor.com (S. McGregor).

http://dx.doi.org/10.1016/j.jvlc.2016.10.007
Received 1 December 2015; Received in revised form 19 August 2016; Accepted 24 October 2016
Available online xxxx
1045-926X/ 

During our design process for MDPvis we conducted a series of semi-structured interviews [34] with MDP researchers to elicit current practices for MDP development. We found a variety of ad hoc testing systems in support of an "informed trial and error" [33] process whereby MDP practitioners iteratively explore the parameter space. MDP practitioners generally first write an interactive client to manually execute transitions, followed by a visualization of state development as a policy rule is followed. None of the researchers we interviewed use a generic tool supporting this process. We hypothesize this is because researchers have heretofore not had access to a visualization they can easily connect to their MDP simulator and MDP optimizer.

MDPvis makes three contributions. First, it introduces a domain independent protocol by which the visualization system can interact with the MDP optimizer and MDP simulator. Second, it provides a visual interface that supports the data analysis tasks that problem domain experts, software developers, and optimization researchers need to perform. Third, it provides an interface by which the users can interact with the MDP simulator and optimizer to test and compare different parameters and explore their effects on the resulting behavior of the system.

These three contributions relate to the three challenges identified by Sedlmair et al. [33]: the data acquisition gap (getting the data into the visualization tool), the data analysis gap (helping the user visualize the data), and the cognition gap (helping the user uncover important behavior embedded within high-dimensional systems).

Software testing is a subfield of software engineering that includes more precise definitions of testing, and bugs. In this paper, we take a high-level view of testing. In particular, we define testing as the "... execution of a program with the intent to produce some problems - especially a failure" [41]. These failures are generally called "bugs".

In other words, developers test software for bugs by comparing the results of execution to the expected results. It is clear from this definition that testing requires the developer 1) associate program output with program input in order to create test cases by executing the code under specific input conditions and 2) compare actual outputs to expected outputs. Our visualization tool supports these two tasks within the context of MDPs.

Our design process for MDPvis followed Munzner's nested model [25], which includes steps for characterizing the problem domain and designing visual encodings. In the following sections we formally introduce MDPs with their Data and Task Abstraction, detail the contexts where users test and debug MDPs in Section 3, review the optimization visualization literature in Section 4, give our visual encoding for MDPs in Section 5, and present the visualization's prototype for the Wildfire Suppression domain in Section 6.

## 2. Data and task abstraction

While several different MDP formulations are used in the literature, there is a de facto standard formulation from which other formulations are viewed as specializations. We formally state the MDP as the standard infinite horizon discounted Markov Decision Process (MDP) with a designated start state distribution [5,30] $\mathcal{M} = \langle S, A, P, R, \gamma, P_0 \rangle$. $S$ is a finite set of states of the world; $A$ is a finite set of possible actions that can be taken in each state; $P$: $S \times A \times S \rightarrow [0, 1]$ is the conditional probability of entering state $s'$ when action $a$ is executed in state $s$; $R(s, a)$ is the reward received after performing action $a$ in state $s$; $\gamma \in (0, 1)$ is the discount factor, and $P_0$ is the distribution over starting states. Generally the goal for optimizing an MDP is to find a policy, $\pi$, that selects actions maximizing the discounted expected value of the MDP. For convenience we also define $P_{n|\pi}$ to be the distribution of states at time $n$ when following policy $\pi$.

This formulation specifies a model that generates data in the form of "Monte Carlo rollouts" detailed in Fig. 1. These rollouts are the output of the system under test and their distribution is controlled by the parameters of the MDPs' constituent functions.

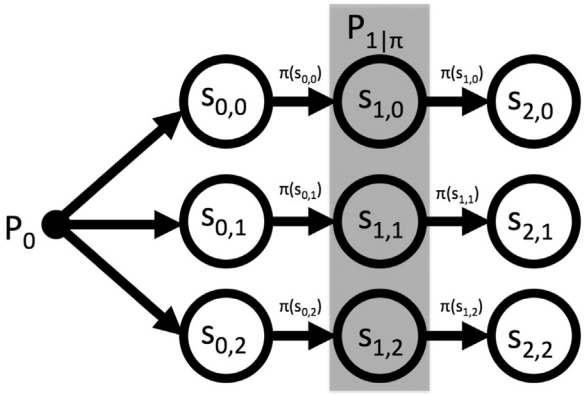


**Fig. 1.** A set of three rollouts generated starting at states drawn from the initial state distribution ($P_0$) and transitioned between states $s_{i,j}$ where $i$ is the time step and $j$ is the rollout identifier. The current policy ($\pi(s_{i,j})$) selects actions until a rollout depth of 3. The initial states and all subsequent states are defined on a set of quantitative and categorical variables. Each state transitions to resulting states by evaluating the transition function until reaching the time horizon or terminating state. The transition function draws the resulting states from a distribution that is a function of the current state and the action selected by the policy function. We highlight a set of states drawn from the distribution of states at a particular time horizon under policy $\pi$. We label this set $P_{1,\pi}$ for time step 1 under policy $\pi$.

Sedlmair et al. [33] label techniques for understanding the relationship between input parameters and outputs as Parameter Space Analysis (PSA), "...the systematic variation of model input parameters, generating outputs for each combination of parameters, and investigating the relation between parameter settings and corresponding outputs". This is a suitable definition for the MDP testing process. Finding MDP bugs requires exploring the rollouts to test for bugs.

Tables 1–6 give a series of testing questions derived from experience optimizing for a wildfire suppression policy domain and from interviewing MDP algorithm researchers not involved in the wildfire policy project. The tables label these questions according to the tasks of Sedlmair et al. [33] for visual parameter space analysis: *fitting, outliers, partitioning, optimization, uncertainty, and sensitivity*. In several cases we broaden the definition from Sedlmair et al. [33] to fit the scope of MDPs. We also indicate whether the testing question invloves the reward function ($R$), the transition function ($P$), the policy ($\pi$), or the optimization function ($M$).

**Fitting**, ***"Where in the input parameter space would actual measured data occur?"*:** In many applications the MDP simulator simulates real-world phenomena. While we typically do not have access to real-world validated data across the entirety of the state and action space, we do often have state transition data for a subset of the parameter space. Further, while the MDP practitioner may not have access to ground truth data for their system, they can often identify when the system is producing unrealistic outcomes. See Table 1 for fitting questions.

**Outliers**, ***"What outputs are special?"*:** Sedlmair et al. [33] breaks the outlier task into learning from the outliers of a model and checking the plausibility of the more extreme cases. The outliers are particularly important for plausibility during the testing phase of MDP development because an optimization algorithm moves through many potential policy functions while searching for an optimal policy. Each of these functions may push the MDP towards more extreme, and potentially less plausible states or state transitions. Even carefully crafted reward functions can result in the optimizer exploiting unforeseen interactions between the MDP's constituent parts. The capacity for a policy to earn rewards from unlikely events mean outliers influence policy optimization even when they are rare events. See Table 2.

**Partition**, ***"How many different types of model behaviors are possible?"*:** Here Sedlmair et al. [33] focus on relating partitions of the output space to the inputs. We expand Sedlmair et al.'s definition

**Table 1**
**Fitting MDP testing questions** .

| ID | Question | Interaction | $R$ | $P$ | $\pi$ | M |
|---|---|---|---|---|---|---|
| 1 | Is the reward function giving the expected rewards? | View the discounted or undiscounted rewards as a fan chart and filter down the rollouts or regenerate rollouts with fixed policies. | ● | ● | ● | |
| 2 | Do the rewards reflect stakeholder preferences? | Filter to individual rollouts and examine its rewards. | ● | | | |
| 3 | Do simulated transitions result in realistic states? | Examine individual rollouts. | | ● | ● | |
| 4 | Do simulated transitions result in realistic state distributions? | View the histograms of state variables at the horizon. | | | ● | |
| 5 | Does the historical policy produce the historical results? | Add a variable for each variable with historical data that gives the variable's percentile. Display this derived variable in a fan chart. | | ● | ● | |

**Table 2**
**Outlier MDP testing questions** .

| ID | Question | Interaction | $R$ | $P$ | $\pi$ | M |
|---|---|---|---|---|---|---|
| 6 | Are rollouts that complete different from rollouts that break? | Load the failing and completing rollouts as a comparison. | ● | ● | ● | ● |
| 7 | Does the policy inappropriately exploit modeling choices? | Detecting this unforeseen problem requires exploration. | ● | ● | ● | ● |
| 8 | Are the most extreme state transitions realistic? | Filter to the most extreme transitions. | ● | ● | ● | ● |

**Table 3**
**Partition MDP testing questions** .

| ID | Question | Interaction | $R$ | $P$ | $\pi$ | M |
|---|---|---|---|---|---|---|
| 9 | What is the state of the world when the transition function breaks? | Select only the rollouts that don't complete and explore them. | ● | ● | ● | ● |
| 10 | Does one policy have a higher risk of catastrophic outcome despite having a better expected value? | Compare the rollouts from both. | | ● | ● | |
| 11 | Are action selections meaningful? | Filter the histograms to a single state and see what action is selected. | | ● | ● | |
| 12 | Does an optimized policy realistically outperform a hand-coded policy? | Compare the rollouts from both. | | ● | | ● |
| 13 | Are state transitions realistic? | View state detail. | | ● | ● | |
| 14 | Do policies differ in their resultant state distributions as expected? | Generate two sets of rollouts under different policies and compare. | | | ● | |

for testing MDPs to also partition the parameters producing similar or divergent distributions of outputs. MDP testers have expectations for parameter sets, particularly when changing the parameters to atypical situations that should force extreme outcomes. Thus a tester will generate rollouts for one partition of the parameter space, and compare it to a different partition and check for the expected difference or similarity in outcome. In interactive visualization for MDPs, many

**Table 4**
**Optimization MDP testing questions** .

| ID | Question | Interaction | $R$ | $P$ | $\pi$ | M |
|---|---|---|---|---|---|---|
| 15 | Can the user do better than the optimized policy by changing the parameters? | Change the parameters of the policy function and generate new Monte Carlo rollouts. | | ● | ● | ● |
| 16 | Is the policy converged to a local optimum? | Ask it to optimize from the current position. | | ● | ● | ● |
| 17 | Is the policy converged to an acceptable local optimum? | Change the starting policy to a completely different policy and re-optimize. | | ● | ● | ● |
| 18 | Is the optimization algorithm making efficient use of computation? | Add variables to the output describing the learning process. | | | | ● |

partitioning tasks involve filtering rollouts (defined here as interactively subsetting the rollouts) to look at more specific cases or generating rollouts under new parameters. These steps are often part of other PSA tasks, as can be seen in the overlap of Table 3 with other PSA tasks.

**Optimization**, **"*Find the best parameter combination given some objectives.*":** We expand on this definition to also include testing the optimization algorithm itself. When testing new optimization algorithms on complex MDPs, it is often not obvious whether the algorithm is getting stuck in a local optimum, exploited a bug in the simulator, or could not find a way to improve outcomes. It is important to have tools that enable the optimization researcher and domain expert to collaborate around the optimization towards either improving the optimizer or the specification of the MDP. See Table 4.

**Uncertainty**, **"*How reliable is the output?*":** Optimization algorithms produce policies that select actions regardless of how confident it is in the selected action. States where the optimization algorithm is most uncertain about which action to select require testing to determine whether the uncertainty results from a near equivalence in outcomes among actions or a bug. Another view of uncertainty is in the aleatoric/statistical uncertainty of the transition functions. Testers need to know the full distribution of outcomes that are possible at every time step. See Table 5 .

**Sensitivity**, **"*What ranges/variations of outputs to expect with changes of input?*":** Sedlmair et al. [33] definition of sensitivity shows the fluidity of PSA tasks since it touches all the other PSA tasks in one respect or another. A particularly important aspect for MDPs is sensitivity of the policy to changes in the reward function to find why a learned policy is deemed near-optimal. If a learned policy is not stable within its neighborhood of similar reward functions, it is likely that the policy is not one that should be relied on in the real-world. This sensitivity question uses the partition of state distributions produced by policies optimized under different parameters to assess modeling uncertainty. See Table 6 for similarly cross-cutting questions.

Section 5.3 has additional MDP testing discussion in the context of visuals from MDPvis. Next we describe the roles that may be filled by users of MDP visualization.

**Table 5**
**Uncertainty MDP testing questions** .

| ID | Question | Interaction | $R$ | $P$ | $\pi$ | M |
|---|---|---|---|---|---|---|
| 19 | What is the distribution of states at a particular horizon? | View the fan charts. | | ● | ● | |
| 20 | How certain is the policy function about a specific choice? | View the shifting distribution of action selection while filtering the state variables. | | ● | | |

**Table 6**
**Sensitivity MDP testing questions** .

| ID | Question | Interaction | $R$ | $P$ | $\pi$ | M |
|---|---|---|---|---|---|---|
| 21 | Do small changes in the parameters produce vastly different outcomes | Change parameters then compare the two rollout sets. | ● | ● | ● | ● |
| 22 | Do small changes in the parameters produce different policies? | Change the parameters and reoptimize. | ● | ● | ● | ● |
| 23 | Do different policies earn reward through maximizing different components of the reward function? | Compare rollout sets. | ● | ● | ● | |
| 24 | Does the policy respond properly to changes in the reward function? | Change the reward parameters and re-optimize. | ● | | | ● |
| 25 | What are the differences in outcomes produced by different policies? | Load the two sets of rollouts as a comparison. | | ● | ● | |
| 26 | What are the most important drivers of policy? | Filter variables in the histogram and watch how the proportion of selected actions update. | | ● | | |

## 3. User roles

We identify four MDP development roles and give examples of these roles in the context of wildfire suppression and autonomous helicopters.

*Policy stakeholders*: MDP-derived policies often affect people that are not party to the MDP optimization process. In the wildfire domain, this stakeholder could be a home owner in the region modeled by the MDP. For an autonomous helicopter, this stakeholder could be a person purchasing a finished helicopter product. Our visualization in Section 6 does not target this population because it limits the design options for the following three groups.

*Non-programmer domain experts*: The MDP formulation can be applied to many problems for which a non-programmer is the authority on the simulated phenomena. A domain expert in the wildfire domain could be a US Forest Service land manager tasked with writing timber harvest and fire suppression policies. In the autonomous helicopter domain, a domain expert could be a professional RC stunt pilot. Visualizations help these domain experts validate the MDP by viewing the output of the system without needing to directly interact with code. Here the domain expert provides input to the software team regarding where the current system is producing unrealistic results.

*Software developers*: Software developers may develop complex MDPs from a specification written by domain experts. In the wildfire setting, developers collaborate with several domain experts from silviculture, fire modeling, and economics to produce a model integrating all these disciplines. In the autonomous helicopter domain the developers may collaborate with domain experts from mechanical engineering, computational physics, aviation, and the RC community. Here software developers may use visualization to test against expectations, or use visualization as a collaboration tool to share simulations and policies with domain experts.

*Optimization researchers*: After constructing an MDP, the optimization researcher finds policies by applying existing optimization libraries or developing new optimization algorithms. In either case, the complexities of representing a problem domain in a form that can be efficiently optimized requires optimization experience. The optimization researchers may use visualization to test the correctness or performance of their algorithms. In both the wildfire and autonomous helicopter domains, optimization researchers will typically have formal training in optimization methods but little grounding in the problem domain.

In real-world settings these roles are not well-defined. Each role can be filled by a single person, or by a large team of developers and domain experts. MDPvis' target users are domain experts that are willing to spend time learning the MDP formulation, software developers tasked with developing an MDP, and optimization researchers tasked with optimizing a policy for the MDP.

## 4. Related work

Many problems have been formulated as MDPs, including domains as diverse as RC car control [1,11], invasive species management [13], and real time strategy games [39].

While no general large MDP visualization has heretofore been proposed covering all these domains, there are numerous works that could be viewed as visualization for more restricted classes of MDPs.

Several works present systems for exploring decisions at a single time step. Broeksema et al. [10] give a decision analysis tool to examine recommendations made by an expert system. Decisions are plotted as Voronoi diagrams by means of Multiple Correspondence Analysis (MCA), which is a version of Multidimensional Scaling (MDS). The Voronoi diagrams lack a comprehensible coordinate system in two dimensions, but adjacency of attributes plotted over the diagram show how the decision variable changes as other attributes vary.

MDP policies are often specified via learned classifiers. Effectively debugging classifiers is an active area of research, but published research does not address debugging classifiers for MDPs. Groce et al. [17] explore methods for prioritizing classified data points for user inspection. Once a datapoint is selected the end user can decide whether they agree with its classification. The user debugs the classifier by correcting data labels, which leads to an update of the model. This debugging strategy assumes that the only form of error was an incorrect data label. In MDPs, however, there are no labels on the data, and the central testing question is whether the simulator and policy are generating the right data to begin with.

Kulesza et al. [19] includes a classifier debugging system for email messages. The user provides model feedback and correction through an interactive bar chart for a naive Bayes model. Kulesza et al. selected naive Bayes for its interpretability since many classifiers are too complex for end users to understand.

Migut and Worring [24] compose several information visualizations into a visual analytic dashboard for exploring a dichotomous choice as determined by a machine learning classifier. The system does not examine multiple sequential timesteps.

In ensemble visualization, the goal is to provide a compact representation of many predictive models of a singular ground truth [29]. Uncertainty in the predicted result is reduced by viewing a visualization of model agreement. In contrast, the uncertainty in MDP visualization arises from the stochastic responses of the world as the agent acts upon it. Ensemble visualization requires building consensus, but MDP visualization requires exploring the complete set of the world's potential responses to a policy.

A noteworthy visualization for natural resource management, Vismon [7], gives an interface for exploring tradeoffs in a set of management choices for an Alaskan fishery. Here the fishery manager filters 121 different management choices derived from varying two management parameters. The manager cannot view any management options that are not pre-computed before selecting a policy.

Simulation steering is one branch of visualization that attempts to bring the user into a optimization process by allowing the user to select actions at each timestep as the simulator executes. Simulation steering for epidemic response decisions in Afzal et al. [2] show individual outcomes through time. The user can change decisions at various points along a future rollout to see how the mortality rate responds. This visualization provides user-based optimization for a deterministic MDP.

Waser et al. [36] give another simulation steering visualization, “World Lines,” that invites users to control emergency response for flooding events. This visualization generates a small set of alternative

futures based on an action in the present. Subsequent versions of to World Lines [38,31,32,37] support stochasticity through secondary simulation controls (random levee breach locations) on the probabilistic parameters of the model, but the visualization does not center on machine learned policies.

In contrast to the current approaches in the literature that attempt to give the best visualization possible for a particular problem domain, our approach defines the visualization in terms of the formal properties of a class of problems. While our analysis focuses on the core MDP formulation, it is possible to use MDPvis with broader classes of MDPs, including ones with *partial observability*, *continuous-time*, *continuous-actions*, and *multiple agents*.

*Partially observable MDPs* (POMDPs) generalize by hiding state variables from the optimizer. For example, a helicopter simulator will simulate wind effects on the helicopter's position while not giving the optimizer access to wind data because a physical helicopter will not know the velocity of wind impacting its frame. When optimizing POMDPs from Monte Carlo rollouts, the simulator will typically strip hidden variables after they are simulated. While this may be necessary for faithfully simulating the operational conditions of the policy, it is often possible to visualize and test hidden variables because they are known to the simulator.

In *continuous-time MDPs*, the action can be selected at any time scale. For instance, in a helicopter domain the rotor's angles could change at any fraction of a second. MDPvis assumes actions occur at consistent time scales, so to visualize continuous-time domains it is necessary to discretize time steps. In the helicopter domain, this would take the form of selecting a sampling frequency within the simulator that saves the state and action information. MDPvis handles *continuous actions* like changing the angle of the rotor blade by treating them as a continuous state variable.

*Multi-agent MDPs* model the interaction of multiple decision makers. Each decision maker can have its own reward and policy function, but exist within the same MDP simulator. In a helicopter domain, this could take the form of two helicopters engaged in combat maneuvers. MDPvis can support multi-agent MDPs by having separate variables and parameters for each agent, but a visualization intended specifically for multi-agent MDPs would be valuable future work.

## 5. Visual encoding of MDPs

Our visual encoding for MDPs in Fig. 2 is shown in four parameter controls and three visualization panels. The controls give the reward, model, and policy parameters that are exposed by the MDP's software. These panels are cached in an exploration history that records the parameters and rollouts computed by the MDP. Here we explain the visual interface of MDPvis, followed by implementation details in Section 6. Each of the following subsections describes the example in Fig. 2.

### 5.1. Parameter panels

**Rewards Specification**: $R(s, a)$

Policy optimization is highly sensitive to changes in the parameters of the reward function. To explore these changes we expose the set of real valued reward parameters specified by the MDP as a list HTML input elements.

Whenever the reward function parameters change, elements of MDPvis related to optimality and expected value are no longer valid. The user interface updates to offer buttons for optimizing a new policy and generating rollouts.

**Model Definition**: $P$.

The MDP's simulator may expose model parameters to tune the system, or eliminate complexity for testing a specific component. These can include modifiers of the transition function, the total number of transitions to simulate (otherwise known as the horizon or sampling depth), the number of potential futures to sample (otherwise known as the sampling width), modifiers to the initial state distribution, flags for deactivating parts of the model, and fidelity switches for trading execution time for higher fidelity simulations.

A second purpose of the model parameters is to expose elements of the MDP's optimization algorithm to the user. Many MDP optimization algorithms are highly sensitive to parameters for learning rate, search depth, heuristics, convergence tolerance, and optimality requirements. Selecting a reasonable set of these parameters is often an iterative process.

When these parameters change, MDPvis enables buttons for optimizing a new policy and/or generating new rollouts.

**Policy Definition**: $\pi(s)_{\mapsto} a$

The policy controls specify the current policy. If the user chooses to optimize then they can explore the sensitivity of the policy to changes in the model or rewards. Checking this linkage between parameters and policy determines whether the policy is stable for minor changes in the reward function or whether the policies earn reward via different parts of the reward function.

Just as was the case in the prior sections, it is appropriate to present this control as a set of user-editable real numbers. When the policy is represented by parameters that have no human interpretable meaning as is the case with neural networks and sufficiently large decision trees, then this simplistic policy representation is no longer appropriate. While the visualization still answers many testing questions without rendering the policy parameters, we believe the work of Broeksema et al. [10] could be a good stand-in for this area when the user does not need to modify the policy.

When the user updates the policy function it is necessary to generate additional Monte Carlo rollouts for visualization in the areas below.

**Exploration History**

As the user repeatedly generates sets of rollouts under different parameter settings they may want to return to a prior parameter set to continue exploration from that point or to compare the sets of rollouts experienced under the prior parameter set. Here we add two buttons for every set of rollouts that have been generated. One button will reload the prior set of parameters and the rollouts that they generated into the visualization. The other button will put the visualization into "comparison mode," which displays the difference between two rollout sets in the visualization areas.

When in comparison mode the reward, model, and policy parameters cannot be edited since they display the differences in the parameter values between the current set of rollouts and the one being applied as a comparator.

More information about the comparison mode is included in the following visualization areas.

### 5.2. Visualization panels

Having specified all the parameters, we can request rollout sets from the MDP simulator and display them in the visualization areas. The first two visualization areas show sets of rollouts and support applying filters to the current rollout set. When we apply filters to the rollout set in one visualization area, the displayed rollout sets update throughout the visualization, i.e. the visualization is cross-linked.

**State Variable Distribution at Event**: $P_{n|\pi}$

We show the distribution of states at a particular timestep as a histogram. The user can select a range of values in the histogram to filter rollouts. This supports a global-to-local [33] testing process where exploration starts with an overview of all rollouts before drilling down into specific rollouts. When the user drills down to specific rollouts the count of filtered rollouts is shown as the unfilled portion of the histogram bar.

When the visualization is in comparison mode the histograms transform into a bar chart showing the difference in counts for the bins

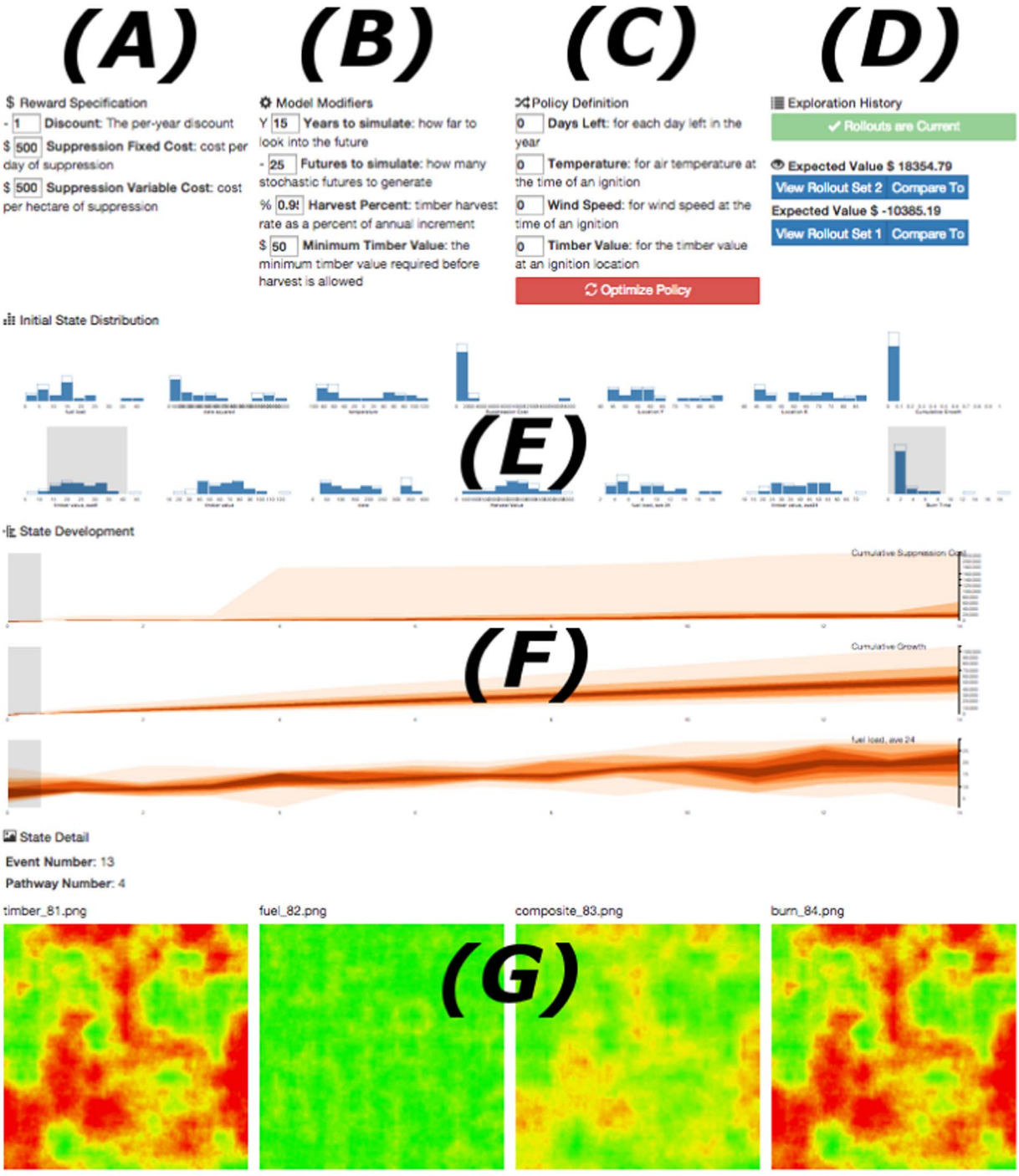


**Fig. 2.** A high level overview of the Markov Decision Process visualization prototype: MDPvis. The top row has the three parameter controls for (A) the reward specification, (B) the model modifiers, and (C) the policy definition. A fourth panel gives the history of Monte Carlo rollout sets generated under the parameters of panels (A) through (C). Changes to the parameters enable the optimization button found under the policy definition and the Monte Carlo rollouts button found under the Exploration History section. The visualization has two buttons in the History panel for each set of Monte Carlo rollouts, one for visualizing the associated Monte Carlo rollouts and another for comparing two sets of rollouts. Below the control panels are visualization areas for (E) histograms of the initial state distribution, (F) fan charts for the distribution of variables over time, and (G) a series of individual states rendered by the simulator as images. For a readable version of the visualization we invite users to load the visualization in their browser by visiting *MDPvis.github.io*.

between the two sets of rollouts.

**State Variable Evolution**: *P*.

An important question when testing an MDP is "Do the state distributions reflect the real-world? " (See Table 1). In MDP problems, there can be many variables that evolve over time to produce a distribution of outcomes at timesteps. In this area we represent distributions as fan charts giving the deciles of variables across rollout timesteps.

To produce the fan chart, we first plot a lightly colored area whose top and bottom represent the largest and smallest value of the variable in each time step. On top of this lightest color we plot a series of colors increasing in darkness with nearness to the rollout set's median value for the timestep.

By giving the percentiles, we are able to show both the diversity of outcomes and the probability of particular ranges of values. If the number of rollouts is small enough to avoid the visual clutter of many

intersecting lines, we render the rollouts as a line chart.

The user explores the conditional state distributions, $P(S|S_n \in F)$ by filtering ($F$) the rollouts. The filters between the fan charts and histograms are cross-linked. When filtering the fan chart the filters in the corresponding histogram updates, and vice versa. Changing the timestep of the fan chart's filter updates the histograms to the newly selected timestep.

When in comparison mode, the color extents are plotted by subtracting a color's maximum (minimum) extent from the corresponding maximum (minimum) extent of the other fan chart. Figs. 6,7, and 15 shows fan charts rendered in comparison mode.

Once the user filters enough rollouts the fan charts switch to line charts. Clicking on one of the lines in the line charts requests the state detail.

**State Detail**:$S_{i,j}$

We considered but ultimately rejected rendering the details of individual rollouts with a scatterplot matrix, parallel coordinates, Multi-Dimensional Scaling (MDS), or Multiple Correspondence Analysis (MCA). We found all these approaches would unnecessarily tie the MDP practitioner to an inadequate representation. Instead, we allow the MDP simulator to render a two dimensional array of images or videos that will be shown at the bottom of MDPvis.

For example, in the wildfire domain the state of the world is captured by four images of the landscape's timber and fuel values. Our co-authors in forestry were already rendering these landscapes, as shown in Fig. 2, so we integrated MDPvis with these standard visualizations. Non-spatial MDPs would not render landscapes, but they could return a visualization relevant to their practitioners.

### 5.3. Parameter space analysis (PSA) examples

Here we demonstrate the functionality of MDPvis for the PSA categories given by Sedlmair et al. [33] with fictitious examples from a wildfire domain. The intent of these examples is to illustrate how visualization with MDPvis is analogous to unit testing with visual expectations. In some cases these expectations could be written in closed form as traditional unit tests, but traditional unit tests presume the developer has the resources, expertise, and foreknowledge to codify the expectation. We expand on the examples of this section with real-world bugs in Section 7.

*Testing Sensitivity with Interaction*: Testing Question 26 in Table 6 asks "What are the most important drivers of policy? " Policies are typically functions mapping states to actions. This is an instance of a function visualization problem of $|S|$ inputs and $|A|$ outputs. However, a policy's tendency to inhabit a particular region of the state space means we do not need to examine how the policy function maps *all* states to actions. We can focus on the states produced by Monte Carlo rollouts under the policy function. Testing a policy's sensitivity to the state variables involves filtering (excluding) actions to see which state variables in the rollout set mapped to the unfiltered actions. Fig. 3 shows a visual expectation next to a buggy result.

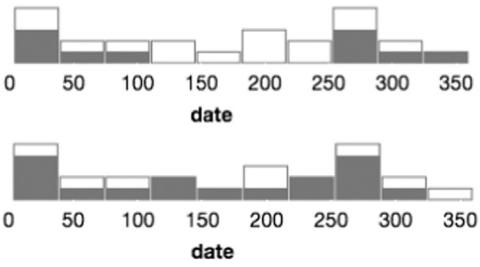


**Fig. 3.** We generated two histograms in MDPvis showing the dates a fire started on a landscape, then filtered the rollouts based on the action taken in this time step. In both these charts the filled portion represents fires that were allowed to burn unsuppressed, and the unfilled portion represents fires that were suppressed. Since ignition date is related to weather, we expect (top) to allow more fires to burn at the start and end of the season, but not in the middle of the season when the landscape is much drier. A buggy result (bottom) shows no apparent relationship.

*Testing Optimization by Optimizing*: Question 17 in Table 4 asks, "Is the policy converged to an acceptable local optimum? " The expected reward of MDPs are typically non-convex functions, meaning the optimization can terminate with policies that are clearly suboptimal when viewing how the states map to actions. For bad policies we want to know if the optimizer received an unlucky set of Monte Carlo rollouts, started from a bad initial policy, or broke due to a buggy implementation. To detect these three cases, we can change the parameters of the MDP from to form an expectation of how the optimizer should improve the policy.

For example, let's use optimization to test the optimizer of a wildfire MDP. Specifically, let's start with a landscape covered by a species whose timber is not valuable at harvest time. We would expect suppression expenses to dominate timber harvests and produce a policy that never suppresses fires as shown at the top of Fig. 4. If we change the parameters of the MDP so that harvests are worth billions of dollars, we would expect a newly optimized policy to begin suppressing fires (the bottom of Fig. 4).

If the optimized policy does not change in response to these large reward function changes, then there is likely a bug.

*Testing Outliers with State Detail*: Question 8 in Table 2 asks, "Are the most extreme state transitions realistic? " Many formal guarantees of correctness or optimality are expressed in terms of the maximum reward for a single time step. With optimization algorithms it is important to validate the most extreme rollouts since they have disproportionate effect.

In the wildfire domain, we may ask whether the most extreme wildfires respect firebreaks established by previous fires. We can generate state details for the largest fires by filtering the time step with the largest fires in the fan chart. Once fewer rollouts are displayed, we can generate the state snapshots whose expectations and buggy results are shown in 5.

*Testing Partition by Comparing*: Question 14 in Table 3 asks, "Do policies differ in their resultant state distributions as expected? " Domain experts may have mental models of MDP behavior subject to various policies. These policies have expected partitions of the state space. In MDPvis the domain expert can compare two different state partitions through time using the fan charts. Fig. 6 shows a comparison's expectation and a buggy result.

*Testing Uncertainty by Re-Parameterizing*: Question 20 of Table 5, "How certain is the policy function about a specific choice? " We don't want the policy to change if we change the simulator's parameters within the range consistent with real-world data.

For example, fire management practices are more effective today than in the year 1900, but what if fire fighting practices continue to improve (a form of structural uncertainty)? Fire managers will want to know if the policy optimized for a future with better firefighting technology will be different from the policy optimized for a continued status quo. MDPvis explores structural uncertainty by permitting the user to change parameters, such as "suppression effect," and re-optimizing. Fig. 7 shows two potential results of this analysis.

*Testing Fit by Augmenting*: Question 4 of Table 1 asks, "Do simulated transitions result in realistic state distributions? " Many domains will have datasets for historical policies. These data give distributions of state variables that can be compared against the simulated distributions. We can add historical distributions by augmenting the dataset with a derived variable for each state variable having historical data. This technique is best demonstrated with an example.

In the wildfire domain, we take a variable for which we have many real-world samples like "Growth Percentile". Based on these historical data, we can assign a "Historical Growth Percentile" as the percentile value of each Growth Percentile value within the historical dataset. Fig. 8 shows the visual expectation in the fan chart, which has

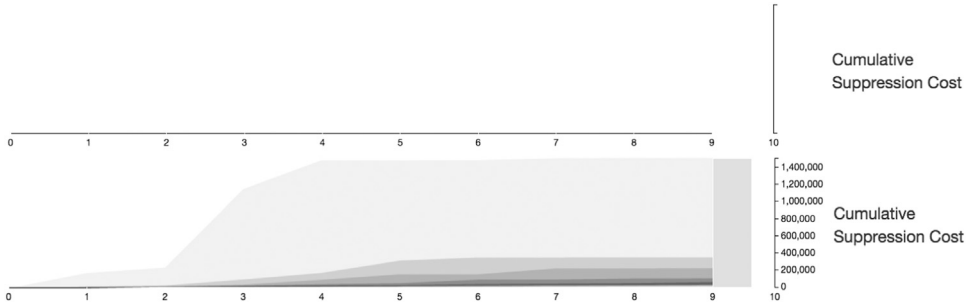


**Fig. 4.** When we make timber harvest more valuable, we would expect an optimized policy to suppress more fires. Here we optimize a policy after changing the reward function to increase timber value. We expect the suppression expenses of the top chart to update to the bottom chart after optimization.

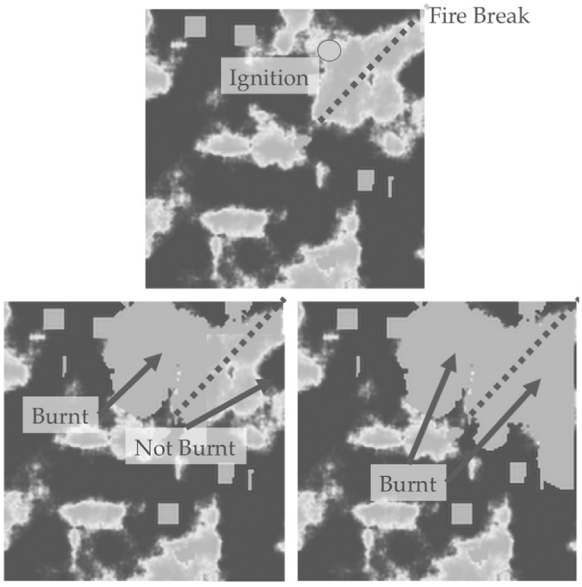


**Fig. 5.** The top row contains a single starting state and the bottom row contains an expected result state (left) next to a buggy result (right). The dotted lines show a "fire break" formed by a previous fire that should prevent future fires from spreading directly across the landscape. By filtering to this outlier we can check whether the largest wildfires are respecting fire breaks.

consistent percentiles across all time steps. A buggy result given in Fig. 8 shows the percentiles of the simulated dataset departing from the historical distribution.

## 6. MDPvis implementation

We built MDPvis as a data-driven web application. Building on the web application stack affords two primary benefits. First, Brehmer and Munzner [8] identify sharing as an important feature to implement and the ubiquity of web browsers makes it an ideal platform for collaboration. Second, the web stack emphasizes standard data interchange formats that ease integration with MDP simulators and optimization algorithms. We identified four HTTP requests *(initialize, rollouts, optimize, and state)* that are answered by the MDP simulator. These requests do not assume a particular domain or implementation language. In most cases the requests should be able to interface with the MDP simulator and optimizer using the same command-line client they would typically implement for testing a domain.

MDPvis issues the following HTTP requests to the MDP simulator and optimizer:

1. */initialize* – Ask for the parameters that should be displayed to the user. The parameters are a list of tuples, each containing the name, description, minimum value, maximum value, and current value of a parameter. These parameters are then rendered as HTML input elements for the user to modify. Following initialization, MDPvis automatically requests rollouts for the initial parameters.
2. */rollouts?QUERY* – Get rollouts for the parameters that are currently defined in the user interface. The server returns an array of arrays containing the state variables that should be rendered for each time step.
3. */optimize? QUERY* – Get a newly optimized policy. This sends all the parameters defined in the user interface for the MDP and the MDP's optimization algorithm returns a newly optimized policy.
4. */state? IDENTIFIER* – Get the full state details and images after the user selects a rollout.

All relevant languages have web server libraries that can be integrated with the MDP's code base for serializing Monte Carlo rollouts. We integrated the wildfire domain with the visualization by adding a serialization library (one line of code) and modifying a dummy data file to initialize the domain. The most challenging integration task was parsing the HTTP parameters into the expected data types for the simulation code and writing the loop to invoke the simulator multiple times.

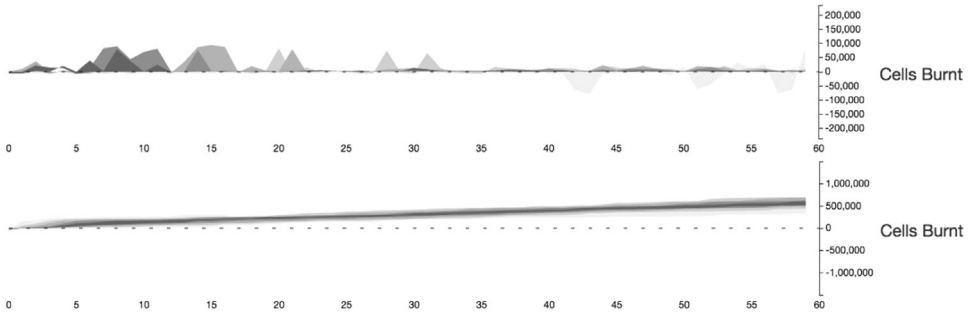


**Fig. 6.** These charts show two comparisons of the number of cells burnt for policies that allow all fires to burn or suppress all fires. When the color in the chart is above the center line, the let-burn-all policy has more burnt pixels in that time step. We would expect (top) the number of cells burnt to be greater initially in the let-burn-all policy, but for the number of cells burnt to decline as fires reduce fuel levels. A buggy result (bottom) shows a steady increase in cells burnt relative to the suppress-all policy.

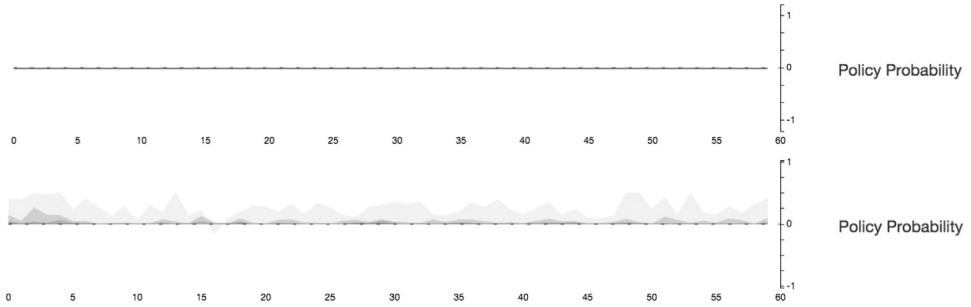


**Fig. 7.** Since we are uncertain how fire suppression effectiveness will change through time, we want policies optimized under different effectiveness levels to be similar. We can explore policy similarity by comparing policies optimized under two different effectiveness parameters. The top chart is a comparison fan chart showing a single line straight across at zero. This means there is no difference in the policy probabilities between the policies optimized under different suppression effectiveness parameters. The second chart gives a buggy result, showing many differences in the policy probability between the two parameterizations.

### 6.1. Integration with reinforcement learning frameworks

Machine learning researchers typically evaluate new algorithms on interesting challenge domains or toy domains that illustrate some capability or failure of an algorithm. The problem domain collections of Geramifard et al. [16], Bellemare et al. [4], Duan et al. [14], and Brockman et al. [9] aim to unify the reinforcement learning research community behind shared implementations with consistent APIs. We integrated MDPvis with two reinforcement learning frameworks to test our claims of the implementation's generality.

**RLPy** (Reinforcement Learning Python) is a collection of 26 problem domains. We integrated three of RLPy's 26 problem domains, including the toy domains of Mountain Car and Grid World, and the more challenging domain of HIV Treatment introduced by Ernst et al. [15]. The web server and data munging code developed for the RLPy domains can be found at McGregor [22] and is easy to adapt to the 23 additional domains in RLPy. We discuss our experience with three of the domains below.

*Mountain Car* models an underpowered car trapped between two hills. Successful policies in Mountain Car store energy by rocking the car back and forth between the hills until it has enough momentum to escape. We created a policy function for Mountain Car with a parameter controlling the probability of reinforcing the current direction of travel by accelerating in that direction. We exposed this parameter to MDPvis to explore different policy outcomes. The domain has a parameter for "noise," which gives the probability of randomly assigning the action taken in a state transition.

After integrating RLPy's Mountain Car implementation with MDPvis, we discovered the capitalization of the noise parameter in the constructor differed from the default noise level. The result was a failure to update the domain to the selected noise value. We discovered this bug when testing RLPy's integration with MDPvis. Fig. 9 shows two fan charts for the car's x position. The top chart has the domain as initially visualized by a fan chart in MDPvis. The second fan chart in Fig. 9 shows the fan chart after we fixed the bug. This bug would not be apparent to Mountain Car's developers because their visualization tools do not show the absence of variance. We subsequently submitted a bug fix to the RLPy maintainers, who merged the fix a day later. For more details, see McGregor [20].

*Grid World* models navigation tasks in two dimensions. Researchers often use it to explore an algorithm's ability to avoid obstacles (pits) and learn increasingly long series of actions before reaching a terminal reward. Unlike Mountain Car and our wildfire domain, which only have two actions, the Grid World domains have an action for each of the four cardinal directions. To accommodate filtering rollouts based on action selection we munged the actions to have their own indicator variable for each of the cardinal directions. Fig. 10 shows the resulting fan charts for the actions as selected for a set of rollouts generated by a random policy.

Grid World domains also illustrate the limitations of using a single visualization for all MDP domains. Grid Worlds have a natural 2-dimensional representation (a grid) that could be extended to support the interactions of MDPvis. While our current design can render the grid generated by RLPy in the state detail view, we implemented MDPvis so developers can contribute different representations for either the fan charts or the histograms.

*HIV Treatment* is a medical treatment domain introduced by Ernst et al. [15]. HIV treatment has six immune system state variables that are measured every 5 days over 1000 days:

- $T_1$: the number of healthy CD4$^+$ T-lymphocytes
- $T_2$: the number of healthy macrophages
- $T_1^*$: the number of infected CD4$^+$ T-lymphocytes
- $T_2^*$: the number of infected macrophages
- $V$: the number of free virus particles
- $E$: the number of HIV-specific cytotoxic T-cells

HIV Treatment policies dynamically select one of four drug therapy options, including reverse transcriptase inhibitor (RTI), protease inhibitor (PI), a combination of these drugs, or no treatment. Each

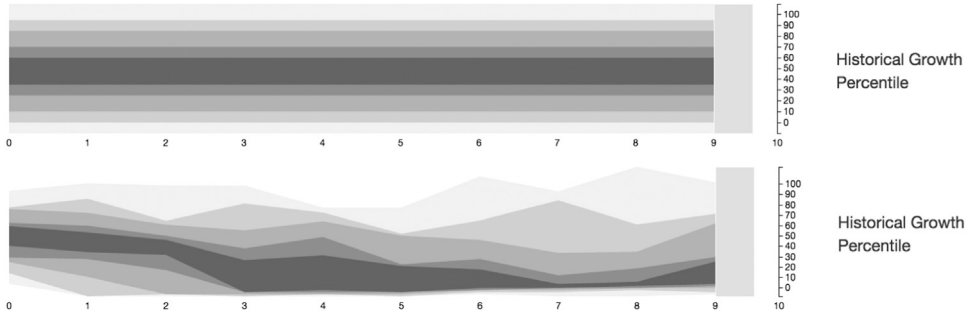


**Fig. 8.** Both charts give the simulated percentiles within the historical dataset. We expect (top) the percentile lines to go straight across the fan chart and meet the y-axis at the percentile value matching the color. A bug (bottom) is indicated by a chaotic distribution that does not match up with the proper percentiles.

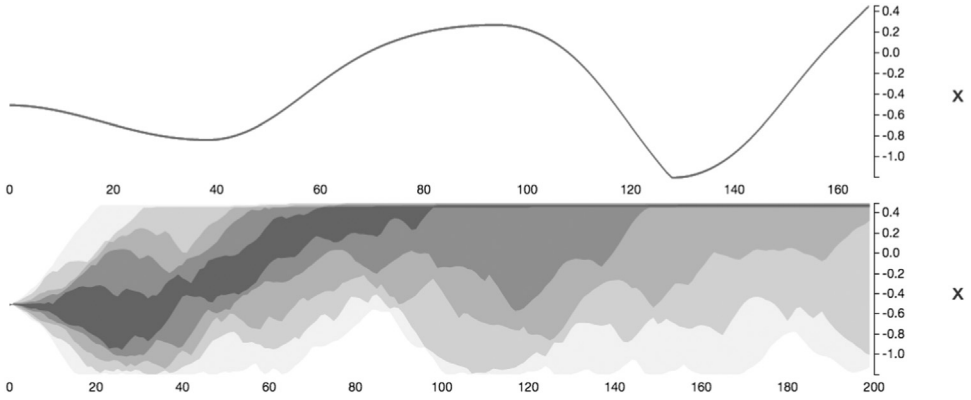


**Fig. 9.** The two charts show the Mountain Car domain before (top) and after (bottom) fixing the noise parameter, which is set to 0.2 in both cases. The top chart shows no variation across rollouts despite the expectation that the actions would stochastically change in proportion to the noise parameter.

drug affects a different step of the virus's replication process. Successful policies produce a drug schedule that jointly minimizes virus count and medication side effects. A typical real world policy includes "Structured Treatment Interruption" (STI), which cycles patients between treatment and non-treatment periods.

We created a parameterized policy class that mimicked our interpretation of STI. The parameters are independent probabilities of administering RTI and PI. We automatically administer both HIV drugs if the patient's infected cells or free viruses are "spiking".

Fig. 11 shows fan charts under three different policies. The top chart is for a policy with high probabilities of administering both drugs, and the bottom chart only administers drugs under the "spiking" condition. Since the domain is implemented as a deterministic integration of differential equations, we can see the outcomes converge to a single patient history as the policy becomes more deterministic. The scale of the spikes on the top chart is also greater than the spikes on the charts administering less medication. This suggests that the medication carries risks requiring further study.

The second set of domains we integrated was through the **OpenAI** "**Gym**" framework of Brockman et al. [9]. OpenAI Gym is both a collection of problem domains for researchers, and a leaderboard website showing the relative performance of optimization algorithms developed by researchers. It includes eight "environments" that range from the simplistic classical control problems to Atari 2600 games from Bellemare et al. [4]. The environments provide a total of 176 domains.

OpenAI Gym includes custom rollout animations for domains.

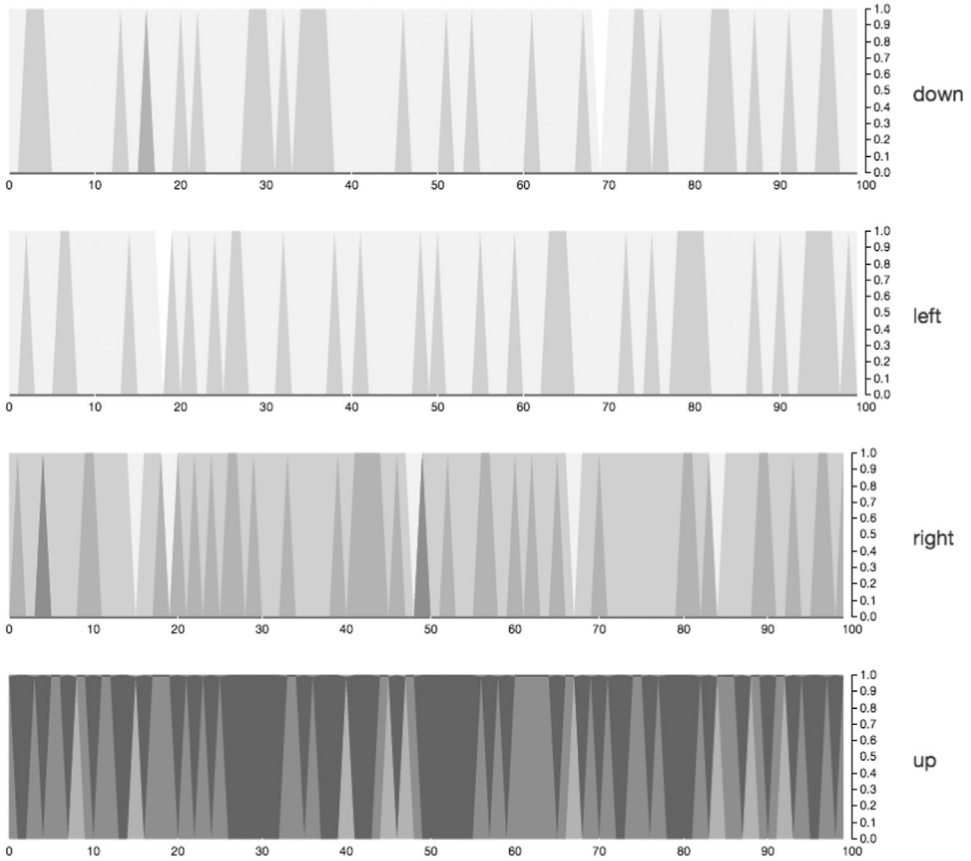


**Fig. 10.** Four fan charts represent each of the actions taken for a Grid World domain. In this grid world map the agent starts in the lower left corner with a pit immediately to its right. Successful policies will move up, right, then down to reach the goal grid cell. We created a stochastic policy that more frequently moves "up, " followed by less frequent moves in other directions. The fan charts show our simplistic policy does not change action selection probability through time. If we want to know what actions lead to or from a particular state, we can apply filters to state variables and view the action fan charts.

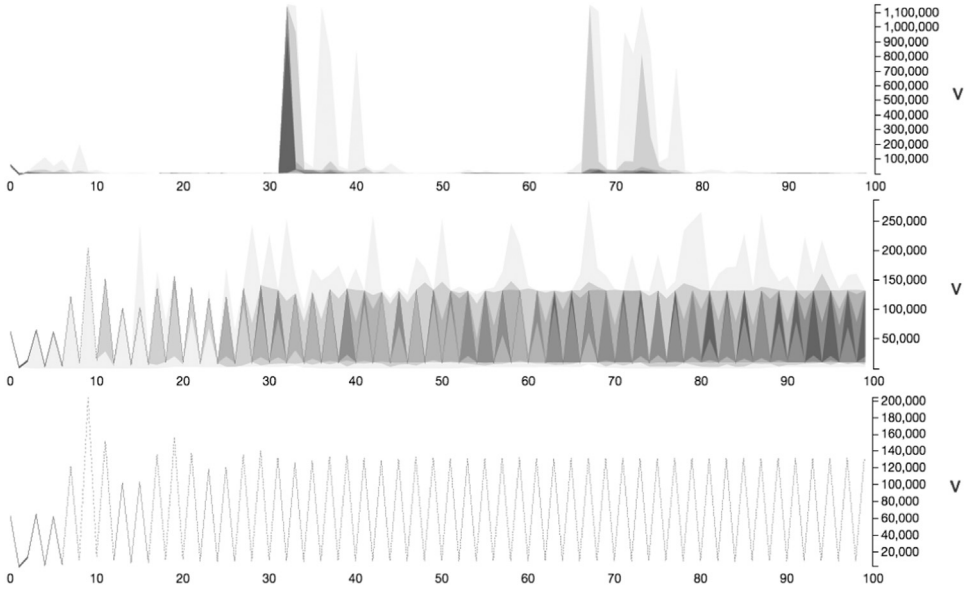


**Fig. 11.** The virus counts (V) for patients under three policies. The top fan chart was generated by a policy with a probability of administering RTI and PI of 0.4. The middle chart has the probabilities of 0.01 and the bottom chart only administers the drug when the virus count is spiking. You can see the top chart largely keeps the virus to a minimum. The middle chart is dominated by the automatic administration of RTI and PI when the patient is spiking, but the stochastic administration of the drugs in other time steps produces some variation. The last chart shows the determinism of the HIV domain. All the patients have the same patient histories when the policy and domain are both deterministic.

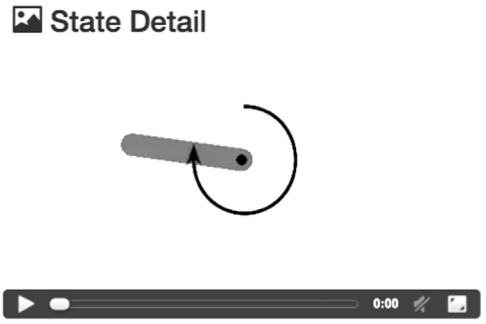


**Fig. 12.** MDPvis displays a media player for the video generated by the Pendulum domain included in OpenAI Gym. This video is displayed by clicking the rollout associated with it in the Fan Charts.

Developers can upload videos of these animations to OpenAI's servers for display with their leaderboard entries. We integrated these videos with MDPvis through the state detail panel.

OpenAI Gym problem domains can be classified as classic control problems, or raw perception problems. Optimization in classic control problems use relatively short state vectors where each entry in the vector is a "feature". These vectors may have fewer dimensions than the complete state space, as is the case in our wildfire example that optimizes on summary statistics of a landscape rather than the landscape directly. Raw perception optimization approaches problems from the perspective of the human eye and shows the optimization algorithm complete state information in the form of an image.

Optimization in raw perception domains blend selecting actions and learning what combinations of pixel values are important. Machine learning research labels the task of finding relationships in the data that are important for decision making "feature learning". The advantage of feature learning is the ability to find complex relationships among variables that are not readily detectible or compactly describable by humans. Complex features represent a challenge for interpreting policies is an active area of research by Zahavy et al. [40].

We integrated all four of OpenAI Gym's classical control domains, which include Pendulum, Acrobot, Cartpole, and Mountain Car. We also integrated all 55 Atari games bundled into OpenAI Gym by the Arcade Learning Environment of Bellemare et al. [4]. The web server and data munging code developed for both these domain collections share a single lightweight web server found at McGregor [21]. The server integrates with each domain by specifying the name of the domain and the names of the visualized state variables when starting the server. This makes it possible to switch between the domains without touching the web server's implementation. Next we highlight two of the domains we integrated.

*Pendulum* is a domain concerned with balancing a pendulum by applying torque either left or right so as to maintain the position of the pendulum. Unlike all domains presented to this point, Inverted Pendulum has a continuously valued action space bounded by the maximum and minimum torque values. Continuous action spaces are typically more challenging to optimize than discrete action spaces, but they have an intuitive mapping within MDPvis. The histograms and fan charts both vary continuously.

OpenAI Gym includes a visualization for pendulum that shows the position of the pendulum and the force applied. Fig. 12 shows the video for a selected rollout in the State Detail area of MDPvis.

*Ms. Pac-Man* is a perception domain based on the popular 1982 title of the same name. The goal is to produce a series of actions that avoid ghosts while consuming all the dots on the screen. To support perception domains like Ms. Pac-Man we extended MDPvis with the ability to define variables by images. Fig. 13 shows a time series defined by an image saved from the initial game state. We use this image to generate a similarity score for all of a rollout's frames, which is displayed in the fan chart next to the image.

Our similarity metric comparing pixels at consistent screen coordinates is not tied to the visual features that the optimization algorithm determines are important. The work of Zahavy et al. [40] show we can use components of neural networks for both selecting the images and determining the similarity among images based on what the network is paying attention to. We leave this effort to future work.

### 6.2. Debugging capabilities

Bugs may result from bad implementation (code) or from a bad set of parameters. In the case of parameter choice, it is possible to fix (debug) the problem without leaving MDPvis. However, non-programmer domain experts cannot fix bad implementation without sharing a bug report with the software developer. Thus, MDPvis supports code

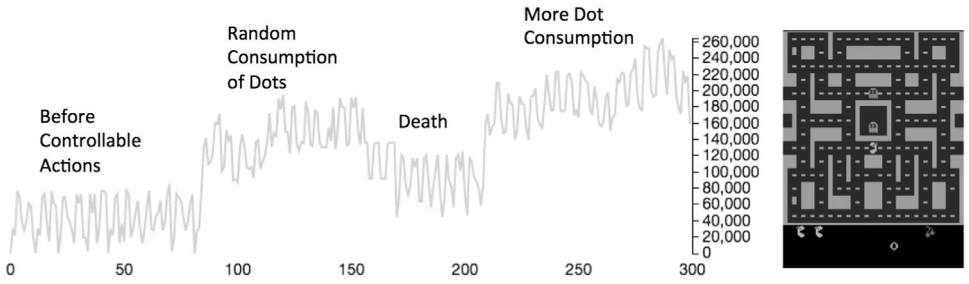


**Fig. 13.** An annotated time series displayed within MDPvis for the Ms. Pac-Man Atari domain. The graphic on the right side is the image used to generate similarity scores by summing the difference in each pixel between the displayed image and the image generated within the rollout. Many Atari games render different game state every other frame, which accounts for the jitter in the time series. The text annotations (from left to right) show the initial phase of the game, followed by a phase in which the ghosts are hunting the dot-consuming player, before a ghost catches the character and the character and ghost positions reset to the position found in the image on the right.

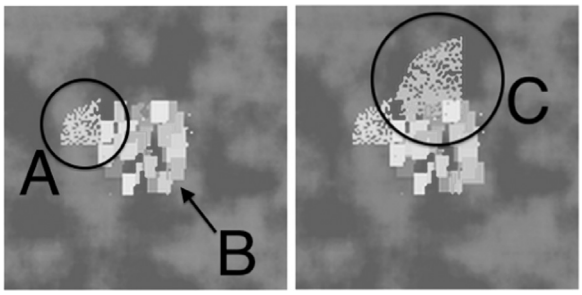


**Fig. 14.** Two sequential spatial snapshots of timber values for a state transition that includes one of the largest fire loss events from 200 rollouts of 60 years. The management area is visible in the center due to the rectangular timber harvests. These rectangular harvests obscure the irregular boundary of small fires. (A) shows a medium size fire obscured by a mixture of small fires and timber harvests in (B). The straight edge of the largest fire introduced in (C) clearly shows the fire is not spreading in all directions.

debugging through collaboration, but more robust integration of MDPvis with the software developer's Integrated Development Environment (IDE) and version control would expand MDPvis's debugging capabilities. In particular, MDPvis is not currently aware of versions of the MDP simulator or optimizer. As code changes, it would be useful to treat each build as a selectable parameter in the interface. This would allow for more robust collaboration between team members as implementations change.

## 7. Use-case study: wildfire suppression policies

We arrived at our generalized visualization by following Munzner's nested model [25] for the problem domain of Wildfire Suppression then generalized the results to all MDPs. Since the Wildfire Domain has high-dimensional states, it is representative of a particularly challenging group of MDPs.

We expressed the suppression policy as a differentiable function with interpretable coefficients. Each coefficient resembles a weight that would be generated by a logistic regression. Increasing a coefficient's value makes it more likely that a wildfire will be suppressed for increasing values of the corresponding state variable. We use policy gradient methods [35,3,28] to optimize new policies, which have the advantage of being both fast and likely to improve the policy from an uninformed policy for all sample sizes.

Our wildfire suppression domain combines models for fire spread, vegetative growth, weather, suppression effectiveness, suppression cost, and harvest. One of the most difficult questions any landscape modeler faces is how much realism is necessary to address the hypothesis at hand? More realism means greater development, computer processing, and computer memory requirements, while less realism can lead to results that are nonsensical at best and misleading at worst. The fact that the state space is so large makes it difficult to comprehend how each assumption affects the final outcome. MDPVIS allows us to explore ways in which the simulator is failing, either through hard-to-find technical bugs or "garbage in, garbage out" assumptions that are affecting the way we value different outcomes.

e used MDPvis in a use-case study to provide anecdotal evidence of the utility of MDPvis. This case study is based on user sessions with our forestry economics collaborators who formulated fire suppression policies as an MDP optimization problem. Throughout the design and development process, we worked closely with these domain experts, who are co-authors on this paper, to identify their needs as developers of MDP solutions. The analyses in this section were performed by these experts during their first use of MDPvis, in conjunction with a Ph.D. student from computer science who implemented MDPvis but did not contribute to the development of the MDP.

We detected bugs for most of the questions of Tables 1–6 and highlight interesting cases with their interactions under their corresponding analysis tasks below.

**Fitting:** Several failures to simulate real conditions were detected. Upon filtering the rollouts to the ones containing the most extreme fires, we examined the state details and found that the fires were not spreading east or south from the ignition site (see Fig. 14). We could only see this on the larger fires because the rectangular timber harvests were masking the unusual shape of the fire spread.

**Outliers:** A common real-world wildfire suppression policy suppresses the vast majority of wildfires so it forms a natural baseline for comparisons. To better illustrate the outcomes of a suppress-all policy, we compared it to a let-burn-all policy and found a surprising fact: the let-burn-all policy has higher expected reward than the suppress-all policy. This shows that either the models do not balance the various rewards of fire suppression properly, or a policy that is completely opposite from current forestry practices produces better outcomes. We found the reason for this counterintuitive result after filtering rollouts to only display outlier fires. Large fires immediately increase the harvest reward because the maximum allowable harvest is a function of tree growth. The harvest level is depressed without large fires creating the conditions for explosive post-fire growth.

**Partition:** When comparing two different policies (see Fig. 15) under otherwise consistent parameters, we observed a slight difference in the percentiles of the weather events. Since these weather events are exogenously determined by a random number generator with a consistent seed, this difference indicates that the random number generator is called differently depending on the action that is selected. Without fixing this bug we cannot compare a landscape's response to different policies under the same set of ignition events. Further, this means fire managers can choose the weather by running experiments under different policies until it experiences the best possible conditions.

**Optimization:** Although the policies reported by our policy gradient algorithm improved upon our naive baselines, we found it easy to improve upon the machine optimized policies by changing the policy parameters. This shows that the optimizer is failing to find a

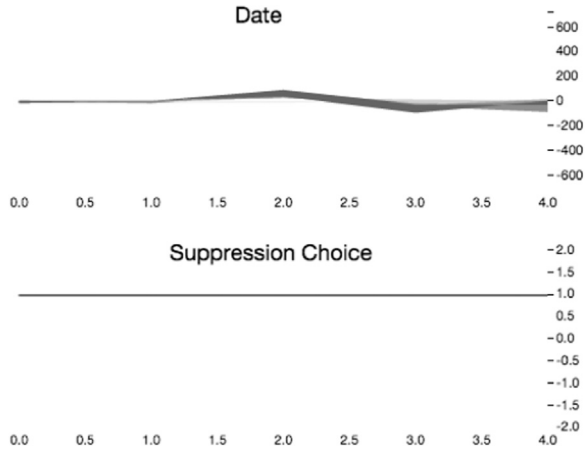


**Fig. 15.** Fan charts for the suppression choice and the ignition date shown in comparison mode for two rollout sets. We generated one rollout set under a suppress-all policy and a second rollout set under a let-burn-all policy. We confirmed that the proper action is being selected for each state by observing the differences in suppression choices is always 1. However, there is an unexpected difference in the dates, which should be consistent between the two rollout sets.

local optimum.

We were able to identify the most likely source of the problem: when we optimized policies for increasing rollout depths we found a runtime bug with our implementation of importance sampling that produces a division by zero. We hypothesize that this runtime fault is causing our optimization function's other anomalous results.

**Sensitivity:** When viewing the timber harvest chart in comparison mode, the lack of substantial differences in harvest volumes for different policies indicates that the harvest volume is not sensitive to the policy. Additional comparisons after fixing bugs showed harvests are only significantly impacted when old growth forest dominates the landscape and trees stop growing. Since the model rarely reaches 100% old growth, the harvest level is not sensitive to the policy.

**Uncertainty:** The invariant harvest reward means it is always better to let the wildfire burn. After using MDPvis with other US federal land wildfire simulators, we found this lack of uncertainty to be faithful to real world tradeoffs. This poses an epistemological problem to our team of wildfire suppression optimization researchers. Finding interesting research questions requires changing to a privately managed landscape, or including a case analysis of policies optimized for different harvest levels.

## 8. Conclusion and availability

This paper presented MDPvis, a domain-independent tool for supporting the testing and debugging of MDP simulation and optimization software. MDPvis employs a simple web service protocol to interact with the MDP simulator and optimizer, and supports many visual analysis tasks related to MDP testing. MDPVIS supports viewing rollout distributions over time and temporal comparisons between policies (either policies produced by the optimizer or policies designed by the user). When we integrated MDPvis with a simple reinforcement learning domain in a widely used research framework we unexpectedly found a bug. We presented a use-case study in which our users immediately discovered several serious bugs. We also discovered interesting behavior that is either a bug or an indication that real-world policies diverge significantly from the optimal policy. Our users report that MDPvis is already greatly accelerating their testing and debugging processes, and they are looking forward to applying it to other MDP simulators.

A live version of MDPvis, the source code, and integration instructions are available at *MDPvis.github.io*.

## Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 1331932.

## References

[1] P. Abbeel, M. Quigley, A.Y. Ng, Using inaccurate models in reinforcement learning, in: Proceedings of the International Conference on Machine Learning, 2006, pp. 1–8.
[2] S. Afzal, R. Maciejewski, D.S. Ebert, Visual analytics decision support environment for epidemic modeling and response evaluation, in: Proceedings of the IEEE Conference on Visual Analytics Science and Technology (VAST). IEEE, 2011, pp. 191–200.
[3] J. Baxter, P.L. Bartlett, L. Weaver, Experiments with infinite-horizon, policy-gradient estimation, J. Artif. Intell. Res. 15 (2001) 351–381.
[4] M.G. Bellemare, Y. Naddaf, J. Veness, M. Bowling, The Arcade Learning Environment: An Evaluation Platform for General Agents 47 (IJCAI), 2012, pp. 253–279.
[5] R. Bellman, Dynamic Programming, Princeton University Press, New Jersey, 1957.
[6] D.P. Bertsekas, D.P. Bertsekas, D.P. Bertsekas, D.P. Bertsekas, Dynamic Programming and Optimal Control 1, Athena Scientific, Belmont, MA, 1995.
[7] M. Booshehrian, T. Möller, R.M. Peterman, T. Munzner, Vismon: Facilitating analysis of trade-offs, uncertainty, and sensitivity in fisheries management decision making, in: Proceedings of the Eurographics Conference on Visualization 2012 (EuroVis 2012). Computer Graphics Forum, 2012, pp. 1235–1244.
[8] M. Brehmer, T. Munzner, A multi-level typology of abstract visualization tasks, IEEE Trans. Vis. Comput. Graph. 19 (12) (2013) 2376–2385.
[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI Gym (2016).
[10] B. Broeksema, T. Baudel, A. Telea, P. Crisafulli, Decision exploration lab: a visual analytics solution for decision management, IEEE Trans. Vis. Comput. Graph. 19 (12) (2013) 1972–1981.
[11] M. Cutler, T.J. Walsh, J.P. How, Reinforcement learning with multi-fidelity simulators, in: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) (1), pp. 3888–3895.
[12] M.P. Deisenroth, A survey on policy search for robotics, Found. Trends Robot. 2 (2011) (2011) 1–142.
[13] T. Dietterich, M. Taleghan, M. Crowley, PAC optimal planning for invasive species management: improved exploration for reinforcement learning from simulator-defined MDPs, in: Proceedings of the Twenty Seventh AAAI Conference on Artificial Intelligence, 2013.
[14] Y. Duan, X. Chen, R. Houthooft, J. Schulman, P. Abbeel, Benchmarking deep reinforcement learning for continuous control, in: Proceedings of the International Conference on Machine Learning (ICML-16). vol. 48., 2016, p. 14.
[15] D. Ernst, G.-B. Stan, J. Goncalves, L. Wehenkel, Clinical data based optimal STI strategies for HIV: a reinforcement learning approach, in: Proceedings of the 45th IEEE Conference on Decision and Control, 2006, pp. 667–672.
[16] A. Geramifard, R.H. Klein, C. Dann, W. Dabney, J.P. How, RLPy: The Reinforcement Learning Library for Education and Research, ⟨http://acl.mit.edu/RLPy⟩, 2013.
[17] A. Groce, T. Kulesza, C. Zhang, S. Shamasunder, M. Burnett, W.-K. Wong, S. Stumpf, S. Das, A. Shinsel, F. Bice, K. McIntosh, You are the only possible oracle: effective test selection for end users of interactive machine learning systems, IEEE Trans. Softw. Eng. 40 (3) (2014) 307–323.
[18] R.M. Houtman, C.A. Montgomery, A.R. Gagnon, D.E. Calkin, T.G. Dietterich, S. McGregor, M. Crowley, Allowing a wildfire to burn: estimating the effect on future fire suppression costs, Int. J. Wildland Fire 22 (7) (2013) 871–882.
[19] T. Kulesza, M. Burnett, W. Wong, S. Stumpf, Principles of explanatory debugging to personalize interactive machine learning, in: Proceedings of the ACM Conference on Intelligent User Interfaces, 2015.
[20] S. McGregor, (Pull Request) fixed self.noise capitalization, URL ⟨https://github.com/rlpy/rlpy/pull/10, 2015
[21] S. McGregor, Flask Server, ⟨https://github.com/MDPvis/gym/blob/feature-mdpvis-updated/flask_server.py⟩ 2016a.
[22] S. McGregor, Flask Server for RLPy, ⟨https://github.com/MDPvis/rlpy/blob/mdpvisV2Flask/flask_server.py⟩, 2016b
[23] S. McGregor, H. Buckingham, T.G. Dietterich, R. Houtman, C. Montgomery, R. Metoyer, Facilitating testing and debugging of markov decision processes with interactive visualization, in: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, Atlanta, 2015.
[24] M. Migut, M. Worring, Visual exploration of classification models for risk assessment, in: Proceedings of the IEEE Symposium on Visual Analytics Science and Technology (VAST), 2010, pp. 11–18.
[25] T. Munzner, A nested model for visualization design and validation, IEEE Trans. Vis. Comput. Graph. 15 (6) (2009) 921–928.
[26] A.Y. Ng, Shaping and Policy Search in Reinforcement Learning, Doctor of Philosophy, University of California, Berkeley, 2003.
[27] A.Y. Ng, A. Coates, M. Diel, V. Ganapathi, Autonomous Inverted Helicopter Flight Via Reinforcement Learning, Experimental Robotics IX, 2006, pp. 1–10.
[28] A.Y. Ng, M.Jordan, Pegasus: A policy search method for large mdps and pomdps, in: Proceedings of the Sixteenth conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 2000, pp. 406–415.
[29] K. Potter, A. Wilson, P.T. Bremer, D. Williams, C. Doutriaux, V. Pascucci, C.R.

Johnson, Ensemble-vis: A framework for the statistical visualization of ensemble data, in: Proceedings of the ICDM Workshops on IEEE International Conference on Data Mining, 2009, pp. 233–240.

[30] M. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, 1st edition, Wiley-Interscience, Hoboken, New Jersey, United States of America, 1994.

[31] H. Ribičić, J. Waser, R. Fuchs, G. Bloschl, E. Groller, Visual analysis and steering of flooding simulations, IEEE Trans. Vis. Comput. Graph. 19 (6) (2013) 1062–1075.

[32] B. Schindler, H. Ribičić, R. Fuchs, R. Peikert, Multiverse data-flow control, in: Proceedings of the IEEE Transactions on Visualization and Computer Graphics, vol. 19, 2013, pp. 1005–1019.

[33] M. Sedlmair, C. Heinzl, S. Bruckner, H. Piringer, T. Möller, Visual parameter space analysis: a conceptual framework, IEEE Trans. Vis. Comput. Graph. 20 (12) (2014).

[34] N.A. Stanton, P.M. Salmon, L.A. Rafferty, G.H. Walker, C. Baber, D.P. Jenkins, Human Factors Methods: A Practical Guide for Engineering and Design, 2nd edition, Ashgate Publishing Company, Burlington, VT, 2013.

[35] R.S. Sutton, D. Mcallester, S. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, Adv. Neural Inf. Process. Syst. 12 (2000) 1057–1063.

[36] J. Waser, R. Fuchs, H. Ribičić, B. Schindler, G. Bloschl, M.E. Groller, World lines, IEEE Trans. Vis. Comput. Graph. 16 (6) (2010) 1458–1467.

[37] J. Waser, A. Konev, B. Sadransky, Z. Horváth, H. Ribičić, R. Carnecky, P. Kluding, B. Schindler, Many plans: multidimensional ensembles for visual decision support in flood management, in: Proceedings of the Eurographic Conference on Visualization (EuroVis) 33, 3, 2014.

[38] J. Waser, H. Ribičić, R. Fuchs, C. Hirsch, B. Schindler, G. Blöschl, M.E. Gröller, Nodes on ropes a comprehensive data and control flow for steering ensemble simulations, IEEE Trans. Vis. Comput. Graph. 17 (12) (2011) 1872–1881.

[39] S. Wender, I. Watson, Applying reinforcement learning to small scale combat in the real-time strategy game starcraft:broodwar, in: Proceedings of the IEEE Conference on Computational Intelligence and Games, CIG 2012, pp. 402–408.

[40] T. Zahavy, N.B. Zrihem, S.Mannor, Graying the black box: Understanding DQNs, in: Proceedings of the 33rd International Conference on Machine Learning, vol. 48, arXiv:1602.02658, 2016.

[41] A. Zeller, Why Programs Fail: A Guide to Systematic Debugging, Elsevier, Burlington, Massachusetts, United States of America, 2009.