THE TOOLS CHALLENGE: RAPID TRIAL-AND-ERROR LEARNING IN PHYSICAL PROBLEM SOLVING

Kelsey R. Allen,*[1] Kevin A. Smith,* & Joshua B. Tenenbaum
{krallen, k2smith, jbt}@mit.edu
Department of Brain and Cognitive Sciences,
Center for Brains, Minds, and Machines (CBMM),
Computer Science and Artificial Intelligence Laboratory (CSAIL),
Massachusetts Institute of Technology
[1]Corresponding author *These authors contributed equally to the work

ABSTRACT

Many animals, and an increasing number of artificial agents, display sophisticated capabilities to perceive and manipulate objects. But human beings remain distinctive in their capacity for flexible, creative tool use – using objects in new ways to act on the world, achieve a goal, or solve a problem. Here we introduce the "Tools" game, a simple but challenging domain for studying this behavior in human and artificial agents. Players place objects in a dynamic scene to accomplish a goal that can only be achieved if those objects interact with other scene elements in appropriate ways: for instance, launching, blocking, supporting or tipping them. Only a few attempts are permitted, requiring rapid trial-and-error learning if a solution is not found at first. We propose a "Sample, Simulate, Update" (SSUP) framework for modeling how people solve these challenges, based on exploiting rich world knowledge to sample actions that would lead to successful outcomes, simulate candidate actions before trying them out, and update beliefs about which tools and actions are best in a rapid learning loop. SSUP captures human performance well across 20 levels of the Tools game, and fits significantly better than alternate accounts based on deep reinforcement learning or learning the simulator parameters online. We discuss how the Tools challenge might guide the development of better physical reasoning agents in AI, as well as better accounts of human physical reasoning and tool use.

1 Introduction

While trying to set up a tent on a camping trip, you realize that the ground is too hard for you to push the tent stakes in with your bare hands. If you had a hammer, you could pound them in – but you don't have one, so you search the campsite for another object to help you achieve your goal. Would you choose a pinecone? A stick? A water bottle? Or a rock? Probably you would first try to use the rock, to generate the force needed to get the stakes into the ground. And after trying that, if you failed to drive in the stakes, you might search for a more suitable (perhaps heavier) rock, or try a different grip or angle of impact.

Figuring out how to pound in tent stakes without a hammer might seem trivial at first, but it is an instance of the rich human capacity for creative tool use, and more generally, flexible physical problem solving. It requires a causal understanding of how the physics of the world works, and sophisticated abilities for inference and learning to construct plans that solve a problem we might never have faced before, and might not initially know how to solve. Consider how, when faced with the tent stake challenge, we do not choose an object at random; we choose a rock because we believe we know how we could use it to generate sufficient force of the right nature on the stake. And when we find that the first rock does not work, we again search around for a solution, but use the knowledge of our failures to guide our future

This manuscript is an extended version of a paper ("Rapid Trial-and-Error Learning in Physical Problem Solving") accepted for oral presentation at the 41st Annual Meeting of the Cognitive Science Society (2019). It represents ongoing work on the part of the authors. Further results and a link to the Tools game can be found at `https://k-r-allen.github.io/tool-games/`

search. This style of problem solving is a very structured sort of trial-and-error learning: our search has elements of randomness, but within a plausible solution space, such that a solution can often be found very quickly.

These abilities are not just human, but quintessentially human. While many animals have evolved to use specific rudimentary tools, only a few species of birds and primates show spontaneous, flexible, and creative tool use in the wild (Shumaker, Walkup, & Beck, 2011), and no other species approaches the flexibility and creativity of humans. These abilities also start early. By age four, children can figure out how to use an unusual object that is given to them as a tool to solve a non-obvious task (e.g., using a hooked straw to retrieve an item in a narrow container; Beck, Apperly, Chappell, Guthrie, & Cutting, 2011). More basic tool use and the ability to appreciate objects as tools emerges much earlier, between 18 and 24 months (Lockman, 2000). Evolutionarily, distinctive tool use and construction (in the form of handaxes) dates back to early hominids more than two million years ago, and has always been one of the defining dimensions of any human culture (Henrich, 2017).

Our goal here is to understand these human abilities in computational terms, with sufficient precision to both quantitatively explain people's flexible physical problem-solving behavior, as well as to ultimately implement analogous abilities in artificial intelligence (AI) systems. We begin (**Section 2**) by describing a new task environment for studying physical problem-solving that we call the "Tools" game, which presents a suite of challenges for both human and machine agents. We then (**Section 3**) propose a computational framework, called "Sample, Simulate, Update" (SSUP, pronounced "ess-sup"), for modeling how people might solve these challenges. The SSUP framework is based on two core ideas. First, people come to any new physical problem solving task equipped with rich knowledge about the world, in the form of a structured prior on candidate tools and actions likely to solve the problem and a mental simulator that allows them to imagine (albeit noisily and imperfectly) the likely effects of their actions. Second, people learn how to solve a new task by updating a belief distribution over high-value tool and action choices, after either simulating a candidate action or trying out that action in the real world and observing its effect; they then generate new candidate actions by sampling from this updated (posterior) belief distribution, or sampling more exploratory moves from their prior, and iterate until the problem is solved.

Our main results (**Section 4**) consist of benchmark human behavioral data on 20 Tools tasks, and quantitative comparisons with a model instantiating the SSUP framework, as well as multiple alternative models including a standard deep reinforcement learning baseline. We think of our SSUP models primarily as cognitive science contributions; they are intended to capture how people learn and think in physical problem solving, and represent only an initial foray into understanding that process. Yet the general framework they instantiate identifies key ingredients we believe will be needed in developing more human-like AI systems for physical reasoning, planning, and learning as we discuss further in **Sections 5 and 6**. In particular, our modeling suggests that the rapid trial-and-error learning people often display when using novel tools fits neither of the traditional modes of reinforcement learning (RL) in AI or cognitive science: it is not a purely model-free policy update, nor is it an instance of model learning (learning more about the dynamics of the environment); rather, it is a kind of simulation-based policy update, using simulated and real experience to update a distribution over actions to guide further sampling. Sections 5 and 6 discuss how the Tools game could motivate innovations in both AI and cognitive science approaches to physical problem solving.

2 The Tools game

We propose the Tools game as a platform for investigating the priors, representations, and planning and learning algorithms used by humans and machines in physical problem solving. Inspired by how people — and to a lesser extent, other animal species such as birds and apes — are able to use tools to solve complex physical problems (Shumaker et al., 2011), as well as mobile physics games such as Brain It On (*Brain it On*, 2015), we created a suite of problems which require various kinds of physical reasoning to solve.

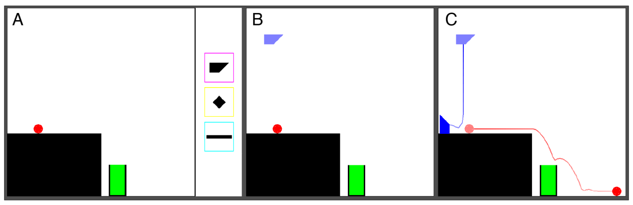


Figure 1: One illustrative human trial of the *Basic* level in the Tools game. (A) The player must get the red object into the green goal using one of the three tools on the right. (B) The player chooses a tool and where to place it. (C) Physics is then turned "on" and the tool interacts with other objects under forces of gravity, friction and collision. This particular action results in a near miss, as the red ball bounces off the rim of the cup.

The game asks players to place one of several objects ("tools") into a two-dimensional dynamic physical environment in order to achieve a goal: getting a red object into a green region, in a way that is stable for at least one second. This goal is the same for every level, but what is required to achieve it varies greatly (Fig. 1). As soon as a single tool is placed, the physics of the world is enabled

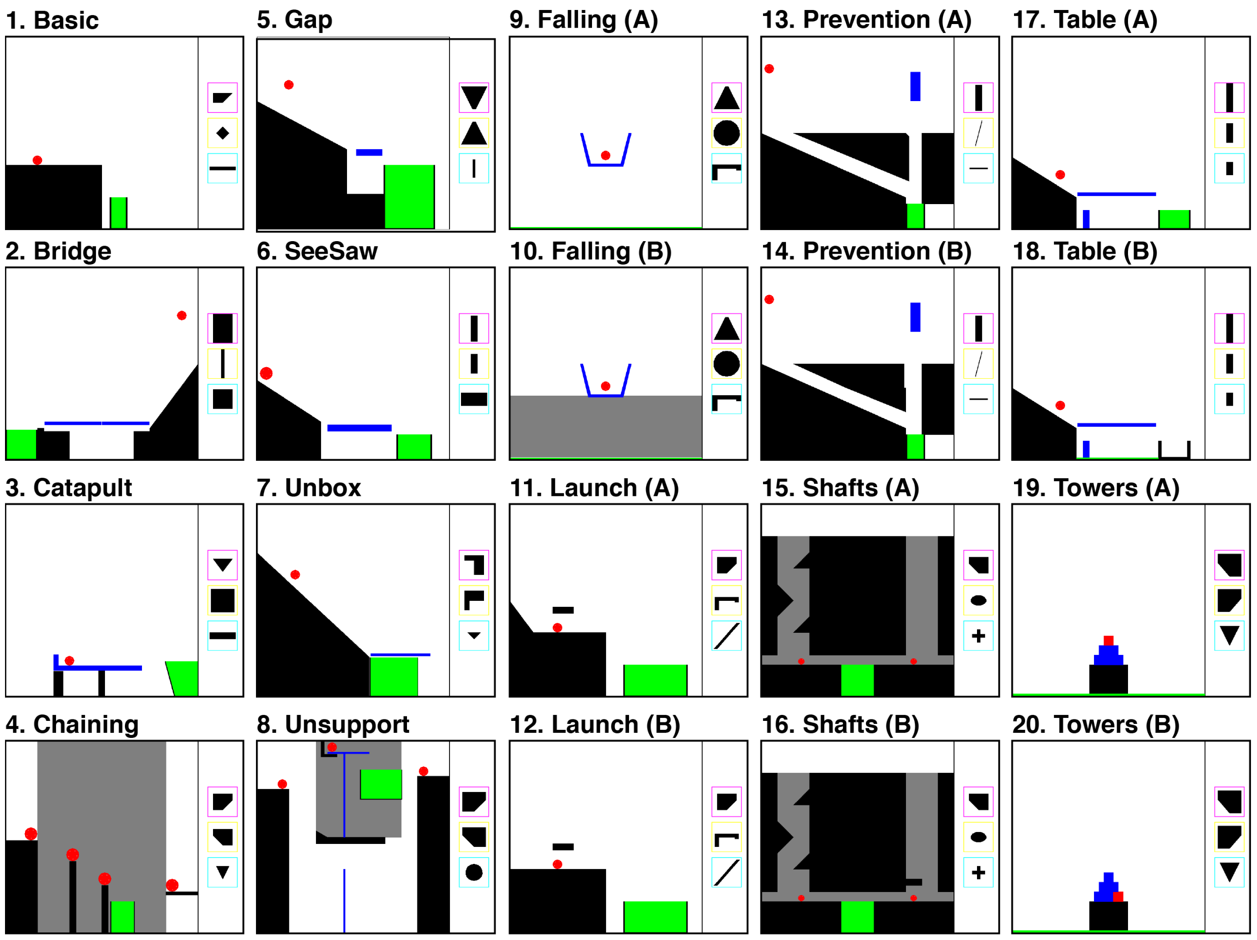


Figure 2: Twenty levels used in the Tools game. Players choose one of three tools (shown at right of each level) to place in the scene in order to get a red object into the green goal area. Black objects are fixed, while blue objects also move; grey regions are prohibited for tool placement. Levels denoted with A/B labels are matched pairs. See `https://k-r-allen.github.io/tool-games/` for a selection of example videos.

so that players see the effect of the action they took. If they fail, players can "reset" the world to its original state and try again; they are limited to a single action on each attempt. Although human tool use in the real world is more complex than the action space in this game – requiring sequences of actions, a choice of tools from among a much larger set, or even tool creation – having this more limited action space enables easier comparisons across participants, and seems to already be a challenging problem for AI approaches.

The Tools game presents particular challenges that we believe underlie the kinds of reasoning required for rapid physical problem solving more generally. First, there is a **diversity of levels** that require different strategies and physical concepts to solve, but employ shared physical dynamics that approximate the real world. Second, the Tools game requires **causal reasoning**. Most of the levels can be solved by considering how to intervene on a scene in order to achieve a goal, such as creating a "support" for an object that you realize is falling, or "preventing" an object from blocking another. Third, the Tools game requires **long horizon predictions**. Since players are only able to interact with the game on the very first time-step, they need to be able to predict the effects of their actions long into the future when they can no longer intervene. Finally, the Tools game prompts **few-shot trial-and-error learning** in humans. Human players do not generally solve the level on their first attempt, but also generally do not require more than 5-10 attempts in order to succeed. Few-shot learning is a frontier for further progress in machine learning, and the Tools game is a measure of this capacity in the domains of physical reasoning and action.

Two further difficulties in the Tools game must be addressed by both humans and machines. First, there is a challenging search problem: in any situation there are a large number of actions one *could* take, but few of them will be relevant to the problem at hand, and most of those that are relevant will not solve the task. Second, any (human or machine) agent's model of the physics for this novel environment is bound to be imperfect and uncertain, so planning must be robust to those sources of uncertainty, and revisable in adaptive ways when the agent sees that their initial plan was not successful (Battaglia, Hamrick, & Tenenbaum, 2013; Smith & Vul, 2013; Sanborn, Mansinghka, & Griffiths, 2013).

2.1 Levels

We constructed 20 levels to investigate a range of physical principles such as "launching" or "catapulting" a ball, "supporting" a table for an object to roll across, or "preventing" an object from blocking the goal (see Fig. 2). Of these 20 levels, 12 were constructed in 6 pairs, with small variations in the goals or objects in the scene so that we could test whether subtle differences in stimuli would lead to observable differences in behavior. Some levels have dynamic objects (in blue), as well as grey regions in which objects cannot be placed. While the levels were constructed to test particular physical concepts (and named appropriately), participants in our experiment were not aware of these categories or names, and therefore were expected to discover these principles for themselves.

Direct launching We classify direct launching levels as those that require a participant to use a tool in order to directly hit a red object to get it into the goal. This covers levels *Basic*, *Launch A & B*, *Shafts A & B*, and *Towers A*.

Indirect launching Indirect launching levels are those that require players to interact with some dynamic object in the scene which will then launch a goal object. These include *Towers B*, *Chaining*, and *Catapult*.

Destroying structure *Unbox*, *Unsupport*, and *Table B* all require destroying the structure of the level in some way. For Unbox and Unsupport, some dynamic object needs to be removed in order to accomplish the goal, while in Table B, extra space needs to be made so the ball can touch the ground.

Maintaining initial structure Several levels (*Bridge*, *Table A*, *SeeSaw*, *Prevention A & B*, and *Gap*) require players to find some way of maintaining the initial structure of the scene after gravity is turned on. This involves putting a tool object in a supporting position for one (or two) of the dynamic objects in each level, to allow the key object to travel over a surface (e.g., Bridge) or prevent an object from obstructing motion (e.g., Prevention).

Tipping/angular motion To examine how rotation affects the plans players consider, *Falling A & B*, *Table B*, *Catapult*, and *Launch B* were designed such that their solutions required tipping an object over, or causing some object to acquire angular momentum.

In each case of a matched pair, the scene is almost identical with only a minor modification in the 'B' version which causes one strategy to no longer be viable. For example, in Shafts B, a small static platform is introduced over the easily accessed ball, such that this option is no longer feasible. In Launch B, the ramp to the left is removed, such that the easier strategy is eliminated, and so rotating an object to launch the ball is required. In Table B, the goal is changed from being in the container to being on the floor, which necessitates "unhinging" the table object to make space for the ball to reach the floor. These pairs ensure that any player learning to play the game will need to generalize and adapt to the specific details of each level. A rote strategy, or nonparametric approach for an 'A' level, will fail on 'B' without updating.

2.2 Rapid trial-and-error learning

The Tools game was designed to expose a particular facet of human problem solving: initial structured search, then exploitation of promising solutions. At the start of each level, people are often unsure of what sorts of strategies or actions will solve the level, and therefore begin by taking exploratory actions that may appear useful, but will not achieve the goal. However, when they take an action that does not solve the level but is similar to an action that will, they will often notice that this is a promising strategy and exploit this information, fine-tuning their actions to produce a solution. Figure 3 demonstrates how this occurs in practice, showing five different examples of participants learning rapidly or slowly, and discovering different ways to use the tools across a variety of levels. We are interested in exploring how people perform this rapid trial-and-error learning, and what this implies for artificial agents that are expected to interact with the world in a human-like fashion.

3 The Sample, Simulate, Update framework

In order to capture the richness of human trial-and-error learning on the Tools game, including both local, incremental search, as well as "a-ha" insights for a new strategy, we introduce the "Sample, Simulate, Update" framework (SSUP; Fig. 4A). This framework relies on three components: a method for quickly initializing search to a set of promising action candidates ("Sample"), an internal simulator that lets the agent form very general (if imperfect) predictions about the effects of their actions on the world ("Simulate;" e.g., Battaglia et al., 2013), and a learning mechanism that can update the agent's beliefs about what they think will work based on the outcomes of both imagined and experienced actions ("Update").

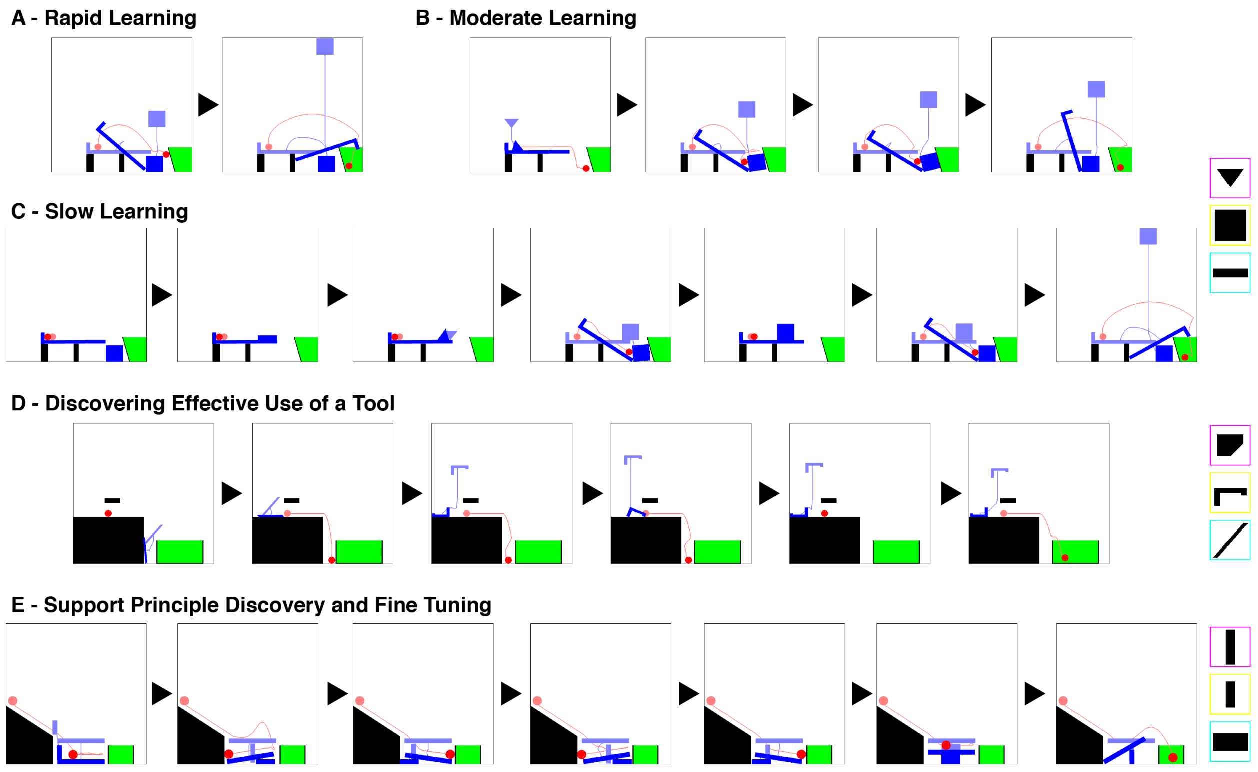

A - Rapid Learning
B - Moderate Learning
C - Slow Learning
D - Discovering Effective Use of a Tool
E - Support Principle Discovery and Fine Tuning

Figure 3: Example of five people's actions on three different levels. Objects start in the light outlines, and end in the position with solid figures, following the path traced out by each line. Possible tool choices shown to the right. **(A)** A person who identified a useful strategy initially and rapidly learned how to solve the level. **(B)** A person who began with a strategy that would not work, but quickly converged on actions that were close to a solution. **(C)** A person who tried multiple, unsuccessful actions before discovering that using the large block as a lever is useful. **(D)** A person who may have thought the ball would have some initial velocity, and then immediately figured out they needed to hit the ball. They discovered how to use a tool in a surprising way (by having it rotate around the static black platform) in order to accomplish this, and then fine-tuned this action to hit the ball in just the right way. **(E)** A person who learned in a single trial that they needed to support the platform for the ball to roll across, but then had to try multiple different ways of making this happen. Indeed, they tried all three different tools for this task, eventually finding that the smaller pillar was successful. These people's behavior is emblematic of "rapid trial-and-error" learning, initially searching around objects, and then identifying and exploiting actions identified as promising.

While a simulator is required to assess whether candidate actions will accomplish our goals before we act, just having a simulator is not enough if we consider agents with limited computational resources. In general, even having a perfect simulator still presents a *search* problem, since there is a large action space to consider. To overcome this, SSUP includes structured priors and a mechanism for updating our beliefs about which actions are likely to be successful (e.g., Forbus, Gentner, & Law, 1995). When combined with a noisy simulator, this allows SSUP to simulate fewer yet more promising potential actions, and requires fewer (failed) interactions with the world.

This framework is inspired by the theory of "problem solving as search" (Newell & Simon, 1972), as well as Dyna and other model-based policy optimization methods in planning and reinforcement learning (Sutton, 1991; Deisenroth, Neumann, Peters, et al., 2013). Crucially, we posit that structured priors and physical simulators must already be in place for the learner in order to solve problems as rapidly as people. Unlike most model-based policy optimization methods, we do not perform online updates of the dynamics model. We do compare with an alternative model where learning comes from tuning the dynamics model in the ablation studies below.

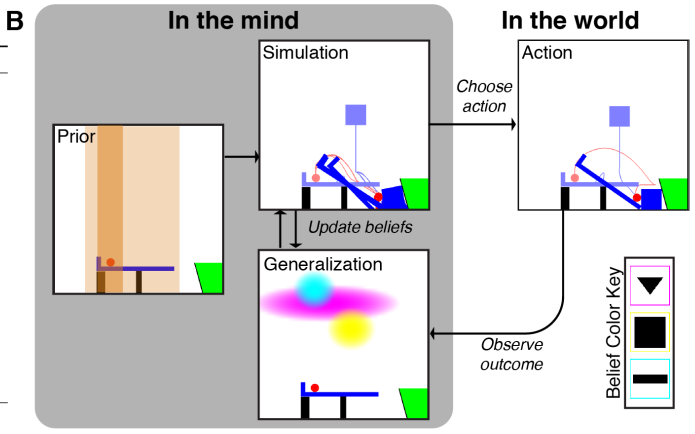
3.1 A model for the Tools game

In our Tools game, SSUP includes an object-oriented prior ("Sample"), a noisy physics engine ("Simulate"), and a stochastic policy gradient update procedure ("Update"; Fig. 4B). Many other choices for each of these components are possible, but we believe these key ingredients are necessary to capture the richness of human trial-and-error learning. For further details on implementation, see Section S2.

A

Algorithm 1 SSUP algorithm

```
Sample actions from prior $a \sim \pi(s)$
Simulate action to get noisy rewards $\hat{r} \sim model(s, a)$
Initialize distribution $\pi'(s)$ using samples $\hat{r}, a$
while not successful do
    Sample action $a$
    Simulate action to estimate noisy reward $\hat{r} \sim model(s, a)$
    if condition for action holds then
        Try action $a$ in environment
        Observe $r$
        If successful, exit
        Update policy with $r, s, a$.
    else
        Update policy with $\hat{r}, s, a$
    end if
end while
```



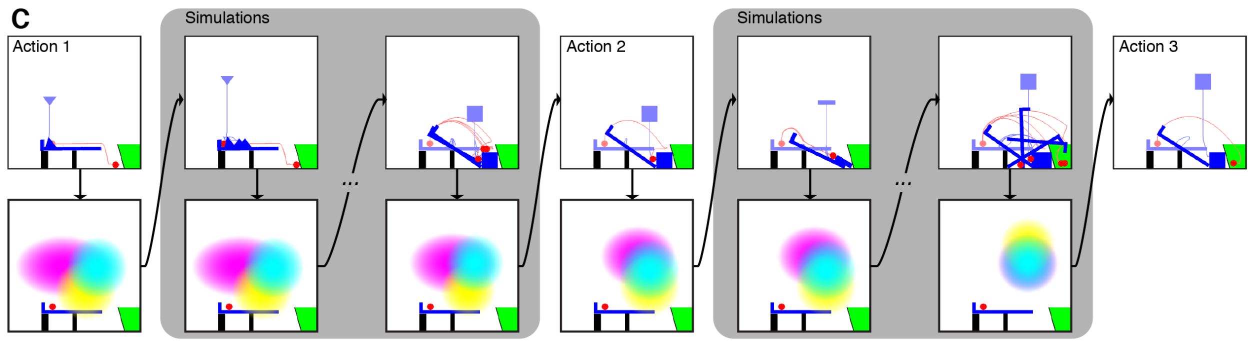


Figure 4: **(A)** The SSUP framework algorithm. **(B)** A diagram of the model instantiating the SSUP framework for the Tools game. This SSUP model incorporates an object-based prior, a simulation engine for filtering proposals, and an update module that suggests new proposals based on observations "in the mind" and from actions taken in the world. **(C)** Illustration of this model performing a level. Initially, the model chooses an action that will not succeed. It then considers other possible actions, and finds that dropping a heavy block on the right side might work, so it tries that as its next action. While this does not quite achieve the goal, the model considers similar actions, and ends up dropping the block from higher to win the level.

3.1.1 Sample: object-based prior

In line with other work on physical problem solving showing that object-oriented and relational priors are important (Hamrick et al., 2018), we incorporate an object-based prior for sampling actions. Since all tools in the game were designed to be unfamiliar to participants, we place a uniform prior over the three tool choices. We do assume, however, that participants will have inductive biases on *where* they place the tools: they should select actions that are likely to interact with movable objects, thus having an effect on their trajectory. The model implements this prior by first selecting one of the movable objects in the scene. It then chooses an x-coordinate in an area that extends slightly beyond the width of the object, and a y-coordinate either above or below that object (Fig. 4B: Prior).

3.1.2 Simulate: a noisy physics engine

We assume people use an "Intuitive Physics Engine" (Battaglia et al., 2013) to filter proposed actions as being potentially feasible or not. This engine is able to simulate the world forwards in time with approximately correct but stochastic dynamics (Smith & Vul, 2013; Smith, Battaglia, & Vul, 2018). Determining the effect of a proposed action therefore involves applying that action to one's mental representation, and using the Intuitive Physics Engine to posit the range of ways that action might cause the world to unfold (Craik, 1943; Dasgupta, Smith, Schulz, Tenenbaum, & Gershman, 2018).

We therefore implement simulation using a game physics engine with noisy dynamics. As found in prior work, humans characteristically have noisy predictions of how collisions will resolve (Smith & Vul, 2013), and so we assume uncertainty over the effects of actions is driven by uncertainty only in those collisions (the direction and amount of force that is applied between two colliding objects).[1]

In order to decide if a proposed action is worth attempting in the game, the model produces a small number of stochastic simulations of this action (n_{sims}, set here at 4) to form a set of hypotheses about the outcome of that action. For each hypothesis, the minimum distance between the goal area and one of the objects that must get into that area is recorded; these values are averaged across the simulations. We normalize each distance by the minimum distance that would have been achieved if no action had been taken, to better account for how much of a *difference* an action made to the outcome (Gerstenberg, Goodman, Lagnado, & Tenenbaum, 2015). Since low values are indicative of actions that almost achieve the goal, if the average is below a threshold, the model takes that action "in the world." If the model considers more than T different action proposals without acting (set here at 5), it takes the best action it has imagined so far. We evaluate the effect of these particular choices in a sensitivity analysis (see Figure S2).

3.1.3 Update: learning from thoughts and actions

We parameteterize beliefs over likely successful actions $\pi'(s)$ as a mixture of three Gaussians: one distribution each for all three tools, with a weighting across distributions that captures the relative probability of selecting each tool.[2] $\pi'(s)$ is updated using a simple policy gradient algorithm (Williams, 1992) that calculates reward based on the minimum distance to goal metric calculated from both internal simulations and observations of action outcomes (similar to the intrinsic rewards sugested by Juechems & Summerfield, 2019). This algorithm will shape the posterior beliefs around areas to place each tool which are expected to move target objects close to the goal, and therefore is likely to contain a solution. In order to allow for structured exploration, the algorithm samples actions according to an ε-greedy strategy, where the exploratory actions are drawn from the prior.

Because the Gaussians for each tool are modeled independently, the model additionally generalizes across tools by considering whether alternate tools might succeed when an action has failed. This is instantiated by simulating the expected reward if the other tools had been placed in the same location as the performed action. In this way, if the model takes an action in a good location with the wrong tool, it can transfer knowledge about its placement across the object types, without generically assuming that all tools will be equally good for all positions.

3.2 Alternate models: ablations and machine baselines

We propose that "trial-and-error" problem solving requires (1) an object-based prior, (2) a simulation engine, and (3) policy updates. We therefore considered ablations of the model that lack these various pieces to determine their relative contributions, and find that removing any of these three modules negatively affects the model's performance, and its ability to explain human behavior.

We additionally compare the model to two alternate learning approaches. One alternate model, based on deep reinforcement learning, allows us to test whether good action policies can be learned from repeated encounters with similar trials, but without structured priors or simulators. This is analogous to attempting to learn the "Sample" step of the SSUP framework from substantial experience. The other alternate model supposes that people are learning more about the properties of objects in the scene after each observation (e.g., tuning the parameters of their intuitive physics engine), but are not learning about good or bad action choices in a particular scenario.

Sampling + Simulator (no updating) To test whether updating policies impacts participants' performance on this task, we consider a model that samples proposals from the prior and evaluates them with the simulation module, but does not update its policy to guide search. Instead, it simply samples options from the prior until either (a) the thinking time is exceeded, and it takes a random action, or (b) the model takes the proposed action using the same decision criterion as in the full model.

Sampling + Updating (no simulation engine) We next consider the contribution of having predictive world models by using an ablation that does not use simulation to evaluate proposals, relying only on the object-based prior for initialization of the policy. This model therefore can only take external actions, except for a set of points used for initializing the policy, to allow a fair comparison to the full model. This can be thought of as model-free reinforcement learning, but with an inductive bias towards object-oriented actions (since the exploration policy and initialization are object-oriented).

[1]We also considered models with additional sources of physics model uncertainty added, but found that the additional parameters did not improve model fit, so we do not analyze those models here.

[2]It is possible that human posterior beliefs are multi-modal or non-parametric, but we found that using a simple unimodal distribution worked well for these levels.

Simulator + Updating (no object-based prior) Our final reduced model does not use the object-based prior to form initial proposals, but instead samples uniformly from all legal actions; all other aspects of this model are identical to the full model. This ablation can be thought of as similar to the Dyna architecture proposed by Sutton (1990), in which an agent uses its internal model of the world to determine good value estimates for a model-free policy when taking external actions. The main difference is that Dyna assumes that the internal model should be learned while performing the task, whereas we assume people have already learned a model of physical dynamics before engaging with our experiment (or within the three practice levels they encounter).

Deep reinforcement learning + Updating As an alternate learning scheme, we compare to a model from deep reinforcement learning, Deep Q Networks (DQN; Mnih et al., 2015), that attempts to learn and generalize a good policy for taking actions based on training on a large number of variants of five of the levels. We imagine this as analogous to attempting to learn a good prior in the SSUP framework. Thus this model could be comparable to the "Prior + Updating" ablation, but using a learned prior instead of an object-based one. Although this policy is learned offline before attempting each of the main trials, it has the capability of learning online in the process of solving each trial via a policy gradient. See Sections 5.2 and S3 for further details.

Physical parameter tuning Another alternate scheme is to eschew learning a link between actions and outcomes, and instead use learning to improve one's internal world models, and use that updated model to re-plan on future attempts. This approach is often used when applying model predictive control (Feldbaum, 1960; C. Xie, Patil, Moldovan, Levine, & Abbeel, 2016; Fu, Levine, & Abbeel, 2016; Janner et al., 2018; Deisenroth & Rasmussen, 2011), also often called system identification (Chiuso & Pillonetto, 2019). Physical parameter tuning learns better estimates of object and world properties (such as gravity or density) from action observations, using Bayesian inference to match simulated outcomes to observations (similar to Hamrick, Battaglia, Griffiths, & Tenenbaum, 2016; Yildirim, Smith, Belledonne, Wu, & Tenenbaum, 2018). It is therefore very similar to the "Sampling + Simulator" ablation, but with a tuned simulator instead of a fixed one. See Section S4 for further details. While this could in principle be combined with policy learning, we wanted to test whether physical parameter learning alone could capture the learning curves we see in participants.

Guessing We finally consider a model that randomly selects valid actions. In this case, the probability of success on a given level is the fraction of placements that are successful relative to the total number of valid positions. While we do not believe that people are randomly responding, this serves as a baseline against which to compare other models' improvement.

4 Results

In this section, we analyze human performance on the Tools game and compare humans to our SSUP model and its ablations, as well as to the two alternate learning baselines. We show that humans display rapid trial-and-error learning across all levels, and that only the full SSUP model captures human performance.

4.1 Human results

We recruited 94 participants through Amazon Mechanical Turk and asked each participant to solve 14 levels: all 8 of the unmatched levels, and one variation of each of the 6 matched pairs (randomly selected). The level order was randomized across participants. Participants were given two minutes to solve each problem, and could move on immediately if they solved it. We recorded all attempted actions by participants, including which tool was used, where it was placed, and the time between placements. See Section S1 for further details.

The variation in difficulty between levels of the game was substantial. Participants showed an average solution rate of 81% (sd = 19%), with the range covering 31% for the hardest level to 100% for the easiest. Similarly, participants took an average of 4.5 actions (sd = 2.5) for each level, with a range from 1.5 to 9.4 average attempts. Even within trials, there was a large amount of heterogeneity in the number of actions participants used to solve the level. This would be expected with "rapid trial-and-error" learning: participants who initially tried a promising action would solve the puzzle quickly, while others will explore around trying different actions before happening on promising ones (e.g., Fig. 3).

Behavior differed across all six matched level pairs, with participants requiring a different number of actions to solve them (all $ts > 2.7$, $ps < 0.01$), and often producing different patterns of actions, even from the first trial (see Fig. S3). This suggests that people are paying attention to subtle differences in the scene or goal to choose their actions, not using gross associative strategies.

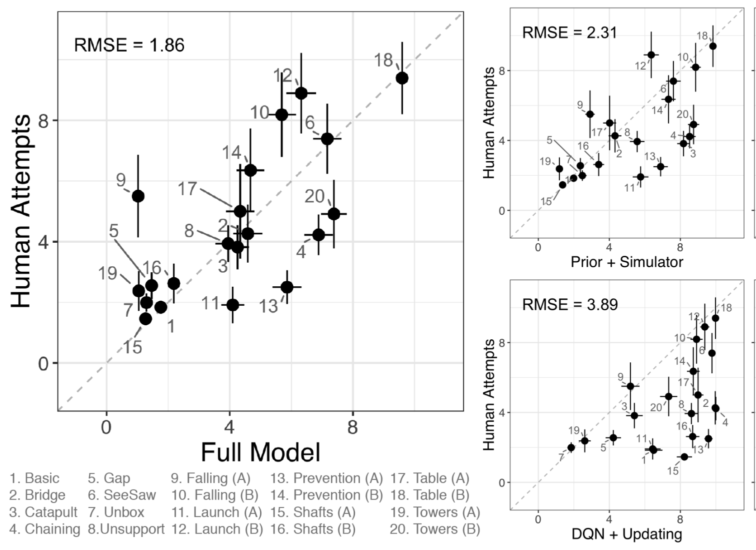


Figure 5: Comparison of average number of human participants' attempts for each level with average number of attempts for the full model (*left*) and six alternate models (see Section 3.2 for descriptions). Numbers correspond to the trials in Fig. 2. Bars indicate 95% confidence intervals on estimates of the means. The number of placements was capped at 10 for all models. If a model took more than 10 attempts on a particular level, it is considered unsolved. Model results are combined over 250 runs.

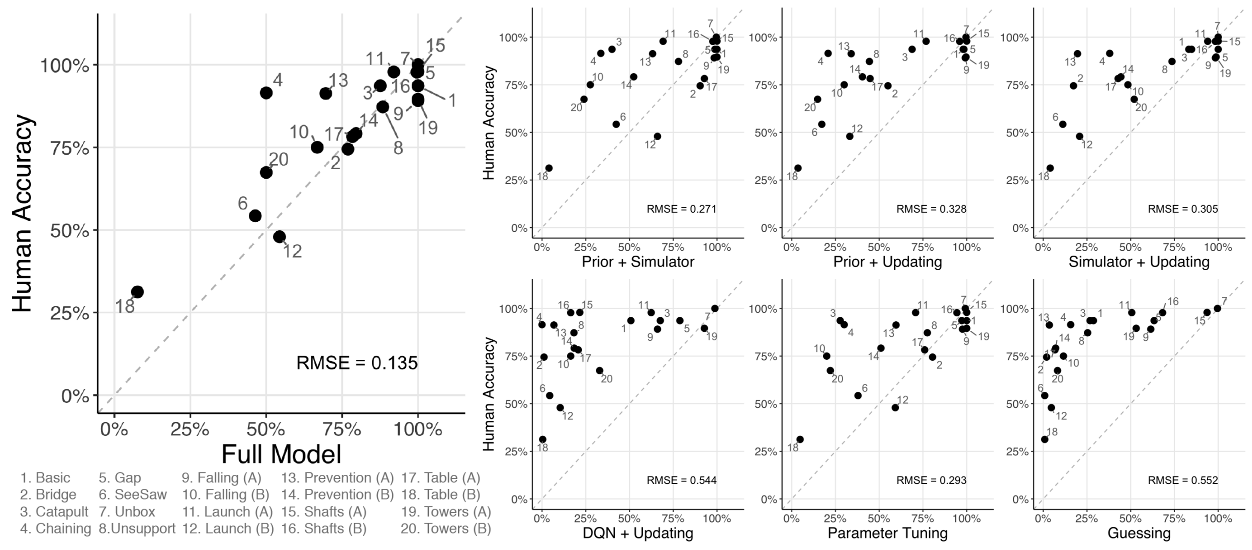


Figure 6: Comparison of human participants' accuracy on each trial versus the accuracy of all models. Numbers correspond to the trials in Fig. 2. DQN+Updating and Guessing perform badly across most levels.

While participants improved in solution rate over the course of the experiment (76% solution rate on the first three trials to 86% on the last three; $\chi^2(1) = 9.7$, $p = 0.002$), they did not solve the levels more efficiently ($\chi^2(1) = 2.0$, $p = 0.15$), taking an equal number of attempts to arrive at a solution across the experiment.

4.2 Model results

We investigate several metrics for comparing the models to human data. First, we look at how quickly and how often each model solves each level, and whether that matches participants. This is measured as the root mean squared error (RMSE) between the average number of participant attempts for each level and the average number of model attempts for each level (see Fig. 5, Table 1). We also compare a model's solution rate to the human solution rate across all trials in the same way (see Figs. 6 and Table 1).

We further define a metric to quantify the difference between empirical and model cumulative solution curves (as shown in Fig. 8) called *Area Between Cumulative Solutions* (ABCS). This metric captures the evolution of behavior over time:

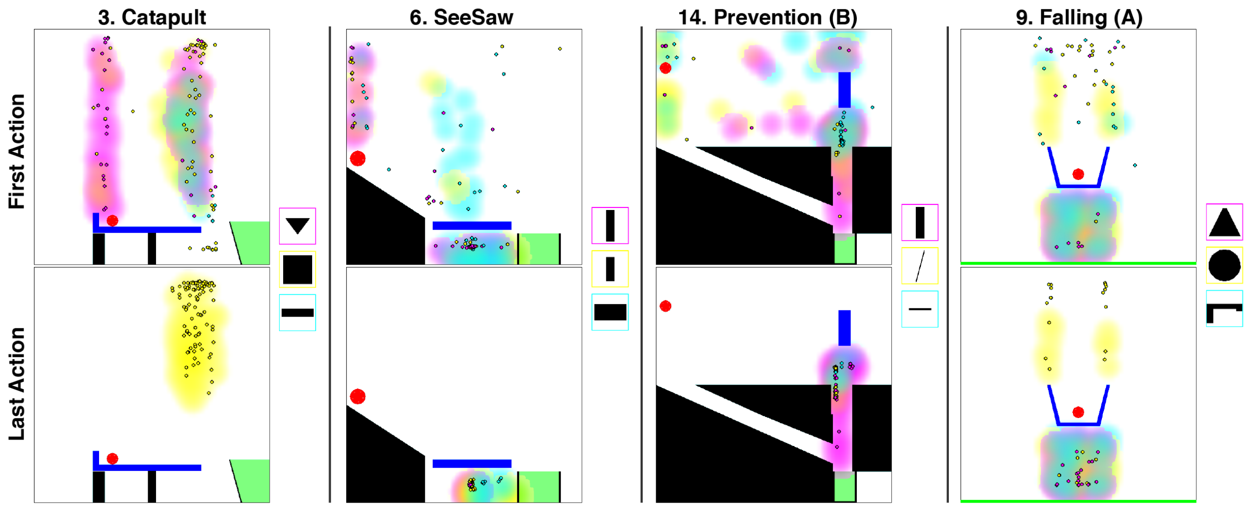


Figure 7: Distribution of predicted model actions (background) versus human actions (points) on the first attempts of the level (*top*) and the attempt used to solve the level (*bottom*) for a selection of four levels. Colors indicate the tool used, with the tools and associated colors shown to the right of each level. Across many levels that include using concepts such as launching, catapulting, and blocking, the model captures the diversity of where people initialize their search, as well as the types of solutions they eventually find. However, there are certain levels where people appear to have different prior beliefs than the model (Falling (A); *right*), which lead to different solution patterns.

over all participants and model runs, what proportion solved each level within X placements. If these curves are similar between participants and a model, it suggests that the model's beliefs about good action proposals are evolving at a similar rate to people. We can then calculate the total area between empirical and model cumulative solution curves, normalized between 0 (perfect match) to 1 (participants or the model always solve the level instantaneously, while the other never solves the level).

ABCS has the benefit of capturing information about differences in both the accuracy and and number of placements used, as well changes in the rate of solutions due to rapid trial-and-error learning. We can therefore compare models by summing ABCS across all 20 trials (Total ABCS), such that models that capture human performance better will have lower values (see Fig. 8 and Table S1 for ABCS values by trial).

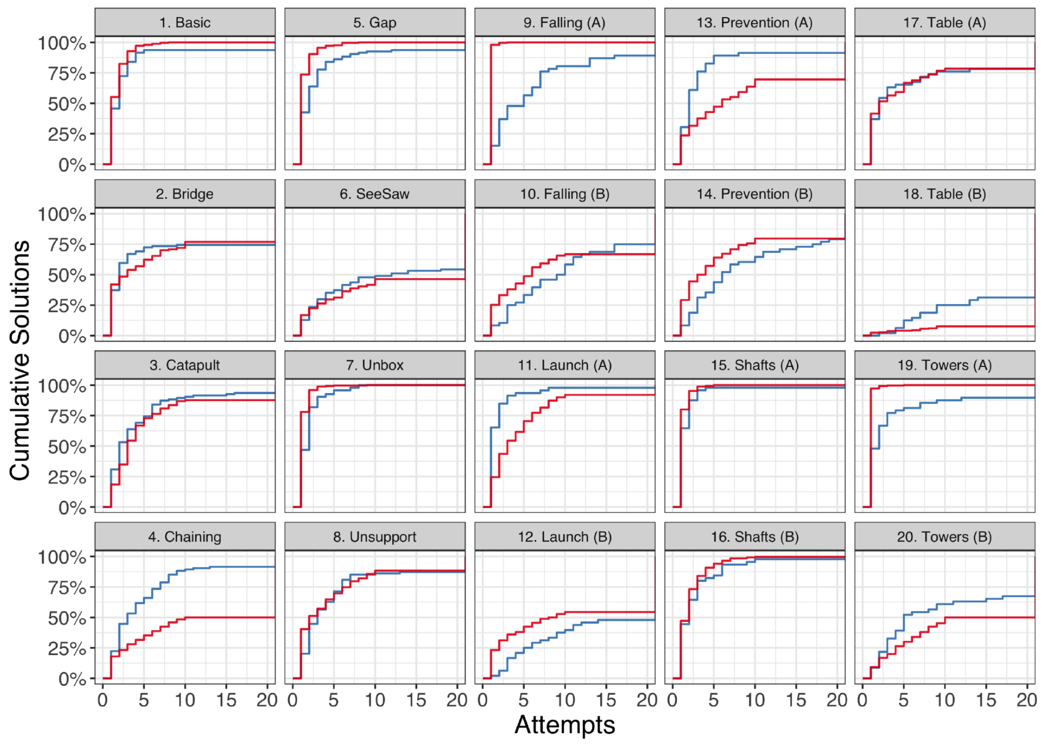
4.2.1 Model comparisons on the Tools game

The full model explains the patterns of human behavior across the different levels well. It uses a similar number of attempts on each level as people do ($r = 0.71$; 95% CI $= [0.62, 0.76]$; mean empirical attempts across all levels: 4.48, mean model attempts: 4.24; see Fig. 5, left). It also achieves similar accuracy to participants ($r = 0.86$; 95% CI $= [0.76, 0.89]$; see Fig. 6).

Across many levels, the SSUP model not only achieves the same overall solution rate as people, but approaches it at the same rate (Fig. 8, top). We can look qualitatively in more detail how the full model accomplishes this by comparing both the first actions that people and the model takes (Fig. 7, top), and the actions that both take to solve a level (Fig. 7,

Model	Avg. Attempts	Attempt RMSE	Accuracy	Accuracy RMSE	Total ABCS
Human	4.48 [4.25, 4.66]	-	0.81	-	-
Full Model	4.24 [4.17, 4.32]	1.86 [1.66, 2.17]	0.77 [0.76, 0.78]	0.135 [0.121, 0.169]	2.21
Prior + Simulator	5.38 [5.32, 5.46]	2.31 [2.16, 2.59]	0.69 [0.68, 0.7]	0.271 [0.248, 0.299]	3.78
Prior + Updating	5.23 [5.14, 5.31]	2.29 [2.12, 2.58]	0.59 [0.58, 0.6]	0.328 [0.307, 0.353]	4.37
Simulator + Updating	5.55 [5.48, 5.62]	2.45 [2.29, 2.71]	0.61 [0.6, 0.62]	0.305 [0.288, 0.33]	4.05
DQN + Updating	7.52 [7.44, 7.61]	3.89 [3.76, 4.1]	0.34 [0.33, 0.35]	0.544 [0.527, 0.566]	7.95
Parameter Tuning	5.7 [5.64, 5.78]	2.39 [2.25, 2.65]	0.65 [0.64, 0.66]	0.293 [0.275, 0.323]	3.88
Guessing	8.88	4.91 [4.75, 5.1]	0.32	0.552 [0.531, 0.573]	8.04

Table 1: Comparisons with alternate models. Brackets indicate bootstrapped 95% confidence intervals on the estimate. 'Total ABCS' refers to the sum of the *Area Between Cumulative Solution* curves of participants and the model (see Fig. 8). 'Guessing' model placements and accuracy can be calculated exactly and therefore have no confidence intervals.



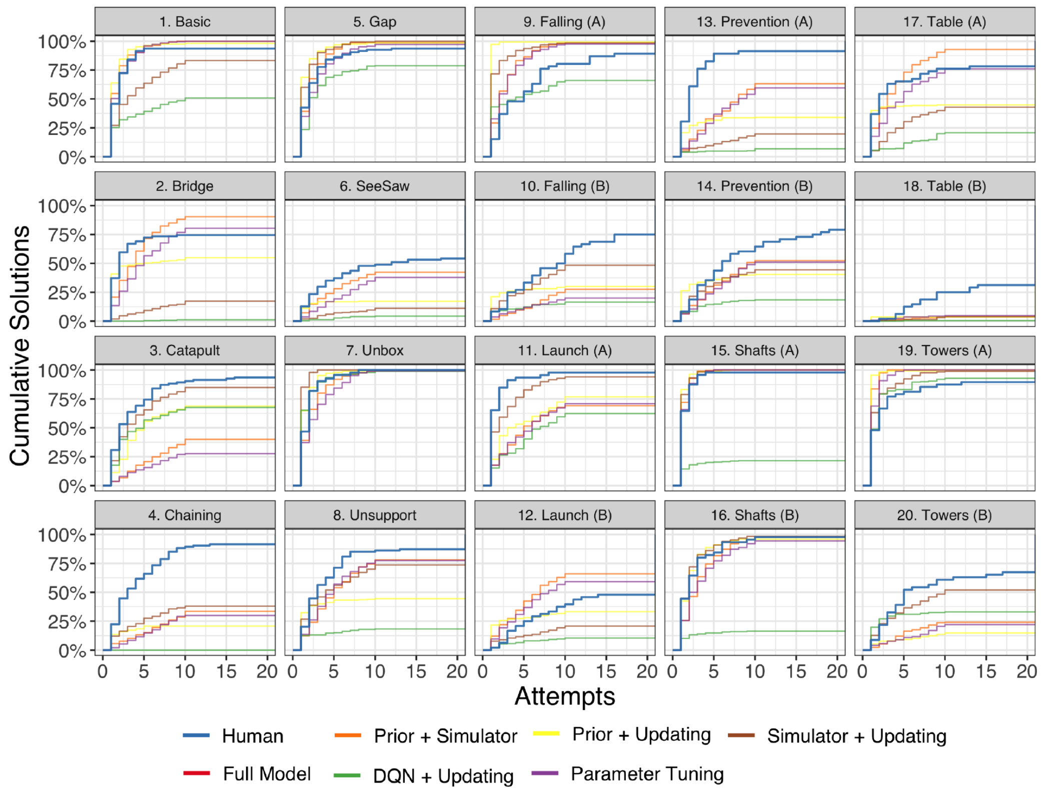


Figure 8: Cumulative solution rate over number of placements for participants vs. the full model *top* and participants vs. all alternative models *bottom*. In most levels, the full model captures the evolution of participants' solutions well; in the few cases that it differs, there is no alternative model that systematically explains these differences. In general, the deep reinforcement learning baseline (DQN+Updating) is insufficient for solving tasks in this number of attempts, even with online updating.

bottom). For example, both people and the full model will often begin with a variety of plausible actions (e.g., Catapult). In some cases, both will attempt initial actions that have very little impact on the scene (e.g., SeeSaw and Prevention (B)); this could be because people cannot think of any useful actions and so decide to try *something*, similar to how the model can exceed its simulation threshold. However, in other cases, the model's initial predictions diverge from people, and this leads to a different pattern of search and solutions. For instance, in Falling (A), the model quickly finds that placing an object under the container will reliably tip the ball onto the ground, but people are biased to drop an object from above. Because of this, the model often rapidly solves the level with an object below, whereas a proportion of participants find a way to flip the container from above; this discrepancy can also be seen in the comparison of number of attempts before the solution, where the model finds a solution quickly, while people take a good deal longer (Fig. 5, left). For comparisons of the first and last actions across all levels, see Fig. S3.

Eliminating any of the three SSUP components causes a significant decrease in performance (see Table 1 and Fig. 5), with the ablated models typically requiring more attempts to solve levels because they are either searching in the wrong area of the action space (no prior), attempting actions that have no chance of being successful (no simulator), or do not guide search towards more promising areas (no updating).

DQN + Updating performs worst of all plausible alternate models, using the most actions and solving levels at a rate barely over chance. Because this is equivalent to the Prior + Updating model with a different prior, its poor performance suggests that generalized action policies cannot easily be learned from repeatedly playing similar levels (see Sec. 5.2 for further discussion of this model's learning).

Because the Parameter Tuning model is equivalent to the Prior + Simulator except that the properties of the dynamics model can be learned in Parameter Tuning, comparing those two models allows us to test whether we need to assume that people are learning the dynamics of the world in this Tools game. The fact that both models perform roughly equivalently (see Table 1) suggests that we do not need to make this assumption. If we consider that human performance throughout the experiment showed only marginal improvements (see Section 4.1), this suggests that most learning in this task is not building better models of the game, but rather how actions lead to success within specific levels.

5 The Tools game as an AI challenge

In this section, we propose the Tools game as a challenge domain for building and measuring human-level learning and planning capacities in AI. The game presents several conceptual challenges that we believe are at the root of more general rapid problem solving, and that should push the boundaries on sample efficiency and generalization of current methods. These challenges (discussed in depth in Section 2) are not tested in existing benchmarks such as the Atari environments (Bellemare, Naddaf, Veness, & Bowling, 2013) or the standard Mujoco benchmarks (Todorov, Erez, & Tassa, 2012; Brockman et al., 2016). For instance, this game has a diverse set of levels that share a transition function with each other and (roughly) with the real world, so that strong generalization should be possible if an agent has a good model of dynamics in the world. In addition, to achieve human-level performance requires finding a solution within just a handful of attempts with feedback, suggesting a need for few-shot learning.

We envision the Tools game as providing two distinct challenge settings for the AI learning and planning communities. The first setting would use the game strictly as a set of test tasks: Agents should build general physics models and problem-solving strategies before they come to these tasks, from real-world experience or perhaps from playing other physics games, like our human players do. Our game tasks then present novel problems to be solved, probing the flexibility and creativity of what agents have learned, along with their ability to adapt quickly to these new tasks with the rapid forms of trial-and-error learning we observe in humans. The second setting would allow agents to learn within our tools domain, but require them to generalize to new tasks (new levels) within it. While we believe that the former setting is ultimately the more important and more interesting one for developing human-level AI capabilities, we recognize that the latter is more achievable in the short term, especially for current learning-driven approaches. For instance, the latter setting could be seen as a new domain for few-shot meta-learning with rich object-centric physical dynamics.

To support research in these directions, we will release the python environment for our game at `https://k-r-allen.github.io/tool-games/`, as well as the level descriptions for the 20 levels that we have tested humans on so far. For training agents within the tools domain (i.e., the latter, meta-learning setting proposed above), we will provide a set of parameterized background levels as a set of level generators, like the ones presented in Section S3.2. Performing well on all 20 test levels, even with substantial training on background levels, will require strong generalization since the strategies needed on a smaller set of background levels will not fully encompass those needed for the test games. We also intend to release more complex versions of the game (see future extensions) that can be easily developed in the same python environment.

Below we discuss related work from AI, robotics and deep reinforcement learning, then present our implementation of one baseline deep reinforcement learning agent for the tools game, and suggest how the game might be extended to provide further challenges for AI. Based on our preliminary analyses outlined in section 5.2, we expect that solving the Tools game (and its extensions) is beyond the capabilities of current deep reinforcement learning baselines, and will push future model-based and model-free methods towards learning more flexible, generalizable physical reasoning.[3]

5.1 Related work in AI

Deep model-free reinforcement learning has traditionally focused on model-free learning, where the input to the model is an observation (often an image), and the output is an action. Such models are optimized “end-to-end” and learn non-linear representations of policies parameterized as deep neural networks. Representing policies in this way implicitly combines perception, planning, modelling and acting into a single deep network, which lacks the modularity that we expect is necessary for strong generalization across different tasks. Nevertheless, this community has found that model-free approaches can achieve very good performance on a wide range of physical and non physical tasks (Mnih et al., 2015; Levine, Finn, Darrell, & Abbeel, 2016; Lillicrap et al., 2015), at least when the training and test tasks are drawn from very similar distributions. We imagine that the Tools game, in which there is substantial variation between tasks and where humans demonstrate rapid learning in just a handful of attempts, will test the limits of these approaches. In our experiments, we find that current methods fail to solve our tasks, and hope that our challenge could help further progress on more flexible, sample efficient deep reinforcement learning techniques.

Deep model-based reinforcement learning has made major progress on sample-efficient learning for complex tasks (Kaiser et al., 2019; Hafner et al., 2018; A. Xie, Ebert, Levine, & Finn, 2019; Holland, Talvitie, & Bowling, 2018; Hamrick, 2019) but the learned models are generally unstructured, and do not necessarily work well for long horizon predictions. In the Tools game, long horizon predictions are critical for finding good policies, which will strain many of these approaches. We expect that progress in object-oriented, event based model learning will be necessary to learn with such long horizons successfully (Jayaraman, Ebert, Efros, & Levine, 2018; Pertsch et al., 2019; Becker-Ehmck, Peters, & Van Der Smagt, 2019; Battaglia, Pascanu, Lai, Rezende, et al., 2016; Chang, Ullman, Torralba, & Tenenbaum, 2016).

Meta-learning for multi-task reinforcement learning has emerged recently as an exciting direction for tackling multi-task reinforcement learning in model-based and model-free settings (Finn, Abbeel, & Levine, 2017; Rakelly, Zhou, Finn, Levine, & Quillen, 2019; Gupta, Mendonca, Liu, Abbeel, & Levine, 2018; Schmidhuber, Zhao, & Schraudolph, 1998; Wang et al., 2018) . While such approaches seem promising for the Tools game, they have mostly been analyzed in settings with narrow generalization between train and testing tasks, and it is therefore difficult to predict how well they will do when much broader generalization is required.

Model-based policy search has closely examined the role of learned models in forming plans, and has generally tried to come up with solutions for how to handle imperfect models effectively (see Deisenroth et al. (2013) for a survey). We expect that in more complicated, sequential versions of the game (such as drawing tools), planning and search methods from this community may need to be further improved in order to handle large action and state spaces that involve complex geometry and dynamics.

Tool use research has focused on the role of curiosity and social cues to develop the motor primitives necessary for manipulating simple objects to accomplish a goal (Forestier & Oudeyer, 2016), as well as how to use existing abilities to form more complex plans (Toussaint, Allen, Smith, & Tenenbaum, 2018). Recent work has also investigated combining imitation learning with model learning and policy updates to learn how to use objects as tools from pixel inputs alone (A. Xie et al., 2019), as well as how to use specific tools to accomplish new tasks with self-supervised objectives (Fang et al., 2018). These approaches require large amounts of self-supervised interactions with objects to learn good models and policies for manipulating the world.

5.2 A Deep Reinforcement Learning baseline: Preliminary studies

We looked at whether a popular approach from deep reinforcement learning, Deep Q Networks (DQN, a Q-learning method; Mnih et al., 2015), could solve the Tools game from pixel inputs alone.[4] While we expected that such

[3]Unlike traditional reinforcement learning tasks, the tools game as presented here is not obviously a sequential decision making task; instead, it could be viewed as a contextual bandit. However, because there are observable long-term effects of actions on reward outcomes, we believe that viewing the task as a sequential decision making task where the state information includes rich histories (similar to Hamrick et al., 2017) will be a fruitful future direction for this work.

[4]We also considered Proximal Policy Optimization (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017), but were unable to train the network to perform above chance levels, even within a single level template; see Allen, Smith, and Tenenbaum (2019).

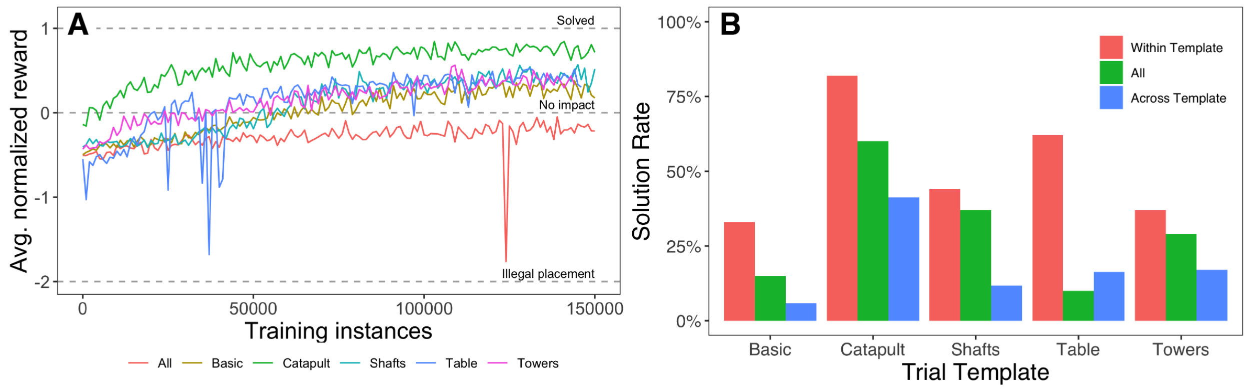


Figure 9: **(A)** Training curves for DQN training on individual level templates, or all templates combined (All, red curve). Reward is averaged over each epoch of 1,000 training instances. Models trained on individual scene templates are able to consistently choose successful or promising actions; however, the model trained to generalize across templates underperforms, not even consistently achieving a reward that is better than having no impact on the scene. Note that this is because the model has over-specified to some levels and not others. In some levels it has positive impact, and in other levels it has negative impact. **(B)** Accuracy of DQN on the validation template levels. Models are grouped by training regime: within template, all templates simultaneously, or averaged across all models trained on alternate templates.

approaches would require significant amounts of experience to learn useful representations, we hoped it might discover generally useful priors (like being object-oriented, or even understanding whether something should be placed above or below another object) that may transfer across different level types.

Our DQN is provided separate pixel-level representations of the game screen and each tool (e.g., see Fig. 1A), which it parses into a single representation for that level. This representation was used to separately learn a policy for tool choice, and a policy for where to place tools.[5] Since the game consists of a hybrid action space (consisting of one discrete tool choice, and one continuous tool position), we discretized the continuous space into a 20x20 grid, which is sufficient to solve most of the levels. See Section S3 for further architecture details.

Because we do not have a large enough number of different levels from our full game for learning useful state representations from visual input, we created a set of parametric level generators for a subset of the levels shown in Figure 2. Each parametric level generator takes a level “template" and varies parameters of that template (such as the location and size of various objects) to create different levels, and assigns a random set of three tools to that level. We designed level templates based off of the Basic, Catapult, Shafts, Table (A), and Towers (A) levels (see Fig. 2). From these templates we generated 1,000 levels, split into 900 training levels and 100 levels held out for validation. See Section S3.2 for further detail on random level generation.

We considered two training schemes for DQN: training on only a single level template, or training on levels from all templates at once. Training continued until the models observed 150,000 instances, at which point performance appeared to stabilize for all models (see Fig. 9A). The reward function for DQN is identical to the one used for SSUP: a normalized reward with respect to what would have happened if no action was taken. We only consider the version of DQN trained on all templates in our model comparisons in Section 4.2.1, since it performs better overall than models trained on individual level templates.

Learning curves for each model are shown in Figure 9A. In all cases, models trained specifically for one level solve the validation levels from that template more often than the model trained across all levels (Fig. 9B). Indeed, the model trained on all levels seems to have learned a “launching” strategy which is sometimes successful for those levels which involve launching an object (Basic, Catapult, Shafts and Towers). However, it is not able to simultaneously learn a “supporting” strategy for the Table template, where solutions require supporting an object from below.

These training results suggest why the DQN + Updating model fared so poorly in comparison to the SSUP model: while it is possible to learn policies from action-reward feedback in a single scenario, it is difficult to do so for multiple scenarios simultaneously.

[5]We also considered a model that learned a position for each tool, but this did not perform as well as a factorized model.

5.3 Future extensions

There is exciting work currently being developed to set up multi-task reinforcement learning benchmarks, such as OpenAI's Sonic and Coin Run challenges (Nichol, Pfau, Hesse, Klimov, & Schulman, 2018; Cobbe, Klimov, Hesse, Kim, & Schulman, 2018). The Tools game expands on this growing set by providing a strong test of transfer learning for physical planning and acting, but additionally requires few-shot learning within the test domains.

While we see the Tools game as presented to be a challenge for current AI approaches, we also consider ways of extending it to keep pace as a challenge problem for future AI. We could make the action and state space more complex by requiring sequential tool placements. Or, inspired by the way that crows and children can shape objects to make tools (Shumaker et al., 2011; Beck et al., 2011), we could require agents to design their own tools instead of selecting from a set of provided objects. We could also make the dynamics more complex by introducing objects with various densities, frictions, and elasticities, which would need to be learned through interaction. Each of these changes would be a simple extension to the game, yet we would expect them to push the boundaries of artificial agents for years to come.

6 Discussion

We introduced the Tools game for investigating flexible physical problem solving in humans and machines, and showed that human behavior on this challenge is rich and captures a wide variety of different trial-and-error problem solving strategies. We also introduced a candidate framework for understanding human physical problem solving: "Sample, Simulate, Update." This framework presumes that to solve these physics problems, people rely on existing knowledge of how the world works that they can apply to these games. Learning in this game, therefore, does not involve learning better models of physics for the game, but instead involves rapidly learning how to act in each specific instance, using a structured trial-and-error search.

Our specific instantiation of this framework captures how participants' attempted strategies evolve from start to finish (see Fig. 7), and the overall difficulty of the different levels. In contrast, alternate learning approaches – Deep Q learning and learning better parameterizations of a physical model – do not explain human physical problem solving on these tasks as well. We find that all three components of this model independently help to explain human performance on the Tools game. The prior helps the model focus on areas of the action space that are likely to make a difference in the problem, just like people do. The noisy simulation engine is necessary to filter out poor action proposals that people do not choose. The policy generalization from internal and external actions allows the model to efficiently search the hypothesis space in a more human-like way.

6.1 Related work in cognitive science

6.1.1 Problem solving

While "problem solving" has a long history in cognitive science, the trial-and-error based problem solving we discuss here differs in flavor from traditional problem solving. Unlike Complex Problem Solving where the goal is to learn the dynamics of an evolving situation on the fly (Frensch & Funke, 1995), people come equipped with knowledge of physics but must determine how to apply it to a specific situation. Insight Problem Solving, in which the solution requires an 'a-ha' moment of restructuring the problem space (Chu & MacGregor, 2011; Gick & Holyoak, 1980), is not enough to capture the iterative, incremental progress people show in finding solutions to these problems.

Instead, we consider the Tools game to be more similar to "problem solving as search" (Newell & Simon, 1972): finding a solution requires sifting through a large set of possible actions to select the useful ones. The challenge for people, then, is searching efficiently through this space by learning which actions lead to good outcomes.

6.1.2 Learning from observations and from thinking

A crucial dimension of the Tools game is that it relies on a body of knowledge that develops in infancy: physical interactions between objects (Spelke & Kinzler, 2007). Because people come equipped with a predictive model that they can quickly apply to the game, they have a reasonable sense of whether an action would help them solve the level. Yet despite this knowledge, people do not solve most levels immediately – in some cases people might be unsure whether a specific action will lead to a solution, and in other cases people might easily identify that an action will solve a level but never consider that action to begin with.

The first case arises because our physical predictions are probabilistic (Battaglia et al., 2013; Smith & Vul, 2013). These predictions give us a range of possible outcomes for any action, and so in many cases we will have some uncertainty about whether that action will be successful. Taking an action in the world, therefore, does not just allow us to exploit our knowledge and potentially solve the level, but also provides us with more information about this action-outcome link (Dasgupta et al., 2018).

The second case arises because the space of possible actions is so large, and using mental simulation to hypothesize the result of an action is not cost-free (Hamrick, Smith, Griffiths, & Vul, 2015). While we could in theory know an (approximate) link between any action and an outcome, in practice there are many actions that we have not yet thought about. This is therefore an instance of "learning by thinking" (Lombrozo, in prep): translating knowledge from one source (internal models of physics) to another, more specific instantiation (a mapping between actions and outcomes on *this particular* level).

This "learning by thinking" can be thought of in the reinforcement learning sense as training a model-free policy from a model-based system. This technique has been proposed as a way to leverage limited experience through a system that builds a model, yet is able to act with limited computation like a model-free system (Sutton, 1990). Recently, evidence for this sort of knowledge transfer has been found to exist in people (Gershman, Markman, & Otto, 2014; Gershman, Zhou, & Kommers, 2017). However, this work, inspired by classical reinforcement learning, studies processes wherein a model of the world is being learned and used to train a model-free system at the same time; in the Tools game, on the other hand, peoples' models of physics do not need to be learned, but instead can be used to derive these action-outcome mappings in particular cases (see also Dasgupta et al., 2018). This forms the basis of our proposed "Sample, Simulate, Update" framework, where the model is assumed to exist prior to the task, and then used to fine-tune a model-free policy.

6.1.3 Motor learning and tool use

The Tools game and the SSUP framework draws on inspiration from human motor learning and tool use, with some important differences. While dynamics models have been shown to play a central role in human motor planning and control (Heald, Ingram, Flanagan, & Wolpert, 2018; Wolpert, Ghahramani, & Jordan, 1995), as well as in the early stages of task learning (Daw, Niv, & Dayan, 2005), we are not aware of work showing that they are used for direct policy optimization.

Our results additionally show the importance of internal models beyond motor planning and control. Osiurak and Badets (2016) argue that an important component of general tool use is "mechanical knowledge," which can be instantiated by mentally simulating how one would use the tool. However, this mechanical knowledge can be difficult to measure in everyday tool use, because real-world tool use can also rely on stereotyped motor actions. We have developed a framework which explicitly decouples manipulation knowledge from tool use (Osiurak & Heinke, 2018), showing that people can still quickly solve these physical problems involving tools even when manipulation is not involved. We hope that this task will be used to further investigate to what extent manipulation and motor programs interact with physics-based problem solving for general tool use.

6.2 Towards even more human-like SSUP models

Here we have proposed that people solve the physics problems of the Tools game using a set of cognitive systems that map onto the "Sample, Simulate, Update" framework. While we show that a model that instantiates this framework can often explain human behavior on the Tools game, we believe that this particular model is only an approximation of the cognitive processes that people bring to this task. Here we discuss ways we believe our model does not quite capture the SSUP framework in the mind.

6.2.1 Sampling from rich priors

We argue that people are biased to consider only actions that will have an effect on the scene in order to avoid wasting cognitive effort on considering useless actions. For the purposes of our model, we instantiated this prior as placing tools above or below movable objects in the scene, but did not consider how this might be affected by the scene or specific tools used. However, people likely have a much more fine-grained prior for proposing candidate actions.

Consider the model's failure case of Falling (A) (Fig. 7). Here, participants must tip over a container with a ball inside it in order to accomplish the goal. This is most easily achieved by placing an object underneath the container. However, a much greater number of participants used the L shaped object, presumably as a "hook" to tip the container over from above, than the SSUP model predicted. This strategy does not lead to success, and people quickly change their plan after observing the first failure. Likewise, consider the hypothetical trial shown in Fig. 10A. Because there is only one tool which fits into the hole, it is immediately obvious that this is the right action to perform. This is a much richer prior than we have accounted for, which takes into account "suspicious coincidences" between the scene's geometry and the geometry of each of the tools.

These examples suggest that peoples' notions of what are good actions to consider are specific to the tool and context that it is used in. We believe this is connected to the notion of *affordances* (Norman, 1988): we see how we might use a particular tool in a particular context. In future work, it will be important to consider how specific tools and contexts might bias the types of actions that come to mind. This could be instantiated in the SSUP framework as different priors for each object (or each object-scene interaction), but it remains an open question how people might form these object-specific priors, or how they might be shaped over time via experience with different tools.

6.2.2 Simulating over abstract states

Our simulator can be thought of as providing the imagination-based reasoning to determine how tools could be used to accomplish tasks (Osiurak & Badets, 2017). We assumed a noisy physics engine as the simulator in this model, which requires planning over the full continuous action space. However, work in robotics has developed approaches to planning in more abstract *event* spaces (Toussaint et al., 2018; Kaelbling, Pasula, & Zettlemoyer, 2007) that can be grounded to continuous physical states. We believe such mechanisms could provide a way of bridging more qualitative physical simulation (Forbus, 1988) with the physics engines that simulate over continuous spaces used here. Further, we imagine that improvements from the machine learning community on predictive model learning may be able to learn these simulators (Chang et al., 2016; Battaglia et al., 2016), and hope that the Tools game will spur further progress in this direction.

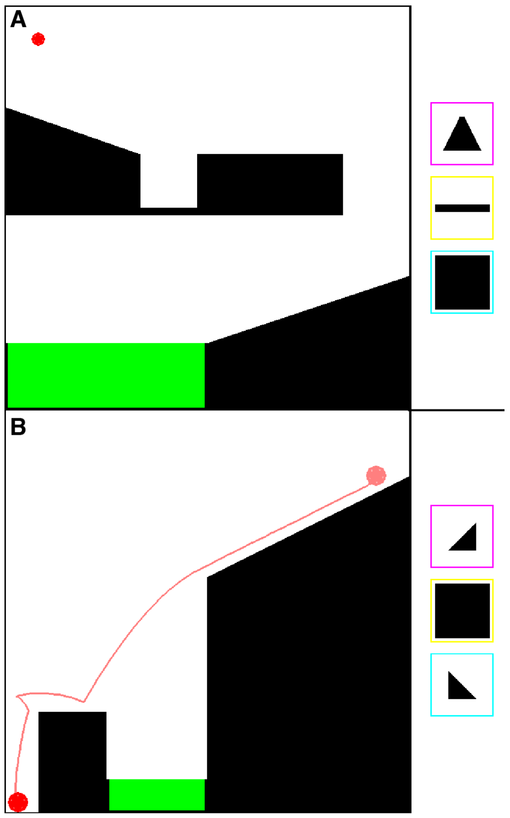


Figure 10: Two problems for future challenges. **(A)** A trial involving a "suspicious coincidence" in that there is a perfectly shaped hole for one of the tools to fit into. **(B)** An observation requiring causal intervention in order to know where to intervene on the path of the object to get it in the goal.

6.2.3 Updating and planning over complex actions

The current update framework is relatively straightforward, due to the simple nature of our action space. Even this simple action space proved challenging for model-free deep reinforcement learning methods, but we expect that more recent approaches may fare better, and that there will be rapid progress in this direction over the next few years. The next challenge will be to move towards richer action spaces involving creating tools from scratch, modifying tools, and using multiple tools in order to reach a solution (Shumaker et al., 2011). Our model, as a sampling-based approach, will certainly break down in these more complex action spaces, which will demand more sophisticated planners than we investigated here.

The current update strategies are also too simplistic in that they assume that only a reward signal is received, which is somehow indicative of what one might do next. However, observing a full trajectory such as that shown in Figure 10B, provides a substantial clue about what we might need to do. In this case, we need to intervene on the path of the ball by putting something in the way of the ball *at the right point* along its trajectory. This kind of reasoning is not about updating a model, or updating a policy based on reward, but rather understanding the causal structure of the world and acting to change the outcome.

6.2.4 Learning over multiple time scales

We found that in the current instantiation of the Tools game, learning happened rapidly and only at the level of action-outcome mappings. However, this is not to suggest that there is not learning within the prior or simulator components. As discussed in Section 6.2.1, people likely have more complex priors than the simple object-oriented ones we used in our models here. These priors are likely shaped by experience: if one discovers a novel way to use a tool, or sees another person using a tool in a novel way, that capability might spring to mind much more easily in the future. Because we tested participants using a set of levels that presented a variety of different physical principles with a variety of different tools, there was little opportunity for re-use of tool knowledge of this sort. However, studying how people learn and update their beliefs over repeated exposures to the same object is an exciting area for future study.

Similarly, while we found that participants did not learn better parameterizations of their internal physics model on this task, this might have been because there was little need to do so: the game was designed to roughly mimic realistic physics, and there was virtually no variability in object properties that would require learning individualized attributes. But we know that people can make inferences about physical object properties (such as mass or elasticity) from observing those objects interact dynamically (Warren, Kim, & Husney, 1987; Sanborn et al., 2013; Hamrick et al., 2016; Yildirim et al., 2018; Bramley, Gerstenberg, Tenenbaum, & Gureckis, 2018; Ullman, Stuhlmüller, Goodman, & Tenenbaum, 2018). Thus we believe that when it is relevant, people can learn on the two levels: refining their simulation model, and learning the action-outcome mapping.

Finally, understanding how to use tools is often a social phenomenon: young children learn how to use objects not just from individual exploration, but from observations of their parents or peers (Hopper, Flynn, Wood, & Whiten, 2010). Indeed, the transmission of tool creation and use across generations is considered to be a crucial component of human civilization (e.g., the 'cultural ratchet,' Tomasello, 1999). While it is often difficult to study the generation and transmission of tools in a laboratory setting, we believe that the Tools game could be used to study how people pass along created objects or knowledge demonstrations in an online setting.

7 Conclusion

The Tools game taps into a quintessentially human capability to flexibly use objects in our environment to achieve our goals. We have proposed the SSUP framework, and shown that a model within this framework can solve these problems faster and more accurately, and in more human-like ways, than many alternatives. We hope that this work will inspire both the Cognitive Science and Artificial Intelligence communities to perform further research into the computations underlying flexible physical reasoning, serving as a *tool*box for studying human capabilities for complex, object-based planning and manipulation, and as a challenge domain for AI agents to demonstrate generalizable knowledge about physical dynamics and planning.

Acknowledgements

The authors would like to thank Leslie Kaelbling, Roger Levy, Eric Schulz, Evan Shelhamer, and Tom Silver for their helpful comments in preparing this manuscript, and Mario Belledonne for helping to set up the probabilistic programming inference framework. This work was supported by NSF STC award CCF-1231216, ONR MURI N00014-13-1-0333, and research grants from ONR, Honda, and Mitsubishi Electric.

References

Allen, K., Smith, K., & Tenenbaum, J. (2019). Rapid trial-and-error learning in physical problem solving. *Structure and Priors in Reinforcement Learning Workshop, ICLR*.

Battaglia, P. W., Hamrick, J. B., & Tenenbaum, J. B. (2013, November). Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, *110*(45), 18327–18332. doi: 10.1073/pnas.1306572110

Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D. J., et al. (2016). Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems* (pp. 4502–4510).

Beck, S. R., Apperly, I. A., Chappell, J., Guthrie, C., & Cutting, N. (2011, May). Making tools isn't child's play. *Cognition*, *119*(2), 301–306. doi: 10.1016/j.cognition.2011.01.003

Becker-Ehmck, P., Peters, J., & Van Der Smagt, P. (2019). Switching linear dynamics for variational bayes filtering. *arXiv preprint arXiv:1905.12434*.

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, *47*, 253–279.

Brain it on. (2015). `https://brainitongame.com/`.

Bramley, N. R., Gerstenberg, T., Tenenbaum, J. B., & Gureckis, T. M. (2018, September). Intuitive experimentation in the physical world. *Cognitive Psychology*, *105*, 9–38. doi: 10.1016/j.cogpsych.2018.05.001

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Chang, M. B., Ullman, T., Torralba, A., & Tenenbaum, J. B. (2016, December). A Compositional Object-Based Approach to Learning Physical Dynamics. *arXiv:1612.00341 [cs]*. (arXiv: 1612.00341)

Chiuso, A., & Pillonetto, G. (2019). System identification: A machine learning perspective. *Annual Review of Control, Robotics, and Autonomous Systems*, *2*, 281–304.

Chu, Y., & MacGregor, J. N. (2011, February). Human Performance on Insight Problem Solving: A Review. *The Journal of Problem Solving*, *3*(2). doi: 10.7771/1932-6246.1094

Cobbe, K., Klimov, O., Hesse, C., Kim, T., & Schulman, J. (2018). Quantifying generalization in reinforcement learning. *arXiv preprint arXiv:1812.02341*.

Craik, K. J. W. (1943). *The Nature of Explanation*. CUP Archive. (Google-Books-ID: wT04AAAAIAAJ)

Cusumano-Towner, M. F., Saad, F. A., Lew, A., & Mansinghka, V. K. (2018, November). *Gen: A general-purpose probabilistic programming system with programmable inference* (Tech. Rep. No. MIT-CSAIL-TR-2018-020). Cambridge, Massachusetts: Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Dasgupta, I., Smith, K. A., Schulz, E., Tenenbaum, J. B., & Gershman, S. J. (2018). Learning to act by integrating mental simulations and physical experiments. In *Proceedings of the 40th Annual Meeting of the Cognitive Science Society.* doi: 10.1101/321497

Daw, N. D., Niv, Y., & Dayan, P. (2005). Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature neuroscience*, *8*(12), 1704.

Deisenroth, M., Neumann, G., Peters, J., et al. (2013). A survey on policy search for robotics. *Foundations and Trends® in Robotics*, *2*(1–2), 1–142.

Deisenroth, M., & Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th international conference on machine learning (icml-11)* (pp. 465–472).

Fang, K., Zhu, Y., Garg, A., Kurenkov, A., Mehta, V., Fei-Fei, L., & Savarese, S. (2018). Learning task-oriented grasping for tool manipulation from simulated self-supervision. *arXiv preprint arXiv:1806.09266*.

Feldbaum, A. (1960). Dual control theory. i. *Avtomatika i Telemekhanika*, *21*(9), 1240–1249.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th international conference on machine learning-volume 70* (pp. 1126–1135).

Forbus, K. D. (1988). Qualitative physics: Past, present, and future. In *Exploring artificial intelligence* (pp. 239–296). Elsevier.

Forbus, K. D., Gentner, D., & Law, K. (1995, April). MAC/FAC: A model of similarity-based retrieval. *Cognitive Science*, *19*(2), 141–205. doi: 10.1016/0364-0213(95)90016-0

Forestier, S., & Oudeyer, P.-Y. (2016, October). Modular active curiosity-driven discovery of tool use. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 3965–3972). Daejeon, South Korea: IEEE. doi: 10.1109/IROS.2016.7759584

Frensch, P. A., & Funke, J. (1995). *Complex problem solving: The european perspective*. Psychology Press.

Fu, J., Levine, S., & Abbeel, P. (2016). One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *2016 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 4019–4026).

Gentle, J. (2017). *ChipmunkJS*.

Gershman, S. J., Markman, A. B., & Otto, A. R. (2014, February). Retrospective revaluation in sequential decision making: A tale of two systems. *Journal of Experimental Psychology: General*, *143*(1), 182–194. doi: http://dx.doi.org/10.1037/a0030844

Gershman, S. J., Zhou, J., & Kommers, C. (2017, July). Imaginative Reinforcement Learning: Computational Principles and Neural Mechanisms. *Journal of Cognitive Neuroscience*, *29*(12), 2103–2113. doi: 10.1162/jocn_a_01170

Gerstenberg, T., Goodman, N. D., Lagnado, D. A., & Tenenbaum, J. B. (2015). How, whether, why: Causal judgments as counterfactual contrasts. In *37th Annual Meeting of the Cognitive Science Society* (p. 6).

Gick, M. L., & Holyoak, K. J. (1980, July). Analogical problem solving. *Cognitive Psychology*, *12*(3), 306–355. doi: 10.1016/0010-0285(80)90013-4

Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., & Levine, S. (2018). Meta-reinforcement learning of structured exploration strategies. In *Advances in neural information processing systems* (pp. 5302–5311).

Gureckis, T. M., Martin, J., McDonnell, J., Rich, A. S., Markant, D., Coenen, A., ... Chan, P. (2016, September). psiTurk: An open-source framework for conducting replicable behavioral experiments online. *Behavior Research Methods*, *48*(3), 829–842. doi: 10.3758/s13428-015-0642-8

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., & Davidson, J. (2018). Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*.

Hamrick, J. B. (2019, October). Analogues of mental simulation and imagination in deep learning. *Current Opinion in Behavioral Sciences*, *29*, 8–16. doi: 10.1016/j.cobeha.2018.12.011

Hamrick, J. B., Allen, K. R., Bapst, V., Zhu, T., McKee, K. R., Tenenbaum, J. B., & Battaglia, P. W. (2018, June). Relational inductive bias for physical construction in humans and machines. *arXiv:1806.01203 [cs, stat]*. (arXiv: 1806.01203)

Hamrick, J. B., Ballard, A. J., Pascanu, R., Vinyals, O., Heess, N., & Battaglia, P. W. (2017). Metacontrol for adaptive imagination-based optimization. *International Conference on Learning Representations*.

Hamrick, J. B., Battaglia, P. W., Griffiths, T. L., & Tenenbaum, J. B. (2016, December). Inferring mass in complex scenes by mental simulation. *Cognition*, *157*, 61–76. doi: 10.1016/j.cognition.2016.08.012

Hamrick, J. B., Smith, K. A., Griffiths, T. L., & Vul, E. (2015). Think again? The amount of mental simulation tracks uncertainty in the outcome. In *Proceedings of the 37th Annual Meeting of the Cognitive Science Society.*

Heald, J. B., Ingram, J. N., Flanagan, J. R., & Wolpert, D. M. (2018). Multiple motor memories are learned to control different points on a tool. *Nature human behaviour*, *2*(4), 300.

Henrich, J. (2017). *The secret of our success: How culture is driving human evolution, domesticating our species, and making us smarter*. Princeton University Press.

Hinton, G., Srivastava, N., & Swersky, K. (n.d.). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.

Holland, G. Z., Talvitie, E. J., & Bowling, M. (2018). The effect of planning shape on dyna-style planning in high-dimensional state spaces. *arXiv preprint arXiv:1806.01825*.

Hopper, L. M., Flynn, E. G., Wood, L. A., & Whiten, A. (2010, May). Observational learning of tool use in children: Investigating cultural spread through diffusion chains and learning mechanisms through ghost displays. *Journal of Experimental Child Psychology*, *106*(1), 82–97. doi: 10.1016/j.jecp.2009.12.001

Janner, M., Levine, S., Freeman, W. T., Tenenbaum, J. B., Finn, C., & Wu, J. (2018). Reasoning about physical interactions with object-oriented prediction and planning. *arXiv preprint arXiv:1812.10972*.

Jayaraman, D., Ebert, F., Efros, A. A., & Levine, S. (2018). Time-agnostic prediction: Predicting predictable video frames. *arXiv preprint arXiv:1808.07784*.

Juechems, K., & Summerfield, C. (2019). *Where does value come from?* (Preprint). PsyArXiv. doi: 10.31234/osf.io/rxf7e

Kaelbling, L. P., Pasula, H. M., & Zettlemoyer, L. S. (2007, July). Learning Symbolic Models of Stochastic Domains. *Journal of Artificial Intelligence Research*, *29*, 309–352.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., … others (2019). Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*.

Kersten, D., Mamassian, P., & Yuille, A. (2004). Object Perception as Bayesian Inference. *Annual Review of Psychology*, *55*(1), 271–304. doi: 10.1146/annurev.psych.55.090902.142005

Lembcke, S. (2013). *Chipmunk 2d Physics Engine.* Howling Moon Software.

Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, *17*(1), 1334–1373.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., … Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lockman, J. J. (2000). A perception–action perspective on tool use development. *Child development*, *71*(1), 137–144.

Lombrozo, T. (in prep). Learning by thinking in science and in everyday life. In *The Scientific Imagination.* Oxford University Press.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., … others (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529.

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Oxford, England: Prentice-Hall.

Nichol, A., Pfau, V., Hesse, C., Klimov, O., & Schulman, J. (2018). Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*.

Norman, D. A. (1988). *The psychology of everyday things.* Basic books.

Osiurak, F., & Badets, A. (2016). Tool use and affordance: Manipulation-based versus reasoning-based approaches. *Psychological Review*, *123*(5), 534–568. doi: 10.1037/rev0000027

Osiurak, F., & Badets, A. (2017). Use of tools and misuse of embodied cognition: Reply to Buxbaum (2017). *Psychological Review*, *124*(3), 361–368. doi: 10.1037/rev0000065

Osiurak, F., & Heinke, D. (2018, February). Looking for intoolligence: A unified framework for the cognitive study of human tool use and technology. *American Psychologist*, *73*(2), 169–185. doi: http://dx.doi.org/10.1037/amp0000162

Pertsch, K., Rybkin, O., Yang, J., Derpanis, K., Lim, J., Daniilidis, K., & Jaegle, A. (2019). Keyin: Discovering subgoal structure with keyframe-based video prediction. *arXiv preprint arXiv:1904.05869*.

Rakelly, K., Zhou, A., Finn, C., Levine, S., & Quillen, D. (2019). Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning* (pp. 5331–5340).

Sanborn, A. N., Mansinghka, V. K., & Griffiths, T. L. (2013, April). Reconciling intuitive physics and Newtonian mechanics for colliding objects. *Psychological Review*, *120*(2), 411–437. doi: http://dx.doi.org/10.1037/a0031912

Schmidhuber, J., Zhao, J., & Schraudolph, N. N. (1998). Reinforcement learning with self-modifying policies. In *Learning to learn* (pp. 293–309). Springer.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shumaker, R. W., Walkup, K. R., & Beck, B. B. (2011). *Animal Tool Behavior: The Use and Manufacture of Tools by Animals*. JHU Press. (Google-Books-ID: Dx7slq__udwC)

Smith, K. A., Battaglia, P. W., & Vul, E. (2018, June). Different Physical Intuitions Exist Between Tasks, Not Domains. *Computational Brain & Behavior*, *1*(2), 101–118. doi: 10.1007/s42113-018-0007-3

Smith, K. A., & Vul, E. (2013, January). Sources of Uncertainty in Intuitive Physics. *Topics in Cognitive Science*, *5*(1), 185–199. doi: 10.1111/tops.12009

Spelke, E. S., & Kinzler, K. D. (2007, January). Core knowledge. *Developmental Science*, *10*(1), 89–96.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990* (pp. 216–224). Elsevier.

Sutton, R. S. (1991, July). Dyna, an Integrated Architecture for Learning, Planning, and Reacting. *SIGART Bull.*, *2*(4), 160–163. doi: 10.1145/122344.122377

Todorov, E., Erez, T., & Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 ieee/rsj international conference on intelligent robots and systems* (pp. 5026–5033).

Tomasello, M. (1999). *The Cultural Origins of Human Cognition*. Harvard University Press. (Google-Books-ID: hRFfCgAAQBAJ)

Toussaint, M., Allen, K. R., Smith, K. A., & Tenenbaum, J. B. (2018, June). Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning. In *Robotics: Science and Systems XIV.* Robotics: Science and Systems Foundation. doi: 10.15607/RSS.2018.XIV.044

Ullman, T. D., Stuhlmüller, A., Goodman, N. D., & Tenenbaum, J. B. (2018). Learning physical parameters from dynamic scenes. *Cognitive psychology*, *104*, 57–82.

Wang, J. X., Kurth-Nelson, Z., Kumaran, D., Tirumala, D., Soyer, H., Leibo, J. Z., … Botvinick, M. (2018). Prefrontal cortex as a meta-reinforcement learning system. *Nature neuroscience*, *21*(6), 860.

Warren, W. H., Kim, E. E., & Husney, R. (1987, June). The Way the Ball Bounces: Visual and Auditory Perception of Elasticity and Control of the Bounce Pass. *Perception*, *16*(3), 309–336. doi: 10.1068/p160309

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*(3-4), 229–256.

Wolpert, D. M., Ghahramani, Z., & Jordan, M. I. (1995). An internal model for sensorimotor integration. *Science*, *269*(5232), 1880–1882.

Xie, A., Ebert, F., Levine, S., & Finn, C. (2019). Improvisation through physical understanding: Using novel objects as tools with visual foresight. *arXiv preprint arXiv:1904.05538*.

Xie, C., Patil, S., Moldovan, T., Levine, S., & Abbeel, P. (2016). Model-based reinforcement learning with parametrized physical models and optimism-driven exploration. In *2016 ieee international conference on robotics and automation (icra)* (pp. 504–511).

Yildirim, I., Smith, K. A., Belledonne, M., Wu, J., & Tenenbaum, J. B. (2018). Neurocomputational Modeling of Human Physical Scene Understanding. In *2018 Conference on Cognitive Computational Neuroscience.* Philadelphia, Pennsylvania, USA: Cognitive Computational Neuroscience. doi: 10.32470/CCN.2018.1091-0

Supplementary Information

S1 Experiment

S1.1 Procedure

We recruited 94 participants from Amazon Mechanical Turk using the psiTurk framework (Gureckis et al., 2016). The experiment lasted 15-20 minutes, for which participants were compensated $2.50.

On each trial, participants were initially presented with a freeze-frame of the scene and a goal description (physics switched off) (Fig. 1A). They were instructed that all of the black objects were immovable, but when physics was turned on, the blue and red objects may move. They were provided with three 'tools' that they could place anywhere in the scene (that did not overlap with other objects, goals, or out-of-bounds areas), by clicking on a tool and then clicking where they would like to place it (Fig. 1B). As soon as this tool was placed, physics would be switched on using a Javascript version of the Chipmunk 2D physics engine (Gentle, 2017; Lembcke, 2013), and all movable objects would start to fall under the force of gravity (Fig. 1C); after this participants could no longer intervene on the scene. However, participants could click a 'reset' button at any time to return the scene to its initial state and try another action. Participants were given two minutes to solve the problem – if they solved it they could move onto the next level immediately. Otherwise, they could choose to move on any time after two minutes had passed. Within each trial, we recorded all attempted actions: which tool was used, where it was placed, and the clock time when it was placed since the start of the trial. See `https://k-r-allen.github.io/tool-games/` for videos demonstrating this procedure.

To familiarize participants with the experiment, we initially instructed them on the way the game worked, then gave them a 'playground' level with no goal to introduce them to the dynamics of the world. Participants had to remain in the playground for at least 30s and try at least two tool placements before moving on. Finally, participants were given two simple practice levels that they were required to solve before the main part of the experiment began; these were not analyzed.

Participants were asked to solve 14 of the 20 levels: all eight unpaired levels and one each of the six paired levels (so that learning in one instance of a pair would not affect performance in the other). The choice of which instance of a pair was determined randomly at the start of the experiment. The order in which levels were presented was additionally randomized.

S1.2 Data cleaning

To standardize results across participants, we eliminated any actions taken after 120s had passed. This affected 7.1% of all trials (93); of those, 33 were eventually solved but are counted as unsolved for our analyses. To further standardize results between participants and the model, we treated any actions that would have accomplished the goal within 20s as successes, regardless of whether participants reset the scene before the success could be counted or waited longer than 20s for a solution; this caused 4.5% (59) of all trials to be analyzed differently than participants experienced them.

S2 SSUP model implementation

Algorithm S1 demonstrates how the model of SSUP is implemented to perform the Tools game. Further details are below.

S2.1 Implementing *sampling*: an object-oriented prior

The object-oriented prior can be described as a categorical sample on which object to interact with, and a Gaussian distribution specifying the position of the tool relative to that object. Formally, the generative procedure for the prior follows:

- Sample a dynamic object in the scene $object \sim Multinomial(\{\frac{1}{n_{obj}} || i \in [0,...,n_{obj}]\})$
- Sample a position y relative to that object, using a Gaussian distribution parameterized with mean $object_y$ and standard deviation σ_y, truncated on each side by the legal lowest and highest positions respectively
- Sample a position x relative to that object:
    - Compute the left and right edges of the bounding box for that object, BB_{left} and BB_{right}
    - Sample a value v uniformly between $BB_{left} - \sigma_x$ and $BB_{right} - \sigma_x$.

Algorithm S1 SSUP model for the Tools game

```
Sample $n_{init}$ points from prior $\pi(s)$ for each tool
Simulate actions to get noisy rewards $\hat{r}$ using internal model
Initialize policy parameters $\theta$ using policy gradient on initial points
while not successful do
    Set $acting = False$
    With probability $\varepsilon$, sample action $a$ from prior
    With probability $1-\varepsilon$, sample action $a$ from policy
    Estimate noisy reward $\hat{r}$ from internal model on action $a$
    if $r > T$ then
        Set $acting = True$
        Try action $a$ in environment
    else if $i \geq n_{iters}$ then
        Set $acting = True$
        Try best action $a^*$ simulated so far which has not yet been tried
    end if
    if acting then
        Observe $r$ from environment on action $a$.
        If successful, exit.
        Simulate $\hat{r}$ assuming other two tool choices.
        Update policy based on all three estimates and actions.
    else
        Update policy using policy gradient
    end if
end while
```

- If $v < BB_{left}$ or $v > BB_{right}$, sample x from a normal centered on the edge of the bounding box with standard deviation σ_x.
- Otherwise, $x = v$.

This has the effect of sampling mostly uniformly around object extents in the x direction, but otherwise dropping off around the edges of objects proportionally to σ_x.

σ_x and σ_y are then free parameters which were chosen to reflect a relatively uniform prior in y and a tighter distribution in x. These decisions were examined using a sensitivity analysis shown in Figure S2.

We experimented with an alternative geometric prior, which could sample “above”, “below”, “left”, “right”, and “middle” of objects before then committing to Gaussian positions respecting that geometric decision, but found that this did not perform better than the more uninformed prior. We therefore use the more uninformed prior to reduce the number of free parameters in the model.

To initialize search, we then sample a number of initial points, $n_{initial}$, for each tool from this prior, and run each action through the noisy simulator. The policy is initialized with these noisy reward estimates.

S2.2 Implementing *simulation*

The noisy simulation engine within the SSUP model is meant to capture the essence of the human Intuitive Physics Engine (Battaglia et al., 2013). It is based on the Chipmunk physics engine just as the experiment is, but introduces stochasticity in the dynamics when objects collide, similar to how the uncertainty in human physical predictions increases when they must simulate through collisions (Smith & Vul, 2013; Hamrick et al., 2015). Collisions are made stochastic by inserting noise into the direction that objects bounce off of each other, and how much energy is transferred. This is accomplished by adjusting the direction that collision forces are applied (ϕ) and the elasticity (bounciness) of the collision (e), by adding noise according to two parameters (σ_ϕ, σ_e):

$$\begin{aligned} \phi' &\sim wrappedNormal(\phi, \sigma_\phi) \\ e' &\sim \mathcal{N}(e, \sigma_e) \; s.t. \; e' > 0 \end{aligned} \tag{1}$$

The SSUP model runs n_{sims} simulations (set here at 4) per imagined action to determine an average reward (r; see Section S2.3.2). This reward is then used to update the action-outcome policy parameters θ, and to decide on whether to take an action.

S2.3 Implementing *updating*

The main body of the SSUP model is the simulation-action loop, which updates the policy π with reward estimates from sampled actions. The policy π consists of first choosing which tool to use, and then conditioned on each tool, where to put it in the scene. We model this as a mixture of Gaussians, one Gaussian for the position of each tool. This gives 2 parameters for the tool weights because their probabilities have to sum up to 1, and 2 parameters for the Gaussian position on each tool (for a total of 2+6 parameters). In principle, the SSUP framework is agnostic to the particular form of the update. In practice, we found that a simple policy gradient worked well, outlined below, but other methods such as Bayesian optimization resulted in similar trends amongst the tested ablations and full model.

We use a simple policy gradient algorithm (Williams, 1992) to update our policy parameters:

$$\theta \leftarrow \theta + \alpha r \nabla_\theta \log \pi_\theta(a) \tag{2}$$

where a is the action taken, α is the learning rate, r is the reward, and θ are the policy parameters to be estimated.

We include exploration in our action selection, using an epsilon greedy strategy. ε is considered to be a free parameter of the model. When exploring, we sample actions from our object-oriented prior, outlined in section S2.1.

SSUP continues to sample actions until it either finds an action with a reward greater than a given threshold, T, or until it reaches a maximum number of simulations, n_{iters}. If it finds an action with high reward, it executes that action in the environment. Otherwise, if it reaches the maximum number of internal simulations, n_{iters}, it takes the best action it has simulated so far which has not been tried in the real world. If it succeeds, the model is finished. Otherwise it resumes simulation.

S2.3.1 Counterfactual updates

Whenever our model takes an action in the environment, it additionally queries its noisy simulator for what would have happened if it had used the other two tools. We found that this stabilized the policy gradient, such that it could determine whether the reward was due to the tool used, or the position chosen. We imagine that such an update might be generally useful in increasingly structured action spaces when a strong model is available.

S2.3.2 Reward function definition

Our reward is defined as the normalized minimum distance to the goal along the observed or simulated trajectory. This reward function was provided for every model that we considered. Normalization is performed with respect to what the outcome of this metric would be if the agent had taken no action, such that the reward can be calculated as:

$$r = 1 - \frac{min_{t=0,T;obj \in objects} d(obj, goal, action)}{min_{t=0,T;obj \in objects} d(obj, goal)} \tag{3}$$

where d is the distance between a goal object (red ball) and the goal under a particular action, and T is the total number of time-steps in the trajectory. The subtraction is to flip signs such that getting into the goal (at a distance of 0) results in the highest possible reward.

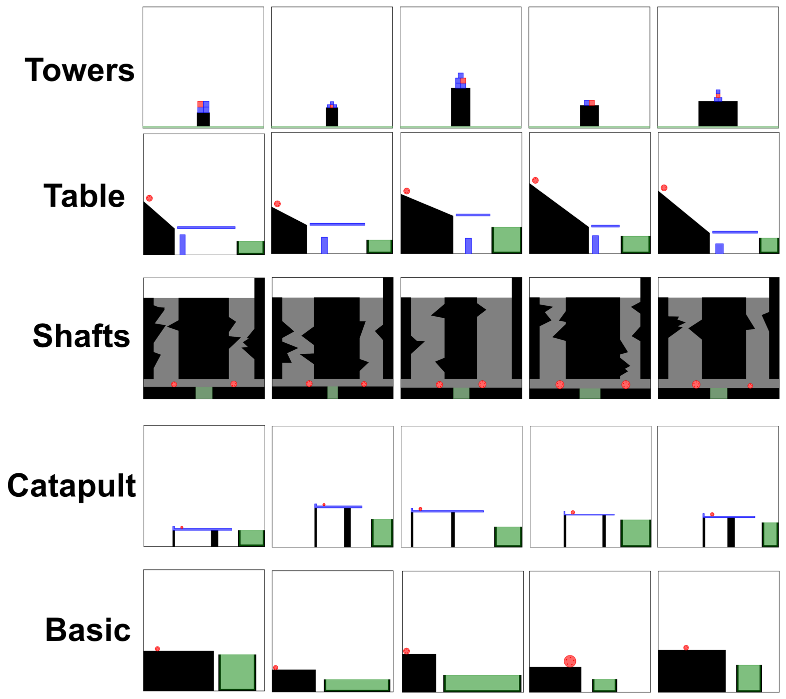
This is therefore a measure of *intervention* rather than generic distance to the goal. Across all experiments, we found this reward function to perform substantially better than one which was based only on the unnormalized minimum distance metric.

S3 DQN model details

S3.1 Architecture details

Our DQN architecture takes in an image of the screen and images of each tool in order to compute Q values for a discretized version of the space. The images are down-sampled such that the screen is 90x90 pixels and each tool is 30×30 pixels. We choose a discretization of 20×20 which is sufficient to solve most of the levels, and found this reliably converged to a reasonable policy after 150,000 iterations.

The architecture consists of four networks: a tool image network, a screen image network, a tool policy network, and a position policy network. We run each tool image through the network, fusing this with the screen image run through the network before passing the fused representation to both the tool policy and position policy networks. This gives us a 400-dimensional action vector for the position, and a 3 dimensional action vector for the tool choice.



(a) Randomly generated level screens

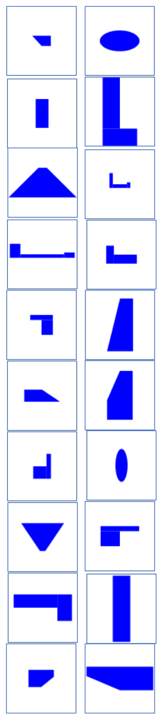

(b) Randomly generated tools that are paired with level screens

Figure S1: Randomly generated levels for deep learning baselines

- The tool image network consists of 4 convolutional layers, with 12, 24, 12, and 3 channels respectively at each layer. The kernel sizes are 3, 4, 3, and 2 respectively.
- The screen image network also consists of 4 convolutional layers, with 32, 64, 32 and 16 channels respectively, and kernel sizes 8, 4, 3, and 2, following the original DQN Atari network (Mnih et al., 2015).
- The tool policy network is a simple 2-layer multi-layer perceptron (MLP) with 100 hidden units.
- The position policy network is similarly a simple 2-layer MLP with 100 hidden units.

We use epsilon greedy exploration during training, with a linearly decreasing epsilon schedule. We use a batch size of 32 and a constant learning rate of 2.5×10^{-4}. For optimization, we use RMSProp (Hinton, Srivastava, & Swersky, n.d.).

S3.2 Random level generation

For the purposes of comparing human and model performance, we used 20 hand-designed levels. However, Deep Q Learning requires large amounts of data for training, and though we could allow training by taking extensive actions on the given levels, this would risk over-learning particulars of those 20.

Instead, we randomly generated levels from a set of five templates based on five of the hand-designed levels. These templates were designed such that they contained the same set of objects, and could be solved in similar ways, but the sizes and configurations of objects could vary, which enforced some variability in the solution actions. The specific algorithm for generating levels varied by template, but involved resizing or shifting many of the object, subject to some geometric constraints (e.g., the ‘tabletop’ in each Table level was always between the slope and goal, and the ball in each Catapult level always rested on the catapult object). See Fig. S1A for example scenes from each template.

Each generated level included three tools randomly drawn from a pool of possible shapes that were used to construct tools for the 20 hand-designed levels. These tools could further be randomly resized or rotated at angles of 90°, subject to the constraint that they continued to fit in the 90×90 pixel area that each of the original tools fit into. See Fig. S1B for examples of randomly generated tools.

To guarantee that each level has a reasonable but non-trivial solution, we proposed a “solution region” for each template that comprised a similar area to the primary solution for the base level. For instance, the solution region for the Basic template was a rectangle above the key ball, while the solution region for the Shafts template contained both areas directly above the shafts. We then randomly generated 100 tool placements from this region and selected only trials for which between 3% and 80% of actions would solve the level.

We used these templates to generate 1,000 levels each. These levels were then split to provide 900 training levels for each template, and 100 validation levels.

S4 Physical parameter tuning model details

As an alternate learning scheme, we suppose that people use observations from failed actions to refine their internal dynamics models for the game, and then use those updated models to choose the next action.

We instantiate this learning scheme using a model that tunes parameters within its physics engines that are related to object dynamics: object densities d, frictions f and elasticities e, which we collectively call ψ. This model relies on the same physics engine as the full SSUP framework, including uncertainty introduced during collisions (see Section S2.2).

After each failed action is performed, the parameter tuning model attempts to update its internal parameters to match the observed outcome of that action using Bayesian inference; this is an instance of "analysis-by-synthesis" (Kersten, Mamassian, & Yuille, 2004). The model observes 20s of a failed trajectory, and samples the location of all movable objects at 50 points evenly spaced in time. To approximate the likelihood of each observation given a parameterization ψ, the model produces 20 simulations of the same action, and records the positions of all movable objects in that simulation at the same time points. From these records, the model takes the positions of each object at each time point, and parameterizes a Gaussian over its likelihood of where that object should be at that point in time $(\mu^t_{obj}, \sigma^t_{obj})$. The likelihood of observing the full trajectory is therefore the product of each of the individual observed object positions at each time point:

$$P(O|\psi) = \prod_{obj}\prod_{t} P(o^t_{obj}|\mu^t_{obj}, \sigma^t_{obj}) \tag{4}$$

We parameterize the prior for each property as a Gaussian centered at the true value for that property (1 for density, 0.5 for elasticities and frictions), with variance 0.025. Our proposal function is defined as a set of truncated Gaussians with variances of 0.02 (and mean equal to the current estimate).

We implement this inference using the Metropolis-Hastings (MH) algorithm for 20 iterations using the likelihood function and proposal above, using 5 chains and 5 burn-in samples. We found that this was enough to give reliable estimates for each of the parameters. This procedure was implemented in the probabilistic programming language Gen (Cusumano-Towner, Saad, Lew, & Mansinghka, 2018).

Every time a new action is observed, we initialize the MH procedure using the parameters determined from the previous observation. In this way, learning (in the form of more refined priors) can occur throughout a set of actions within a level.

S5 Parameter sensitivity analysis

To ensure that the choice of parameters did not unduly affect model performance, we tested how well the SSUP model performed under different settings of each of its parameters. For a measure of model performance, we used the Total Area Between Cumulative Solutions (ABCS; see Section 4.2.1), as this captures the by-trial accuracy, number of actions used, and evolution of solution rate into a single metric.

Because the SSUP model includes 10 parameters, we could not reasonably test model performance across a full grid of parameter choices; instead, we test model performance along a single dimension at a time, varying one parameter but keeping all others constant. As can be seen in Fig. S2, model performance did not differ significantly across a wide range of parameter settings around the values used in the SSUP model. This suggests that more precise but computationally intractable parameter fitting would lead to at best marginal improvements in model fit.

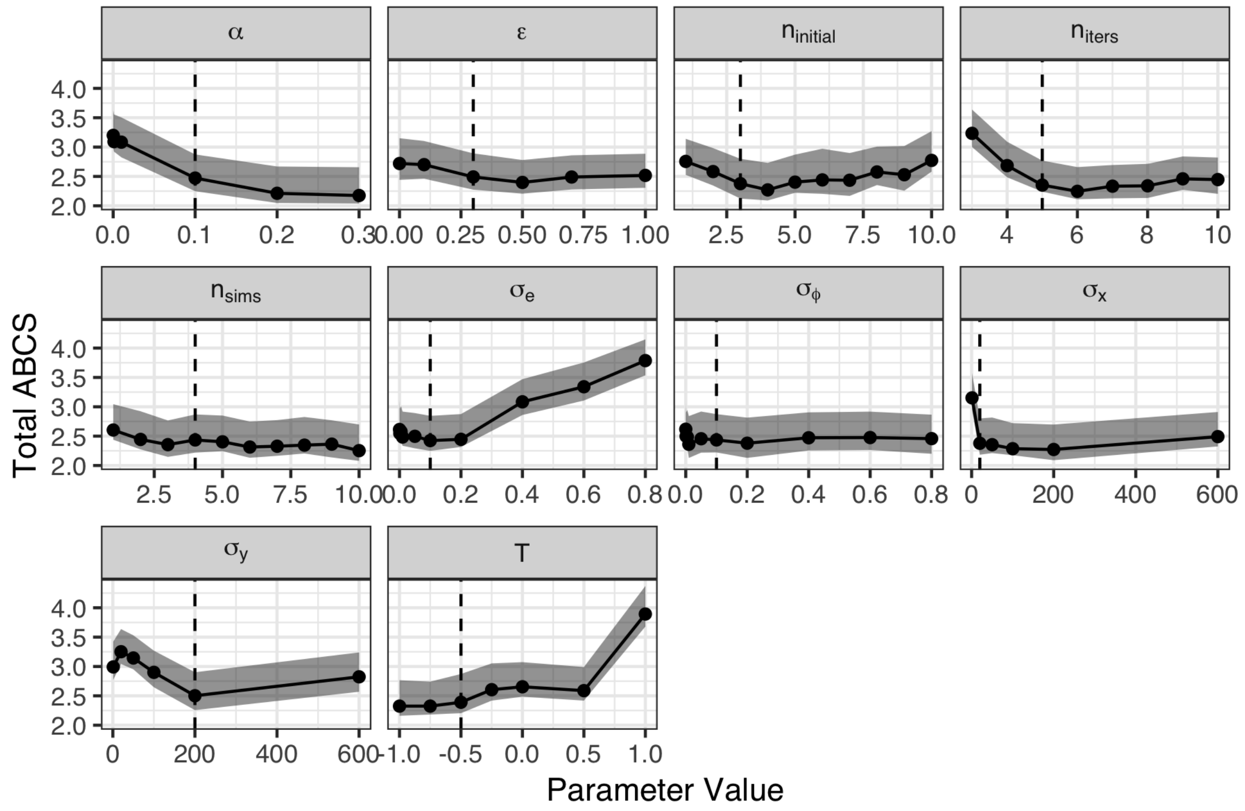


Figure S2: Total Area Between Cumulative Solutions (ABCS) values for different parameter values, keeping all other parameters at the default values. The default values are denoted by the dashed lines. Grey areas indicate 95% bootstrapped confidence intervals.

	SSUP	Pri + Sim	Pri + Upd	Sim + Upd	DQN + Upd	Param Tune
Basic	0.063	0.053	0.052	0.157	0.420	0.048
Bridge	0.045	0.134	0.179	0.546	0.668	0.101
Catapult	0.057	0.493	0.225	0.081	0.217	0.576
Chaining	0.340	0.504	0.561	0.441	0.750	0.528
Gap	0.261	0.162	0.253	0.213	0.155	0.153
SeeSaw	0.097	0.304	0.258	0.137	0.365	0.349
Unbox	0.099	0.053	0.079	0.080	0.143	0.038
Unsupport	0.137	0.335	0.265	0.077	0.403	0.329
Falling (A)	0.121	0.180	0.102	0.182	0.255	0.124
Falling (B)	0.250	0.332	0.503	0.660	0.756	0.358
Launch (A)	0.114	0.155	0.226	0.195	0.400	0.169
Launch (B)	0.052	0.090	0.267	0.341	0.393	0.143
Prevention (A)	0.029	0.024	0.030	0.027	0.713	0.018
Prevention (B)	0.034	0.035	0.022	0.017	0.712	0.064
Shafts (A)	0.016	0.123	0.261	0.340	0.524	0.062
Shafts (B)	0.147	0.173	0.173	0.176	0.198	0.165
Table (A)	0.154	0.147	0.148	0.100	0.041	0.134
Table (B)	0.135	0.335	0.402	0.107	0.237	0.358
Towers (A)	0.033	0.025	0.020	0.039	0.019	0.048
Towers (B)	0.030	0.117	0.340	0.130	0.576	0.111
Total	2.211	3.776	4.367	4.046	7.946	3.875

Table S1: Area Between Cumulative Solutions metrics for each level (row) by model (column).

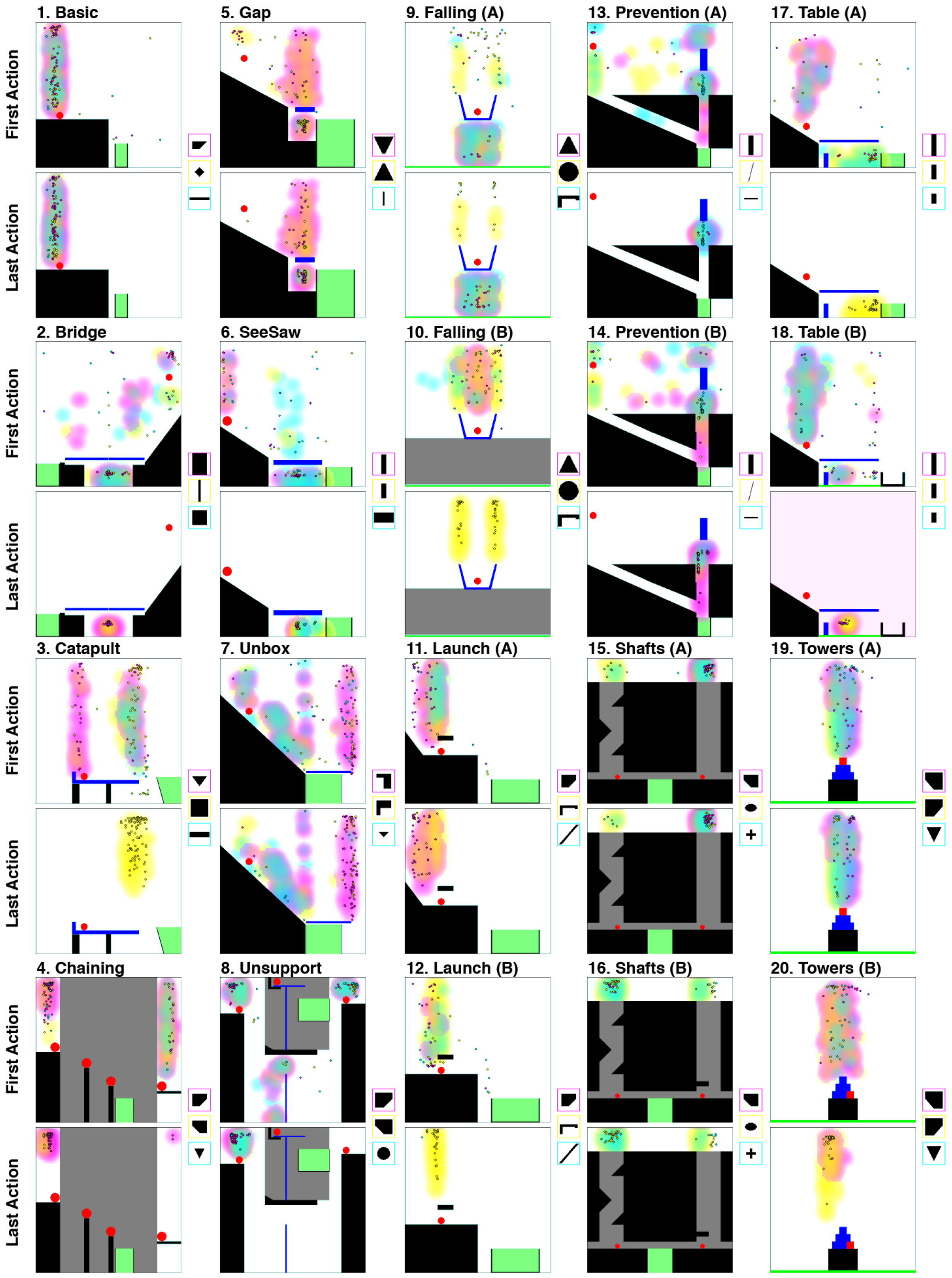


Figure S3: Distribution of predicted model actions (background) versus human actions (points) on the first attempts of the level (*top*) and the attempt used to solve the level (*bottom*) across all levels tested.