PNAS
www.pnas.org

Supplementary Information for

Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning

Kelsey R. Allen, Kevin A. Smith, Joshua B. Tenenbaum

Kelsey Allen.
E-mail: krallen@mit.edu

This PDF file includes:

Supplementary text
Figs. S1 to S12
Tables S1 to S4
References for SI reference citations

Supporting Information Text

1. Experiment

A. Procedure. Participants were recruited from Amazon Mechanical Turk using the psiTurk framework (1). We recruited 94 participants for the first experiment and compensated them $2.50 for 15-20 minutes of work. For the validation experiment we recruited 50 additional unique participants and compensated them $2.00 for slightly less than 15 minutes of work.

On each trial, participants were initially presented with a freeze-frame of the scene and a goal description (physics switched off) (Fig. 1D(i)). 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. 1D(ii)). This object would then be added as-is to the scene; participants could not rotate or rescale it. As soon as this tool was placed, physics would be switched on using a Javascript version of the Chipmunk 2D physics engine (2, 3), and all movable objects would start to fall under the force of gravity (Fig. 1D(iii)); 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.

In the validation experiment, all participants were asked to solve all 10 levels. Other than the level choice, all procedures and introductory materials were identical.

B. Data cleaning. To standardize results across participants, we only analyzed the first 120s of play, even though participants were allowed to continue play past the 120s mark. In the first experiment, this affected 7.1% of all trials (93); of those, 33 were eventually solved. In the validation experiment, this affected 16.8% of all trials (84), of which 25 were later solved. 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 in the first experiment, and 3.2% (16) of all trials in the validation experiment.

C. Learning over the experiment. While participants improved in solution rate over the course of the first 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.

2. SSUP model implementation

Algorithm 1 demonstrates how the SSUP model is implemented to perform the Virtual Tools game. Further details are below.

A. 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 $BB_{right} - \sigma_x$.
  - 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, $x = v$.

Algorithm 1 SSUP model for the Virtual Tools game

```
Sample n_init points from prior π(s) for each tool
Simulate actions to get noisy rewards r̂ using internal model
Initialize policy parameters θ using policy gradient on initial points
while not successful do
    Set acting = False
    With probability ϵ, sample action a from prior
    With probability 1 − ϵ, sample action a from policy
    Estimate noisy reward r̂ from internal model on action a
    if r > T then
        Set acting = True
        Try action a in environment
    else if i ≥ n_iters then
        Set acting = True
        Try best action a* simulated so far which has not yet been tried
    if acting then
        Observe r from environment on action a.
        If successful, exit.
        Simulate r̂ assuming other two tool choices.
        Update policy based on all three estimates and actions.
    else
        Update policy using policy gradient
```

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 S1.

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, the model first samples a number of initial points, $n_{initial}$, for each tool from this prior, and runs each action through the noisy simulator. The policy is initialized with these noisy reward estimates. Not initializing the policy in this way is detrimental, as can be seen in the sensitivity analysis.

B. Implementing *simulation*. The noisy simulation engine within the SSUP model is meant to capture the essence of the human Intuitive Physics Engine (4). 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 (5, 6). 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 for each collision 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):

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

The SSUP model runs n_{sims} simulations (set here at 4) per imagined action to determine an average reward (r; see Section C.2). This reward is then used to update the action-outcome policy parameters θ, and to decide on whether to take an action.

C. 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 4 parameters for the Gaussian position and variance on each tool (for a total of 2+12 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 (7) to update our policy parameters:

$$\theta \leftarrow \theta + \alpha r \nabla_\theta \log \pi_\theta(a) \quad [2]$$

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

The policy is initialized such that each Gaussian is centered in the middle of the screen with isotropic variance of 50px (1/12th of the height and width). It is then updated using noisy reward estimates from n_{init} sampled actions from the prior before any actions are taken. We found this dramatically improved stability and performed much better than a policy which was parameterized with respect to the objects in the scene.

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 A.

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.

C.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. The policy gradient is therefore calculated on three data points: one from each of the possible tools. These are also counted as the first two simulations of each inner loop. 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.

C.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:

$$r = 1 - \frac{min_{t=0,N;obj \in objects} d(obj, goal, action)}{min_{t=0,N;obj \in objects} d(obj, goal)} \quad [3]$$

where d is the distance between a goal object (red object) and the goal, under a particular action; N 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.

D. Running on the Virtual Tools game. The SSUP algorithm described in Algorithm 1 will continue to run until it finds a solution. However, to match human performance under time limits, we capped the number of actions that the model could take to 10, which was the median number of actions people took on trials for which they did not solve the level. If the model did not find a solution within 10 actions, it was considered to be unsolved.

3. 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. Critically, this alternate learning scheme *does not* sample actions from a policy which is updated from observations. It always samples actions from the object-centered prior, and only the noisy dynamics model can be improved and refined based on observations.

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 B).

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" (8). 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:

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

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 (9).

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.

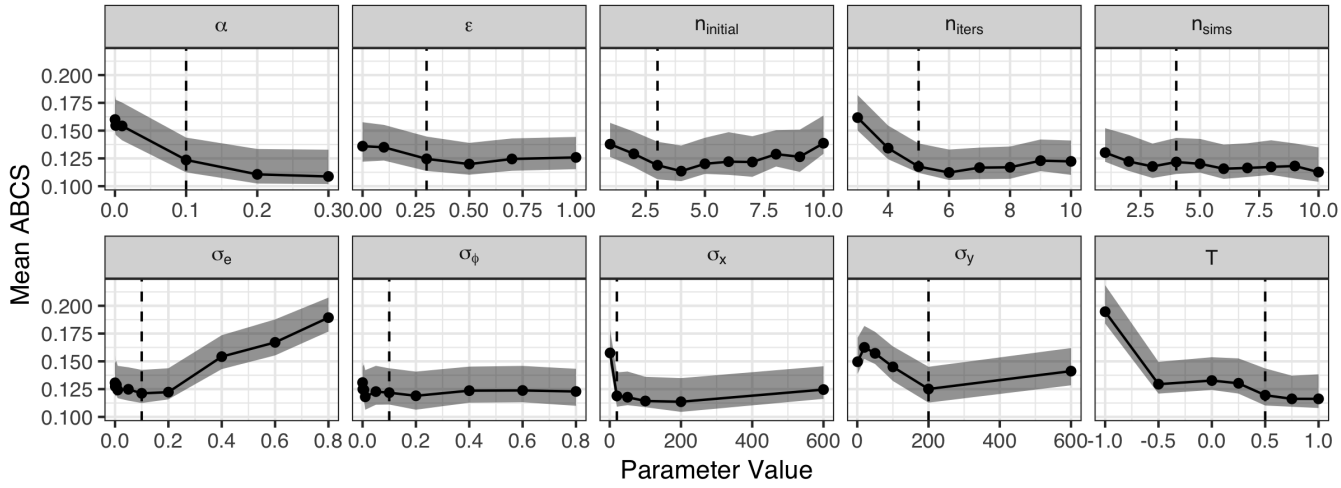


Fig. S1. Mean Area Between Cumulative Solutions (ABCS) values across all trials for different parameter values, keeping all other parameters at the default values. The default values are denoted by the dashed lines. Grey areas indicate 95% bootstrapped confidence intervals. Legend: α: learning rate for policy gradient, ϵ: the probability of sampling an exploratory action from the prior, $n_{initial}$: number of initial samples from the prior used to initialize the policy, n_{iters}: number of internal iterations before action must be taken, n_{sims}: the number of simulations run for each imagined action, σ_e: the noise (in radians) for collision elasticity, σ_ϕ: the noise (in radians) for collision direction, σ_x: the standard deviation of the prior in the x direction, σ_y: the standard deviation of the prior in the y direction, T: -1$\times$ the reward threshold for acting

4. 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 a metric called total Area Between Cumulative Solutions (ABCS). This metric is defined for each level as the area between the human and model cumulative solution curves (see Fig. 6A in the main text), normalized between 0 (perfect match) to 1 (participants or the model always solve the level instantaneously, while the other never solves the level). This metric combines 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. S1, 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.

As further evidence that these parameters generalize, we used an identical parameterization of the model for the 10 validation levels, and found good fits to human data there as well.

5. A Deep Reinforcement Learning baseline

We looked at whether a popular approach from deep reinforcement learning, Deep Q Networks (10), could solve the Virtual Tools game from pixel inputs alone.* While we expected that such 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.

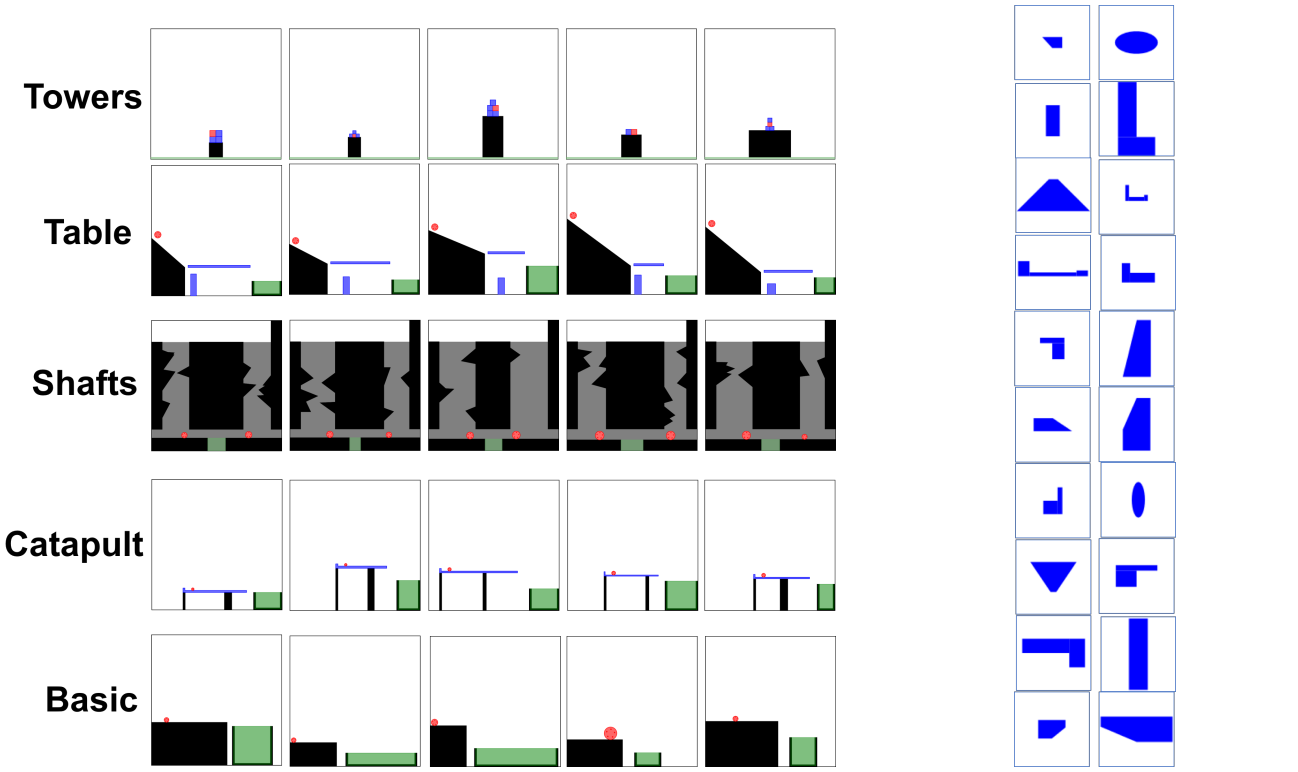
A. Random level generation. For the purposes of comparing human and model performance in the first experiment, we used 20 hand-designed levels. However, Deep Q Learning requires large amounts of data for training, and though we could allow training by taking extensive actions on the given levels, this 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. S2A 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 degrees, subject to the constraint that they continued to fit in the 90 $\times$ 90 pixel area that each of the original tools fit into. See Fig. S2B 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

*We also considered Proximal Policy Optimization (11), but were unable to train the network to perform above chance levels, even within a single level template; see (12).



(a) Randomly generated level screens

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

Fig. S2. Randomly generated levels for deep learning baselines

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.

B. 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.

- 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 (10).
- 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 (13).

C. Training and results. 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. S3A). 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.

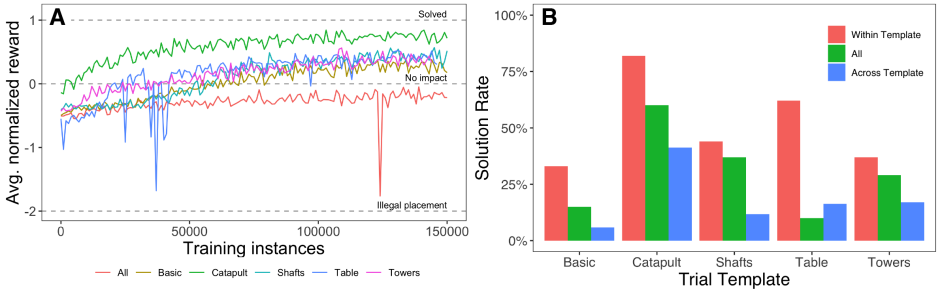


Fig. S3. (A) Training curves for DQN training on individual level templates, or all templates combined (All, red curve). Reward is averaged over each epoch of 1,000 training instances. Models trained on individual scene templates are able to consistently choose successful or promising actions; however, the model trained to generalize across templates under performs, 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.

Learning curves for each model are shown in Figure S3A. 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. S3B). 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. Thus, even when we include an update mechanism to support rapid learning within a trial, the model-free policy learned from extensive experience does not guide the updating appropriately across the variety of levels used in the Virtual Tools game.

D. 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 (14, 15). The Virtual 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 Virtual 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 (16, 17), 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.

6. Additional results

A. Analyzing differences in behavior for matched trials. We used a similar mechanism to evaluate whether there was any difference in actions taken between matched level pairs (A and B). Specifically, for each of level A and B, we use a leave-one-out cross-validation procedure to obtain the probability that each participant's action came from the level they actually played (A or B) as opposed to the alternative matched level (B or A). We fit a Kernel Density Estimate (KDE) plus background guessing to the actions taken by all the people in the opposing level, as well as all actions except the participant's in the true level. We can then obtain likelihood estimates for the participant's actions under both models. This allows us to calculate the confusability score of the participant's action as $p(A_i = \mathrm{A}|A_{-i}, B)$ with A_i being the particular action chosen by a participant playing level A, A_{-i} being all actions except the participant's action, and B being all actions from participants in level B. We obtain the same confusability metric for participants in level B by swapping A and B. Finally, we average the confusability across both variants to calculate how differentiable actions were on a template.

If these scores are greater than 0.5, it implies that the actions are different across the matched levels because we can determine which level caused which action. We find that participants' actions are differentiable in all levels except for 'Towers' and 'Falling' (see Table S1). In 'Falling', this is because people have a strong prior to drop objects from above, so blocking placements below the cup has little effect (see the main paper results for further discussion). In 'Towers', people do not seem to initially understand that they must hit the red block from the left in 'Towers B' and so in both settings they just try to disturb the pile of blocks.

We can perform a similar analysis for the SSUP and DQN models (Table S1), treating each model run as a different participant. We find that the SSUP model does take different actions on all of the levels, but the DQN model does not. We compare to the DQN *without* updating, to demonstrate that the learned policy does not notice these subtle differences in the scenes.

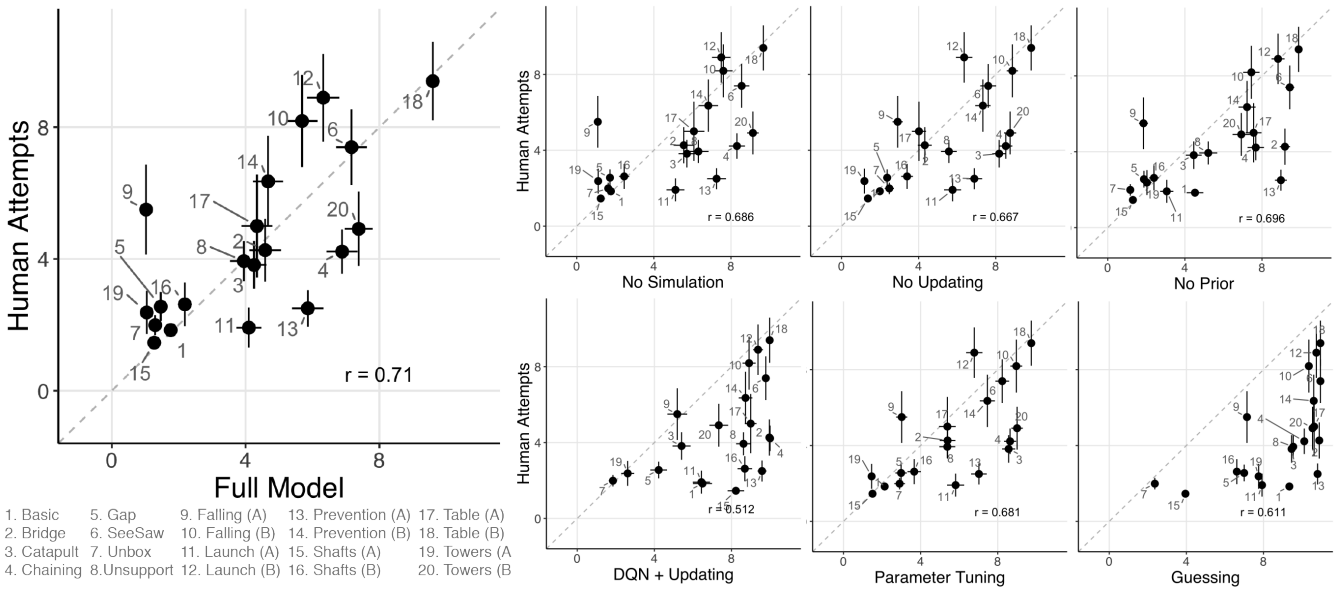


Fig. S4. Comparison of average number of human participants' attempts for each level with average number of attempts for the full model (*left*) and six alternate models. Bars indicate 95% confidence intervals on estimates of the means. The number of placements was capped at 10 for all models. If a model took more than 10 attempts on a particular level, it is considered unsolved. Model results are combined over 250 runs.

B. Comparisons across models. Figure S4 shows the correlations between alternate models and human performance for the average number of placements across levels from the main experiment. Figure S5 shows correlations between alternate models and humans for both accuracy and placements in the validation experiment.

Table S2 shows the RMSE comparing each of the models with human data from the main experiment, and Table S3 shows the same information for the validation experiment.

Table S4 shows the ABCS scores for each level individually. By examining which levels the varying alternate models perform well in, it is clear what the relative contributions of the prior, simulator and update mechanism are. For example, in Catapult, where updating an action to move along a gradient is useful, the "No Updating" ablation performs worst of all methods. Similarly for Bridge, where the prior focuses the search particularly well but a structured search is less necessary (since almost all actions with a positive outcome lead to solutions), removing the object-oriented prior is most problematic for fitting human data. These results provide intuition for why the different model components are important for explaining rapid trial-and-error learning.

Although all alternate models have higher ABCS scores than the full SSUP model, we can test whether these are statistically

Table S1. Confusability metric between matched level pairs. Brackets indicate bootstrapped 95% confidence intervals on the estimate.

Trial	Humans	SSUP	DQN+Updating
Shafts	0.72 [0.66, 0.78]	0.77 [0.73, 0.83]	0.50 [0.47, 0.52]
Prevention	0.54 [0.51, 0.58]	0.53 [0.50, 0.57]	0.51 [0.49, 0.54]
Launch	0.61 [0.56, 0.68]	0.60 [0.56, 0.63]	0.51 [0.49, 0.54]
Falling	0.48 [0.42, 0.54]	0.92 [0.89, 0.95]	0.51 [0.49, 0.54]
Table	0.56 [0.51, 0.61]	0.74 [0.69, 0.79]	0.49 [0.47, 0.52]
Towers	0.47 [0.41, 0.52]	0.64 [0.60, 0.68]	0.48 [0.46, 0.51]

Table S2. Comparisons between people and all models on the original experiment. Brackets indicate bootstrapped 95% confidence intervals on the estimate. 'Average ABCS' refers to the average of the *Area Between Cumulative Solution* curves of participants and the model. 'Guessing' model placements and accuracy can be calculated exactly and therefore have no confidence intervals.

Model	Avg. Attempts	Attempt RMSE	Accuracy	Accuracy RMSE	Average 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]	0.111 [0.103, 0.132]
No Updating	5.38 [5.32, 5.46]	2.31 [2.16, 2.59]	0.69 [0.68, 0.7]	0.271 [0.248, 0.299]	0.189 [0.173, 0.208]
No Simulation	5.23 [5.14, 5.31]	2.29 [2.12, 2.58]	0.59 [0.58, 0.6]	0.328 [0.307, 0.353]	0.218 [0.204, 0.237]
No Prior	5.55 [5.48, 5.62]	2.45 [2.29, 2.71]	0.61 [0.6, 0.62]	0.305 [0.288, 0.33]	0.202 [0.187, 0.222]
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]	0.397 [0.381, 0.416]
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]	0.194 [0.180, 0.214]
Guessing	8.88	4.91 [4.75, 5.1]	0.32	0.552 [0.531, 0.573]	0.402 [0.385, 0.417]

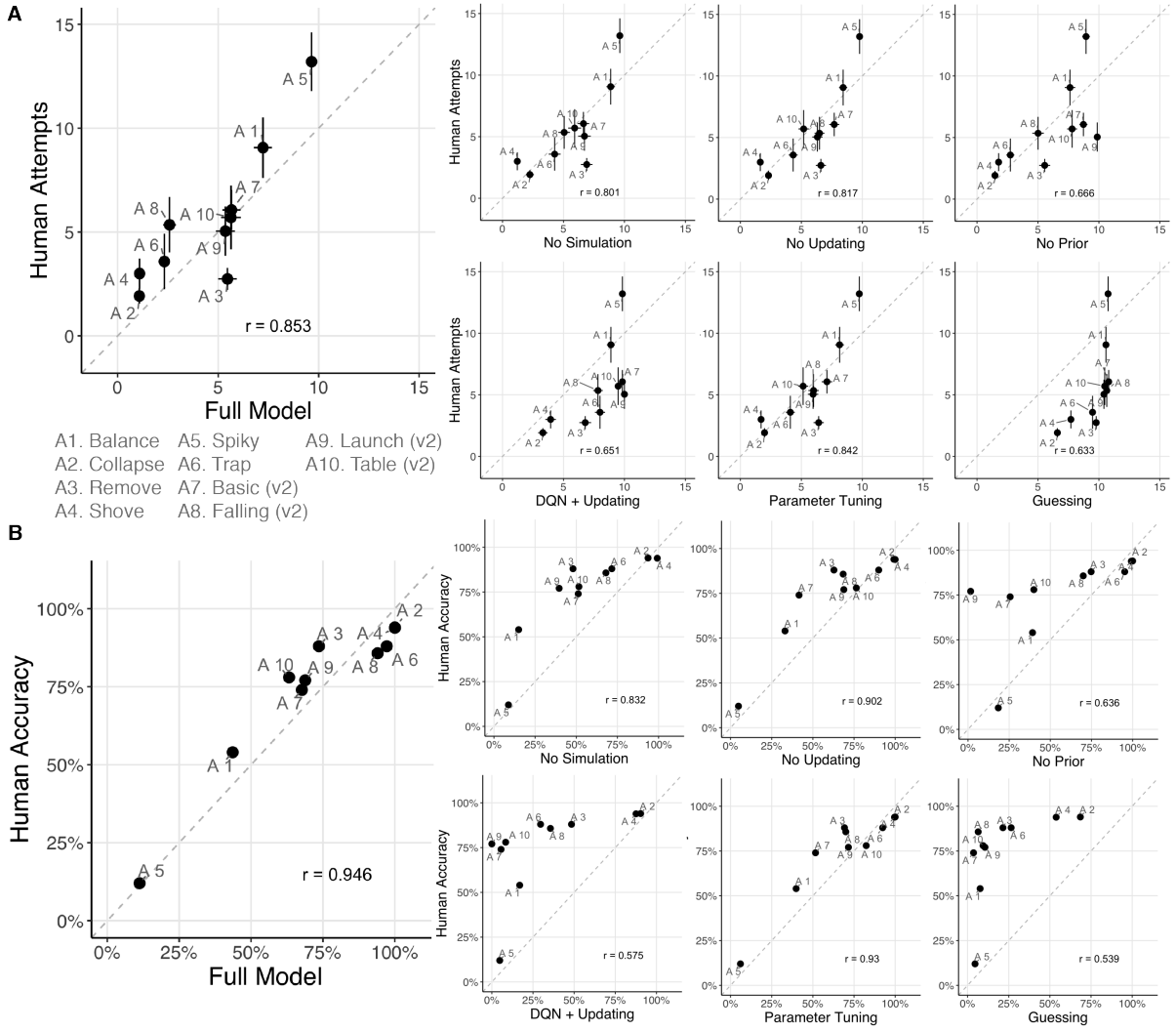


Fig. S5. Comparisons of human vs. all model performance for the validation experiment. **(A)** Human vs. model placements per trial. **(B)** Human vs. model trial accuracy. Bars indicate 95% confidence intervals on estimates of the means.

reliable differences. We form a null hypothesis distribution by bootstrapping the difference between the ABCS scores of two runs of the full SSUP model, taking $10,000$ samples. We then test where the differences between the scores of the full and alternate models fall in that distribution. On the main levels, we find that the differences between ABCS scores for all models would be extremely unlikely (the values were more extreme than *all* null hypothesis samples; all $ps < 0.0001$). On the validation levels, we find that the models that remove the prior or simulation engine, or replace the prior with a DQN all have ABCS score far outside the expectations from the null hypothesis distribution (all $ps < 0.0001$), but the ABCS scores of the models without an update mechanism or with parameter tuning in place of the update mechanism fell within the null hypothesis distribution (no updating: $p = 0.055$, parameter tuning: $p = 0.346$). Since the validation levels were not explicitly chosen to

Table S3. Comparisons between people and all models on the validation experiment. Brackets indicate bootstrapped 95% confidence intervals on the estimate. 'Average ABCS' refers to the average of the *Area Between Cumulative Solution* curves of participants and the model. 'Guessing' model placements and accuracy can be calculated exactly and therefore have no confidence intervals.

Model	Avg. Attempts	Attempt RMSE	Accuracy	Accuracy RMSE	Average ABCS
Human	5.57 [5.11, 6.03]	-	0.74	-	-
Full Model	4.61 [4.5, 4.73]	1.93 [1.64, 2.4]	0.72 [0.71, 0.73]	0.0933 [0.0708, 0.145]	0.0936 [0.080, 0.124]
No Updating	5.88 [5.77, 5.98]	1.88 [1.65, 2.33]	0.65 [0.63, 0.66]	0.162 [0.135, 0.209]	0.120 [0.105, 0.152]
No Simulation	5.75 [5.62, 5.88]	1.93 [1.71, 2.36]	0.55 [0.53, 0.56]	0.252 [0.215, 0.295]	0.170 [0.150, 0.200]
No Prior	5.95 [5.84, 6.07]	2.56 [2.25, 3.01]	0.56 [0.55, 0.58]	0.321 [0.284, 0.364]	0.201 [0.178, 0.228]
DQN + Updating	7.79 [7.69, 7.89]	3.31 [2.93, 3.75]	0.33 [0.32, 0.34]	0.494 [0.456, 0.534]	0.344 [0.320, 0.374]
Parameter Tuning	5.63 [5.52, 5.73]	1.76 [1.52, 2.18]	0.68 [0.67, 0.7]	0.121 [0.0953, 0.17]	0.1 [0.086, 0.130]
Guessing	9.72	4.89 [4.62, 5.22]	0.21	0.575 [0.544, 0.61]	0.422 [0.397, 0.451]

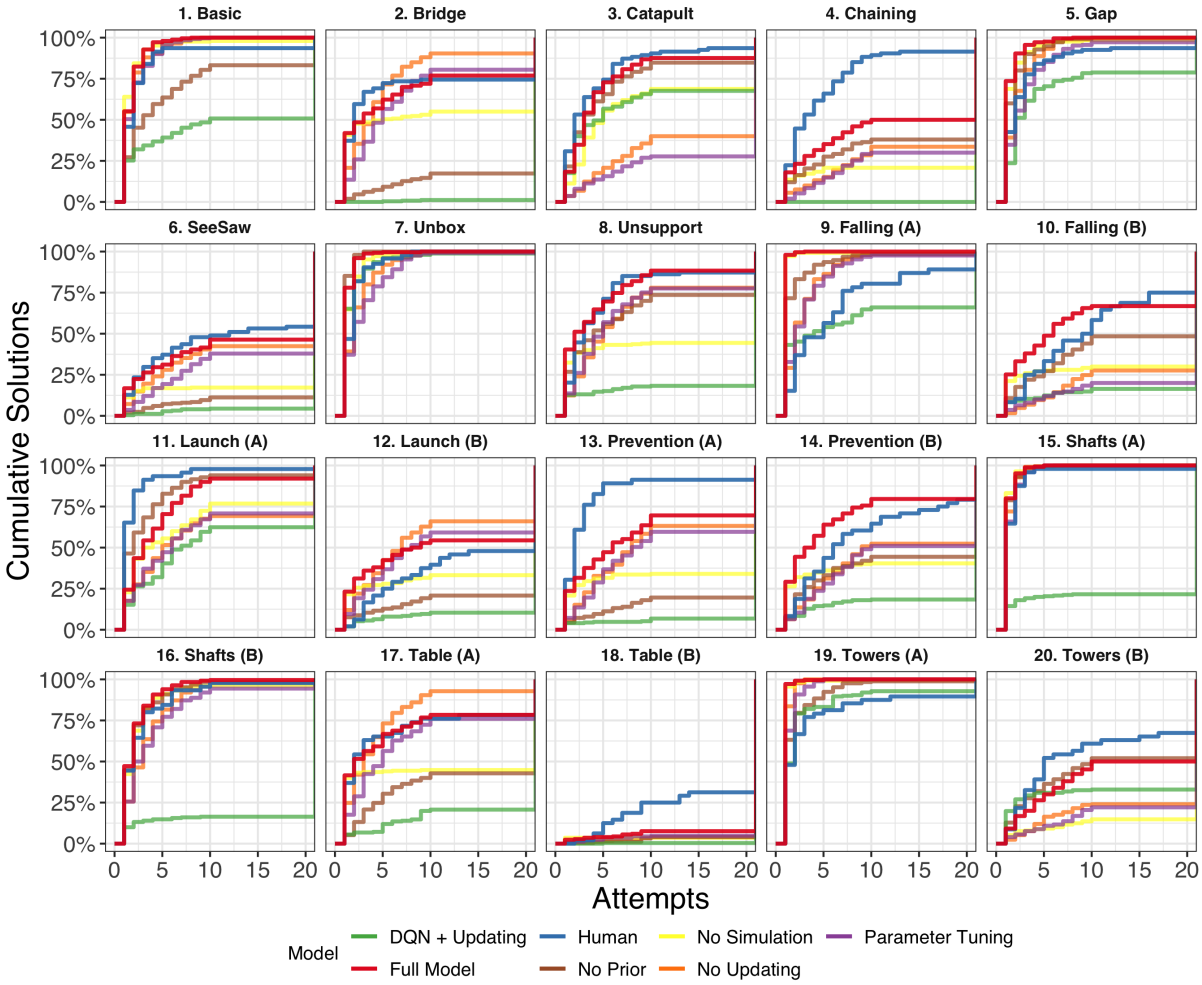


Fig. S6. Cumulative solution rates for all alternate models on the main experiment levels.

differentiate models, only one level, "Basic (v2)", would be expected to benefit from a policy updating mechanism, as hitting the ball from higher is necessary to succeed. Indeed, this is the one level where we do see an improvement from the SSUP model as compared to to the Parameter Tuning alternative.

C. Likelihoods of placements. In addition to testing whether each of the models capture the evolution of human successes and failure over the course of each trial, we can study how well each model captures the specific actions that people take. We measure this as the likelihood of an action under the model's predictions.

However, this measure not perfectly reliable past the first action for two reasons. First, the model and people take actions that depend on the results of prior actions, and therefore the probability density for the second action calculated by the model is conditional on the *model's* prior actions, not *people's*. Second, because of varying solution rates, the reliability of the probability density and the likelihood will vary by trial – for instance, only 2% of model runs on 'Falling (A)' even take a second action, and only a third of participants did not solve 'Shafts (A)' on the first attempt, and these numbers grow smaller with subsequent action orders. We therefore measure this likelihood in two cases, to mirror Fig. S11: the likelihood of the first attempt (where these problems do not exist), and the likelihood of the final solution, to estimate how well the models capture the pattern of final solutions.

Because we only obtain samples from the model, we cannot calculate the likelihood function directly but instead must approximate it via a kernel density estimation (KDE) based around each of the sampled points. This requires two additional parameters to fit: (1) the width of the kernel, and (2) a background "guessing" rate. We fit these parameters for each model to maximize likelihood on the first placement of the basic levels, but use those same fit parameters to calculate the likelihood of placements on the validation levels, and of the last placements on the basic levels.

In addition, we calculate a noise ceiling by asking how well participants' placements can be explained by all other participants' actions on that level. We use the same KDE procedure described above, fit using leave-one-out validation and treating other participants' actions similar to model observations. We also calculated the likelihood under a model that took random actions, so that we could scale these model fits between chance and the noise ceiling.

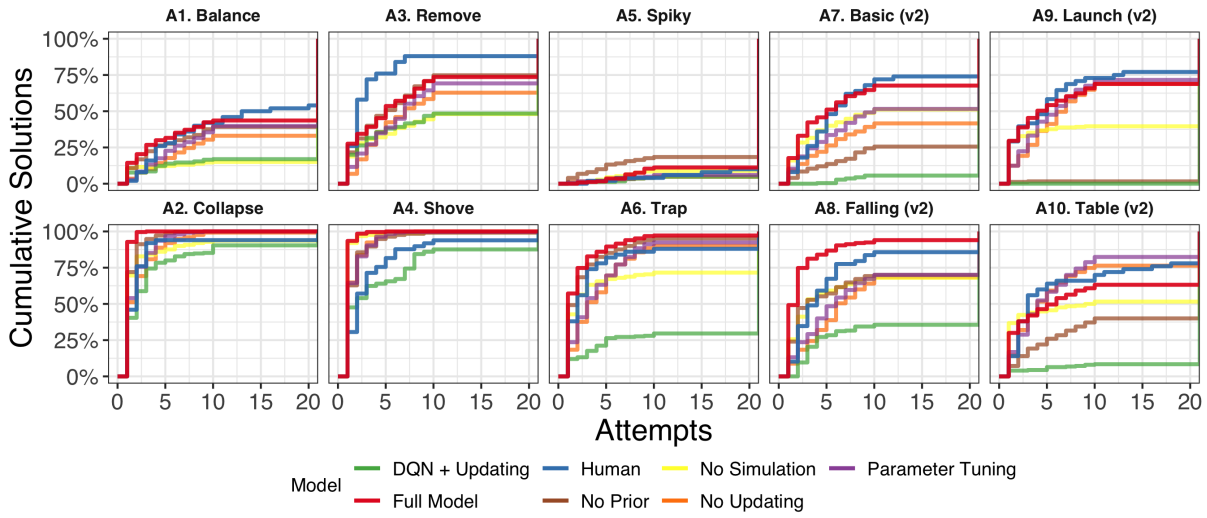


Fig. S7. Cumulative solution rates for all alternate models on the validation levels.

Finally, to test how stable these fits are, we calculated bootstrapped 95% confidence intervals on the likelihoods. We bootstrap by trial to control for particular levels where one model might happen to excel or fall short due to chance sampling of actions, then weight the log-likelihoods by the number of human observations to avoid unbalancing the likelihoods by this scheme.

The results of this analysis can be seen in Fig. S8. We find that this is a noisy measure (due largely in unexplanable variability in participants' actions; see Figs. S11, S12), and so cannot reliably differentiate any of the models' performance, with the exception of the 'DQN + Updating' model which does reliably worse than all the rest. However, we do find that *all* models fall short of the noise ceiling, suggesting that there are additional features of human trial-and-error problem solving that these models are not capturing (see the Discussion for some proposals).

We can see this in more detail by looking at the first actions taken by each alternative model in two representative trials: "Bridge" and "Catapult" (Fig. S9). In "Catapult", the SSUP model and alternatives which include shaping around *reward* correctly narrow down the set of possible actions towards the same set that people consider. Ablations without this reward narrowing perform worse in terms of likelihood of human actions.

In "Bridge", the SSUP model and ablations which are guided by the reward narrow the action space *too* much with respect to the heterogeneity of human actions. The likelihood of "Prior+Physics" and "Parameter tuning" is therefore better in these cases.

D. Analyzing switching behavior. We looked at whether the model was capturing the likelihood of participants "switching" from one strategy to another. In general, it is not obvious what constitutes a particular strategy for participants. Strategies are intentional - just because you did not successfully hit the ball towards the right does not mean that you did not plan to hit the ball such that it moved towards the right. To approximately quantify strategy switching, we therefore consider a "switch" to have occurred if the tool changed, or if the closest object to the performed action changed. Although this does not capture the the full richness of human "strategies," here we use these measures as a useful proxy and leave precise definitions of this notion to future work.

We quantified how often participants and the model switched strategies under this definition. For both tool switches and relative object switches, we find a good correlation between the human participants and the model (see Figure S10).

Finally, we provide additional results showing the first and last placements of the model relative to participants for all levels from the main experiment (Figure S11, as well as for the validation levels, Figure S12). In most cases, these show qualitatively similar initial attempts, and evolutions towards a solution (however, see the Results in the main text for discussion of exceptions).

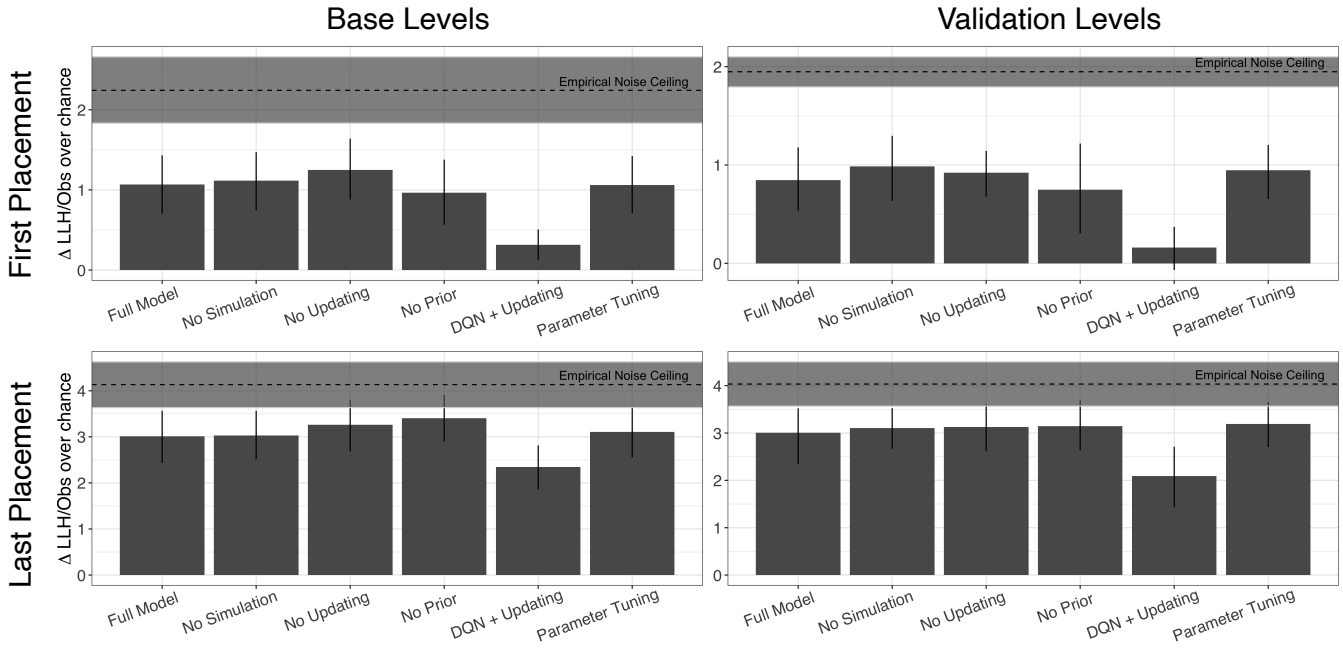


Fig. S8. Improvement in log-likelihoods per observation over chance placements for the first placement of a trial (*top*) and final solution placement (*bottom*), for the 20 regular levels (*left*) and 10 validations levels (*right*). Bars represent 95% bootstrapped CIs. The dashed line represents the maximum noise ceiling calculated as likelihood of placements given all other participants' placements, with the grey area representing the 95% CI on that metric.

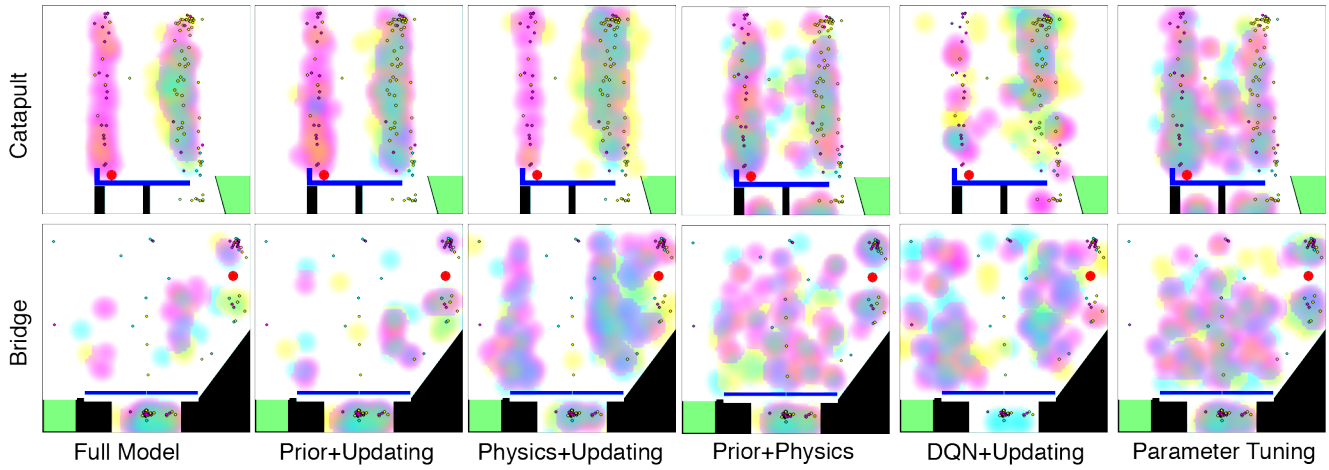


Fig. S9. Comparisons of model predictions across “Catapult” and “Bridge” for the first action. In “Catapult”, the full model has the best likelihood, as it is able to narrow down the set of actions into those that are most promising - like people. In “Bridge”, the “Prior+Physics” and “Parameter Tuning” ablations perform best in terms of likelihood as they have higher coverage for human participants that took actions above the bridge.

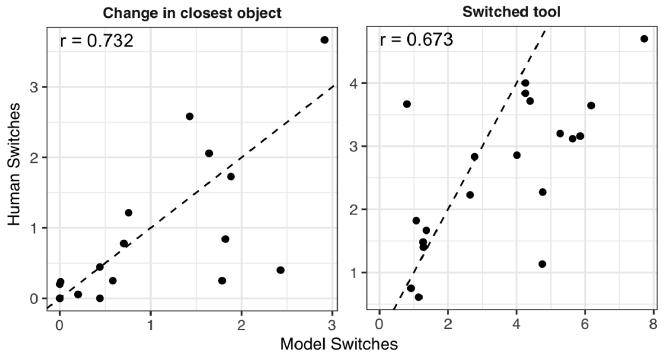


Fig. S10. Comparison of human participants' average number of strategy switches on each trial versus the average number of strategy switches of the full model.

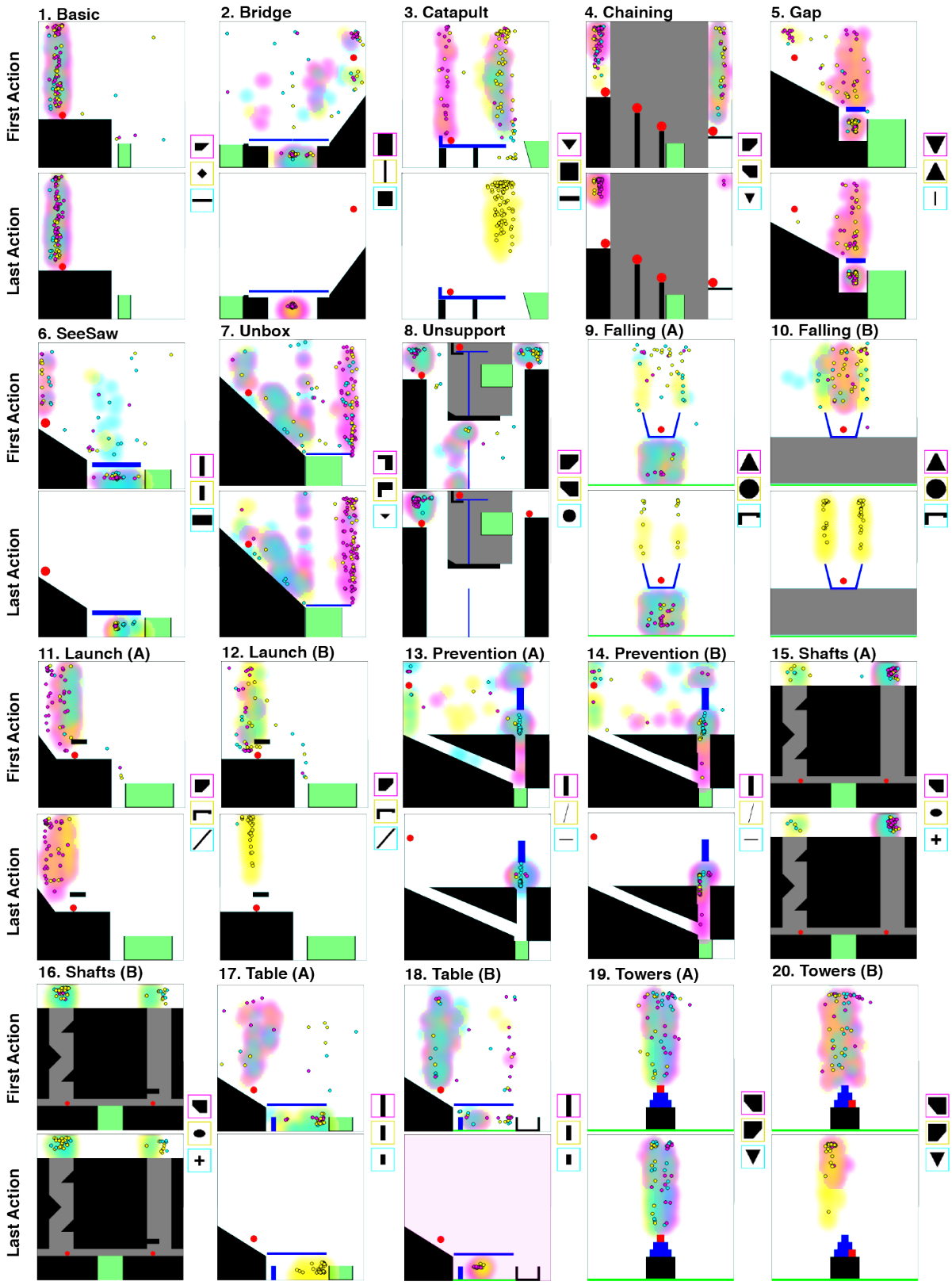


Fig. S11. Distribution of predicted model actions (background) versus human actions (points) on the first attempts of the level (*top*) and the attempt used to solve the level (*bottom*) across all levels from Experiment 1.

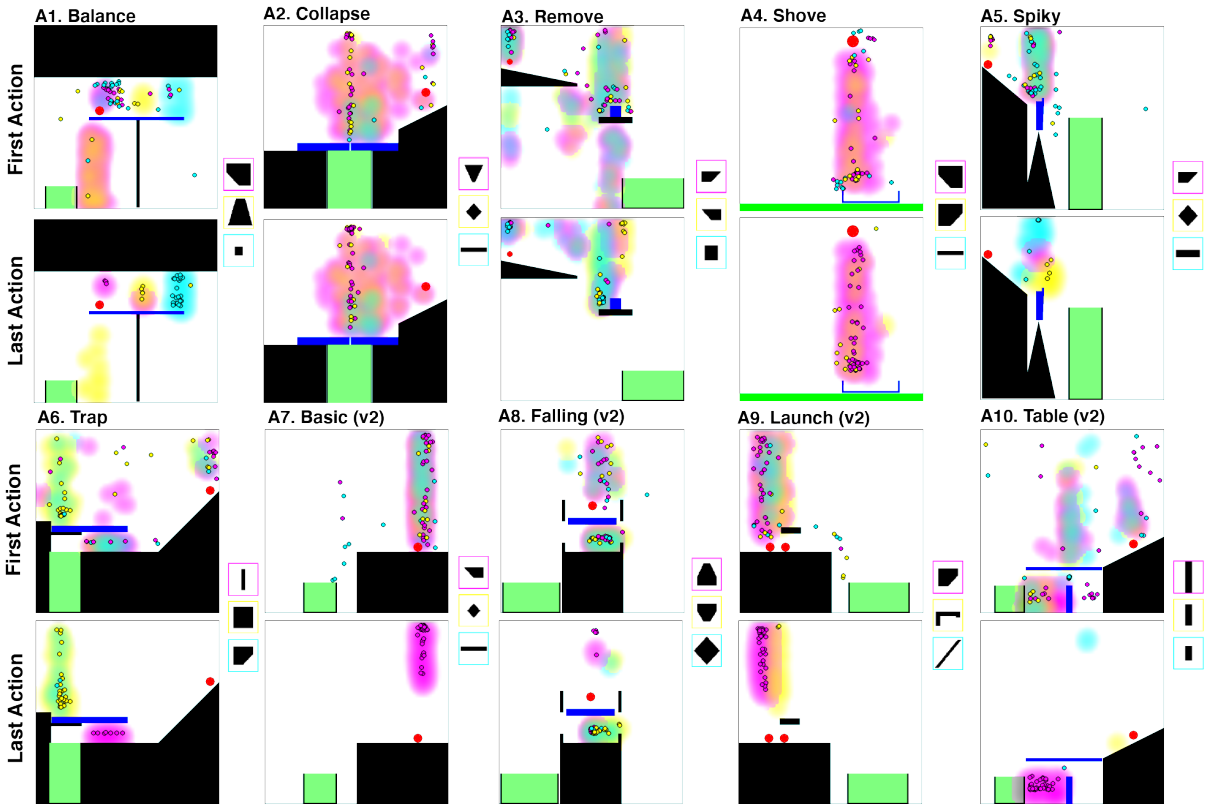


Fig. S12. Distribution of predicted model actions (background) versus human actions (points) on the first attempts of the level (*top*) and the attempt used to solve the level (*bottom*) across all levels from Experiment 2.

	SSUP	No Updating	No Simulator	No Prior	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	0.099	0.053	0.079	0.080	0.143	0.038
SeeSaw	0.052	0.090	0.267	0.341	0.393	0.143
Unbox	0.033	0.025	0.020	0.039	0.019	0.048
Unsupport	0.030	0.117	0.340	0.130	0.576	0.111
Falling (A)	0.261	0.162	0.253	0.213	0.155	0.153
Falling (B)	0.097	0.304	0.258	0.137	0.365	0.349
Launch (A)	0.137	0.335	0.265	0.077	0.403	0.329
Launch (B)	0.121	0.180	0.102	0.182	0.255	0.124
Prevention (A)	0.250	0.332	0.503	0.660	0.756	0.358
Prevention (B)	0.114	0.155	0.226	0.195	0.400	0.169
Shafts (A)	0.029	0.024	0.030	0.027	0.713	0.018
Shafts (B)	0.034	0.035	0.022	0.017	0.712	0.064
Table (A)	0.016	0.123	0.261	0.340	0.524	0.062
Table (B)	0.147	0.173	0.173	0.176	0.198	0.165
Towers (A)	0.154	0.147	0.148	0.100	0.041	0.134
Towers (B)	0.135	0.335	0.402	0.107	0.237	0.358
Average	0.111	0.189	0.218	0.202	0.397	0.194
Balance	0.055	0.124	0.250	0.075	0.234	0.074
Collapse	0.086	0.047	0.032	0.067	0.071	0.050
Remove	0.173	0.282	0.374	0.170	0.367	0.240
Shove	0.136	0.110	0.126	0.099	0.081	0.105
Spiky	0.027	0.018	0.018	0.097	0.018	0.012
Trap	0.095	0.062	0.137	0.065	0.530	0.068
Basic (v2)	0.061	0.242	0.164	0.374	0.531	0.163
Falling (v2)	0.140	0.180	0.139	0.130	0.416	0.151
Launch (v2)	0.059	0.098	0.270	0.619	0.634	0.067
Table (v2)	0.104	0.038	0.189	0.313	0.559	0.072
Average	0.094	0.120	0.170	0.201	0.344	0.100

Table S4. Area Between Cumulative Solutions metrics for each level (row) by model (column). *Top:* 20 levels from the main experiments. *Bottom:* 10 levels from the validation experiment.

References

1. Gureckis TM, et al. (2016) psiTurk: An open-source framework for conducting replicable behavioral experiments online. *Behavior Research Methods* 48(3):829–842.
2. Gentle J (2017) ChipmunkJS.
3. Lembcke S (2013) Chipmunk 2d Physics Engine.
4. Battaglia PW, Hamrick JB, Tenenbaum JB (2013) Simulation as an engine of physical scene understanding. *PNAS* 110(45):18327–18332.
5. Smith KA, Vul E (2013) Sources of Uncertainty in Intuitive Physics. *TopiCS* 5(1):185–199.
6. Hamrick JB, Smith KA, Griffiths TL, 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*.
7. Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.
8. Kersten D, Mamassian P, Yuille A (2004) Object Perception as Bayesian Inference. *Annual Review of Psychology* 55(1):271–304.
9. Cusumano-Towner MF, Saad FA, Lew A, Mansinghka VK (2018) Gen: A general-purpose probabilistic programming system with programmable inference, (Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts), Technical Report MIT-CSAIL-TR-2018-020.
10. Mnih V, et al. (2015) Human-level control through deep reinforcement learning. *Nature* 518(7540):529.
11. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
12. Allen K, Smith K, Tenenbaum J (2019) Rapid trial-and-error learning in physical problem solving. *Structure and Priors in Reinforcement Learning Workshop, ICLR*.
13. Hinton G, Srivastava N, Swersky K (2012) Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.
14. 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*.
15. Cobbe K, Klimov O, Hesse C, Kim T, Schulman J (2018) Quantifying generalization in reinforcement learning. *arXiv preprint arXiv:1812.02341*.
16. Shumaker RW, Walkup KR, Beck BB (2011) *Animal Tool Behavior: The Use and Manufacture of Tools by Animals*. (JHU Press).
17. Beck SR, Apperly IA, Chappell J, Guthrie C, Cutting N (2011) Making tools isn't child's play. *Cognition* 119(2):301–306.