Video Games Require a Metacognitive Module

John Anderson

Video Games Require a Metacognitive Module

- Not really video games but really any task that requires thinking and acting quickly in a high dimensional space.
- We had successful ACT-R models that performed such tasks. The challenge is to **learn** to perform such tasks.
- Despite the recent success of deep reinforcement learning at many classic games, a further challenge is to learn in human time and in a human way.
- The game we have focused on is in the genre that causes significant challenges for deep reinforcement learning techniques.

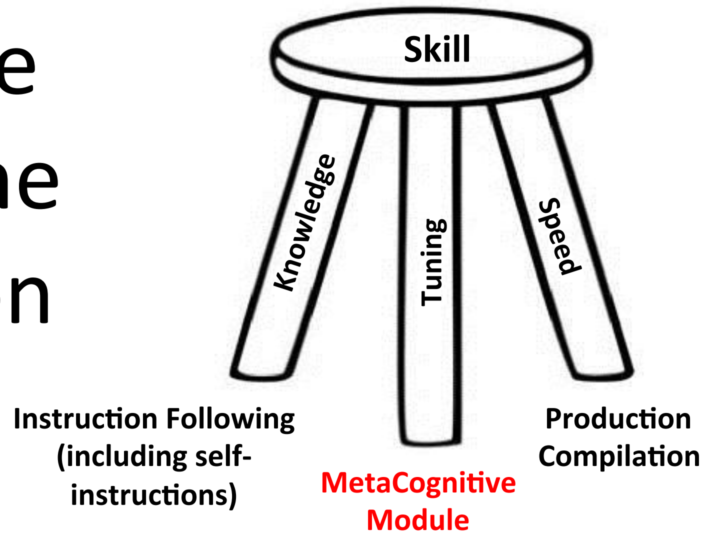
Space Fortress

Not your father's Space Fortress – a fast-paced environment where we can manipulate factors of workload and automation.

Note: 2x Real Speed

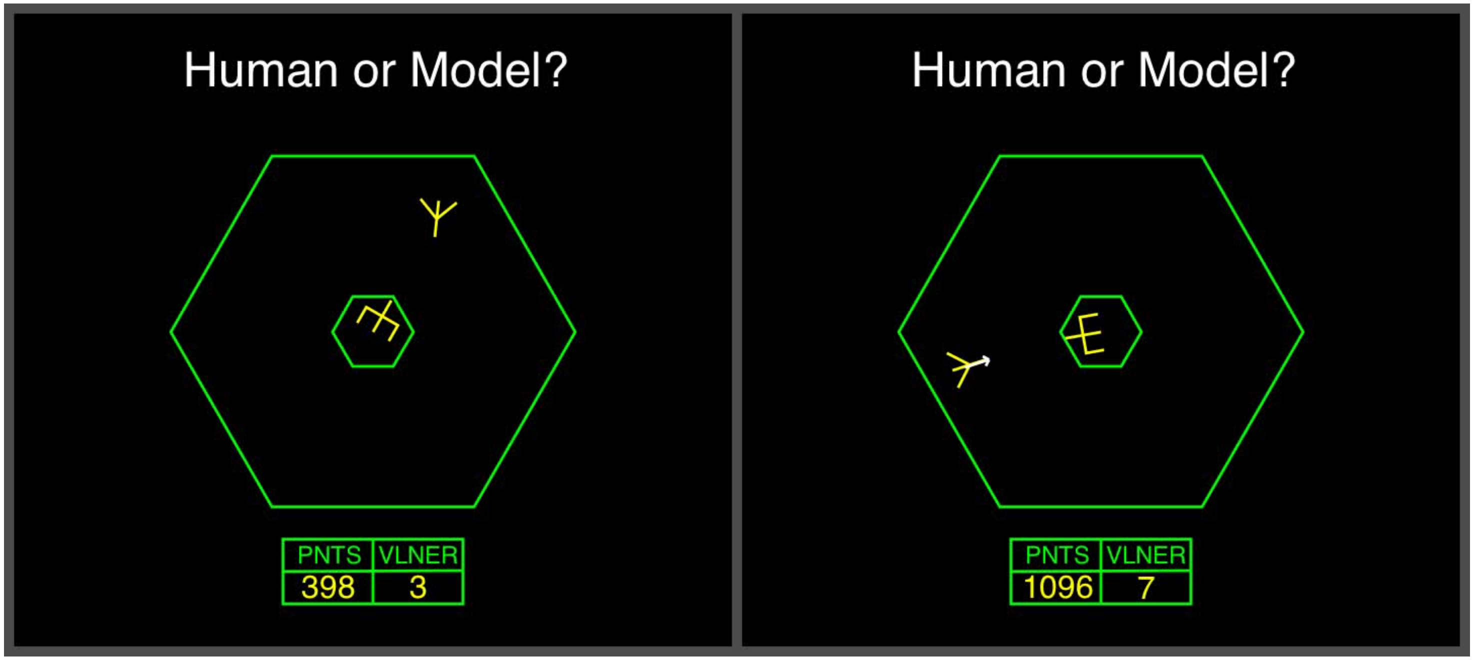
- Navigate a ship between two hexagons, shooting at fortress while avoiding be shot by it.
- A major challenge is flying in a **frictionless** space.
- Our version is challenging but can be mastered to some degree after 20 3-minute games.

Space Fortress is an Example of a Task that Depends on the Three Legs of Skill Acquisition



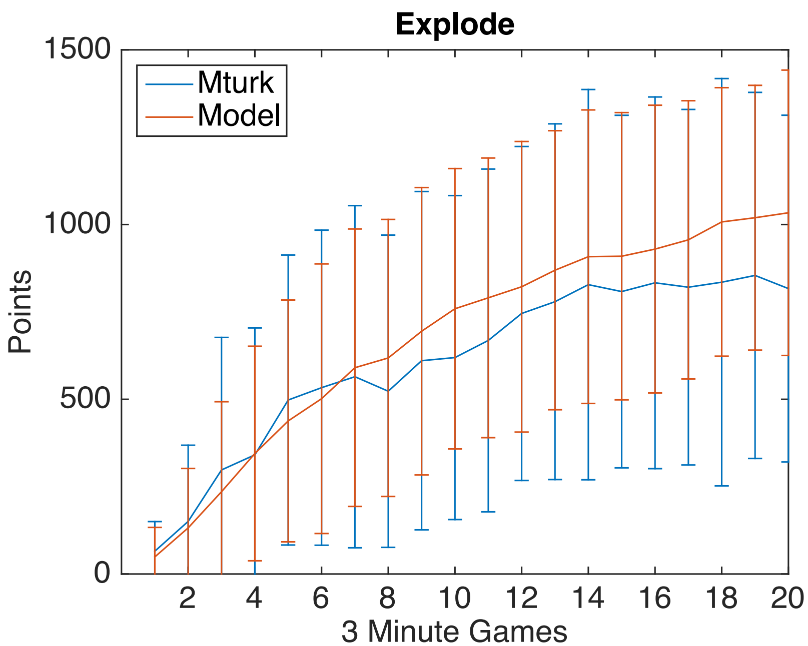
- ➢ The system needs to deploy task knowledge – without this it would take thousands of trials to master what participants master in 20 games.
  - ACT-R instruction following offers a way to model this.
- ➢ Participants start out deploying that knowledge slowly but speed up.
  - ACT-R production compilation offers a way to model this.
- ➢ Successful performance requires learning the parameters that define successful actions (e.g. when to thrust, how to pace shots).
  - In such a fast-paced game, there is not enough time for ACT-R to perform the task and monitor these parameters within its cognitive cycle.
  - Therefore, we are developing a Metacognitive module that can be informed by the cognitive cycle but monitors performance off-cycle.

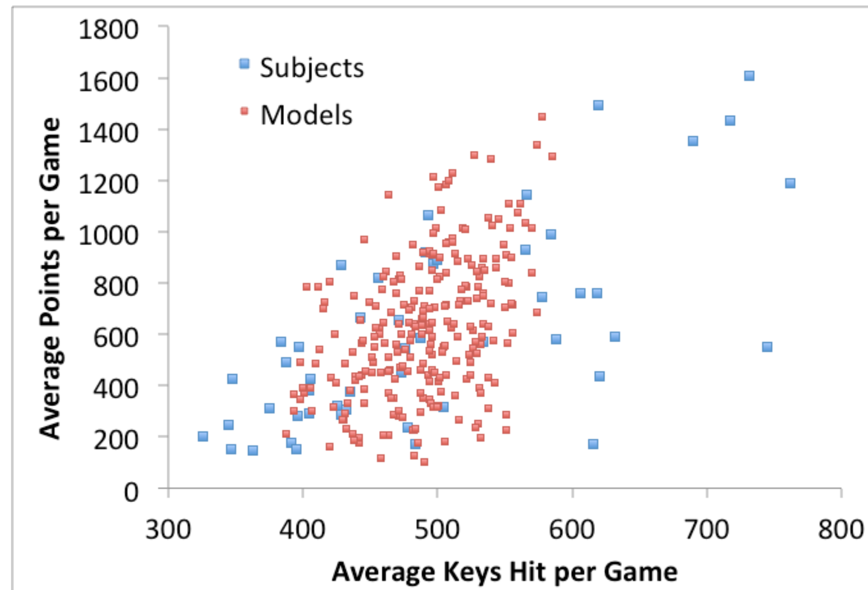
Model or Human



Note: 2x Real Speed

Points





Stategies Present?	Yes	No
Aggressive Motor Programming	734	627
Turn to Control Speed	774	587
Border Rather than Distance	755	606
Thrust when Fortress Absent	683	678

Parameter Controlling Production Compilation

alpha = .05	alpha = .10	alpha = .20	alpha = .40
545	688	736	752

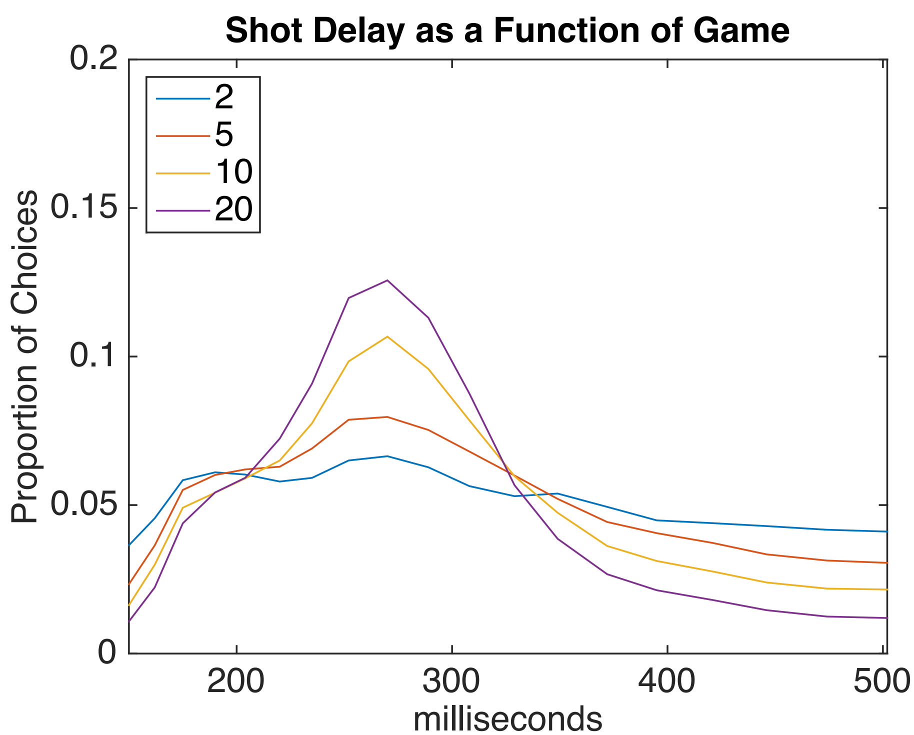
Parameter Controlling Parameter Learning

factor = .01	factor = .02	factor = .04	factor = .08
411	562	769	980

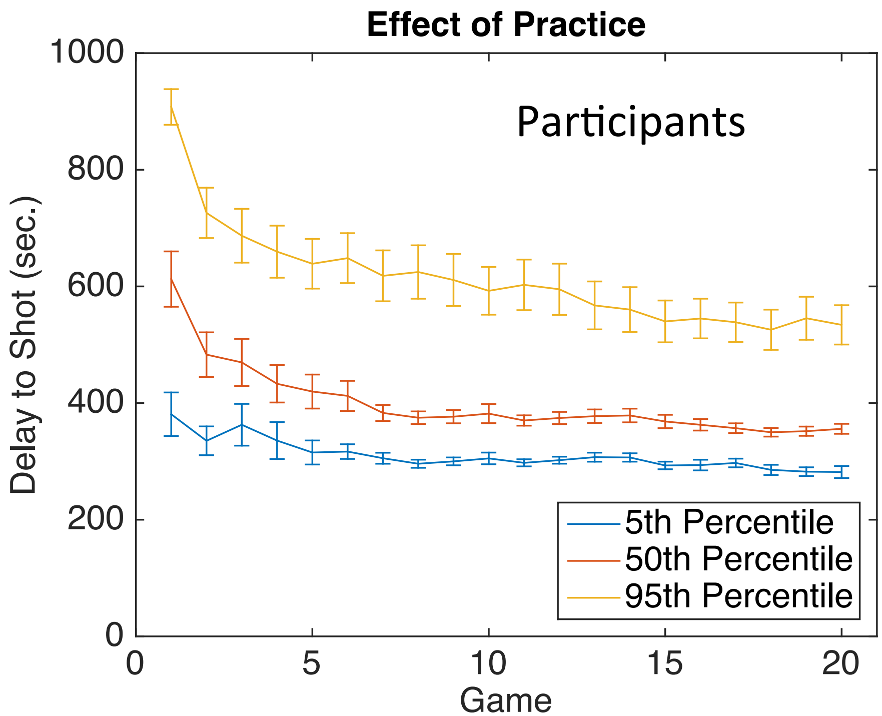
Metacognitive Module: Parameter Learning

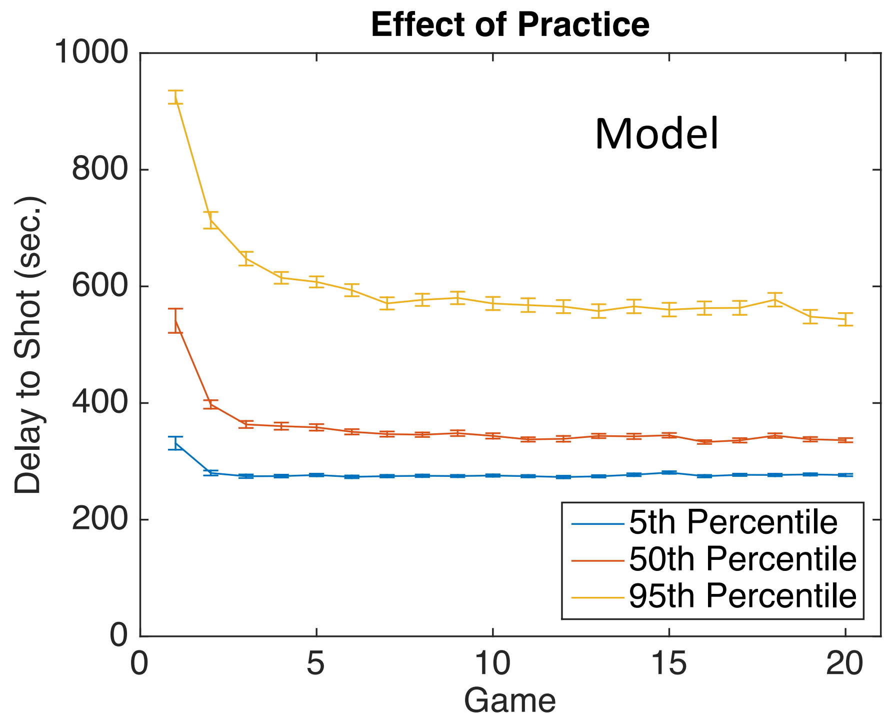
- The Metacognitive module experiments with candidate values for control variables and assesses how well they affect performance.
- Candidate values are sampled according to the current estimate of payoff V(i) of the options according to the softmax rule:
- The temperature T decreases as experience accumulates, producing a shift from exploration to exploitation.

$$P(i) = \frac{e^{V(i)/T}}{\sum_j e^{V(j)/T}}$$

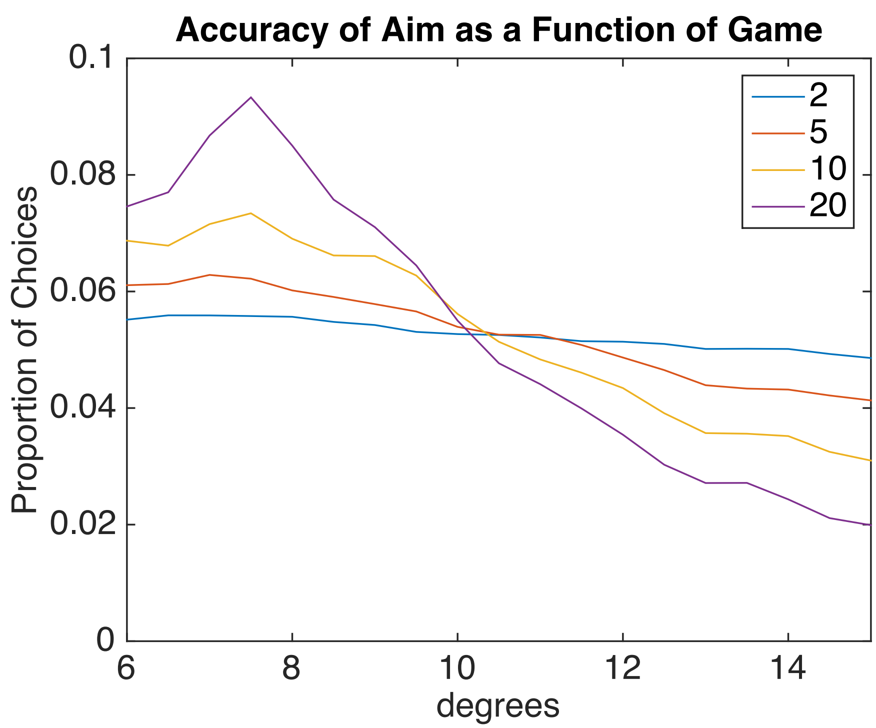


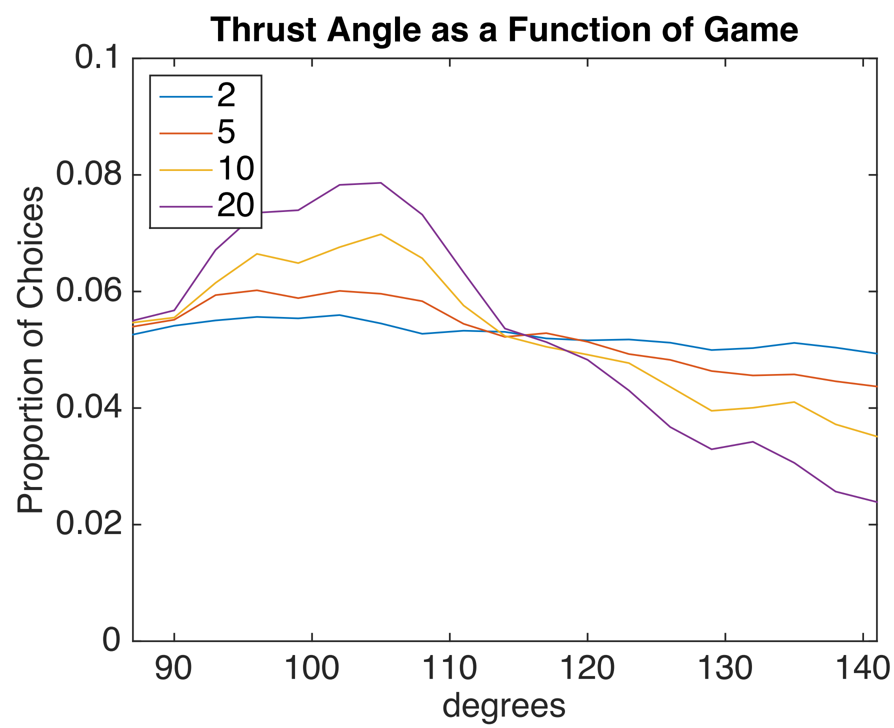
Tuning of Intershot Duration over Trials



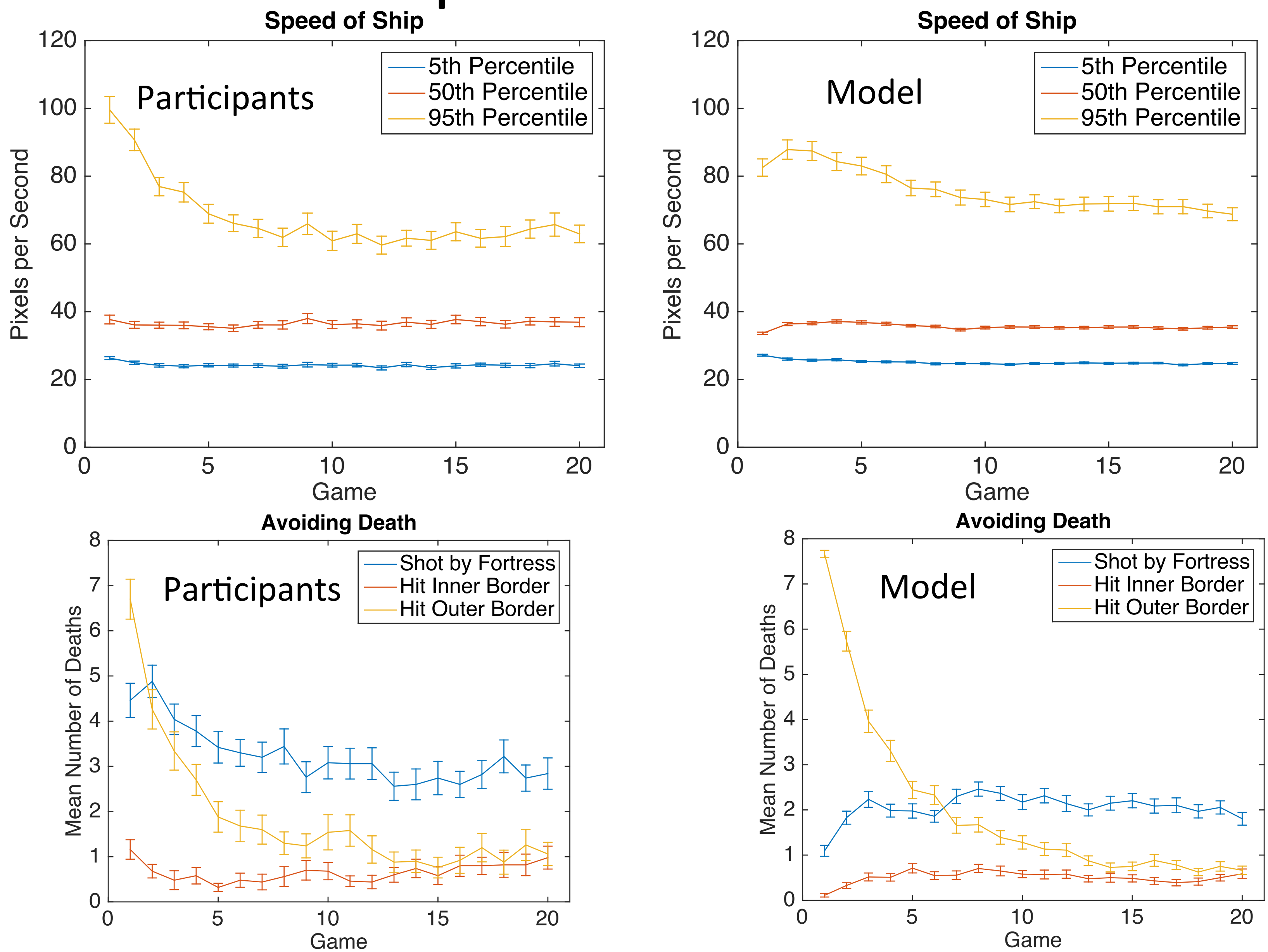


Controlling Aim and Thrust Angle





Speed and Deaths



Conclusions

1. A **single** model can match detailed human learning across a range of dynamic tasks.
2. It was key for ACT-R to have a Metacognitive module to monitoring performance and tune control variables.
3. In a fast-paced task there is not enough time to do this monitoring within the same cognitive cycle that is responsible for task performance.
4. It was key that the Metacognitive module knows what aspects of performance are relevant to what parameters.

The Metacognitive Module in More Detail

- Implemented by Dan Bothell, currently called the tracker module and available.
- Created by a standard module call:

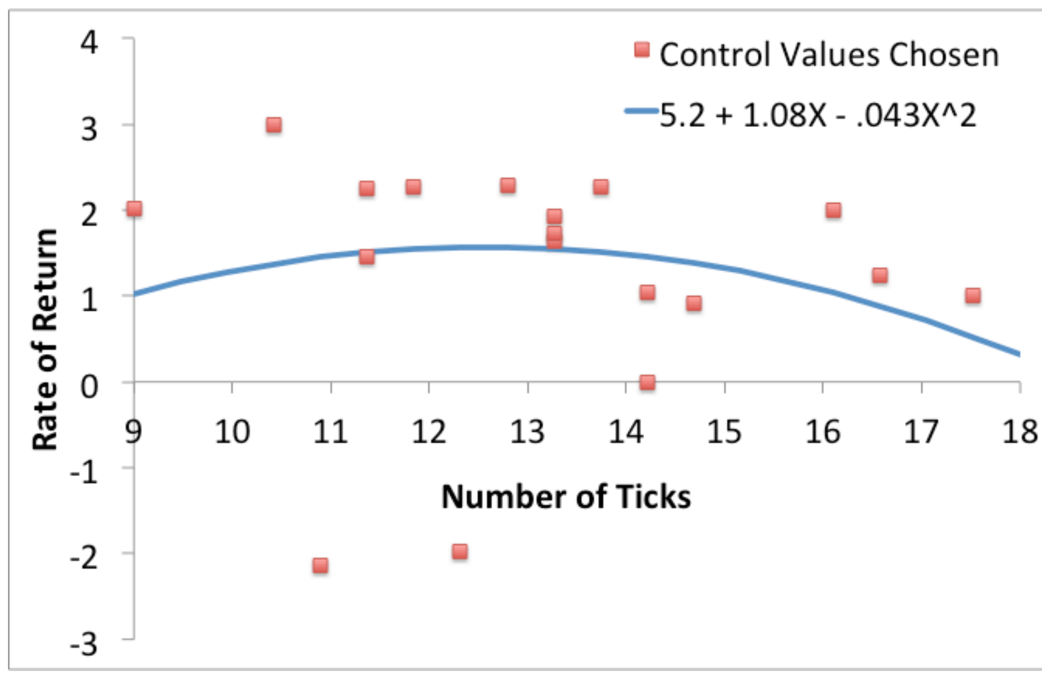
```
 (p start-playing
.....
      ==>
    +tracker>
       control-slot time-thresh
       min 9.0
       max 18.0
       good-slot hit
       bad-slot reset
       bad-weight -10
```

Control variable (# ticks)
Min of range (9 ticks)
Max of range (18 ticks)
Indication of success (inc vuln)
Indication of failure (dec vuln)
Weight of bad relative to good

After a Tracker is Set in Motion

- Randomly starts with a control value from the range and experiments with new ones at random intervals.
- Maintains record of good and bad instances from which it can calculate a mean rate of return for the current setting.
- Estimates from these instances a quadratic function giving rate of return for the range of control settings:

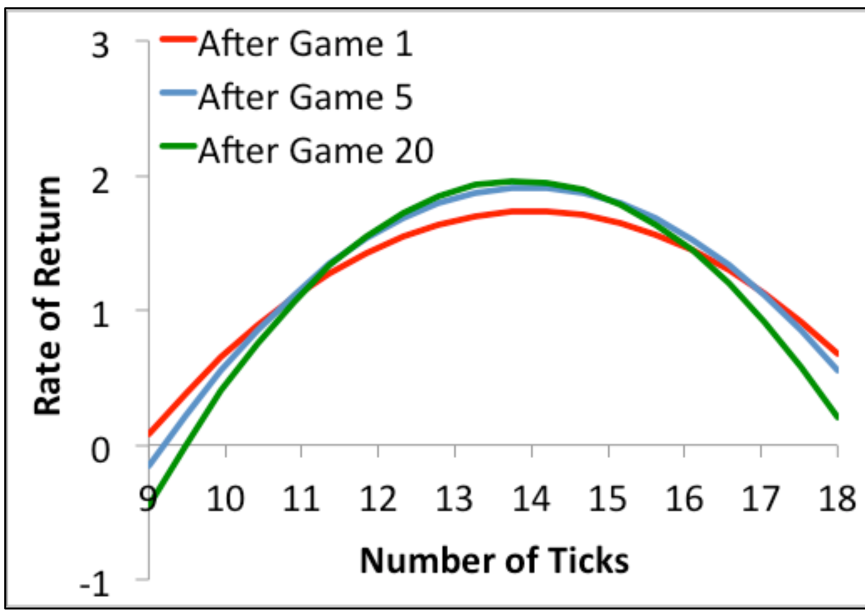
- Function estimated after one 3-minute game.
- Note: Points are weighted by the duration of the interval over which they were sampled.

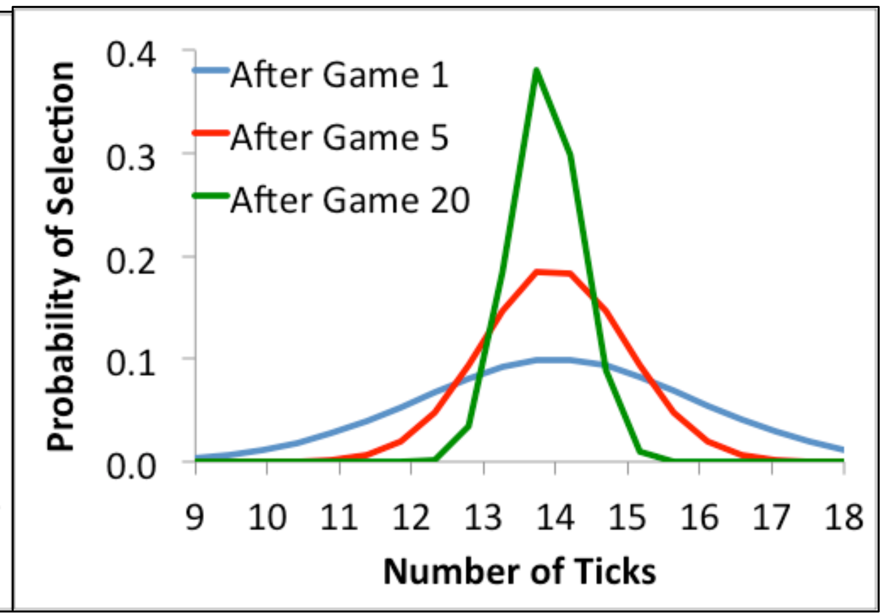


Selecting a New Control Value

- Use a softmax on quadratic function V with a temperature T.
- T=A/(1+B*time) -- A defaults to 1, B defaults to 1/180 s.

$$P(i) = \frac{e^{V(i)/T}}{\sum_j e^{V(j)/T}}$$





Global Parameters for Tracker

:INITIAL-TEMP	default: 1	:initial temperature
:TEMP-SCALE	default: 180	: default scale for decreasing temperature
:PRIOR	default: 20	:weight given to a value of 0 everywhere
:UPDATE-DELAY	default: 10	: mean of update times
:TRACKER-DECAY	default: NIL	: exponent for power-law decay.

Further Points about Metacognitive Tracker

- Can be changed in response to new information such as payoff:

```
+tracker>
      control-slot time-thresh
      bad-weight -1
```

- There is the option to discount past statistics according to exponential or power functions.
- Current quadratic function estimator can be extended to mapping more than 1 dimension onto value – for instance, learning ideal thrust angle for each speed (relation to blending?)
- Unrealistically, there is no limit on the number of trackers that can be simultaneously run.
- Unrealistically, tracker continues estimating when the task is not at hand and would conclude all settings have 0 payoff.