Learning Rapid and Precise Skills[1]

John R. Anderson

Carnegie Mellon University

Shawn Betts

Carnegie Mellon University

Daniel Bothell

Carnegie Mellon University

Ryan Hope

Carnegie Mellon University

Christian Lebiere

Carnegie Mellon University

[1] This work was supported by the Office of Naval Research Grant N00014-15-1-2151. We would like to thank Cvetomir Dimov, Niels Taatgen and Matthew Walsh for their comments on the paper. We also thank Samuel Schneider for his work on the initial design of Space Track. *Some of the results of these experiments were described at the 2018 ACT-R workshop and in a presentation at the 2018 meeting of the Japanese Cognitive Science society. An earlier version of this paper was archived at Anderson, J., Betts, S., Bothell, D., Hope, R. M., & Lebiere, C. (2018, June 4). Three Aspects of Skill Acquisition.* https://doi.org/10.31234/osf.io/rh6zt. The model, data, and video demos are available at http://act-r.psy.cmu.edu/?post_type=publications&p=31435.

Abstract

A theory is presented about how instruction and experience combine to produce human fluency in a complex skill. The theory depends critically on four aspects of the ACT-R architecture. The first is the timing of various modules, particularly motor timing, which results in behavior that closely matches human behavior. The second is the ability to interpret declarative representations of instruction so that they lead to action. The third aspect concerns how practice converts this declarative knowledge into a procedural form so that appropriate actions can be quickly executed. The fourth component, newly added to the architecture, is a Controller module that learns the setting of control variables for actions. The overall theory is implemented in a computational model that is capable of simulating human learning. Its predictions are confirmed in a first experiment involving two games derived from the experimental video game Space Fortress. The second experiment tests predictions from the Controller module about lack of transfer between video games. Across the two experiments a single model, with the same parameter settings, is shown to simulate human learning of three video games.

Keywords: skill acquisition, cognitive architecture, computational modeling, learning, video games

The course of our species has been determined by learning skills as varied as hunting with a bow and arrow to driving a car. This paper describes a general theory of the mechanisms that underlie human ability to acquire skills that require rapid and precise actions, which we have applied to learning a video game. Like many interesting skills, learning to play a video game involves simultaneously mastering components at perceptual, motor, and cognitive levels. The level of expertise players can achieve with extensive practice in these games is truly remarkable (e.g., Gray, 2017), but also remarkable is the speed with which people can acquire a serviceable competence at playing games. There are now appearing artificial systems using deep reinforcement learning that can out-perform humans in some video games (e.g., Mnih et al., 2015). Whatever the judgment on the achievements of such systems, so far they have offered little insight into how humans can learn as fast as they can. As examples of the speed of human learning, Tessler et al. (2017) and Tsividis et al. (2017) both report that humans can achieve in a matter of minutes the expertise that these reinforcement-learning algorithms require hundreds of hours of game play to achieve.

An essential feature of the task in this paper is that it requires simultaneously learning many things at many levels. There has been considerable research on perceptual learning such as visual patterns (Gauthier et al., 2010), motor learning (Wolpert, Diedrichsen, & Flanagam, 2011), and complex tasks like learning algebra (Anderson, 2005), but little research that has addressed how these are put together. Our task also stresses the combination of instruction and experience. While one cannot become good at a video game just by reading instructions, upfront instruction certainly speeds up learning (see Appendix B). This combination of learning by instruction and learning by practice is typical of human skill acquisition. As a test of whether our theory has really put the many pieces together, we required that the model learn by playing

the actual game. The model is implemented within the ACT-R cognitive architecture (Anderson et al., 2004). Using ACT-R avoids the irrelevant-specification problem (Newell, 1990) – that is, the need to make a large number of under-constrained design decisions to allow the simulation to run. This is because these details have been developed and verified in other empirical studies. Of particular importance, ACT-R has a model of the speed and accuracy of perception and action and so prevents us from implementing a system with super-human response speeds – as is the case in the typical reinforcement-learning approach to video games. While existing ACT-R theory provides many components for understanding the learning of these skills, it did lack an essential component for mastering rapid and precise skills. This led us to develop a new **Controller module** for ACT-R.

Theoretical discussions of skill acquisition (e.g., Ackerman, 1988; Anderson, 1982; Kim et al., 2013; Rasmussen, 1986; Rosenbaum et al., 2001; VanLehn, 1986) frequently refer to the original characterization by Fitts (1964, Fitts & Posner, 1967) of skill learning as involving a Cognitive, Associative, and an Autonomous phase. In the Cognitive phase the learner develops a declarative encoding of the skill. Performance is typically halting and error prone. In the Associative phase errors in the initial understanding are gradually detected and eliminated and performance speeds up. In the Autonomous phase performance speeds up further and achieves greater and greater degrees of automacity.

While Fitts' original theory was focused on motor skills, Anderson (1982) extended it to cognitive skills in terms of the then current ACT* theory. Anderson (1982) interpreted Fitts' phases as successive periods in the course of skill acquisition. Fitts' Cognitive Phase corresponded in ACT* to the acquisition and utilization of declarative knowledge about the task domain. While ACT* focused on acquisition and use of problem-solving operators, more recent

ACT-R models have also used declarative knowledge in analogical processes (particularly use of examples) and instruction following (e.g., Anderson, 2005; Taatgen et al., 2008; Taatgen & Anderson, 2002). This interpretive use of declarative knowledge is gradually converted into procedural knowledge by a process called production compilation (both in ACT* and ACT-R). This results in **action rules** that directly perform the task without retrieval of declarative knowledge for guidance. This both speeds up performance and frees retrieval processes to access other knowledge. Fitts' Associative phase was interpreted as the period over which compilation was occurring. ACT* accounted for learning in Fitts Autonomous phase in terms of production-rule tuning, a process by which a search is carried out over the selection criteria of productions to find the most successful combinations. Production tuning was heavily symbolic and did not apply to learning over continuous dimensions in the skills of concern to Fitts and Posner or in video games. Production tuning was not carried forward in the modern ACT-R theory because it did generalize in a workable way[2] to the wide range of tasks that have occupied ACT-R. Thus, current ACT-R does not have a characterization of how a skill becomes tuned to features that are predictive of success. This gap is filled by the new Controller module.

The Controller module has similarities to theoretical proposals in the motor learning literature on how movements like reaching for a target are learned (Wolpert, Ghahramani, & Jordan, 1995). This research has emphasized the development of an internal model mapping controllable properties of the movement (forces produced by the muscles) to features of the movement (e.g., position, velocity, and acceleration). This internal model enables selection of a range of desired movements not just the trained movements. While our video game (controlled

[2] That is to say, it produced a lot of broken or malfunctioning models.

by discrete key presses) is not a typical motor learning task, it requires learning a model of how parameters of one's game play affect significant outcomes in the game.

A challenge in learning appropriate behavior in complex environments is that the same action choice can have different success because of variability in the environment and variability in the learner's behavior. A further complication is that the environment may not be stable and payoffs might change over time (Wilson & Niv, 2011). A significant source of non-stability, relevant in video games, is that as one's skill changes different choices may become optimal. The best approach to these issues in the reinforcement literature (e.g., for domains like n-arm bandit problems, e.g., Cohen et al., 2007, Lee et al., 2011, Zhang & Yu, 2013) involves continuously updating estimates of the payoffs of the options, choosing them with probabilities that reflect their payoffs, and becoming more deterministic with experience. Not only does it lead to adaptive behavior, it can be a good characterization of human learning (Daw et al., 2006).

Space Fortress Video Game

The Space Fortress video game has had a long history in the study of skill acquisition, first used in the late 1980's by a wide consortium of researchers (e.g., Donchin, 1989; Frederiksen & White, 1989; Gopher et al., 1989). Figure 1a illustrates the critical elements of the game. Players are supposed to try to keep a ship flying between the two hexagons. They are shooting "missiles" at a fortress in the middle, while trying to avoid being shot by "shells" from the fortress. The ship is

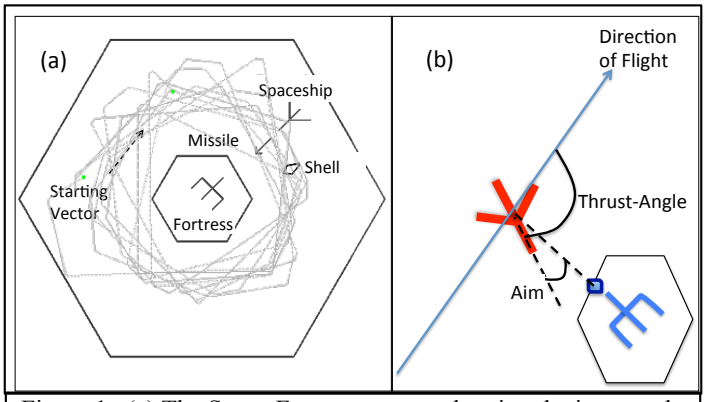


Figure 1. (a) The Space Fortress screen, showing the inner and outer hexagon, a missile shot at the fortress, and a shell shot at the ship. The dotted lines illustrate an example path during one game. (b) A schematic representation of critical values for shooting and flight control.

flying in a frictionless space in which to navigate the player must combine thrusts in various directions to achieve a path around the fortress. Mastering navigation in the Space Fortress environment is challenging. While our subjects are overwhelmingly video game players, most have no experience in navigating in a frictionless environment.

Details about our version of Space Fortress are available in Appendix A. Here we describe what is necessary to understand the results. We used the Pygame version of Space Fortress (Destefano & Gray, 2008) where subjects interact by pressing keys on a keyboard. In keeping with a finger mapping used in many video games, this implementation maps the keys A, W, D onto counterclockwise turn, thrust, and clockwise turn, respectively. Subjects pressed the space bar to shoot. The original version of Space Fortress is more complex than our version and requires tens of hours to achieve mastery. To focus on learning in a shorter time period we created a simpler version that just required flying the ship and destroying the fortress. The ship starts out at the beginning of the game aimed at the fortress, at the position of the starting vector in Figure 1a, and flying at a moderate speed (30 pixels per second) in the direction of the vector. To avoid having their ship destroyed, subjects must avoid hitting either the inner or outer hexagons and keep their ship flying fast enough to prevent the fortress getting an aim on it and shooting it. When subjects are successful the ship goes around the fortress in a clockwise direction. If their ship is destroyed it respawns after a second in the starting position flying along the starting vector. Our version of the game eliminated much of complexity of scoring in the original game and just kept three rules:

1. Subjects gained 100 points every time they destroyed the fortress.
2. Subjects lost 100 points every time they died.
3. To reinforce accurate shooting, every shot cost 2 points.

To keep subjects from being discouraged early on, their score never went negative.

Good performance involved mastering two skills – destroying the fortress and flying the ship in the frictionless environment. To destroy the fortress one must build up the vulnerability of the fortress (displayed at the bottom of the screen). When the vulnerability reaches 10, subjects can destroy the fortress with a quick double shot. Each time a shot hits the fortress the vulnerability increments by one, provided the shots are paced at least 250 ms apart. If the inter-shot interval is less than 250 ms the vulnerability is reset to 0 and one must begin the build up of vulnerability anew. In contrast to the shots that build up the vulnerability, the double shot to destroy the fortress must be a pair of shots less than 250 ms apart. While subjects could easily make sure the shots building up vulnerability are at least 250 ms apart by putting long pauses between them, this would mean that they would destroy few fortresses and gain few points. Thus, they are motivated to pace the shots as close to 250 ms as they can without going less than 250 ms and producing a reset. The other aspect of shooting at the fortress is aiming (see Figure 1b). While shots need to be aimed roughly at the fortress, they do not need to be aimed at the dead center because the fortress has a width. Not to waste time on needless aiming but not to miss, subjects need to learn the boundary that defines an adequate aim.

The goal in flying is to maintain a trajectory that requires the fewest navigation key presses (enabling the most shot key presses), that avoids being killed, and that optimizes one's ability to aim at the fortress. Subjects are instructed that they can achieve this by flying in an approximate circular orbit (created by a sequence of legs producing a polygon-like path – see Figure 1a). This can be achieved by applying thrust when they start to get too close to the outer hexagon. They can only maintain a successful flight path if they control the speed of their ship. If the ship moves too slowly the fortress can shoot it and destroy it. If they start moving too fast

they will not be able to make navigation actions in time. They can change the speed of the ship by the direction the ship is pointed in when they thrust (thrust angle in Figure 1b). The ship can be sped up by changing its thrust angle so that it points more in the direction it is currently going in, while the ship can be slowed down by aiming more in the opposing direction. Exactly how current speed, current direction, and orientation of the ship determine future speed and direction is something that subjects must learn from experience. Also, the amount of thrust is determined by the length of time they hold down the key and they need to learn appropriate durations for the thrust.

Experiment 1

Thanks to the Pygame implementation of Space Fortress it is easy to create experimental variations on the basic game and both experiments in this paper compare subject performance in two games. The first experiment used a version, called AutoTurn, which simultaneously reduced the demands of aiming at the fortress and of maintaining a successful flight path. In this version the ship was automatically turned to always aim at the fortress. One can control the path and speed of the ship by timing thrusts of appropriate durations. These two games created the two conditions in Experiment 1, YouTurn and AutoTurn.

While these versions were piloted with CMU student subjects, the data come from Mechanical Turk subjects. The performances of the two populations were similar but the Mechanical Turk subjects performed slightly better, probably because the Mechanical Turk subjects who selected this hit were almost all video gamers. Only one subject reported no video game experience, although no one reported experience with flying in a frictionless environment. A minority reported having played Asteroids, which has some similarity to Space Fortress. No subject mentioned Star Castle, which was the arcade game that Space Fortress was based on.

The subjects played 20 3-minute games. Some subjects failed to show learning over the course of the game and some of these might not have been trying. We regressed subjects' scores on game number and excluded subjects who did not show a positive slope with significance of at least .1. This mainly eliminated poor performers, including a number who scored 0 points for all games (Appendix B reports average results for all subjects). There were 65[3] subjects in YouTurn (19-55 years, mean 31.9 years, 29 female) of which 48 met the learning criterion for inclusion (19-55 years, mean 32.5 years, 22 female, 1 non video game player, 6 who had played Asteroids). YouTurn subjects were paid a base pay of $7.50 for completing the experiment and .02 cents per points (average bonus $2.11). There were 52 subjects in AutoTurn (19-41 years, mean 31.7 years, 18 female) of which 42 met the learning criterion for inclusion (24-41 years, mean 32.5 years, 15 female, all had played video games, 10 who had played Asteroids). AutoTurn subjects were paid a base pay of $4 for completing the experiment and .02 cents per points (average bonus $7.88).

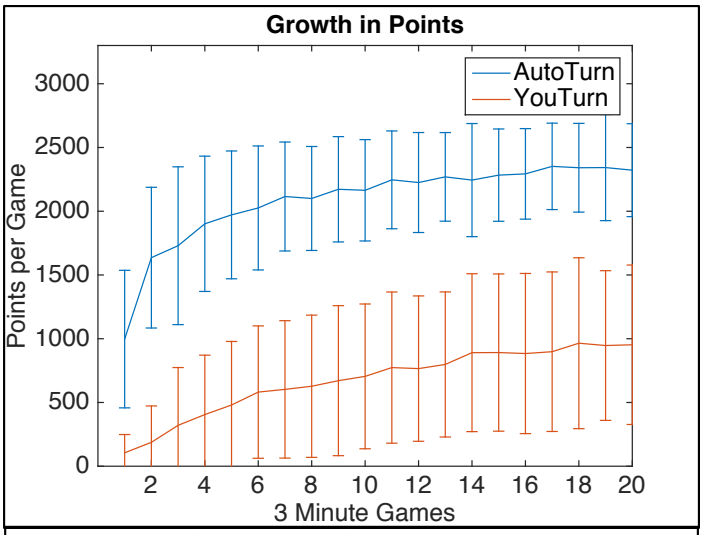


Figure 2. Growth in mean number of points earned per game. Error bars reflect one standard deviation of the population.

Figure 2 shows the average performance of subjects in the two versions of the game. To stress the magnitude of individual differences, the error bars denote a standard deviation of the population (not of the mean). Both populations show substantial improvement over the course

[3] There were 26 subjects who selected the Mechanical Turk hit in YouTurn but did not start the experiment (dropping out before consent) and 8 subjects who quit before playing the game. In Autoturn these numbers were 12 and 13.

of the 20 games, but throughout the experiment AutoTurn subjects are much better than YouTurn subjects, indicating a major cost associated with having to turn one's ship. Subjects in YouTurn only achieve 53% as many shots at the fortress as AutoTurn subjects and averaged 3.8 times as many deaths. We postpone further discussion of the data until after the description of the model of the task and its theoretical assumptions.

A Model that Plays Space Fortress and Learns

A single ACT-R model played either AutoTurn or YouTurn depending on the instructions that it received. The modeling effort took advantage of many mechanisms that have been developed within the ACT-R architecture and tested in prior research. Figure 3 shows the module architecture of ACT-R as described in Anderson et al. (2004) and Anderson (2007). **Modules** are independent computational systems that run in parallel and asynchronously. They can communicate through limited capacity **buffers** where they can deposit small amounts of information. Critical to the coordination of the overall system is the Procedural module, which can detect patterns in the buffers of other modules and issue commands to the modules to perform actions. It consists of production rules, which reflect pattern-action associations that have been built up through experience. Execution of production rules averages 50 ms.

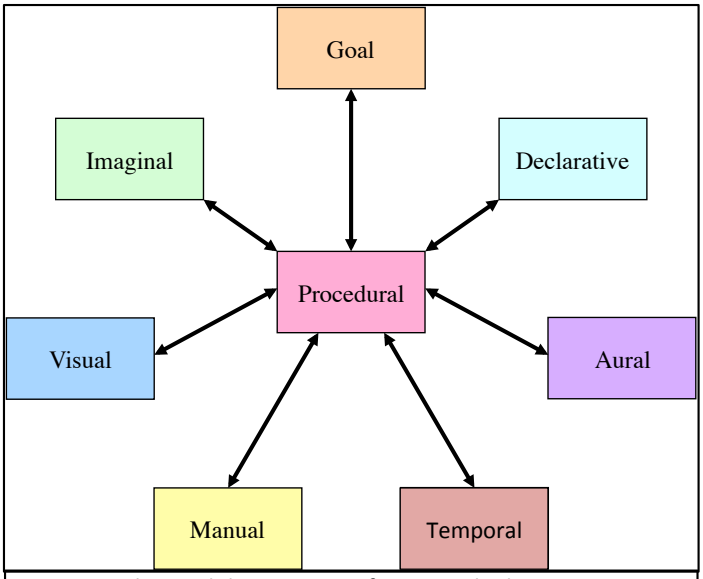


Figure 3. The module structure of ACT-R. The boxes represent high-capacity, independent modules, which process information in a parallel. The central Procedural module can recognize patterns in module buffers and recommend cognitive and motor actions.

A number of other modules are also critical to the performance of the model in this paper. Particularly important in these games are timing of

finger actions. ACT-R has a Manual module based on the EPIC model of Kieras & Meyer (1997). The minimum time between most finger presses will average 250 ms (50 ms production plus motor time) although motor timing varies around this average. The one exception to this is a double tap of the same finger, which averages about 100 ms (50 ms production plus motor time).

Other modules and their buffers also play a critical role in the game: Information about the screen like where the ship is held in the Game State buffer (part of the Visual module); information about various control variables like the time to wait between shots is held in the Goal buffer; information about passage of time is in the Temporal buffer[4]; information about recent outcomes (like a shot having missed) is in the Imaginal buffer; retrieved information about the game instructions is available in the Retrieval buffer. The development of game competence is based on three critical theoretical mechanisms: a system of rules that interprets declarative representations of instructions, the production compilation mechanism that converts declarative knowledge into direct-action productions, and the new Controller module that identifies successful control values for action. We discuss these three below.

Declarative Interpretation of Instructions

The subjects were given instructions relevant to playing YouTurn and AutoTurn (available in the online versions of the game – http://andersonlab.net/sf-paper-2017/turning.html). We did not encode these written instructions into the model but rather self-instructions for playing the game that reflected the knowledge conveyed in the written instruction. For instance, where the

[4] The Temporal module (Taatgen et al., 2007) uses ticks from a mental clock to represent time. The delay between ticks increases according to a noisy logarithmic function. The model was based on research by Matell and Meck (2000) who proposed a neuropsychological timekeeper that provided an explicit representation of time.

instructions said "When Fortress vulnerability reaches 10, it can be destroyed with a double shot. A double shot is firing two shots very quickly (less than 250 milliseconds apart)." we encode a self instruction that said "When the vulnerability is greater than or equal to 10 double tap the space bar"

The instructions were represented in ACT-R's declarative memory as a set of **operators (**as in many prior ACT-R models – e.g., Anderson et al., 2004, Anderson, 2007; Taatgen et al., 2008, Anderson & Fincham, 2014). Operators take the form of specifying a state in which they apply, specifying the action to take in that state, and then specifying the new state that will result. These instructions specify a strategy for playing the game and different subjects report adopting somewhat different strategies. For instance, some subjects report choosing to thrust on the basis of distance from the fortress while other subjects report choosing to thrust on the basis of time to hit the outer hexagon.

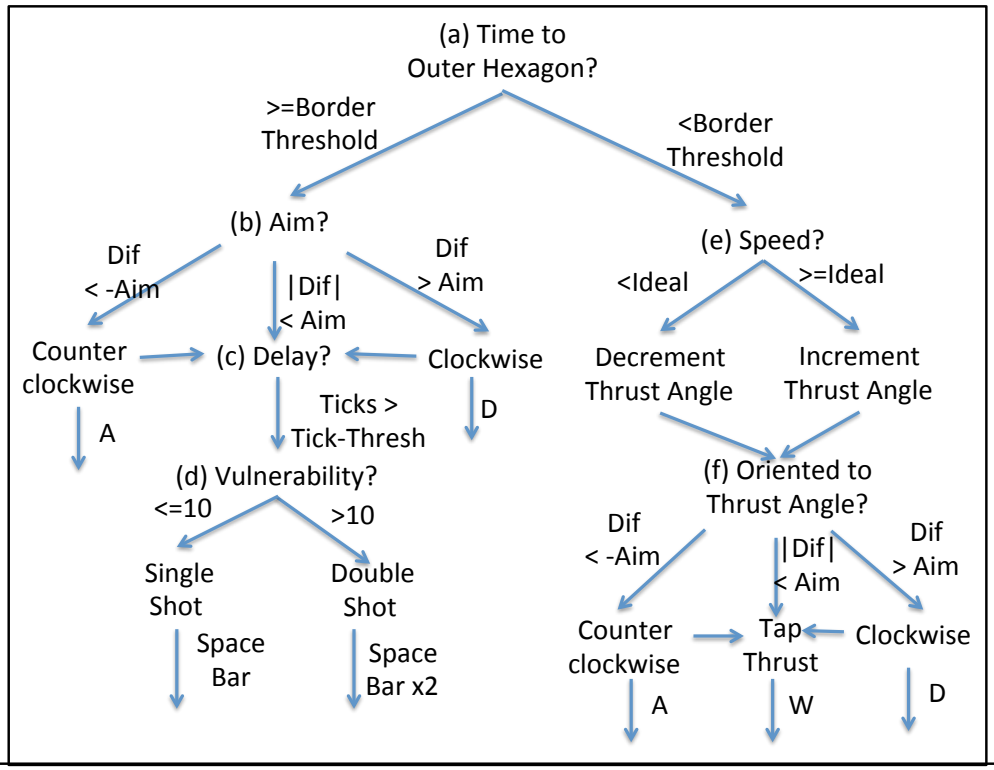


Figure 4. A representation of the instructions in YouTurn. Each set of branching arrows is an operator going from one state one of the next. Except for the bottom operators they test conditions to determine the next state. The bottom operators perform actions.

Figure 4 illustrates the operators behind one of the strategies we implemented. It shows the state structure of these operators in a graph form where each node represents an operator. This graph has a tree-like form because many of the operators in Space Fortress serve a selection function in that their action is to test for a condition in the game and the resulting state depends on the outcome of the test. The bottom nodes of the tree are all action operators that specify keys to press for shooting (space bar), thrusting (w key), turning counter-clockwise (a key), and clockwise (d key). Table 1 describes the branching operators that lead down to these action operators.

While we cannot represent all the slightly different strategies subjects could adopt, we created 4 variations on these declarative instructions that crossed 2 dimensions of strategy choice. One was whether the operator specified attending to distance or to time in deciding when to thrust (node a in Figure 4). The other concerned whether subjects adopted the same strategy during the one second that the fortress was absent before it respawned. Some subjects reported trying to concentrate their thrusting in that interval so that they could concentrate on shooting when the fortress re-appeared. There were the same 4 strategies for AutoTurn.

The model has productions that are designed to interpret these operators and

(a)

Production:
If STATE in GOAL BUFFER = DO-STEP
and ACTION in RETRIEVAL BUFFER = SPACEBAR
Then tap-press right index finger

RETRIEVAL BUFFER

Slot	Value
ACTION	SPACEBAR
PRE	SHOOT
POST	DONE

GOAL BUFFER

Slot	Value
STATE	DO-STEP
TIME-TO-OUTER	1.48
TICKS	12
AIM	4.70
VULNERABILITY	10

(b)

Production:
If STATE in GOAL BUFFER = DO-STEP
and ACTION in RETRIEVAL BUFFER = TEST-DIMENSION
Then *dimension* := ARG1 in RETRIEVAL BUFFER
If (*dimension* in GAME STATE BUFFER) > (*dimension* in GOAL BUFFER)
Then request retrieval of step SUCCESS in RETRIEVAL BUFFER

RETRIEVAL BUFFER

Slot	Value
ACTION	TEST-DIMENSION
PRE	FORTRESS
ARG1	TIME-TO-OUTER
FAIL	SPEED?
SUCCESS	AIM?

GOAL BUFFER

Slot	Value
STATE	DO-STEP
TIME-TO-OUTER	1.48
TICKS	12
AIM	4.70
VULNERABILITY	10

GAME STATE BUFFER

Slot	Value
X	243.0
Y	310.14
AIM	7.05
VEL	1.0
TIME-TO-OUTER	3.40

Figure 5. Pseudocode for two productions that interpret instructions and examples of buffer contents that would evoke these productions: (a) A production that performs a motor action. (b) A production that performs a comparison of a game quantity with a control variable.

others. Figure 5 illustrates two of these productions. Figure 5a shows a relatively simple production that interprets the instruction to hit the space bar. The Retrieval buffer holds the retrieved instruction and the Goal buffer holds the state (what the system is to do – in this case, perform a step of instruction) plus the values of various control variables. The production specifies that the motor module should issue a tap-press of the right index finger, which is over the spacebar, when the state in the goal is to do a step and the step retrieved is to press the spacebar.

Figure 5b illustrates a more complex production that applies at the binary branches a, d, and e in Figure 4. This production responds to the retrieval of a step that specifies testing a dimension. The dimension to be tested is in the ARG1 value of the retrieved instruction, which in this case is time to outer hexagon (node a in Figure 4). The production applies because the time to the outer hexagon in the Game State buffer, 3.40 seconds, is greater than the control value of Goal buffer, 1.48 seconds. In this case it branches to retrieve the step associated with a successful comparison, the success case of the current step in the retrieved instruction (AIM? in Figure 5b), which concerns aim (node b in Figure 4).

In total there are 31 instruction-following productions (see also discussion surrounding Figure 22). Some of these, like the two productions in Figure 5, are the same as or similar to instruction-following productions that have been used in previous models. Other instruction-following productions, like the ones that interpret instructions to increment and decrement thrust angle, are new because we had not previously needed to interpret this kind of instruction.

In the ACT-R model, the average time to retrieve an instruction is about 100 ms and the average time to execute a production is 50 ms. Thus, it takes on the order of 150 ms to interpret this operator. The time to interpret all the operators in Figure 4 leading down to a physical

action can take well over half a second with possibly some motor execution time[5]. The ability to interpret such declarative instructions allows the model to immediately play the game. However, this instruction following is much too slow for success in a fast-paced game.

Production Compilation

Without the instructions just discussed the model is not able to play the game. With these instructions but without production compilation, the model is able to play the game but accumulates virtually no points because it averages 4 to 8 times as many deaths as fortress kills (see Figure 16 later in the paper). It often takes too long to step through these instructions to act in time. (The timing parameters that control the speed through these steps are based on a history of successful ACT-R models – Anderson, 2007.) Subjects also initially find the task overwhelming. In YouTurn the subjects average 100 seconds to destroy their first fortress, while they only take about 5 seconds to kill the fortress by the end of play (for the model runs the average is 81 seconds for first kill and 5 seconds asymptotically).

Production compilation (Taatgen & Anderson, 2002) has been used in past instruction-following models of skill acquisition (e.g., Anderson et al., 2004, Anderson, 2005, Taatgen et al., 2008; Taatgen, 2013) to capture the transition that people make from acting on the basis of retrieved declarative information to responding directly to the situation. If two productions fire in sequence, then they can be combined into a single new production that is added to procedural memory. If the first of these productions makes a request to declarative memory, and the second production uses the retrieved fact to determine what to do, then the retrieved fact is substituted into the new production and the retrieval step is eliminated. A compiled rule can further be combined with another rule and so become still more specialized and more efficient. Eventually

[5] Counting motor execution into the total time is complex because execution of a motor action can overlap with retrieval of what to do next.

rules will be created that respond to game situations with direct actions without requiring retrieval of any declarative knowledge. This is critical in games like Space Fortress that put a premium on fast actions. Figure 6 shows one of the direct-action productions that are eventually acquired. This production checks that all the tests on control variables are satisfied so that a single shot can be taken: the ship is far enough from the border (3.40 > 1.48 seconds); the aim is good enough (3.05 < 4.70 degrees); enough time has passed between shots (15 > 12 ticks), and the fortresses' vulnerability is under the threshold for a kill shot (6 < 10). In this case it issues a request for a tap of the right index finger.

Production:
If STATE in GOAL BUFFER = PLAY
and (TIME-TO-OUTER in GAME STATE BUFFER) > (TIME-TO-OUTER in GOAL BUFFER)
and (TICKS in TEMPORAL BUFFER) > (TICKS in GOAL BUFFER)
and (AIM in GAME STATE BUFFER) < (AIM in GOAL BUFFER)
and (VULNERABILITY in GAME STATE BUFFER) < (VULNERABILITY in GOAL BUFFER)
Then tap press right index finger

TEMPORAL BUFFER

Slot	Value
TICKS	15

GOAL BUFFER

Slot	Value
STATE	DO-STEP
TIME-TO-OUTER	1.48
TICKS	12
AIM	4.70
VULNERABILITY	10

GAME STATE BUFFER

Slot	Value
X	243.0
Y	310.14
AIM	3.05
VEL	1.0
TIME-TO-OUTER	3.40
VULNERABILITY	6

Figure 6. Pseudocode for a direct-action production that issues a single shot and an example of buffer contents that would evoke its application.

High levels of performance require such production rules that directly respond with key presses to different game situations. However, the model can start to achieve positive points even with productions rules that compile parts of the operator sequences and so speed up performance. The development of such intermediate rules allows the model to begin to achieve kills about midway through the first game.

Newly compiled productions do not immediately start to take effect because they are initially dominated by existing productions. In particular, a compiled production must compete with the first production that it originated from (the "parent" production) since that parent production will match in any situation that the new production will match. Utility values of productions determine which production applies when more than one matches the current

situation. The utility of a production reflects the value of what they eventually achieve minus the amount of time it takes to do that. What this model is trying to achieve is to press the next key to play the game and it treats all key presses as having the same utility. Thus, the factor that determines the relative utility of productions is how fast they lead to key presses.

This model uses the standard ACT-R utility learning (Anderson, 2007). In ACT-R new productions start with zero utility and must learn their true utility. Thus, at first a new production will lose in competition with its parent and will only gradually begin to take effect as its true utility is learned. Whenever a pair of productions fire that would result in the compilation of the same production, ACT-R will adjust the utility of the already compiled production in the direction of the first of the pair of productions (the parent)[6]. ACT-R adjusts the utility of the new production towards the parent's utility according to a simple linear learning rule:

$$\text{Utility} = \text{Utility} + \alpha(\text{Parent-Utility} - \text{Utility}) \qquad \textbf{Equation 1a}$$

The utility values fluctuate moment by moment from the value prescribed above because of a normal-like noise. Thus, when the utility of the new production approaches that of the parent, the new production will have some probability of applying. When the production rule does apply, its value can be determined, and its utility changes according to the linear learning rule:

$$\text{Utility} = \text{Utility} + \alpha(\text{Value} - \text{Utility}) \qquad \textbf{Equation 1b}$$

The value of a compiled rule will be greater than its parent's value because it is more efficient. Eventually the compiled rule will acquire greater utility and tend to be selected rather than the parent. Thus, new productions are only introduced slowly, which is consistent with the gradual nature of skill acquisition. The parameter α determines how fast the utility of the new production

[6] Different pairs of productions can result in the compilation of the same production and so the parent can change from compilation to compilation.

grows and comes to exceed the parent. It takes many opportunities before rules come to dominate that skip all instruction-following steps and just perform the appropriate actions. The speed with which direct-action productions are acquired also depends on their frequency in the game. For instance, the direct-action single-shot production (i.e., Figure 6) is learned faster than the direct-action double-shot production because the single-shot action occurs approximately 10 times as often (to build up fortress vulnerability) as the double-shot action that destroys the fortress.

Control Tuning

A model that has the declarative instructions and production compilation still does not learn to play at human level. Runs of such models playing YouTurn reach an average a few hundred points per game (see Figure 16 later in the paper), much under the average of YouTurn subjects in Figure 2. To learn how to play YouTurn well, the model must learn settings for 5 control values: shot timing, aim (see Figure 1b), thrust timing, thrust duration, and thrust angle (see Figure 1b). Here we will use shot timing to illustrate the general principles behind control tuning in the Controller module. Control tuning involves three critical components: selecting a range of possible values for a parameter that will control an action, attending to relevant feedback, and converging to good range for that parameter.

Setting a Range for a Control Value. Subjects have some sense of what good values are for the various control dimensions. For instance, the instructions tell subjects that while they are building the vulnerability of the fortress up to 10 their shots must be at least 250 ms apart. Subjects have a vague idea of what constitutes 250 ms and tune that range to something more precise. ACT-R can be sensitive to the passage of time through its Temporal module that ticks at somewhat random and increasing intervals producing the typical logarithmic discrimination for

temporal judgments (Taatgen et al., 2007). The model places the possible range between 9 and 18 temporal ticks. Given the standard parameterization of the Temporal module, 9 to 18 ticks correspond to a range from approximately 150 to 500 ms.

Attending to Relevant Feedback. One of the reasons human players can learn faster than the typical deep reinforcement learners is that they know what features of the game are relevant as feedback to the setting of each parameter. They adjust their behavior according to these features and not raw points scored. The model looks for a control setting that yields an optimal combination of features specific to the dimension it is controlling. For instance, subjects know that a good setting for shot timing results in increments to vulnerability and not resets of vulnerability. Therefore, the features that the model attends to are increments and resets, weighting the first positively and the second negatively. For simplicity, for shot timing and all other control variables the positive and negative weights are equal (+1 and -1). Note that the Controller module treats increments in vulnerability as positive outcomes even though each shot results in a 2-point decrement in score.

Convergence. The Controller module samples a candidate control setting for a while, determines a mean rate of return per unit time for that control setting, and then tries another control setting. It gradually builds up a sense of the payoff for different control values. It learns a mapping of different choices onto outcomes through a function learning process (e.g., Koh & Meyer, 1991; Kalish, Lewandowsky, & Kruschke, 2004; McDaniel & Busemeyer, 2005). As suggested in Lucas et al. (2015) we assume a learning process that favors a simple function as most probable and in the current context this leads to estimating a quadratic function V(S) that describes payoff as a function of the control setting S. A quadratic function is a simple function

that has a maximum and so facilitates identification of a good point[7]. The model starts off seeded with a function that gives 0 for all values in the initial range and weights this function as if it is based on an amount of prior observations (20 seconds in the current model). It starts to build up an estimate of the true function as observations come in.

The control settings are changed at random intervals. The durations of these intervals are selected from an exponential probability distribution with a mean of 10 seconds[8]. Figure 7 shows the quadratic function that was estimated for shot timing in one run of the model after playing a single game (180 seconds). The red dots reflect performance of various choices during that game. They are quite scattered reflecting the random circumstances of each interval. For instance, note the value at 9 ticks, which is approximately 150 ms. This brief an interval would typically lead to resets, but at the beginning of the game when instruction interpretation slows down action, it results in about 2 hits per second. As discussed with respect to reinforcement learning, in a variable environment control tuning needs to balance exploration to find good values with exploitation to use the best values it currently has. For this purpose, control tuning uses a softmax equation that gives the probability of selecting a value S at time t:

$$P(S,t) = \frac{e^{V(S)/T(t)}}{\int_{\min}^{\max} e^{V(x)/T(t)}}$$

Equation 2a

[7] It also is efficient to update and would generalize to multi-dimensional space.

[8] It seemed unreasonable to propose that the model always used the exact same period for evaluation. The exponential was used as an uninformed guess about the true distribution. While having such variability in evaluation period might be important in some contexts, the results in Space Fortress would not substantially change with a constant 10 seconds. For an analysis of the effect of the mean duration (rather than variability) see Figure 24d.

V(S) is the current value of the quadratic function for control value S. T(t) is the temperature. As it decreases the model will more frequently select the option it thinks is best. Temperature is a parameter used in reinforcement learning schemes (Kaebling et al., 1996) to control the trade-off between exploration and exploitation. Frequently it is started at a high value to allow the agent to explore options and decreased over time to allow the agent to focus on the most profitable options. While originally introduced in artificial intelligence, it finds frequent use in current models of human learning (e.g., Jepma et al., 2011; Kool et al., 2017). In our model, temperature decreases over time playing the game, t, according to the function

$$T(t)=A/(1+B*t) \quad \textbf{Equation 2b}$$

where A is the initial temperature and B scales the passage of time (in the model this is set to 1/180 of a second, reflecting a 180 second game). Figure 8 illustrates convergence for one random run of 20 games. Figure 8a shows that in this run the estimated quadratic function became fairly tuned by game 5. However, Figure 8b shows that the choices continue to concentrate more around the optimal value as the model plays more games.

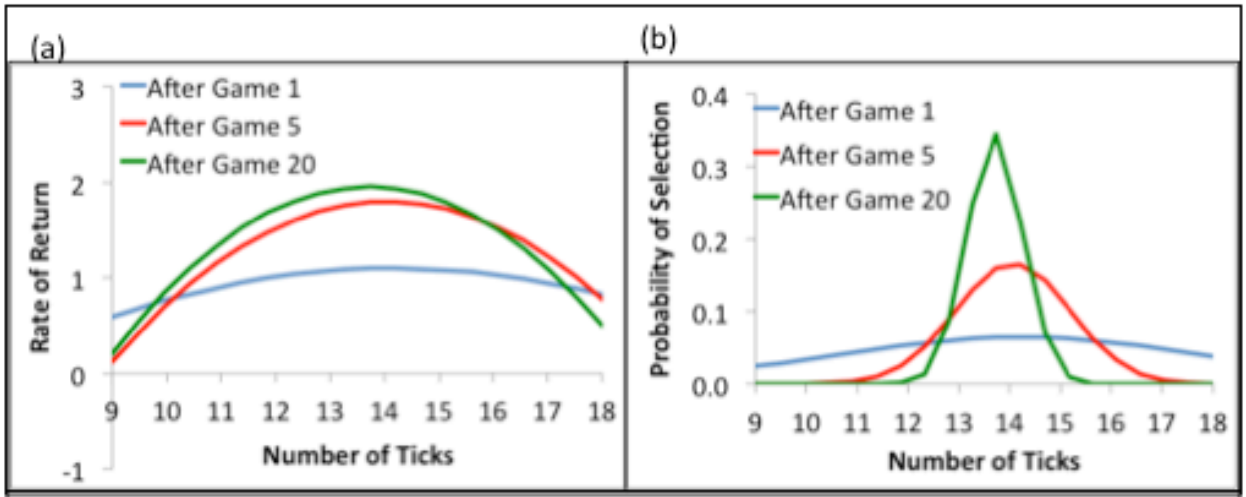


Figure 8. (a) The quadratic functions estimated for different delays at different points in the one sequence of model runs through 20 learning games. (b) The probability of choosing different delays after different numbers of games.

Summary. The Controller module learns to map a continous space onto an evaluation by exploring that space and converging on an optimal point. While the spaces being mapped in this task are all one-dimenisonal, the methods could be generalized to multi-dimensional spaces. Other function-learning approaches could be applied to this task. Indeed, deep reinforcement learning methods could apply to this task. This would differ from the typical application of deep reinforcement learning methods to games in that they would apply to reduced spaces (e.g. distance to border) relevant to a particular control variable rather than the full space of the game (e.g., all pixels) and the evaluation would be specific to that control variable. Such an informed focus seems characteristic of human learning. In some cases this focus can mean that humans miss learning the best solutions, but the focus enables rapid learning.

Comparison of Model and Data

Figure 9 compares subjects and model in terms of total score. We ran 100 models in each condition. Besides matching the overall growth in points of the subjects, the models show a high variability in performance, almost matching the range shown by the subjects. The 100 models were created by crossing 4 different strategies, with 5 different settings for production compilation, with 5 different settings for control tuning:

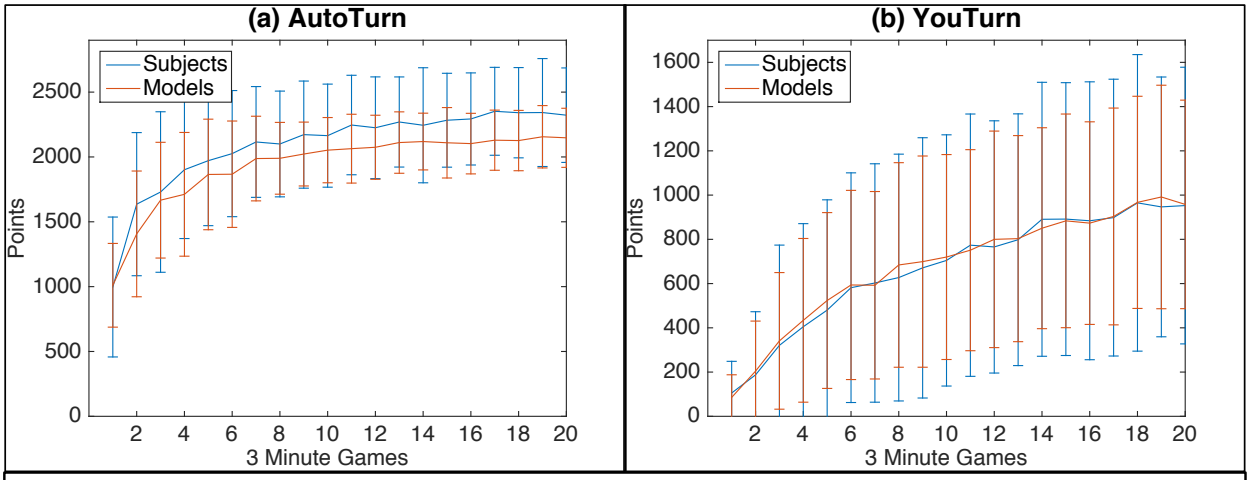


Figure 9. A comparison of subject and model performance: Error bars reflect one standard deviation of the population.

1. **Strategies.** Four different strategies for navigation were implemented to reflect different things subjects reported. The models either used distance from fortress or time to hit the outer hexagon in deciding when to thrust. Models either uniformly used this strategy for thrusting or emphasized thrusting when the fortress was absent by thrusting as long as the ship was moving from the fortress. Crossing these two factors yielded the 4 strategies. Models perform slightly better when attending to time to outer hexagon rather than distance from fortress (1,951 vs. 1,920 mean points in AutoTurn; 707 vs. 658 mean points in YouTurn). The benefit of emphasizing navigation when the fortress was absent was even smaller (1,950 vs. 1,921 mean points in AutoTurn; 684 vs. 681 mean points in YouTurn).
2. **Production Learning Rate α.** We used values of .025, .05, .1, .2, and .3 for this learning parameter that controls the rate at which compiled productions replace the productions they are compiled from. This range corresponds to values that have been used in past models. This parameter loosely reflects differences in relevant prior experience in subjects. High values of α mean that models will soon start applying productions that collapse some of the steps in taking an action, which loosely mimics subjects starting off with such productions from past experience. This is similar to Taatgen's (2013) transfer model. Mean points varied strongly with this factor: 1748, 1850, 1958, 2043 and 2069 points in AutoTurn and 277, 553, 702, 893, and 990 points in YouTurn.
3. **Initial temperature A.** We used values of .1, .25, .5, 1, and 2. Lower temperature will result in less random behavior and greater choice of values that have performed best. To some degree starting with a lower temperature will correspond to more

attentive subjects who will incorporate more of their learning experiences and more motivated subjects will weight the payoff higher. Scores showed a curvilinear relationship to this factor with best scores for an initial temperature of 0.5: 1862, 1974, 2020, 1971, and 1851 in AutoTurn and 569, 743, 871, 737, and 596 in YouTurn. Performance can be poor with low initial temperature because this can lead to convergence on non-optimal values and can be poor with high initial temperature because there never is convergence.

The goal in producing a set of different models was not to create a model of each subject's unique behavior, but to show that the theory predicts subject-like behavior over a range of individual variation of the sort that is occurring in subjects. The values chosen for the learning rate and temperature gave a reasonable approximate to human behavior and we did not make an effort to carefully tune them to maximize fit.

While absolute levels of performance are quite similar, subjects are outperforming the model in AutoTurn (mean points of 2,086 vs. 1,936 – $t(140)=2.49$, $p < .05$) while they are slightly worse in YouTurn (mean points of 672 vs. 683 – $t(146)=0.13$). Little can be made of such direct comparisons of level of performance. The subjects do not reflect all the variation in a larger population that did not self-select to be in this experiment on Mechanical Turk. The models also do not reflect all the possible models that could be created with different strategies or parameters. Also, the models are trying all the time, something that is not true of every subject. More important than the absolute level of scores are the learning trends over games, which are strikingly similar. The correlation of the mean scores across the 20 games is .989 in AutoTurn and .995 in YouTurn.

Each time the fortress is destroyed subjects gain 100 points and each time they are killed they loose 100 points. Thus, kills and deaths largely determined their total points. The only additional factor is the 2 points that each shot costs. Figure 10 shows the growth in fortress kills over trials and the decrease in ship deaths. The correspondence between models and subjects is strikingly close for fortress kills (Figures 10a and 10b). However, there are differences in deaths. Subjects are showing a greater decrease in deaths over games than models in AutoTurn (Figure 10c) but less decrease in YouTurn (Figure 10d). In AutoTurn the principal cause of death for both subjects and models is being shot by the fortress (Figure 10e) when the ship is flying too slow. In AutoTurn, because one cannot turn to easily adjust speed, ship speed is a long-term consequence of the pattern of thrusting. Subjects are learning patterns that avoid such deaths while the model is not. In YouTurn ship speed is largely determined by the thrust angle of the last thrust and models are better at learning this thrust angle and show more improvement (Figure 10f).

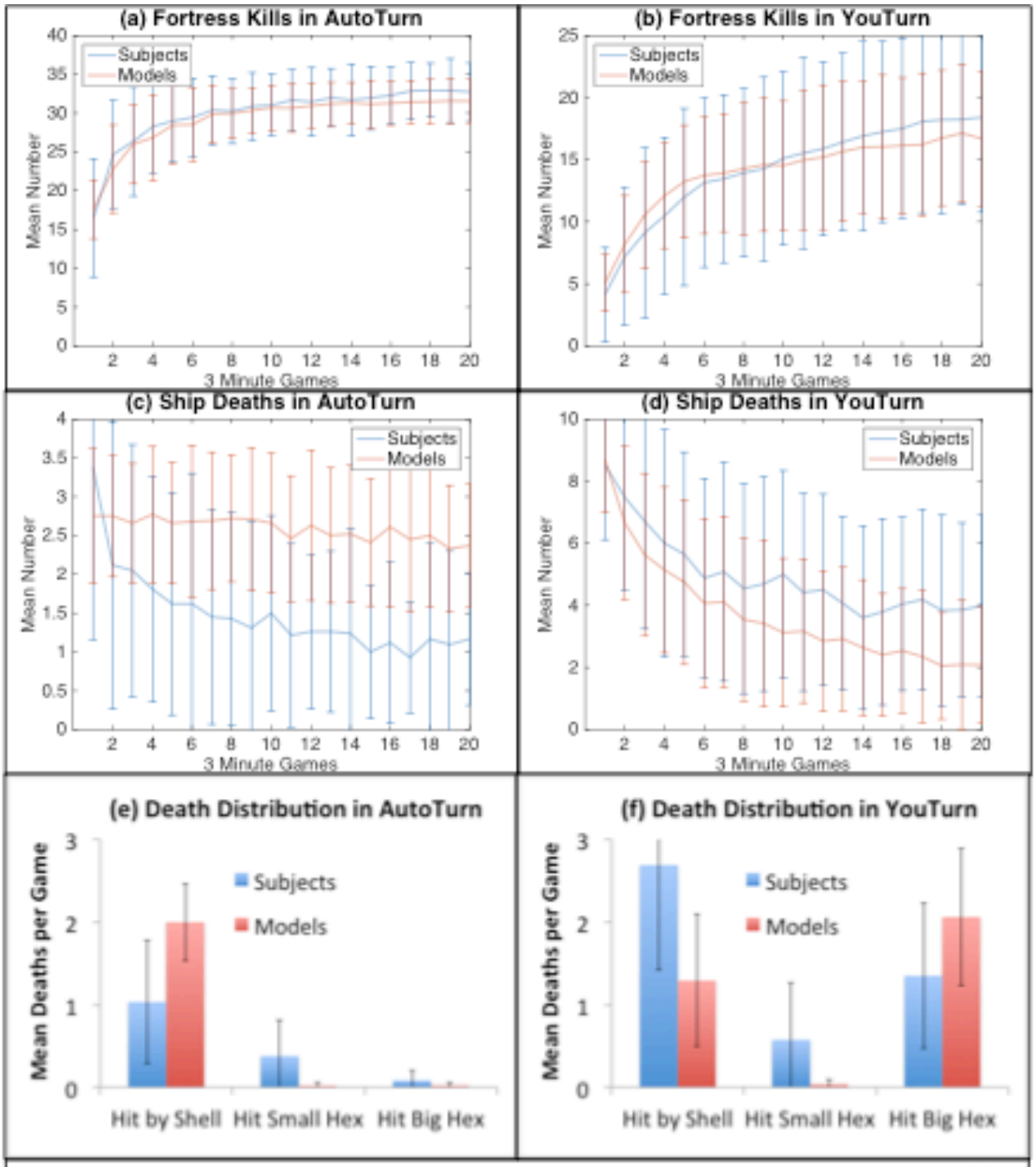


Figure 10. A comparison of subjects and models on two critical factors determining score. Parts (a) and (b) show growth in mean number of kills over games. Parts (c) and (d) show decrease in mean number of deaths over games. Parts (e) and (f) show the average number of the 3 kinds of deaths per game. Error bars reflect one standard deviation of the population.

Given that the new addition to ACT-R is control tuning, the remainder of these data-model comparisons will involve results relevant to control values being learned.

Shot Timing. The simplicity of AutoTurn makes it good for assessing the learning of shot timing. Most of the key presses are sequences of shots in a row (an average length of 6.3 for subjects and 7.8 for the models). This means one can use intershot times to get an estimate of how subjects are pacing their shots. Figure 11a compares subjects and models in terms of intershot time. Subjects show a decrease from about 650 ms to about 350 ms. Models show a decrease from about 600 ms to about 400 ms. The asymptotic value of 350 ms is 100 ms above the minimum to avoid a reset, but given variability in one's ability to time actions and the game variability, the optimal setting for this control variable is above the minimum. On the first game the models average an intershot time of 600 ms even

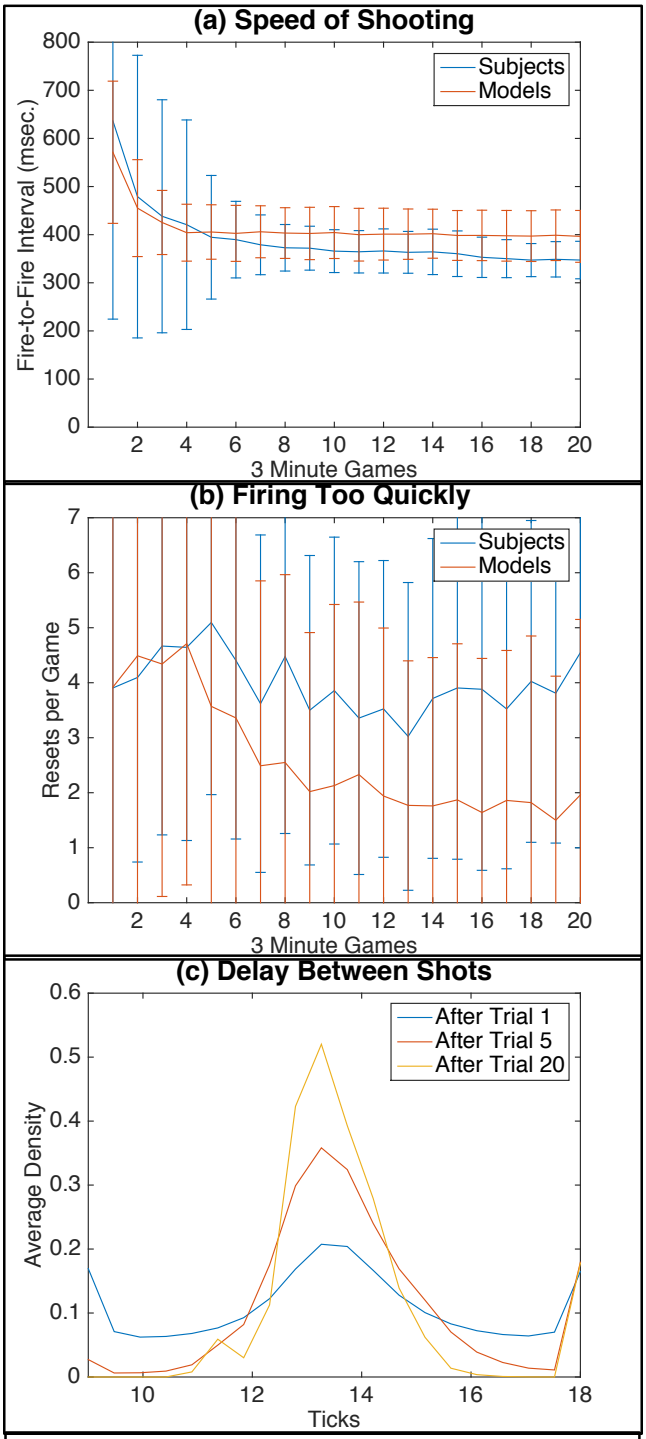


Figure 11. Shot Timing in AutoTurn: (a) Mean intershot duration per game for subjects and models; (b) Mean number of resets per game for subjects and models. (c) Probability of choosing different time thresholds, averaged over the 100 models. Error bars in (a) and (b) reflect one standard deviation of the population.

though the upper bound of their range for timing (18 ticks) is about 500 ms. They take this long because of the time to reason through all of the steps in determining that they can shoot. We assume the slow initial times of subjects also reflect the cost of thinking through what they should do.

Figure 11b shows how often subjects and models reset the vulnerability to 0. Subjects average about 4 resets. The models start at this value but decrease to about 2 resets. While resets are not decreasing for subjects, the number of shots taken is increasing from about 250 to 450 reflecting the decrease in intershot time and other improvements in performance. So the proportion of shots resulting in resets is decreasing in both populations.

Figure 11c shows the probability of the Controller module choosing different control values for waiting times. This figure shows the choice behavior averaged over the 100 models. The models tend to converge on a sweet spot between 13-14 ticks, which is about 300 ms. (production execution adds another 50 ms to intershot times). However, 8 models converged to very slow shooting (resulting in the upswing at 18 ticks in Figure 11c). Such slow shooting results in no resets but more than 100 fewer shots than the other models and an average of 500 points less.

Aiming. Subjects have to turn their ship in YouTurn and turning is critical both to aiming at the fortress and navigation. The model uses hits and misses as feedback on what is an adequate aim. Figure 12 displays information on aiming comparable to Figure 11 on shot timing. Part (a) shows how far the aim was off the fortress when a shot was taken. Both subjects and models show a decrease in the average angular disparity over games. Part (b) shows the resulting decrease in proportion of missed shots. The results are strikingly similar. Figure 12c shows that the models tend to converge on a maximum disparity of about 9-10 degrees.

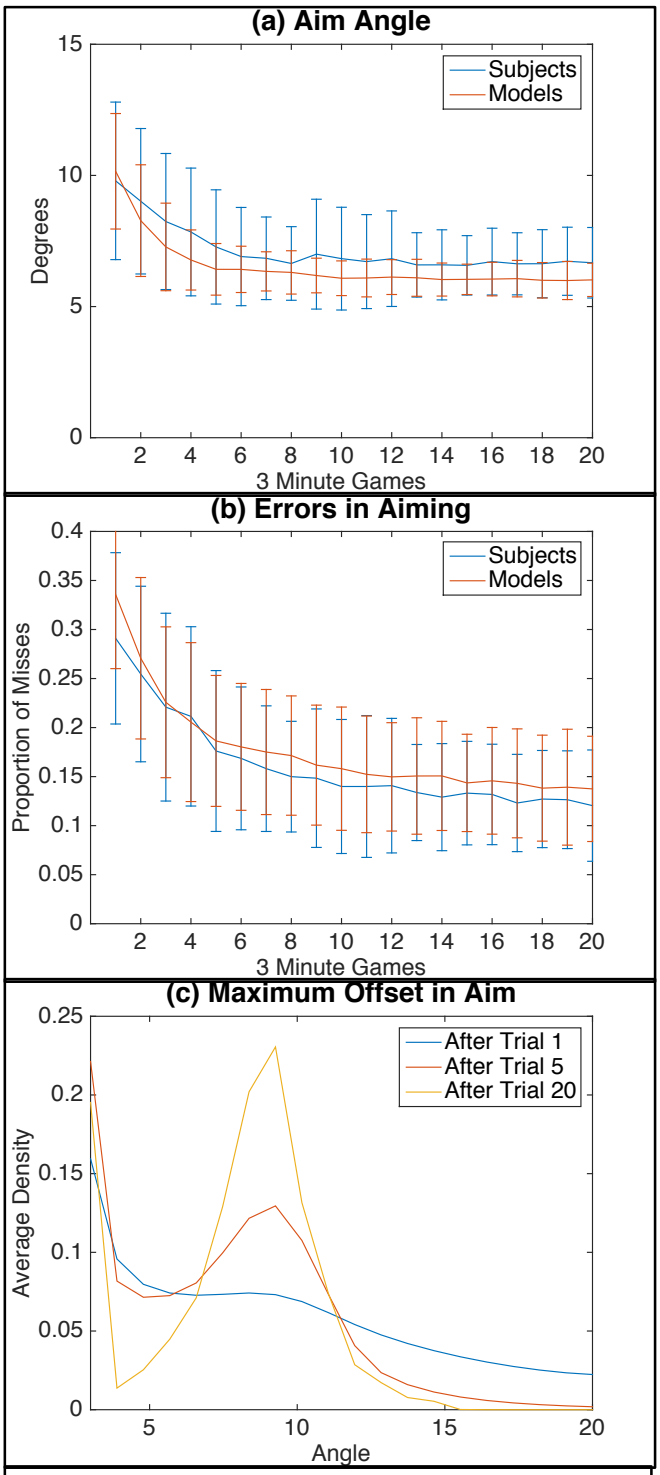


Figure 12. Aiming in YouTurn: (a) Mean offset of aim at shot per game for subjects and models; (b) Mean proportion of misses per game for subjects and models. (c) Probability of choosing different maximum aim offsets, averaged over the 100 models. . Error bars in (a) and (b) reflect one standard deviation of the population.

When to thrust. The model chooses to thrust according to either how soon it will hit the outer hexagon or its distance to the outer hexagon. If using time, the range of the control values is between 1 and 5 seconds. If using distance, the range is between 10% of the way to boundary to 90%. The longer the model waits to thrust the greater chance of completing a fortress kill. If it waits too long to thrust it will crash into the outer border. Therefore, it treats fortress kills as positive feedback and deaths as negative feedback.

As is the case for shot timing, good data on when thrust decisions are made can be collected in AutoTurn because there is never any need for preliminary turning. Figure 13a displays the mean distance at which subjects and models chose to thrust. Both populations show a bit of a downward drift but subjects are thrusting on average 10-20 pixels earlier. Figures 13b and 13c show the average probability of the choices that

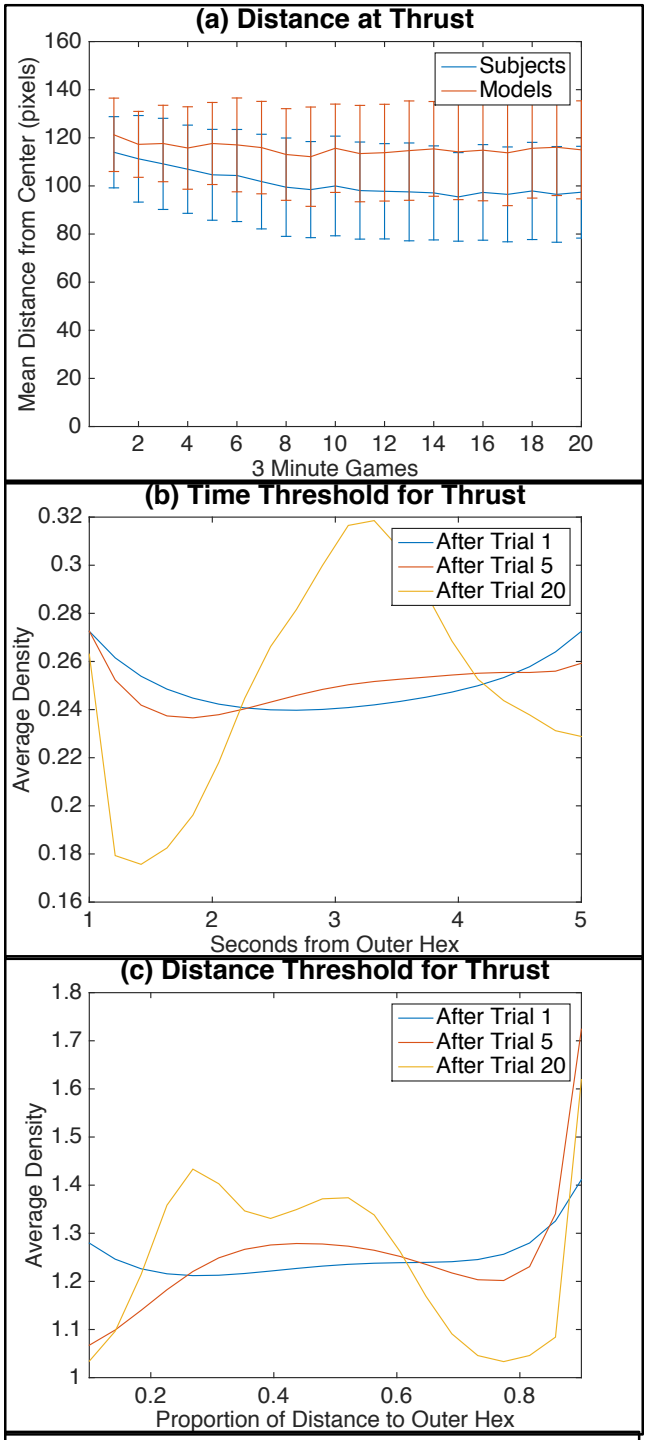


Figure 13. Distance in AutoTurn: (a) Mean distance from fortress at which subjects and models chose to thrust; (b) Probability of choosing different time thresholds, averaged over 50 models.. (c) Probability of choosing different distance thresholds, averaged over 50 models. Error bars in (a) reflect one standard deviation of the population.

determine when the model chooses to thrust.

Thrust time. The effect of a thrust increases linearly with the duration that the thrust key is depressed (a vector of .3 pixels per second in the direction pointed is added each game tick, which is a 30th of a second). The model searches between 67 ms (which is 2 game ticks, the typical time associated with a tap) to 167 ms (5 game ticks, a noticeable duration of sustained pressing). While the model may choose to press for a certain period, motor noise and asynchrony between model and game will result in the thrust key being depressed for a duration somewhat different than the selected time. A thrust that is very long may result in driving the ship into the inner hex. A thrust that is too brief may result in a slow ship speed and the ship

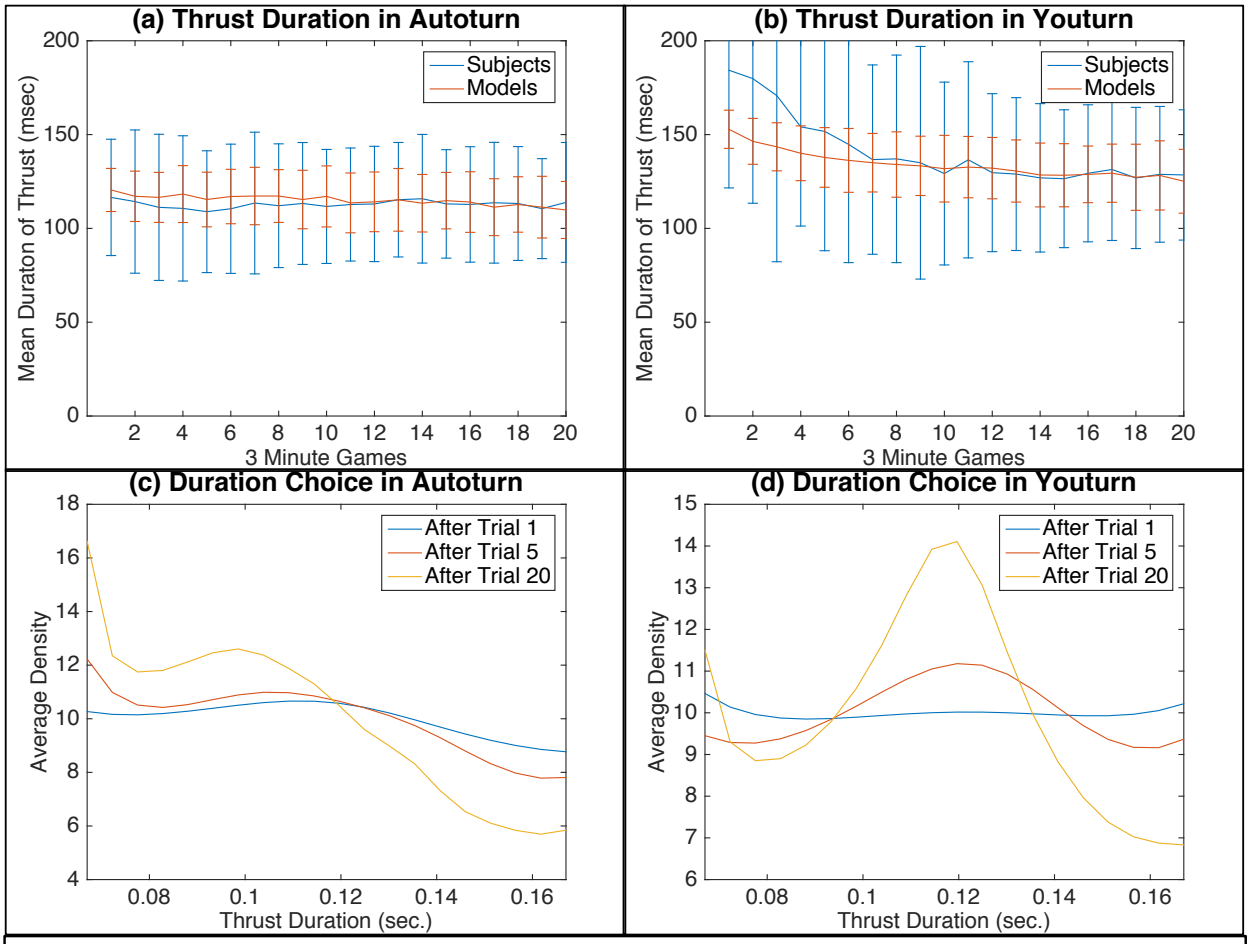


Figure 14. Thrust Duration in AutoTurn (a) and YouTurn: (b) Choice of thrust durations in the models for AutoTurn (c) and YouTurn (d). Error bars in (a) and (b) reflect one standard deviation of the population.

being shot by the fortress. As with when to thrust, the Controller module treats fortress kills as positive feedback and ship deaths as negative feedback.

In both AutoTurn and YouTurn the duration that the models and subjects press thrust provides estimates of what they think is a good thrust time. Figures 14a and 14b show that the thrust times are similar between models and subjects. In YouTurn thrusts are longer in the early games. In the model this is because of long emergency thrusts the model takes when it gets too close to the border. This mainly happens early in play when the model is taking too long to decide its next action (and there are more things to decide and do before a thrust in YouTurn than in Autoturn). Figures 14c and 14d show the choice of thrusts in the AutoTurn and YouTurn. In AutoTurn there is little effect of thrust choice and so the average distribution (over models) of choices stays rather flat with some preference for briefer thrusts in later games. In contrast, in YouTurn the consequences of a thrust too long or too short can be serious in terms of controlling speed. Here most models converge to thrusts of about 120 ms.

Thrust angle. If the ship is not at a good thrust angle, thrusts can increase the speed of the ship and once the ship starts moving too fast it cannot be controlled. The instructions explain that to prevent the ship from picking up speed the thrust angle (difference between flight of ship and ship orientation – see Figure 1b) should be greater than 90 degrees. In this case the ship is pointing somewhat away from the flight vector of the ship and will subtract from that vector. Each subject's thrust duration determines a thrust angle that would maintain current speed[9]. This *thrust angle* is the control variable that the subjects need to learn. The model searches for a

[9] As a reference for what a plausible thrust angle should be, if the thrust key was held down long enough to add as much velocity in the direction of the thrust angle as the current velocity, a thrust angle of 120 degrees would maintain current velocity. The average velocity added by the thrust is just a little less than the average ship speed, implying an average thrust angle a little less than 120 degrees.

good value for the thrust angle between 85 and 140 degrees. The sign of a good thrust angle is that resulting ship speed stays within the good range (30 to 50 pixels per second) and of a bad thrust angle that it is slower or faster. Therefore, the model evaluates thrust angle in terms of the speed of the ship after the thrust, treating speeds in the good range as positive feedback (+1) and speeds slower or greater as negative feedback (-1).

Subjects are told they can adjust the speed of the ship by adjusting the thrust angle. Correspondingly, how the model treats the thrust angle it has chosen for its control value depends on the speed of the ship (Figure 4e). It will increment the angle it thrusts beyond the chosen thrust angle if it is flying too fast and decrement the angle if it is flying too slow.

Figure 15 compares subjects and models with respect to thrust angle in YouTurn. Part (a) shows that the average thrust angle

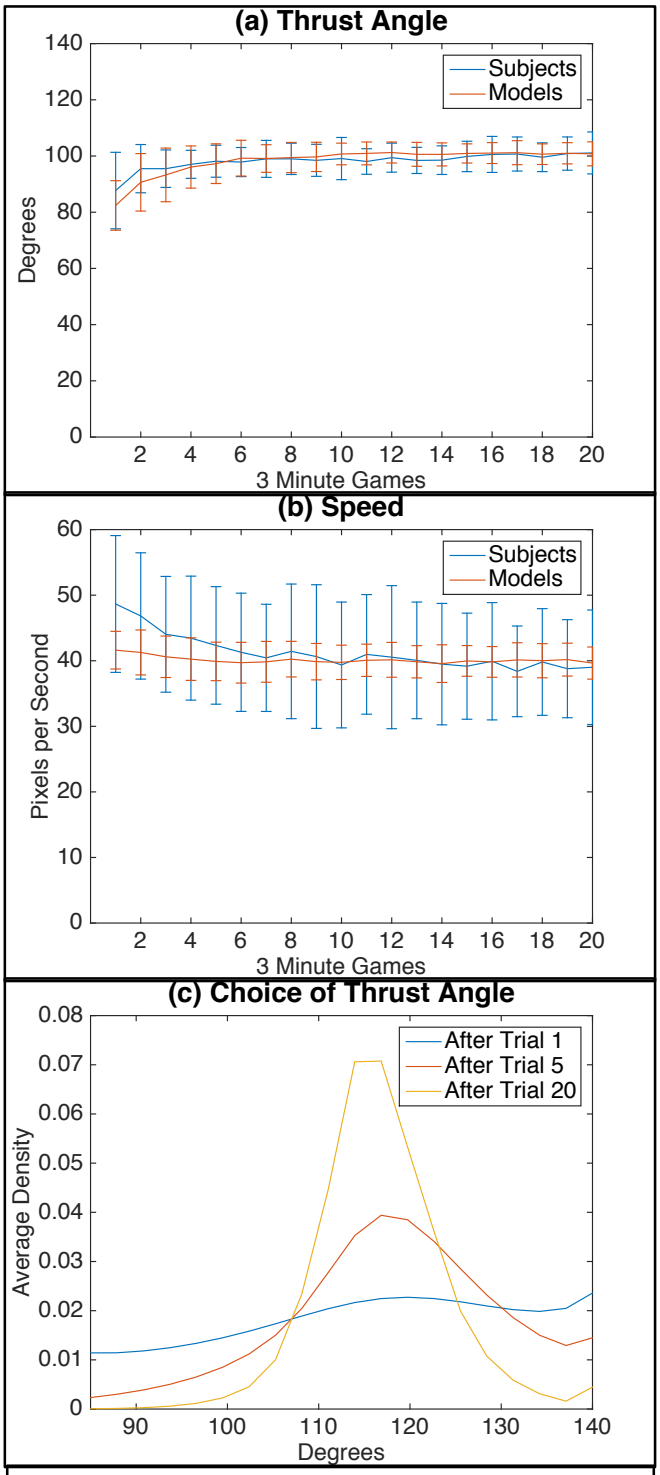


Figure 15. Thrust angle in YouTurn: (a) Mean thrust angle per game for subjects and models; (b) Mean speed of ship for subjects and models. (c) Probability of choosing different thrust angles, averaged over the 100 models. . Error bars in (a) and (b) reflect one standard deviation of the population.

of both subjects and models increases a little over the games to about 100 degrees. Part (b) shows that they both converge to an average speed of about 40 pixels per second. Part (c) illustrates how the models' choices for a control value converge over time. Although the preferred control value is over 110 degrees by the end, the average angles at which the ship thrusts are just over 100 degrees. Aiming often leaves the ship a little under the chosen thrust angle but not enough to provoke a turn.

Summary of Comparison of Model and Data

Figures 9-15 provide a comparison of the play of subjects and models on multiple dimensions. There were four strategic variations in model play in the models, but they do not begin to cover the range of things that subjects reported (often things idiosyncratic to one subject). So there is no reason to expect a perfect correspondence in behavior. In some cases the differences between subjects and models were significant, but in all cases there was considerable similarity and overlap between the subjects and models.

There are multiple reasons for the similarity between model and subject behavior. First, the nature of games constrains what a player can do and succeed. Second, the models reflected human limitations and thus cannot achieve the super levels of a pure AI approach. Third, the instructions provide both subjects and models with a starting point of information about how to play the game. Fourth, and this is a major theoretical claim, the key human learning mechanisms in the model are the key learning mechanisms in humans: Production compilation moves the players to the point where they can directly respond to what they see without having to think it through. Control tuning allows players to discover the correct setting on continuous parameters critical to successful play.

To investigate the contribution of production compilation and control tuning, we created 9 variants of the original model by crossing 3 choices with respect to production compilation and 3 choices with respect to control tuning. For production compilation the three choices were:

1. Already Compiled. The model started out with the direct-action productions that would result from production compilation.
2. Compiling. New productions were created from the instruction-following rules as in the models reported in the prior section.
3. No Compiling. The model worked by interpreting the declarative instructions throughout the game. Production compilation was turned off.

For control tuning the three choices were:

1. Tuned. The model started with the control variables fixed at high performing values.
2. Tuning. The control tuning mechanism was in operation as in the models reported in the prior section.
3. No Tuning. The choice of control values stayed uniform across the full range of values sampled. Control tuning was turned off.

All models had the strategy of attending to time to border and used a uniform navigation policy whether fortress was present or not. For the models that were compiling, the learning rate (α in Equation 1) was set to .2. For the models that were tuning, the initial temperature (A in Equation 2b) was set to .5.

Figures 16a-c shows the average performance in YouTurn of these different combinations extended out to 100 games to clearly identify asymptotic performance. The three No Compiling versions are scoring close to zero points (averaging well under 100 points per game) because they cannot act with sufficient speed to play the game properly. The Compiling models come close to the level of models of the corresponding Compiled models over the course of the games. They tend to perform a little worse because occasionally non-direct-action productions will apply. The Compiled, Tuned models have nothing to learn and define the optimal performance of the model with human performance parameters of about 1760 points over the course of the games. The Compiled, Tuning models that are learning control values reach an average about 1670 points by the end of the 100 games. Their slightly lower score than

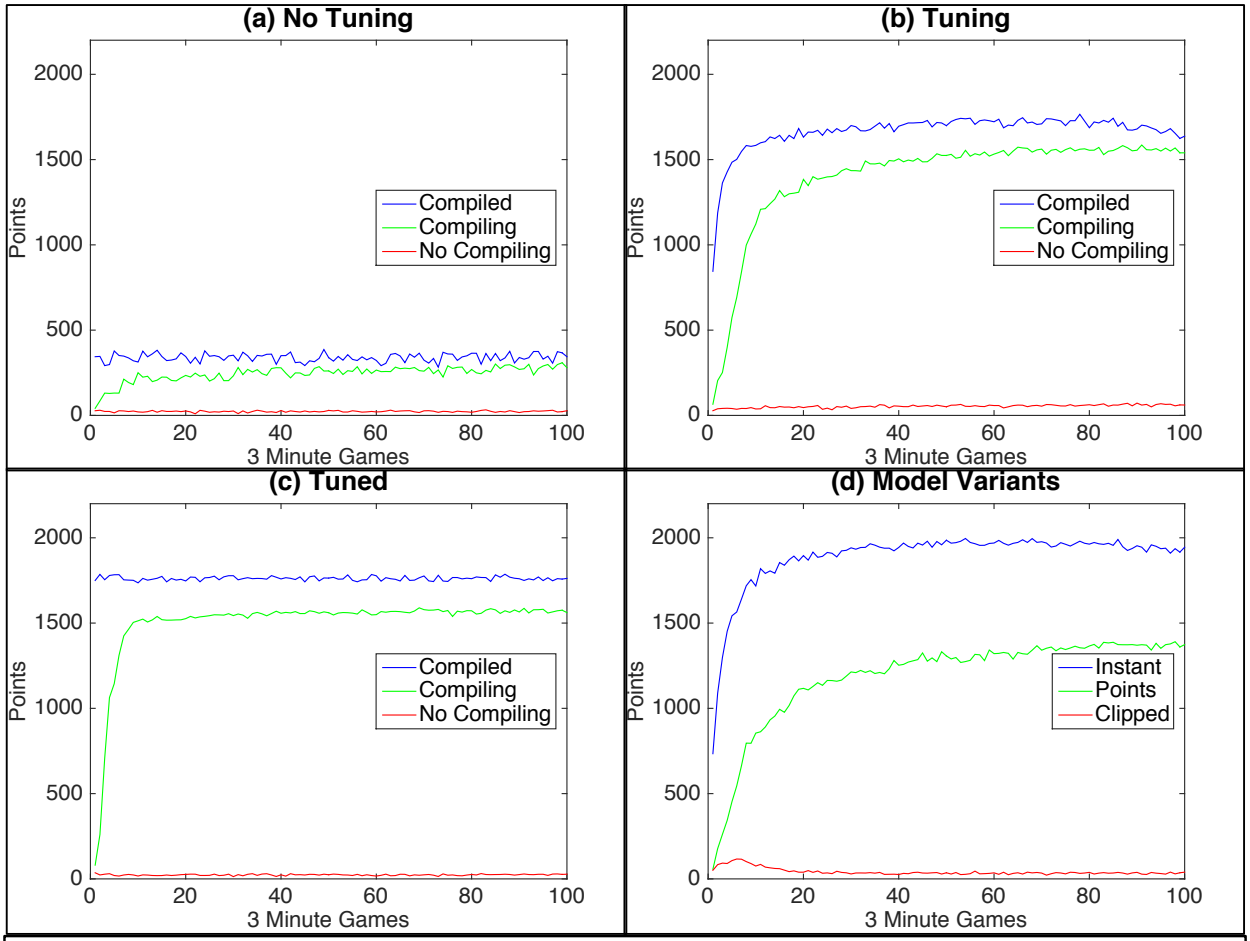


Figure 16. Performance of various model variants in YouTurn. Parts (a) – (c) show a crossing of a tuning factor and a compiling factor. Part (d) shows the behavior of three models that deviate from the architectural assumptions of the model in specific ways.

the Compiled, Tuned models reflects that some of the models are converging to suboptimal control choices. For instance, see the discussion involving Figure 11c.

Figure 16d shows the results with three models that were created by changing aspects of the ACT-R architecture, running the same model as the Compiling, Tuning model in Figure 16d:

Instant. ACT-R models the time costs of the perceptual, cognitive, and motor actions. The line labeled "Instant" represents a model with those limitations removed, giving it an advantage that the artificial deep reinforcement learning systems have. This version still needs to press keys long enough to get the effects it wants and needs to pace it shots to avoid resets, but it can begin executing an action as soon as it is needed. Not surprisingly this model performs better than the ACT-R model (asymptotic performance of 1940 points rather than 1670), but it still has to learn appropriate settings for control variables. This model has the advantage that the artificial deep reinforcement learning systems have.

Points. We created a version of the model that used change in points to assess each control variable rather than features directly relevant to that variable like resets for shot timing and misses for aiming. Points were divided by 10 to get a magnitude of reinforcement in the same range as the magnitudes of the ACT-R model. This version does not do as well as the ACT-R with more direct feedback (asymptotic performance of 1379 points rather than 1670), but we were surprised by how well it did do.

Clipped. To deal uniformly with different games that have different scoring schemes, Mnih et al. (2013) clipped the point changes to simply be plus or minus 1, reflecting gains or losses. This scheme behaves particularly poorly when the model is applied to Space Fortress. This is because each shot costs the player 2 points and it takes at least 12

shots to kill the fortress, which is worth 100 points. This means in the clipped version the models suffer 11 -1's before it gets a +1 for the kill.

Experiment 2

While the model depended on many aspects of the ACT-R architecture, what was new was the Controller module. The Controller module learns control values that are specific to the game. For instance, the parameters it is learning for controlling the ship orientation and flight are specific to the goals of Space Fortress and would not generalize to flying the ship in other circumstances. We realized this when we created another game to train the flight skills, which are quite challenging to subjects in YouTurn. The new game, called Space Track, involved flying a ship in the same frictionless environment with the same controls. Figure 17 provides a schematic representation of Space Track and its contrast with Space Fortress. In Space Track the player is flying through a series of rectangles. There are always two rectangles on the screen but as soon as one is cleared, it disappears and a new one appears at a random angle at the end the remaining rectangle. Points are gained as a function of the number of rectangles cleared and points are lost if a player hits the boundaries of the rectangles. The critical feature of Space Track is that it has the same flight dynamics as Space Fortress. The same keys are used to turn the ship and add thrust. In both games the new direction the ship goes in is determined by the vector addition of its current velocity and direction, and the thrust magnitude and the orientation of the ship.

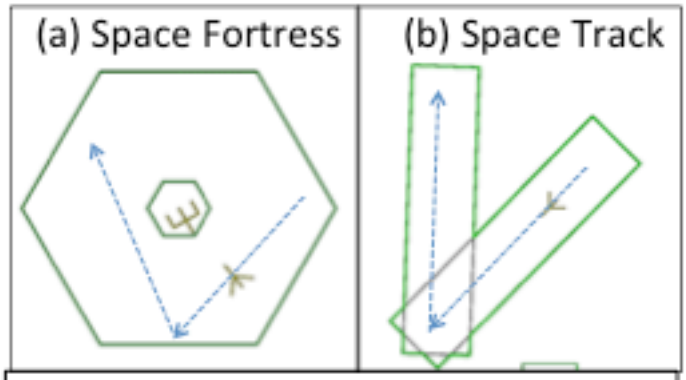


Figure 17. Schematics of the two games: (a) In Space Fortress one is flying a ship between two hexagons shooting at the fortress in the middle. (b) In Space track one is flying a ship down a rectangle and making a turn to fly down the next rectangle.

One might think that learning to fly in this frictionless space involves learning an internal model of how current direction and speed combine with angle of ship and thrust to determine future angle and speed. Subjects would choose angle and thrusts for their ship to achieve the desired angle and speed of flight. By practicing Space Track, subjects would learn how to achieve the desired angle and thrust by the duration that they press the desired keys. If they learned this first, then when they had to play Space Fortress they could focus on learning shooting skills. We had thought this when we first designed Space Track, but discovered that the model predicted otherwise. In Space Track the Controller module needed to learn a set of control variables that were different than those needed to play in Space Fortress. As we will show, the model predicted little or no transfer between the two games.

Subjects played two sessions of 20 3-minute games. Four conditions were created by crossing whether they played Space Fortress or Space Track in each session. For both Space Fortress and Space track there were three kinds of sessions:

Session 1 Learning: Performance over the 20 games in the first session. Given that subjects did not know what game they would play in the second session, we averaged over those who continued to play the same game in the second session and those who switched games.

Session 2 Learning: Performance during the second set of 20 games for subjects who continued playing the same game. The difference between this and Session 1 Learning provides a measure of what is learned in the first 20 games.

Session 2 Transfer: Performance during the second set of 20 games for subjects who switched to the other game. The difference between this and Session 1 Learning provides a measure of transfer.

Space Track was implemented in the same software as Space Fortress and used the same A, W, D keys and the ship responds in the same way to the key presses. Details of its implementation are specified in Appendix A. The scoring scheme is even simpler than Space Fortress:

1. Subjects gained 25 points for every rectangle their ship cleared.
2. Subjects lost 100 points every time their ship flew outside the rectangle walls and died.

Again, to keep subjects from being discouraged early on, their score never went negative.

Good performance depends on flying as fast as one can without flying into the sides of the rectangles. If one is flying on a trajectory that will not hit the side of the current rectangle, one can increase speed by pressing thrust. The challenge with flying at a fast speed is making the **Hard Turns** to be able to fly down the next rectangle. In principle, a hard turn can be achieved by first turning the ship to an angle that roughly bisects the angle between the two rectangles. Then, if one thrusts at the right time for the right duration, the ship will turn to fly down the middle of the next rectangle. To the extent that the timing of a hard turn is off (either in initiation or duration), the ship will take a trajectory that would lead it to crash into one of the two rectangles. In this case it is necessary to rapidly make some further turns and thrusts to get the ship on a successful path down the next rectangle. The faster one is flying the less time there is to correct the path produced by an error in turning.

Subjects. 121 subjects completed both sets of 20 games[10] – 31 who played Space Track on both sets and 30 in the other 3 transfer combinations. When subjects chose this Mechanical Turk hit they were not told what condition they would be in nor whether they would be transferring to a different game for the second set. They were only told they would be playing a game in which they would have to control a spacecraft in a 2D frictionless environment.

We excluded 9 subjects who averaged less than 25 points a game (the highest average of these 9 was 19 points while the lowest average of the included subjects was 64 points per game)[11]. This left 29 subjects who played Space Track both sessions, 29 who transferred from Space Track to Space Fortress, 28 who transferred from Space Fortress to Space Track, and 26 who play both sessions in Space Fortress. There were 81 males and 31 females in the included group with a mean age of 31 years (range 18 to 49). All had played video games: 8 reported playing less than a hour per week, 37 reported 1-5 hours, 21 reported 10 to 20 hours, 16 reported 20 to 30 hours, and 7 reported more than 30 hours. They received a base pay of $15.00 for completing the experiment and .02 cents per point (average bonus over the 2 set of game of $9.11 – ranging from $0.85 to $24.58).

[10] There were 59 subjects who selected the Mechanical Turk hit but did not start the experiment (dropping out before consent), 51 subjects who did not finish the first 20 games, and 12 subjects who did not return for the second session.

[11] We did not want to eliminate subjects on a learning criterion like in Experiment 1 because such criteria might be differentially biased against some conditions (e.g., it might be harder to improve score in one game or in transfer conditions).

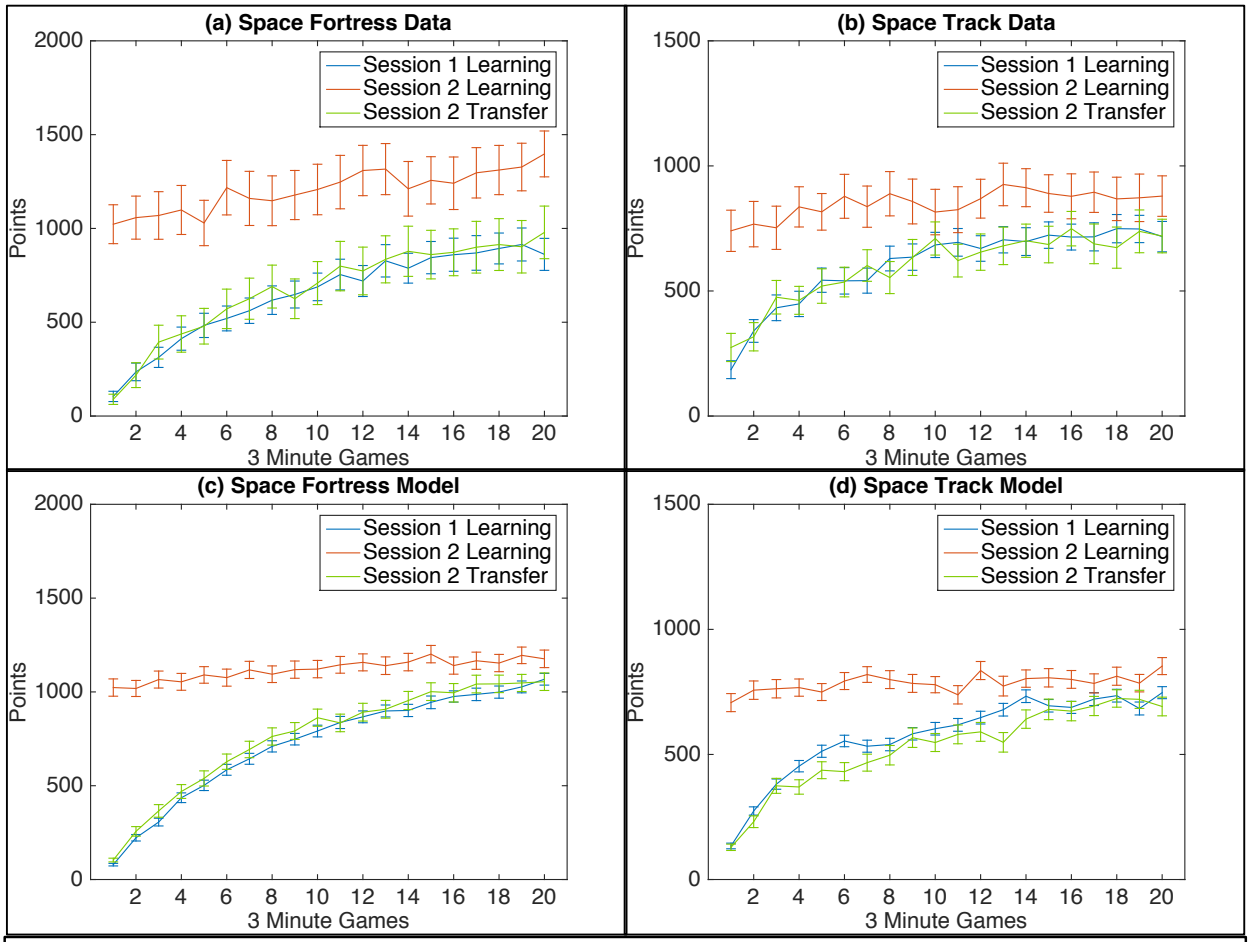


Figure 18. Increase in score as a function of game number in the two sessions. Parts (a) and (b) show the results for the subjects in the two games. Parts (c) and (d) show the results for the models in the two games. Error bars are standard deviations of the mean.

Results

Figures 18a and b show mean points scored as a function of game in a 20-game set in Space Fortress and Space Track. Each graph shows three learning curves. Session 1 Learning plots performance of all subjects playing that game in the first session regardless of whether they would stay in that condition in the second session or transfer to the other game[12]. Session 2 Learning plots performance of those subjects who continue to play the same game in the second session. Session 2 Transfer plots performance of those subjects who are playing a different game

[12] The differences between the two conditions contributing to Learning 1 were not significant ($t(52)=1.54$ for Space Fortress and $t(55)=0.19$, both ps $>.10$).

in the second session than in the first session. Statistics strongly support the inferences that seem apparent in the graphs. All conditions show significant improvements over the 20 games as measured by slope (the least significant slope is for Session 2 learning in Space Track ($t(28)=3.28$, $p < .005$). Looking only at subjects who played the same game in both sessions, subjects are better on the last 5 games on Session 2 than they were on Session 1 (within subject contrasts: $t(25)=5.51$ in Space Fortress, $t(28)=5.91$ in Space track, both $ps<.0001$). Finally and most critically, there is no difference between the Session 1 Learning condition and the Session 2 Transfer for either Space Fortress or Space Track ($t(80)=0.50$ for Space Fortress and $t(83)= 0.13$ for Space Track).

Points are mainly determined by fortress kills and ship deaths in Space Fortress and by rectangles cleared and ship deaths in Space Track. Figures 19a and b show the positive events – kills in Space Fortress and cleared rectangles in Space Track. Subjects show highly significant improvement over the 20 games in all cases (the least significant slope is for Session 2 learning in Space Track ($t(28)=4.41$, $p < .0001$). Subjects end up better for the last 5 games of their second session on the same game ($t(25)=4.30$ in Space Fortress, $p < .0005$, and $t(28)=6.80$ in Space track, $p<.0001$). On the other hand, there is no difference between the Session 1 Learning condition and the Session 2 Transfer for either game ($t(80)=0.57$ for Space Fortress and $t(83)= 0.04$ for Space Track).

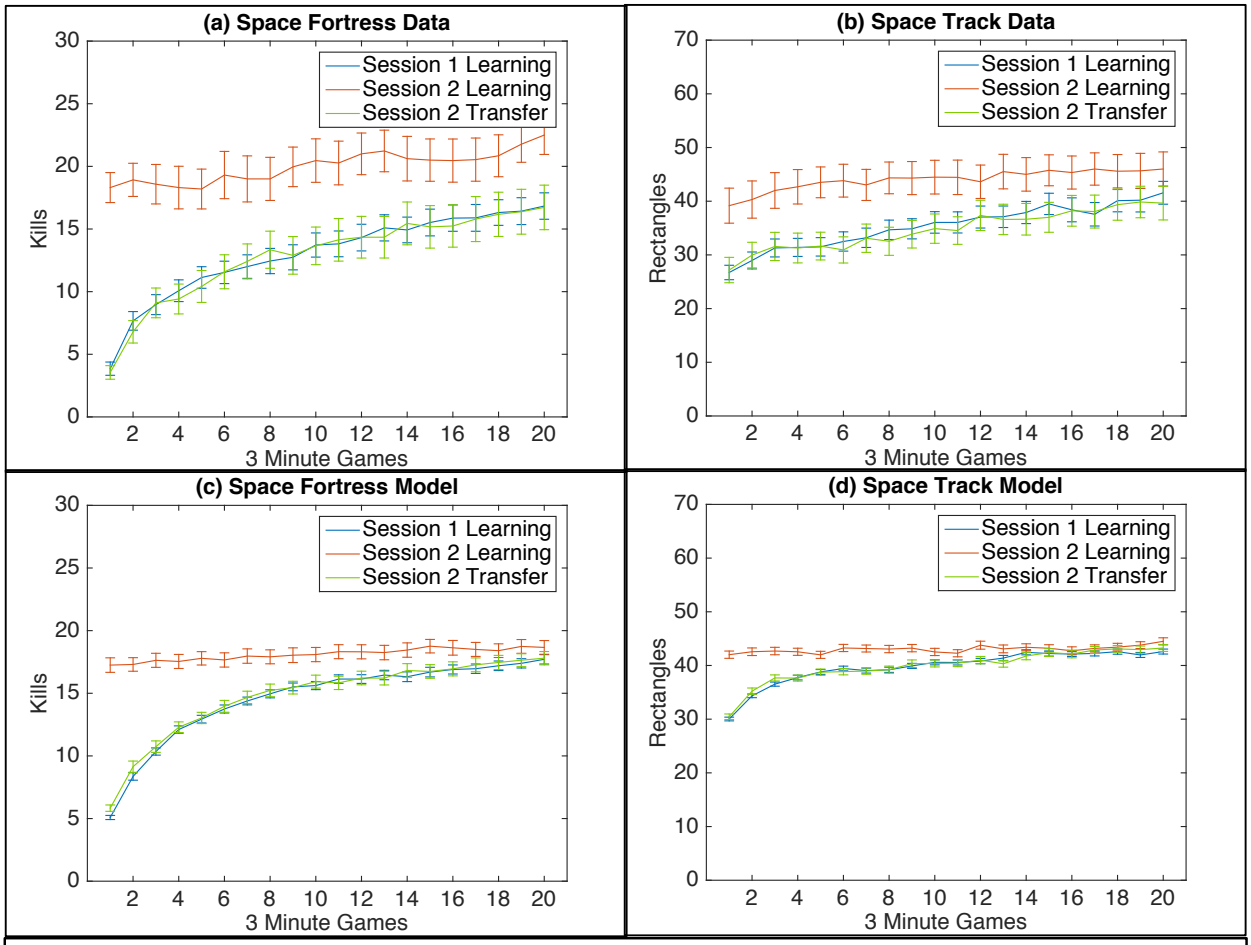


Figure 19. Increase in point-gaining activity (fortress kills in Space Fortress; clearing rectangles in Space Track) as a function of game number in the two sessions. Parts (a) and (b) show the results for the subjects in the two games. Parts (c) and (d) show the results for the models in the two games. Error bars are standard deviations of the mean.

Figures 20a and b show the deaths in the two games. Subjects show highly significant decreases in deaths in both Session 1 Learning and Session 2 Transfer – the least significant for the decreasing curves is Session 2 Transfer in Space Track ($t(27)= -3.50$, $p < .001$). However, there is no significant improvement in the Session 2 Learning curves ($t(25)=-1.39$ for Space Fortress and $t(28) = 0.15$ in Space Track). The improvement in the number of deaths in the last 5 games for those who play the same game in both sessions is between marginal and non-significant ($t(25)=-1.86$ in Space Fortress, $p < .10$ and $t(28)= -0.59$ in Space Track). Thus, subjects achieve most of their decrease in deaths in the first 20 games. Again, subjects coming from the other game show no significant improvement in deaths over subjects playing that game

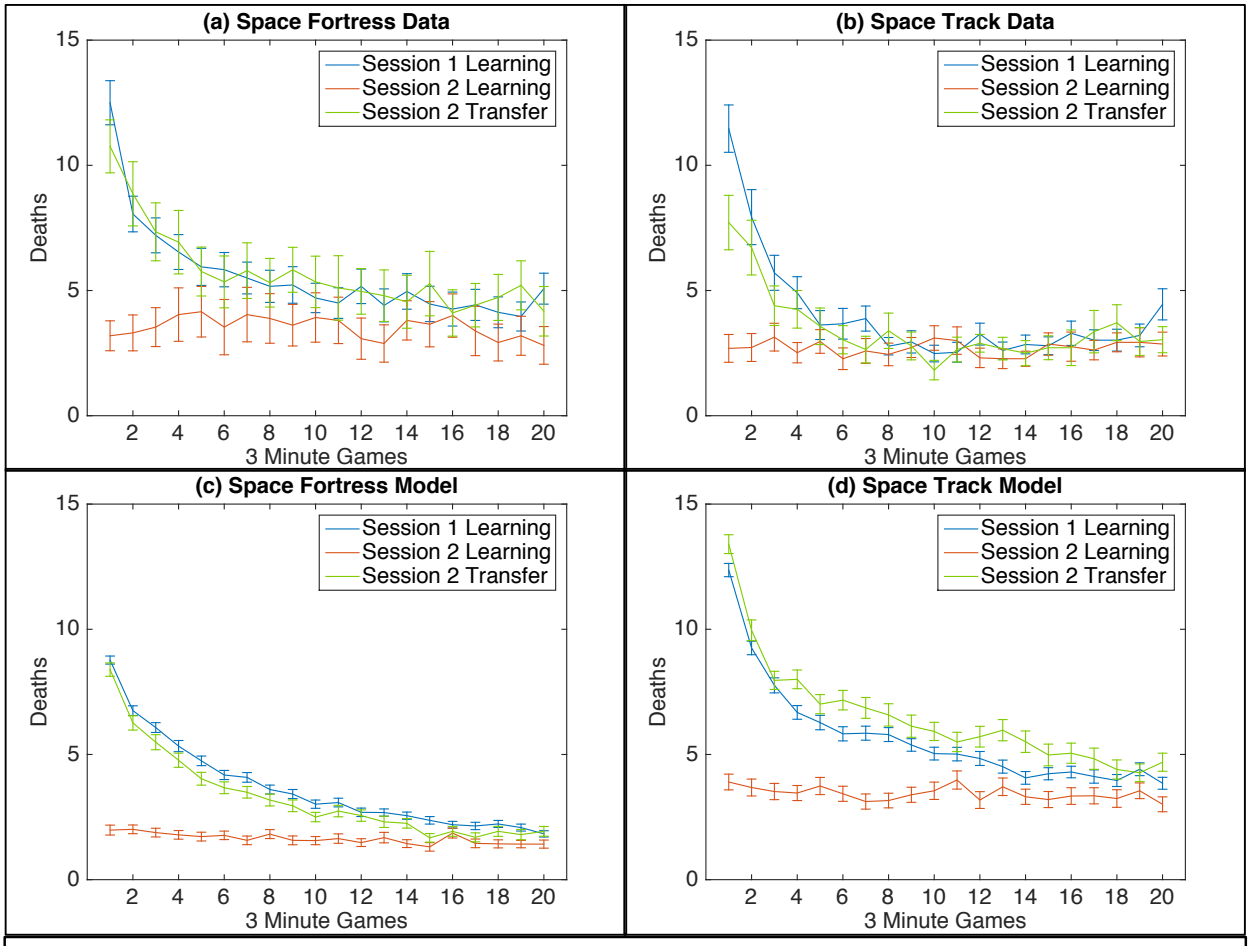


Figure 20. Decrease in ship deaths as a function of game number in the two sessions. Parts (a) and (b) show the results for the subjects in the two games. Parts (c) and (d) show the results for the models in the two games. Error bars are standard deviations of the mean.

on Session 1 ($t(80)=0.50$ for Space Fortress and $t(83)=-0.33$ for Space Track). In Space Track, there does appear to be an advantage for Session 2 Transfer over Session 1 Learning for the first game in Figure 20b. The difference is significant for the first game ($t(83)=2.59$, $p<.01$) but not for any of the other Space Track games in a session. There is a similar first game transfer advantage in deaths for Space Fortress but it is not significant ($t(80)=1.49$, $p>.10$). If we were to correct these first-game tests for the 20 possible single-game comparisons, neither would be significant. Thus, there may be a brief advantage for the transfer players in avoiding deaths but not substantial enough to have a significant impact on points in Figure 18.

The Same Model Plays both Space Fortress and Space Track

Parts c and d of Figures 18-20 show the results of the ACT-R model, which are strikingly similar to the actual data. The results in these figures come from same model as in Experiment with the same parameters. Whether the model played Space Fortress or Space Track depended on what was on the screen, which would result in the retrieval of instructions for one game or the other. The instructions for Space Track were the same YouTurn instructions as Experiment 1, with the 4 strategic variations.

The Space Track instructions are illustrated in Figure 21 and described in Table 2. In brief, these instructions specify first assuring that the ship is flying with the desired aim and speed down the current rectangle. If that is satisfied, the model turns its ship in anticipation of the hard turn at the end of the rectangle. When the model gets close to the end it presses an extended thrust to try to change the direction down the new rectangle. It then applies the same procedure to the new rectangle, which initially corrects any errors from the hard thrust. Highlighted in Figure 21 are the portions of the instructions that require the same use of navigation information as used in parts (b) and (f) of Figure 4.

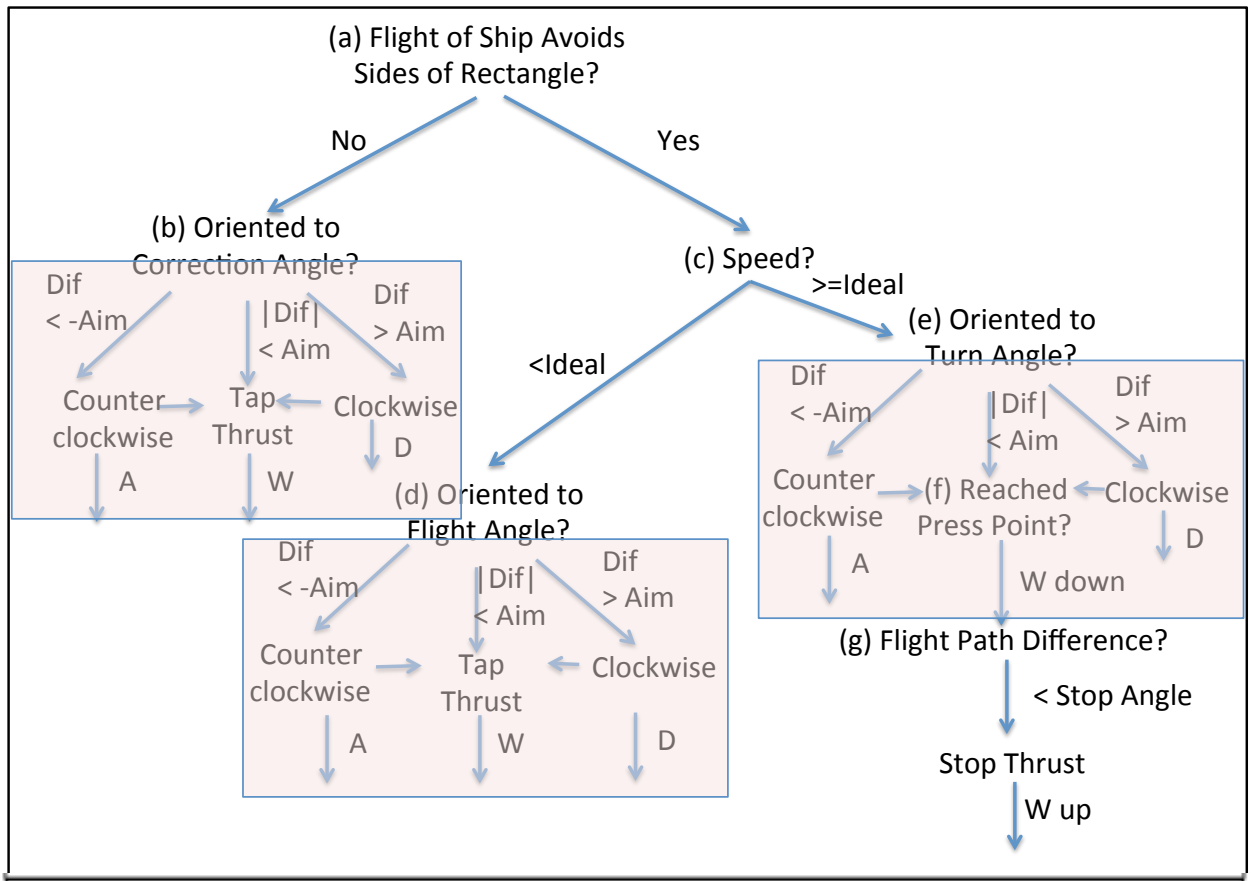


Figure 21. Model's instructions for playing Space Track. Shaded portions contain navigation knowledge shared with Space Fortress.

The instructions represented in Figures 4 and 21 are encoded declaratively. The model starts with 39 productions of which 31 retrieve individual pieces of these instructions and execute them. The other 8 involve recognizing game start up, explosions, and game ending. The 31 instruction-following rules involve 17 rules used in both games (for retrieving instructions, making the comparisons that result in the branching in Figures 4 and 21, for making the turns, and for converting knowledge about the roles of individual keys into actions). There are 14 instruction-following rules that deal with instructions about situations that only come up in one of the games – 11 rules only used in Space Fortress that calculate changes to the Thrust angle and deal with shooting, and 3 rules in Space Track that deal with unique angle calculations and the closed-loop hard thrust.

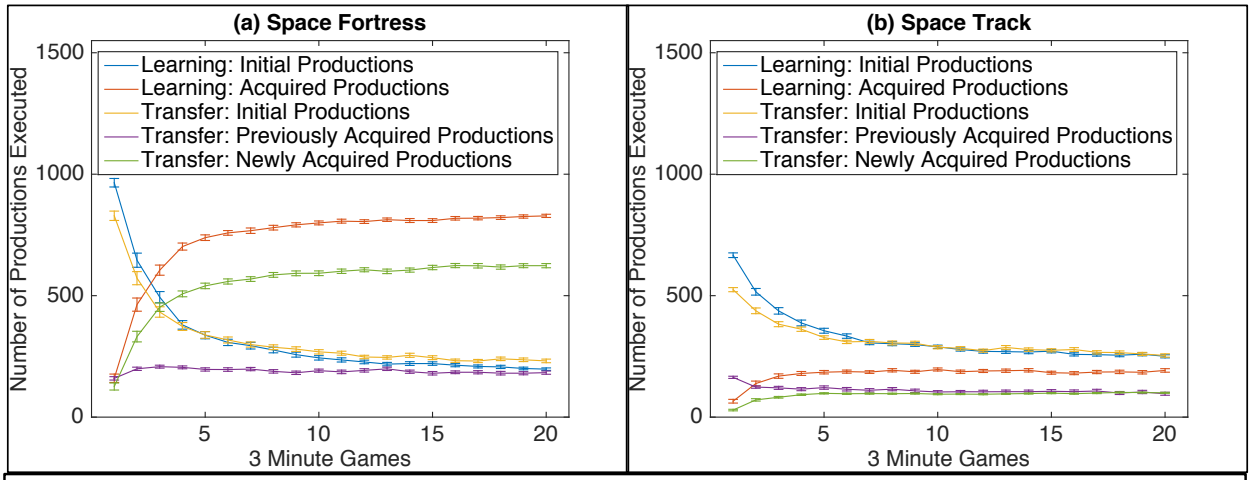


Figure 22. Frequency of production use in the first 20 games (Learning) and the second 20 games in the transfer condition. (a) Space Fortress; (b) Space Track.

As game play progresses many new rules are learned through production compilation eventually resulting in direct-action productions. Because of the shared navigation knowledge and the shared rules for interpreting that knowledge, some of the production rules that are learned in one game can apply to the other game. Figure 22 shows the change in the rule mix over the first 20 games either in the first session (Learning curves) or in the transfer session (Transfer curves). There are 2 kinds of productions for Learning and 3 kinds for Transfer:

1. **Learning Session: Initial Productions**. The rules that the model starts out with for following instructions.
2. **Learning Session: Acquired Productions**. The rules that the model learns in the current game.
3. **Transfer Session: Initial Productions.** The rules that the model started out with in the first session.
4. **Transfer Session: Previously Acquired Productions.** The rules that the model learned in the prior Session 1 game.

5. **Transfer Session: Newly Acquired Productions.** The rules the model that learns in the current transfer game.

In the first session, in both Space Fortress and Space Track, there is a rapid drop in the use of instruction following rules and increase in reliance on learned rules[13]. In the case of transferring to a new game in Session 2, instruction following rules are used nearly as frequently as in the first game of the first session, but again drop off rapidly. Rules transferred from the first game are used with a frequency that is fairly constant across the transfer games. New rules learned in the transfer game come to be used with increased frequency. Twice as many rules are firing in Space Fortress as Space Track, largely reflecting the fact that Space Fortress also involves shooting rules and a shot can be taken every 250 ms. In contrast, in Space Track there are times when the ship is flying as fast as it safely can down the rectangle, leaving the model with nothing to do but wait until it can make the hard turn.

In the transfer games, rules learned in one game constitute about 20% of the rule executions. These shared rules do not necessarily convey an advantage on model performance. At the beginning of the game before the control variables are tuned, these transferred rules can often result in taking the wrong action faster. Since no path from top to bottom is shared between Figures 4 and 21, these shared rules are also not the most efficient direct-action productions. Rather they reflect compilations of the shared steps and need to be further compiled with the unique steps to become direct-action productions for the transfer game.

The instructions for Space Track involved learning 5 control variables. The control variables were:

(a) **Speed.** The minimum speed for flying down a rectangle (choice point c in Figure 21).

[13] If models continue to play the same game in Session 2 (not shown in Figure 22), rule use stays at the same levels as the end of Session 1. Rarely is any new rule learned in Session 2 and used.

(b) **Press Point.** How far into the new rectangle the plane should be before pressing thrust to turn the ship (choice point f).

(c) **Aim.** As in the case of Space Fortress how close the orientation of a ship should be to the desired orientation (choice points b, d, and e).

(d) **Stop Angle.** When holding down the thrust key, how close should the direction of the ship be to the direction of the new rectangle before lifting thrust (choice point g).

(e) **Tap Thrust Duration.** How long a tap thrust should be (action under choice points b and d).

Learning of control variables begins anew when the model transfers to a new game. However, the state of the model is not the same because of the acquired productions that can transfer (Figure 22). Appendix C provides a full summary of the convergence of the control variables for both games in both learning and transfer. There seems very little difference in the convergence between initial learning and transfer.

Although aim and thrust duration are the same control variables as in Space Fortress, the Controller starts their learning over again upon transfer to a new game because the same values are not optimal. Figure 23 assesses whether there is any transfer of these shared control variables from Space Track to Space Fortress[14]. Part (a) and (c) show the accuracy of aim when a shot is taken. Part (b) and (d) show the thrust duration, which also decreases in Session 1 Learning and Session 2 Transfer. In both cases for subjects and models, there is little difference between the learning that happens in Session 1 Learning and Session 2 Transfer.

[14] It is hard to get a measure in Space Track about what the subjects consider an adequate aim because one needs to know the player's intended angle; whereas in Space Fortress it can be taken as the direction to the fortress when a shot is taken. It is also hard to measure in Space Track what the player considers to be the right duration for a tap thrust because there are both tap thrusts and long hard thrusts. There is no need for long thrusts in Space Fortress.

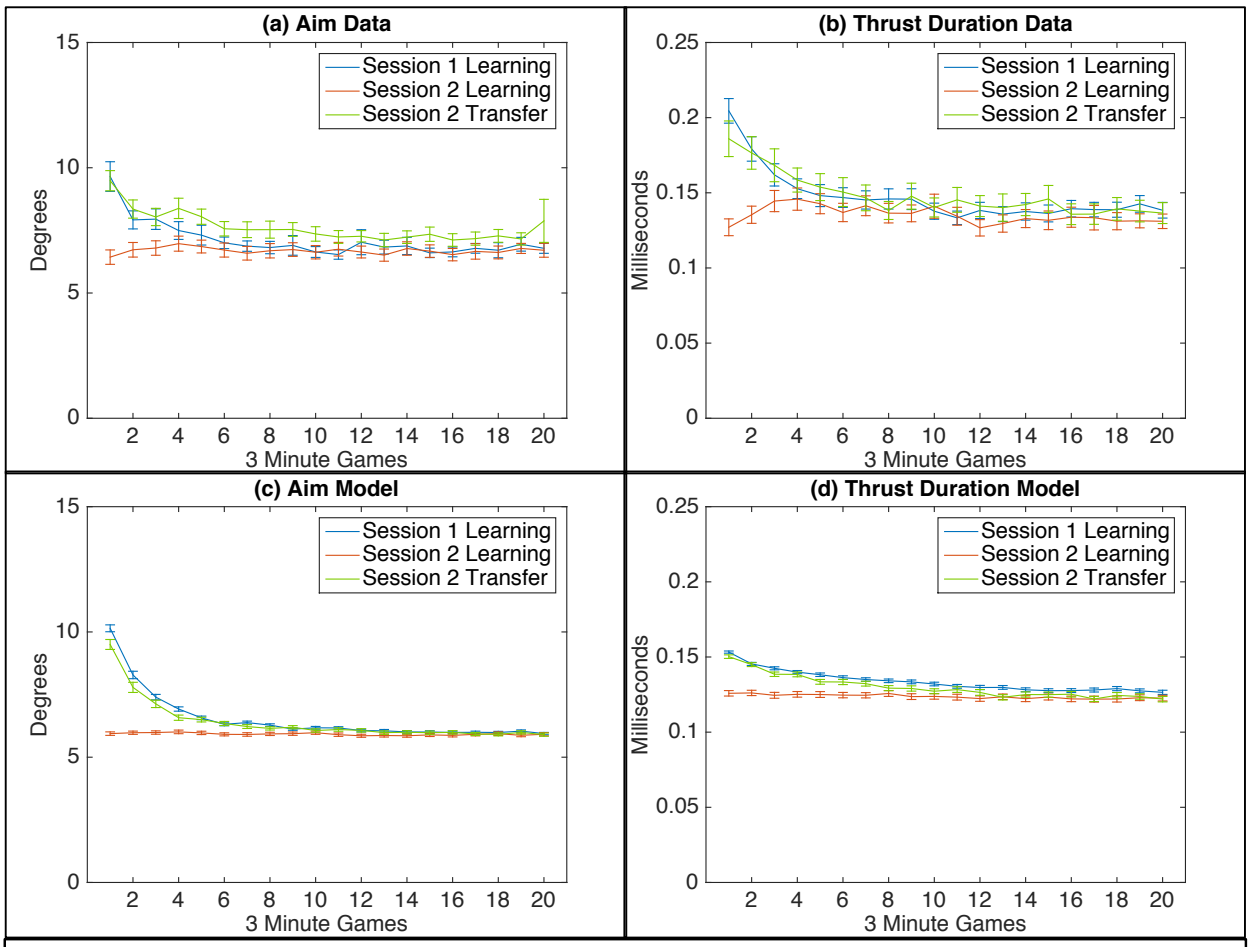


Figure 23. Change in aim and thrust duration in Space Fortress as a function of game number in the two sessions. . Parts (a) and (b) show the results for subjects while Parts (c) and (d) show the results for the models.. Error bars are standard deviations of the mean.

Experiment 2 Summary

Even though good performance in both Space Fortress and Space Track is based on the same knowledge about how to control ship motion, there was virtually no transfer between the games. The game-learning model predicted virtually no transfer in performance measures. The model represents the navigation knowledge in a way that it could be used in both games and as a consequence some of the productions learned in one game did transfer to the other game. However, control variables needed to be tuned to the specific games. Before they became tuned in the transfer game, the transferred production rules often were allowing the model to make the wrong action faster.

This striking failure of transfer would probably not be found in all tasks that require rapid and precise skills. For instance, declarative knowledge discovered in one skill can transfer to another. Had we not provided subjects with any game instruction and they had to discover how to play the game, presumably they could carry such knowledge to the transfer game and would do better at least in the early games. Second, if the goals remain the same and the controls are only slightly changed, it seems likely that the control parameters can become quickly retuned. One example of such a transfer, on which many people depend, is transferring from driving one car on city streets to driving another car. However, our situation was one where the controls stayed the same but the goals changed. With respect to this, it is worth noting that racecar drivers, who are undoubtedly highly skilled at driving on racetracks, do not have better off-track records than normal drivers (Williams & O'Neill, 1974). This is like the failure of transfer observed from Space Track to Space Fortress.

There is research showing transfer from playing video games to general cognitive skills (e.g., Bediou et al., 2018, but not without dispute – e.g., van Ravenzwaaij et al., 2014). There are multiple ways of reconciling our results with these. First, Space Fortress and Space Track may not be of the kind of video game that produces such transfer (Lee et al., 2012). Second, one hour of training is much less than typically used in studies that look for transfer of effects of playing video games. Third, the studies finding positive transfer have focused on transfer to general cognitive skills, not on achieving high levels of performance in another video game. Combining these three points, while general cognitive skills may be important to learning our games, all of the subjects in this transfer study had many prior hours of experience playing other video games. Adding one hour of playing one of our games would add little to whatever general skill they had. What could have transferred but did not transfer was skill at flying in a

frictionless space. Such skill failed to transfer because these flying skills had to be tuned to achieve different goals.

General Discussion

A single model learned to play three games – AutoTurn, YouTurn, and Space Track. Using the same parameterization in all cases, it learns in a way that displays much similarity to human learning, making it a plausible model of human learning. This is the first model in any architecture that learns to achieve human-like performance playing video games[15]. Since the work of Mnih et al. (2015) there have been many efforts to apply deep reinforcement learning methods to the learning of video games. Sometimes the results are agents that play better than humans, sometimes as good, and sometimes still not as good a humans. But the result is never an agent that learns and plays like a human.

While these deep reinforcement learners often come to achieve superior play to humans (in part helped by their super-human response times) they require much more experience than humans to achieve this level. Tsividis et al. (2017) discuss why humans can learn to play Atari video games so much faster than the deep learning systems. As they note, humans (and this is true of the ACT-R models) come into the task with many advantages, including the ability to parse the screen and prior understanding about the games. The ACT-R models also start with instructions about how to play the game. Tsividis et al. (2017, and also Tessler et al., 2017) show such instructions confer a substantial advantage on human players (and this is true for Space Fortress – see Appendix B). However, the performance of the No Tuning and No Compiling models in Figure 16 provides an illustration that abstract knowledge about how to play the game is not enough for ACT-R to achieve high human performance. Humans similarly need to perfect

[15] To see a minute of model play compared to a minute of human play, both late in training, see the videos at http://act-r.psy.cmu.edu/?post_type=publications&p=31435

that knowledge through experience with the task at hand. In the ACT-R models, production compilation and control tuning produced this gradual improvement in skill performance.

A recent effort has applied the deep reinforcement learning approach to Space Fortress (Agarwal et al., 2018). Standard state-of-the-art reinforcement algorithms were not able to learn to play this game. A major obstacle turns on a detail of the game that might seem unimportant: To encourage accurate shooting every shot costs the player 2 points. This means that the reinforcement-learning agent receives a negative reinforcement every time it shoots (see also the Clipped model in Figure 16d). There was success at using reinforcement learning when the reward structure was changed so that every shot that incremented vulnerability received a positive reward. This reinforcement learning approach, similar to the ACT-R model with control tuning, uses relevant features rather than points to evaluate its choices. However, even with this change, it took hundreds of games to reach the level humans achieved in 20 games and thousands of games to exceed that level.

This deep reinforcement learner is learning without receiving instruction (although the change in reward scheme could be considered a sort of instruction). Appendix B reports a pilot study to see if humans could learn to play Space Fortress without instruction. While they were disadvantaged compared to instructed subjects, about half of the human learners were able to figure the game out over the course of the 20 games. It remains a challenge how to model their discovery learning.

The new theoretical addition to the ACT-R architecture is the Controller module, which allows ACT-R to learn a parametric relationship between continuous variables and outcomes. There are a number of parameters underlying the implementation of the Controller module, only one of which, initial temperature, was varied in the simulations. Figure 24 shows the

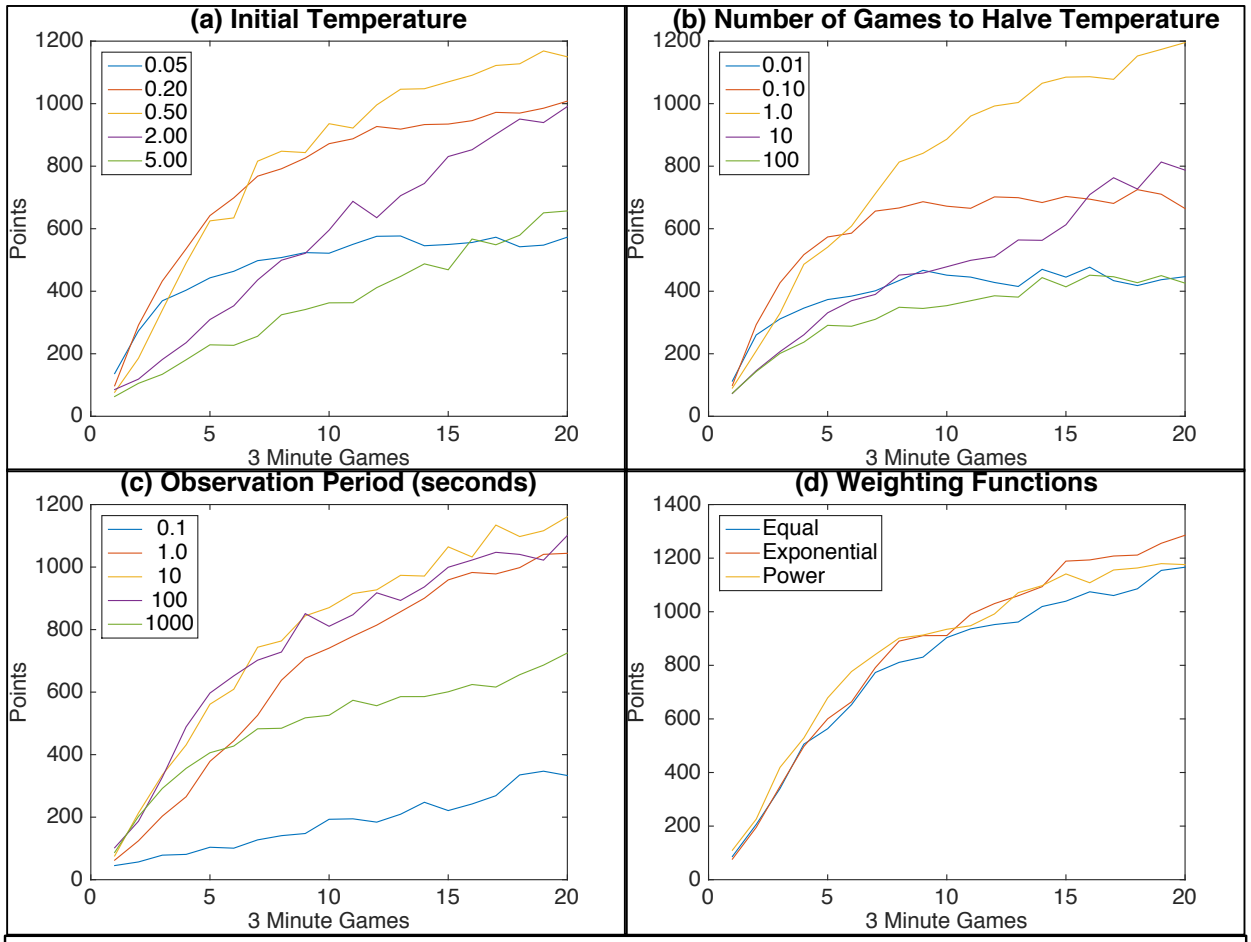


Figure 24. Learning performance when various choices in the Controller module are varied.

conseuqences of variation of four important parameters in Controller module. To go through the four panels in the Figure.

(a) Initial temperature was the one parameter (A in Equation 2b) varied in the simulations (from .1 to 2) as a source of individual differences, where large values mean more variable initial behavior. Figure 24a shows the results that would be obtained over a larger range from .05 to 5. The best performing value for this experiment seems to be around .5. Lower values often result in the model converging too soon on non-optimal values and larger values mean that the model is still not selecting the best values by the end of the game.

(b) The temperature decreases from its inital value A according the the function A/(1+Bt) where t is measured in seconds. In our model data, B was set to 1/180 meaning the temperature is decreased in half by the end of the first game. Figure 24b shows the results that would have been obtained had we changed B so that different durations would cut the temperature in half. The initial temperature in all cases was .5. Of the choices plotted, the setting in the current model produces the best results. More rapid lowering of temperature results in inadequate exploration before focusing on a value and slower lowering prolongs the exploration phase to the point where the model has not settled on good values by the 20^{th} game.

(c) One difference between the Controller module and typical reinforcement learning is it settles on a control value for a period of time and calculates the mean rate of return over that interval. This contrasts with looking at the return after each action and allows the model to get a sense of the longer term consequences of a choice. The duration of these observation periods was exponentially distributed with a mean of 10 seconds. Figure 10c shows the consequence of different mean values for the distribution of observation times. It shows that there is rather equivalent performance for a range of 1 to 100 seconds. longer obsrvation periods like a 1000 seconds mean that the system takes too long to update its control values. Shorter obsrvation periods like a tenth of a second do not provide adequate information about a control setting.

(d) In estimating the quadratic functions the model weights equally all observations from the beginning of the game. It does not discount old experiences which might seem a reasonable policy in some domains. Figure 24d compares this Equal weighting with two options for discounting experiences according to how long ago they were. A

common discount option is to exponentially decay past experiences. Therefore, we tried a Exponential option where the weighting of a past experience was discounted by the value $.999^t$ where t is time in seconds. There is also evidence for a power function discounting in the tracking of continuous values (Elliott & Anderson, 1995). Therefore, we tried a power discounting where the weight of a past experience was $t^{-.5}$ where t is the time in seconds. The three options performed comparably, which is not surprising since the game does not change over time. The two discount possibilities were slightly better, probably because they allowed the system to adapt to its own increased competence in playing the game.

Figure 24 indicates that the choices made in the model lead to good performance in the game. A point for further experimental investigation is how stable these results are in different environments which run at different paces and perhaps change over time.

In concluding we would like to comment on Fitts three-phase characterization of skill learning in the light of the model. As noted in the introduction, it has been used to frame many discussions of skill learning including early presentations of ACT theories of skill learning. While Fitts' three-phase characterization of skill acquisition has often been discussed as a three-stage theory, Fitts did not originally intend it as a stage characterization. As Fitts (1964) wrote

> "it is misleading to assume distinct stages in skill learning. Instead, we should think of gradual shifts in the factor structure of skills" (p. 261).

Nonetheless, even in Fitts' original paper he described skills as being in a particular phase and his characterization has since been used with the understanding that a skill can be treated as in one of the phases at most points in the course of learning. With complex skills, such phase assignments probably only make sense as characterization of parts of the skill. For instance, in

Space Fortress the hitting of keyboard keys is highly automated at the beginning of the game while deciding what keys to hit is slow.

Following the 1982 ACT* characterization (Anderson, 1982) one might map this ACT-R model onto Fitts and Posner 3-phase theory as follows: The interpretation of declarative instruction corresponds to the Cognitive phase of Fitts and Posner's 3-phase theory. Production compilation, which produces direct actions, corresponds to the Associative phase. Control tuning, which produces the highly level of performance, corresponds to the Autonomous phase. However, upon closer inspection it turns out that this mapping does not work, and what is happening in the model is closer to what is suggested by the above quote from Fitts.

The three different processes (knowledge interpretation, production compilation, and tuning) are progressing in parallel for each bit of knowledge that is part of the skill in the ACT-R model. At any point in time some parts of taking a single action may depend on interpreting declarative knowledge and other parts may be a result of compiled productions that skip retrieving declarative instructions. Also, choices for different control variables can be at different levels of tuning reflecting the amount of experience the player is getting on those variables. The control values chosen will jump around over time as part of the exploration process, only slowly converging reliably on optimal values.

The learning in ACT-R nicely instantiates Fitts' description of "gradual shifts in the factor structure of skills". While there is not a truly satisfying mapping of the growth of skills onto his phases, the best choices are probably

1. The Cognitive phase corresponds to the period of time before the model is achieving positive points, which is typically midway through the first game. This is the period when the model is laboriously retrieving and executing declarative instructions.

2. The Associative phase is the period of time when production compilation and control tuning are converging on a final structure.
3. The Autonomous phase would be when the model reaches asymptotic performance.

Under this characterization the Autonomous phase is when the model's learning is complete. The models and the subjects do come close to asymptoting after 40 games in Experiment 2 (Session 2 Learning in Figure 18). The model takes about 60 games to asymptote in YouTurn (the Compiling, Tuning curve in Figure 16b) and subjects may also have reached an asymptotic score had they played that many games. Subjects did appear to asymptote after this much practice in the Fortress-Only condition of Anderson et al. (2011), which is similar to YouTurn. However, that condition and YouTurn are rather simple compared to many tasks including the original Space Fortress. Improvement in the original version of Space Fortress continues for as long as 31 hours (Destefano & Gray, 2016). Interestingly, as Destefano and Gray describe, part of continued improvement involves discovering strategies late in the game that go beyond what is instructed. Players deliberately try to apply what they have discovered about the game much like they did with the initial instructions. This suggests that some of that late learning is really declarative, the kind of learning that one might have thought was in the Cognitive Phase.

Motor learning is another potential kind of learning that might continue beyond where the current models asymptote. ACT-R hits individual keys rather than cording keys where pairs of keys will be hit together. This means that the lower bound on interkey times is about 100 ms. While this is also true of initial subject behavior, some subjects did start to simultaneously press two keys (e.g., thrust and shoot), something that the model never comes to do.

References

Ackerman, P. L. (1988). Determinants of individual differences during skill acquisition: Cognitive abilities and information processing. *Journal of Experimental Psychology: General*, *117*(3), 288-318.

Agarwal, A., Hope, R., & Sycara, K. (2018). Challenges of Context and Time in Reinforcement Learning: Introducing Space Fortress as a Benchmark. *arXiv preprint arXiv:1809.02206*

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review*, *89*(4), 369-406.

Anderson, J. R. (2005) Human Symbol Manipulation within an Integrated Cognitive Architecture, *Cognitive Science, 29*, 313-342.

Anderson, J. R. (2007). *How Can the Human Mind Occur in the Physical Universe?* New York: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M.D., Douglass, S., Lebiere, C., Qin, Y. (2004) An integrated theory of Mind. *Psychological Review, 111,* 1036-1060.

Anderson, J. R., Bothell, D., Fincham, J. M., Anderson, A. R., Poole, B., & Qin, Y. (2011). Brain regions engaged by part-and whole-task performance in a video game: a model-based test of the decomposition hypothesis. *Journal of Cognitive Neuroscience*, *23*(12), 3983-3997.

Anderson, J. R., & Fincham, J. M. (2014). Extending problem-solving procedures through reflection. *Cognitive Psychology*, *74*, 1-34.

Bediou, B., Adams, D. M., Mayer, R. E., Tipton, E., Green, C. S., & Bavelier, D. (2018). Meta-analysis of action video game impact on perceptual, attentional, and cognitive skills. *Psychological Bulletin, 144*(1), 77-110.

Cohen, J. D., McClure, S. M., & Angela, J. Y. (2007). Should I stay or should I go? How the human brain manages the trade-off between exploitation and exploration. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, *362*(1481), 933-942.

Daw, N. D., O'Doherty, J. P., Dayan, P., Seymour, B., & Dolan, R. J. (2006). Cortical substrates for exploratory decisions in humans. *Nature*, *441*(7095), 876-879.

Destefano, M. (2010). *The mechanics of multitasking: The choreography of perception, action, and cognition over 7.05 orders of magnitude* (Doctoral dissertation, Rensselaer Polytechnic Institute).

Destefano, M., & Gray, W. D. (2008). *Progress report on Pygame Space Fortress*. Troy, NY: Rensselaer Polytechnic Institute.

Destefano, M., & Gray, W. D. (2016). Where should researchers look for strategy discoveries during the acquisition of complex task performance? The case of space fortress. In *Proceedings of the 38th Annual Conference of the Cognitive Science Society* (pp. 668-673).

Donchin, E. (1989). The learning-strategies project: Introductory remarks. *Acta Psychologica, 71*(1-3), 1-15.

Elliott, S. & Anderson, J. R. (1995). The effect of memory decay on predictions from changing categories. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 11,* 815-836.

Fitts, P. M. (1964). Perceptual-motor skill learning. *Categories of Human Learning*, *47*, 381-391.

Fitts P.M., Posner M.I. (1967). *Learning and Skilled Performance in Human Performance*. Brock-Cole, Belmont, CA.

Frederiksen, J. R., & White, B. Y. (1989). An approach to training based upon principled task decomposition. *Acta Psychologica, 71*(1-3), 89-146.

Gauthier, I., Tarr, M., & Bub, D. (Eds.). (2010). *Perceptual expertise: Bridging brain and behavior*. OUP USA.

Gopher, D., Weil, M., & Siegel, D. (1989). Practice under changing priorities: An approach to training of complex skills. *Acta Psychologica, 71*, 147–178.

Gray, W. D. (2017). Game-XP: Action Games as Experimental Paradigms for Cognitive Science. *Topics in Cognitive Science*, *9*(2), 289-307.

Jepma, M., & Nieuwenhuis, S. (2011). Pupil diameter predicts changes in the exploration–exploitation trade-off: Evidence for the adaptive gain theory. *Journal of Cognitive Neuroscience*, *23*(7), 1587-1596.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237-285.

Kalish, M. L., Lewandowsky, S., & Kruschke, J. K. (2004). Population of linear experts: knowledge partitioning and function learning. *Psychological Review*, *111*(4), 1072-1099.

Kieras, D. E., & Meyer, D. E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human–Computer Interaction*, *12*(4), 391-438.

Kim, J. W., Ritter, F. E., & Koubek, R. J. (2013). An integrated theory for improved skill acquisition and retention in the three phases of learning. *Theoretical Issues in Ergonomics Science*, *14*(1), 22-37.

Koh, K., & Meyer, D. E. (1991). Function learning: Induction of continuous stimulus-response relations. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *17*(5), 811-836.

Kool, W., Gershman, S. J., & Cushman, F. A. (2017). Cost-benefit arbitration between multiple reinforcement-learning systems. *Psychological Science*, *28*(9), 1321-1333.

Lee, H., Boot, W. R., Basak, C., Voss, M. W., Prakash, R. S., Neider, M., ... & Low, K. A. (2012). Performance gains from directed training do not transfer to untrained tasks. *Acta Psychologica*, *139*(1), 146-158.

Lee, M. D., Zhang, S., Munro, M., & Steyvers, M. (2011). Psychological models of human and optimal performance in bandit problems. *Cognitive Systems Research, 12*(2), 164-174.

Lucas, C. G., Griffiths, T. L., Williams, J. J., & Kalish, M. L. (2015). A rational model of function learning. *Psychonomic Bulletin & Review*, *22*(5), 1193-1215.

Matell, M.S., & Meck, W.H. (2000). Neuropsychological mechanisms of interval timing behavior. *Bioessays, 22*, 94-103.

McDaniel, M. A., & Busemeyer, J. R. (2005). The conceptual basis of function learning and extrapolation: Comparison of rule-based and associative-based models. *Psychonomic Bulletin & Review*, *12*(1), 24-42.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. A., (2013). Playing Atari with Deep Reinforcement Learning. CoRR abs/1312.5. http://arxiv.org/abs/1312.5602

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015. Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529-533.

Moon, J., & Anderson, J. R. (2013). Timing in multitasking: Memory contamination and time pressure bias. *Cognitive Psychology*, *67*(1), 26-54.

Newell, A. (1990). *Unified theories of cognition*. Harvard University Press.

Rasmussen, J. (1986). *Information Processing and Human–Machine Interaction: An Approach to Cognitive Engineering, North-Holland Series in System Science and Engineering, 12*. New York: North-Holland.

Rosenbaum, D. A., Carlson, R. A., & Gilmore, R. O. (2001). Acquisition of intellectual and perceptual-motor skills. *Annual Review of Psychology*, *52*(1), 453-470.

Shebilske, W. L., Volz, R. A., Gildea, K. M., Workman, J. W., Nanjanath, M., Cao, S., & Whetzel, J. (2005). Revised Space Fortress: a validation study. *Behavior Research Methods*, *37*(4), 591-601.

Taatgen, N. A. (2013). The nature and transfer of cognitive skills. *Psychological Review*, *120*(3), 439-471.

Taatgen, N. A., & Anderson, J. R. (2002). Why do children learn to say “broke”? A model of learning the past tense without feedback. *Cognition*, *86*(2), 123-155.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General*, *137*(3), 548-565.

Taatgen, N., & van Rijn, H. (2011). Traces of times past: representations of temporal intervals in memory. *Memory & Cognition*, *39*(8), 1546-1560.

Taatgen, N. A., van Rijn, H., & Anderson, J. (2007). An integrated theory of prospective time interval estimation: The role of cognition, attention, and learning. *Psychological Review*, *114*(3), 577-598.

Tessler, M. H., Goodman, N. D., & Frank, M. C. (2017). Avoiding frostbite: It helps to learn from others. *Behavioral and Brain Sciences*, *40*.

Tsividis, P. A., Pouncy, T., Xu, J. L., Tenenbaum, J. B., & Gershman, S. J. (2017). Human learning in Atari.. In *The AAAI 2017 Spring Symposium on Science of Intelligence: Computational Principles of Natural and Artificial Intelligence*.

van Ravenzwaaij, D., Boekel, W., Forstmann, B. U., Ratcliff, R., & Wagenmakers, E. J. (2014). Action video games do not improve the speed of information processing in simple perceptual tasks. *Journal of Experimental Psychology: General*, *143*(5), 1794-1805.

VanLehn, K. (1996). Cognitive skill acquisition. *Annual Review of Psychology*, *47*(1), 513-539.

Williams, A. F., & O'Neill, B. (1974). On-the-road driving records of licensed race drivers. *Accident Analysis & Prevention*, *6*(3-4), 263-270.

Wilson, R. C., & Niv, Y. (2011). Inferring relevance in a changing world. *Frontiers in Human Neuroscience*, *5*.

Wolpert, D. M., Diedrichsen, J., & Flanagan, J. R. (2011). Principles of sensorimotor learning. *Nature Reviews Neuroscience*, *12*(12), 739-751.

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

Zhang, S., & Yu, A. (2013). Cheap but clever: human active learning in a bandit setting. In *Proceedings of the 35th Annual Conference of the Cognitive Science Society* (pp. 1647-1652).

Table 1: Branching Operators in Figure 4

(a) Time to Outer Hexagon? Subjects need to decide whether they can shoot or need to thrust to keep the ship from drifting into the outer hexagon. The operator specifies making this decision on the basis of the time before the ship will reach the outer hexagon. If that time is greater than a threshold, the operator specifies shooting and if it is less it specifies thrusting.

(b) Aim? The ship needs to be aiming at the fortress for a shot to hit it. The operator specifies turning to better aim if the difference between the ship's orientation and a direct aim (**Dif** in Figure 4) is greater than a threshold (**Aim** in the figure).

(c) Delay? This operator specifies waiting until the time (in terms of number of ticks) since the last shot exceeds a threshold before shooting again.

(d) Vulnerability? This operator specifies taking a single shot if the vulnerability of the fortress is less than 10 or taking a double shot to destroy the fortress if the vulnerability is 10 or greater.

(e) Speed? Subjects are instructed to pay attention to the speed of their ship in determining how to orient their ship before thrusting. The model is developing a sense of a **thrust angle** (see Figure 1b) that will maintain the ship's speed. This operator specifies decrementing the thrust angle if the ship is flying slower than ideal and incrementing the thrust angle if the ship is flying faster.

(f) Oriented to Thrust Angle? The operator specifies turning if the difference between the ship's orientation and the desired thrust angle is greater than a threshold.

Table 2: Branching Operators in Figure 21

(a) Flight of Ship Avoids Sides of Rectangle? Players must monitor whether their ship is going to fly into the side of the rectangle before reaching the end. This operator tests for this condition and branches to correcting the path if necessary or otherwise goes on to make other tests.

(b) Oriented to Correction Angle? To change the direction of the ship, the model calculates a correction angle[16]. If the difference between its orientation and this correction angle is greater than its aim threshold, this operator corrects the angle before issuing a tap thrust.

(c) Speed? This operator specifies speeding up if the ship is flying slower than its threshold for a desired speed and preparing for the hard turn if it is flying faster.

(d) Oriented to Flight Angle? In thrusting to increase the speed of the ship, the ship should be oriented in the direction of the rectangle. If the difference between its orientation and the direction of the rectangle is greater than its aim threshold, this operator corrects the angle before issuing a tap thrust.

(e) Oriented to Turn Angle? To make a hard turn the ship needs to be oriented in a direction that bisects the angle between the current rectangle and the next rectangle. This operator specifies turning the ship to achieve this turn angle.

(f) Reached Press Point? This operator specifies beginning a sustained thrust when the ship reaches a threshold distance with respect to entering the future rectangle.

(g) Flight Path Difference. As the thrust key is held down the angle of flight will change towards that of the future rectangle. This operator specifies releasing the thrust key when the difference between the flight angle and the direction of the future rectangle falls below a stop angle. It takes some time for the decision to lift the finger to result in and release the key. Thus, the release decision action needs to be taken in advance of the flight path lining up with the future rectangle.

[16] If the desired direction is *a* degrees to the left or right it adds thrust at $a+(180-a)/2$ degrees to the left or the right.

Table B1. Mean Values and Standard Error of the Means

(a) First 10 Games		AutoTurn		YouTurn	
		Instructed	Uninstructed	Instructed	Uninstructed
Scoring	Points	1867 +/- 87	712 +/- 180	365 +/- 56	128 +/- 70
	Kills	27.5 +/- 1.1	12.7 +/- 2.4	9.6 +/- 0.9	4.8 +/- 1.5
	Deaths	2.4 +/- 0.6	6 +/- 1.9	9.3 +/- 0.8	13.5 +/- 1.5
Shooting	Shots	383.5 +/- 11.6	324.1 +/- 29	201.3 +/- 13.9	229 +/- 31.6
	Resets	5.9 +/- 1.9	29.3 +/- 10.8	10.6 +/- 3.9	28.3 +/- 11
	Misses	0.06 +/- 0.03	0.03 +/- 0.03	38 +/- 3.1	57.2 +/- 7.9
	Aim	0.37 +/- 0.02	0.37 +/- 0.02	8.6 +/- 0.5	18.7 +/- 5
Navigation	Distance	104.4 +/- 2.5	108 +/- 3.8	127.7 +/- 1.5	123.3 +/- 2.4
	Thrust Angle	100.5 +/- 0.9	97.1 +/- 1.9	95.3 +/- 1.3	84.6 +/- 4.9
	Speed	1.11 +/- 0.02	1.09 +/- 0.03	1.41 +/- 0.03	1.33 +/- 0.05

(a) Second 10 Games		AutoTurn		YouTurn	
		Instructed	Uninstructed	Instructed	Uninstructed
Scoring	Points	2127 +/- 91	1124 +/- 251	678 +/- 77	384 +/- 136
	Kills	30 +/- 1.2	17.5 +/- 3.2	13.8 +/- 1.1	8.2 +/- 2.3
	Deaths	2.2 +/- 0.6	5.5 +/- 2.1	6.9 +/- 0.8	11.1 +/- 1.9
Shooting	Shots	402.6 +/- 13.8	328.4 +/- 33.3	252.9 +/- 17.2	240 +/- 40.7
	Resets	4.1 +/- 0.5	22.9 +/- 10.1	12.6 +/- 7	32.8 +/- 17.4
	Misses	0.02 +/- 0.01	0.0 +/- 0.0	36 +/- 3.1	50.2 +/- 12
	Aim	0.4 +/- 0.02	0.41 +/- 0.03	8.2 +/- 0.7	18.5 +/- 5
Navigation	Distance	97 +/- 2.7	97.8 +/- 5.6	120.8 +/- 2.1	119.1 +/- 4.4
	Thrust Angle	101 +/- 1	94.1 +/- 3.2	98.2 +/- 1.3	85.8 +/- 5.7
	Speed	1.12 +/- 0.02	1.1 +/- 0.03	1.31 +/- 0.03	1.22 +/- 0.06

Figure Captions

Figure 1. (a) The Space Fortress screen, showing the inner and outer hexagon, a missile shot at the fortress, and a shell shot at the ship. The dotted lines illustrate an example path during one game. (b) A schematic representation of critical values for shooting and flight control.

Figure 2. Growth in mean number of points earned per game. Error bars reflect one standard deviation of the population.

Figure 3. The module structure of ACT-R. The boxes represent high-capacity, independent modules, which process information in a parallel. The central Procedural module can recognize patterns in module buffers and recommend cognitive and motor actions.

Figure 4. A representation of the instructions in YouTurn. Each set of branching arrows is an operator going from one state to the next. Except for the bottom operators they test conditions to determine the next state. The bottom operators perform actions.

Figure 5. Pseudocode for two productions that interpret instructions and examples of buffer contents that would evoke these productions: (a) A production that performs a motor action. (b) A production that performs a comparison of a game quantity with a control variable.

Figure 6. Pseudocode for a direct-action production that issues a single shot and an example of buffer contents that would evoke its application.

Figure 7. An example of estimating a payoff function from the first game in Space Fortress. Each dot reflects the mean payoff for a sampled value. The function is the best fitting quadratic to these points weight by the duration of time over which they are estimated.

Figure 8. (a) The quadratic functions estimated for different delays at different points in the one sequence of model runs through 20 learning games. (b) The probability of choosing different delays after different numbers of games.

Figure 9. A comparison of subject and model performance: Error bars reflect one standard deviation of the population.

Figure 10. A comparison of subjects and models on two critical factors determining score. Parts (a) and (b) show growth in mean number of kills over games. Parts (c) and (d) show decrease in mean number of deaths over games. Parts (e) and (f) show the average number of the 3 kinds of deaths per game. Error bars reflect one standard deviation of the population.

Figure 11. Shot Timing in AutoTurn: (a) Mean intershot duration per game for subjects and models; (b) Mean number of resets per game for subjects and models; (c) Probability of choosing different time thresholds, averaged over the 100 models. Error bars in (a) and (b) reflect one standard deviation of the population.

Figure 12. Aiming in YouTurn: (a) Mean offset of aim at shot per game for subjects and models; (b) Mean proportion of misses per game for subjects and models; (c) Probability of choosing different maximum aim offsets, averaged over the 100 models. Error bars in (a) and (b) reflect one standard deviation of the population.

Figure 13. Distance in AutoTurn: (a) Mean distance from fortress at which subjects and models chose to thrust; (b) Probability of choosing different time thresholds, averaged over 50 models; (c) Probability of choosing different distance thresholds, averaged over 50 models. Error bars in (a) reflect one standard deviation of the population.

Figure 14. Thrust Duration in AutoTurn (a) and YouTurn: (b) Choice of thrust durations in the models for AutoTurn (c) and YouTurn (d). Error bars in (a) and (b) reflect one standard deviation of the population.

Figure 15. Thrust angle in YouTurn: (a) Mean thrust angle per game for subjects and models; (b) Mean speed of ship for subjects and models; (c) Probability of choosing different thrust angles, averaged over the 100 models. Error bars in (a) and (b) reflect one standard deviation of the population.

Figure 16. Performance of various model variants in YouTurn. Parts (a)–(c) show a crossing of a tuning factor and a compiling factor. Part (d) shows the behavior of three models that deviate from the architectural assumptions of the model in specific ways.

Figure 17. Schematics of the two games: (a) In Space Fortress one is flying a ship between two hexagons shooting at the fortress in the middle. (b) In Space track one is flying a ship down a rectangle and making a turn to fly down the next rectangle.

Figure 18. Increase in score as a function of game number in the two sessions. Parts (a) and (b) show the results for the subjects in the two games. Parts (c) and (d) show the results for the model in the two games. Error bars are standard deviations of the mean.

Figure 19. Increase in point-gaining activity (fortress kills in Space Fortress; clearing rectangles in Space Track) as a function of game number in the two sessions. Parts (a) and (b) show the results for the subjects in the two games. Parts (c) and (d) show the results for the model in the two games. Error bars are standard deviations of the mean.

Figure 20. Decrease in ship deaths as a function of game number in the two sessions. Parts (a) and (b) show the results for the subjects in the two games. Parts (c) and (d) show the results for the model in the two games. Error bars are standard deviations of the mean.

Appendix A: Game Implementation

Here is the pointer to web versions of the games subjects played, which contain the instructions:

http://andersonlab.net/sf-paper-2017/turning.html

Space Fortress

There have been many implementations of Space Fortress over the decades. The development prior to the Pygame version is described in Shebilske et al. (2005) and the original Pygame implementation is described in Destefano (2010). The most consequential change in the Pygame implementation over previous implementations was replacing the use of the joystick for navigation by key presses. We considerably simplified the game from the Pygame implementation. We eliminated mines and bonus points, both of which were aspects of the original game. We also made hitting the hexagon boundaries fatal rather than just a loss of control points. We also simplified the death rules so that being shot or hitting the boundary immediately resulted in death rather than building up towards death.

To review the critical parameters of the game:

1. The game updates every 30th of a second (game tick).
2. The total screen is 710 pixels wide by 626 pixels high.
3. The fortress is centered at x=355, y=315.
4. The ship starts at x=245, y=315, moving up .5 pixel and to the right .866 pixels per game tick. Thus, its initial speed is 1 pixel per game tick.
5. The ship starts aimed horizontally aimed at the fortress. The ship automatically turns to maintain aim in AutoTurn, but subjects must turn the ship in YouTurn.

6. Every game tick that the thrust key is held down, the velocity vector will change .3 pixels in whatever direction the ship is pointing.
7. Every game tick that a turn key is held down, the ship will turn 6 degrees.
8. If the ship does not traverse 10 degrees of arc around the fortress in a second the fortress will shoot a shell at it.
9. For purposes of determining when the ship or fortress is hit, everything is treated as a circle:
    a. The ship has a radius of 10 pixels,
    b. The fortress has a radius 18 pixels,
    c. Shells have a radius of 3 pixels,
    d. Missiles have a radius of 5 pixels.
10. A collision occurs if two circles intersect on a game tick.
11. Shells travel at 6 pixels per tick and missiles travel at 20 pixels per tick.

Space Track

The ship controls and responses are the same in Space Track as in Space Fortress. The display always has 2 rectangles: the "current" rectangle, which is the rectangle the ship is in and the "next" rectangle, which is the rectangle the ship must fly to. The rectangles are built around two line segments, AB and BC, which share the common point, B. The "current" rectangle is built around line segment AB and the "next" rectangle is built around line segment BC. A line segment is 363 pixels long. The rectangle is 473 pixels long because it extends past the line segment's end points by 55 pixels. Rectangles are 110 pixels wide.

If the ship is in the "current" rectangle, it remains alive. If the ship is in neither rectangle, it explodes and the player loses 100 points. When the ship explodes it's position is reset back to point A with an orientation matching line segment AB's angle and a speed of 1.0. If the ship is not in the "current" rectangle but it's in the "next" rectangle, then a new rectangle is created and the player is awarded 25 points.

To create a new rectangle, the "current" rectangle is removed and the "next" rectangle becomes the "current" rectangle. Then a new line segment is created. The new line segment starts at point C. To create the line segment's end point, point D, the game picks a random "base angle" between 30 and 150 degrees and a random "parity", either -1 or 1. The angle for the new line segment CD is the line segment BC's angle plus the "base angle" multiplied by the "parity". Point D is calculated as:

D_x = C_x + 363 * cos(CD_angle)

D_y = C_y + 363 * sin(CD_angle)

Finally, the new "next" rectangle is created around line segment CD.

Appendix B: Comparison of All Instructed Subjects with Uninstructed Subjects

We ran 20 Mechanical Turk subjects without instruction in both AutoTurn and YouTurn. AutoTurn subjects were only told that the relevant keys were **W**and **spacebar** while YouTurn subjects were told that the relevant keys were **A**, **W**, **D**, and **spacebar**. To keep mean compensation comparable to the instructed subjects, we doubled the conversion of points earned for bonus payment. For comparison we will use all subjects who were run with instruction, rather than just those filtered to show signs of learning, who were the focus in the main paper. Table B1 reports average values for the measures of performance reported in the paper. We have broken this out into performance during the first 10 games and the last 10 games.

In terms of mean number of points, uninstructed subjects clearly are showing learning, but they are also scoring less than instructed subjects. (first half AutoTurn, $t(70) = 5.78$, $p < .0001$; second half AutoTurn, $t(70) = 3.76$, $p < .0005$; first half YouTurn, $t(83) = 2.64$, $p < .01$; second half YouTurn, $t(83) = 1.88$, $p < .1$; all 2-tailed). The individual variability in both populations is huge. Nonetheless, we can conclude that without instructions at least some game players can learn to play these games. A measure of having some mastery by the end of the game is averaging more than 100 points in the last two games. 48 of the 52 instructed AutoTurn subjects achieved this measure of mastery, while just 13 of the 20 uninstructed AutoTurn subjects did ($\chi^2=27.90$, $p < .0001$). 50 of the 65 instructed YouTurn subjects did while just 7 of the 20 uninstructed YouTurn subjects did ($\chi^2=12.17$, $p < .0005$).

Appendix C: Model Learning of Control Variables in Experiment 2

When the model transfers to a new game it starts learning control variables for that experiment. Even control variables that are shared start their learning over. Although the learning begins anew, the model state is not the same because of learned productions (Figure 22). So it is of interest to compare the learning of control variables for a game when there are no learned productions with the transfer conditions when there are learned productions. Figures C1 – C4 provide information about the learning of control variables in the same form as Figures 11-15. They show the average probability of selecting different values for each of the control variables after playing different numbers of games. These are averaged over the 100 runs. Rises at the low end or high end of the range of control values reflect cases where some models get locked into poor choices. Control variables that show a strong intermediate peak are ones for which the feedback provides clear signals. They are variables that are particularly important to performance in that game.

Figure C1 shows the learning of the 6 control variables in Space Fortress in the Learning condition. Since the model played 40 games in the Learning condition, we can see the impact of 20 more games than in Figures 11-15. For comparison, Figure C2 shows the learning of these control variables over the 20 games in the Transfer condition when Space Fortress is the game being transferred to. There are strong correspondences between Learning and Transfer for the tuning of delay between shots (parts a), aim (parts b), and thrust angle (parts f). These are also control variables where most models succeed in identifying good values. The two thresholds for thrusting, either time to outer border (parts c) or distance to outer border (parts d) are not well identified and are not very consistent between Learning and Transfer. There does seem to be some better success in identifying a time threshold in Transfer (Figure C2c).

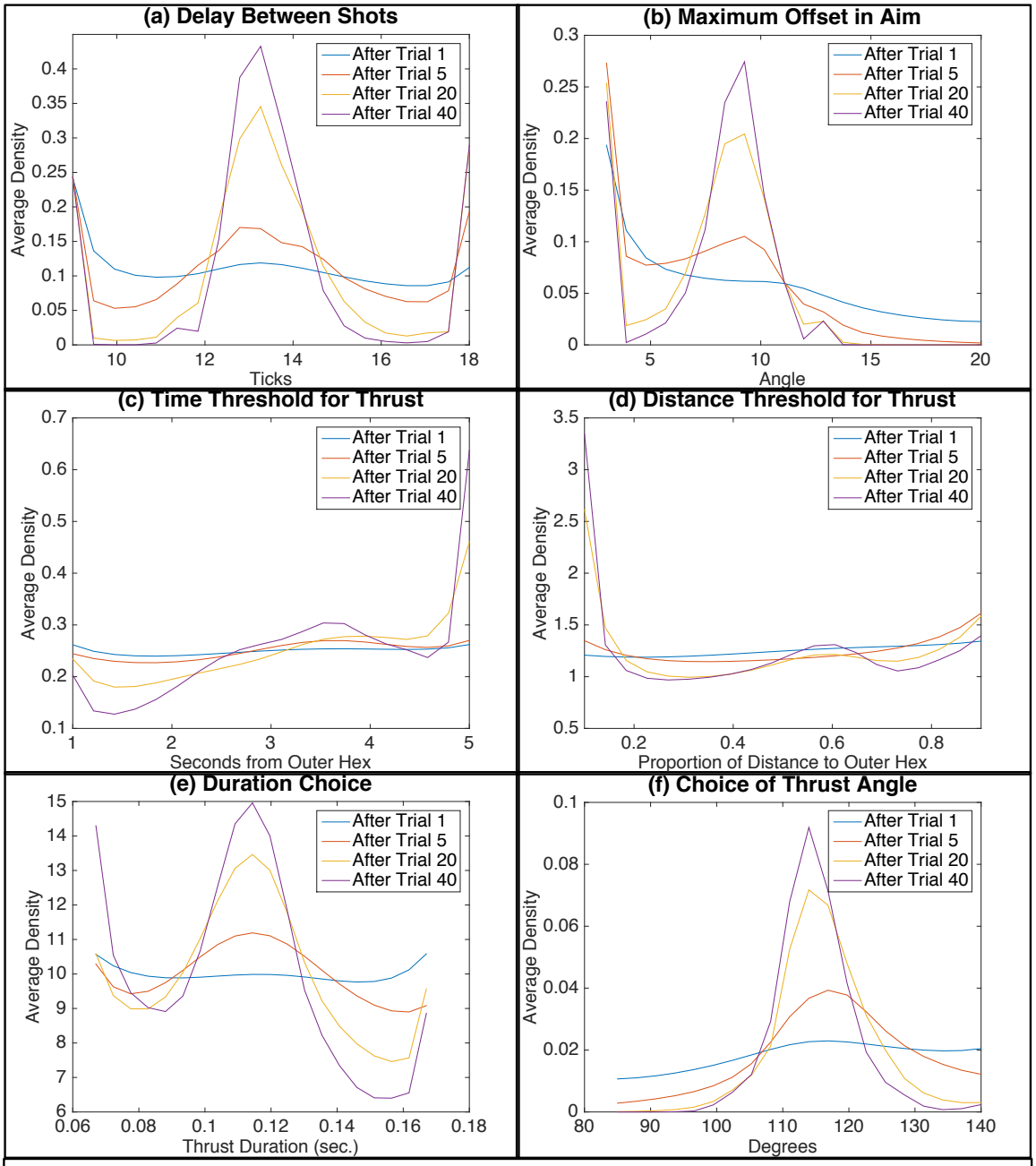
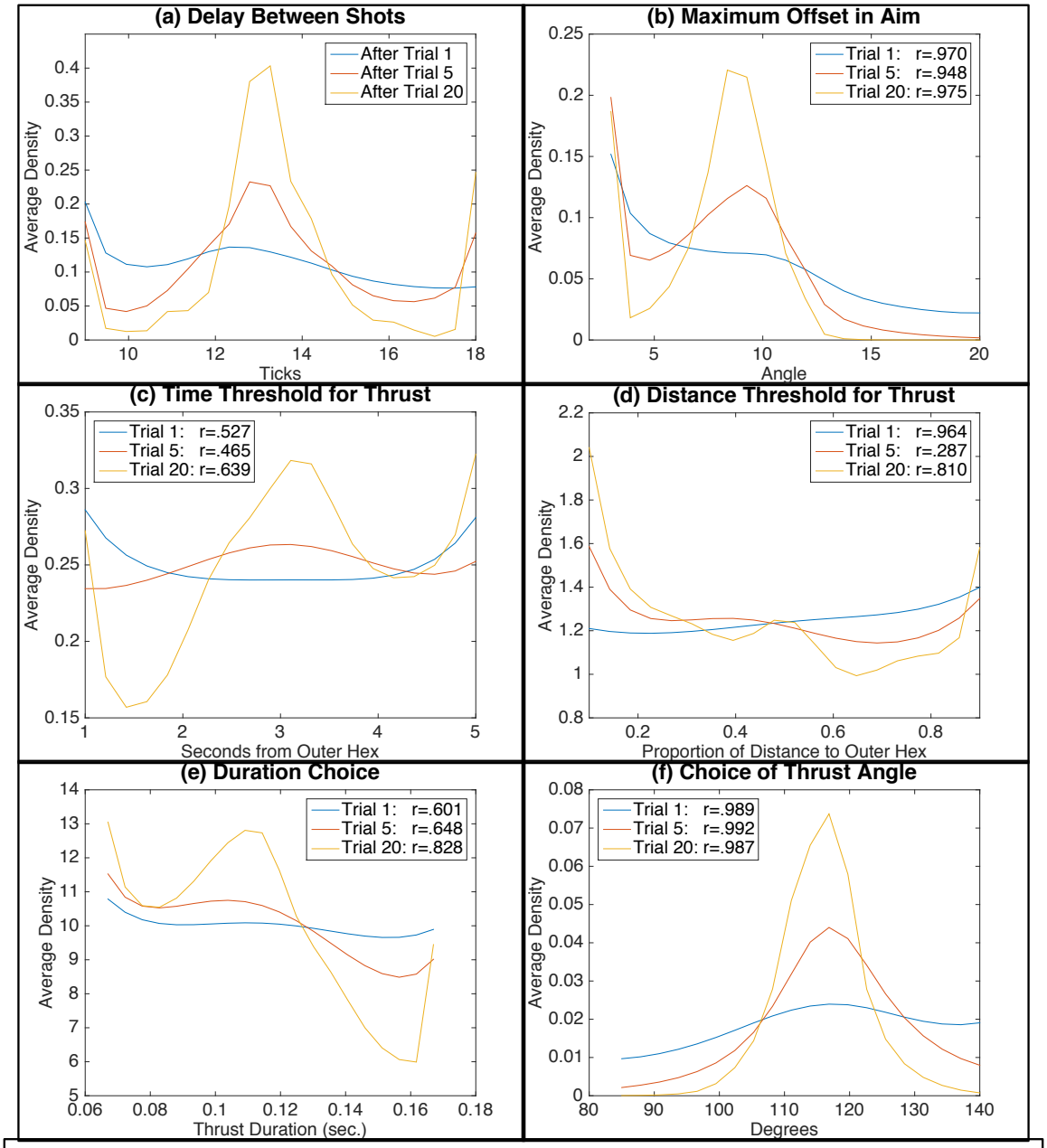


Figure A1. Probability of different values for control variables, averaged over the 100 models that played 40 Learning games in Space Fortress.



Probability of different values for control variables, averaged over the 100 models that played 20 Transfer games in Space Fortress. Correlations are with the corresponding curves in Figure A1.

Figure C3 shows the learning of the 5 control variables in Space Track over the 40 Learning games and Figure C4 shows their learning over the 20 Transfer games. There are strong correlations between Learning and Transfer for the target speed that the model tries to reach going down the rectangle (parts b), the point at which the model issues a hard press to turn into the next rectangle (parts c), and the angle at which the model lifts its finger during to stop

the hard press (parts d). The models are more successful in finding a good aim angle in the Learning condition than the Transfer condition (Figure C3a versus Figure C4a). The correlations on thrust duration are comparable to those for Space Fortress (parts e in all figures).

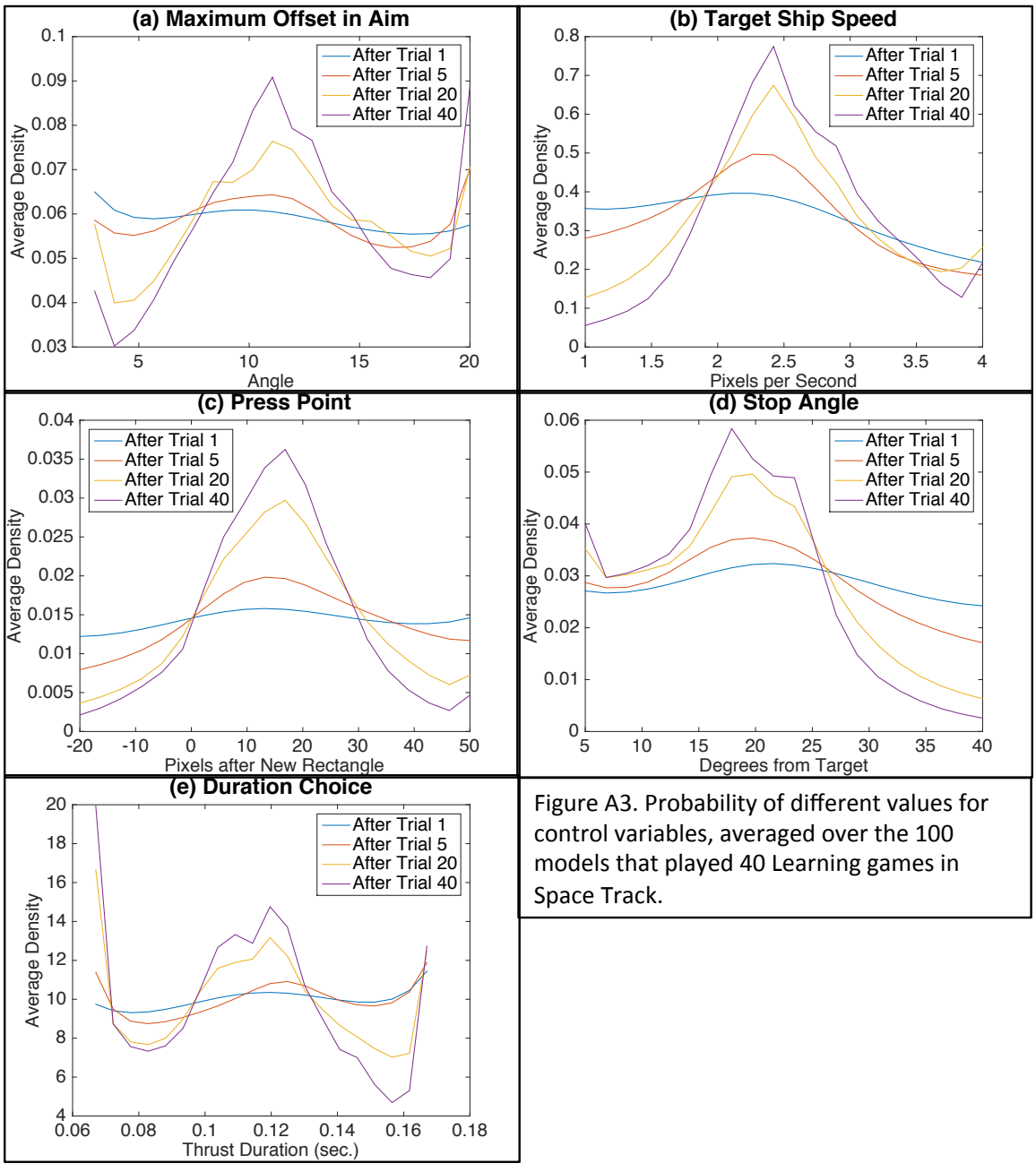


Figure A3. Probability of different values for control variables, averaged over the 100 models that played 40 Learning games in Space Track.

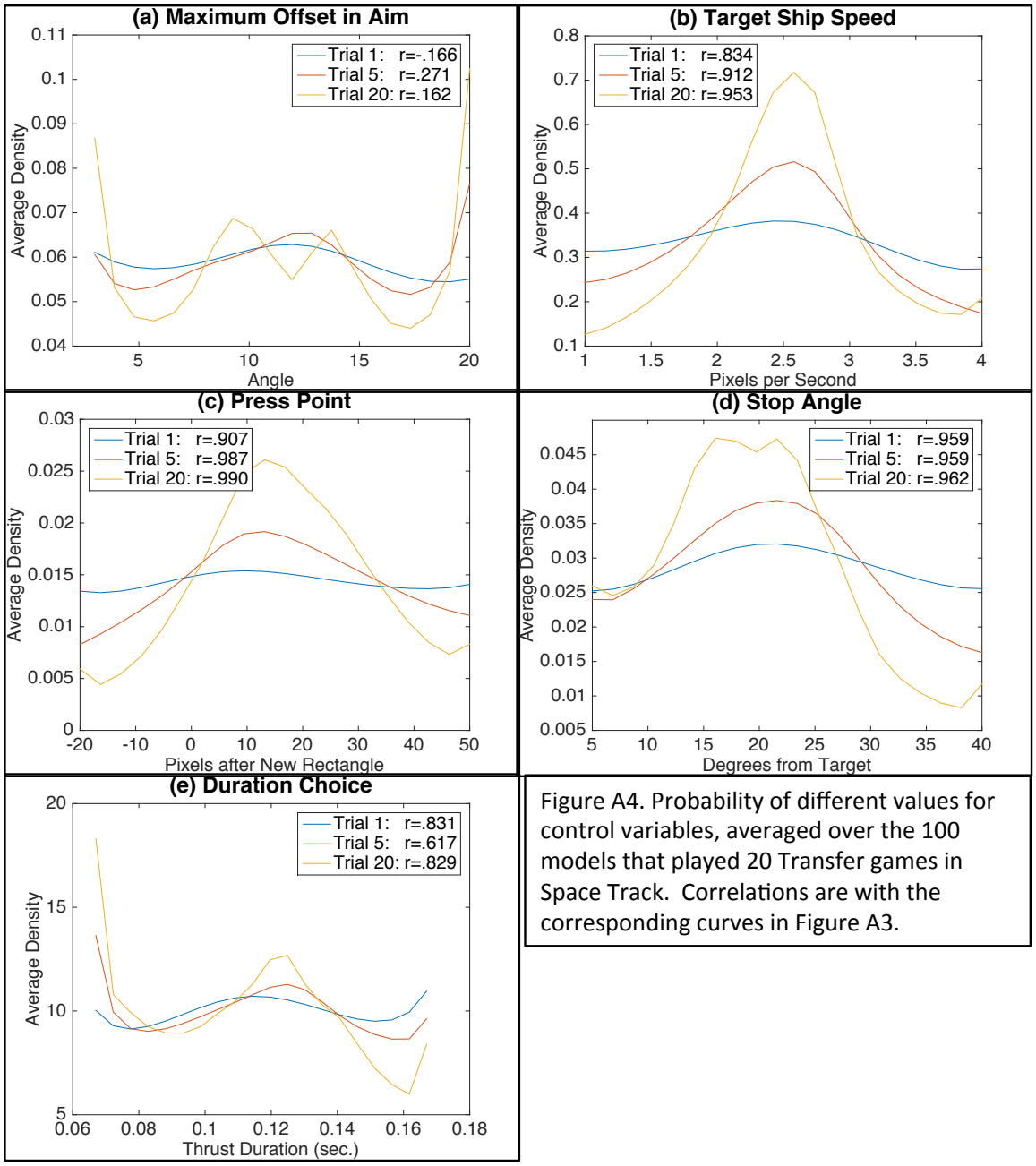


Figure A4. Probability of different values for control variables, averaged over the 100 models that played 20 Transfer games in Space Track. Correlations are with the corresponding curves in Figure A3.

There may be some differences between the learning of the control variables in the Learning and Transfer conditions. In the Transfer condition the model has learned some productions to more rapidly act on them. However, the differences are relatively small. In either Learning or Transfer, control variables for which there are clear feedback signals are being well identified and correlate highly. ‘

Figure Captions

Figure C1. Probability of different values for control variables, averaged over the 100 models that played 40 Learning games in Space Fortress.

Figure C2. Probability of different values for control variables, averaged over the 100 models that played 20 Transfer games in Space Fortress. Correlations are with the corresponding curves in Figure C1.

Figure C3. Probability of different values for control variables, averaged over the 100 models that played 40 Learning games in Space Track.

Figure C4. Probability of different values for control variables, averaged over the 100 models that played 20 Transfer games in Space Track. Correlations are with the corresponding curves in Figure C3.