This appendix describes the ACT-R model which was developed to perform the Space Fortress task. It is an expert model designed to perform the task at the same level as the participants achieved by the end of the experiment. The model was tuned to fit the data for the ninth session because of the drop off in performance seen in the tenth session which was performed in the fMRI scanner.

The model performs the task using the same Python version of the Space Fortress software which was used for the human participants with only a few small changes. The game was extended to allow keyboard input to come from an external source as well as from the physical keyboard. It was also changed so that when the game state changes (ship moves, mines appear, bonus letters change, etc) it sends a description of that information out to an external system. Finally, the internal clock used to time the events in the game was modified to allow an external system to provide the time instead of always using the computer's real-time clock. This external interface was implementing using simple text messages over a TCP/IP socket connection and was not specific to ACT-R.

To perform this task several modifications and additions were made to the standard ACT-R modules. Those additions and changes will be described before discussing the details of the model itself.

The task requires the player to accurately time the intervals of some actions like the IFF processing and shooting at the fortress. To perform this timing the model uses a temporal module which was developed by Taatgen, van Rijn, & Anderson (2007) for performing time estimations. That module was not a component of ACT-R while this model was being developed, but it has since been made a standard component of the architecture.

Control of the ship in the task is sensitive to the duration of the actions, and as long as one holds down a key the ship will continue to change state, either rotating or accelerating. However, the ACT-R motor module, which is based on the manual motor processor of EPIC (Meyer & Kieras, 1997), only has simple keyboard actions like punch and peck-recoil which act as a quick strike and release. The first change was to add two new actions which are a hold key and a release key action. While these separate actions worked well to perform the relatively long actions like the initial thrust and the large turn at the start of the task they were not capable of performing many of the shorter actions which occur during the typical navigation sequence because of the additional time needed to have a production fire to initiate the release. To allow for those shorter key presses the module was also extended with another action which was called a delayed-punch. The delayed-punch is a key press which is held down and then automatically released after a duration that is specified when the action is initiated.

The default ACT-R motor module only has one execution path which means that only one finger may be in motion at a time. In the expert performance we often find that the participants are performing multiple actions concurrently, and because we have fMRI data we wanted to look for

differences in right and left hemisphere activity. To handle that, we extended the motor module to have a separate execution stage for each hand. The motor module still has a serial bottle neck in that it will only accept one request to make an action at a time, but with two execution stages a second request can progress in parallel with a previous one as long as they are using fingers on different hands.

The final change to the motor module was to add more detail to the state information which it could report. Additional queries were added which allow for testing each finger individually for being currently held down or not currently in use. This removed the need to explicitly record state information such as "currently thrusting" in the model because that could now be determined by testing if the left middle finger (the thrust key) was being held down.

The ACT-R vision module provides the model with a visual attention system that has the ability to track an item and update its features as it moves about the screen. Two changes were made to the vision module for modeling this task. The first was to adjust how the module records activity during tracking for the purpose of predicting fMRI data. The default mechanism is to record that the module is busy during the entire interval of tracking activity. Because the model will be tracking the ship almost all of the time in every trial, that constant level of activity would not allow for any discrimination among conditions. Thus, we decreased the activity for tracking to be the standard ACT-R cost of an attention shift, 85ms, once every 2 seconds while tracking is ongoing. This will allow us to see the effect of the additional attention shifts required for handling subtasks like encoding the bonus items and processing mines.

The other modification made to the vision module has to do with how it makes available information about changes in the visual scene. The standard vision module has a very simple mechanism for change detection which makes available the features of a single new or changed item when the visual scene is updated. While that mechanism works fine for simple tasks which have few or infrequent changes it was not sufficient for handling this task with its multiple potential item onsets and the constantly changing ship features. To deal with that the vision module was extended to make its pre-attentive feature information available for testing by the productions. This change does not remove the need for the model to make an explicit attention shift to an item to fully encode it. What it does do is allow for a detection and prioritization mechanism to be implemented within the productions of the model which can be sensitive to important feature onsets and choose which to attend to when there are multiple onsets or to ignore new stimuli if a high priority subtask is being executed.

The ACT-R imaginal module was extended to provide three special purpose capabilities to the model. The first was the ability to maintain and increment a count of shots which have hit the fortress without the need to consult declarative memory for counting facts. The next was an ability to predict the location of the moving mine so as to be able to determine the optimal orientation needed to shoot it (taking into account the movement of the ship, the movement of the mine, and the time required to make the turn) as well as an estimation of how long it has

before collision with the mine. The final addition was a mechanism to compute the orientation and thrust necessary to get back into a stable orbit around the fortress if it has drifted too far away, which can happen while trying to shoot a mine.

To perform the task the model uses the vision module to process the display provided by the game. That visual information is represented as a set of objects each with a set of symbolic features which the model may use. The model may only attend to the features of any one object at a time, but can be sensitive to the addition of or change to objects as described above. Not all of the items on the display are actually used by the model and there are some special objects which we have created to represent information about objects which disappear. There are three basic types of visual objects which the model uses for the task: text items, simple objects, and the player's ship. The items of each type and their specific features will be described next.

The text objects have one important feature which is simply the word or the number itself. The text objects which the model can attend to are: the initial foe letters for study, the two condition indicators for whether or not the mines and fortress are present, the bonus symbols, the IFF letter, and the vlnr count. None of the other text displayed in the task, like the scores or IFF timing are used by the model.

The simple objects which the model can see do not have any specific feature information. They merely indicate the presence or absence of a particular item. The items of this type from the game are the fortress, the mine, the fortress explosion, and the end of game screen. In addition to those, special objects are created to indicate situations where something disappears from the display. The model can also see objects which represent the following situations: when the player's ship explodes, when a player's shot hits the fortress, and when the mine explodes.

The last visual object used for the model is the player's ship. This object contains all of the features necessary for navigation and aiming and will typically be the object to which the model is attending. The details of the important navigation features used are shown in figure 1. Vel is the magnitude of the ships velocity vector. Orientation is the absolute direction the ship is pointing. The other features in the diagram are all relative to the location of the fortress (the center of the display). Dist is the distance from the ship to the center of the fortress location. Angle is the difference between where the ship is pointing and the line to the fortress center. Vdir is the angle formed between the ship's current velocity vector and the line to the center of the fortress. In addition to those features, the ship also contains features relative to the mine position with respect to angle and distance, a feature indicating that the fortress is between the ship and the mine (thus blocking a possible shot at the mine), and a feature which indicates when the ship collides with the small hex.

The model itself consists of a set of 77 production rules. The rules fall into 8 basic subtasks for playing the game. Those tasks and how they are related are shown in the state diagram in figure 2 and each will be described in more detail below.

The initial encoding productions are responsible for reading the foe mine letters from the screen at the start of the trial and rehearsing them for the 10 seconds until the current game type is presented. Once the game type is displayed the model reads and encodes that information and then waits for the game to start.

When the ship is reset, either at the start of a game or when it gets destroyed, the model attends to the ship and starts visually tracking it. Then, it gives it an initial thrust and makes a right turn to point toward the fortress. These are performed using the explicit hold and release actions since the timing of these actions is not critical to successful orbiting.

Once the ship has started moving and been turned to face the center of the display the basic orbiting productions take over. These productions are responsible for keeping the ship traveling in a clockwise orbit around the center hex at a reasonable velocity while remaining pointed toward the fortress (or center of display if the fortress is not there). There are two primary components to the orbiting strategy. The first is to keep the ship pointed toward the center. In a stable clockwise orbit this will only require turning the ship to the right occasionally as it travels. The second component is to apply thrust to keep the ship in orbit. This can be done by applying a small thrust when the angle between the ships velocity vector and the line to the center of the fortress is 90 degrees (assuming that the ship is pointed toward the center). The model performs these turns and thrusts using the delayed-punch motor action.

For various reasons the model may encounter situations when the orbit needs to be corrected. That can be occur because of noise in the model's actions (the ACT-R motor module adds noise to all of the action times), from the accumulation of small errors in navigation from the timing of the actions themselves, or when some other task like checking the bonus symbol requires visual attention to shift away from the ship. There are potentially a large number of errors which may need to be corrected while playing the game, but because the current model is a model of an expert player it does not make many of the errors which could potentially occur. It is only looking for four abnormal flight conditions while orbiting. The first is the need for a left turn to get back to pointing at the center. The next two errors have to do with the speed of the ship. If the ship is traveling too slowly then it will correct that by turning farther out from the center (aim more in the direction it is traveling) before the next thrust to increase the velocity. If the ship is going too fast then it will aim farther in (past the center in the opposite direction of the current ship velocity) before thrusting to decrease the speed. The last error that the model detects is bouncing off of the inner hexagon which causes the ship's velocity vector to immediately reverse direction. To correct that the model turns out from the fortress and then applies thrust to get back to moving in a clockwise direction.

When there is a fortress present the model will shoot at it. The model uses the temporal module to time its shots to the fortress. When it sees that it has shot the fortress it uses the imaginal module to increment a running count of how many hits it has made. When it has fewer than 10 hits on the fortress it will wait at least 250 ms before taking another shot. After it gets to 10 hits

then it will make two quick shots to attempt to kill the fortress every time it shoots until the fortress is destroyed. The orbiting productions keep the model pointed at the fortress thus it does not need to have any additional aiming done for shooting and only needs to verify that it is pointed in the right direction before taking a shot. When the model sees that it has destroyed the fortress it resets its current count and will resume the slow shooting at the next fortress when it appears.

When a new bonus symbol appears the model will attend to it if it is currently orbiting and in a safe point of its orbit. A safe point has been chosen to be immediately after a thrust action since that is likely to be the point from which the longest time will be required to make another correction. When the model attends to the bonus symbol it compares it to the last symbol it has attended and records that symbol in the imaginal buffer. If both the current and last attended symbols are the target symbol then the model taps the key to collect the bonus. After attending to the symbol it will return to visually tracking the ship and go back to orbiting. The processing of the bonus symbol is considered a low priority task and thus it will not interrupt other subtasks, like handling the mine or performing the double shot to destroy the fortress.

When a mine comes on the screen, the model will attempt to handle it as soon as possible. First it will attend to the IFF letter at the bottom of the screen and then return to tracking the ship. It will then attempt to retrieve whether or not that letter is a foe letter for the current game from declarative memory, and at that time it will also make a request to the imaginal module to compute the aiming information needed to shoot the mine. If the IFF letter is remembered as a foe the model will initiate the two quick presses necessary to tag the mine using the temporal module to achieve the appropriate interval. It will continue orbiting and shooting the fortress (if available) while the imaginal module computes the aiming information. Once the imaginal module returns the aiming information the model will make the necessary turn immediately if there is no fortress present. If there is a fortress and the mine is far away then it may take shots at the fortress before making the turn necessary to shoot the mine when it gets closer. While aiming at the mine and attempting to shoot it the model is will no longer attempt to keep a stable orbit about the fortress because that would interfere with the aiming. Once any necessary turn to aim has completed it will shoot at the mine as soon as it is in the ship's line of fire. If the model does not destroy the mine by the time it has expected to have done so based on the initial prediction, it will make a new request to the imaginal module to recompute the aiming information. If the mine gets too close to the model's ship it will just shoot at it regardless of whether any of the other actions have completed.

When the mine is gone (either because it was shot or because it hit the ship) the model will return to orbiting. If the ship is still close to the fortress then it can immediately start orbiting again. However, if the ship is too far away from the fortress for the standard orbiting productions to apply then the model must make some corrective actions before resuming the orbit. The corrections necessary to reenter a stable orbit are computed by the imaginal module similar to the way it computes the aiming information. That may include a turn to either the left

or right and will be followed by a thrust. Once that thrust has been performed the model will resume orbiting. If there is a fortress in the current condition, then after returning to orbiting the model will check the current vlnr value on the display and store that as the current count in the imaginal buffer. This must be done because the previous count was lost when the imaginal module computed the aiming information.

There is one last set of productions in the model which are not part of a specific subtask. Those productions are responsible for detecting situations which require the model to stop playing. That can happen if the ship is destroyed by either the fortress or a mine or if the trial has ended. If either of those situations are detected then the model will stop pressing any keys and return to the appropriate initialization state to either wait for the ship to reappear at the starting location or to wait for the foe letters at the beginning of the next trial.

The model was tuned to fit the human performance in terms of general ship control, number and duration of key presses, and component scores. The data from the ninth session of the study was used as a target for the model's performance. The results presented below are from 20 simulated participants each performing one session. All of the human data for comparison is from the ninth session, and all of the error bars shown are a standard deviation.

The general orbit the model flies was based around the average performance of participants using several measures taken from the fortress condition. The fortress condition was chosen because it contained the most consistent flight pattern across participants with all of the participants flying in a clockwise orbit around the fortress. The orbit only condition, which would seem like the better candidate for modeling the orbiting, was not used because it had a lot more variability in the paths flow by participants, including some who flew non-orbital paths. That was presumably because there was nothing else to do on those trials and there was no scoring penalty as long as the speed was sufficient and the ship stayed between the hexagons.

The first thing to consider was the general path which the ship should use to orbit. This was determined by looking at the average radius of the orbits and the average ship speed. Then, using that as the target flight plan, we tuned the model to match up with participants in terms of the number of presses for each of the navigation keys (left, right, and thrust) as well as the distribution of times for the lengths of those presses (as measured in 33ms game ticks which was the resolution of the data recorded).

In fitting the flight performance there was a lot of potential freedom available with respect to how many productions to use, which features to test, what values to use for those feature tests, and how long each response should be. To avoid getting bogged down in the complexity we tried to keep things as simple as we could in the model. We used the ship speed (vel feature) to break the orbiting into three categories: too slow, in control, and too fast. We also wanted to use a small set of key press durations, and chose to use three lengths with durations of 90, 125, and

160 milliseconds. For each of the categories we then developed a small set of productions for controlling the ship.

In total, the model has 13 productions for orbiting which apply in the orbiting and orbit correction states described above. All of those productions test the vel feature. The productions for turning also test the angle feature. The farther the angle is from the desired heading the longer the key press used to correct the turn. The speed determines the desired heading and the limits at which a press is needed. The thrusting productions test the distance, vdir, and angle values. The model will only thrust when it is aimed in the proper direction for the current speed, sufficiently far away from the inner hex, and when the vdir angle is large enough.

Figures 3 & 4 shows the model's performance in terms of speed and distance compared to the participants. Figure 5 shows the number of navigation key press per game, and figure 6 shows the distributions of those key presses.

After tuning the orbiting productions the firing at the fortress productions were tuned to match the number of fire key presses and the average number of fortresses destroyed. This required adjusting the delay between shots as well as the angle used to determine when the ship was aimed at the fortress. The noise in the temporal judgments was decreased so that the model would not make any errors of timing for the spacing of the shots for either the initial slow shots or the quick kill shots.

Figure 7 shows the number of fortresses destroyed in the games that have a fortress and figure 8 shows the average number of times the fire key was pressed in all game types. For figure 8 the data from 3 participants was excluded from the mines conditions because they reported practicing shooting at the fortress and that added several hundred extra shots to their trials.

For the mine handling the model was tuned to match the human performance in the points and speed scores in the mine condition. It was not tuned for any other specific aspects of the human performance because of large variations in how people did that task. The model was built to perform all of the components of the mine handling without error. However, the motor module's noise for the aiming actions and the amount of time available to act based on the distance between the mine's starting point and the ship can affect the model's ability to shoot a mine.

Participants made very few errors in regard to the IFF letters with less than 3% of the mines being treated incorrectly i.e. hitting iff for a friendly or failing to hit iff for a foe. So, we chose to not model those errors and the ACT-R declarative parameters were set so that the 10 seconds of rehearsal for the current letters is sufficient to have them be reliably and quickly retrieved throughout the entire 3 minute game. Unlike participants, the model makes no errors in the timing of the IFF strikes because of the very low noise added to the temporal judgments and the relatively small amount of noise which the motor module adds compared to the 250ms window for an IFF response.

Figure 9 shows the averaged results for mines destroyed as a function of type for the two conditions which have mines.

After adjusting the model for handling mines and fortresses it was run through the orbit and full conditions as well for scoring.  The results of the component scores by game type can be found in figure 10.

References

Meyer, D. E. & Kieras, D. E. (1997). A computational theory of executive cognitive processes and multiple-task performance. Part 1. Basic mechanisms. *Psychological Review, 104*, 2-65.

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

figure 1: Diagram of the geometric features of the ship which the model uses for navigation.

figure 2: A state diagram of the model's operation showing the major operations which it can perform and the events which cause it to transition among them.

figure 3: Comparison of the average ship speed over the course of a game for participants and the model.

figure 4: Comparison of the average ship distance from center of the screen over the course of a game for participants and the model.

figure 5: The average number of navigation key presses per game by game condition for participants and the model.

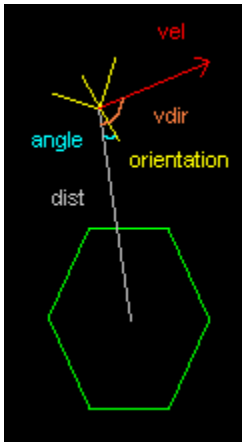
figure 6: The percentage distribution of key press durations as measured in game ticks over all games played.

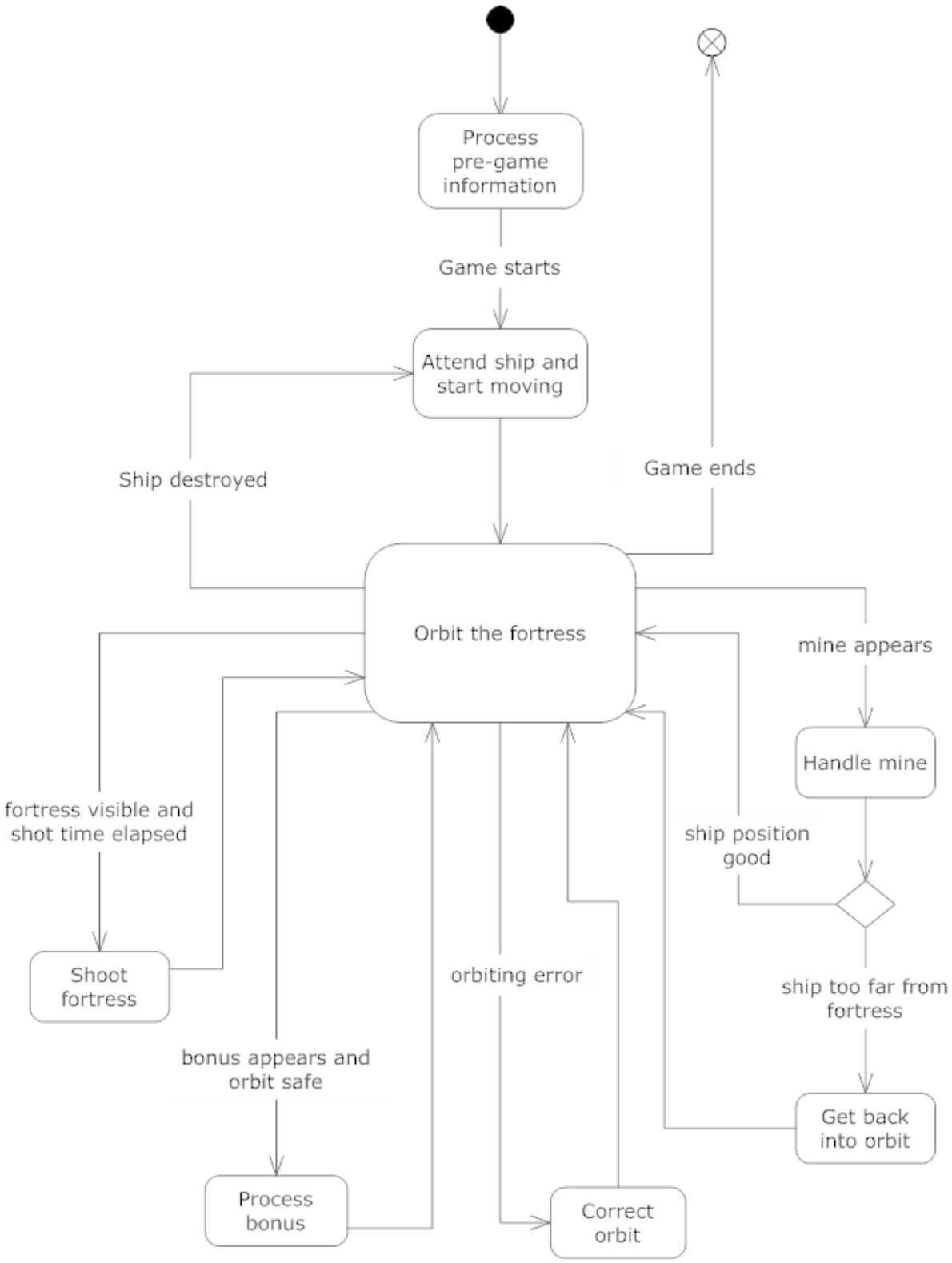
figure 7: The average number of fortresses destroyed per game by game type for those games which have a fortress.

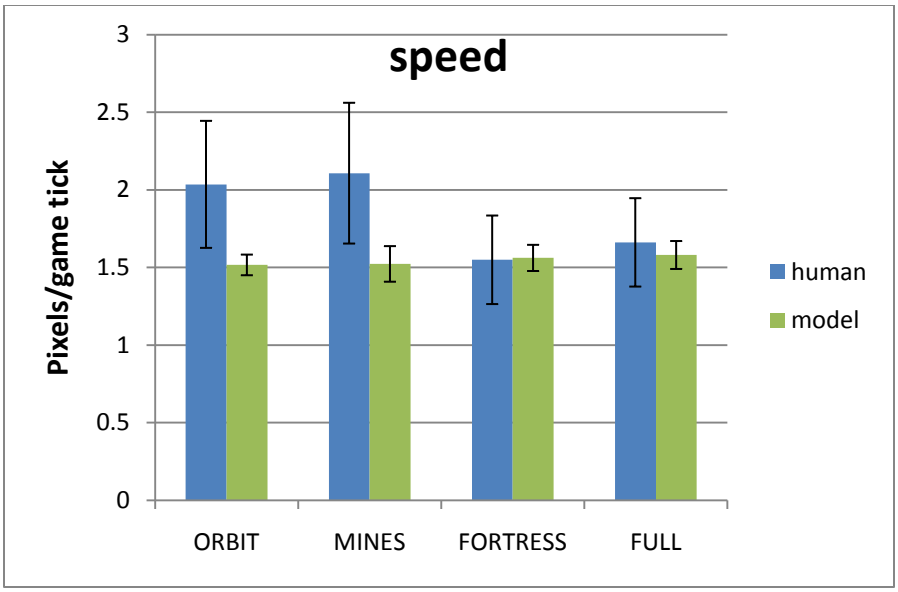
figure 8: The average number of times the fire key was pressed per game by game condition for the conditions which require firing.

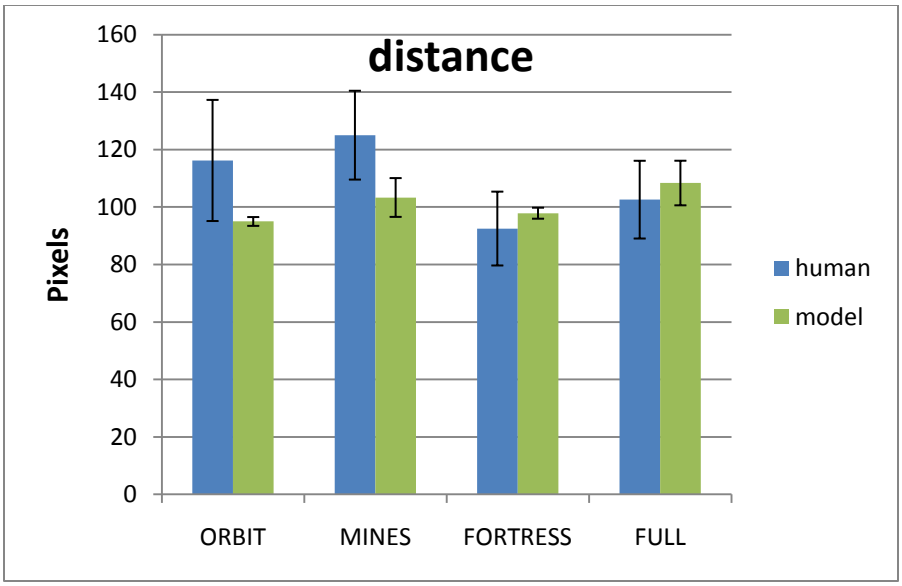
figure 9: Each mine that appears in the ninth session is either shot by the player (the hit-tagged-foe and hit-friend events) or hits the player's ship (the foe-hit-ship and friend-hit-ship events). This graph shows the average number of those events per game in the conditions with mines.

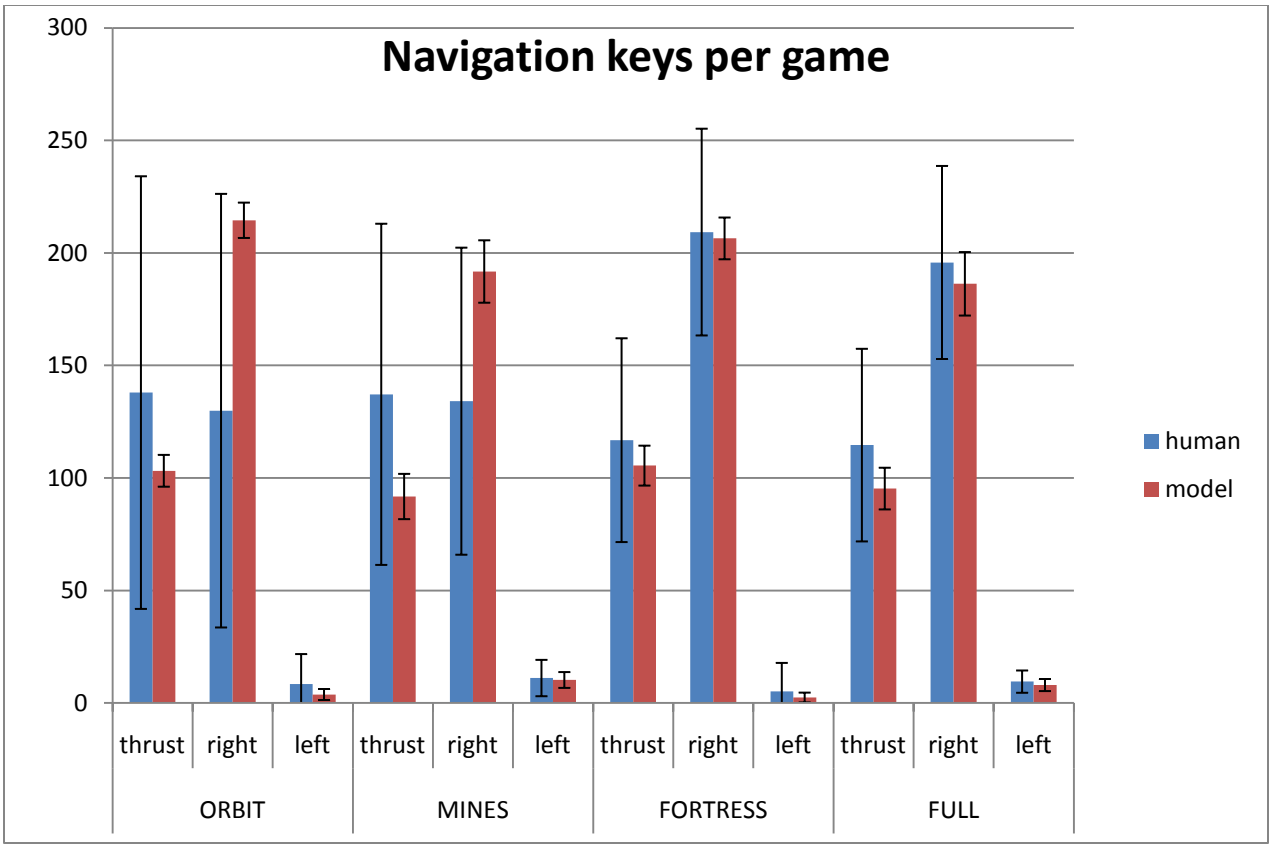
figure 10: Average component scores per game over all conditions.

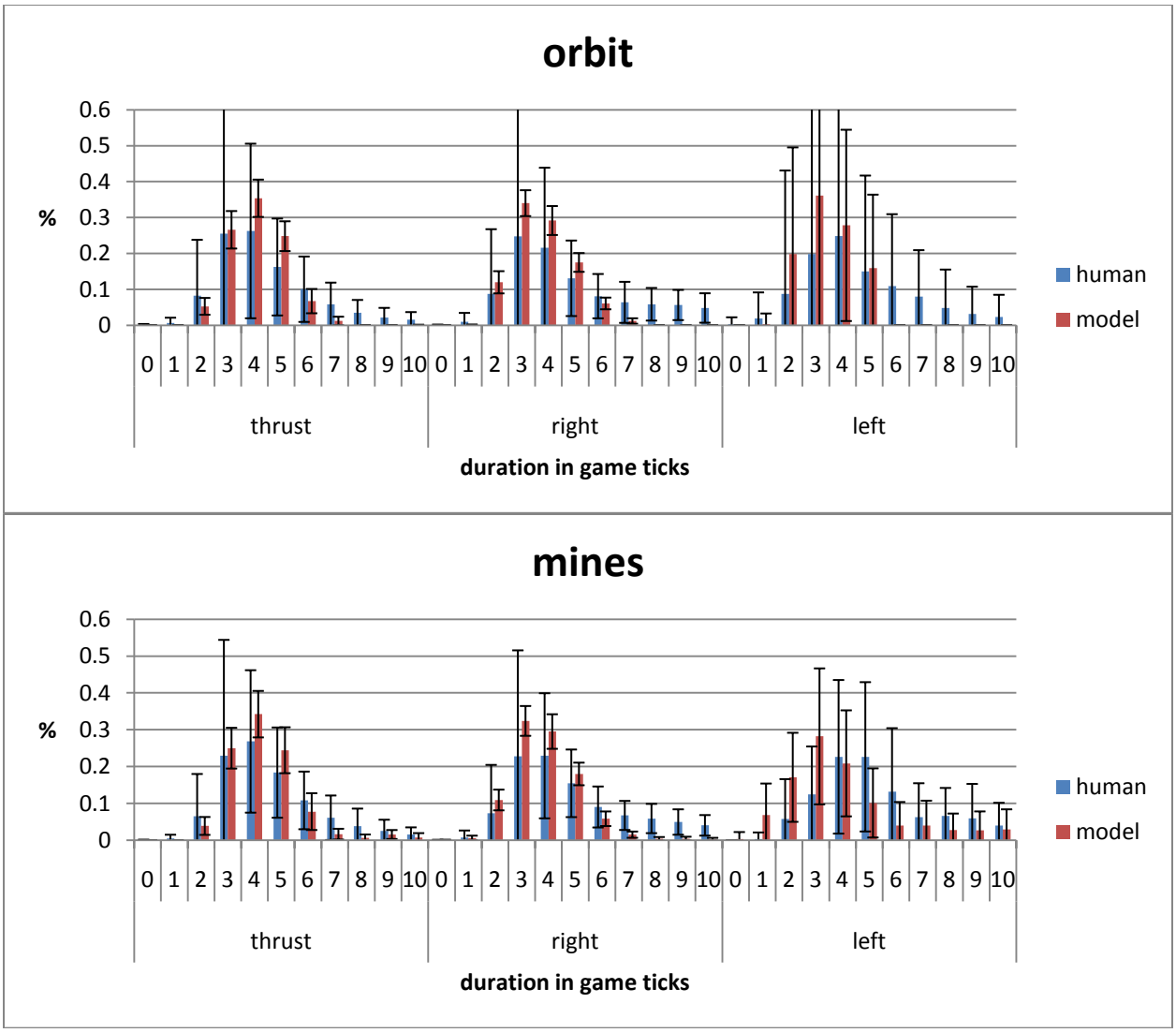

vel
vdir
angle
orientation
dist

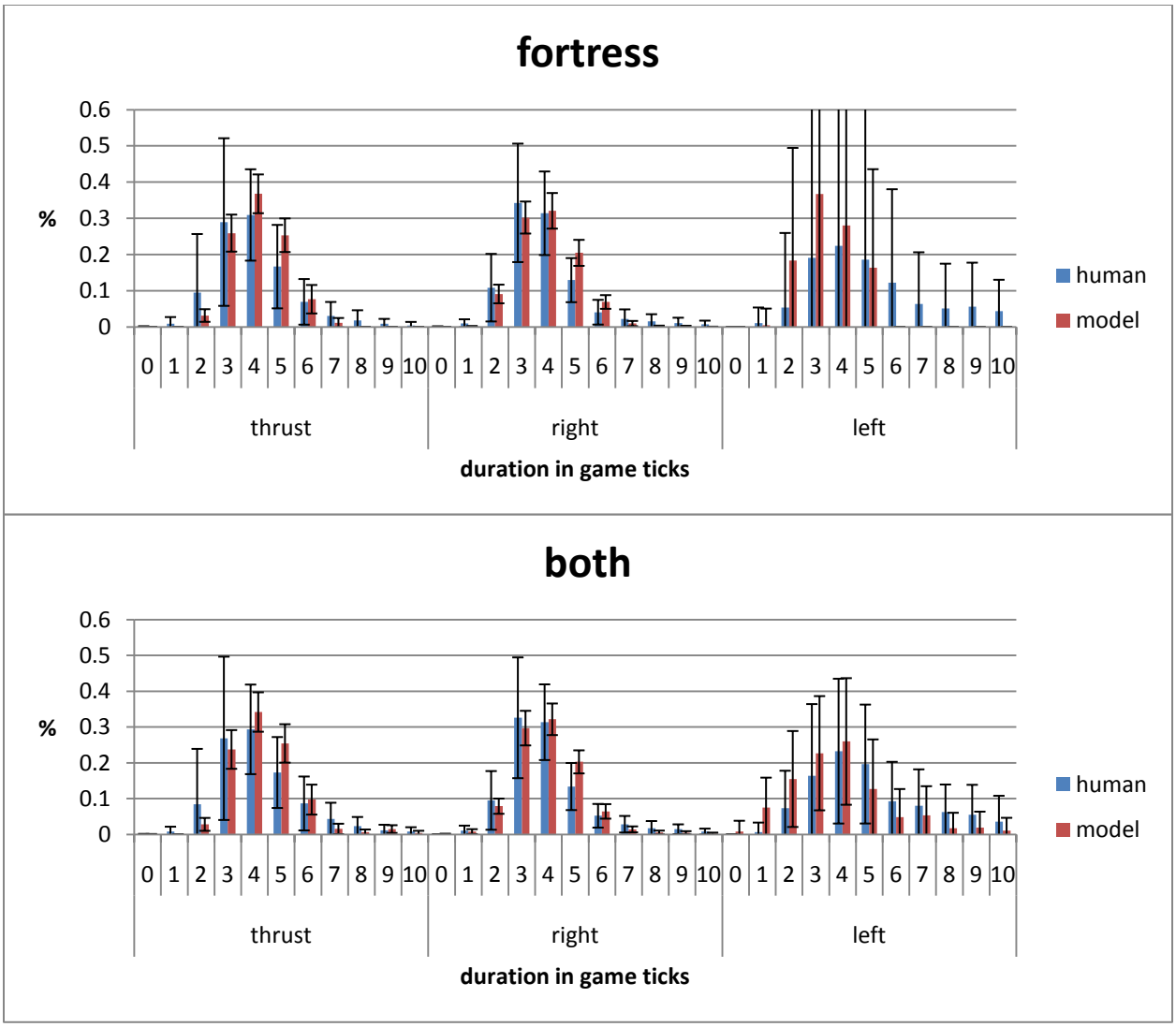

Process pre-game information
Game starts
Attend ship and start moving
Ship destroyed
Game ends
Orbit the fortress
mine appears
Handle mine
fortress visible and shot time elapsed
ship position good
Shoot fortress
orbiting error
ship too far from fortress
bonus appears and orbit safe
Get back into orbit
Process bonus
Correct orbit

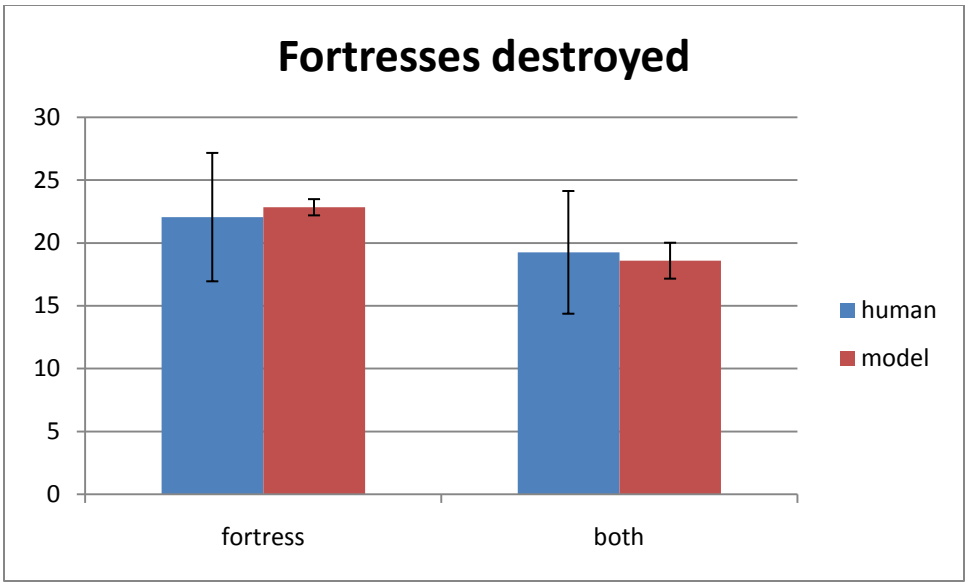

speed
Pixels/game tick
3
2.5
2
1.5
1
0.5
0
ORBIT
MINES
FORTRESS
FULL
human
model

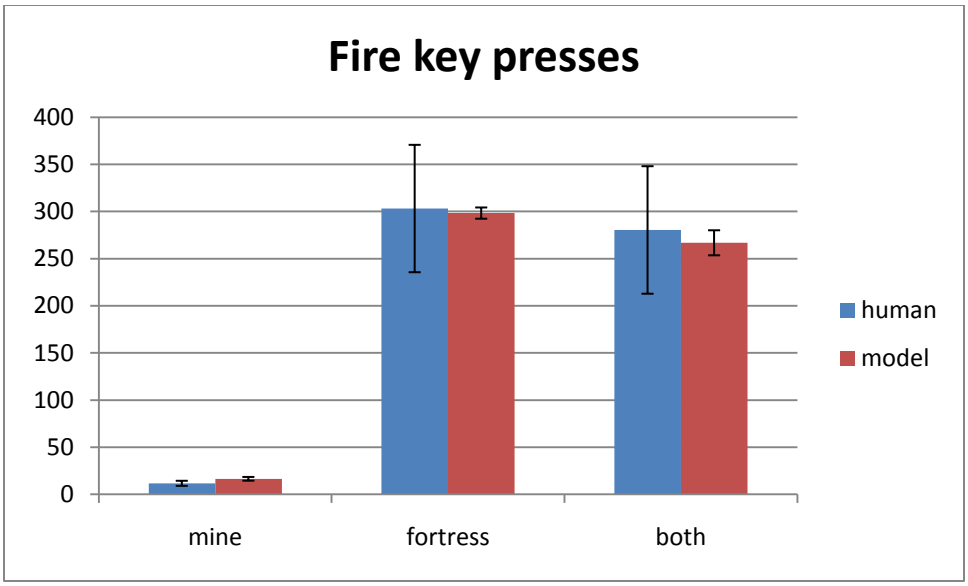

distance
Pixels
160
140
120
100
80
60
40
20
0
ORBIT
MINES
FORTRESS
FULL
human
model

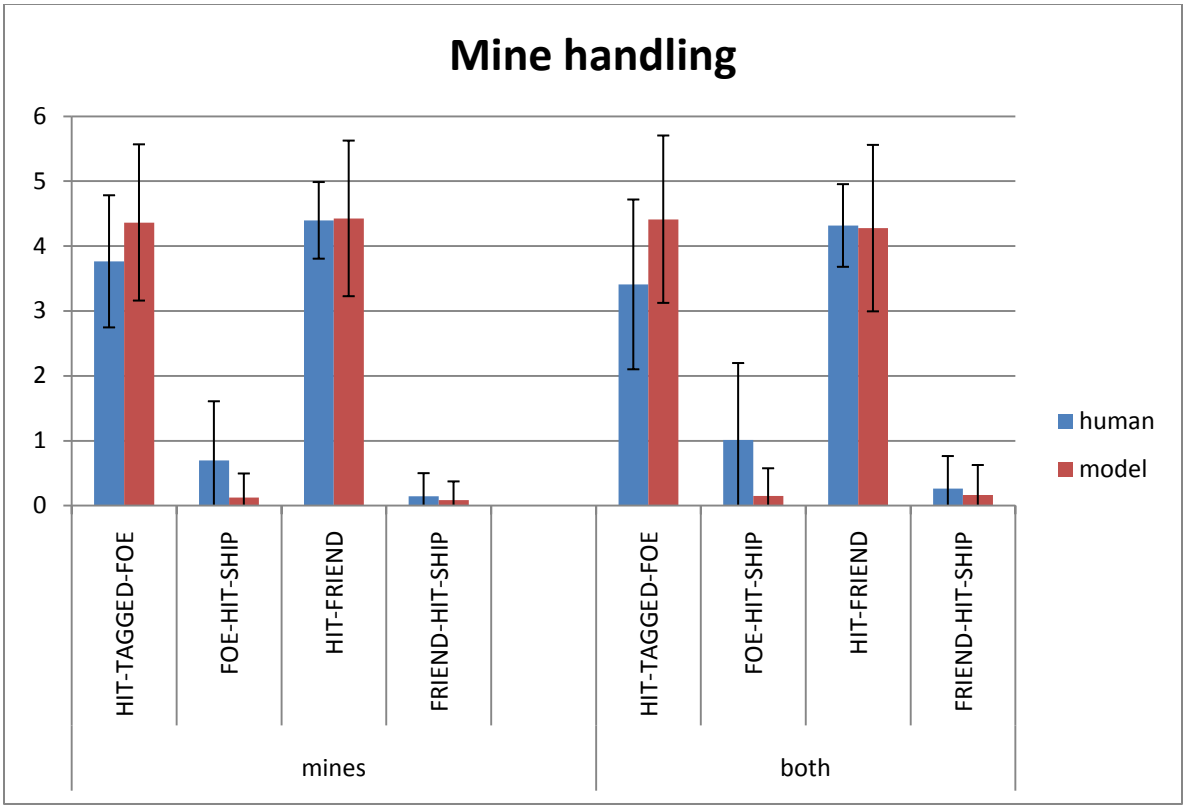

Navigation keys per game
300
250
200
150
100
50
0
thrust
right
left
thrust
right
left
thrust
right
left
thrust
right
left
ORBIT
MINES
FORTRESS
FULL
human
model

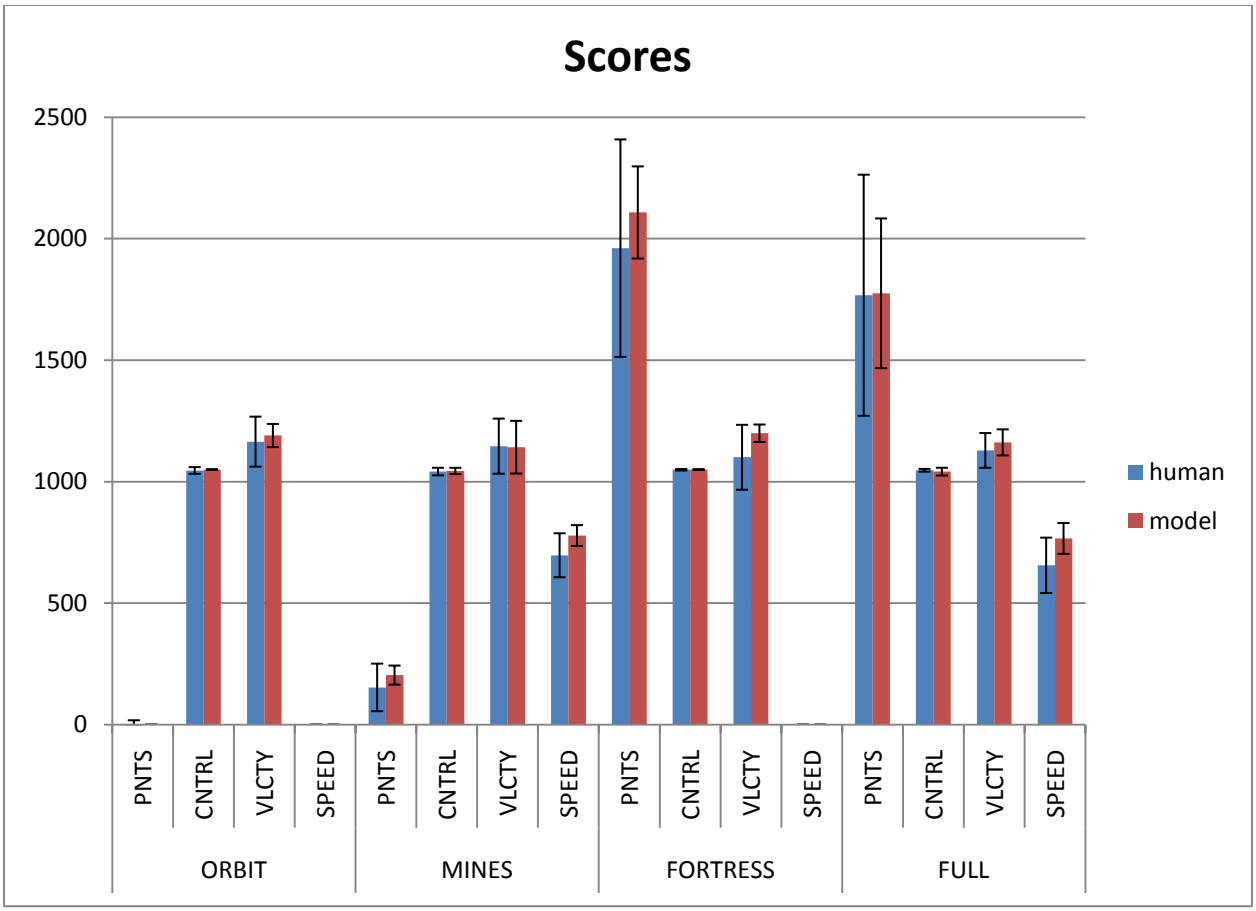

orbit
0.6
0.5
0.4
0.3
0.2
0.1
0
%
0 1 2 3 4 5 6 7 8 9 10 0 1 2 3 4 5 6 7 8 9 10 0 1 2 3 4 5 6 7 8 9 10
thrust
right
left
duration in game ticks
human
model
mines
0.6
0.5
0.4
0.3
0.2
0.1
0
%
0 1 2 3 4 5 6 7 8 9 10 0 1 2 3 4 5 6 7 8 9 10 0 1 2 3 4 5 6 7 8 9 10
thrust
right
left
duration in game ticks
human
model


fortress
%
0.6
0.5
0.4
0.3
0.2
0.1
0
0 1 2 3 4 5 6 7 8 9 10 0 1 2 3 4 5 6 7 8 9 10 0 1 2 3 4 5 6 7 8 9 10
thrust
right
left
duration in game ticks
human
model
both
%
0.6
0.5
0.4
0.3
0.2
0.1
0
0 1 2 3 4 5 6 7 8 9 10 0 1 2 3 4 5 6 7 8 9 10 0 1 2 3 4 5 6 7 8 9 10
thrust
right
left
duration in game ticks
human
model


Fortresses destroyed
30
25
20
15
10
5
0
fortress
both
human
model


Fire key presses
400
350
300
250
200
150
100
50
0
mine
fortress
both
human
model


Mine handling
6
5
4
3
2
1
0
HIT-TAGGED-FOE
FOE-HIT-SHIP
HIT-FRIEND
FRIEND-HIT-SHIP
HIT-TAGGED-FOE
FOE-HIT-SHIP
HIT-FRIEND
FRIEND-HIT-SHIP
mines
both
human
model


Scores
2500
2000
1500
1000
500
0
PNTS
CNTRL
VLCTY
SPEED
ORBIT
MINES
FORTRESS
FULL
human
model