Reconstructing Fine-Grained Cognition from Brain Activity

John R. Anderson, Shawn Betts, Jon M. Fincham, Ryan Hope, Mathew W. Walsh

a Department of Psychology, Carnegie Mellon, United States
b The Rand Corporation, United States

Abstract

We describe the Sketch-and-Stitch method for bringing together a cognitive model and EEG to reconstruct the cognition of a subject. The method was tested in the context of a video game where the actions are highly interdependent and variable: simply changing whether a key was pressed or not for a 30th of a second can lead to a very different outcome. The Sketch level identifies the critical events in the game and the Stitch level fills in the detailed actions between these events. The critical events tend to produce robust EEG signals and the cognitive model provides probabilities of various transitions between critical events and the distribution of intervals between these events. This information can be combined in a hidden semi-Markov model that identifies the most probable sequence of critical events and when they happened. The Stitch level selects detailed actions from an extensive library of model games to produce these critical events. The decision about which sequence of actions to select from the library is made on the basis of how well they would produce weaker aspects of the EEG signal. The resulting approach can produce quite compelling replays of actual games from the EEG of a subject.

Keywords:
Cognitive reconstruction, Cognitive Modeling, EEG, Game Playing

*Corresponding author at: 5000 Forbes Ave., Pittsburgh, PA 15213
Email address: ja@andrew.cmu.edu (John R. Anderson)
1. Introduction

The goal of this research is to track moment-by-moment what someone is thinking and doing over an extended period using the high temporal resolution of EEG. We will describe a method for achieving this goal that merges bottom-up information from classification of the EEG signal with top-down information from cognitive modeling. A great deal of research has studied classifying EEG signals and the results have been applied to a number domains such as brain-computer interfaces (Lotte et al., 2018), emotion recognition (Kim et al., 2013), understanding human memory (Noh et al., 2014), estimating workload (Brouwer et al., 2012), among others. With few exceptions (e.g. Su et al., 2018), this research involves tasks where the experimenter has control over the presentation of stimuli and examines activity in predefined intervals, typically locked to the presentation of these stimuli. However, in many realistic situations such as driving one does not have such experimental control and the sequence of events emerges as an interaction between the subject and the environment. Furthermore, the intervals between actions in such situations can be much shorter than the typical intervals used in most classification efforts.

To explore the tracking of mental state in such a context we chose video game play. There has been work on EEG and video games (e.g. Kerous et al., 2018), but typically focused on using traditional BCI methods to serve as a controller for the game. These applications typically leverage three types of EEG signals: (1) Signals sensitive to the occurrence of rare events, such as the presentation of a letter that the individual is thinking of (i.e., the P300); (2) Signals sensitive to how objects that the individual is attending to are presented (i.e., the SSVEP); and (3) Signals sensitive to planned and imagined movement (i.e., the Mu rhythm). These studies demonstrate the potential of inferring control signals from EEG, yet they do not involve highly dynamic tasks with significant perceptual-motor demands.

There has been little focus on recognizing events that occur in free-flowing
games. One exception is a study by Cavanagh & Castellanos (2016) that trained a neural classifier on controlled pre-game exemplar events. These events included the presentation of unexpected stimuli in an oddball detection task, and the presentation of positive and negative feedback in a gambling task. Both types of events—the occurrence of unexpected events and the delivery of rewards and punishment—also occur in many video games. Cavanagh and Castellanos found that classifiers trained using data from the control tasks could be used to categorize positive and negative events that occurred during Escape from Asteroid Axon, an 8-bit video game with continuous play. This demonstrates the transferability of EEG signals traditionally studied in simple laboratory tasks to a real-time game. A point of departure from the current study, however, is that Cavanagh and Castellanos directly provided the classifier with subsets of epochs of video-game play that contained critical events. Thus, the classifier did not need to detect epochs that contained critical events, nor did it fill in the sequence of actions and states between those events.

Video games offer an excellent opportunity to test methods for tracking human cognition because one can collect a record of what the subject did and what the game did on each game tick. Additionally, video games offer an opportunity to bridge the gap between carefully controlled laboratory studies that seek to isolate one or a small number of EEG signals, and the far more complex tasks that people routinely perform like driving a car in traffic.

This paper describes a method that attempts to reconstruct the actual game play from the EEG signal. This is a high bar because even getting a couple of actions out of synch can lead to disastrous reconstruction that is not at all human-like. Nonetheless, we have had some success in achieving the goal of reconstructing video game play from EEG signals (for examples, see http://andersonlab.net/reconstruction/). This success requires more than just an EEG classification algorithm. No matter how good the classification method is, it will misclassify some things, leading to an incoherent reconstruction of the full game (i.e., improbable or impossible sequences of events). Rather than directly choosing actions from the classifier, we use the output of the classifier
to select sequences of actions from a cognitive model \cite{Anderson2019} that can play the game like actual players. The result is a reconstruction of the game that is coherent, human-like, and typically very similar to the game of the player whose EEG signal we are working from.

1.1. Space Fortress Game

The video game we studied was a variant of Space Fortress. This game has a long history in the study of skill acquisition and training methods, first being used in the late 1980’s by a wide consortium of researchers (e.g., \cite{Donchin1989, Frederiksen1989, Gopher1989}). Part (a) of Figure 1 illustrates the critical elements of the game. Players are instructed to fly a ship between the two hexagons. They are firing missiles at a fortress in the middle, while trying to avoid being hit by shells fired by the fortress. The ship flies in a frictionless space. 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 subjects are overwhelmingly video game players, most have no experience in navigating in a frictionless environment.

There have been EEG studies of Space Fortress. \cite{Maclin2011} recorded EEG from subjects as they played Space Fortress while concurrently performing a secondary task that involved counting rare auditory oddball stimuli. The amplitude of the P300 to rare stimuli in the oddball detection task increased following training on Space Fortress, while the amplitude of the P300 to stimuli in Space Fortress decreased. These results indicate that with training, the primary task of playing Space Fortress became less attentionally demanding, freeing resources for the secondary task. In subsequent work, \cite{Mathewson2012} found that event-related increases in frontal theta, an oscillation associated with attentional control, predicted individual differences in learning rate. Together, these results show the importance of attention in Space Fortress and that there is a reduction in attentional demands with practice.

We used the Autoturn version of the game introduced in \cite{Anderson2019}.
Figure 1: (a) The Space Fortress screen, showing the inner and outer hexagon, a missile fired at the fortress, and a shell fired at the ship. The distance to the corners of the outer hexagon is 200 pixels and the distance to the corners of the inner hexagon is 40 pixels. The ship starts 120 pixels to the left of the center, flying at 30 pixels per second, parallel to the upper left side of the hexagon. The dotted lines illustrate an example path during one game. (b) A schematic representation of critical values for firing and flight control.

In this variant of the game, the ship is always aimed at the fortress and subjects do not have to turn it. The ship begins each game aimed at the fortress, at the position of the starting vector in Figure 1a, and flying at a moderate speed in the direction of the vector. To avoid having their ship destroyed, subjects must avoid hitting the inner or outer hexagons, and they must fly fast enough to prevent the fortress from aiming, firing at, and hitting the ship. When subjects are successful the ship goes around the fortress in a clockwise direction. They can destroy the fortress by shooting missiles at it to build up its vulnerability and then destroying it with a "kill shot". If the fortress is destroyed it leaves the screen for 1 second before respawning. If the ship is destroyed it respawns after 1 second in the starting position flying along the starting vector. Our version of the game eliminated much of the 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 the ship was destroyed.
3. To reinforce accurate firing, every fire costs 2 points.

To keep subjects from being discouraged early on, their score never went negative. The replay site (http://andersonlab.net/reconstruction/) offers examples of game play.

[Anderson et al. (2019)] found that subjects can achieve relatively high and fairly stable performance within an hour of playing AutoTurn (much faster than in original Space Fortress where subjects are also responsible for turning their ship among other things). To maintain a constant challenge of game play, a staircase procedure decreased the separation between the inner and outer hexagons as subjects got better. Subjects played 1-minute games. During the first 10 games the inner corners were 40 pixels from the center and the outer corners were 200 pixels from the center producing a width of 160 pixels. After the tenth game, the border width was reduced by 10 pixels if the subject had 0 or 1 deaths in the prior game and it is increased by 30 pixels (to a maximum width of 160 pixels) if they had 2 or more deaths. In this way the death rate in the game was maintained at about 1 death per 1-minute game. For each 10 pixels the border is reduced, subjects get an additional 10 points for each fortress they destroy. Navigation becomes increasingly difficult as one has to fly between narrow borders, with many deaths resulting from thrusting into the inner hexagon, a rare event with the original 160 pixel border.

The game advances at 30 ticks per second. Only two keys are pressed—a left hand press of the W key to add thrust to the ship and a right hand press of the space bar to fire at the fortress. Exactly when one thrusts and fires is critical to performance. The difference of a single game tick can mean the difference between destroying the fortress and being destroyed. Critically, the impact of a key press depends on the past history of key presses as well: the consequence of a thrust depends on the ship’s current position and flight path (determined by past thrusts) while the consequence of a fire depends on how preceding fires have affected the fortress’s vulnerability.

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 11, subjects can destroy the fortress by quickly firing an additional missile at it. Each fire increases the fortress’s vulnerability by one, provided the fires are paced at least 250 ms apart. If the inter-fire interval is less than 250 ms the vulnerability is reset to 0 and one must begin the build up of vulnerability anew. While subjects could easily make sure the fires building up vulnerability are at least 250 ms apart by putting long pauses between them, this would reduce the number of fortresses destroyed and points gained per game. Thus, subjects are motivated to pace the fires as close to 250 ms as they can without going below than 250 ms. threshold and producing a reset. In contrast to the fires that build up the vulnerability, the fire to destroy the fortress must be less than 250 ms from the last fire.

Since the ship is always aimed at the fortress, subjects do not need to turn their ship as in the original version of Space Fortress. To navigate around the fortress, they must press the thrust key at appropriate times and for appropriate durations. The direction of the ship after a thrust is determined by a vector sum of the current flight velocity and the acceleration they add in the direction of the fortress. The acceleration is determined by how long they hold the thrust key down. Subjects’ average ship speed is a little over 1 pixel per game tick (the ship starts out flying at 1 pixel per game tick). Every game tick the thrust key is held down adds .3 units of speed in the current orientation of the ship (i.e., towards the fortress). As an example, suppose the ship is flying at 1.2 pixels per second, the angle between aim and ship direction (Thrust Angle in part a of Figure 1) is 120 degrees, and the thrust key is held down for 4 game ticks. This will result in a force of 1.2 pixels in the direction the ship is aimed. The resulting trajectory would still have a velocity of 1.2 pixels per second (more if the thrust angle was less than 120 degrees, less if it was more), and would now

\[\text{This is only a close approximation because the flight of the ship and its orientation update after each tick of thrust.}\]
be in a direction that bisected the thrust angle. Thrusts at the wrong time or for the wrong duration can lead to death of the ship, which happens if the ship hits the inner or outer hexagons or if the ship flies so slowly the fortress can shoot it.

1.2. Overview of Sketch-and-Stitch Reconstruction

We developed the **Sketch-and-Stitch** method to infer a trace of the subjects’ cognition. While we apply the method here to a video game because it provides a demanding test, the underlying approach could be applied to any task. The method involves first developing a sketch of the critical mental events in a task that extend over a substantial period using an extension of a method called HSMM-MVPA (HSMM: hidden semi-Markov models for identifying the structure of events in time; MVPA: multivariate pattern analysis for identifying patterns in brain activity). We have applied earlier versions of the HSMM-MVPA method to both parsing of fMRI data (e.g. [Anderson et al., 2010, 2012]) and to the processing of EEG and MEG data (e.g. [Anderson et al., 2016, 2018]), but nothing as time-critical as reconstructing video game play. After describing the current application of HSMM-MVPA in the results section, we will highlight its key features and innovations relative to past applications that enable it to succeed in the task.

Having produced a hypothesis about when the critical events happened in a game using HSMM-MVPA, the method then stitches in a detailed reconstruction of the subject’s cognition that led to these critical events. In this video game these detailed steps of cognition are directly associated with a detailed trace of actions providing a rigorous ground truth for judging the success of the effort. Stitching uses sequences of actions from runs of a simulation model that can produce human-like sequences of cognition. In our case, that simulation model is the ACT-R model described in [Anderson et al., 2019], which produced a high-quality match to subject game play. Such a model (because it is stochastic like subjects) can be used to create a large library of candidate sequences for stitching between critical events. The Stitch-and-Sketch method selects among
these candidate sequences according to how well they would produce EEG signals that match the subject.

2. Methods

2.1. Subjects

A total of 25 subjects were recruited from the CMU population of students and researchers between the ages of 18 and 40. 5 subjects were excluded because of poor performance (1 subject) and equipment problems (4 subjects), leaving 20 subjects (11 male, 9 female). All were right-handed. None reported a history of neurological impairment. Subjects were paid $75 for participation in the experiment.

2.2. Game Play

After subjects studied game instructions, they played 60 1-minute games choosing to move on to the next game at their own pace. Each game lasted 1819 game ticks (each game tick a 30th of a second, making the game a little longer than 1 minute). The game records the state of the screen (where the ship is if alive, the direction and speed of movement, whether shells or missiles are on the screen, and whether a key is depressed) at each game tick. This serves as the ground truth both for training the decoder and for testing its predictions.

2.3. EEG Analysis

The EEG was recorded from 128 Ag-AgCl sintered electrodes (10-20 system) using a Biosemi Active II System (Biosemi, Amsterdam, Netherlands). The EEG was re-referenced online to the combined common mode sense (CMS) and driven right leg (DRL) circuit. Electrodes were also placed on the right and left mastoids. Scalp recordings were algebraically re-referenced offline to the average of the right and left mastoids. The EEG and EOG signals were filtered with a bandpass filter of .1 to 70.0 Hz and were digitized at 512 Hz. The vertical EOG was recorded as the potential between electrodes placed above and below
the left eye, and the horizontal EOG was recorded as the potential between electrodes placed at the external canthi. The EEG recording was decomposed into independent components using the EEGLAB FastICA algorithm (Delorme & Makeig, 2004). Components associated with eye blinks were automatically identified and projected out of the EEG recording.

The EEG signal was recorded continuously for the entire experimental session and broken into 1-minute games. There was also a complete record of what happened in each game. Portions of the game periods were identified as bad signals were excluded. This resulted in loss of the signal for an average of 1.7 seconds per game for games used in the decoding (52.5% of the games had no lost signal; the worst game had 21.8 seconds of lost signal). This reflects a realistic complication in decoding where useful signal can be lost for some fraction of time.

The EEG was down-sampled to 30 Hz to match the game ticks. A one-second window around each game tick (14 game ticks before, the game tick, and 15 game ticks after) was used to classify whether a game tick contained a critical event. This means that each game tick had associated with it a vector of 30*128=3840 electrode readings, representing regional effects, frequency effects (below 30 Hz), and their interactions. Because the vector associated with a game tick requires a complete signal for 1 second, game ticks at the beginning and end of a game do not have corresponding vectors nor do game ticks in or near lost signal. Thus, 29 ticks at the beginning and end of the game have no vectors as well as an average of 71.6 ticks in the vicinity of lost signals, leaving an average of 1,718.4 vectors per game. The available vectors for each game were z-scored to standardize them across games. To reduce dimensionality and filter out noise, the vectors for all games and subjects were subjected to a PCA analysis and the 1000 top dimensions were kept. Thus, we had an average of 1718.4 1000-element vectors per game. These are what were used for all classification analyses.

2 A segment was excluded if the majority of channels were marked as bad.
2.4. Classification

All reconstructions efforts focused on the last 55 games where performance is relatively stable. We excluded an additional 20 games of the remaining 1100 games because of particularly bad EEG signal or low activity by the subjects. This left 1080 games\(^3\), which will serve as the focus of analyses. We used a leave-one-game-out approach: For a given target game of one subject, the training was done with all remaining games for that subject and all games for all other subjects. A linear discriminant classifier was trained to label the vectors of EEG activity with the category associated with the game tick that the vector describes. To reflect the fact the sensor activity of the subject may be most relevant, that subject’s other games are weighted 15 times more than the games of other subjects. This was repeated for each game to get results for all 1080 games. We have neither explored different weightings of that subject to other subjects nor different classification methods. Thus, while the classification results are quite good, they probably are not the best possible.

2.5. Model

We used the same ACT-R model as described in [Anderson et al. (2019)](Anderson2019). To summarize the model: it starts with a declarative representation of the instructions about when to do what. This produces slow performance initially, but over time the model builds action rules that directly perform the actions in the appropriate situations (bypassing the need for declarative retrievals). Critical to its performance in Autoturn are learning when to thrust and when to fire. A Controller module has been implemented within ACT-R that explores a range of values for when to fire and when to thrust and converges on appropriate settings, which it comes to exploit. The creation of action rules and the learning

\(^{3}\text{Excluded from the original 20 (subjects) \times 60 (games) = 1200 games were the 100 first 5 games for each subject (a total of 100), 1 game without good signal throughout, 8 further games where subjects failed to destroy a fortress without resetting or being killed, and 11 games with 12 or fewer critical events.}
of control values for action underlie the improvement with practice in the model. The behavior of the model is similar to subjects because it uses established ACT-R settings (on the basis of prior experiments) for the timing and variability of mental steps and motor execution. While this model was developed only for the 160-wide pixel border separation, it generalizes to the narrower borders in this experiment because the model monitors for closeness to the borders.

We simulated 100 subjects by running the model 100 times for 60 games under the same game conditions as humans: As the model got better, the borders narrowed. If the model suffered more than one death in a game, the borders expanded. In addition, to collect enough games at each width to have a library for reconstruction, for each possible width 50 model runs of 60 games were executed at a fixed width. In all runs the model was learning and got better with later games. Since the first 5 games of subjects were excluded in the reconstruction efforts, we similarly excluded the first 5 games from each of these runs, yielding a library of 50*55=2750 games at each border width. There are 13 possible widths from 40 to 160 pixels, making for a library of 2750*13=35,750 model games to serve as a basis for reconstructing the 1080 subject games.

3. Results

3.1. Behavioral Results: Subjects and Models

Figure 2 shows how various measures changed over the course of the 60 games for the experiment participants and the 100 simulated subjects. Part (a) tracks the width of the space between the two hexagons. This is held constant at 160 pixels by the experiment for the first 10 games, after which the staircase process sets in. The width then decreases to an average of about 100 pixels. Points and kills (Parts b and c) increase rapidly over the first 10 or so games and then increase more gradually. Ship deaths drop rapidly over the first 10 games before rising to about 1 death per game, which is the goal of the staircase procedure (Part d). Unlike human subjects, the model flies fairly safely from the
beginning. Once the staircase procedure sets in both humans and the models show the expected rate of about 1 death per game.\footnote{The instructions provide information about the importance of thrust angle (see Figure 1b) for flight—not too large or one will slow down and not too small or one will speed up too much. The model starts with a perfect encoding of this information whereas some subjects only gradually appreciate this.}
In later games subjects fly at a range of widths. For instance, on the last game, different subjects are flying at ranges varying from 70 to 160 pixels. Subjects vary in how tight a space they manage to fly in, but 13 of the subjects manage to reach a width of 70 pixels at some point and all but 1 reach 90 pixels (the other reaching 110 pixels). The 100 simulated subjects show a similar range with 43 reaching 70 pixels, and all but 9 reaching 90 pixels. The best subject reached 40 pixels while 3 of the 100 simulated subjects reached 40 pixels. Figure 3 shows how performance varies as a function of width (omitting the first 5 games where the rapid changes were taking place). Subjects earn somewhat more points with greater widths (Figure 3a). There is relatively little effect number of fortresses destroyed with width (Figure 3b) but a large effect on number of deaths (Figure 3c). Speed is somewhat greater with wider borders (Figure 3d). The model also has these trends.
Figure 3: Mean values (line) and standard errors (area around lines) per game for subjects and models as a function of border width (a) points before bonuses for kills at narrow borders; (b) number of fortress destructions; (c) number of deaths; (d) speed of ship. Because of the few cases of 40 pixels (1 game for subjects, 4 for models), these data are averaged in with 50 pixels (16 games for subjects, 20 games for models).

3.2. Response-Based Classification

A direct model-free approach to reconstructing game play would be to try to recognize when the left (thrust) and right (fire) fingers are pressed. A fair amount of research has shown good discrimination between imagined left- and right-hand movements for application in BCI (Lotte et al., 2018) and classification of actual key presses by hand is also good (Krauledat et al., 2004). If we could correctly identify when the fingers are pressed we would be able to recreate the game play. Two features of the current task make it more challenging.
than the typical left-right discrimination. First, it requires not just a binary discrimination but rather a 4-way discrimination because the game includes ticks when neither finger is pressed (the majority: 68.2\%) and when both fingers are pressed (rare: 0.2\%). Second, the discrimination among these four categories must be made separately for every game tick.

Table 1 shows the classification performance on game ticks that have associated EEG vectors. Part a of the table shows the results when each game tick is assigned the label that makes the EEG vector most likely. Clearly, considerable discrimination can be achieved as indicated by the large values on the main diagonal. The overall accuracy is 47\%. A d-prime (Wickens, 2002) measure of discriminability is .90\(^5\). Excluding ticks with no or both keys pressed, a binary discrimination of ticks with fires from ticks with thrusts is better (d-prime 1.54; 77.8\% accuracy). Part b of Table 1 shows the results using the posterior

---

### Table 1: Response-Based Classification

<table>
<thead>
<tr>
<th></th>
<th>Keys Labelled for Game Tick</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>None</td>
<td>Fire</td>
<td>Thrust</td>
<td>Both</td>
<td>Total</td>
</tr>
<tr>
<td>Keys Pressed during Game Tick</td>
<td>None</td>
<td>566,338</td>
<td>285,646</td>
<td>255,966</td>
<td>1,263,035</td>
</tr>
<tr>
<td></td>
<td>Fire</td>
<td>105,799</td>
<td>261,162</td>
<td>66,504</td>
<td>82,210</td>
</tr>
<tr>
<td></td>
<td>Thrust</td>
<td>10,169</td>
<td>7,082</td>
<td>41,927</td>
<td>10,128</td>
</tr>
<tr>
<td></td>
<td>Both</td>
<td>383</td>
<td>911</td>
<td>992</td>
<td>1,652</td>
</tr>
<tr>
<td>Total</td>
<td>682,689</td>
<td>554,801</td>
<td>365,389</td>
<td>249,075</td>
<td></td>
</tr>
</tbody>
</table>

### Table 1: Response-Based Classification

<table>
<thead>
<tr>
<th></th>
<th>Keys Labelled for Game Tick</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>None</td>
<td>Fire</td>
<td>Thrust</td>
<td>Both</td>
<td>Total</td>
</tr>
<tr>
<td>Keys Pressed during Game Tick</td>
<td>None</td>
<td>1,197,618</td>
<td>62,745</td>
<td>2,613</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>Fire</td>
<td>392,235</td>
<td>122,988</td>
<td>405</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>Thrust</td>
<td>62,227</td>
<td>3,023</td>
<td>4,038</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Both</td>
<td>3,023</td>
<td>874</td>
<td>33</td>
<td>8</td>
</tr>
<tr>
<td>Total</td>
<td>1,655,103</td>
<td>189,630</td>
<td>7,089</td>
<td>132</td>
<td></td>
</tr>
</tbody>
</table>

---

\(^5\)The d-primes reported in this paper for an n-class categorization are the average of n 2x2 classifications of the ability to discriminate each of the n categories from their absence. In the case of single game classifications, where there can be zero entries, the matrices are augmented by adding .5 to all cells.
probability that weights the likelihood by the prior probability of the category. The d-prime measure in this case is 1.03. Only 25 of the 1080 games have negative d-primes. Calculating d-primes per subject, the mean is .83 with the smallest d-prime .50.

The categorization uses a combination of spatial and frequency information over a 1-second interval surrounding a game tick and so the full patterns are complex. Figure 4 shows the average temporal pattern for three central electrodes and the full electrode patterns 5 ticks preceding and 10 ticks after the event. At this level of aggregation the most prominent feature is that left frontal activation tends to drop after a fire and rise after a thrust. The difference between fire and thrust at the central electrode FZ 10 ticks after a thrust is highly significant (t(19) = 6.04, p < .0001). There is also greater right lateralized central negativity prior to a thrust and slightly greater left lateralized negativity prior to a fire, which would be the expected lateralized readiness potential opposite the hand pressing the key (e.g. Smulders et al., 2012). For instance, the difference between fire and thrust at the right central electrode F4 5 ticks before the thrust is highly significant (t(19) = 6.30, p < .0001). Notwithstanding these clear patterns that appear in average data, on single trials there is no electrode comparison on any game tick that provides even as much as 53% correct binary
classification. The higher accuracy achieved by the classifier depends on more complex patterns.

How well could one use these key classifications to reconstruct the games? As a test we took the posterior most probable class for each game tick\(^6\) and ran the resulting action sequence through the game. The average score was 13 points per game in contrast to 1218 points achieved by the subjects who generated the EEG signal. Could one do better with a better classifier? To simulate a classifier that was "10 times" more accurate we generated a sequence of key activity that chose the true key presses for each game tick with probability .9 and otherwise the keys of the current classifier. The activity generated in this manner (which matches the actual press pattern 97.2\% of the game ticks) averages 92 points a game. We created a classifier that was "100 times" better by using the true key presses 99\% of the time. The sequence of key activity generated in this manner (which matches the actual press pattern 99.7\% of the game ticks unrealistically good) still fell short of human performance with 818 points per game. Many of these games still contained unusual episodes such as repeated resets or easily avoided crashes. Thus, even with improvement in accuracy greater than seems possible with improved classification, it does not seem possible to reconstruct game play from direct action classification.

3.3. Classification of Critical Events

Higher classification performance can be achieved if rather than trying to classify every action one just tries to classify the critical events that happen during the game. Five critical events could occur in the course of game play:

1. **Kills.** The fortress is destroyed the player gains 100 points.

2. **Fortress Respawns.** The fortress is respawned after a second and the player can resume firing.

---

\(^6\)Game ticks without an EEG vector were assigned to the category of no key activity, which is the most common category.
3. **Deaths.** The destruction of the player’s ship results in the loss of 100 points.

4. **Ship Respawns.** The fortress is respawned a second after death and the player can resume thrusting and firing.

5. **Resets.** If the interval between fires is less than 250 ms the fortress vulnerability will be set back to zero and the subject must begin anew.

The 1080 1-minute games in the pool averaged 9.38 kills, 0.86 deaths, and 1.15 resets. The numbers of critical events with EEG vectors are 9,742 kills, 9,731 fortress respawns, 819 deaths, 805 ship respawns, and 1,159 vulnerability resets. In total, these are far less numerous than the thrust key and fire key events (row sums in Table 1).

Figure 5 shows the EEG signals around the critical events. These critical events are associated with much stronger responses than the key presses in Figure 4. Part a shows the 1-second around a kill. There is a posterior positivity that reaches a maximum 100 ms before the kill, then a general negativity 100 ms after the kill, and then an anterior positivity about 300 ms after the kill. Part b, which is a continuation of Part a shows the activity around the respawn of the fortress. There is return to an anterior positivity about 300 ms after the fortress reappearance. Parts c and d show the activity around a death and the respawn of the ship. There is negativity in anticipation of the death, which switches to strong central positivity peaking 400 ms after the death. The positivity remains but becomes left lateralized after the ship respawns. Part e shows the response to a reset where there is a strong central negativity 300 ms after the reset followed by an even stronger central positivity 500 ms after the reset. One feature common to kills, deaths, and resets (Parts a, c, and e) is a post-event positivity, although that positivity varies somewhat in its timing and distribution across the scalp. The magnitude of this positivity varies with the rareness of the event, with the most common kills showing the smallest response and the least frequent deaths showing the largest response, as we would expect from a P300 (Polich, 2012). The delay in the positivity for resets relative to the
Figure 5: (a, b) EEG activity around the destruction of the fortress and its respawn with the scalp profiles ranging from -4 to 4 microvolts. (c, d) EEG activity around the death of the ship and its respawn with the scalp profiles ranging from -10 to 10 microvolts. (e) EEG activity around a reset of the vulnerability with the scalp profiles ranging from -6 to 6 microvolts. Shaded areas represent a standard error of the mean calculated from the standard deviation of the subject means.
Table 2: Classification of GameTicks by Critical Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Classified by Likelihood</th>
<th>Fortress Kill</th>
<th>Fortress Respawn</th>
<th>Ship Death</th>
<th>Ship Respawn</th>
<th>Reset</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>F. Kill</td>
<td>7,695</td>
<td>425</td>
<td>132</td>
<td>136</td>
<td>195</td>
<td>1,159</td>
<td></td>
</tr>
<tr>
<td>F. Respawn</td>
<td>475</td>
<td>6,687</td>
<td>63</td>
<td>451</td>
<td>235</td>
<td>1,820</td>
<td></td>
</tr>
<tr>
<td>S. Death</td>
<td>49</td>
<td>40</td>
<td>591</td>
<td>12</td>
<td>68</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>S. Respawn</td>
<td>31</td>
<td>123</td>
<td>1</td>
<td>569</td>
<td>11</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>Reset</td>
<td>74</td>
<td>73</td>
<td>25</td>
<td>17</td>
<td>749</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>Null</td>
<td>164,756</td>
<td>390,063</td>
<td>13,312</td>
<td>76,326</td>
<td>97,403</td>
<td>1,087,838</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event</th>
<th>Classified by Posterior Probability</th>
<th>Fortress Kill</th>
<th>Fortress Respawn</th>
<th>Ship Death</th>
<th>Ship Respawn</th>
<th>Reset</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>F. Kill</td>
<td>2,470</td>
<td>1</td>
<td>26</td>
<td>0</td>
<td>4</td>
<td>7,241</td>
<td></td>
</tr>
<tr>
<td>F. Respawn</td>
<td>2</td>
<td>103</td>
<td>2</td>
<td>12</td>
<td>0</td>
<td>9,612</td>
<td></td>
</tr>
<tr>
<td>S. Death</td>
<td>8</td>
<td>0</td>
<td>457</td>
<td>0</td>
<td>20</td>
<td>352</td>
<td></td>
</tr>
<tr>
<td>S. Respawn</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>141</td>
<td>0</td>
<td>663</td>
<td></td>
</tr>
<tr>
<td>Reset</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>227</td>
<td>924</td>
<td></td>
</tr>
<tr>
<td>Null</td>
<td>3,275</td>
<td>153</td>
<td>1,792</td>
<td>1,644</td>
<td>1,271</td>
<td>1,821,545</td>
<td></td>
</tr>
</tbody>
</table>

other two events may reflect the fact that resets are not anticipated until they occur.

The same classification method described earlier was used to distinguish these 5 events from other game ticks as used for response classification. Given that critical events are rare, to prevent the classifier from being overwhelmed by non-events we limited the number of non-events in training the classifier by randomly choosing two non-critical game ticks for each critical game tick in the game. Once trained on other games the classifier was applied to all game ticks in the target game. Part a of Table 2 shows classification of all game vectors by likelihood of the data (d-prime = 2.00) and Table 2b shows classification by posterior probability (d-prime = 2.18). None of the 1080 games show negative d-primes. While this is considerably better than classification of key presses (Table 1), these classification results by themselves would not produce particularly good game reconstructions. In Part a of Table 2, 97.5% of all labels are false alarms to game-ticks that do not involve the ascribed event. In Part b of Table 2, the false labels are reduced to 70%, but now 84.4% of all critical events are missed.
Figure 6: Probabilities of transitions between events and distribution of intervals between events in the simulations. These statistics are averaged over border widths from 40 to 160 pixels.

In addition to these problems, identification of critical events alone does not provide the detailed behavior of the subject that produced them.

3.4. Model-based Distributions of Events and Transitions between Events

Despite the relatively high d-primes, the classifier produced many sequences of critical events that were unlikely, impossible, or in a plausible sequence but after unlikely or impossible delays. The strongest game constraints that the classifier typically violated in its reconstructed games are that fortress respawns must occur 31 game ticks after fortress kills and that ship respawns must occur 31 game ticks after ship kills. Aside from these structural game constraints, in human play different sequences of events and the timing intervals between those events are more or less likely. Figure 6 shows the distributions in the model be-
tween other events and the probability that one event will follow another. These distributions and probabilities also vary with the width between the borders (see Figure 3), with the most dramatic effect being on the probability of a death.

We used an HSMM to efficiently combine these model-based statistics and the conditional probabilities from the EEG classifier to estimate the most likely sequence of critical events in a game. Any sequence of events can be denoted \( a_1, a_2, ..., a_n \) occurring at game ticks \( t_1, t_2, ..., t_n \) where \( a_1 \) is the start of the game (hence \( t_1 \) is the first game tick), \( a_n \) is the end (hence \( t_n \) is the last game tick), and the rest are fortress kills and respawns, ship deaths and respawns, and resets. The following proportionality describes the probability of any such sequence relative to the probability of other sequences:

\[
\text{Prob}(a_1, a_2, ..., a_n) \approx \prod_{i=1}^{n-1} t(a_i, a_{i+1}) \times f(t_{i+1} - t_i | a_i, a_{i+1}) \times P(EEG(t_{i+1} : t_{i+1}) | a_{i+1})
\]

where \( t(a_i, a_{i+1}) \) is the probability of transition between the events \( a_i \) and \( a_{i+1} \), \( f(t_{i+1} - t_i | a_i, a_{i+1}) \) is the probability of the \( t_{i+1} - t_i \) game ticks between the events \( a_i \) and \( a_{i+1} \), \( P(EEG(t_{i+1} + 1, t_{i+1}) | a_{i+1}) \) is the conditional probability of the EEG signal for this period if it ends in \( a_{i+1} \). The conditional probabilities come from the classifier. Their use can be made much more efficient and robust by using the fact that, if the signals at different ticks were independent:

\[
P(EEG(t_{i+1} : t_{i+1}) | a_{i+1}) = P(EEG(t_{i+1}) | a_{i+1}) \times \prod_{x=t_i+1}^{t_{i+1}-1} P(EEG(x) | \text{Null}) = \frac{P(EEG(t_{i+1}) | a_{i+1})}{P(EEG(t_{i+1}) | \text{Null})} \times \prod_{x=t_i+1}^{t_{i+1}} P(EEG(x) | \text{Null})
\]

The product involving the conditional probabilities \( P(EEG(t) | \text{Null}) \) will be the same for all sequences of critical events and can be ignored in determining which sequence has the highest proportionality. Therefore we can rewrite
Proportionality 1 as

\[ \text{Prob}(a_1, a_2, ..., a_n) \approx \prod_{i=1}^{n-1} t(a_i, a_{i+1}) \times f(t_{i+1} - t_i | a_i, a_{i+1}) \times \frac{P(\text{EEG}(t_{i+1}) | a_{i+1})}{P(\text{EEG}(t_{i+1}) | \text{Null})} \]

Thus, the critical EEG quantity is ratio of probability of the signal given a critical event to the probability of no critical event. Because these ratios involve events that are far apart, temporal correlation will be weak and the independence assumption will be approximately correct. The HSMM-MVPA applies to all game ticks, including the 4.7% that do not have a corresponding signal vector. For these game ticks the ratios were 1 for all critical events. Thus, for these stretches of time without signal, the only information about possible events comes from the transition probabilities and distribution of delays between events.

We used the Viterbi algorithm (Rabiner, 1989) for hidden semi-Markov models to find the assignment of events that maximized \( \text{Prob}(a_1, a_2, ..., a_n) \). This produced for each game a set of inferred events and the game ticks at which they occurred. For each game the match between the assigned events and the actual events was calculated as a sum of a recall and a precision measure (Buckland & Gey, 1994) calculated from locations of kills, deaths, and resets (since the respawns of the fortress and ship were tied to the kills and deaths). The measure of recall focused on the events that occurred in the game and identified the closest predicted event. If that event type matched and was within 2.5 seconds it was scored according to how many game ticks it was away — thus the maximum score was 75. If the closest event was further away or failed to match it was also scored 75. The average of these recall scores for a game can vary from 0 (perfect match of all events) to 75 (worst possible). The measure of precision applied the same scoring procedure but now started with all predicted events and found the closest actual event. These recall and precision measures were identical for 379 of the 1080 games, but can be somewhat different for the rest because of differences between the actual game versus the reconstruction in number critical of events or their timing. Even when they are different they do
Figure 7: Distribution of summed precision plus accuracy ratings on a scale where 0 is perfect match between game and reconstructed critical events and 150 is the maximum possible mismatch score.

tend to be correlated (r = .624). The average measure of recall was 14.1 and the average measure of precision was 11.8 for a total average match rating of 25.9 out of 150.

Figure 7 shows the distribution of reconstruction scores (sum of recall and precision). To provide a chance measure one can use the average rating between actual events and predicted events of other games, which is 95.2. 847 games are best matched by the game constructed from their EEG signal rather than any other reconstructed game. 9 games have all critical events predicted perfectly to the game tick. The mean rank of the reconstruction of a game out of the 1080 reconstructions is 7.7 (chance ranking would be 540.5).
Figure 8 shows a pair of games to illustrate the range of prediction. Part a is the 276th best-matched game with a rating of 10.6. All events but the last kill are closely predicted – the model does not complete its last kill by game’s end. Part b is 787th best-matched game with a score of 36.4. 8 of the actual events (6 kills and 2 deaths) are identified relatively accurately. However, one death, one reset, and 2 kills are missed while one kill and one death are predicted that did not occur.

Table 2 reported how well the classifier labeled game ticks. We constructed a similar matrix from the results of the Viterbi algorithm looking at how well it classifies each game tick. Table 3 presents those results, which have a d-prime of 2.75, superior to both matrices in Table 2. All 20 subjects are classified better using the Viterbi algorithm than just the classifier (either method in Table 2). Moreover, the Viterbi algorithm results include classification of game ticks without a signal vector. Also, Figure 8 shows that even if the Viterbi algorithm does not identify the exact game tick it often identifies a nearby game tick. Most important and not reflected in Table 3, the resulting positioning of events render a coherent interpretation of game play, which is to say that one can create sequences of key presses that would produce these critical events. While this is

\textsuperscript{7}Table 3 is most comparable to Table 2b, which incorporates information about the frequency of the events.
Table 3: HSMM-MVPA Classification of Game Ticks by Critical Events

<table>
<thead>
<tr>
<th>Classified by Viterbi Algorithm</th>
<th>Classified as</th>
<th>Fortress Kill</th>
<th>Fortress Respawn</th>
<th>Ship Death</th>
<th>Ship Respawn</th>
<th>Reset</th>
<th>Null</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F. Kill</td>
<td></td>
<td>4,853</td>
<td>12</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>5,251</td>
</tr>
<tr>
<td>F. Respawn</td>
<td></td>
<td>12</td>
<td>4,853</td>
<td>2</td>
<td>10</td>
<td>1</td>
<td>5,250</td>
</tr>
<tr>
<td>S. Death</td>
<td></td>
<td>20</td>
<td>2</td>
<td>346</td>
<td>1</td>
<td>1</td>
<td>560</td>
</tr>
<tr>
<td>S. Respawn</td>
<td></td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>340</td>
<td>0</td>
<td>547</td>
</tr>
<tr>
<td>Reset</td>
<td></td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>163</td>
<td>1</td>
<td>1,070</td>
</tr>
<tr>
<td>Null</td>
<td></td>
<td>5,243</td>
<td>5,242</td>
<td>425</td>
<td>419</td>
<td>683</td>
<td>1,908,653</td>
</tr>
</tbody>
</table>

3.5. Stitching in Exact Game Play

Stitching involves finding sequences of key presses (fires and thrusts) in simulated games that would produce the same timing of critical events as in reconstructed sketches like those in Figure [8]. It is unlikely to find a complete simulated game whose critical sketch perfectly matched a reconstructed sketch. Even though there were 2750 simulated games for each border width, no simulated game had a critical sketch matched the critical sketch of any other simulated game or actual subject game (which makes the fact that 9 of the Viterbi-reconstructed critical sketches perfectly matched the actual games quite remarkable). Rather than looking for complete games that matched, the stitching procedure just searched for segments of simulated games that could reproduce segments of the reconstructed sketch. Given that the effects of firing and thrusting are nearly independent,[8] the procedure selected thrust patterns and

---

[8] There are rare possibilities for an interaction: A fire sequence can change the consequences of a thrust sequence when the fires destroy the fortress and saves the ship from being shot when it is flying too slowly. A thrust sequence may change the consequences of a fire sequence because it controls the distance of the ship, which determines when a shot hits the fortress, potentially turning a vulnerability reset into an increment or vice versa.

---

27
fire patterns from the simulated games independently.

Thrusts determine the flight path of the ship. The overall path in a game can be divided into a number of episodes that begin with the ship flying in a starting configuration and ending either in a death or terminating with the end of the game. For critical sketches of games with no deaths, there is no shortage of simulated games without deaths for that border width that can serve as candidates for reconstructing flight paths. Beginning segments of these games also provide thrust sequences for episodes in critical sketches that go from respawning of the ship after a death to the end of the game. The challenging episodes to match are segments in critical sketches that go from the appearance of the ship until a death. There are nearly 1800 possible numbers of game ticks that can pass until death. Even if they were equally likely (which they are not), ten times more simulations than the current 35,750 simulated games in the library would be required to come close to guaranteeing model runs with a death at each game tick for each width. As it were, ship deaths did not occur at 32% of possible game ticks in the inferred critical sketches for a border width. We relaxed the criterion and selected any deaths within 5 ticks of the desired duration. This reduced the number of missing cases to 2%. 42 cases were still not covered and for these we expanded the difference until there was a case. With these relaxations, there were on average 21 simulated sequences leading to a death for each death in a critical sketch. Unlike the case with the earlier action classifier, these were plausible thrust sequences that produced a plausible flight pattern even if it ended in a death.

The EEG signal was used to select among the candidate model thrust sequences, exploiting information that the signal provides about where the ship is on the screen. Part a of Figure 9 shows the average scalp activity when the ship is in different 30-degree sectors. We trained a classifier to recognize which sector the ship was in plus when the ship was off the screen due to a death. Using the likelihood of the EEG signal, the accuracy of the classifier in discriminating among these 13 categories was \( d' = 0.57 \). We fed the candidate
sequences of thrusts into the game engine to produce flight paths and took the sequence whose flight path had the greatest summed log probability from the EEG classifier for ship position.

Part b of Figure 9 shows the offset between the angle of the reconstructed flight and the actual angle in the line called Informed Stitching. For comparison, the figure also shows the results of three other bases for positioning the ship:

1. **Key Classifier.** The ship positions produced by feeding into the game engine the thrusts from the key classifier using the posterior criterion (Part b of Table 1).

2. **Position Classifier.** The posterior most probable ship positions according the position classifier (Part a of Figure 9). Note that, because these positions are not produced by the game engine, they typically do not correspond to a possible flight pattern.

3. **Random Stitching.** Feeding into the game engine thrusts from segments of the right length without regard to the EEG signal of the subject.

For Informed Stitching the average distance between reconstructed ship and actual ship is 40 degrees; for Random Stitching it is 55 degrees, for the Position Classifier it is 68 degrees, for the Key Classifier it is 71 degrees (randomly placing
the ship on the screen would produce an average distance 90 degrees).

The stitching procedure needed to find sequences of fires that were typically shorter than the thrust sequences: sequences that would span the time in the critical sketch from either a fortress respawn or a reset (either starting the vulnerability at 0) to a kill or a reset. There tended to be many candidates from the model for reconstruction of fire sequences. Another classifier was constructed to select the most probable of these candidate sequences by matching the build up of vulnerability. To trace the build up of vulnerability, the classifier learned to recognize different categories of fires by how they changed the vulnerability. Part a of Figure 10 illustrates how the EEG signal changes with the build up of vulnerability. Altogether there were 17 categories of game ticks that the classifier had to distinguish among:

1-11: Game ticks with fires that raised the vulnerability to 1 through 11. In the averaged Figure 10a these would be the game ticks associated with the first 11 vertical lines.

12: Game ticks with fires that raised the vulnerability to more than 11 but were too slow to destroy the fortress (not shown in part a of Figure 10).

13: Game ticks with fast fires that killed the fortress after reaching a vulnerability of at least 11 (the last vertical line numbered 12 in part a of Figure 10).

14: Game ticks with fast fires that reset the vulnerability to 0 before reaching a vulnerability of at least 11 (not shown in part a of Figure 10—typically the fire will be about 150 ms before the reset which is time 0 in part e of Figure 5).

15: Game ticks with both the ship and fortress present without a fire (the majority of game ticks).

16: Game ticks with the fortress absent because of a kill. (ticks covering 0 to 0.5 sec. in part a of Figure 5 and -0.5 to 0 sec. in part b of Figure 5).

17: Game ticks with the ship absent because of a death (ticks covering 0 to 0.5 sec. in part c of Figure 5 and -0.5 to 0 sec. in part d of Figure 5).

Focusing only on the ability to discriminate among the first 15 categories, which are the game ticks that occur in a fire sequence, the d-prime for discrim-
Figure 10: (a) EEG warped to the period of an average kill (4.12 seconds – only games with exactly 12 fires included). The black lines show the fires that increase the vulnerability to 11 with the last fire being the fire (12) that destroys the fortress. They are numbered with the vulnerability they produce. The EEG profiles between vulnerability changes have been warped to the average vulnerability intervals. The scalp profiles are for .4 seconds, 2.25 seconds, and 4.0 seconds. (b) Distance from an assigned fire to a fire with a matching vulnerability in the actual game. Negative values mean that the fire came before the matching actual fire and positive values mean it came after.

Figure 10 illustrates the accuracy in shot placement. It plots the time between a shot with an inferred effect on vulnerability and the closest actual shot that had the same effect on vulnerability. Part b of Figure 10 plots the distribution of offsets for:

1. **Key Classifier.** The vulnerabilities produced by feeding into the game engine the fires from the key classifier using the posterior criterion (Part b of Table 1).
2. **Vulnerability Classifier.** Game ticks assigned that vulnerability by the classifier. As this was not run through the game engine the sequence of vulnerabilities often was not a possible sequence.
3. **Random Stitching.** Feeding into the game engine fires from segments of the right length to match the critical sketch without regard to the EEG
signal of the subject.

4. **Informed Stitching.** Selecting the segment that produced the best match to the EEG signal of the subject and feeding that into the game engine.

The Key Classifier locates 3% of the fires within .1 sec of a fire with the same vulnerability change, Vulnerability Classifier 15% of the fires within a .1 sec of a fire with the same vulnerability change, Random Stitching 32%, and Informed Stitching 43%.

While Informed Stitching, which uses both the game library and the classifier, does best both for ship position (Part b of Figure 9) and vulnerability (Part b of Figure 10), it is more important to have a coherent segment of the right length (Random Stitching) than it is the use the results of the stitching classifiers alone (Position and Vulnerability Classifiers). This reflects the importance of aligning the thrusts and fires with the identification of the critical events.

### 3.6. Summary Evaluation of Reconstruction Accuracy

The combination of EEG classification and the model can do well at reconstructing the critical events (Figures 7 and 8) and capturing where the ship is in space (Figure 9) and what the fortress vulnerability is (Figure 10). To provide a summary measure of the overall correspondence between actual games and their reconstructions we used an equally weighted sum of three factors:

1. The z-score of the rating of the critical events (Figures 7 and 8). As noted earlier, the reconstruction scores well on this measure with 818 of the 1080 games best predicted by their reconstruction and a mean ranking of the reconstruction is 8.4.\footnote{These are not exactly the same as the values reported with respect to Figure 7 because slight differences in the timing of the deaths and occasional unintended interactions between fire and thrust sequences in the reconstructed games.}

2. A z-score of the distance between the position of reconstructed ship and the actual ship averaged across the game ticks. While much better than
chance this detail is less well reconstructed, with only 106 of the 1080 games best predicted by their reconstruction and a mean rank of 176.2 out of 1080.

3. The z-score of the difference between the vulnerability of the reconstructed ship and the vulnerability of the actual ship averaged across the game ticks. 803 of the games are best matched by their reconstruction and the mean rank is 14.2.

While we weighted these three equally in coming up with a combined measure, they seem ordered as above in how compelling the reconstruction is as match to the original game. By this combined measure 846 of the games are best predicted by their reconstruction and the mean ranking of the reconstruction is 6.1. The highest ranked seem quite compelling as reconstructions of the original game. Only one was worse as a reconstruction of the game than the average of the reconstructions of other games.

4. Discussion

The focus in this paper has been reproduction of the details of game play. A less detailed reproduction may be adequate for many purposes. For instance, one might just want to assess how well the person has played the game. One can used these detailed reconstructions for such summary evaluations. Figure [11] shows the correspondence between the final scores of reconstructed and actual games, for which the correlation is .848.

While we have focused on reproducing the subjects’ actions because that is where we have the ground truth of the game record, the models also make commitments about the cognitive processes giving rise to these actions. For instance, as described in [Anderson et al., 2019] decisions about how to pace fires is made by an evolving sense of the appropriate pacing. One could infer the threshold a subject is currently using to time fires by considering the threshold in the model segment that has been stitched in. Similarly, one could infer the subject’s threshold for thrusting from the model’s threshold. If there were
alternative strategies for game play, one could infer the subject’s strategy by which strategy provided the matching segments for stitching. One could use these inferences to provide feedback on what the subject needed to do to improve their performance.

The Space Fortress game has provided a good testbed for judging reconstruction from EEG. The domain is one where small differences in the timing of events can greatly affect the course of subsequent events. There is a strong sequential dependence where the earlier actions can change the consequences of later ones. The approach described in this paper handles these challenges well, and is capable of creating compelling reconstructions of game play. The approach combines training of classifiers to detect events and using a computational model to provide the statistical information about the distribution of events and the details about how these events were achieved. While reconstruction performance is good, it might be improved by better approaches to classification or tuning cognitive models to individual player’s style. However, the current results do establish the potential of combining classification and
cognitive models to reconstruct mental activity.

The complete game record of Space Fortress provides a reliable ground truth for training classifiers and for judging the success of reconstructions. In other work (e.g., Anderson et al. 2016) we have had similar success in parsing EEG signals using the HSMM-MVPA method without such training events. Rather, HSMM-MVPA discovered critical events in the absence of labelling. However, the intervals were much briefer and the EEG signal was constrained to be a simple ERP-like bump. Although those conditions accurately represent the vast majority of ERP studies with brief trials, this unsupervised method did not scale to the current situation with much longer intervals.

Both the classification and model-based aspects of this approach are critical if one wants to be able to reconstruct complex cognition that spans extended periods. The cognitive processes create EEG signals that classification methods can use to guide reconstruction. However, classification algorithms will always have errors that result in implausible or impossible reconstructions. Therefore, one needs a model that represents cognitively plausible constraints on the reconstruction.

The Sketch-and-Stitch method uses information from classification and the model at both the Sketch and the Stitch level. The Sketch level takes advantage of both the strong signals associated with critical events and statistical information about the order and timing of those events from the model. The Stitch level required the model to provide sequences of actions of the appropriate length to stitch in and selected among these according to their EEG signatures. Even if the procedure failed to choose the right detail for a period between two critical events, the reconstructed game stayed on a path that was consistent with the sketch. Thus, even after a period of significant mismatch between reconstruction and subject, the two would often come back to close correspondence.

At an abstract level, the approach in this paper bears similarities to BCI efforts for merging statistical regularities in language with EEG signals using a Bayesian inference strategy. These regularities can be leveraged to tailor the presentation of letters on an external device based on their base rates and pos-
terior probabilities given the currently typed word. Likewise, these regularities can be used to support inference of intended spelling using EEG signals [Mora-Cortes et al., 2014]. Our cognitive model effectively provides a "grammar" of gameplay and action. More generally, using such cognitive models could enable successful application of EEG decoding to a vastly wider range of tasks.

5. Acknowledgements

This work was supported by the Office of Naval Research Grant N00014-15-1-2151. We thank Cvetomir Dimov for his comments on the paper. The data and analyses which were used to create the figures in the paper are available at http://act-r.psy.cmu.edu/?post_type=publications&p=31867.
References


