Meta-T: Tetris[®] as an experimental paradigm for cognitive skills research

John K. Lindstedt · Wayne D. Gray

Published online: 12 March 2015
© Psychonomic Society, Inc. 2015

Abstract Studies of human performance in complex tasks using video games are an attractive prospect, but many existing games lack a comprehensive way to modify the game and track performance beyond basic levels of analysis. Meta-T provides experimenters a tool to study behavior in a dynamic task environment with time-stressed decision-making and strong perceptual-motor elements, offering a host of experimental manipulations with a robust and detailed logging system for all user events, system events, and screen objects. Its experimenter-friendly interface provides control over detailed parameters of the task environment without need for programming expertise. Support for eye-tracking and computational cognitive modeling extend the paradigm's scope.

Keywords Cognitive skills · Behavioral methods · Eye tracking methods · Experiment design · Cognitive tasks · Levels of analysis · Log files · Games · Tetris

Introduction

Ideally, researchers who seek to understand complex, cognitive behavior in dynamic task environments would have access to: (1) a task of societal and/or commercial importance, (2) a population of expert performers, (3) a population of novices who can be motivated to acquire expertise in the

task, and (4) task manipulation and data collection tools of high sensitivity.

Traditionally, there have been disconnects between the first three and the last of this list. Experts and populations of motivated novices exist for important real-world tasks such as laparoscopic surgery (Keehner et al., 2004; Keehner, 2011), piloting aircraft and helicopters (Hays et al., 1992; Proctor et al., 2007), detecting enemy submarines hiding in deep waters (Ehret et al., 2000), and call-center operators (Gray et al., 1993). Whereas, data collection tools capable of recording hand and eye movements to the nearest millisecond, and experimentation tools capable of manipulating the task environment in ways designed to isolate or highlight theoretical factors of interest, exist in the university laboratory. Fortunately, the world has turned towards a direction more favorable to the university researcher.

Although we do not argue that our tasks are of societal or commercial importance,¹ we do argue that with the spread of video gaming, university-based researchers now have ready access to a population of performers who have put in the hundreds and thousands of hours needed to achieve true expertise. What we have not had, however, are the tools to manipulate and collect data from these populations of experts in their tasks of expertise. We have not been able to use our eye trackers to collect location information for dynamic regions of interest in video games. We have not been able to track or collect mouse or keyboard or joystick data with millisecond accuracy. Perhaps, most importantly, we have not been able to manipulate proprietary software so

J. K. Lindstedt (✉) · W. D. Gray
Cognitive Science Department, Rensselaer Polytechnic Institute,
Troy, New York, USA
e-mail: john.k.lindstedt@gmail.com

¹ We do, however, recognize that others might make these arguments.

as to create reproducible, factorial, experimental paradigms. In short, we have not been able to manipulate the tasks or to collect user data, event data, and system data at the level of detail required to address *our research questions of interest*.

Enter Tetris®

Tetris®² meets our second and third criteria as most campuses have a population of expert Tetris players as well as novices who can be motivated to put in the hours needed to acquire, at least, moderate expertise. For example, several researchers have investigated the near- and far-transfer effects of Tetris in reducing gender differences in spatial skills (Okagaki & Frensch, 1994; Sims & Mayer, 2002; Terlecki et al., 2008) or enhancing the spatial skills of Engineering students (Martin-Gutierrez et al., 2009). Others (Haier et al., 2009) have used Tetris to search for general cognitive enhancements from training by examining the potential growth of associated brain areas in subjects who played Tetris. These studies varied hours of Tetris play and looked at how the performance measures provided by the game (such as total score, level of play, and number of lines cleared) correlated with measures of spatial ability.³

An approach that focused less on overall Tetris performance and more on details of game play was taken by Kirsh and Maglio (1994) and Maglio and Kirsh (1996) to advance the general (i.e., not specific to Tetris) claim that the use of epistemic actions increased with expertise. By *epistemic action* they meant physical actions that generate information rather than implementing an immediate goal. For example, when trying to decide if a Tetris zoid (i.e., piece) could fit into an available space, Tetris players could obtain the same information by either mentally

(as per Shepard and Metzler (1971)) or physically rotating it. Physical rotation, in lieu of mental rotation, would be considered an epistemic action.

Though the Kirsh and Maglio papers are short on some specifics, for their 1996 paper (Maglio & Kirsh, 1996) they collected data from two novice Tetris players at various periods during 20 hrs of Tetris play. It is not clear from their reports whether keystroke and event data were logged to a computer file, recorded by videotape, or tallied by observers in real time. However, getting beyond the performance scores provided for game players to keystroke and event data clearly required a lot of effort. In contrast, to get at the same sort of data that Kirsh and Maglio report, we wrote a simple flash program with our own logging software. This software allowed us (Lindstedt & Gray, 2011) to collect detailed response time, system event, and screen object data from 59 experienced Tetris players as they participated in two studies with our software. Our much more detailed data allowed us to algorithmically distinguish between five definitions of epistemic action to conclude that while there was some evidence that weak players exhibited epistemic actions, the frequency of epistemic action decreased with moderate expertise and vanished by extreme expertise.

This first study with experienced Tetris players in an experimental paradigm (Lindstedt & Gray, 2011) convinced us that there were many interesting questions ready to be asked beyond game performance score or beyond the use of Tetris as a treatment condition in studies of spatial skill acquisition (e.g., as per Haier et al. (2009), Sims and Mayer (2002), and Terlecki et al. (2008)). Indeed, though Tetris is more complex than most experimental psychology paradigms, we believe its complexity is manageable—at least for certain research questions

²“Tetris” is a registered trademark of Tetris Holdings, LLC for video games, online computer games and other goods and services. We, the authors, make no claim to any right to use the term “Tetris” other than to refer to such games as they exist on the market, and as played by our target subject population.

³A *Web of Science* search conducted on 2014.10.13 found 118 papers in fields as diverse as computer science, psychology, engineering, neuroscience, educational research, robotics, psychiatry, linguistics, and rehabilitation that used Tetris or players with experience playing Tetris in their research. Our personal files contain 15 conference or journal papers not found in the WoS search, which bring the total scholarly papers using Tetris experienced players to a minimum of 133.

Fig. 1 Growth curve of players' average lines cleared as practice days increased. Reprinted from Haier et al. (1992)

Fig. 2 An illustration of the task environment as it appears to players

of interest. In particular, it provides us with a manageable domain in which to study skill acquisition, control of cognition, strategy discovery, time-stressed decision-making, and the interactions among cognition, perception, and action in a dynamic task environment. What is missing from Tetris *as an experimental paradigm* is the ability to easily manipulate isolated parts of the game environment.

Here is where Meta-T, an experimental paradigm analogous to Tetris developed in our laboratory, succeeds. Take, for example, Fig. 1 (Haier et al. 1992): this growth curve depicts only one simple measure of performance, the average number of lines cleared. This top-level response variable leaves much to the imagination; what were these players doing? Were they strategizing better as they progressed, or were they simply getting faster at manipulating the game's controls? Was the change in performance due to something general or something more task-specific? If this experiment had employed Meta-T, a detailed picture of human skill acquisition would be available for scrutiny.

In what follows, we first describe the classic Tetris game for those few readers who have never had serious exposure

to it. We then describe the experimental manipulation tools and data logging capacities of Meta-T.

Tetris for the uninitiated

As a cognitive task, Tetris hits the sweet spot of being simple to comprehend and tractable to analyze, but complex enough to remain cognitively interesting and rewarding to master. The game space is simple to parse (both by humans and machines), and though a novice may be taught the game's rules in a matter of moments, to become highly proficient at the task, players must acquire strategies, methods, or procedures⁴ that efficiently deploy working memory, mental rotation, strategic planning, prediction, manual dexterity, etc., to negotiate the ever increasing time pressure of the task. It is no simple feat to become a Tetris grand

⁴We are not sure if these three are inherently distinct, however, we mention all three for completeness.

champion⁵ (i.e., a nation's highest-scoring player, as determined by a championship tournament), and to that end it is an excellent space in which to examine the phenomenon of human skill acquisition. An additional bonus is that Tetris is not a boring experimental paradigm, but a fascinating game that has a life outside of academia: people already exist who go to great lengths to perform well in the task, due entirely to the fact that the game is fun and engaging.

Rules of Tetris In the game, players use keystrokes to manipulate a series of falling shapes (*zoids*) into an arrangement of previously placed zoids at the bottom of the game space (the *pile*). The players' goal is to maximize their score by filling and clearing rows in the pile. Once a row is filled, it is removed, the pile is lowered, and points are awarded. Since it is not always possible to clear lines, the pile gradually rises. The game ends when the pile rises above the upper threshold of the game space. A basic representation of a game board is shown in Fig. 2. Interestingly, despite its widespread appeal, Tetris is unwinnable. If you play it long enough, you will lose (Baccherini & Donatella, 2008; Kendall et al., 2008; Fahey, 2013)!

There are seven unique shapes for zoids, one for each permutation of four contiguous block segments. The time between a zoid appearing and it being placed into the pile defines an *episode*. The player can move the zoid left, right, or rotate it; however, while this is happening, the zoid will continuously and automatically drop from the top of the game space. At the start of the game this drop rate is very slow, offering players time to deliberate; however, as the player clears lines, the game level increases, and the zoids begin dropping much more quickly offering very little time for deliberate thought. To add an incentive for achieving higher levels of difficulty, the amount of points awarded increases multiplicatively with level of difficulty.

Very important to the game is that the points increase when multiple lines are cleared at once. In the original version of Tetris, at level 1 (i.e., the start of the game with a very slow drop rate), a single line cleared is worth 40 points. Clearing two lines at once is worth 100 points, three are worth 300 points, and four yields 1,200 points. In contrast, by the time a player reaches level 20, clearing one, two, three, and four lines simultaneously are worth 800,

2,000, 6,000, 24,000 points, respectively. Incidentally, clearing four lines at once is known as “scoring a Tetris,” the game's namesake, and is notable because of its high payoff and difficulty.

Cognitive depth of Tetris For the purposes of investigating cognitive phenomena and effects, Tetris may not appear at first to be “deeply cognitive”. Tetris does not offer the depth of forethought required by Chess, the broad situational awareness of Go, or abstract thinking involved in formal logic, but does require the use of these cognitive abilities *in real time*. Compared to many classic laboratory paradigms for testing cognitive effects (i.e., N-Back or AX-CPT), Tetris is a rather complex task in which players need to respond to immediate time pressures, executing chains of manual commands while simultaneously planning ahead for known upcoming zoids (displayed in the Preview box, hereafter referred to as the Next Zoid); moreover, if players wish to successfully score high numbers of points, they need to plan many zoids ahead, over uncertainty, to successfully create the opportunity to perform the high risk but rewarding “Tetris” maneuver. The task can be so demanding that players may need to rely on epistemic actions to succeed at intermediate-level play (Kirsh & Maglio, 1994; Maglio & Kirsh, 1996). Though Tetris may not be the ultimate in interactive cognitive paradigms, we respectfully suggest that Tetris represents a challenge to understanding the interaction between perception, action, cognition, and decision making, and that understanding this challenge would mark a great advance in the state of our art!

Meta-T: Tetris as experimental paradigm.

Meta-T provides experimenters with a host of experimental manipulations and a robust and detailed logging system for all user events, system events, and screen objects at millisecond precision. Its experimenter-friendly interface provides control over detailed parameters of the task environment without need for programming expertise.

In what follows we walk through the process of using Meta-T to plan and execute an experiment comparing game performance for a player both playing the game: 1) under normal circumstances (not guided), and 2) with the presence of some visual stimuli designed to ease the execution of placing a zoid (guided). In this walk-through we will: (a) install Meta-T, (b) explain

⁵e.g., <http://www.youtube.com/watch?v=eQOswiAGLU4> and <http://www.youtube.com/watch?v=gbyCSFrccM>

the details of the program's experimental configuration, (c) document features for data collection during the experiment, and (d) show the results of our example experiment.

Installation

The Meta-T software package uses the version control software Git, and is privately hosted on the GitHub website. Those interested in downloading the software should: 1. Create an account with GitHub. 2. Contact the author at email:lindsj3@rpi.edu to gain access to the repository. 3. Log into the GitHub website. 4. Visit the project page, located at <https://github.com/CogWorks/Tetris>.

From here it is possible to install the package by either using the Git software to clone it to the desired computer or by downloading a compressed archive of the current version.

Meta-T is licensed under a GNU General Public License. As the original authors⁶ of the software, we (1) assert copyright on the software, and (2) offer legal permission to copy, distribute and/or modify it. This provision allows other researchers the right to carry on development of the software for their own purposes.

In addition to installing our software, Meta-T requires: (a) a version of the Python programming language prior to Python 3.0 (Python 2.7 preferred), (b) PyGame, a python package for video game development, (c) Numpy, a python package for streamlining mathematical operations, and (d) Zope, and (e) Twisted, which are packages for handling threaded programming. These dependencies are easily installed using the common package manager “pip” for python. From a command line, one can simply run the code:

```
pip install pygame [or numpy, or zope, etc.]
```

Alternatively, one can search for and manually install these packages. Once these dependencies are installed, the program will be runnable from the Meta-T package's root directory.

⁶Our contributions include implementing the algorithms we copied from publicly available websites and developing the ideas, algorithms, and code for data logging and experimental manipulations. Those desiring a free game of Tetris will not find our software worth their trouble to download, install, or use. Indeed, we reserve the right to distribute our software only to members of established research entities.

Configuring an experiment

The default Meta-T game environment resembles the classic 1985 *NES*⁷ *Tetris*. The rules of the game match those outlined in the online, community-driven Tetris resource wiki⁸ (we detailed some of these rules in the previous section). This version is considered one of the most difficult as it offers little leniency for the indecisive player, and its difficulty increases more rapidly than in many newer versions of Tetris.

Classic Tetris provides a starting point, but has features that may not be desired by the experimental researcher: game length is unconstrained, zoid sequences are fully randomized, highly salient game music is continuously present, and the default difficulty curve may be too steep. Furthermore, there is no interesting experimental manipulation present: no game rules or visual elements are tweaked to produce differential performance among experimental groups. Foreseeing this, the Meta-T program implements a system of text-based configuration files, easily altered to create customized experimental configurations.

Configuration files for Meta-T are simple text documents (with the “.config” extension) wherein experimenters can override the default parameters of the baseline Meta-T task environment. These files can be seen as analogous to the circuit board of an old video arcade cabinet, to which the experimenter is granted full access for customizing their Meta-T task! A complete list of manipulable parameters is available in the documentation included in the “default.config” file, which (as its name suggests) is the .config file loaded if another file is not explicitly provided.

Configuration files can be sparse, meaning they do not require every variable to be defined. Therefore, with a caveat discussed below, any customized configuration file needs to include only a simple list of the overridden parameters, and not a listing of all parameters. This feature keeps the process of defining a new task environment simple.

For the example experiment, we investigate the effect of visual aids on game performance. We begin by defining one configuration file for each condition. Each file consists of *FIXED parameters* and *VARIED parameters*. The *FIXED* parameters are the same across conditions and are those

⁷Nintendo Entertainment System

⁸<http://tetris.wikia.com/>

that distinguish all experimental conditions from the default Meta-T game. For example, in both conditions we want each player to play four games (one FIXED parameter) using the same random number generation seed (another FIXED parameter), and we want to limit all games to 100 episodes, rather than until the game is lost, which is the default (our third FIXED parameter).

#FIXED parameters (listed in each config file)

#Presentation

fullscreen = true

music_vol = 0.0

#Randomization

fixed_seeds = true

random_seeds = 1, 2, 3, 4, 5

permute_seeds = true

#Game length

continues = 5

max_eps = 100

#Game difficulty

starting_level = 4

lines_per_lvl = 9000

#Player actions allowed

pause_enabled = false

The *VARIED parameters* are those that differ between the conditions. For example, rather than setting the number of zoids per game to 100 for each condition, it may be the case that the experimenter wants one condition to have 100 zoids per game and the other condition to have

50. Note that in some cases, one condition will use the default setting of a parameter whereas that setting will be varied in another condition. (In this situation, it is good documentation to list the default setting in that condition's .config file and to add an annotation to that file stating that you are using the default setting in condition A but not in condition B.)

There are five primary types of definable parameters in the .config files: string, integer, floating-point numbers, integer list, and Boolean. Strings are straightforward literal entries, accepting any entered text; integers accept any whole-number; floating-point numbers accept decimal values; and integer lists are integers separated by commas. Booleans are true or false flags that accept any of the following values as “true”, regardless of capitalization: “true”, “t”, “yes”, or “y”. All other values result in a value of false, though it is advisable to explicitly use “false” or “no”.

For a complete listing of all possible configuration file parameters, see either Appendix A, or the “README-configs.rst” file in the root directory of the Meta-T package.

Let us examine the parameters configured by this segment.

1. Presentation:

fullscreen: Ensure the computer screen is filled with the task environment to avoid distractions.

music_vol: Turn off the game music, as it may be differentially distracting to some players.

2. Randomization:

fixed_seeds: Indicates that the experimenter wishes to use a fixed set of numerical “seeds” for the random number generator. If set false, the following will be ignored.

random_seeds: Provide the environment with the list of random number generator “seeds”. A given seed will always produce the same sequence of zoids.

permute_seeds: Within a single session, randomize the list of seeds without replacement to avoid any sequence effects (if more games are played than seeds are provided, the sequence will loop but not be re-randomized. As such, if re-randomization is desirable, setting up separate sessions is advised). If set false, seed order will remain fixed.

3. Game length:

continues: Allow the player to play a maximum of 5 games in a given session.

max_eps: Constrain the number of episodes (zoids placed) to be played in a game to 100 to help ensure comparable observations. (By default, players play until they lose. For players of exceptional skill this could entail thousands of zoids.)

4. Game difficulty:

starting_level: This example begins the player at level 4 rather than the slower (and easier) default level 1.

lines_per_lvl: With only 100 zoids per game (see max_eps, above) setting the number of line-clears needed to advance in level to 9000 means that each 100 zoid game will be played entirely at level 4 (see starting_level, above).

5. Player actions allowed:

pause_enabled: Remove the ability for the player to disruptively pause the game during data collection.

To summarize the result of this combination of FIXED parameter settings: Each participant will play five games in a session, all of which will consist of 100 episodes of moderate difficulty. The randomized sequences of zoids within each game are controlled and the same sequences will be presented in random order to all participants. Visual and auditory distractions are minimized, as is the ability of the participant to disrupt their own data collection.

With these settings fixed for all conditions of our study, we now turn to those which we vary between conditions: the presence of both visual grid lines and a helpful “ghost zoid” to aid in lining up and dropping zoids in the desired location (see Fig. 3). This parameter would be set in the experimental parameters section of the configuration file:

```
#VARIED parameters -- GUIDED CONDITION
```

```
# Experiment and Condition ID
```

```
ECID = E1_GDD ## abbreviation of "guided"
```

```
ghost_zoid = true
```

```
gridlines_x = true
```

```
gridlines_y = true
```

```
#VARIED parameters -- NOT-GUIDED CONDITION
```

```
# Experiment and Condition ID
```

```
ECID = E1_notGDD ## abbreviation for "not guided"
```

```
# The next three settings are the default Meta-T settings for these
```

```
# parameters. Hence, they could be omitted. However, it is good practice
```

```
# to explicitly document the setting of key parameters in both conditions.
```

```
ghost_zoid = false
```

```
gridlines_x = false
```

```
gridlines_y = false
```

Fig. 3 Sample task visuals of the non-guided game environment (left) and the guided game environment with (right). The guided environment puts gridlines in the background and a faint “ghost” zoid indicating where that zoid would end up if dropped

Any line in the configuration file that is preceded by a hash mark (i.e., #) is a comment line. Hence, in the above example the “#FIXED parameters”, “#Presentation”, etc., lines are comments and will not be read by the Meta-T software. We encourage researchers to use comments throughout their configuration files.

For our two-condition example, after construction each configuration file has been named and saved to the “configs” directory within the Meta-T package’s file tree. We have chosen to name these configuration files: `e1-cond_notGuided.config` and `e1-cond_guided.config`. The ECID parameter explicitly states which experimental condition the current session is in for logging later.

For clarity, one configuration file would include both FIXED and VARIED parameters and look as follows:

```
#FIXED parameters

fullscreen = true

music_vol = 0.0

fixed_seeds = true

random_seeds = 1, 2, 3, 4, 5

permute_seeds = true

continues = 5

max_eps = 100
```

```
starting_level = 4

lines_per_lvl = 9000

pause_enabled = false

#VARIED parameters -- GUIDED CONDITION

ECID = E1_GDD

ghost_zoid = true

gridlines_x = true

gridlines_y = true
```

To begin data collection, we run the program from the command line. From the root directory of the software package, the following command runs the experiment with a desired configuration file:

```
python meta-t.py -c e1-cond_notGuided

or

python meta-t.py -c e1-cond_Guided
```

The “-c” or “configuration file” argument of this command indicates which file (sans the file extension of “.config”) the program will load from the “configs” directory. In the case of the example investigation, each condition of the experiment will be run with a different configuration file.

Logging data output

By default, data files for each player for each session can be retrieved from the package’s “data” directory (if another sub-directory is desired, it can be specified in the configuration file). In this directory, for each session Meta-T creates a subdirectory labeled: [subject_id]_yyyy-mm-dd_hh-mm-ss (e.g., S1_2014-03-27_15-36-22. (Where “S1” is specified at runtime and tells us that subject’s unique identifier as specified within the scope of the experiment. The runtime and the date and time elements are set based on the date and time the subject was run)).⁹

⁹Note that in response to IRB concerns for “non-person identifiable log files” there is nothing in the log files, other than the date and time, that can be used to specify the person’s identity. Keeping track of the same individual across multiple sessions is the responsibility of the experimenter. For this purpose, an optional repository “pycrypto” can enable the use of an encrypted version of a subject’s personal identification numbers, such as Social Security Number or student ID. See the documentation for usage instructions.

For a complete list of all output variables, see Appendix B, or the “README_data.rst” file in the root directory of the Meta-T package.

The system generates five files in the session directory by default, and, optionally, one subdirectory.

- */_config_<subject id and session run time>.tsv*—An automatically generated .config file that explicitly defines every parameter used in this session. This is excellent as a record of the session, as well as a way to quickly replicate it if needed. This file should be carefully examined during your pilot studies to make sure that the settings for the different conditions are exactly as you intend them to be.
- */_hist_<subject id and session run time>.tsv*—The history file for this player’s session includes a portable record of many of the session’s variables and subject information.
- And three log files containing gameplay and performance data:
 - */complete_<subject id and session run time>.tsv*—This is the complete log file. It contains all of the data collected during the session, including moment to moment game state and event information, summaries for episodes and games, and eyetracking data if available. A single 100-zoid game’s complete log file (about 2.5 min of play) is roughly 10 Mb of data.
 - */episodes_<subject id and session run time>.tsv* – This file is a subset of the “complete” file; that is, it contains just the episode- and game-level summaries. These files are much smaller than the complete files and, therefore are both handy and useful for examining performance at a higher level. If you do not wish to devote harddrive space to saving redundant data, the option for saving episode files can be disabled by setting the “ep_log” flag to false. A single 100-zoid game’s episode log file is roughly 200 Kb of data.
 - */games_<subject id and session run time>.tsv* – Similar to “episodes” files, “games” files contain only game summaries, for the coarsest level of analysis of performance. These files are extremely light weight and easy to inspect visually. They can be disabled by setting the “game_log” flag to false. A single 100-zoid game’s game log file is roughly 2 Kb of data.
- Optional subdirectory: */screenshots/* – If the “ep_screenshots” flag is enabled in the .config file, this directory will be created. It contains one image

of the game state for each episode completed by the subject.

Log files can be generated either in fixed-column format or keyword format, set using the “fixed_log” parameter in the .config file (fixed-column is enabled by default). Fixed-column format uses a fixed header length and logs only the relevant entries for each line of the log. The fixed-column format will yield larger files with more redundant data than the keyword format. However, the fixed-column files can be directly read into statistical packages such as R or SPSS™. For the keyword format, each data item is preceded by its keyword item in a tab-delimited text file. This organization trades data redundancy for value name redundancy, and requires pre-parsing before being loaded into a statistical package. However, it produces smaller files and may be easier to use by an analyst who intends to parse and manipulate the raw data prior to analysis.

File: games_<session directory name>.tsv: The game-level log file includes a handful of measurements for each game played during a session and also includes the high-level performance measures typically provided by the game to the player. For Meta-T, the available measurements for a given game are as follows:

Identifying information: Five pieces of identifying information: (a) the unique subject ID (as specified at runtime), (b) the experiment condition ID, (c) the Session (e.g., “2014-03-27_15-36-22”), (d) the Game Type (particular to, and as specified by, the config file), and (e) the game number.

Performance information: The three pieces of performance information provided to the player by the game: (a) final score, (b) highest level of difficulty achieved, and (c) total number of lines cleared.

Episodes: The number of episodes played (zoids placed).

Game duration: The duration of the game played, in seconds.

Complete: A True/False flag indicating whether the game was completed by the player, or if the game being logged was terminated early by the experimenter, allotted time expiring, or a system failure.

Zoid sequence: The exact sequence of zoids presented to the player in a list form.

These measures give a top-level assessment of the basic characteristics of the player’s performance for each game played and will often suffice for an initial analyses of player performance between groups. With respect to the example investigation, we are initially interested in examining a player’s average “score” value for each game played.

Though these high-level entries might suffice for many purposes (e.g., in the studies of Haier et al. (2009), Martin-Gutierrez et al. (2009), Sims and Mayer (2002), and Terlecki et al. (2008)), they do not suffice for understanding how a player's skill manifests and changes throughout the course of a game in response to the dynamics of the task environment. To better detail how a player is approaching the game, we can delve deeper into the episode-level log file.

File: episodes_<session directory name>.tsv: In the episode log file, each record (aka entry or line in the file) contains information about a particular episode of a game (a single zoid placement action). This level of analysis gives information about the individual decisions made by a player throughout the course of the game, with respect to the location in which a zoid was placed on a given pile, and a description of key features of the pile (see also, Lindstedt (2013) and Lindstedt and Gray (2013)).

Information available in an episode summary entry includes:

Identifying information: Similar to those in the game-level files, but also including unique episode numbers.

Game-relevant information: For each record, it provides the difficulty level, the score, and the total lines cleared.

Episode-relevant state information: The current and next zoid, and an exact representation of the pile and the zoid at the end of the episode.

Danger mode flag: Indicates the state in which a loss may be imminent, determined by whether or not the pile has climbed to within five rows of the top of the game space, and indicated to the player by a quickening of the tempo of the music (if enabled).

Features of game state: This is information that reduces the raw state of the pile and zoid position into simpler features, able to be examined as individual outputs.

The features derived from the game state are of particular interest when measuring player performance. Drawn from work on artificial controllers of Tetris in the AI literature (see Robertson (2012) for a nice, one page summary, of the history and current state of that work), these features transform the pile representation and the player's zoid placements into more conceptually meaningful measures of performance. Some of these measures are: (a) maximum height, the highest point in the pile, showing how close the player may be to a game-over; (b) pits, the number of unworkable holes in the pile, an indication of cumulative error commission; (c) wells, the number of narrow 1 block-wide holes in the pile, affording fewer good-fitting zoid placements as they deepen; (e) transitions, a measure of the smoothness of the pile (e.g., many pits or separations in the pile lead to more transitions, i.e., less smooth).

These measures take us much deeper than do the high level performance scores of total points, lines cleared, and number of episodes and they allow us to construct measures which might be meaningful to a variety of different research questions. Still, these measures only demonstrate a player's ultimate decisions from each episode. To dive deeper, we need to look at the comprehensive log file.

File: complete_<session directory name>.tsv: The complete log file captures everything from the previous two file types and more by continuously logging the complete game state at each frame of animation (i.e., game time) and every action as it occurs (i.e., real time). This level of analysis is not necessary for every investigation, but it is complete and capable of highlighting a player's millisecond-by-millisecond interactive performance. Different information is contained in each type of entry, so every line of the file contains an "event_type" field, allowing for easy filtering of the data. The types of entries are outlined below.

For an example log file, see Appendix C.

State information Information about the state of the game is logged as often as it is updated on the screen, at a rate equivalent to the specified frame rate (the "fps" parameter), which does not vary when the game speed increases in higher difficulty levels. This information includes a complete representation of the current state of the board, the exact position and orientation of the current zoid, and more covert elements, such as whether the player is currently performing a drop or whether the player is in "danger mode" (classically defined as when the maximum height is within five rows of the top, indicating a game over is near).

This information is exceptionally useful when using technologies such as eye-tracking or electroencephalography (EEG) that have a higher temporal resolution than the framerate of the game. Having a complete account of every game-state stimulus offered to the player during performance allows such technologies to account for more precise phenomena.

Event information Game events consist of any event that may occur asynchronously with the updating of the game state, logged in real time. These include: key-press events by the player; system-generated events, such as the zoid being released into the environment at the start of an episode or automatically dropping by gravity; and various other events, such as documenting the setting of task environment variables.

Summary entries Also logged are summary listings from the previous two levels of log files, in the form of game-summary and episode-summary entries for completeness.

Unique to the comprehensive log file are the sequences and durations of a player's immediate interactions. Using this information, we can access information about a player's behavior at the sensory-motor level, answering questions about manual dexterity and also exploring slips and immediate corrections.

Data output summary The Meta-T paradigm outputs multiple levels of log files for the researcher's convenience. Contained in these files are measures from top-level performance, to mid-level decision-making and strategy implementation, down to immediate sensory-motor performance. Using information from each of these levels of analysis, the researcher can compose a very robust account of each subject's behavior.

Results of example analysis

Combining Meta-T's high level of control over the task environment with its robust logging of behavioral data allows researchers to perform a variety of experimental procedures. To demonstrate how all of these elements come together, we next present a pseudo-experiment, using one subject, that varies the visual guides present during gameplay. Please note that the data shown were collected as part of a pilot study and, as such, are meant only for the purpose of demonstrating the mechanics of carrying out an experiment in the Meta-T paradigm.

Game-level results The highest level of analysis is whether our Ghost-Zoid visual-aid (see Fig. 3) has any effect on overall performance. Extracting performance data from our games.txt file, Fig. 4 shows overall game scores with and without the manipulation.

Episode-level results Drilling down a level, we examine two measures from the features available at the episode level: height (a measure of overall survivability and control of the game-space); and pits (a cumulative measure of unworkable but covered holes in the pile, commonly considered to be errors). Figure 5 shows the average value for heights and pits per episode of gameplay under each condition of the visual aid manipulation.

Interactive-level results Next we investigate the fine-grained temporal details of player performance, those involving the way the player rapidly interacts with the task environment. We examine three measures of performance at this level: (a) initial latency, or the time taken to make the first keypress response in an episode; (b) average latency, or the mean time taken between keypresses, a measure of dexterity; and (c) drop latency, the time taken to decide where the player finally desires to drop the zoid. Figure 6

Fig. 4 Full game score for games played with no visual aids and with the “grid lines” and “ghost zoid” visual enhancements present. Error bars are 95 % confidence intervals

shows these three measures per episode, with and without the visual aid manipulation.

Discussion

This section illustrated a few of the many possible detailed data analyses that Meta-T makes possible. What we have presented is very far from being comprehensive or being a representative sample. The combinatorics of analyses possible with Meta-T are far too many to innumerate. Rather, we hope this section has demonstrated the ease of manipulating Meta-T and a sample of the type of detailed performance data that Meta-T provides.

Additional features

Eye-tracking support

Meta-T supports use of SMI™ eye tracking equipment. This enables the study of players' eye movements while performing a time-pressured, interactive task. Eye gaze data can serve to illuminate the nature of the player's information gathering and processing strategies, and how they integrate with their more overt behavior (Holmqvist et al. 2011).

Fig. 5 Features of the pile indicating decision-making strategy measured per episode either without visual aids or with the “grid lines” and “ghost zoid” visual enhancements present. Overall pile height (all columns) measures left, cumulative errors (pits) right. *Error bars* are 95 % confidence intervals

To use this function, one must install the PyViewX python package (from <https://github.com/RyanHope/PyViewX>) and then simply provide Meta-T with the IP address of an SMI eyetracking system at run-time. For example, if the program were run on an SMI eyetracking system’s main computer, one could simply provide the program with the “localhost” IP address, 127.0.0.1:

```
python meta-t.py -e 127.0.0.1
```

This simple addition is automatically recognized by the Meta-T software, which will begin logging all eye-samples collected by the eyetracker in the comprehensive log file. Please note that this will substantially increase the log file’s size by a factor of the sampling rate of the eye tracking

system used. For this reason, it is advisable to plan for files an order of magnitude larger when using the eye-tracking features.

In addition to logging the user’s point-of-gaze, eye-tracking with Meta-T enables some interesting manipulations: an option in the current version is to place a mask over the “next zoid” area of the screen that is only removed when the player gazes at it. This manipulation, and others like it, can serve either pragmatic goals (i.e., restricting the usefulness of peripheral vision to be more certain when precisely the player is examining “next” box) or provide a set of truly alien game mechanics for even the most seasoned players. More eye-gaze contingent game features are in development.

Fig. 6 Temporal measures of interactive behavior measured per episode either without visual aids or with the “grid lines” and “ghost zoid” visual enhancements present. **a** Time until first response (initial

latency). **b** Mean time between keypresses (average latency). **c** Time until dropping zoid (drop latency). *Error bars* are 95% confidence intervals

Note that these real-time gaze-contingent manipulations depend on the communication latency between the eye-tracking and local machines over TCP-IP communication; as such, it is advisable to run experiments with gaze-contingent elements over a local network, if not on the very same computer as the eye-tracking hardware.

ACT-R support

Currently in active development is the ability for Meta-T to interact with ACT-R models (Anderson, 2007), through the use of local TCP connections. Using the JSON Network Interface package for ACT-R (Hope, Schoelles, & Gray, 2014), Meta-T can send visual information to the model for processing, and receive manual key-press information back, creating a complete interactive loop between the task environment and the model. This will enable ACT-R modelers interested in Meta-T to create models that play in the same task environment as their human counterparts, having available the full set of features for environment manipulation and robust logging described in this paper.

Playback

Also included in the package is a system for playing back games from the logfiles in real time. This system is located in the “playback” directory as “playback.py”. Upon running, the program will automatically open to the “data” directory and allow selection of a subject’s data folder. The system can play back both fixed-column and keyword format logs. During playback, the operator can manipulate the speed of playback with the “up” and “down” arrow keys, and can skip backward or ahead by one full episode at a time with the “left” and “right” arrow keys. Playback of eye-data is not yet implemented, but is in development.

Ongoing development

The Meta-T program is in active development. Improvements to the code’s efficiency, new manipulations and options, and support for additional types of hardware (such as EEG) will continue to be added. For this reason, it is advisable for the interested researcher to stay abreast of updates to the program by examining the change logs on the GitHub website for the Meta-T project (<https://github.com/CogWorks/Meta-T>).

Conclusions

Many studies have used Tetris for reasons as diverse as enhancing the spatial abilities of student engineers

(Martin-Gutierrez et al., 2009) or as a treatment for post-traumatic stress syndrome (Holmes et al., 2010). However, most studies have simply used the game *out of the box* and have only collected the high level performance measures available to game players.

Meta-T offers a suite of easily accessible parameters that range from minor tweaks in timing to radical revisions of the fundamentals of gameplay. The player data collected by Meta-T is detailed enough so that the *playback* of the log file will be indistinguishable from watching the player play in real time. This capacity means that most any aspect of player-game interaction is available for research, across multiple levels of analysis from whole-game to millisecond-level behavior.

Acknowledgments The work was supported, in part, by grant N000141310252 to Wayne Gray from the Office of Naval Research, Dr. Ray Perez, Project Officer.

Appendix A: Configurable parameters

The following are all of the currently implemented parameters specifiable in a .config file. The default assignment is shown, followed by the variable’s type, and description.

variable_name = default_value [type]

- description

e.g.:

SID = Test [string]

- sets the subject ID, as specified by the experimenter.

The subject ID (SID) is of type [string] and defaults to “Test”.

Variable types

The following are all possible types of values accepted by the parameter variables”

[integer]: requires whole numbers.

[integer list]: accept whole numbers separated by commas.

[color]: accepts precisely three whole numbers (red, green, blue values between 0 and 255), separated by commas.

[float]: can accept decimal points.

[string]: does not require quotation marks (”) around the item.

e.g., when specifying the logging directory, entering:

logdir = data

is preferable to

logdir = “data”

[Boolean]: accepts true, t, yes, or y as a True value (case ignored). All others are considered false.

e.g., when specifying full screen mode:

fullscreen = y (or) **fullscreen = True**

will enable full screen mode, while

fullscreen = no (or) **fullscreen = f** (or) **fullscreen = abcd** (or) **fullscreen = Tru**

will all evaluate to false, disabling full screen mode instead running the game in a window.

Some parameters allow only a small set of values, as specified in their descriptions.

Logging variables

logdir = data [string]

- specify the logging directory for new data files (within the root package).

fixed_log = True [Boolean]

- use fixed column width logging format. Disable for keyword-access format.

ep_log = True [Boolean]

- enable an additional reduced-form log storing only episode- and game-summaries.

game_log = True [Boolean]

- enable an additional reduced-form log storing only game-summaries.

ep_screenshots = False [Boolean]

- enable screenshots to be taken at the end of each episode.

Identifier variables

SID = Test [string]

- sets the subject ID, as specified by the experimenter.

RIN = 000000000 [string]

- sets an encrypt-able universal identifier (school ID or SSN).

ECID = NIL [string]

- sets the experimental condition ID, if known at runtime.

game_type = standard [string]

- sets the game type label, as specified by the experimenter.

Rules and scoring variables

game_ht = 20 [integer]

- set number of rows in game space.

game_wd = 10 [integer]

- set number of columns in game space.

lines_per_lvl = 10 [integer]

- set the number of lines to be cleared before difficulty increases.

scoring = 40,100,300,1200,6000 [integer list]

- set the default value (at level 0) of clearing 1, 2, 3, 4 (or 5 in Pentix) lines.

drop_bonus = True [Boolean]

- enable a small point bonus for dropping manually dropping the zoid.

starting_level = 0 [integer]

- set the game's starting level.

continues = 0 [integer]

- set number of games to be played.

- 0 allows for infinite games.

time_limit = 3600 [integer]

- set time limit for the session (across games) in seconds.

- defaults to one hour.

max_eps = -1 [integer]

- maximum episodes allowed per game.

- set to -1 for infinite episodes.

tetris_zoids = True [Boolean]

- enable the normal Tetris zoids (the 4-segment "IOTSZJL" set).

pentix_zoids = False [Boolean]

- enable a set of additional, larger zoids (5 segments each).

tiny_zoids = False [Boolean]

- enable a set of additional, smaller zoids (1, 2, and 3 segments).

Timing variables

fps = 30 [integer]

- set number of frames drawn per second.

- higher values are more processor intensive. lower values are more distracting.

tps = 60 [integer]

- set number of "ticks" per second, i.e., how often the game logic updates.

- higher values result in faster gameplay, regardless of the value of fps.

- defaults are advised.

gravity = True [Boolean]

- enables time pressured difficulty, i.e., whether or not the zoid drops automatically.

intervals = 48, 43, 38, 33, 28, 23, 18, 13, 8, 6, 5, 5, 5, 4, 4, 4, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1 [integer list]

- set the difficulty for each game level, i.e., speed at which the zoid naturally falls.

- measured in "ticks". If "tps" is set to 60, a value of 30 results in a zoid dropping two rows per second.

Game controls variables

pause_enabled = true [Boolean]

- enables pausing the game.

- default key is P.

two_player = False [Boolean]

- enables two-player mode, in which one player controls rotations while the other controls translations and dropping.

joystick_type = NES_RETRO-USB [string]

- set the type of joystick to be used. Supports classic Nintendo Entertainment System USB controllers.

- accepts only NES_RETRO-USB or NES_TOMEE-CONVERTED.

undo = False [Boolean]

- allows the player to undo all zoid movement this episode up to, but not including, a zoid placement.

- default key is R.

zoid_slam = False [Boolean]

- allows player to instantaneously “slam” the zoid down its current column into the pile.

- default key is Spacebar.

keep_zoid = False [Boolean]

- allows the player to “keep” a zoid to be switched out later for better placement.

- default key is E.

wall_kicking = False [Boolean]

- enables zoid rotations to “kick” off of walls or the pile.

- rotations that would normally fail due to collision are bumped left or right.

drop_interval = 2 [integer]

- set the speed at which the player may manually drop the zoid.

- default values result in a zoid being able to be dropped at 30 rows per second.

das_delay = 16 [integer]

- time in “ticks” the left or right key must be held before “delayed auto shift” begins

- default values result in waiting roughly 1/4 second.

das_repeat = 6 [integer]

- number of “ticks” between automatic column translations once “delayed auto shift” begins.

- default values result in 10 columns per second.

das_chargeable = true [Boolean]

- enable the player to hold the key before a zoid appears to prepare a “delayed auto shift” in advance.

das_reversible = true [Boolean]

- enable the player to move the zoid in the opposite direction, but preserve the “delayed auto shift” speed.

are_delay = 10 [integer]

- sets the delay before a new zoid enters play, measured in ticks.

- defaults to 1/6 of a second.

lc_delay = 20 [integer]

- sets the animation duration for line clears, adding to the are_delay.

- defaults to 1/3 or a second.

Randomization variables

fixed_seeds = false [Boolean]

- specify whether to use a fixed set of random seeds at all.

random_seeds = 1, 22, 3000000, 456789 [integer list]

- specify which, if any, random seeds to use. (Consistent across platforms.)

permute_seeds = true [Boolean]

- specify whether to randomize the order of the chosen random seeds.

seven_bag_switch = False [Boolean]

- enable the 7-bag zoid randomization mode.

- guarantees all 7 default zoid types will appear in each set of 7 zoids presented.

Look-ahead variables

look_ahead = 1 [integer]

- sets how many upcoming zoids are displayed.

- currently only accepts 0 and 1.

far_next = False [Boolean]

- if enabled, moves the Next Box (look_ahead) to the far right of the screen.

next_dim = False [Boolean]

- if enabled, dims the next box with transparency.

next_dim_alpha = 50 [integer]

- sets how dim the next box should be, if dimmed.

next_mask = False [Boolean]

- if enabled, places a mask over the next box that must be removed.

- default key for removal is Q.

board_mask = False [Boolean]

- if enabled, masks the board while the Q key is held, forcing players to focus on one or the other.

Audio variables

music_vol=0.5 [float]

- set the volume of the music.

- 0.0 is muted, 1.0 is full.

sfx_vol=1.0 [float]

- set the volume of the sound effects.

- 0.0 is muted, 1.0 is full.

song=gb-a [string]

- set the game’s song.

- choose from: gb-a, gb-b, nes-a, nes-b, nes-c.

Video and display variables

fullscreen = False [Boolean]

- set the game to fill up the display.

bg_color = 0,0,0 [color]

- set the background color of the game space.

border_color = 250,250,250 [color]

- set the border color of the game space.

kept_bgc = 50,50,50 [color]

- set the background color of the “kept zoid” area, if enabled.

gridlines_x = False [Boolean]

- enable perceptually helpful lines along the columns.

gridlines_y = False [Boolean]

- enable perceptually helpful lines along the rows.

gridlines_color = 50,50,50 [color]

- set color for any gridlines.

show_high_score = False [Boolean]

- display the highest game score from this session.

inverted = False [Boolean]

- invert the game space such that up and down are reversed (zoids fall up!).

visible_board = True [Boolean]

- disable to make the pile invisible.

visible_zoid = True [Boolean]

- disable to make the zoid invisible.

board_echo_placed = True [Boolean]

- enable to make the pile temporarily visible when a zoid is placed.

board_echo_lc = True [Boolean]

- enable to make the pile temporarily visible when a line is cleared.

dimtris = False [Boolean]

- enable to make the pile gradually disappear as the difficulty level rises.

dimtris_alphas = 255,225,200,175,150,125,100,75,50,25,0 [integer list]

- set the transparency values (0 - 255) for dimtris mode.

ghost_zoid = False [Boolean]

- enable a helpful ghost zoid.

- shows where the zoid would end up if dropped from the current column.

Artificial intelligence variables

controller = dellacherie [string]

- set the controller to be used for AI decisions.

- default controller attempts to optimize simple line-clearing behavior.

- select from the /controllers directory (omit .control).

solve_button = false [Boolean]

- enables a keypress to automatically “solve” the current zoid according to the AI controller chosen.

- solve key is N, press M to enable auto-solve mode.

auto_solve = false [Boolean]

- enables a self-playing, automatic solving game mode, determined by the AI controller chosen.

hint_zoid = false [Boolean]

- displays a dim zoid, similar to the ghost_zoid, at the location chosen by the AI controller.

hint_button = false [Boolean]

- enables a hint to be displayed only if desired by keypress. Designed for training.

- hint key is H.

hint_release = True [Boolean]

- enables the hint to be seen only while the hint key is held.

hint_limit = -1 [integer]

- sets a limit for the number of hints allowed per game.

- set to -1 for unlimited hints.

Classic experiment variables

n_back = False [Boolean]

- enables a mode where zoids are only placed if they match the zoid placed n-many back.

nback_n = 2 [integer]

- sets the n for n_back mode.

ax_cpt = False [Boolean]

- enables a mode wherein if a target zoid follows a cue zoid, it is not placed and instead vanishes.

ax_cue = O [string]

- sets the cue zoid for ax_cpt mode.

- choose from I, O, T, S, Z, J, and L.

ax_target = I [string]

- sets the target zoid for ax_cpt mode.

- choose from I, O, T, S, Z, J, and L.

Eyetracking variables

All variables here require eye-tracking hardware enabled to be meaningful.

draw_samps = False [Boolean]

- enable to display the raw eye tracking samples on-screen in real time.

draw_avg = False [Boolean]

- enable to display the average center point of the raw eye tracking samples on-screen in real time.

draw_fixation = False [Boolean]

- enable to display fixations calculated on-the-fly (inaccurate) on the screen in real time.

draw_err = False [Boolean]

- enable to display average center point of raw eye tracking samples with width and height deviation on-screen in real time.

gaze_window = 30 [integer]

- set the number of samples over which averages are calculated.

spotlight = False [Boolean]

- enable a “spotlight” that occludes all of the screen but an area around where the player is gazing.

spot_radius = 350 [integer]

- set the radius of the “spotlight” in pixels.

spot_color = 50,50,50

- set the color of the spotlight.

spot_alpha = 255 [integer]

- set the transparency of the spotlight.

- between 0 and 255, 255 being opaque.

spot_gradient = True [Boolean]

- if disabled, the spotlight area is fully transparent; if enabled, spotlight fades out to opacity.

eye_mask = False [Boolean]

- enables eye contingency for the Next Box mask (if enabled).

- if enabled, the mask over the Next Box will only disappear when gazed at.

Appendix B: Output variables

The following is a list of all the variables logged by the Meta-T program during gameplay.

General outputs

ts - time stamp in seconds since the beginning of the session

event_type - designates the type of entry this row represents in the log

SID - the Subject ID

ECID - the Experimental Condition ID

session - the date and time of the start of the session

game_type - the game type label

game_number - the number of the current game this session

episode_number - the number of the current episode this game

level - the current level this game

score - the current score this game

lines_cleared - the number of lines cleared this game

tetrises_game - number of tetrises scored this game

tetrises_level - number of tetrises scored this level

Game summary outputs

completed - whether or not a game was completed or terminated early (manually or by failure)

game_duration - length in seconds of the game

avg_ep_duration - length of the average episode this game

zoid_sequence - the exact sequence of zoids seen this game (of the types I, O, T, S, Z, J, and L)

Immediate event outputs

evt_id - the ID of the current event. This is highly flexible and depends on the event.

evt_data1 - the first value of the current event. depends heavily on evt_id.

evt_data2 - the second value of the current event. depends heavily on evt_id.

Game state outputs

curr_zoid - the current zoid this episode (of I, O, T, S, Z, J, and L)

next_zoid - the upcoming zoid for the next episode (of I, O, T, S, Z, J, and L)

danger_mode - whether the maximum height of any one column is greater than 15 (game music speeds up!)

delaying - whether the game is currently delaying, waiting for a new zoid to appear

dropping - whether the player is currently dropping the zoid

zoid_rot - current rotation of the zoid (between 0 and 3 for T, J, and L; 0 and 1 for I, S, and Z; 0 for O)

zoid_col - current column of the upperleft-most square of the zoid's bounding box

zoid_row - current row of the upperleft-most square of the zoid's bounding box

board_rep - exact visual representation of the pile (without the current zoid)

zoid_rep - exact visual representation of the zoid's current position.

Episode summary outputs

evt_sequence - sequences of keypresses and events taken this episode.

rots - number of rotations performed

trans - number of translation (left or right) performed

path_length - total number of rotations and translations

min_rots - minimum possible rotations to reach destination

min_trans - minimum possible translations to reach destination

min_path - minimum possible total path to reach destination

min_rots_diff - number of unnecessary rotations

min_trans_diff - number of unnecessary translations

min_path_diff - total number of unnecessary rotations and translations

u_drops - number of rows the zoid traversed due to player dropping

s_drops - number of rows traversed by the zoid due to system gravity

prop_u_drops - proportion of user-initiated row traversals to total row traversals

initial_lat - latency in milliseconds of the first keypress in an episode

drop_lat - latency in milliseconds until zoid is dropped in an episode

avg_lat - average latency between keypresses this episode

Zoid and pile feature outputs

These are numerical features of the state of the pile and each zoid's placement. Many of these features are only subtly different from one another, but some argue that these subtle differences could differentiate skilled performance. All of these features have been used in Artificial Intelligence work to train artificial controllers to maximize different decisions.

mean_ht - mean height of all columns
 max_ht - highest height of any one column
 min_ht - height of the shortest column
 all_ht - sum of the height of all columns.
 max_ht_diff - difference between max_ht and mean_ht
 min_ht_diff - difference between mean_ht and min_ht
 column_9 - height of the 9th column
 cleared - number of lines cleared by this zoid placement
 cuml_cleared - cumulatively weighted number of lines cleared (1=1, 2=1+2, 3=1+2+3...)
 tetris - whether or not a tetris was scored by this zoid placement
 move_score - points scored by this zoid placement (needs line clear)
 col_trans - total number of transitions from filled to empty (or reverse) for all columns
 row_trans - total number of transitions from filled to empty (or reverse) for all rows
 all_trans - sum of the transitions from filled to empty (or reverse) for all rows and columns
 wells - number of wells, divots with at least one filled cell on either side.
 cuml_wells - cumulatively weighted depth of all wells in the pile (1=1, 2=1+2, 3=1+2+3...)
 deep_wells - number of wells of depth 2 or greater
 max_well - deepest depth of any one well
 pits - number of unworkable holes (empty, but vertically obstructed) in the pile
 pit_rows - number of rows containing pits (as they will be uncovered simultaneously)
 lumped_pits - number of pits wherein contiguous pits count only once
 pit_depth - sum of all pits, weighted by their depths beneath the surface
 mean_pit_depth - mean depth of all pits (pits buried deeper count more)
 cd_1 - difference in height between columns 0 and 1
 cd_2 - difference in height between columns 1 and 2
 cd_3 - difference in height between columns 2 and 3
 cd_4 - difference in height between columns 3 and 4
 cd_5 - difference in height between columns 4 and 5

cd_6 - difference in height between columns 5 and 6
 cd_7 - difference in height between columns 6 and 7
 cd_8 - difference in height between columns 7 and 8
 cd_9 - difference in height between columns 8 and 9
 all_diffs - sum of the differences in height between all columns.
 max_diffs - highest height difference between two columns
 jaggedness - length of the perimeter of the pile (higher is more jagged)
 d_mean_ht - change in mean_ht from before this zoid placement to after
 d_max_ht - change in max_ht from before this zoid placement to after
 d_all_ht - change in all_ht from before this zoid placement to after
 d_pits - change in pits from before this zoid placement to after
 landing_height - row at which this zoid was placed
 pattern_div - number of unique transitions between two columns. measures diversity of pattern.
 matches - number of segments about the perimeter of the zoid matching its placement in the pile
 tetris_progress - number of contiguous, uncovered rows with exactly 9 cells filled.
 nine_filled - number of rows with exactly 9 filled columns
 full_cells - number of filled cells on the board
 weighted_cells - sum of all filled cells, weighted by their height
 eroded_cells - number of segments of the current zoid eroded (cleared) by this zoid placement
 cuml_eroded - cumulatively weighted number of cells eroded (1=1, 2=1+2, 3=1+2+3...)

Eye tracking outputs

smi_ts - timestamp reported by the eyetracker
 smi_eyes - which eyes are visible ("b" for both, "l" or "r" for left or right)
 smi_samp_x_l - x gaze position (left to right), left eye
 smi_samp_x_r - x gaze position (left to right), right eye
 smi_samp_y_l - y gaze position (top down), left eye
 smi_samp_y_r - y gaze position (top down), right eye
 smi_diam_x_l - pupil diameter x (left to right), left eye
 smi_diam_x_r - pupil diameter x (left to right), right eye
 smi_diam_y_l - pupil diameter y (top down), left eye
 smi_diam_y_r - pupil diameter y (top down), right eye
 smi_eye_x_l - 3d eye position x (left to right), left eye
 smi_eye_x_r - 3d eye position x (left to right), right eye
 smi_eye_y_l - 3d eye position y (top down), left eye
 smi_eye_y_r - 3d eye position y (top down), right eye
 smi_eye_z_l - 3d eye position z (depth), left eye

smi_eye_z_r - 3d eye position z (depth), right eye
 fix_x - on-the-fly calculation of fixation position x
 (inaccurate)
 fix_y - on-the-fly calculation of fixation position y
 (inaccurate)

Appendix C: Example log file

The following is an example log file. Due to the sensitivity of recordings offered by the program, just 9 rows of the log

are shown, representing the dropping and placement of an “S”-shaped zoid and the termination of the episode, occurring between 18.36 seconds and 18.73 seconds (less than half a second!). In this time, several game states are logged (event_type GAME_STATE), as well as many game events (GAME_EVENT), and an episode summary (EP_SUMM).

Because the software is still in active development, some of these variables and formats may change, and future variables may not yet be represented here.

Row#	ts	event_type	SID	ECID	session	game_type
1	18.36731600761	GAME_STATE	Test	NIL	2014-5-26.15-35-51	“standard”
2	18.39723515511	GAME_EVENT	Test	NIL	2014-5-26.15-35-51	“standard”
3	18.53024315834	GAME_STATE	Test	NIL	2014-5-26.15-35-51	“standard”
4	18.56442809105	GAME_EVENT	Test	NIL	2014-5-26.15-35-51	“standard”
5	18.70002508163	GAME_STATE	Test	NIL	2014-5-26.15-35-51	“standard”
6	18.73132514954	GAME_EVENT	Test	NIL	2014-5-26.15-35-51	“standard”
7	18.73168110847	GAME_EVENT	Test	NIL	2014-5-26.15-35-51	“standard”
8	18.73291921616	EP_SUMM	Test	NIL	2014-5-26.15-35-51	“standard”
9	18.73320102692	GAME_EVENT	Test	NIL	2014-5-26.15-35-51	“standard”

Row#	game_number	episode_number	level	score	lines_cleared
1	1	0	0	0	0
2	1	0	0	0	0
3	1	0	0	0	0
4	1	0	0	0	0
5	1	0	0	0	0
6	1	0	0	0	0
7	1	0	0	18	0
8	1	0	0	18	0
9	1	0	0	18	0

Row#	evt_id	evt_data1	evt_data2	curr_zoid	next_zoid	danger_mode
1				S	S	FALSE
2	ZOID	U-DOWN		S	S	FALSE
3				S	S	FALSE
4	ZOID	U-DOWN		S	S	FALSE
5				S	S	FALSE
6	ZOID	U-DOWN		S	S	FALSE
7	PLACED	S	[1, 7, 3]	S	S	FALSE
8				S	S	FALSE
9	EPISODE	END		S	S	FALSE

References

- Anderson, J.A. (2007). *How can the human mind occur in the physical universe?* Oxford series on cognitive models and architectures. New York: Oxford University Press.
- Baccherini, D., & Donatella, M. (2008). Combinatorial analysis of Tetris-like games. In *Discrete Mathematics*. issn: 0012-365X. doi:[10.1016/j.disc.2007.08.009](https://doi.org/10.1016/j.disc.2007.08.009) (Vol. 308.18 pp. 4165–4176).
- Lindstedt, D.M.J.K., & Gray, W.D. (2011). Use of complementary actions decreases with expertise. In Carlson, L., Hölscher, C., & Shipley, T. (Eds.) *Proceedings of the 33rd Annual Conference of the Cognitive Science Society* (pp. 2709–2014): Austin, TX: Cognitive Science Society.
- Ehret, B.D., Gray, W.D., & Kirschenbaum, S.S. (2000). Contending with complexity: Developing and using a scaled world in applied cognitive research. In *Human Factors*, (Vol. 42.1 pp. 8–23.)
- Fahey, C.P. (2013). Tetris AI. <http://www.colinfahey.com/tetris/>
- Gray, W.D., John, B.E., & Atwood, M.E. (1993). Project Ernestine: Validating a GOMS analysis for predicting and explaining real-world performance. In *Human-Computer Interaction*, (Vol. 8.3 pp. 237–309).
- Haier, R.J. et al. (1992). Regional glucose metabolic changes after learning a complex visuospatial/motor task: A positron emission tomographic study. In *Brain Research*, (Vol. 570 pp. 134–143).
- Haier, R.J. et al. (2009). MRI assessment of cortical thickness and functional activity changes in adolescent girls following three months of practice on a visual-spatial task. *BMC Research Notes*, 2(1), 1–7. doi:[10.1186/1756-0500-2-174](https://doi.org/10.1186/1756-0500-2-174) English.
- Hays, R.T. et al. (1992). Flight simulator training effectiveness: A meta-analysis. In *Military Psychology*. eprint: http://www.tandfonline.com/doi/pdf/10.1207/s15327876mp0402_1 http://www.tandfonline.com/doi/abs/10.1207/s15327876mp0402_1 (Vol. 4.2 pp. 63–74).
- Holmes, E.A. et al. (2010). Key Steps in Developing a Cognitive Vaccine against Traumatic Flashbacks: Visuospatial Tetris versus Verbal Pub Quiz. In *PLOS ONE*, 5.11. issn: 1932-6203 doi:[10.1371/journal.pone.0013706](https://doi.org/10.1371/journal.pone.0013706)
- Holmqvist, K. et al. (2011). *Eye tracking: A comprehensive guide to methods and measures*. New York: Oxford University Press. isbn: 978.0.19.969708.3.
- Hope, R.M., Schoelles, M.J., & Gray, W.D. (2014). Simplifying the interaction between cognitive models and task environments with the JSON Network Interface. *Behavior Research Methods*, 46, 1007–1012. doi:[10.3758/s13428-013-0425-z](https://doi.org/10.3758/s13428-013-0425-z)
- Keehner, M. (2011). Spatial cognition through the keyhole: How studying a real-world domain can inform basic science-and vice versa. In *Topics in Cognitive Science*, 3.4, 632–647. issn: 1756-8757. doi:[10.1111/j.1756-8756.2011.01154.x](https://doi.org/10.1111/j.1756-8756.2011.01154.x)
- Keehner, M. et al. (2004). Spatial ability, experience, and skill in laparoscopic surgery. In *American Journal of Surgery*, 188.1, 71–75. issn: 0002-9610. doi:[10.1016/j.amjsurg.2003.12.059](https://doi.org/10.1016/j.amjsurg.2003.12.059)
- Kendall, G., Parkes, A., & Spoerer, K. (2008). A survey of NP-complete puzzles. In *ICGA JOURNAL*, 31.1, 13–34. issn: 1389-6911.
- Kirsh, D., & Maglio, P. (1994). On distinguishing epistemic from pragmatic action. In *Cognitive Science*, 18, 513–549.
- Lindstedt, J.K. (2013). Identifying expertise: Data exploration in Tetris. MA thesis. Rensselaer Polytechnic Institute.
- Lindstedt, J.K., & Gray, W.D. (2013). Extreme expertise: Exploring expert behavior in Tetris. In Knauff, M. et al. (Eds.) *Proceedings of the 35th Annual Meeting of the Cognitive Science Society* (pp. 912–917): Austin, TX: Cognitive Science Society.
- Maglio, P., & Kirsh, D. (1996). Epistemic action increases with skill. In *18th Annual Meeting of the Cognitive Science Society* (pp. 391–396): Cognitive Science Society.
- Martin-Gutierrez, J. et al. (2009). Do video games improve spatial abilities of engineering students? In *International Journal of Engineering Education*, 25.6, SI, 1194–1204. issn: 0949-149X.
- Okagaki, L., & Frensch, P.A. (1994). Effects of video game playing on measures of spatial performance: Gender effects in late adolescence. In *Journal of Applied Developmental Psychology*, 15.1, 33–58. issn: 0193-3973.
- Proctor, M.D., Bauer, M., & Lucario, T. (2007). Helicopter flight training through serious aviation gaming. In *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 4.3, 277–294. doi:[10.1177/154851290700400305](https://doi.org/10.1177/154851290700400305) eprint: <http://dms.sagepub.com/content/4/3/277.full.pdf+html> <http://dms.sagepub.com/content/4/3/277.abstract>
- Robertson, J. (2012). A brief history of Tetris AI. <http://liquidnarrative.csc.ncsu.edu/index.php/the-news/the-ln-blog/145-a-brief-history>. (visited on 11/27/2013).
- Shepard, R.N., & Metzler, J. (1971). Mental rotation of 3-dimensional objects. In *Science*, 171.3972, 701–703. issn: 0036-8075. doi:[10.1126/science.171.3972.701](https://doi.org/10.1126/science.171.3972.701)
- Sims, V.K., & Mayer, R.E. (2002). Domain specificity of spatial expertise: The case of video game players. In *Applied Cognitive Psychology*, 16, 97–115.
- Terlecki, M.S., Newcombe, N.S., & Little, M. (2008). Durable and generalized effects of spatial experience on mental rotation: Gender differences in growth patterns. In *Applied Cognitive Psychology*, 22.7, 996–1013.