Standard Corsi Test – Java Swing GUI Application Implementation

Tyler Schmidt

tschmidt@uccs.edu

Abstract

The Standard Corsi Test is a psychological test that tests short-term memory of people who play it. The game arranges 9 blocks in random locations on a screen. A pattern of blocks is then lit up, in a specific random order. The participant is then asked to repeat this pattern in either the same or reverse order. This paper explores an implementation of this test using the Java’s Swing GUI component system. This paper takes a unique approach of investigating the correlations of a participant’s usage of the software and their Corsi score.

Introduction*[[1]](#footnote-1)*

A common visuo-spatial short term memory psychological test is the Standard Corsi test which presents the participant with nine blocks in random location on a screen (or other medium). A pattern of blocks is then “lit up” to the participant, asking the participant to repeat this sequence in either the same, or reversed order. For each successful pattern recognized, the amount of blocks that are “lit up” is increased by one. Studies have shown that this test can help determine the onset and / or presence of certain diseases and disorders such as Alzheimer’s and ADHD and can also help determine the severity. [4][7] This test in no way is an inclusive diagnosis and should not be regarded as the only fabric of evidence in which to base a true diagnosis on. That being said, this test is a useful tool in the determination of one’s mental health.

The intention of this particular project is to test whether a participant’s usage of this software has a correlation with their Corsi score. In other words, will repetitive, prolonged usage of this software have a positive or negative impact on the Corsi score that the individual can achieve?

Hypothesis*[[2]](#footnote-2)*

Based upon the research and the studies that were evaluated during the course of this project, it is the general hypothesis that no significant correlation will be found, at least not in teenaged or adult participants, when comparing the usage of this software and an individual’s Corsi Score. Based upon the work of Farrell Pagulayan that an individual’s visuo-spatial short-term memory plateaus in late middle school. [8]. On the other hand, participants under middle school age may show a correlation in the increase of their Corsi score, and the usage of this software. However, if the data shows that a pattern exists when comparing the usage of this software and an individual’s Corsi score among elementary age children we cannot run under the assumption that the two factors are indeed correlated. Additional tests may be required in determining whether an increase in Corsi score can be associated with the usage of this software or if other factors, such as increase in age, are affecting the results as well.

Software Language

Java was chosen as the language to create this project in for a variety of reasons, but mostly for its built-in GUI elements under the Abstract Window Toolkit API. Java also features dependencies that can be easily added to any project to expand the capabilities of the software language.

Java has the ability to bind methods using both dynamic and static binding, which was very beneficial in the implementation of this software. Most methods use static binding, which is faster due to the fact the methods are binded at compile time. There are a few places, however, that the game has to override a specific method, and those methods are dynamically bound due to their function overriding nature.

Java Swing

Swing is a GUI toolkit that is part of the Java Foundation Classes (JFC) API. It provides “lightweight” GUI classes and components that make it possible to more easily develop GUI applications using only the Java software language.

Simple JSON

JSON, or JavaScript Object Notation is a data-interchange format that is popular for saving and organizing data into a easy to read object, record, struct, hash table, etc. depending on what software language it is being used with. JSON is compatible with many languages out-of-the-box without the use of external libraries or dependencies, and compatible with many other languages through the use of such external libraries/dependencies. Java, is on the latter side of this, requiring some extra configuration in order to use JSON objects natively in the language.

JSON objects and arrays are at core to how this application handles the saving and loading of various players of the game, however, since JSON is not by default supported in the Java language, a library called Simple JSON was added into the project to expand the capabilities of the Java software languages to support JSON arrays and objects.

Software Features

JFrame Menu System

The game starts by making a JFrame component, which is the GUI window that acts as the main window of the game. The initial settings are set for the window, JPanels are then added as the various game “screens”. [2] When a new screen is to be shown, the software removes the current JPanel component that is currently displaying, and adds the needed component in order to display the new screen. There are functions in the Main class to change the JPanel that is being displayed for every screen in the entire game.

Main Game Loop

The main game loop of the program is structured as a JPanel component that creates JButtons from coordinates that are given to it from the map class. The program enters a loop in which it reads in the index, X-coordinate, and Y-coordinate of each of the nine buttons that are required to display onto the screen.

After the block are successfully displaying on the monitor, the software enters another loop in which the “computer player” presses these generated blocks in a random order, according to the player’s level. This works by “pressing” the blocks the amount of the player’s current level added to one (i.e. if the player’s current level is 3, then 4 blocks must be pressed). When choosing what button to press, the software chooses a random integer between 1 and 9, which are the various indexes of the buttons that are being displayed. It then “presses” that button for a specified amount of time, releases the button, for a specified amount of time, and then repeats this process for however many buttons need to be pressed.

When the final button has been released for that specific round, a “Done” button appears at the bottom of the screen indicating to the player that the pattern has finished and the game is ready for the player’s input. The game then reads in the player’s input and when the “Done” button is pressed by the player, the game compares the buttons pressed (and their order) by the player to that of the original pattern to determine if the player has won the round. This is discussed in further detail in the Standard Corsi Forward and Standard Corsi Reversed subsections.

Random Map Generation

The standard Corsi test works by placing nine blocks in random places on the screen for the player to test. The way that the software accomplishes this is by creating an integer array that stores the coordinates and the indexes of each of the “JButtons” that are to be displayed on the screen.

The algorithm creates the X and Y coordinates of the first button and assigns it an index of 1. Afterward, the algorithm creates X and Y coordinates for the next button and compares the coordinates to that of every other coordinate that is in the array. If the coordinates are too close together, the program will attempt to “re-roll” the random number generator that creates the coordinates, these new coordinates are then compared the array of coordinates again the process repeats up to 100,000 times or until the coordinates are not too close together. The entire process is then repeated until all 9 buttons have coordinates assigned to them. The “Map” class then returns an array with 27 integers: an index, an X-coordinate, and a Y-coordinate for each of the nine buttons.

Saving & Loading

In order to allow multiple players to use the software, the application must have a way to store all the player’s information and be able to recall the information easily whenever it needs to be accessed (i.e. for loading and high scores). The application accomplishes this by saving a JSON Array with JSON objects that contain the information of a given player within the object. [5][6]

When the application starts, the game searches for an existing save file. If a save file is not found, then the game creates a new save file and also creates an empty JSON Array to store the various players that are saved in the save file. If a save file is found, the game searches the file and if any data is on the file, it reads it in storing the information from the file as a JSON Array

When a new player is created, the game asks for some basic information from the player and saves all this information into private variables in the Global class. When the player hits the “Game Over” screen of the game, these private variables are then all put into a new JSON Object, and the object is appended to the end of the JSON Array. The whole array is then saved onto the save file, replacing any existing data that may have previously been present.

When the game “loads” a player, the data from that specific JSON object within the JSON Array is loaded into the private fields in the Global class. When the players reaches the “Game Over” screen, any and all variables that have changed in value since the last time this player has played the game are replaced within that JSON object. The JSON Array that contains all of the various players is then written to the file, replacing any data that was previously written to the save file.

For every round that is played, the game generates an “Interaction Array” tied to each player that is saved within the save file. This Interaction Array gathers all of the needed information about the round/ game in order to create a full simulation of the interactions between the player and the software. The interactions array saves multiple integer and double values separated by negative numbers to distinguish the end of a specific variable. Negative numbers were chosen to be the separators because none of the variables to be appended to the array will ever have a negative value. The interactions array appends data to the array in the following order: Map, Level, Times Won, Current Health, Current speed, game mode, game reversed, new game, computer button pressed array, player button pressed array, win/lose flag, round time, and total game time.

Each round all of the above variables are appended to the “Interactions Array” with the integer “-1” appended between each variable to signify the start of a new variable. After the data for that specific round has all been appended to the interactions array, a “-2” is appended to the array signifying the end of a round. Finally, when the player loses the game and gets a “game over”, a “-3” in appended to the data to signify the end of that game session. The player can then play through the game if they so choose and the Interactions array will continue its cycle of appending the player’s now saved interactions array.

Standard Corsi Forward

The application is able to play the Standard Corsi test, both forward and in reverse. In forward mode, players are tasked to press the blocks in the exact order that the software presses them in. The game then needs to compare the “Computer’s” pressed button sequence and the player’s pressed button sequence and determine if the two sequences match.

The way that the game accomplishes this is actually quite simple, every button that appears on the game screen is assigned a 1-9 button *index* that gets put into an array whenever the button is pressed. There are two arrays, one for the “computer player” and one for the “human player”. Once the “Done” button has been pressed, the game then compares these two arrays to determine if they match or not, if the two arrays are the same, the players wins that round.

Standard Corsi Reversed

The game also features the ability to play the standard Corsi test in reverse, which is implemented in the software in a similar, but slightly more complex, way than the forward method. The game still has two arrays that it compares against each other to check if they are matching, but the difference is the array in which the game is checking.

There exists a flag in the software that is activated if the game’s reverse button is chosen. After the player has pressed the “Done” button, effectively ending the player’s turn, the game takes the player’s button index array and reverses it up until the button pressed index, which is a count of how many buttons were pressed by the player in that particular round. The reversed array is then compared the “computer player’s” array. If the two arrays are matching, the player has won that round.

“Classic”, “Pause” & “Modern” Modes

To enable the ability to modernize some aspects of the game without losing key aspects of the original implementation of this software, three separate “game modes” have been included in this software to give the user the ability to choose between the original implementation of the software or “classic mode”, or adding some of the more modern touches that have been added to the game.

“Classic Mode” is the standard version of the Corsi test. The player has two “lives” and can input an incorrect pattern one time and continue playing without immediately hitting a “game over”. The second time that the player inputs an incorrect pattern, the player is presented with the “game over” screen. Each pattern that is successfully inputted will increase the amount to “press” in the pattern by one. And when the player wins a round, they are presented with a “pause screen” that will automatically start the next round in three seconds.

“Pause Mode” acts in a similar fashion to “Classic Mode” with the difference being the screen that the player is presented with after completion of a round. In “Pause Mode”, the “pause screen” will not automatically continue to the next round in a specified amount of time, instead it continues to the next round only after the player has pressed the “Submit” button, allowing for the player to have time for breaks.

“Modern Mode” uses the “pause screen” that “Pause mode” uses, allowing player’s to have time for breaks but also introduces a new gameplay element. Instead of each successful pattern increasing the amount of blocks to “press” each round, this mode increases the amount every other round (even rounds - 2, 4, 6, etc.). On rounds that the block to press amount does not goes up, the speed in which the pattern is shown to the player is sped up. (i.e. less of a delay between individual block presses in the pattern, and the block is “pressed” for a shorter duration of time.

High Scores Screen

One important aspect of any video game software is the scoring system. Points and scores are what drive players to strive for their very best. This game is no different, scores are calculated by how many rounds the player can successfully complete before losing, also known as the player’s “Corsi score”. The highest score that a player has achieved in their attempts at the game is all saved into the game’s save file.

When the game is started, the software looks through all of the saves (if any) that are saved in the save file to calculate the 5 highest scores in the file. More specifically, this works by loading all of these entries into a JSON Array, as stated above. The game then enters a loop that looks for the highest score in the array, and when it finds it, the game records the score and index in which the score is at. It then does this process 4 more times, ensuring to skip over the indexes of the highest scores that already known. This process returns an array of the 5 (if less than 5 than the amount of saves in the array) highest scores in the save file, along with the usernames associated to them.

Memory Management

One very important advantage to using Java as the language for this project is Java’s use of memory management. Java will automatically deallocate the memory that is being used for a variable if that variable is assigned to something else and that memory isn’t being used elsewhere using its built-in garbage collection. The software takes this “garbage collection” to its advantage deallocating objects that are no longer in use by reassigning variables to either null or to a new instance of that particular class.

The game creates a new menu object for screens that can be displayed in the game upon startup. When the main game loop is active, the game must create new map objects and new “Game” objects in order to display the game blocks in a new order each round. When this happens, the game creates a new instance of these objects and assigns it the variable that holds the current instance of the object, successfully deallocating the object that was assigned the variable before. When changing to a menu, the game reassigns the menu object with a new instance of whatever type of menu that needs to be created, deallocating the memory used by the now unneeded previous menu.

Help Menu

In order to prevent confusion, the game has had a “Help Menu” added to the main menu of the game. The purpose if the help menu is to explain how the game is to be played and the various “Modes” that are included in the game.

Diagnosis

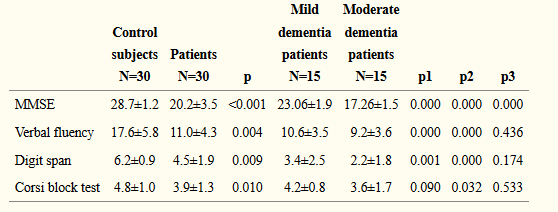
AD / Dementia

Table from *Spatial Working Memory in Alzheimer's Disease: A Study Using the Corsi Block-Tapping Test*, shows mean scores and standard deviation of control subjects and Alzheimer’s patients whom tested on the Corsi Block test.

Based upon research in the article *Spatial Working Memory in Alzheimer's Disease: A Study Using the Corsi Block-Tapping Test* an average adult with correctly functioning brain activity will typically get scores on the average 5-7 repeating blocks before they lose the game. [4] A person may be suffering from acute to moderate dementia if they are unable to perform to the par standard in the Corsi test. Again, this is not meant to be an inclusive diagnosis, this study simply shows that Alzheimer’s patients with acute to moderate forms of dementia score lower on the standard Corsi test than control subjects, suggesting that Alzheimer’s & dementia effect an individual’s visuo-spatial short term memory. The study showed that patients with acute cases of dementia scored on average 4.2 blocks (rounding to nearest integer would be 4). [4] Compared to the control subjects, whom had an average Corsi score of 4.8 (rounding the nearest integer is 5). And finally, the study found patients with a moderate case of dementia came in with an average of 3.6 (which rounded to the nearest integer is 4 again.) Because the study did not find a significant difference in the averages between mild / moderate cases of dementia, the game makes no attempt to distinguish between the two. However, since a more significant change exists among the average between control subjects and patients suffering from Alzheimer’s, with control subjects coming in at an average of 5 blocks and Alzheimer’s patients coming in at an average of 4 blocks when comparing just the rounded integer values of the averages. Due to this difference in averages, the game examines the Corsi score of the player at the time of game over and attempts to evaluate their “diagnosis” based on the Corsi score that they were able to achieve. The game diagnoses players with Corsi scores of 5 or higher as “Likely not AD”, signifying the player likely is not suffering from Alzheimer’s disease. It then diagnoses players with Corsi scores with lower than 5 blocks as “Possibly AD”, signifying that the player could potentially be showing symptoms of Alzheimer’s disease, but would need further medical examination in order to properly diagnosis the individual.

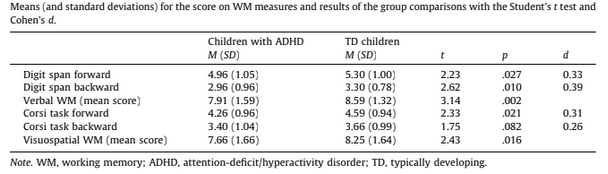
Attention Deficit Hyperactivity Disorder

Table from *Difficulties of Children with Symptoms of Attention-Deficit/Hyperactivity Disorder in Processing Temporal Information Concerning Life Events*, shows mean scores and standard deviations of children with ADHD and children whom are TD (typically developing).

Based upon research in the article *Difficulties of Children with Symptoms of Attention-Deficit/Hyperactivity Disorder in Processing Temporal Information Concerning Life Events*, shows that children whom are subjected to symptoms of Attention Deficit Hyperactivity Disorder and children whom were showing stages of typical development did not have significant differences in Corsi scores. Children with signs of ADHD scored on average that 4.26 as opposed the 4.59 average that typically developing children exhibited [7]. Due to this insignificant change in the average between the two Corsi scores, the game makes no attempt to diagnose signs of the presence of ADHD in children. The research that was collected in the investigation of this project did not state anything about the Corsi block test and the presence of ADHD in adult subjects, so the game makes no attempt to diagnose adult players with signs of Attention Deficit Hyperactivity Disorder either.

Diagnosis within Game

The research that was gathered during the research stage of this project only showed significant changes to the mean scores of control subjects vs. adult patients suffering from symptoms of Alzheimer’s disease. No significant changes were found among children with ADHD and children that are typically developing, and a child’s ability to perform at the Corsi task improves until around the age of 14. [8]

Per the reasons above, the game makes no attempt to diagnose a child under the age of 14, whom are still developing their visuo-spatial short term memory. Among teenagers 14 and above, and adults the game diagnosis players based upon their achieved Corsi score as “Likely not AD” for a score 5 and above, and “Possibly AD” for scores less than 5.

Object-Oriented Software Design

Inheritance

Inheritance is an important part of object-oriented programming, and in turn software created with an object-oriented software language should use the design philosophy of OOP. There are several classes of objects that are used within the game. Most object classes in the game either extend or implement classes / interfaces. All of the in game buttons have classes tied to them that implement the ActionListener interface which has methods that are called when a user performs a particular action, like pressing a button.

While implementation can be used to inherit functions and properties of an interface, the “extends” keyword can be used to inherit functions and properties of classes. Within the game, most of the various “screens” that are displayed within the game are inherited from the Menu class so that they can inherit functions that are utilized within the men system. The “Menu” class itself inherits properties and functions from the JPanel Java class so that it can be added and removed from the main JFrame inherited class, Game Screen.

Encapsulation

Due to the object-oriented design of the Java software language, measures for encapsulating data and variables have already been defined. The game takes advantage of this to hide sensitive variables using encapsulation and private data fields within object classes. All variables that need to be accessed from other classes in the game have been protected with proper “Getter” and “Setter” functions that are required for Java’s built-in variable encapsulation.

Anti-hacking Measures

Hacking and data privacy are some of the biggest concerns facing the software industry in recent years, and video game software is no exception to this. Hackers could potentially want to reverse engineer the software for a number a reasons including cheating, piracy, and gaining access to other people’s sensitive information. To prevent this as much as possible, many steps have been taken in the creation of this game to ensure security.

Encryption

Due to the software featuring a saving & loading system that is able to store potentially sensitive data (i.e. address, diagnosis, etc.) the application must attempt to hide this data via encryption.

To achieve this, all data is encrypted with DES encryption before saving to the designated save file. [3] Each player entry within the save file creates a unique encryption key to use with all data that is stored for that specific record. The encryption key is then stored on the file within that specific player record.

Obfuscation

The structure of the save file that is created by the software is a JSONArray of JSON Objects. Therefore, each “value” that is stored in the file has a corresponding variable name that matches to the value. This means that one could look at the outputted save file, and be able to read variable names and their values. These values are encrypted, but their corresponding names are not, meaning that a potential hacker would be able to read the names of the variables saved inside the file. One could simply look at the file and see a variable named “key” and could correctly assume that variable to be an encryption key, and unlock the ability to reverse engineer the software.

To combat this, variables that are written to the save file obfuscated, meaning their names have been changed in a way that gives no meaning for what that variable represents. For example, the variable that stores the encryption key for each player has been renamed to “rjc8qhtv1w” instead of “key”.

Exceptions

There are a few functions written into the game that attempt to “catch” an exception if one occurs so that normal game functionality is not broken. One example of this, is in the simulation menu, a text input box asks for an integer input. The game has a function that throws and catches the exception of attempting to use whatever is in said text box as an integer value. Typing in a string in that text box, for example, throws an exception and catches the exception within the same function to provide to the game the Boolean value of ‘false’, signifying that the string is indeed not an integer value. Other examples of Java’s try-catch blocks are used within the game, mostly to handle the process of multi-threading.

A great amount of consideration and steps were taken in the implementation of this software to ensure that the user is not able to “break” the game by causing the software to throw exceptions that it cannot handle. All fields that ask for text input are programmed to print an error message to the user that the text field is blank. In addition, text fields that ask for input in a specific format (i.e. zip code and birthdate fields) have been programmed to print an error message if the text is not in the correct format.

Precaution has been taken to ensure that a player cannot create a new game with a username that is already saved in the save file. If a user attempts to do this, an error message will show to the user that the username is already in use.

In order to prevent the game from having display issues, a “pause” screen has been implemented to automatically display in the case of the software being minimized. From both the main game loop and the simulation game loop if the game is minimized, the game automatically goes to the pause screen.

“Dev” Mode

A special mode has been integrated into the game to perform simulations of previously played games. The mode is not accessible from the start of software. Instead in order to access the game simulation mode, a special “dev” mode must be unlocked first. In order to unlock the dev mode, a user must go the “load game” screen ant type in the name “DEV\_MODE”. The game will show a message that dev mode has been unlocked, and “Simulation” will now appear within the main menu.

Game Simulation

Once the “dev” mode has been unlocked, “Simulation” made can be accessed through the main menu. In simulation mode, a user can simulate a passed saved game. The user is brought to a screen that asked to input a username. Once the username has been input, the program states how many times that user has attempted the game and asks which attempt the user would like to simulate. After the user has filled out this information, the software then goes into the simulation game loop, reading in all the data from the interactions array to produce a simulation of the specified game.

Conclusion

Not enough data was collected during development of the software to prove or disprove the hypotheses of this project. Unfortunately, during the development stage of this software I was the only one who played through the game, and had to do so using multiple save profiles that simply not enough data was collected to make an inclusive decision about the hypothesis.

References

1. Aaai.org. (2019). *Formatting Instructions Word*. [online] Available at: http://www.aaai.org/Publications/Author/formatting-instructions-word.pdf [Accessed 7 Oct. 2019].

2. Bodnar, J. (2019). *Java Swing first programs - JFrame, JPanel, JButton, GroupLayout, JLabel*. [online] Zetcode.com. Available at: http://zetcode.com/tutorials/javaswingtutorial/firstprograms/ [Accessed 7 Oct. 2019].

3. Ghataliya, Jatin, et al. “JCE Encryption – Data Encryption Standard (DES) Tutorial.” *JCE Encryption – Data Encryption Standard (DES) Tutorial – Mkyong.com*, www.mkyong.com/java/jce-encryption-data-encryption-standard-des-tutorial/.

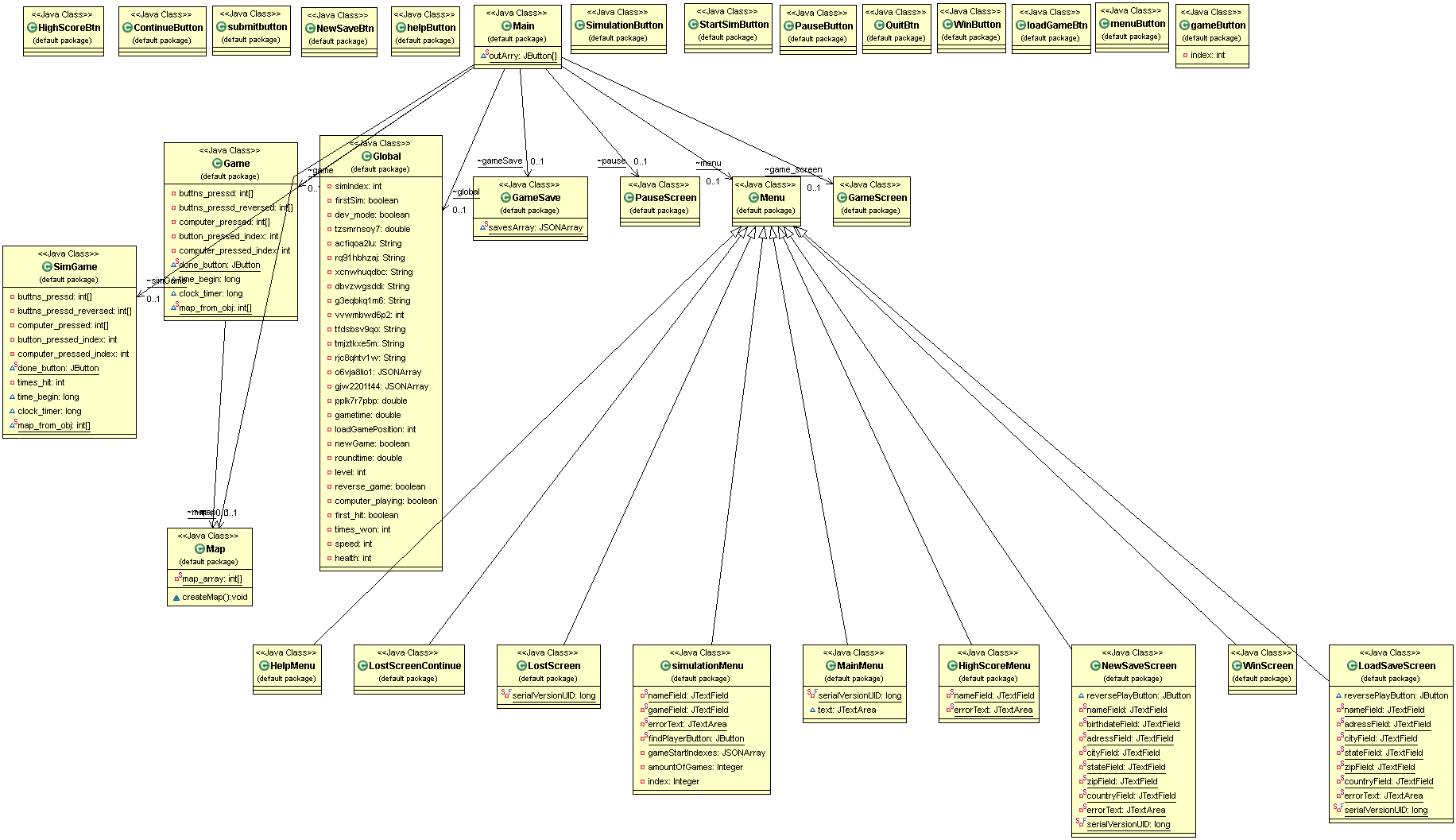
4. Guariglia, Carla Cristina. “Spatial Working Memory in Alzheimer's Disease: A Study Using the Corsi Block-Tapping Test.” *Dementia & Neuropsychologia*, vol. 1, no. 4, 2007, pp. 392–395., doi:10.1590/s1980-57642008dn10400011.

5. Gupta, L. (2019). *Java read json and write json example - JSON.simple tutorial*. [online] HowToDoInJava. Available at: https://howtodoinjava.com/library/json-simple-read-write-json-examples/ [Accessed 7 Oct. 2019].

6. H. (2019). *How to append data on a existing Json file using Java?* [online] Stack Overflow. Available at: https://stackoverflow.com/questions/50402343/how-to-append-data-on-a-existing-json-file-using-java [Accessed 7 Oct. 2019].

7. Mioni, Giovanna. “Difficulties of Children with Symptoms of Attention-Deficit/Hyperactivity Disorder in Processing Temporal Information Concerning Life Events.” *Department of General Psychology, University of Padova, Italy* , vol. 35131, 25 Jan. 2018.

8. Pagulayan, Kathleen Farrell, et al. “Developmental Normative Data for the Corsi Block-Tapping Task.” *Journal of Clinical and Experimental Neuropsychology*, vol. 28, no. 6, 2006, pp. 1043–1052., doi:10.1080/13803390500350977.



1. Copyright © 2019, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved. [↑](#footnote-ref-1)
2. Copyright © 2019, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved. [↑](#footnote-ref-2)