Implementation of a Platform-Agnostic Working Memory Span Task Using Mobile Device Technology *

Brent G. Nelson, Elias Boroda, Kelvin O. Lim

***Abstract*— Working Memory Span (WMS) tasks are some of the most commonly used tools in cognitive psychology. These tasks involve the presentation of a set of items in a specified sequence. The participant is then asked to recall the sequence of items in forward or reverse order with performance quantified by the maximum number of items correctly recalled. Though initially the test was developed for verbal administration, now with modern computing, these tasks are generally administered via a digital interface. As a result, a variety of implementations have been developed but generally target either research or consumer domains. There has yet to be an offering capable of making the necessary transition from validation in the research lab, through implementation in the clinic, to distribution directly to the consumer. The main barrier has been lack of an open, approachable architecture, robust enough for basic scientists but also technically scalable for wide distribution to consumers. Recent advances in software technology provide the necessary infrastructure for the development of a robust offering. We have developed a WMS task, built on the latest in multi-platform gaming technology, to act as a blueprint for how future tasks can be developed, tested, and distributed. Our implementation describes some of the fundamental principles necessary for cultivation of a rich ecosystem of health diagnostic and therapeutic software.**

I. Introduction

Outside of standardized cognitive batteries, Working Memory Span (WMS) tasks are amongst the most widely utilized measurement tools in cognitive psychology [1]. As research tools, WMS tasks have been utilized to help elucidate cognitive impairments in a host of neuropsychiatric disorders including schizophrenia, ADHD, and major depressive disorder [2]–[4]. Clinically, WMS tasks are employed as both diagnostic tools as well as therapeutic interventions to treat these complex disorders [5]–[11]. The common use of WMS tasks highlights the importance of these tasks to both researchers and clinicians in dealing with, and better understanding working memory, a critical cognitive domain.

Historically, WMS tasks have been in use for over four decades and were developed out of a theory of working memory proposed by Baddeley and Hitch [9]. This theory described working memory as an immediate memory recall system that could briefly and dynamically store a limited amount of task relevant information; this was termed working memory capacity (WMC). WMS tasks were developed as a means to test this model by indexing an individual's WMC. WMC tasks involve the participant reading, viewing, or listening to a series of serially presented stimuli and recalling, in correct order (may be forward or reverse order), the set of presented stimuli. The nature of the stimuli used is highly variable (any set of stimuli can theoretically be presented) but digits, words, or symbols are most commonly utilized [4]. The number of stimuli (or elements) recalled in correct order is regarded as the working memory span, a measure of WMC. Many studies then use this span as a measure to correlate to type and severity of disease [11].

While the first span tasks were conducted by verbal interview and written response [12], they were soon digitized as computing technology became more widely available. Specifically, the Digit Span was one of the first cognitive tasks to be computerized [12]. This task is simple in its design and involves showing a participant a series of numbers through a display interface, providing a delay with a blank interface, and then requesting the participant to enter these numbers via a response interface. Typically, there are configurable task parameters such as number of items to display, item composition, delay time, and response time limit. Usually tasks will track participant performance data such as total correct responses and time taken to complete the task.

Many variants of the WMS task has been developed since the introduction of the reading span by Baddeley & Hitch [9] and generally substitute symbols (symbol span) and digits (digit span) for sentences. These tasks, though differing in stimulus type, are comprised of several critical components. First of these is immediate and vigilant stimulus presentation. A crucial feature of any WMS task is that it impedes rehearsal of the presented stimuli. Substantial delays between stimulus presentations will encourage rehearsal, thereby making the task a measure of short term memory storage rather them WM. It is important that stimuli are presented with a brief inter-stimulus interval (ISI), the time delay between stimuli, and subjects must be instructed to act on the stimuli immediately. Related to ISI is the time limit the participant is given to recall the memory item; this is the response time limit (RTL). Though this time limit varies greatly depending on the specific task implementation, it is a common feature that can be used to limit task time and as well as enhance working memory load. Enforcing a time limit to responses creates a stressful situation which can lead to increased recruitment of working memory resources [13].

* Research supported by a University of Minnesota Faculty Seed Grant.

B. G. Nelson is with the University of Minnesota, Minneapolis, MN 55455 USA (corresponding author to provide phone: 952-525-4500; fax: 952-525-1560; e-mail: nels1741@umn.edu) and with the Minneapolis VA Health Care System, Minneapolis, MN 55417 USA.

E. Boroda is with the University of Minnesota, Minneapolis, MN 55455 USA.

K. O. Lim is with the University of Minnesota, Minneapolis, MN 55455 USA and with the Minneapolis VA Health Care System, Minneapolis, MN 55417 USA.

978-1-5090-3090-3/16/$31.00 ©2016 IEEE

Adaptive difficulty is also crucial for proper testing of WM. Item size, or span length (SL), is a unique feature of WMS tasks and refers to the number of elements presented within each item. For example, a digit span task may begin with an SL of just two elements (two digits), which are to be recalled in order (or reverse order – a common variant) by the participant. As the participant demonstrates competence with a given SL, more elements can be added to increase difficulty. On the contrary, if performance at a given SL is not satisfactory, the SL can be decreased, lessening WM load. The maximum number of elements a participant is able to recall is considered the memory span (MS) and is a measure of working memory capacity. The exact range of item sizes, as well as the adaptive algorithms used to control SL, will vary depending on the task design.

The technology used to implement the task, data storage methods, configuration, performance collection, and even visual display are all left to the task implementer to design. The result is a varied and unpredictable landscape of span tasks all with different technical and scientific implications. A standard architecture is needed for task implementation, execution, and deployment. In this paper we will describe existing technology limitations, new technology opportunities, and a novel reference architecture. We will also provide a concrete implementation of a WMS task to show relevant details of our proposed design.

II. Motivation

A. Existing Implementations

A broad range of working memory task implementations are available and exist on a continuum from research lab deployments all the way to commercial consumer offerings. In the past, research tasks, due to available technology, were implemented as part of large, costly, rigid, systems that were temperamental to manage. Frequently the task was developed using niche programming languages requiring a high degree of technical knowledge. Tasks were tied to a specific operating system, and frequently a particular technical environment, requiring re-implementation should the task wish to be shared. Commercial tasks, on the other hand, are generally created using modern languages and technology, but due to their commercial nature, are frequently closed source and unavailable to all researchers.

An underlying goal of all scientific endeavors is to make discoveries that are repeatable, extensible, and capable of being generalized to the broader world. This true of tasks as well. Tasks are developed and tested in the lab often along with other assessments such as magnetic resonance imaging. Once the task has been tested, and shown to have biological validity, it should be distributed, unaltered, to the clinic and then on to consumers. Without standard approaches between research and commercial platforms, a gap will remain preventing scientific translation from the lab to the bedside.

TABLE I
Comparison of past technology design principles compared to present design expectations.

	Present	**Past**
Source Model	Open	Closed
Cost	Free	High
Operating System	Cross Platform	Platform Specific
Interoperability	Composable	Isolated or Tightly-coupled
Development Language	Common (C#, JAVA, JavaScript, Python)	Custom
Distribution	Platform Store (iOS, Android, Windows)	Hand Installed by Developer or Technologist

B. Technology Trends

Due to the advent of new technology, specifically mobile devices, new methods are available. Applications can be small and single-purpose, designed with flexibility and wide distribution in mind. Applications are frequently distributed via application stores such as the Apple Store [14], the Google Play Store [15], and the Microsoft Store [16]. Each distribution platform provides clear standards guiding how applications are packaged and deployed. These mobile application containers are specific to each platform but share common best practices focused on themes such as security, energy consumption, data storage / access, and data sharing. This provides developers with an avenue to deploy sophisticated applications to a wide audience in a predictable and reliable way.

C. Task Containers

Over the last 5 years, many specifications have been developed around application containers [17], especially in the mobile space. Given the early nature of the containerization field, much time has been spent on low level issues such as application execution controls, binary formats, data storage, and security constraints [18]. Surprisingly there is little in the way of inter-container communication standards or domain-specific work-flow standards. This lack of definition has made it difficult to implement and deploy tasks capable of being integrated into complex and varying workflows.

Similar to the container standards built for security and storage, standards need to be developed for task implementations; specifically, with regard to communication between applications when it comes to configuration and data collection. The tasks must contain robust configuration controls for meeting the rigorous requirements of research but also be able to adapt to the more flexible, less technical world of the consumer. This must be able to be accomplished through the use of platform agnostic data standards such as JavaScript Object Notation (JSON) [19] and Universal Resource Identifiers (URIs) [20].

III. Technical Implementation

A. Architecture

We have developed a novel architecture that builds on application containers [21]. Our design is modular in nature

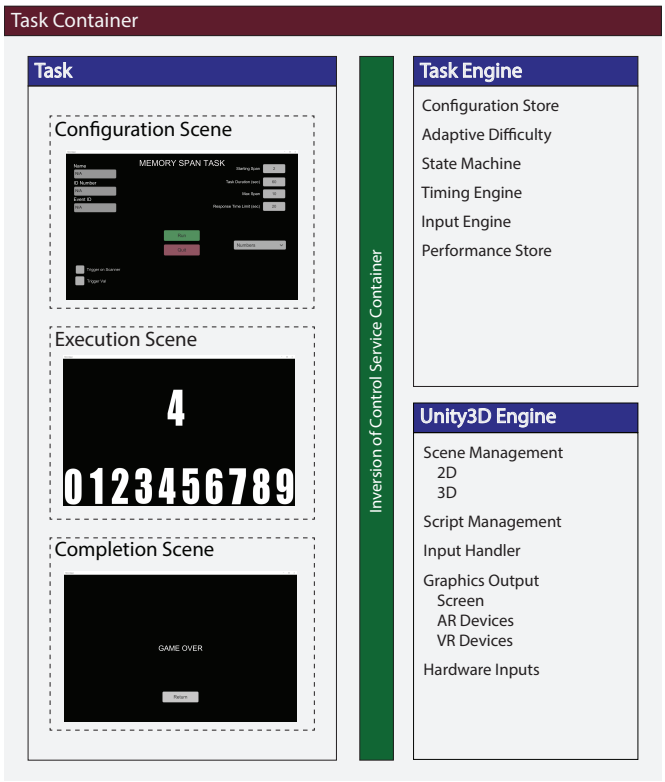

Fig. 1. Graphical illustration of task architecture and components.

and is composed multiple building blocks. Figure 1 shows these blocks along with their relationship to each other.

1) Unity3D Engine: Our platform and tasks are developed using the Unity software development environment. [22] This is a widely-used, multi-platform game engine with an easy to use editor and an advanced physics engine. The environment consists of a code editor and a visual asset development interface. These tools simplify task development, as they not only facilitate the programming that is required for task logic but also manage the complex graphical space which supports gameplay. It is robust enough to support games ranging from simple two dimensional arcade games all the way to rich, immersive, three dimensional worlds.

In order to organize the complexity of the graphics system, Unity utilizes the concept of scenes. Scenes define the virtual three dimensional (or two dimensional) space where the task operates. Scenes are composed of visual objects such as meshes, buttons, and images, as well as non-visual objects such as sounds and cameras. Scenes are typically created for each major piece of visual functionality, such as game levels or configuration screens. Scenes are loaded independently from one another as the game transitions from one module to the next.

2) Task Engine: The task engine is a layer of shared logic modules grouped together into a library and delivered via the Unity Asset Store [23]. These logic modules provide functionality that is shared between tasks of different types. Examples of shared items include configuration management, adaptive difficulty adjustment, timing management, and performance capture.

Internally, the engine is implemented as a standard Inversion of Control (IoC) service layer [24]. Not only does this allow tasks to be more focused in their implementation, it also ensures tasks follow standards for configuration, execution, and data management defined by this architecture

3) Task: This is the core unit of work in our system and is what is created by the research task developers. It contains all of the logic to assess a specific cognitive activity. In our WMS task, it is concerned only with testing working memory. It does not worry about any other operations such as data logging or timing. This is left to the underlying engine which is described above

Task logic is written in either JavaScript [25] or C# [26]; both widely accepted programming languages. Task logic is the code responsible for manipulating objects in a scene. Unity is designed in such a way that common parameters and features in both scenes and code are exposed to the graphical interface. This allows objects to be wired to each other using simple drag and drop techniques. Parameters and functionality are easily changed via the editor without having to access the underlying scripting; making it easy for users who are not proficient in programming to implement changes as needed.

It is preferable for a task to be as focused as possible to a cognitive or functional domain so effects are clear, reliable, and reusable.

4) Task Configuration: A small JSON document (Fig 2), containing configuration information specific to a task for a particular participant. It contains information about task parameters as well as info for how and where the task engine is supposed to log and store data. If the task is not provided with any configuration data, then it is responsible for collecting configuration information interactively or to run with reasonable defaults.

```
{
  Name : "555555",
  StartingSpan : 2,
  TaskDuration : '60s',
  MaxSpan : 10
  ResponseTimeLimit : '20s',
  SpanType : 'Numbers'
}
```

Fig. 2. Example task configuration JSON. Would be provided programmatically to task at runtime.

5) Task Container: This is a thin wrapper created to house a specific task on top of the task engine. This provides execution capability on a specific platform (e.g. Windows, MacOS, Android, iOS) depending on compilation. This wrapper is automatically generated by the Unity build processor. All of the internals are cross-platform and communicate to the outside environment only via data standards such as JSON [19].

6) Task Environment: This is an application, separate from the task, and is compiled to a specific platform. It is responsible for the high-level orchestration of tasks and management of both task configuration and task data collection and management. Most often the task environment application will be a custom, thin, piece of software developed for a specific deployment and data store. While most environments will be developed by individual parties, there is the potential for shared environments to be provided for common cloud-based data stores; with the specific selection of task environment based on deployment needs (research, clinic, consumer). Development specifics of an environment is beyond the scope of this article, but the general profiles are described briefly to illustrate how the architecture and tasks would map onto different environments.

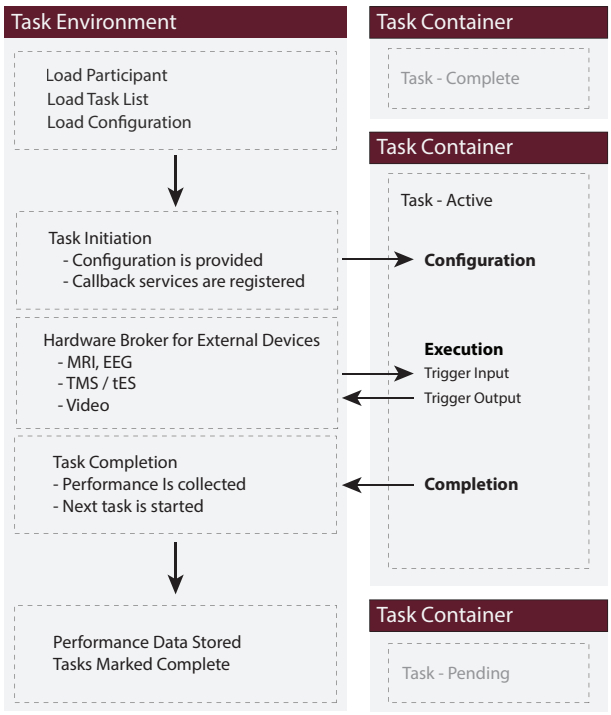

Fig. 3. Visual representation of the interaction between the task environment and the individual tasks.

In a research lab, for example, the environment communicates with the cloud storage provider to get information on specific participant, tasks, and task configuration. Then the environment orchestrates the tasks, one after another, to ensure the tasks are run as expected in a repeatable manner. Once complete, the environment sends the resulting data back to the cloud provider so it can be stored and analyzed. A clinic may use a similar configuration but may also have connections to the electronic health record or other clinic data systems. In a direct-to consumer scenario, the environment is a simple shell that allows the patient to select a task, configure how they would like to run the task, and then give them feedback as to how they performed.

B. Implementation

Our WMS task is organized into 3 different scenes, the configure scene, the main scene, and the end scene. The configure scene is the initial screen presented to the participant to collect any remaining pre-task configuration or user information. The main scene is where the actual WMS task is executed and working memory is measured. The end scene is presented following completion of the task to provide the user with performance feedback.

1) Task Configuration: The configure scene is the opening scene of the WMS task and presents configuration options to the user. As described in the architecture, this scene is optional. Should the user provide configuration to the task, programmatically from another application, this scene will not be shown and the task will go directly to the next scene. Should only part of the configuration values be provided, the task will use defaults for missing mandatory values. If no configuration is provided programmatically, then this scene is shown and will wait for the user to provide values interactively.

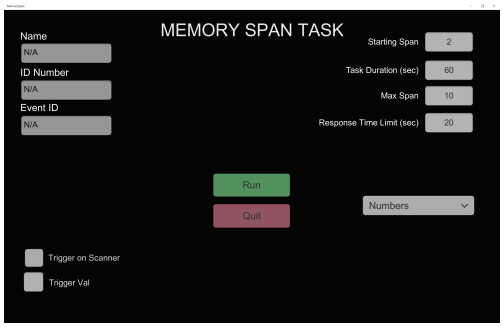

Fig. 4. Visual representation of the configuration scene.

The center of the screen presents two colored buttons, the green Run button will begin the task, while the red Quit button will exit the application. The scene also contains multiple configuration fields. It has fields to collect user identification, device triggering, and task execution settings. User identification is a simple text field for capturing a custom identifier for the user interacting with the task. The task can also be configured to trigger based on an external device as specified by a keystroke code. This is frequently used to align the task with functional magnetic resonance imaging volume acquisition.

The upper right hand of the screen features input fields for various task settings. The first of these is Starting Span. This setting determines the SL of the first iteration presented. If this value is not provided via the corresponding input field, the default length of two will be used. The next two input fields set the conditions for ending the task. The Task Duration input field sets the maximum length of the task in seconds. The Max Span input field sets the maximum unit length. Whichever condition (time or span length) is reached first will cause termination of the task. The Response Time limit field, corresponds to the RTL, and sets the amount of

time (in seconds) that the user will have to recall and respond to the previously presented stimulus.

A drop down menu on the right side of the screen allows the user to select which type of stimuli will be presented and thus what type of WMS task will be executed. Currently there are two options in this list, digits and symbols, allowing the execution of the classic digit span (digits 0-9 as stimuli) or a symbol span using custom made symbols. Other types of stimuli can easily be added to this list by adding a sub-folder of images to a specific resources folder in the Unity hierarchy (while in editor mode). Once images are added to the folder, the name of the folder will appear in the drop down list and those images can then be presented to the user. This feature imparts a high degree of flexibility to our WMS implementation as it allows many types of WMS tasks to be deployed.

2) Task Execution: This main scene handles the actual task execution. Task procession begins with the presentation of a randomly ordered set of stimuli to the participant. Each stimulus is displayed for 800 ms with a 500 ms ISI. To collect user responses, a panel of stimulus images will appear along the bottom of the screen, allowing the participant to click, or touch if on a tablet, the images in the correct order. For example, for a digit span, the panel would consist of the set of images showing 0-9 going from left to right. The participants responses are briefly displayed in the center of the screen as they make their selection. Users have an RTL to make their responses, and after this time period, the program evaluates the input and provides a brief reinforcement in the form of a red cross (incorrect) or a green check (correct) image.

Fig. 5. Visual representation of the execution scene.

In order to test working memory capacity, the task adjusts SL to adequately challenge the participant. To accomplish this, the task employs an adaptive difficulty algorithm by tracking user performance; SL is modified accordingly on each iteration. The algorithm adjusts SL by 1 (in either direction) after the participant meets a satisfactory level of performance. The algorithm does not make an adjustment until at least 5 responses have been made at a given SL. The program calculates a percent correct (number of correct responses over total responses). If performance is better than 90%, and maximum SL has not been reached, then the SL is increased by 1. If performance is below 70%, and current SL is greater than starting SL, difficulty will be decreased by 1. On the other hand, if performance is between these two values, the SL will remain constant.

3) Task Completion: The end scene, also an optional scene, is displayed after the task has completed. This scene is only displayed if the task was run interactively, that is without programmatic configuration by another application. The purpose of this scene is to notify the user the task has ended and to provide the user with feedback on performance.

Fig. 6. Visual representation of the completion scene.

```
{
  Name : "555555",
  StartingSpan : 2,
  MaxSpan : 10
  ResponseTimeLimit : '20s',
  SpanType : 'Numbers'
  Correct : 8,
  Incorrect : 2,
  TaskDuration : '3m 21s 13msec',
}
```

Fig. 7. Example task completion report JSON. Would be provided programmatically back to the task environment at runtime.

Figure 6 shows just one of the output reports provided by the task. Additional raw data is available should more in-depth analyses wish to be performed.

C. Data Capture and Logging

The WMS task uses the task engine, which contains a robust library for capturing and logging all aspects of data generated from its execution. It tracks configuration parameters used for each run, raw performance for each iteration of the task, and summary measures about overall execution accuracy and speed.

Should the task be run programmatically by an external application, the data will be captured and stored in accordance with configuration parameters passed to the task. Typically, this will entail writing data logs to a folder specified by the external application so data can be captured and uploaded to an external data warehouse. Since this task is general purpose it will not need to be changed in order to support any external data warehouse. Instead the calling application, the task environment, will take care of these details and provide

a place for the WMS task to write its data. When the task is run interactively, the data are stored in the application folder which is a standardized location specific to each platform.

D. Distribution

A central feature of the Unity development environment is that it is capable of producing applications for most of the popular technology platforms including Windows, Mac, iOS, Android, and Linux. This allows deployment decisions to be a matter of software build rather than custom programming. Distribution for the WMS task was a matter of downloading the source code from Git, opening Unity, and then building for a particular platform. Once a build is finished, the resulting binary is uploaded to its respective Application Store and deployed directly to users.

For the purposes of this communication, we have provided a working application to serve as both a proof of concept and blueprint for additional task development. To that end, the WMS task has been built for and published to the Windows 10 Application Store [16] as a Universal Windows Platform Application [27]. Our WMS task is downloadable directly from the store and capable of being used directly by users or integrated into a broader work-flow using the aforementioned task environments.

IV. CONCLUSION

Working Memory Span tasks are a central tool in cognitive neuroscience. Up until now, task implementations have required a large effort in order to test in the lab and then distribute to the clinic and consumers. We have proposed a new architecture, with an example implementation of a WMS task, as a modern option for the development and deployment. Through our architecture and design principles, we have shown a new model on which other tasks can be quickly developed and deployed.

Our tasks are currently undergoing validation with functional magnetic resonance imaging to show our implementation provides results consistent with prior work. Also, additional common tasks will be released using this architecture to provide additional examples of how to implement a wide range of complex cognitive tasks. It is our hope that, in the near term, a rich landscape of tasks will be available on the application stores for all to use.

REFERENCES

[1] A. R. A. Conway, M. J. Kane, M. F. Bunting, D. Z. Hambrick, O. Wilhelm, and R. W. Engle, "Working memory span tasks: A methodological review and user's guide." *Psychon Bull Rev*, vol. 12, no. 5, pp. 769–786, oct 2005. [Online]. Available: http://www.springerlink.com/index/10.3758/BF03196772

[2] K. Matsuo, D. C. Glahn, M. A. M. Peluso, J. P. Hatch, E. S. Monkul, P. Najt, M. Sanches, F. Zamarripa, J. Li, J. L. Lancaster, P. T. Fox, J.-H. Gao, and J. C. Soares, "Prefrontal hyperactivation during working memory task in untreated individuals with major depressive disorder." *Mol Psychiatry*, vol. 12, no. 2, pp. 158–166, feb 2007. [Online]. Available: http://dx.doi.org/10.1038/sj.mp.4001894

[3] R. E. Gur, M. E. Calkins, R. C. Gur, W. P. Horan, K. H. Nuechterlein, L. J. Seidman, and W. S. Stone, "The consortium on the genetics of schizophrenia: neurocognitive endophenotypes." *Schizophr Bull*, vol. 33, no. 1, pp. 49–68, jan 2007. [Online]. Available: http://dx.doi.org/10.1093/schbul/sbl055

[4] A. R. Conway, N. Cowan, M. F. Bunting, D. J. Therriault, and S. R. Minkoff, "A latent variable analysis of working memory capacity, short-term memory capacity, processing speed, and general fluid intelligence," *Intelligence*, vol. 30, no. 2, pp. 163–183, mar 2002. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0160289601000964

[5] L. Lawlor-Savage and V. M. Goghari, "Working memory training in schizophrenia and healthy populations." *Behav Sci (Basel)*, vol. 4, no. 3, pp. 301–319, sep 2014. [Online]. Available: http://dx.doi.org/10.3390/bs4030301

[6] T. Klingberg, E. Fernell, P. J. Olesen, M. Johnson, P. Gustafsson, K. Dahlstrm, C. G. Gillberg, H. Forssberg, and H. Westerberg, "Computerized training of working memory in children with adhd–a randomized, controlled trial." *J Am Acad Child Adolesc Psychiatry*, vol. 44, no. 2, pp. 177–186, feb 2005. [Online]. Available: http://dx.doi.org/10.1097/00004583-200502000-00010

[7] A. B. Morrison and J. M. Chein, "Does working memory training work? the promise and challenges of enhancing cognition by training working memory." *Psychon Bull Rev*, vol. 18, no. 1, pp. 46–60, feb 2011. [Online]. Available: http://dx.doi.org/10.3758/s13423-010-0034-0

[8] D. M. Barch, Y. I. Sheline, J. G. Csernansky, and A. Z. Snyder, "Working memory and prefrontal cortex dysfunction: specificity to schizophrenia compared with major depression." *Biol Psychiatry*, vol. 53, no. 5, pp. 376–384, mar 2003. [Online]. Available: http://dx.doi.org/10.1016/S0006-3223(02)01674-8

[9] A. Baddeley and G. Hitch, "Working memory," *Psychology of learning and motivation*, 1974. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0079742108604521

[10] J. M. Gold, C. Carpenter, C. Randolph, T. E. Goldberg, and D. R. Weinberger, "Auditory working memory and wisconsin card sorting test performance in schizophrenia." *Arch Gen Psychiatry*, vol. 54, no. 2, pp. 159–165, feb 1997. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/9040284

[11] D. P. McCabe, H. L. Roediger, M. A. McDaniel, D. A. Balota, and D. Z. Hambrick, "The relationship between working memory capacity and executive functioning: evidence for a common executive attention construct." *Neuropsychology*, vol. 24, no. 2, pp. 222–243, mar 2010. [Online]. Available: http://dx.doi.org/10.1037/a0017619

[12] M. C. Ramsay and C. R. Reynolds, "Separate digits tests: a brief history, a literature review, and a reexamination of the factor structure of the test of memory and learning (tomal)." *Neuropsychol Rev*, vol. 5, no. 3, pp. 151–171, sep 1995. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/8653107

[13] P. Barrouillet, S. Bernardin, and V. Camos, "Time constraints and resource sharing in adults' working memory spans." *J Exp Psychol Gen*, vol. 133, no. 1, pp. 83–100, mar 2004. [Online]. Available: http://dx.doi.org/10.1037/0096-3445.133.1.83

[14] "App store downloads on itunes." [Online]. Available: https://itunes.apple.com/us/genre/ios/id36?mt=8

[15] "Android apps on google play." [Online]. Available: https://play.google.com/store/apps

[16] "Windows apps - microsoft store." [Online]. Available: https://www.microsoft.com/en-us/store/apps/windows

[17] G. Banga, P. Druschel, and J. C. Mogul, "Resource containers: A new facility for resource management in server systems," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 45–58. [Online]. Available: http://dl.acm.org/citation.cfm?id=296806.296810

[18] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support PaaS," in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. ieeexplore.ieee.org, Mar. 2014, pp. 610–614.

[19] "Rfc 7159 - the javascript object notation (json) data interchange format." [Online]. Available: https://tools.ietf.org/html/rfc7159

[20] "www.ietf.org/rfc/rfc2396.txt." [Online]. Available: https://www.ietf.org/rfc/rfc2396.txt

[21] M. J. Scheepers, "Virtualization and containerization of application infrastructure: A comparison," in *21st Twente Student Conference on IT*. referaat.cs.utwente.nl, 2014, pp. 1–7.

[22] "Unity - game engine." [Online]. Available: https://unity3d.com/

[23] "Unity - asset store." [Online]. Available: https://www.assetstore.unity3d.com/

[24] "Inversion of control containers and the

dependency injection pattern." [Online]. Available: http://martinfowler.com/articles/injection.html#ServiceLocatorVsDependencyInjection
[25] "Ecmascript®2016 language specification." [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf
[26] "C# language specification." [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf
[27] "Get started with windows apps - windows app development." [Online]. Available: https://developer.microsoft.com/en-us/windows/getstarted