<Title of the report>

Student technical report – “Interactive Simulation Systems” course

Manuel Hernández Nájera-Alesón  
January, 29th, 2018

Supervisor: Siniša Popović

***Abstract* – Memory development under stress is a useful capacity for human beings. In this project It will be measured through a demanding task.**

# Introduction

Spatial memory has always been crucial for humankind when performing either 2 or 3-dimensional work. We depend heavily on our way of understanding and interacting with space and, of course, our way of remembering It’s setting.

The aim of this project is to evaluate how different human beings can interact with 2D environments while under a situation of sudden stress caused by a fast-paced task accompanied by loud and screaky noises.

The individual will switch from a tranquil and low-demanding environment, in which they will be able to walk and see around the formerly mentioned task, which they will have to complete in a limited amount of time under extremely harsh circumstances.

This will allow us to measure the individual’s capability to handle sudden and stressful situations (e.g. surgery, aviation) and their efficiency whilst performing it.

There are some examples of spatial working memory tests [1].

The project has been developed with Unity 2017.2.0f using scripting language C# and Visual Studio IDE. .NET Socket communication was used to transmit the data from unity to the server.

# Unity approach

When thinking about the best approach to this project realization, It came out that an already-built game-development-engine would be the best option. After considering other alternatives, Unity was the most suitable option.

Unity is a multiplatform videogame engine, available for Microsoft Windows, OS X and Linux. It provides the developer with a compilation support with a wide range of platforms, a graphical engine, a physical engine and a complex object hierarchy.

Unity was the best approach to this problem because it consist's of a set of scenes that can be loaded one after another until the end of the game execution.

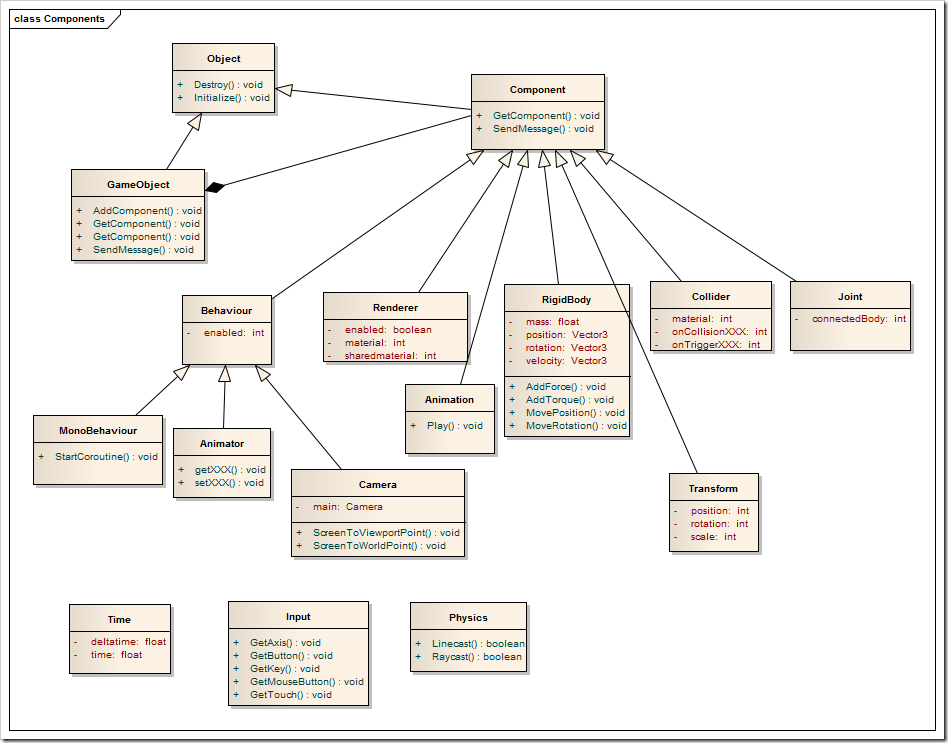


Figure . Basic Unity GameObject structure example. [1].

# General Project Architecture

The general architecture of this project is resembled in figure 2. We have a Unity program running, which will take the parameters we will give it in a Setup file. With those parameters we will generate our program, with it’s scenes and record the users input (clicks made). Then, with a TCP server created during the execution of the Unity Program, we will send the data to a TCP Client, that will output the data to the system console.

So, summarizing we have 2 types of communication involving the Unity program: TCP communication and Setup File communication.

Unity Program

Side Program

SETUP FILE

Figure 2. Basic Project Architecture

## TCP Client and Server

When facing to the communication between the side program and the unity process, .NET socket communication was the option chosen.

TCP Server is run by a parallel thread of the Unity program, and has methods to be notified if one scene starts or ends or even send a string as a message. The Server has also a buffer (implemented as a double linked list), so when the main unity program wants to send something to the client, it stores that message in the buffer. TCP Server will perform a buffer read from time to time, and send any information in the buffer.

TCP Client is a Side process that is executed upon the start of the Unity Program. The lifespan of this process is bound to the main program’s lifespan, as it is closed when receiving an end message.

Communication between these two processes is made using socket communication through the local IP address of the machine, and is transmitted by bytes. When the TCP Server starts reading the buffer, it parses every byte of the string and sends it through the socket. When those bytes reach the client, a 2-phase reading takes place:

* Firstly, the 4 prior bytes of the string are read and stored. If the 4 of them are equal to **0xFF** that means we just received a ‘*Stop’* signal, so the rest of the reading is aborted and the client proceeds to close.
* If former condition is not met, then we continue with the read and parse it to string. Proceeding then to write it in the console.

Communication is implemented in a way that, if another, more complex, side application is needed, the communication would not be a problem to reach.

## File-Unity Communication

The communication between the task-programmer and the application is made via file reading. In this way we can configure how tasks and rests are scheduled.

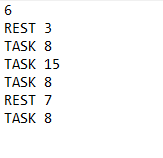


Figure 3. Example of how the setup text file should look like.

The file is found in *“.../Assets/Files/”* and its name should be *“SetupFile.txt”.* The organization of this file is as depicted in figure 3. First, a integer N which value is the number of lines after its line. Then we have the actual setup. N setup lines can be written. In setup lines first, it must be written either “TASK” or “REST” depending on the Unity scene we want to load. After that, a space and an integer must be written, representing the number of total seconds we want the scene to be active before changing to the next one.

# Main Program Architecture

Unity main program is divided into scenes. This scene is act as separated workspaces and graphical environments. It must be considered that we are working with Object Oriented programming, written in C# Scripts. Typically, data from one scenes’ objects is not shared unless we declare an object as non-destructible after the scene finishes. In this project that way of working had to be used and will be explained afterwards.



Figure . Example of a figure. Both grayscale and color figures are appropriate, but color figures should be printed in color when handing in the report. If figure is taken from some source, please indicate this by putting the source in the References section and by stating its reference number in the figure caption, e.g. [1].

This project consists of 5 scenes:

## Start Scene

Simple Scene in which we start with a black screen and white text in it telling the user to press spacebar to start the test. This Scene Jumps to “FileReaderScene”. This scene contains 1 class: StartManager, that implements the input reading.

## FileReaderScene

This is a transitory yet important scene. During the execution of this scene, three scripts will be executed:

* FileReader: Attached to FileReader GameObject it reads the file mentioned in section 3.2. It loads a bi-dimensional array. First row’s values are either 0s or 1s, 0 for rest and 1 for task, the second row is filled with the time value of the *ith* row in the file. On finishing read, this array is passed to SceneLoader object.
* SceneLoader: Created at the start of the scene, it is not destroyed after the scene changes, it is only destroyed when the execution of the Unity program is finished. It stores the task/rest time array. This script is also responsible for scene switching according to the values stored. It performs this task by using the Unity method *Invoke()* which allows the user to make a delayed call to a function of their choice after a given time.
* TCPServer: The Object containing this script is also created at the beginning of the scene, and it is not destroyed after the scene concludes. After its creation, it sets up all the parameters for creating a server and then starts a parallel thread to run that server. (That way, Unity thread will not be stopped when the server is waiting for connection or to send information). The object is killed when execution ends and so is the thread, by a control variable stored in the main unity thread.

## <Subsection 2.1>

This is a specific section which depends on the topic that is being discussed in the report. Therefore, it should be given an appropriate title and its body should contain appropriate text and images. Of course, whether the report will have subsections or not is the author’s decision.

## <Subsection 2.1>

This is a specific section which depends on the topic that is being discussed in the report. Therefore, it should be given an appropriate title and its body should contain appropriate text and images. Of course, whether the report will have subsections or not is the author’s decision.

# Conclusion

Provide conclusions of your work and suggestions of avenues for future work, regardless of whether the report is mostly related to literature review or practical work.

# References

[1] Wang Xuanyi, Unity verview of basic concepts [online]

<http://www.programering.com/a/MDO4UDNwATY>**.**

[2] Cambridge Cognition, Spatial Working Memory

http://www.cambridgecognition.com/cantab/cognitive-tests/memory/spatial-working-memory-swm/

[3] D. S. Lee and J. G. Fossum, “Energy Band Distortion in Highly Doped Silicon”, IEEE Transactions on Electronic Devices, vol. 30, p. 626, 1983.

[4] D. B. Payne and J. R. Stern, “Wavelength-switched passively coupled single-mode optical network,” in Proceedings of *IOOC-ECOC*, 1985, pp. 585–590.

[5] G. Brandli and M. Dick, “Alternating current fed power supply,” U.S. Patent 4 084 217, Nov. 4, 1978.

[6] E. E. Reber, R. L. Mitchell, and C. J. Carter, “Oxygen absorption in the Earth’s atmosphere,” Aerospace Corp., Los Angeles, CA, Tech. Rep. TR-0200 (4230-46)-3, Nov. 1968.

[7] J. Jones. (May 10, 1991). Networks. (2nd ed.) [Online]. http://www.atm.com. Accessed: February 14, 1999.

[8] S. L. Talleen. (June 1996). The Intranet Architecture: Managing information in the new paradigm. Amdahl Corp., CA. [Online]. http://www.amdahl.com/doc/products/bsg/  
intra/infra/html. Accessed: June 8, 1998.

# Appendix

Appendix is an optional part of the report. It is intended for a larger number of similar figures which, if included in the body of the report, would significantly disrupt the flow of text, as well as for multi-page algorithm listings, larger proof derivations etc.