Project Report of Metis 2020:

Experience in Assembly Game Programming

****

Submitted to:

Dr. John F. Doyle

Associate Professor of Computer Science

Presented by:

Junet Bello, Yiliang Lu, and Amanda Schneider

April 29th, 2020

Executive Summary

This team project is to design a game and implement it in the Intel x86/x64 Assembly Language. The target audience will be the students and professor of CSCI-C335 Spring 2020 and any interested visitors to the project Github site. The game is named Metis 2020. It is inspired by a family game played by one of the team members during the holidays, and it is named for the Greek Goddess of Wisdom. Metis 2020 is an interactive, single player, multi-selection trivia game. It has been developed with the Unity frame work using C# and C++ programming languages with in-line Assembly. Utilizing a modern graphical user interface, the player will answer approximately 15 randomly selected questions. The team members hope to learn, apply, and demonstrate Intel Assembly programming skills obtained during the semester.

## Table of Contents

[I. Introduction](#_Toc52873031) 1

[II. Project Members](#_Toc52873031) 1

[III. Project objective](#_Toc52873033) 2

[IV. problems encountered](#_Toc52873034) 2

[V. Problem solutions](#_Toc52873035) 3

[VI. FUTURE WORK and conclusion](#_Toc52873038) 8

[viI. Credits and copyrights](#_Toc52873039) 9

## List of Figures

[Figure1**.** List of key personnel](#_Toc52873031) 1

[Figure 2. The first prototype.](#_Toc52873033) 3

[Figure 3. C++ calls inline assembly functions](#_Toc52873038) 3

[Figure 4. called assembly message function](#_Toc52873038) 4

[Figure 5. called assembly value function](#_Toc52873038) 4

[Figure 6. Progression of early prototype](#_Toc52873038) 4

[Figure 7. Title and summary screens](#_Toc52873038) 5

[Figure 8. Scores and question count element](#_Toc52873038) 5

[Figure 9. Early C# method to handle input](#_Toc52873038) 6

[Figure 10. Translate C# to Assembly](#_Toc52873038) 7

This page is intentionally left blank.

Introduction

This report is a team project to design a game and implement it in the Intel x86/x64 Assembly language. The target audience will be the students and professor of CSCI-C335 Spring 2020 and any interested visitors to the project Github site. The project members hope to learn, apply, and demonstrate Intel Assembly programming skills obtained during the semester.

The game is named Metis 2020. It was inspired by a family game played by one of the team members during the holidays, and it is named for the Greek Goddess of Wisdom. Metis 2020 is an interactive, single player, multi-selection trivia game featuring the MIT license. It has been developed with the Unity framework and Visual Studio 2017+ using C# and C++ programming languages with in-line Assembly. Utilizing a modern graphical user interface, the player will answer approximately 15 randomly selected questions.

Project Members

The developers of this project are three computer science students of Indiana University Southeast with unique interests and skills.

|  |  |  |  |
| --- | --- | --- | --- |
| Position | Name | Major | Responsibilities |
| Developer and project manager | Bello, Junet | Computer Science | Responsible for adopting and managing the concept into a working product |
| Developer and Designer | Schneider, Amanda | Computer Science | Responsible for incorporating new features to the project |
| Developer and Quality Assurance | Lu, Yiliang | Computer Science | Responsible for documentation, unit testing, and quality control |

Figure 1. List of key personnel responsible for the completion of the project

Project Objectives

The project members are required to produce a computer game using primarily the Intel Assembly Language. In addition, the members need to present the project to the class audience, produce a written report that details the implementation, and a “Readme” document that explains the game rules to the end users.

Based on the requirements above, the members have agreed to the following specifications:

* Design a single player trivia game
* Determine the primary and secondary communication and collaboration methods
* Create a set of business rules that govern the logic of the game
* Create the game using programming languages to include the Intel x86/x64 Assembly Language, C#, and C++ in Visual Studio 2017+
* Utilize the Unity framework to simplify the development of graphical user interface and input
* Host the project on Github to facilitate team collaboration and source version control
* Share equal divisions of labor: tasks are assigned to each member evenly
* Control quality: any revisions must be reviewed and approved by another member before the changes are merged to the repository
* Present the project at IUS Student Conference Spring 2020

Problems Encountered

As the project progressed, the project members have encountered several issues. Some of these issues are addressed in the next paragraph. The last issue has not been fully resolved yet, but the members are seeking assistance from the overseeing faculty member.

1. Understand the scope of the project
2. Determine what and how to deliver
3. Translate more C# methods to ASM functions
4. Balance work and study commitments due to current events
5. Determine how to incorporate the assembly code given the limitations in the Unity framework.

Problems Solutions

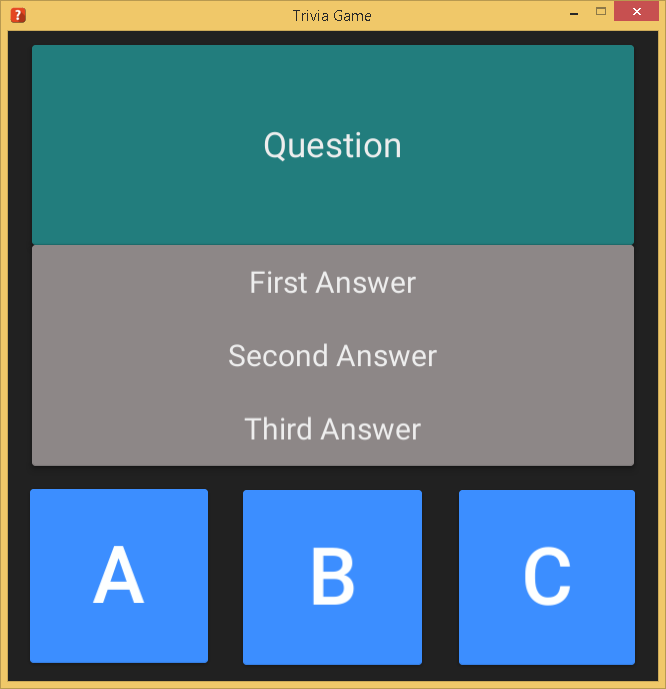
1. *Understand the scope of the project*

When the project was in the brainstorming phase, team members had some great ideas to include elements of role playing and player interaction with in-game objects such as opening doors. The members quickly realized that they needed to be realistic with what is achievable and what is not. Given the limited amount of time and competing courses in the semester, the members had to narrow the scope of the project to stay in the realm of practicality.

1. *Determine what and how to deliver*

It took a few weeks to finish the first prototype (Figure 2). While it looks simple, the team members had to invest much time to learn and use Unity, which has a steep learning curve.

The first prototype was created to prove that the project could indeed incorporate Assembly Language in a Unity project. Unity is mostly developed in .NET, and Unity projects cannot access any features that are not supported by the .NET libraries, such as direct access to the registers. The framework normally uses scripts to create additional functionalities, but it can also load *plug-ins* or code created outside of Unity. To get around the restrictions, the team wrote a *native plug-in*, which is a platform-specific native code library, to be used with Unity. A native plug-in can directly access features like hardware, OS calls, and third-party code libraries that would otherwise not be available in Unity. In this case, the native plug-in was written in C++ in order to call inline Assembly placeholder functions, which utilized *RAX*  in the following code snippets.



// UnmanagedCode.cpp

#include "pch.h"

#include "UnmanagedCode.h"

extern "C" int GetNumberAsm();

extern "C" char\* GetMessageAsm();

int GetTestNumber() {

return GetNumberAsm();

}

char\* GetTestMessage() {

return GetMessageAsm();

}

Figure 2. The first prototype

Figure 3. C++ function that calls inline assembly functions

; GetNumberAsm.asm

.code

GetNumberAsm proc

mov rax, 100

ret

GetNumberAsm endp

end

## **;GetMessageAsm.asm**

.data

message BYTE "Message from assembly", 0

.code

GetMessageAsm proc

mov rax, offset message

ret

GetMessageAsm endp

end

Figure 5. Placeholder assembly value function

Figure 4. Placeholder assembly message function

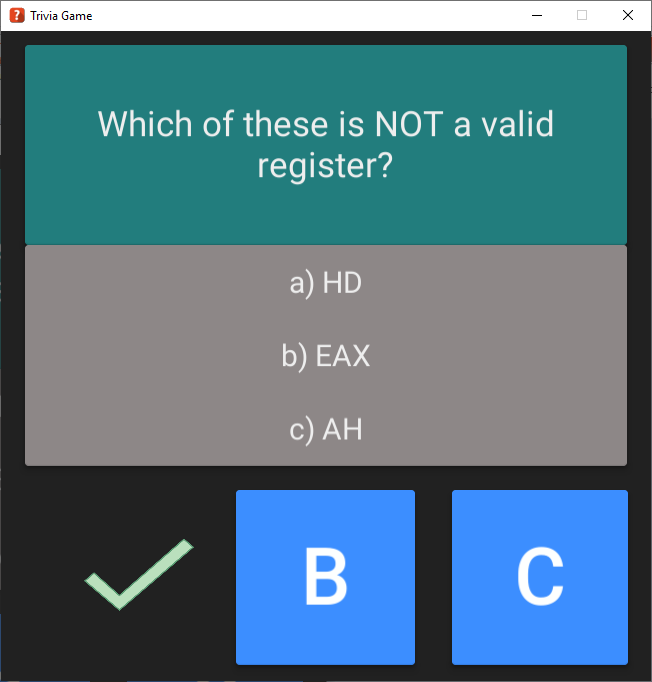
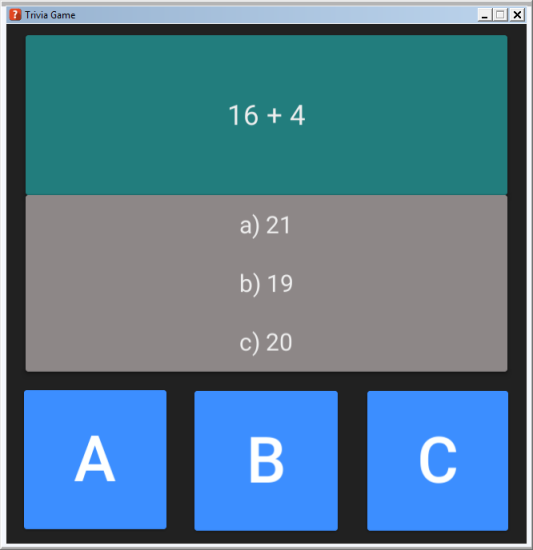


Figure 6. Progressions of the early prototypes

Later progress has replaced placeholder questions and answers with the actual questions and answers seen in the third screenshot although they are still hardcoded. The feature to read content through files was planned but has not been implemented.

Towards the end of the game development, the team added a title screen with a menu and a summary screen. Upon starting the game, users will land at the title screen. Using the left mouse button, two options are available on the menu of the title screen: clicking “Start” to play the game or clicking “Controls” to view a quick introduction of the game play.

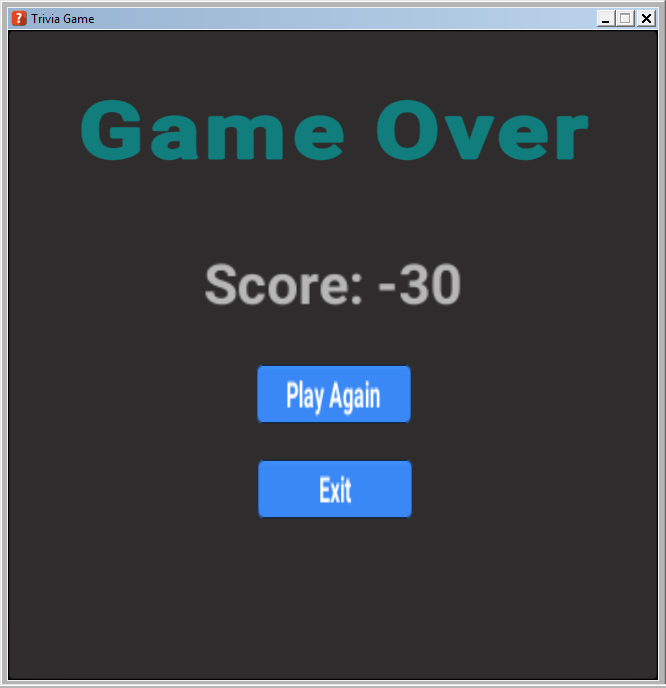
 

Figure 7. Title screen (left) and summary screen (right)

When playing the game, the players are presented with 15 questions selected randomly from a question pool one-by-one. Users can choose one of three answers. Each correct answer awards 10 points, and each incorrect answer deducts 5 points.

At the end of the 15 questions, a summary screen displays the total points earned or lost. Users can play again by clicking “Play Again’ or exit the game by clicking “Exit”.

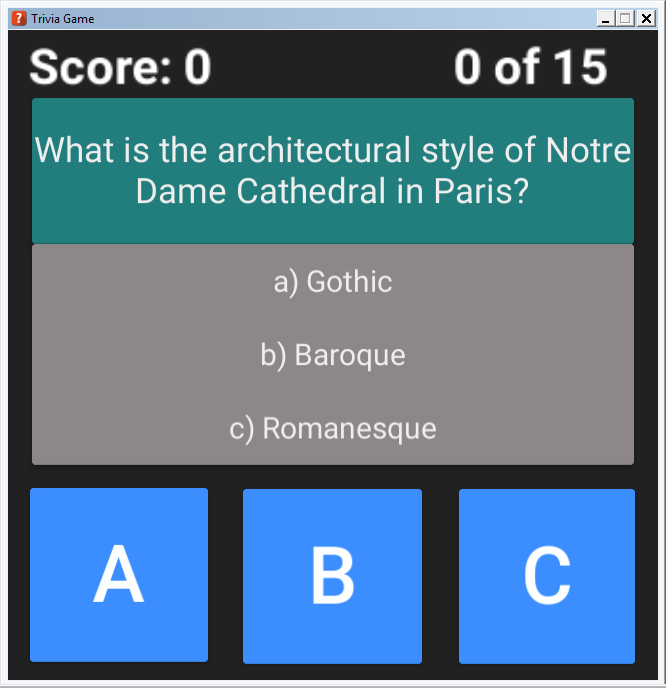
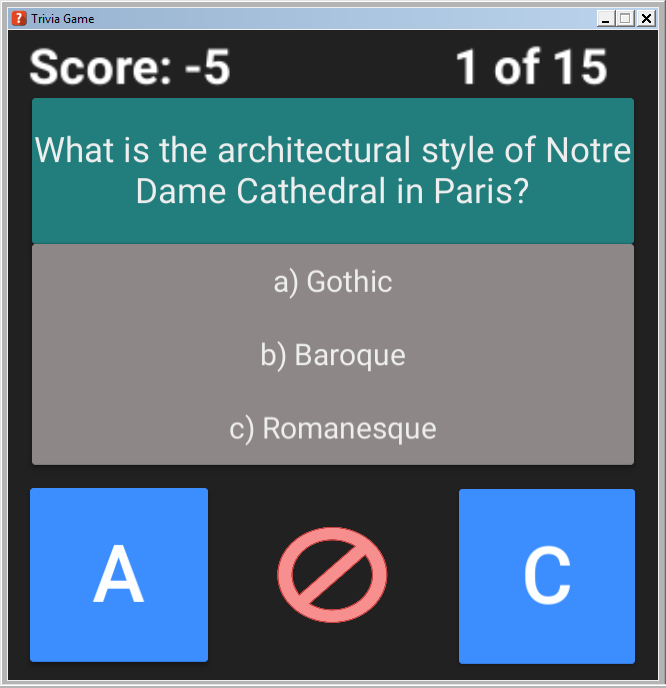
 

Figure 8. Score and question counter elements before answer (left) and post answer (right)

1. *Translate more C# methods to Assembly Language*

In the early weeks, the majority of the functional code was written in C#, for example:

/\*\*Questions are displayed randomly one at a time. Each of the possible answers associated with a \*question have a button assigned to it, and each button has a function that handles the logic to figure \*out if the answer is correct or not.

\*/

public void UserSelectA() {  
 if (CurrentQuestion.Answers[0].Result) {  
 FirstResponseImg.texture = CorrectTexture;  
 // Add score function. If correct add question score.  
 }  
 else {  
 FirstResponseImg.texture = WrongTexture;  
 // Add score function. If incorrect subtract half of question score.  
 }

Animator.SetTrigger("ButtonAClicked");  
 StartCoroutine(TransitionToNextQuestion());  
 }

public void UserSelectB() {  
 if (CurrentQuestion.Answers[1].Result) {  
 SecondResponseImg.texture = CorrectTexture;  
 // Add score function. If correct add question score.  
 }  
 else {  
 SecondResponseImg.texture = WrongTexture;  
 // Add score function. If incorrect subtract half of question score.  
 }

Animator.SetTrigger("ButtonBClicked");  
StartCoroutine(TransitionToNextQuestion());  
 }

public void UserSelectC() {  
 if (CurrentQuestion.Answers[2].Result) {  
 ThirdResponseImg.texture = CorrectTexture;  
 // Add score function. If correct add question score.  
 }  
 else {  
 ThirdResponseImg.texture = WrongTexture;  
 // Add score function. If incorrect subtract half of question score.  
 }

Animator.SetTrigger("ButtonCClicked");  
StartCoroutine(TransitionToNextQuestion());  
 }

Figure 9. Early C# method to handle input.

While many important methods are still written in C#, more and more functions are moved to the native plug-in and translated to Assembly.

For example: The GetTimeInSeconds() method which specifies the time interval between questions to be displayed.

// 1. C# gets triggered on initial load and calls C++ using the imported DLL:

Void Awake()

{

TimeBetweenQuestions = GetTimeInSeconds();

}

// 2. C++ calls Assembly:

int GetTimeInSeconds()

{

return GetTimeInSecondsAsm();

}

// 3. Assembly is where that actual function gets executed and returning the value needed:

.code

GetTimeInSecondsAsm proc

mov rax, 3

ret

GetTimeInSecondsAsm endp

End

Figure 10. Translate C# to Assembly.

Finally, Assembly returns the value 3 (number of seconds between questions) to C++, which then sends it to C# which communicates with `Unity` to assign the giving value to the next question event.

1. *Balance work and study commitments due to the current of events*

The on-going COVID-19 pandemic has temporarily changed how everyone studies and works in the United States. Many people are mostly confined to their homes, and school instructions are given remotely. Some may be ill, while some may have to work more or less. Daycares are closed, so families have to manage childcare with other responsibilities.

The members of this project team have also been impacted by current events. The disruptions have caused delays and inconveniences, but we are all coping and managing. We are determined to continue the project and meet the goals initially set.

1. *Determine how to incorporate the assembly code given limitations*

Previously, in section ii of “determining what and how to deliver”, the team decided to develop a native plug-in in order to incorporate the Assembly Language. About half way through the process, the team had to halt development temporarily because a question was raise about whether the plug-in method would meet the requirement of developing the game in Assembly.

The team consulted with the overseeing faculty member of the problem faced. The faculty member clarified that the team should develop the *main()* in the Assembly Language. As a result, the team had to conduct additional research to see if that was possible.

After many days of research, the team reached a conclusion that a solution of the main function written in the Assembly Language might not be feasible given the limitations of the Unity framework and the skill level of the members. While it may be common practice to use *main()* as the single code entry point for initializing the application, this method is not relevant in the Unity framework. According to senior Unity developer Joe Strout, the Unity framework is based on a component-based architecture approach. Components in Unity are classes that derive from MonoBehaviour. They have a number of magic methods called *messages*, which are found by name and invoked automatically by the Unity runtime. Each component is generally responsible for initializing itself, which can happen in any or all of several stages:

1. The object's public properties are set (de-serialized) with the values set in the Unity editor.
2. The object's Awake() method is invoked.
3. The object's Start() method is invoked.

In most cases, developers are advised to work on components. If there is some type of global game state that developers wish to initialize, they should make a MonoBehaviour and create an empty GameObject in the scene.

After consulting with the faculty member the second time and explaining the team research, the faculty member gave the team permission to proceed with deviation from the original guideline. The team was once again on the track to complete the project.

Future Work and Conclusion

The team members had originally envisioned a game with more features. As the project members slowly narrowed the scope and focus of the project, some of their wonderful ideas had to be dropped or replaced with more practical ones. For example, the team originally planned to record game playing time and include a weighted score feature in the game. Due to time constraints, neither feature made it into the public release. It is not the end of the project however. Once the semester is over, the team will actually have more time to add the missing features and additional content. Metis 2020 may be a pleasant addition for the Bello family on a game night this coming holiday season.

The team members have also noticed strengths and shortcomings while undertaking the project, but everyone kept working on the project and accomplished more than they ever believed was possible. Junet really stood out by taking the project lead and divided the project into manageable pieces; Amanda made tremendous improvements to the project and added new features one after another; Lu also made significant discoveries of system compatibilities and runtime errors as he tested and integrated Unity project objects with Visual Studio. Everyone had to learn new programming skills outside of the classroom, research ideas on unfamiliar subjects, and test time management skills repeatedly.

After executing the final phases of the project, the members have obtained a much deeper understanding of the Intel assembly language. Everyone has greatly enhanced their skill writing assembly code and incorporated the knowledge with other programming language. The team project will serve as an important lesson of time management and project management to all team members. The team hopes to apply the lessons learned to the members’ future endeavors. May the members succeed and be the next generation of programmers.

Credits and Copyrights

Copyright, 2020, Jbello0909

Intel is a trademark or registered trademark of Intel Corporation.

Microsoft, Windows, Visual Studio and DirectX are trademarks or registered trademarks of Microsoft Corporation.

“Unity", Unity logos, and other Unity trademarks are trademarks or registered trademarks of Unity Technologies or its affiliates in the U.S. and elsewhere. Other names or brands are trademarks of their respective owners.