Aureos: AR Project
Documentation

B. Ruiz Sabido Bryan F., B. Lizarraga Franco Mauro J., B. De la Cruz Ramos Carlos J. & B. Martinez Contreras Yeshua Javier. Universidad Politécnica de Yucatán, Tablaje Catastral 4448, Carretera Mérida-Tetiz. Km.4.5, 97357 Ucú, Yuc. Embedded Systems Engineering. Advanced Programming. October, 2018.

I. CODIFICATION STANDARD

STANDARD CODIFICATION IN UNITY

Unity uses C# as a base of its scripts, that's mean that unity doesn't have its own standard codification; therefore we are going to talk about C# standard codification; however Unity implement a mix of the two standard codification of C# that are pascalcase and camelcase, as well as C++ for getting build in Android and iOS. The next chart shows the conventions of C#/C++ used in unity

TYPE	C#/C++
class	PascalCase
constant	PascalCase
method	PascalCase
Namespace/pakage	PascalCase
propeties	PascalCase
parameter	camelCase
Local var	camelCase
interphase	PascalCase

This is a code in unity(example):

```
using UnityEngine;
using System.Collections;

public class DemoScript : MonoBehaviour {

    public Light myLight;

    void Update () {
    if (Input.GetKeyDown ("space")) {
            MyFunction ();
    }
    }

    void MyFunction () {
    myLight.enabled = !myLight.enabled;
```

```
        }
    }
```

C# Code Conventions

The coding conventions have the following objectives:
They create a consistent appearance in the code, so that readers can focus on the content, not the design.
They allow readers to understand the code more quickly by making assumptions based on previous experience.
They make it easy to copy, change and maintain the code.
They show the recommended procedures of C #.

Naming conventions
For short examples that do not include using directives, use namespace qualifications. If you know that a namespace is imported into a project by default, it is not necessary to complete the names of that namespace. Full names can be split after a period (.) If they are too long for a single line, as shown in the following example.

```
var currentPerformanceCounterCategory = new System.Diagnostics.
        PerformanceCounterCategory();
```

It is not necessary to change the names of objects that were created with the tools of the Visual Studio designer to match other guidelines.

Design conventions
A good design uses a format that highlights the structure of the code and makes the code easier to read. Microsoft samples and samples comply with the following conventions:
Use the default Code Editor Settings (automatic indent, 4 character indents, tabs saved as spaces).
Write only one instruction per line.
Write only one statement per line.
If the continuation lines are not automatically indented, do so with a tabulation (four spaces).
Add at least one blank line between the method and property definitions.
Use parentheses so that the clauses of an expression are obvious, as shown in the following code.

```
if ((val1 > val2) && (val1 > val3))
{
        // Take appropriate action.
}
```

Comments Conventions
Place the comment on a separate line, not at the end of a line of code.
Start the comment text with a capital letter.
End the comment text with a period.

Insert a space between the comment delimiter (//) and the comment text, as shown in the following example.

```
// The following declaration creates a query. It does not run
// the query.
```

Language conventions

The following sections describe the practices that the C # team follows to prepare samples and code samples.

String (Data types)

Use string interpolation to concatenate short strings, as shown in the following code.

```
string displayName = $"{nameList[n].LastName}, {nameList[n].FirstName}";
```

To append strings in loops, especially when working with large amounts of text, use a StringBuilder object.

```
var phrase = "lalalalalalalalalalalalalalalalalalalalalalalalalalalalalala";
var manyPhrases = new StringBuilder();
for (var i = 0; i < 10000; i++)
{
        manyPhrases.Append(phrase);
}
//Console.WriteLine("tra" + manyPhrases);
```

Local variables with implicit assignment of types

Use implicit types for local variables when the type of the variable is obvious from the right side of the assignment, or when the exact type is not important.

```
// When the type of a variable is clear from the context, use var
// in the declaration.
var var1 = "This is clearly a string.";
var var2 = 27;
var var3 = Convert.ToInt32(Console.ReadLine());
```

Do not use var when the type is not evident from the right side of the assignment.

```
// When the type of a variable is not clear from the context, use an
// explicit type.
int var4 = ExampleClass.ResultSoFar();
```

Do not trust the variable name to specify the type of the variable. It may not be correct.

```
// Naming the following variable inputInt is misleading.
// It is a string.
var inputInt = Console.ReadLine();
Console.WriteLine(inputInt);
```

Avoid using var instead of dynamic.

Use implicit types to determine the type of the loop variable in for and foreach loops.

In the following example, implicit types are used in a for statement.

```
var syllable = "ha";
var laugh = "";
for (var i = 0; i < 10; i++)
{
```

```
        laugh += syllable;
        Console.WriteLine(laugh);
}
```

Unsigned data type

In general, use int instead of unsigned types. The use of int is common throughout C #, and it is easier to interact with other libraries when int is used.

Matrices

Use concise syntax to initialize the arrays in the declaration line.

```
// Preferred syntax. Note that you cannot use var here instead of string[].
string[] vowels1 = { "a", "e", "i", "o", "u" };


// If you use explicit instantiation, you can use var.
var vowels2 = new string[] { "a", "e", "i", "o", "u" };

// If you specify an array size, you must initialize the elements one at a time.
var vowels3 = new string[5];
vowels3[0] = "a";
vowels3[1] = "e";
// And so on.
```

Delegates

Use concise syntax to create instances of a delegate type.

```
// First, in class Program, define the delegate type and a method that
// has a matching signature.

// Define the type.
public delegate void Del(string message);

// Define a method that has a matching signature.
public static void DelMethod(string str)
{
        Console.WriteLine("DelMethod argument: {0}", str);
}
// In the Main method, create an instance of Del.

// Preferred: Create an instance of Del by using condensed syntax.
Del exampleDel2 = DelMethod;

// The following declaration uses the full syntax.
Del exampleDel1 = new Del(DelMethod);
```

Try-catch and using instructions in exception handling

Use a try-catch statement in most cases of exception handling.

```
static string GetValueFromArray(string[] array, int index)
{
        try
        {
```

```
        return array[index];
        }
        catch (System.IndexOutOfRangeException ex)
        {
        Console.WriteLine("Index is out of range: {0}", index);
        throw;
        }
}
```

Simplify the code by using the C # using statement. If you have a try-finally statement in which the only code in the finally block is a call to the Dispose method, use a using statement instead.

```
// This try-finally statement only calls Dispose in the finally block.
Font font1 = new Font("Arial", 10.0f);
try
{
        byte charset = font1.GdiCharSet;
}
finally
{
        if (font1 != null)
        {
        ((IDisposable)font1).Dispose();
        }
}


// You can do the same thing with a using statement.
using (Font font2 = new Font("Arial", 10.0f))
{
        byte charset = font2.GdiCharSet;
}
```

New (Operator)

Use the concise form of object instance creation with implicit types, as shown in the following statement.

```
var instance1 = new ExampleClass();
```

The previous line is equivalent to the following statement.

```
ExampleClass instance2 = new ExampleClass();
```

Use object initializers to simplify the creation of objects.

```
// Object initializer.
var instance3 = new ExampleClass { Name = "Desktop", ID = 37414,
        Location = "Redmond", Age = 2.3 };

// Default constructor and assignment statements.
var instance4 = new ExampleClass();
instance4.Name = "Desktop";
instance4.ID = 37414;
instance4.Location = "Redmond";
instance4.Age = 2.3;
```

Event control

If you are defining an event handler that you do not need to remove later, use a lambda expression.

```
public Form2()
{
        // You can use a lambda expression to define an event handler.
        this.Click += (s, e) =>
        {
        MessageBox.Show(
        ((MouseEventArgs)e).Location.ToString());
        };
}
```

Static members

Call static members with the class name ClassName.StaticMember. This practice makes the code more readable by clarifying static access. Do not qualify a static member defined in a base class with the name of a derived class. While the code is compiled, its readability is confusing, and may be interrupted in the future if a static member with the same name is added to the derived class.

LINQ Queries

Use descriptive names for the query variables. In the following example, seattleCustomers is used for customers located in Seattle.

```
var seattleCustomers = from cust in customers
                       where cust.City == "Seattle"
                       select cust.Name;
```

Use aliases to ensure that the property names of anonymous types are correctly written with uppercase or lowercase, using the Pascal script.

```
var localDistributors =
        from customer in customers
        join distributor in distributors on customer.City equals distributor.City
        select new { Customer = customer, Distributor = distributor };
```

Change the name of the properties when they may be ambiguous in the result. For example, if the query returns a customer name and a distributor identifier, instead of leaving it as a Name and ID in the result, change its name to clarify that Name is the name of a customer and ID is the identifier of a distributor.

```
var localDistributors2 =
        from cust in customers
        join dist in distributors on cust.City equals dist.City
        select new { CustomerName = cust.Name, DistributorID = dist.ID };
```

Use implicit types in the declaration of query variables and interval variables.

```
var seattleCustomers = from cust in customers
                       where cust.City == "Seattle"
                       select cust.Name;
```

Align the query clauses under the from clause, as shown in the previous examples.

Use where clauses before other query clauses to ensure that subsequent query clauses operate on a reduced and filtered data set.

```
var seattleCustomers2 = from cust in customers
                        where cust.City == "Seattle"
```

```
                orderby cust.Name
                select cust;
```

Use several from clauses instead of a join clause to gain access to internal collections. For example, a collection of Student objects could each contain a set of test results. When the following query is executed, it returns each result greater than 90, in addition to the last name of the student who received the score.

```
// Use a compound from to access the inner sequence within each element.
var scoreQuery = from student in students
                from score in student.Scores
                where score > 90
                select new { Last = student.LastName, score };
```

II. METHODOLOGY

Kanban as our methodology

Aureos is the first group of visual development of the UPY that Works both Android and Apple platform for the creation of augmented reality on Unity software. It arises in the middle of a programming class in order to create an outstanding product for getting a high score. Our methodology to implement our products is based on Kanban methodology from Toyota that use techniques just in time, The main rules of Kanban are the following three:

1. Visualize the work: Kanban is specialized in visualizing the situation of each task; This is done through Post-it, which are some pieces of paper that can be added to a blackboard to divide tasks,

2. Determine the work limit: Kanban uses a blackboard as a task manager which is divided into sections with numbers under the name of each task, as one of the main ideas of Kanban is that the work in progress should be limited.

3. Measure the time to complete a task: it is practically the time it takes to complete each task, at that time called lead in time which is done since a request is made until the delivery of the product.

Since that methodology suits us to our necessities, we decided to implement this methodology that doesn't have many rules and promoting that each member of the group has the same position in the project, nobody is better that other, so in this way we don't have to meet us at a garage or in the house of someone to code our project instead we use a very useful platform named Meistertask that allows us to divide the tasks – the tasks are some papers used in Kanban to verify the activities that are done or not – and let us take any activity on the virtual board. Meistertask is based on Kanban board but with the difference that is a virtual board that has all the speciation that Kanban implements, Meistertask has three divisions: open (task that are not done), in progress (tasks that are working) and done (task that the team has finished). In each one of those divisions have the papers that allow us realize what activity is not done.

How do we work?

Each of the participants of the project write down on the Meistertask platform the specifications (tasks) that the programming professor wanted (client), then each of us pick one paper on the virtual board and start to work in it, the time estimated on

the progress in finishing a task depends on the difficulty of the task -. When someone finishes a task has to upload it in a repository, for this reason every member needs to know about some version control system but our group needs to know specifically git, the ones who has windows must install Gitbash and having an account on the online platform Github and link his/her account with Gitbash, lastly the file is uploaded and checked for the rest of the members. What are the advantages of working with Kanban methodology?

First of all, this methodology doesn't have many rules like crystal or scrum, and it is very suitable to work since house, and not only that as Kanban doesn't have a person who leads the group all the members can contribute in the same way (everyone has the same role) in the project and support each other if someone couldn't do the activity that he picked, how? Just drag the task that you chose and drop it in the part that says "open" on the MeisterTask dashboard.

Another advantage that did us to choose this methodology is the facility of working with MeisterTask, since it allows us to see in real time the activities in progress and the activities that have not been done, and as it is a free virtual platform for anyone, it allows us to work easily from our homes. Finally the meetings that are made in the Kanban methodology were feasible for us, because we could do them every Monday in the second hour of the programming class to talk about the difficulties we had, if each one is all right in his task, or someone dislike something about the project. in this last point if someone did not like something about the project a vote was made in the group and if the others didn't like, it was changed and if there were a majority of votes for not changing it, then it remained the same, in this way we kept the roles of the group so there are no disagreements and we all remain neutral.

Why Kanban and not Scrum?

Kanban is very easy to implement and has very few roles and basically the virtual board of Meistertask is perfect for us because we are a small group, on the other hand scrum has many more roles and also many more meetings that sometimes are not necessary, at least not for this project, and also has a role that we all wanted to avoid and was that someone lead us in the project. We think that since this project is almost 1 month to finish it and nobody was an expert in augmented reality issue, let alone know how to use Unity and Git, we didn't need someone who works like a scrum master, but everyone learns and can contribute something to the team, also the scrum master would have a lot of work to try to fix the mistakes of others or try to do what some member of the group was supposed to do.

Roles

As a single developer team, we were divided in different areas of development. The list is below:

Design Team
Provide sketchs, concept arts, visualizations and render of the Art behind the targets.

Bryan Ruiz & Jafet De la Cruz

Android Developer Team
Build and fix bugs in the Android Version of Aureos.

Yeshua Martinez & Mauro Lizarraga

iOS Developer
Build the iOS version for Apple devices.

Bryan F. Ruiz Sabido

Unity Developer Team
Provide the iterations between Vuforia ann the graphic engine.

Mauro Lizarraga, Yeshua Martinez & Bryan Ruiz