Read me

Question2.

I used NGUI 3.5 to build the calculator and the wizards.

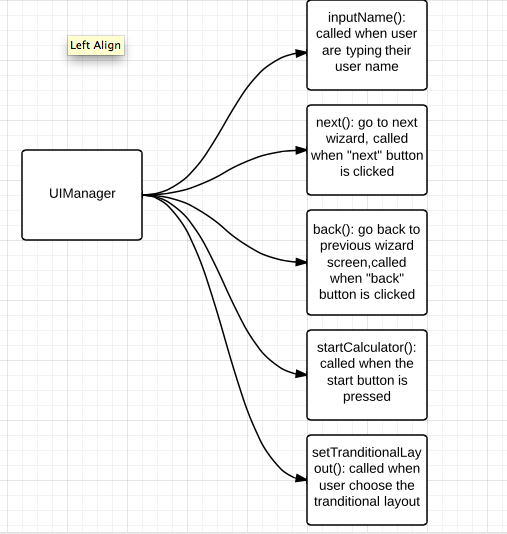
I used MVC design pattern that separated the UI side code and logic code.

This will make the code’s structure cleaner.

There is a UIManager class which in charge of all the UI logic work.

Inside UIManager, there are different functions for different buttons.

All functions are called through NGUI’s delegate.



All the wizards are in one array and all different calculator layouts are in a different array. If we want to add new wizards we just need to drag them into the wizards array and all the “next” “back” functions will work.

If we want to add new calculator layout, just need to make sure the layout game object’s name contains the keyword and drag them into the layout array. Then add one more state in the startCalculator() function.

The calculator logic is in the “Calculator” class. There is a string variable called “strPutKeyCode”. After attach this script on each of the calculator buttons we can identify the key code by ourselves. This made it very easy to add new calculator buttons. After pressed button with the key code of “=”. The script will execute the actual calculation.

I used array a lot in this demo. Although it is not as efficient as dictionary when we need to search for something. But the good thing is in unity array could have a dynamic size in the editor. So it is very convenient to use. Also because the searching algorithm is not used a lot in this app, array is better.

For the Unicode part, I didn’t find the Unicode checkbox in NGUI3.5, I might need to import fonts to make it support different languages. If you have any better solutions please let me know.

Question3.

Authoritative server is used a lot in online games.

The server is running the actual game logic and the client is more like a “controller”.

Client sends serve different request information and server does the processing work.

After process server can send the result back to clients.

The server always has the latest information.

And the server is the one who decide what information to give to the client.

Question4.

What is doing well:

Using array to store different movie clips is convenient because array’s size in unity could change dynamically, it made the UI really easy to extend. The efficiency of using an array is also not bad.

What could be better:

Shouldn’t do the UI code inside Fixed update function. Fixed update is supposed to do all the physics calculations. The time period between each calls is always the same.

Having so many codes in update or fixed update is not efficient.

Could change the if statements in UpdatePegState functions into switch case.

The movie clips in arrays could be changed into sprite animations. That could give more control and do not have problems when compile this application onto mobile devices.

Could load all the art assets through Resources.load(), because this is less memory consuming than making them into a variable.

Possible problems:

Movie clip is not working on mobile device, it also have a lot limitations, might need to change it into sprite animation.

How I would write it:

I will all use C# events instead calling the update every frame.

The callback functions will be called when player’s state changed.

Pseudo code:

**public** class PlayerStatus **:** MonoBehaviour

{

//function to display UI

public void showUI()

{

**for (**int i **=** 0**;** i **<** mCurrentPegArray**.**Length**;** i**++)**

**{**

mCurrentPegArray**[**i**].**ShowInactive**();**

**}**

**if (**mCurrentPegArray **!= null)**

**{**

**for (**int i **=** 0**;** i **<** mCurrentPegArray**.**Length**;** i**++)**

**{**

**if (**mCurrentValue **>** i**)**

**{**

mCurrentPegArray**[**i**].**ShowActive**();**

**}**

**}**

**}**

}

//all the call back functions

//add to C# event and called when the player state changes

public void onHealthChanged()

{

mCurrentPegArray **=** mHealthPointPegs**;**

**showUI();**

}

public void onManaChanged()

{

mCurrentPegArray **=** mManaPoints**;**

**showUI();**

}

public void onFortitudeChanged()

{

mCurrentPegArray **=** mFortitudePointPegs**;**

**showUI();**

}

public void onEnergyChanged()

{

mCurrentPegArray **=** mEnergyPointPegs**;**

**showUI();**

}

}