# CIS568 Game Design Practicum
# ASSIGNMENT #3: INTRO TO AUGMENTED REALITY

**Due: Monday, Feb. 10, 2018**
*(please submit to Canvas by midnight)*

## Assignment Overview:

The goal of this assignment is to enable you to develop a mobile augmented reality app in Unity. This assignment also covers cross-platform networking and using data from mobile sensors.

In this assignment, your goal will be to create an augmented reality paint ball game using Unity. You will be using a phone as a paint ball launcher and the computer display as a canvas.

The assignment consists of two parts. In the first part, you'll be using Photon, a cross-platform networking plugin, to connect your phone and computer; you'll be able to rotate a cube on the computer display using the gyro input from your phone. For the second part, you'll be able to create a paint ball mini game where you can use your phone as the paint ball launcher and your computer display screen as the canvas. The paint ball mini game uses Photon for the network connection between the phone and PC, Vuforia for the augmented reality part, and the data from phone's gyroscope to keep the phone orientation in sync when the target is not in the camera view. The players will lower their phones to aim, then swipe their finger on the phone screen to launch paint balls, trying to hit the canvas on the PC screen.

## Assignment Details:

***There are a number of things that need to be set up in Unity before you can begin coding. As a result, please begin this assignment by reading the document "Setup of Vuforia 7 in Unity 2017.3.pdf", which can be downloaded from Canvas. You can also find more details in Part 5 of the assignment (Appendix).***

### Part 1 – Rotating a Cube (40pts)

In this part of the assignment, you will use Photon to connect the phone and PC, and use data from the phone's gyro sensor to control a cube on the PC screen. In this part you will need to do the following:

a) We have already imported the Photon and Vuforia plugins for you into the project framework, but you are welcome to try importing them yourself, or update them if there are newer versions available. Before beginning, you will need to get a Photon app id (please refer to https://doc.photonengine.com/en/realtime/current/getting-started/obtain-your-app-id ), and enter it into **PhotonServerSettings** at *Assets> Photon Unity Networking > Resources*.

b) You'll need to work on **Cube_PC** and **Cube_Mobile** scenes for this part. Complete the cross-platform connection between phone and computer, the **Minimal_Default** is just a placeholder for setting up the deploy process.

In the *Scripts* folder, refer to **PCNetwork_Cube.cs** and the "Networking with Photon" section of the document, finish **MobileNetwork_Cube.cs**. Build and run **Cube_Mobile**

scene on your phone after running **Cube_PC** scene on your computer (using appropriate build settings), and make sure there is a "Joined" message shown at the upper left corner of your phone.

c) Remove the cube object from **Cube_PC** scene and use **PhotonNetwork.Instantiate()** on **MobileNetwork_Cube.cs** to create the same cube on the mobile client once the room has been joined. Remember to add **PhotonView** to the prefab and add **Transform** to the observed components.

d) Rotate the cube using the gyroscope data from the phone obtained in **GyroController.cs**. You can refer to "Using Gyroscope in Unity" section below. If your phone is in a similar orientation to the cube when lthe app is launched, the cube on the PC screen should basically match the phone orientation when you rotate your phone.

**Part 2 – AR Paint Ball Game (40pts)**

Create your Augmented Reality paint splat game using Unity, the gyroscope input, Photon and Vuforia (see "Using Vuforia AR in Unity" section) with the following features:

- Display an AR marker on the computer display screen (i.e. which will become the "canvas" for the paint splats) so the phone can determine its position and orientation with respect to the display screen (using the Vuforia);
- Once the AR marker has been detected, the user can then lower their phone to aim at the canvas and swipe the phone screen to launch a paint ball;
- The paint balls should be subject to gravity and physics such that when the player raises their phone vertically again they should be able to see the paint balls that were launched previously flying through the air;
- When the paint balls hit the PC screen "canvas", colored paint splatters should be created on the display screen at the point of impact (i.e. collision).
- Since the setting up of everyone's screen and phone is different, you might need to change the size and/or location of Image Target.

There are some resources provided in the homework base project that you can use. Using these resources, create a basic paint splat game by following the steps below:

a) You will need to modify the **PaintBall_Mobile** and **PaintBall_PC** projects. Although the Unity scenes have already been set up for you, you will need to copy some code from **MobileNetwork_Cube.cs** to **MobileNetwork.cs** and make sure the connection is working. You'll also need to obtain an app key for Vuforia and enter it at the **"App License Key"** field of **ARCamera** object.

When you use your phone camera to look at the AR target image on PC screen, a target mark should appear on it.

b) Implement the **TargetBehavior.cs** script which monitors the tracking state of Vuforia and switches the control of the ARCamera between gyroscope and Vuforia. You can try adding a button to disable the Vuforia input to see if your gyroscope input is working correctly. Also, if you like, you can swap out the Image Target's image with one of your

own choosing. Just go to "Image Target Behaviour" and "Add Target". Then follow the prompts on how to do it.

c) In **MobileShooter.cs**, implement **ShootBall()** so you can shoot a paint ball by swiping your finger on the touchscreen. You will need to use Photon's RPC function call to set the velocity, color and ownership of the paint ball.

At this point, the basics of the paint ball game prototype should be working. However, this is just the start.

**Part 3 – Improvements (20-40pts, up to 20pts extra)**

Build upon the paint ball prototype you created in Part 2, adding **at least one** of the following features:

- The prototype you have created is not exactly a game at this point. As a result, try adding some additional rules and a goal to make the prototype more "game" like.
    - You can even totally change the core mechanic, as long as there are AR features, networking and mobile sensors involved - but be sure to make a backup of your part 2 implementation if you decide to do that.
- Collaborate with others. Want to share your canvas? Try enabling multiple phones to connect together, assigning each phone a specific color or some special effect. You can even make this game a snowball fight if you implement both this and the idea above.
- The gyroscope is not always accurate, and it can only track changes in orientation. Try improving the accuracy by using additional sensors to make the control better. For example, you can use the compass to correct gyroscope errors by reducing the drift around the y-axis, and/or you can use the accelerometer (i.e. which returns the gravity vector when the phone is not moving) to update the position of the camera when Vuforia cannot find the target.
- In the current implementation paint balls are launched with a constant velocity. Try improving this by changing the initial speed and direction of paint balls based on the speed of the player's swipe. You can also make the control like a slingshot.
- The prototype could use a little polishing and there can be problems with the z-order of the splatter textures. Try improving the look, feel and sound of the prototype.
- Integrate Google ARCore or Apple ARKit functionality into this project.
    - These provide an alternative way of computing the camera position and orientation. Feel free to experiment.
    - As a starting point, you will find the documents "CIS568-Integrate ARCore" and/or "CIS568-Integrate ARKit" on the CIS568 Canvas site.

**Part 4 – Submission**

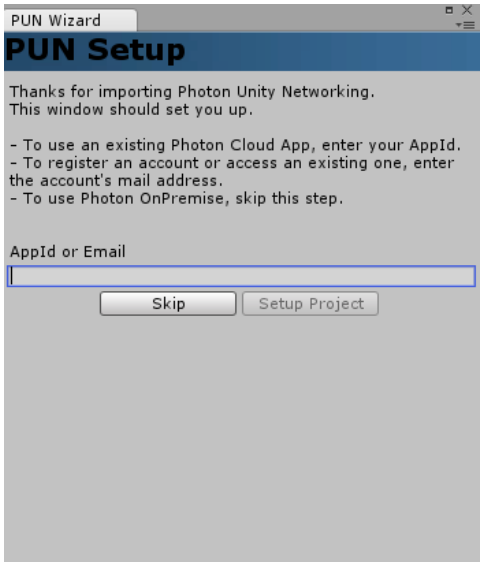
Have your project files, executables (for both PC and phone if applicable), a readme file and a short video (no .avi) in a zip file, upload it to Canvas by the deadline.

## Part 5 – Appendix

### 1. Networking with Photon

You will be using Photon to establish a connection between the phone and the PC computer. Photon is a cross platform multiplayer plugin for Unity, and it is surprisingly easy to use.

In you project, you first need to go to "Window=>Photon Unity Networking=>PUN Wizard". Then click the "Setup project". A dialogue as below will pop up asking for an AppId for Photon. If you don't already have one, you can get one for free using an email address. Then click "Setup Project" and close the dialogue.




The simplest way to make a connection between the phone and the PC using Photon is to use the "lobby" feature. To do this, use the PC as a "Master Client", or host, to create a room in the lobby which is unique to the AppId, then your phone can simply join the room.

In Assets/Photon Unity Networking/Resources/PhotonServerSettings, enable "Auto-Join Lobby" so the application will join the lobby upon launch.

You will use one scene for the PC and a different scene for phone. Like you did before, you'll need a global GameObject in these scenes. To establish a connection, you can attach a script like the one below to the global object of the PC scene:

```
public class PCNetwork : Photon.PunBehaviour
{
   void Start()
   {
      PhotonNetwork.ConnectUsingSettings("0.1");
   }
   void OnGUI()
   {
      GUILayout.Label(PhotonNetwork.connectionStateDetailed.ToString());
```

```
    }
    public override void OnJoinedLobby()
    {
        // Create a room for networking gameplay
        PhotonNetwork.CreateRoom(null);
    }
    public override void OnPhotonJoinRoomFailed(object[] codeAndMsg)
    {
        base.OnPhotonJoinRoomFailed(codeAndMsg);
    }
    public override void OnCreatedRoom()
    {
        base.OnCreatedRoom();
    }
}
```

And in the mobile scene, the code is similar, except that you need PhotonNetwork.JoinRandomRoom() to join the unique room at the lobby under your app id. And you will need to do something at OnJoinedRoom() for this assignment.

The above describes how to enable a Photon connection between the phone and the PC. When you are actually building the project, you should apply different settings for different scenes according to the type of your device (for example **PC, Mac & Linux Standalone** for your computer and **Android** for your phone).

When you run your PC client first and then run your mobile client, if everything is done properly, there should be a "Joined" text message shown on the top-left corner of your phone and your computer. This indicates the connection between the two clients has been successfully established.

Once the room has been joined, you can use **PhotonNetwork.Instantiate()** to spawn objects over the network. In order to decide which data to sync between clients, you need a PhotonView component on the prefab to instantiate. You can drag components (such **Transform**) to the **Observed Components** tab of the PhotonView so the component data will be atomically synced to the network from the owning client.

If you want more information on how these parts piece together, watch this video from a tutorial series on YouTube about using Photon in Unity to create multiplayer games. It goes through the information above in a bit more detail.
(https://www.youtube.com/watch?v=mLAilMkRxYU&index=4&list=PLAkOwsOxpAvpYLAJCd3v295o26Z4WZWc2 )

## 2. Using Gyroscope in Unity

With Unity, you can access data from mobile sensors easily. In this assignment you will be using the gyroscope.

You can use **Input.gyro** to access the gyroscope data. It is an instance of the class **Gyroscope**.

Before you use the data from the gyro to rotate the cube or camera, you need the following code to enable gyroscope input and set the update interval:

```
Gyroscope gyro;
…
  void InitGyro()
  {
        gyro = Input.gyro;
        gyro.enabled = true;
        gyro.updateInterval = 0.01f;
  }
```

Once setup and activated, the gyro class will automatically update its data from the sensor input. Now, you can access the following members:

gyro.attitude: the attitude (ie, orientation in space) of the device.

gyro.gravity: the gravity acceleration vector expressed in the device's reference frame.

gyro.rotationRate: rotation rate as measured by the device's gyroscope.

gyro.rotationRateUnbiased: unbiased rotation rate as measured by the device's gyroscope.

gyro.updateInterval: sets or retrieves gyroscope interval in seconds.

gyro.userAcceleration: the acceleration that the user is giving to the device.

**gyro.attitude** returns a quaternion which represents the current orientation in the device's reference frame with respect to the world. However, although Unity allows you use the same code on difference devices, **gyro.attitude** of difference device types (i.e. Android vs iPhone) may have different coordinate systems and you may need to manually convert it to be consistent with the Unity coordinate system.

When using an Android phone, **gyro.attitude** is associated with a right-handed coordinate system while Unity uses a left-handed coordinate. We can use the following code to convert the representation of orientation from a right handed coordinate system to a left handed coordinate system :

```
Quaternion ConvertRotation(Quaternion q)
{
    return new Quaternion(q.x, q.y, -q.z, -q.w);
}
```

To keep the rotation in the virtual world in sync, we save a reference rotation *R* for the virtual world corresponding to the initial rotation of the rotating cube for the first part of the assignment,

or the rotation of the ARCamera when Vuforia finds the target in the second part. In addition, we also save the corresponding gyro data $R_g$ from **gyro.attitude**. Givene new gyro rotation $R'_g$, we can get the local rotation difference $\Delta R$ and new virtual object rotation $R'$ by:

$$\Delta R = \left(R_g\right)^{-1} R'_g$$

$$R' = R\Delta R$$

### 3. Using Vuforia AR in Unity

In the scene, you can put an ARCamera and a ImageTarget from Vuforia folder into your scene. After you set the **Database** and **ImageTarget** in the property of the ImageTarget and the license key in ARCamera, you can put any 3D geometry above the ImageTarget object. Once you run the application, it will automatically use your device camera to sense the marker, and you should be able to see your geometry above the marker once successfully tracked the target. See https://www.sitepoint.com/how-to-build-an-ar-android-app-with-vuforia-and-unity/ for a more detailed tutorial.

Vuforia automatically adjusts the location, orientation and projection of the ARCamera by the calculated position and orientation based on the view of the marker in the device camera. In the final part of the assignment, you will need to be able to keep the orientation in sync while putting down the device and swiping on the phone screen to launch the paintball. You'll need to switch the controller of the virtual camera between Vuforia and the gyroscope.

To do this, you can monitor the tracking status of Vuforia in your ImageTarget object by adding a script with **ITrackableEventHandler** interface and monitoring the **OnTrackableStateChanged()** function. You can create your code based on the **DefaultTrackableEventHandler** class in **Scripts** folder of Vuforia, which is loaded as a component of ImageTarget prefab by default. When the state goes to **Tracked**, it means the phone can see the marker directly.

When the marker is not detected (i.e. tracked), use gyroscope data to manually control the rotation of the ARCamera. In this assignment, gyroscope control can be toggled with **GyroController.Paused** property.

Now you are good to go with your paint ball AR game. Happy coding!