# Optimising performance

* You should avoid calling the following functions as much as possible inside of Update()
  + GameObject.FindWithTag()
  + GetComponent()
  + Transform.position
* Try to call them once to assign a variable in Start() and use that variable Update(). The exception is transform.position, which you may need to call each frame in Update() as it only gives the position on the current frame. To mitigate this, make sure you only call it once in Update and save it in a Vector3
* Avoid using foreach loops, just use regular for loops instead. The overheads of creating new instances GameObjects within foreach loops makes them expensive in unity

# Pathfinding

* Before pathfinding, a pathfinding graph of the environment needs to be generated, do this by calling Pathfinding.Pathfinder.UpdatePathfindingGraph(). You should call this method at the start of the round and whenever an obstacle is moved significantly by the player (if we’re going to allow players to move stuff during the game)
* To find a path around obstacles in the environment, call Pathfinding.Pathfinder.GetPath(Vector3 source, Vector3 target)
* For an object to be registered as an obstacle to pathfind around, it needs a collider, must have its layer set to ObstaclesOnly and must a PathfindingObstacleScript attached

Device Portal (for deployment from lab machines and extra tools such as Live Capture)  
To build from the lab machines, you must first create an AppPackage (I).  
Then, to deploy, you currently need to connect to the Hololens via the Device Portal (II)

I - Creating an App Package

1. Build the app as usual from Unity, as if you were going to deploy it directly via Visual Studio.
2. Open the .sln solution file in VS and wait for it to load.
3. Right click on the ProjectName (Universal Windows) item in the *Solution Explorer,* and select *Publish > Create App Packages...*
4. For *Select distribution method* choose "Sideloading"
5. Keep the selected certificate.
6. For *Select and configure packages,* select *ARM64 Release* (or whichever deployment type you require, so long as it's ARM64), as well as specifying a version number and file location. The version number must not be the same as one for the same project which has already been deployed to the device (i.e. increment the version number on each deploy)
7. Click *Create*

II - Connecting to Device Portal

1. Connect the Hololens 2 to the machine via USB.
2. On the Hololens, navigate to *Settings* > *Update & Security* > *For Developers*, and scroll down until you find the Device Portal settings, which needs to be turned on.
3. Below these settings there *should* be an "Ethernet" address, which you can navigate to in your computer’s browser. If it says it is unsafe, just proceed anyway (this option is sometimes a bit hidden but it should be on the page somewhere)
4. On first launch you will be prompted to enter a pin (put on hololens to view), and create a username + password, which will be used to authenticate you when accessing the device portal for that headset on different devices.
5. Follow [these instructions](https://learn.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/using-the-windows-device-portal#sideloading-applications) to load your AppPackage application and certificate to the Hololens.
6. You can then find your app in the Hololens' app tray.
7. Device Portal can also be used to get a live capture of the Hololens view.

# QRCodeWatcher

Usage:

* Ensure that the QRCodeWatcher script is attached to a persistent game object, such as the MainCamera
* QRCodeWatcher has a public method, getCurrentList() which will return an Immutable List of the current QRCode objects that it can see, as well as the GameObjects which have been placed on top of them.

Notes:

* You must request permissions in order to track QR codes, which can be done by calling QRCodeWatcherAccessStatus status = await QRCodeWatcher.RequestAccessAsync();
  + The “await” keywork makes this request asynchronously and so it **won’t** block on this line until the user has accepted permission requests, but rather the code will continue to run.
  + This has consequences, such as initially the QRCodeWatcher commands may not succeed and similarly QRCodeWatcher.IsSupported() may return false, that is until the user has actually accepted these permissions. Make sure to check that they have accepted these, and if not display a message prompting them again, or something to that degree
  + If the permissions were accepted but IsSupported() is still returning false, then you may need to terminate the app (in settings) and open it again.
* Once permissions have been accepted, you can instantiate a new QRCodeWatcher object
* QRCodeWatcher keeps a list of QR codes that it has seen recently, QRCodeWatcher.GetList()
* QRCodeWatcher includes four events: Added, Removed, Updated, EnumerationCompleted
  + Added – event called whenever a QR code is added to its list (probably because a new one comes into view
  + Removed – event called when a QR code is removed from its list (I don’t think this is strictly linked to when the code goes out of view, it might instead be when its list of QR codes gets full, or something similar)
  + Updated – even called when the position for a QR code in the list is updated. I think this would be for example if the same code is shown in a different position, but not entirely sure.
  + EnumerationCompleted – event called when the enumeration of a QR has been completed, I think this could be when the QR has been read/processed and the data can be extracted from it
* To create handler methods for these events you need to create `private void` methods, with any sensible name, which need to have the same parameters as the ones on [this](https://github.com/microsoft/MixedReality-QRCode-Sample/blob/main/Sample/Assets/Scripts/QRCodesManager.cs) page (scroll to bottom and see “QRCodeWatcher\_Added(…)” for example)
  + Then, once these methods are created, you can add them as event handlers to your instantiated QRCodeWatcher object via `<QRCodeWatcherObjectName>.Added += QRCodeWatcher\_Added;` for example, which can also be seen in the “SetupQRTracking(…)” method on the github page referenced above.
* Call `<QRCodeWatcherObjectName>.Start()` to begin the QR watcher. (all of the above, including this, should probably be done in GameObject Start() method)
* Then, you can extract QR data either in the handler methods, or in Update,
  + QR Position in 3D can be extracted in handler methods using:

|  |
| --- |
| Pose position; SpatialGraphNode.FromStaticNodeId(args.Code.SpatialGraphNodeId).TryLocate(FrameTime.OnUpdate, out position);  Vector3 pos3D = position.position; |

* + - In this example, the `args` argument is the same as the one in the example event handlers shown in the GitHub page linked above
  + QR Data can be extracted (within handler methods) as above, using:

|  |
| --- |
| string data = args.Code.Data |