Relatório Final

Identificação do estágio

Ano Letivo	3	**Semestre:**	2	**Estágio nº**	4718
Número de Aluno	60288				
Nome Aluno	Guilherme Ribeiro Figueira				
Email	gr.figueira@campus.fct.unl.pt				
Título do estágio	GLARTEK em HoloLens - Futuro Unicórnio Português de Realidade Aumentada				
Nome da Empresa	Glartek				
Coordenador	Ricardo Monteiro				
Email	ricardo.monteiro@glartek.com				
Orientador do DI	Fernando Birra				
Email	fpb@fct.unl.pt				

Objetivos

O estágio teve como objetivo o desenvolvimento de novas funcionalidades relacionadas com a visualização de modelos 3D na componente *HoloLens* do projeto ***Glarvision***.

No início do estágio tive como função desenvolver testes de integração para o mesmo projeto, mas essa fase do estágio foi mais com o propósito de me familiarizar com o código do projeto para facilitar o início do desenvolvimento das funcionalidades relacionadas com os modelos 3D.

Descrição técnica e seu contexto

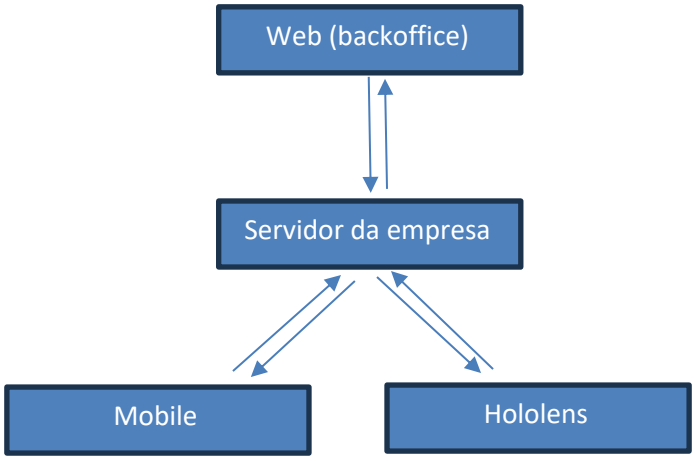
Arquitetura do projeto

Antes de descrever o meu trabalho, é necessário perceber a arquitetura do projeto ***Glarvision***.

Este projeto tem as seguintes componentes:

- Parte Web (*backoffice*);
- Mobile;
- HoloLens;

Sendo que a parte Web funciona como o *backoffice* do projeto e as componentes Mobile e HoloLens devem ter as mesmas funcionalidades entre si, mas em plataformas diferentes.



Durante todo o meu estágio, trabalhei apenas na componente *HoloLens,* mas tive contacto com a equipa de Web para tomar decisões sobre a interação entre a componente Web/servidor e HoloLens.

Testes de integração

Para os testes de integração, tive de escrever código que herda uma interface de testes de integração `ZenjectIntegrationTestFixture.cs` para *Zenject*, que é uma *framework* de injeção de dependências. Como as classes que estou a testar utilizam essa *framework*, tenho também de escrever os testes com a mesma.

Para cada classe que quero testar, devo criar uma classe de testes com um código inicial semelhante ao exemplo a seguir,

```
public class LoginMenuUITest : ZenjectIntegrationTestFixture
    {
        private Mock<IUILoginMenu.Settings> _settings;
        private Mock<IQRLoginController> _qrLogin;
        private Mock<IManualLogin> _manualLogin;
        private Mock<IGlobalEvents> _globalEvents;

        [Inject]
        private IUILoginMenu _loginMenu;
        private void CommonInstall()
```

Aqui estou a criar *Mocks* (simulação da classe) das dependências da classe a ser testada e injeto a implementação da interface a ser testada.

De seguida, um teste terá um formato parecido ao exemplo seguinte,

```
[UnityTest]
        public IEnumerator ConstructorTest1()
        {
            CommonInstall();

            Assert.False(_settings.Object.Self.activeSelf);
            Assert.AreEqual(Vector3.zero,
_settings.Object.Self.transform.localPosition);
            Assert.AreEqual(Quaternion.identity,
_settings.Object.Self.transform.localRotation);
            Assert.AreEqual(Vector3.one,
_settings.Object.Self.transform.localScale);

            Assert.True(_settings.Object.TopObject.activeSelf);
            Assert.AreEqual(Vector3.zero,
_settings.Object.TopObject.transform.localPosition);
            Assert.AreEqual(Quaternion.identity,
_settings.Object.TopObject.transform.localRotation);
            Assert.AreEqual(Vector3.one,
_settings.Object.TopObject.transform.localScale);

            yield return null;
        }
```

Modelos 3D

A funcionalidade de visualização e interação com modelos 3D tem o propósito de facilitar a execução de tarefas por parte de um trabalhador industrial, principalmente em casos de assemblagem. Ao visualizar um modelo 3D relacionado com uma dada tarefa, o utilizador consegue ter uma melhor perceção da tarefa que necessita ser executada do que apenas com uma imagem ou um vídeo.

Tendo este contexto em conta, o projeto *Glarvision* deve suportar:

- Upload de modelos 3D de vários formatos para a plataforma web (Componente Web);
- Descarregar os modelos 3D nos *HoloLens*;
- Instanciar e manipular o modelo 3D;
- Caso o modelo tenha uma animação embebida, permitir a reprodução dessa animação;
- Esconder / Mostrar camadas do modelo individualmente.

Apesar do objetivo ser que o projeto suporte vários tipos de ficheiros 3D, após fazer alguma pesquisa e ter discutido com o meu coordenador, chegamos à conclusão que seria benéfico os HoloLens apenas lidarem com modelos de formato GLTF / GLB, ou seja, os modelos seriam convertidos para esse

formato na parte do servidor, poupando assim recursos aos HoloLens. Isto porque, suportar a instanciação de modelos de vários formatos em runtime resultaria num aumento significativo de bugs durante o resto do desenvolvimento, pois poderia haver partes do programa que funcionariam para alguns formatos, mas não para outros. Essa conversão ainda não foi desenvolvida, mas já está decidido que será. Logo, neste momento o projeto só suporta ficheiros com formato GLTF/GLB.

Para descarregar os ficheiros utilizo um *wrapper* de *GraphQL* desenvolvido pela empresa para descarregar o ficheiro para uma cache local.

Quando o utilizador abre um painel que contém modelos 3D, estes começam a ser descarregados automaticamente para cache, e enquanto descarregam há um *loading* que vai de 0 a 100% como se pode observar na **Figura 1**.

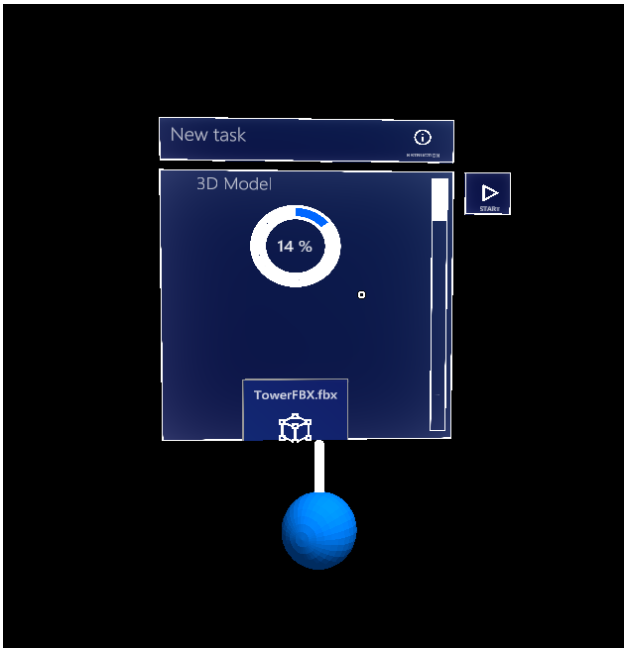


Figura 1 - Loading ao descarregar ficheiro

Ao terminar de ser descarregado este passa a ser um botão que, ao ser pressionado, instancía o modelo e um painel de definições relacionadas com o modelo. Este botão tem ainda o nome do ficheiro por cima dele. Este estado do botão encontra-se na **Figura 2**.

Figura 2 - Botão do modelo

Inicialmente tínhamos planeado ter uma thumbnail para cada modelo por cima do botão, mas a ideia acabou por ser adiada pelo facto de tal imagem ter de ser gerada primeiro. Gerar essa imagem no Unity seria demasiado dispendioso, principalmente ao correr o programa nos HoloLens que são computadores fracos no que conta a poder de processamento. Por isso, decidi que seria melhor gerar essa imagem do lado do servidor e depois passar essas imagens para os HoloLens dinamicamente.

Ao clicar num botão de um modelo temos o resultado mostrado na **Figura 3**.

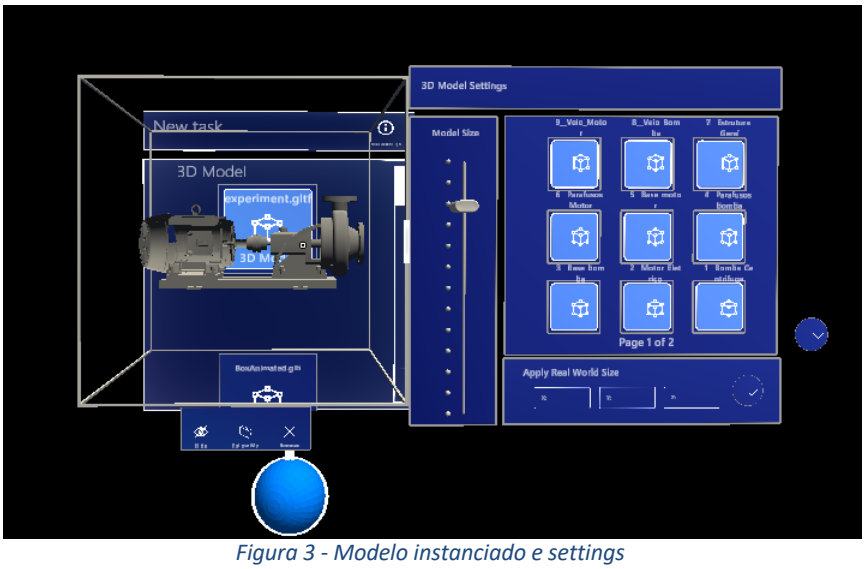


Figura 3 - Modelo instanciado e settings

Como se pode ver, o modelo é instanciado à esquerda e o painel de *settings* à direita.

O modelo é colocado no centro de um cubo de manipulação com um tamanho adequado. Para tal, utilizei o componente ***Bounds*** do *Mesh renderer* do modelo para saber onde está o centro do modelo e qual o tamanho real do modelo, e utilizo esses dados para o alterar a posição e o tamanho do modelo. Eu preciso desse centro da *Mesh*, porque o *pivot* do modelo nem sempre se encontra efetivamente no centro do modelo. Para obter esse componente ***Bounds***, percorro todos os nós do modelo, por largura, e juntos todas as *meshes numa* só para poder extrair o ***Bounds*** total. O código para essa junção de *meshes* é o seguinte,

```
if (combine.Count == 1)
{
    meshFilter.mesh = combine[0].mesh;
}
else
{
    var combineArray = combine.ToArray();
    combinedMesh.CombineMeshes(combineArray, true, true);
    combinedMesh.RecalculateBounds();
    if (combineArray.Length == 1)
    {
        combinedMesh.bounds = combineArray[0].mesh.bounds;
    }

    meshFilter.mesh = combinedMesh;
}

return meshFilter.mesh.bounds;
```

Para normalizar o tamanho do modelo e centrar o modelo dentro do manipulador uso o seguinte código,

```
// Scale GameObject according to mesh scale
Vector3 meshSize = bounds.size;
Debug.Log($"Model meshSize: {meshSize}");
float maxDimension = Mathf.Max(meshSize.x, meshSize.y, meshSize.z);
modelTransform.localScale = Vector3.one / maxDimension;

// Center the model inside the manipulator
modelTransform.localPosition -= Vector3.Scale(bounds.center,
modelTransform.localScale);
```

No painel de *settings* (**Figura 4**) temos um slider para alterar o tamanho do modelo, um conjunto de páginas de botões onde cada botão representa uma camada do modelo, e na parte de baixo temos 3 inputs onde o utilizador pode definir o tamanho do modelo usando unidades reais, neste caso metros. É possível que a introdução de tamanhos reais em cada eixo deforme o modelo, por isso esta solução terá de ser mais bem estudada no futuro.

As páginas e os respetivos botões são carregados dinamicamente. Existem 9 botões sempre instanciados, mas só são ativados o número de botões necessários. Ou seja, tenho um algoritmo que vai buscar os filhos do modelo (que serão as camadas) e para cada filho, ativo um botão e adiciono um evento ***OnClick()*** que faz *toggle* da visibilidade da camada, e também adiciono o nome da camada ao texto do botão.

Permitir que o utilizador insira um tamanho real também foi relativamente fácil porque o rácio de tamanho real para unidades do Unity é 1 metro para 1 unidade do Unity.

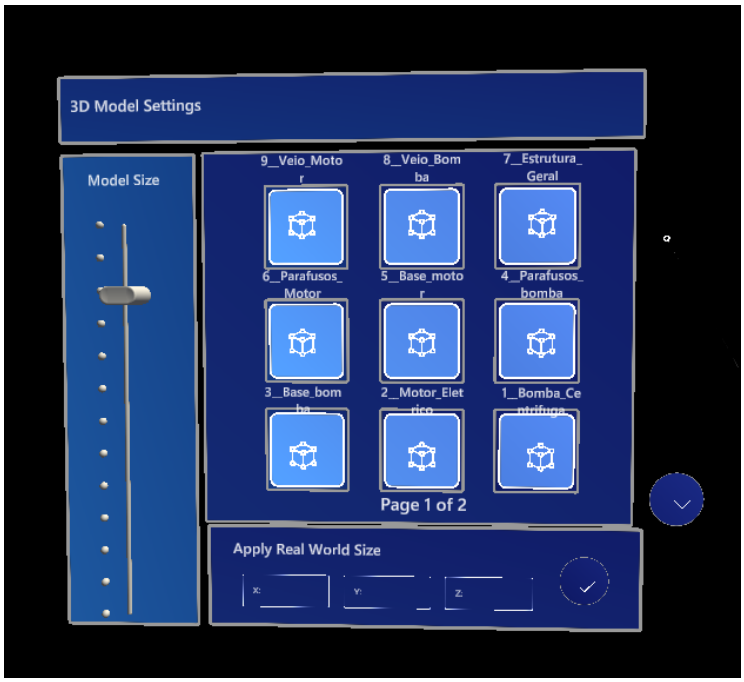


Figura 4 - Painel settings do modelo

Caso um modelo contenha animações, a biblioteca que utilizo para instanciar o modelo devolve também um *array* de ***AnimationClip***. Nesse caso, adiciono um componente ***Animation*** ao *GameObject* do modelo e adiciono o primeiro *AnimationClip* do *array* ao componente. Estamos a assumir que cada modelo só terá uma animação e não várias, mas suporte para várias animações seria trivial de fazer, apenas seria preciso desenhar uma nova interface. O botão de play/pause pode ser visto na **Figura 5**.

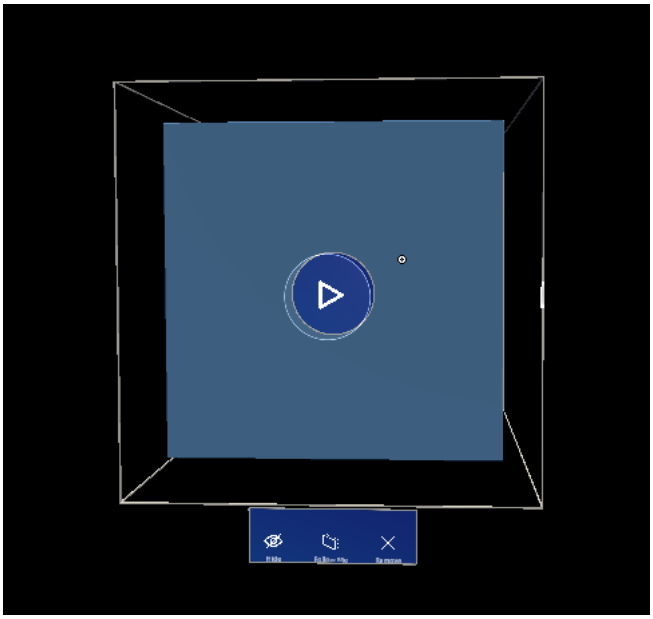


Figura 5 - Botão de play/pause no model handler

Depois adiciono um evento que será *triggered* quando a animação terminar. Este evento tem como função alterar o icon do botão para um icon de *restart* (**Figura 6**), isto porque independentemente do estado da animação, o botão é sempre o mesmo e apenas se altera a sua funcionalidade e o seu icon dependendo do estado da animação.

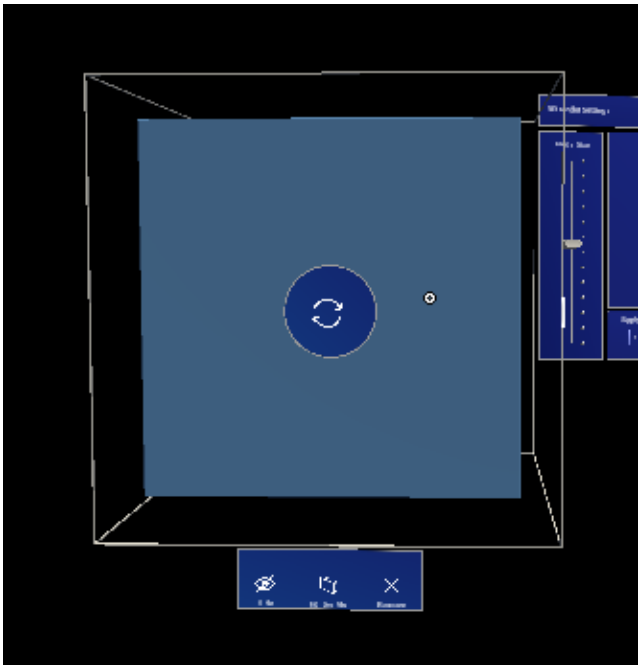


Figura 6 -Botão com icon de restart

Ao clicar no botão, a ação executada dependerá do *timestamp* atual da animação. Se o *timestamp* normalizado for menor que 1 (1 seria o último *frame* da animação) e a animação não estiver a correr, então queremos reproduzir a animação e passar o icon para um icon de Pause (**Figura 7**).

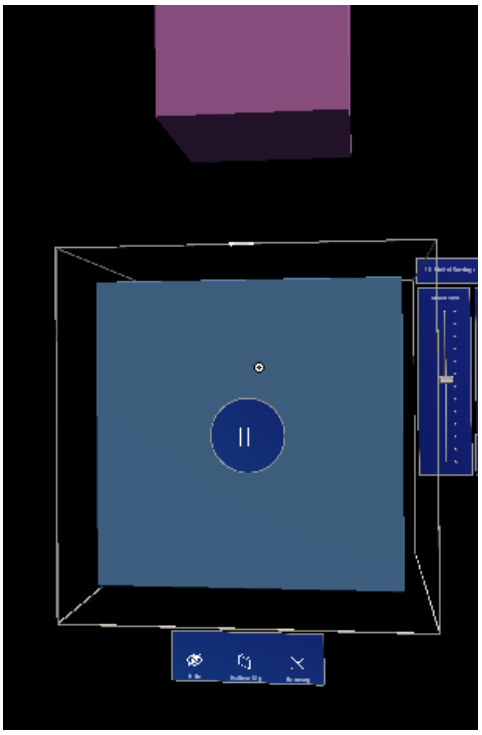


Figura 7 - Botão de Pause enquanto a animação é reproduzida

Se o *timestamp* normalizado for menos que 1 e a animação estiver a correr, queremos pausar a animação e passar o icon para um icon de Play. Se o *timestamp* normalizado for igual a 1 queremos passar o *time* para 0, ou seja, passar novamente para o primeiro *frame* da animação, e reproduzir a animação.

Tecnologias e metodologias utilizadas:

- Unity 3D;
- Microsoft Mixed Reality Tookit;
- Windows 10 (necessário para fazer *build* para UWP);
- C#;

- Hololens 2;
- Biblioteca *Siccity/GLTFUtility*;
- GraphQL.

Estado de desenvolvimento final das tarefas do Plano de Trabalhos

Consegui terminar por completo as *issues* da interação com modelos e animações. A *issue* de interação com camadas independentes do modelo, está funcional, mas queria ter testado mais. O UI desta *issue* também terá de ser alterado pois quando o desenvolvi a equipa de design ainda não tinha feito *mockups* do UI pretendido, por isso tive de fazer um provisório.

Este UI das definições do modelo também ficou com um pequeno bug que não tive tempo para resolver. Trata-se de um bug visual onde os cantos do painel podem ficar virados 90 graus como se pode ver na **Figura 8**. Não encontrei maneira de reproduzir este bug, pelo que não consegui resolver a tempo.

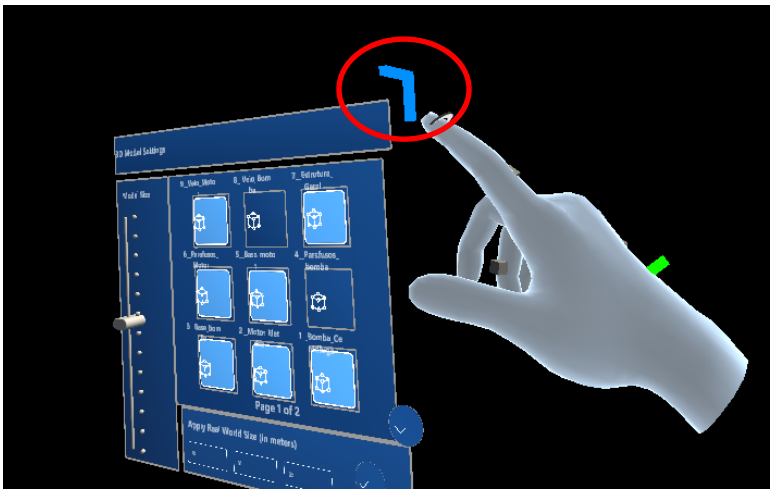


Figura 8 - Bug no canto do painel

O download de modelos 3D também ficou com um pequeno problema onde o programa congela durante uns 2-3 segundos ao instanciar um modelo pela primeira vez. Depois de discutir com o meu coordenador e outro membro da equipa de desenvolvimento HoloLens, chegamos à conclusão que encontrar uma solução seria bastante complicado pois, no Unity, é impossível executar métodos que alterem propriedades físicas de objetos (e.g escala, posição, rotação) numa *thread* diferente. Desconfiamos que a parte que mais contribui para o congelamento do programa seja o método da biblioteca ***GLTFUtility*** que utilizamos para instanciar um modelo em *runtime*, mas neste caso seria demasiado complicado encontrar uma solução. Como resolver este problema não era uma prioridade,

acabou por ficar assim, mas desconfiamos que quando atualizarem a versão do Unity que estamos a utilizar, o congelar do programa não será tão notável.

Resultados obtidos

No final do meu estágio a *Glartek* ficou com as seguintes funcionalidades desenvolvidas para o projeto ***GlarVision***:

- Testes de integração para algumas das funcionalidades do projeto;
- Suporte para carregamento de modelos 3D e visualização e interação deles nos HoloLens;
- Suporte para reprodução de animações de modelos 3D;
- Suporte para interação com camadas do modelo.

Todas as funcionalidades que enumerei excluindo a última, já foram validadas e estão atualmente na *branch master* do projeto. Estas estarão disponíveis aos clientes da empresa na próxima *major release*.

Relativamente aos resultados planeados, não só consegui cumprir o que estava planeado, mas também desenvolvi a funcionalidade das camadas, que não tinha sido inicialmente planeada.

Dificuldades e desafios

Durante o estágio, a componente mais desafiante foi o desenvolvimento dos testes de integração. Tendo sido no início do estágio, ainda não estava familiarizado com o código do programa por isso tive de estudar minuciosamente o código que estava a testar. Para além disso precisei de escrever os testes utilizando a *framework* ***Zenject*** que altera bastante o paradigma de programação, e eu não estava de todo habituado.

A componente que demorou mais tempo a ser desenvolvida foi o desenvolvimento da funcionalidade dos modelos 3D. Este processo envolveu muita pesquisa, toma de decisões e resolução de problemas técnicos e de geometria. Durou cerca de 2 meses.

Formação recebida, conhecimentos adquiridos

- Desenvolvimento em **Unity3D**, bem como todos os aspetos de desenvolvimento de aplicações gráficas 3D;

- Desenvolvimento de aplicações de realidade aumentada utilizando o **MRTK** (Mixed Reality Toolkit) da Microsoft em conjunto com o Unity3D;
- Utilização de controlo de versões em contexto empresarial. Neste caso usámos o ***GitLab*** e tive experiência e formação a criar *issues* e resolver *issues* seguindo todas as boas práticas;
- Programação em **C#**;
- Desenvolvimento de testes de integração no Unity.
- Utilização de uma *framework* de injeção de dependências, neste caso o ***Zenject***. No entanto só utilizei esta *framework* durante o desenvolvimento dos testes de integração e não para o desenvolvimento das outras funcionalidades.
- Desenvolvimento de aplicações dirigidas para hardware específico, no meu caso os **Microsoft HoloLens 2**;
- Trabalho em equipa num contexto empresarial seguindo o método **Agile**;
- Desenho de interfaces UI para programas de realidade aumentada.
- Colaborar com a equipa de Web com o objetivo de manter as componentes Web e HoloLens interligadas como um sistema distribuído.

Para além disso aprendi também a adaptar-me a um *codebase* para mim desconhecido. Esta fui uma das maiores dificuldades com que tive de lidar, pois no início não tinha conhecimento de como funcionava o *backend* e o *frontend* do programa e tive de estudar o código de modo a puder contribuir para ele.