

Laboratório de Desenvolvimento de Software

Trabalho Prático Grupo 15

Fábio Mendes, 8170157

José Baltar, 8170212

Rodrigo Coelho, 8170282

Índice

1. Introdução	3
1.1. Apresentação	3
1.2. Contextualização	3
1.3. Conceitos do jogo atualizados.....	3
2. Arquitetura	6
2.1. Descrição dos componentes do projeto	6
2.2. Autenticação	7
2.3. Tecnologias para cada componente	7
3. Primeira Milestone.....	8
3.1. Planeamento da Primeira Entrega	8
3.2. Criação e Desenvolvimento de Diagramas e outros Sistemas	8
3.3. Criação de Requisitos e Use-Cases	9
3.4. Desenvolvimento de <i>Mockups</i>	9
4. Segunda Milestone.....	10
4.1. Planeamento	10
4.2. Comunicação e Websockets	10
4.3. Lógica de jogo e serviço web correspondente.....	11
4.4. <i>Client-side</i> Unity e Integração com o serviço web	12
4.5. <i>Client-Side</i> Angular e serviço web de Gestão de Salas	13
4.6. Planeamento e incrementos para a próxima Milestone.....	15

Índice de Figuras

Figura 1 – Arquitetura Conceptual do Projeto	6
Figura 2 – Tecnologias para cada Componente	7
Figura 3 – Diagrama planeamento da primeira entrega	8
Figura 4 – Comunicação para processamento e atualizações do Jogo	11
Figura 5 – Exemplo URL de início da aplicação pelo browser	16
Figura 6 – Argumento enviado pelo browser e recebido pela aplicação do Unity	16

1. Introdução

1.1.Apresentação

Para a execução e desenvolvimento deste projeto, incluído no âmbito da Unidade de Curricular “Laboratório de Desenvolvimento de Software” e de tema livre à escolha do grupo, foi escolhido o desenvolvimento de um de um videojogo de estratégia multijogador, do estilo *Tabletop*, no qual quatro jogadores poderão competir entre si até que exista apenas um vencedor.

Neste relatório irão ser explicadas as decisões e mudanças tomadas neste novo ciclo de desenvolvimento, assim como as tarefas de implementação do software anteriormente desenhadas na primeira *milestone*.

1.2.Contextualização

Na segunda *milestone* do projeto, foi decido dar prioridade à implementação inicial da lógica de jogo e comunicação com os clientes. Desde então, as tecnologias a serem utilizadas e a sua estrutura básica manteve-se, embora a lógica do jogo, tal como estava planeada, sofreu alterações e mudanças, pelo que alguns dos requisitos delineados na primeira *milestone* sofreram algumas modificações. Estas modificações são tanto visíveis no atualizado documento de *SRS*, como no GitLab do projeto.

Nesta entrega foi então desenvolvida a ligação entre o servidor e os diversos clientes que a ele se irão juntar, e criada grande parte da lógica base que torna o jogo jogável tanto no cliente como no servidor. Como descrito, os métodos lógicos encontram no servidor, pelo que o cliente tem a responsabilidade de mostrar aos jogadores o novo estado de jogo derivado das suas ações, isto é, um estado atualizado do jogo. Adicionalmente foi já criado o *frontend* da plataforma que irá servir de “GameClient” em browser.

1.3.Conceitos do jogo atualizados

Neste tópico são abordados os principais conceitos que fazem parte da estrutura do jogo.

Começando pelo mapa, este terá sempre uma forma fixa e Hexagonal com três lados maiores e três lados mais pequenos. Três fações (jogadores) localizadas nos lados mais pequenos, com outra facção centrada no mapa. O terreno entre as diversas fações é neutral e poderá ser anexado por estas, até se tornar o centro do confronto.

Existem, portanto, quatro Fações que representam os quatros jogadores de cada partida, sendo estas e as suas características:

- Remnant-> Cor Vermelha, centrada no centro do mapa, Ponto forte: Defesa, Tropas de Infantaria Pesada e Cerco; Condições de Derrota: Derrota Total (Aniquilação da Fação);
- Confederation-> Cor Amarela, Sul do mapa; Ponto forte: Defensiva, Tropas de Míssil e Infantaria Ligeira; Condições de Derrota: Conquista de Centro de Hegemonia ("Capital"), Capitulação ou Perda de 30% sobre Manpower Original
- Royalists-> Cor Azul, Nordeste do mapa; Ponto forte: Ataque, Tropas de Infantaria Pesada e Míssil; Condições de Derrota: Conquista de Centro de Hegemonia ("Capital"), Capitulação ou Perda de 50% sobre Ouro Original;
- Hordes-> Cor Verde, Noroeste do Mapa; Ponto forte: Ataque, Tropas de Cavalaria Míssil e Ligeira; Condições de Derrota: Conquista de Centro de Hegemonia ("Capital"), Capitulação ou Perda de 60% da Moral da População

Em relação aos Recursos disponíveis, existem quatro que podem ser gerados e utilizados: Ouro e Metal (Minas), Madeira (Serraria), Comida (Quintas) e Manpower (Cidades). Os recursos são gerados pelas diversas regiões que o jogador ocupa. Devido a esse facto, certas regiões são especializadas em tipos de recursos diferentes, sendo que apresentam diferentes bónus dependendo do tipo de região. Os recursos são usados unicamente para criar e manter unidades de tropas e exércitos.

A nível diplomático, cada Fação encontra-se, inicialmente, em paz com as restantes. É através de declarações de guerra que a guerra começa, quando as fações criam fronteiras possíveis umas com as outras. Paz pode ser declarada apenas 1 vez entre cada fação. Alianças oficiais entre jogadores não são possíveis.

Existem vários tipos de tropas, todos eles disponíveis a todas as fações, sendo estas:

- Infantaria Pesada
- Infantaria Ligeira
- Míssil
- Cerco
- Cavalaria Pesada
- Cavalaria Ligeira
- Cavalaria Míssil

Cada fação tem, no entanto, bónus diferentes com base nos seus pontos fortes, para cada tipo de unidade, como referido anteriormente no tópico referente às mesmas.

As fações lutam através de Exércitos, compostos por diversos tipos de tropas, até um máximo de 15 unidades. Para se criar um exército é necessário usar os recursos disponíveis, numa taxa fixa. Para se treinar tropas, , deverá estar estacionado numa região em que o

recrutamento é possível. Os exércitos têm uma *manpower* associado a cada unidade. Esse *manpower* é o total de "tropas", ou equivalente a um HP e a Força, simultaneamente. Em combate, este será drenado com base no resultado dos exércitos. Para ser reabastecido, o exército terá de ser "reparado" para recrutar novas tropas para unidades. Cada região apenas pode ter um exército estacionado. Um exército só se poderá mover para uma região diretamente vizinha e sem barreiras ou outros exércitos estacionados. É possível atacar uma região, vizinha, caso tenha exércitos estacionado, combate será iniciado. Caso não tenha, a região é anexada sem qualquer custo.

2. Arquitetura

2.1. Descrição dos componentes do projeto

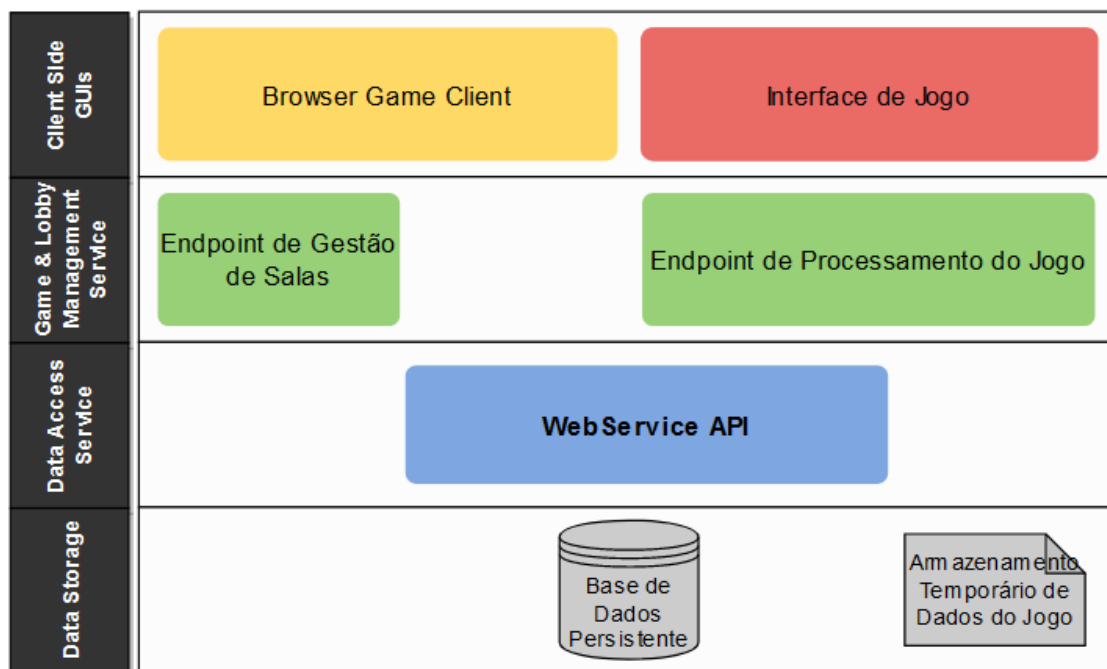


Figura 1 – Arquitetura Conceptual do Projeto

Como versão final, a arquitetura assenta num Serviço Web (ASP.NET Core) responsável pelo armazenamento e gestão de toda a informação relativa ao Sistema, desde dados de autenticação e utilizadores até ao histórico de jogos. Comunicam com o serviço tanto os Clientes, descritos mais à frente, como o Servidor Dedicado de jogo.

Então, sendo o jogo um multiplayer de estratégia, deverá existir uma ferramenta para que os jogadores se consigam organizar entre si, ou automaticamente, numa *sala* antes de dar início ao jogo. Sendo assim, foi decidido o desenvolvimento de um *Game Client* em browser para o efeito. Este *Game Client* será responsável por disponibilizar aos Utilizadores uma interface de acesso aos seus dados pessoais, histórico de jogos, autenticação e ao sistema de *salas* ou *matchmaking*. Um segundo Cliente será a própria interface do jogo, desenvolvida em Unity, que fornece todos os mecanismos de interação gráfica com as mecânicas do jogo, sendo que todo o processamento se irá realizar no próprio servidor.

Finalmente, a comunicação entre os jogos dentro do servidor dedicado será feita através de Sockets. E sendo que uma *sala* em princípio deverá conter um sistema de chat, também serão utilizados Sockets neste contexto. Ou seja, o sistema de *salas* e o servidor dedicado ao jogo estarão contidos num mesmo serviço, desenvolvido à base de Sockets.

2.2. Autenticação

Para garantir privacidade e consistência em todo o projeto, existirá um processo de autenticação que permite ao utilizador manter a sessão iniciada entre qualquer componente, desde o Login no browser até ao início do jogo no Desktop.

Para mais detalhes, está presente no documento SRS, em anexo, uma descrição em detalhe deste sistema, tal como um diagrama de sequência UML acompanhante.

É também importante notar que, no cliente, a interface de jogo estará completamente separada da interface de interação com as salas. Contudo, esses dois componentes comunicam com um mesmo serviço, dispostos em dois *endpoints* diferentes. Ou seja, quando um cliente faz ligação ao servidor pela interface de jogo, já deverá estar identificado num *lobby* que deu início a esse jogo. É assim necessário existir uma identificação única para cada jogador e ser possível relacionar a conexão em *socket* a essa identificação.

2.3. Tecnologias para cada componente

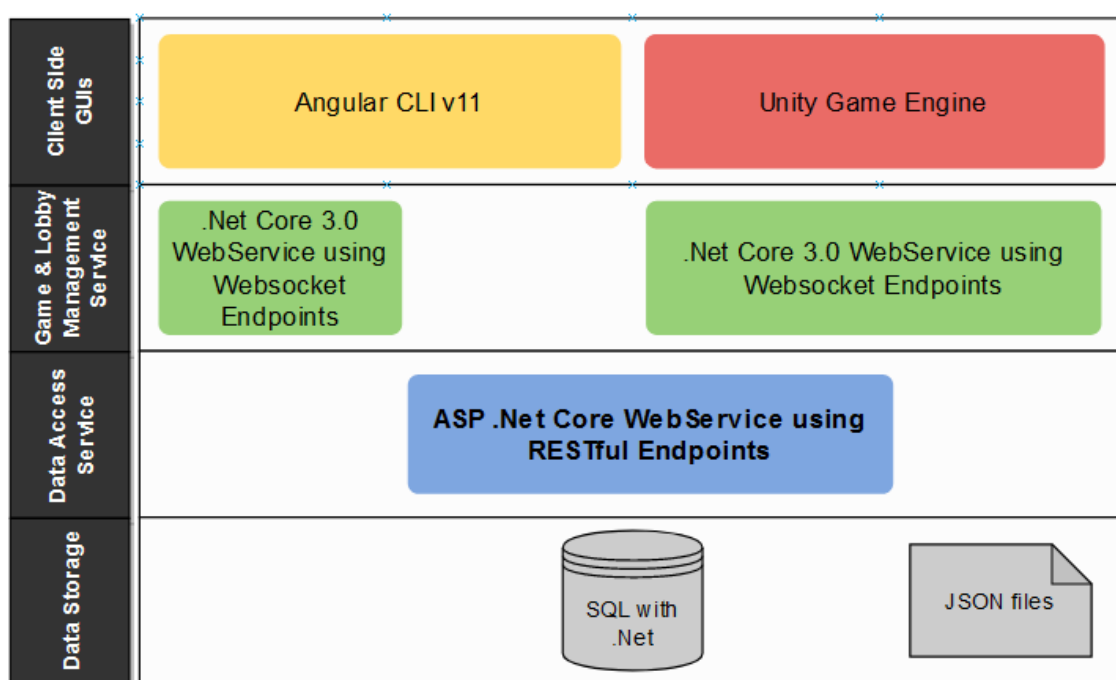


Figura 2 – Tecnologias para cada Componente

Adicionalmente, foi realizado um estudo com o intuito de confirmar que é possível iniciar a Interface de Jogo, desenvolvida em Unity, pelo do Browser. Este processo pode ser visto em maior detalhe no **Anexo-A** deste documento.

3. Primeira Milestone

3.1. Planeamento da Primeira Entrega

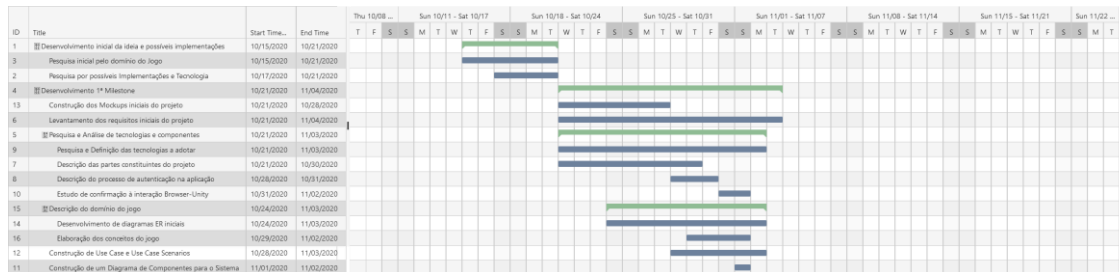


Figura 3 – Diagrama planeamento da primeira entrega

3.2. Criação e Desenvolvimento de Diagramas e outros Sistemas

Foram elaborados alguns diagramas, com o presente propósito de explicar, uma forma simples as diversas componentes do projeto.

Os diagramas ER criados procuram explicar as diversas relações entre as entidades que irão permanentes e estarão guardadas de forma a suportar a lógica de funcionamento dos diversos sistemas e aquelas que descrevem a situação em que estes se encontram naquele momento. Ou seja, são estes os dados em específico que os componentes necessitam de forma a poder funcionar e modela de forma bastante inicial a forma como certas componentes de cada um dos sistemas funcionam.

Relativamente ao sistema de comunicação e componentes globais do projeto, são de notar alguns dos *mindsets* seguidos durante a pesquisa desenvolvida durante a primeira milestone:

- Inicialmente a ideia do "Game Client" não existia, ou seja. seria tudo desenvolvido dentro do motor do Unity. Esta seria uma ideia igualmente viável, no entanto decidimos seguir com a versão do browser por ser mais enriquecedor e multiplataforma. Um utilizador poderá assim aceder ao seu histórico de jogos através de qualquer máquina que contenha um Browser, como um smartphone, contudo não poderá entrar numa "sala" para dar início a um jogo.
- Também se pensou em desenvolver o sistema de salas através do serviço HTTP, mas isto impossibilita ou tornaria complicado a adição do sistema de chat dentro das salas, tal como a transição da sala para uma instância de jogo no servidor dedicado.

Sobre o sistema de autenticação, outras ideias pertinentes pensadas pelo grupo na altura da primeira milestone, mas que não conseguiram o *final cut*, foram:

- Foi considerada a alternativa de fazer a autenticação do cliente na API e nos servidores de jogo no *mesmo local*. No entanto, sendo que a implementação dos dois serviços difere bastante (HTTP requests e TCP/UDP sockets, respetivamente) ficou decidido que deverão estar separados. Conclui-se então que a autenticação é realizada sempre pelo serviço web.

3.3. Criação de Requisitos e Use-Cases

O principal esforço desta entrega consistiu precisamente na construção de Requisitos funcionais. Os requisitos desenvolvidos consistem no entendimento presente acerca daquilo que planeamos para o jogo e também para o que cada utilizador poderá fazer e consultar. Com isto, os requisitos apresentam a forma como pretendemos que o jogo funcione de uma maneira mais prática, isto é, além de meros conceitos, apresentando um conjunto de características e condições que são de fácil entendimento e possível transcrição para a programação do próprio jogo e também de todas as componentes do cliente, tais como *matchmaking*, entre outros. No entanto, estes requisitos não são necessariamente os finais. Uma vez que estamos a trabalhar utilizando metodologias ágeis, estes servem como *kickstart* para o entendimento geral do projeto, mas são passíveis a mudança e a incremento de novos requisitos, tendo em conta o desenvolvimento do projeto e gestão de mudança que poderá ter de ser efetuada.

Em relação a Use-Cases e a Scenários, pretende-se mais uma vez mostrar funcionalidades do projeto, mas desta vez da perspetiva do utilizador. Ao longo do correr do projeto prevê-se que os use-cases cenários sejam alterados principalmente em relação aos passos a percorrer para executar a funcionalidade, visto que nesta fase do projeto ainda não temos uma ideia muito solidificada relativamente à interação do jogador com o jogo.

3.4. Desenvolvimento de Mockups

Ao longo do desenvolvimento do projeto foram desenvolvidos mockups, com o principal objetivo de ajudar a ilustrar as potencialidades do jogo e das interações entre os jogadores e a interface. Os mockups desenvolvidos correspondem, sobretudo ao GameClient (estilo BattleLog) em browser e à UI de Jogo, embora não sejam, de qualquer maneira, finais ou indicativas do estado final do produto, uma vez que complexidades de implementação ou outras particularidades tornam uma implementação exata e adequada destes mockups difícil e até desnecessário.

4. Segunda Milestone

4.1. Planeamento

Ao contrário da milestone anterior, onde foi construído um diagrama de Gant para o planeamento, todo o planeamento desta segunda milestone está descrito em dois *Sprints*, detalhados no repositório GitLab relativo ao projeto do grupo.

4.2. Comunicação e Websockets

Partindo do princípio de que o nosso jogo é um Strategy Game que funciona à base de timers e decisões, sendo o tempo de reação quase inexistente, não será importante dar muita atenção a grandes implementações para gestão de *Lag*. Contudo, isto não significa que se vá ignorar o tema por completo! No fundo, o *lag* será gerido pela forma em que os dados serão enviados entre os jogadores e não tanto pela qualidade das conexões.

Então, visto que a implementação do servidor dedicado do jogo e gestão de conexões é concretizada como um serviço web, concluiu-se que será utilizado o protocolo WebSockets para toda a comunicação client-server em tempo real, em que múltiplos clientes se conectam a um único servidor que fornece os serviços necessários. Contudo, o grupo tem consciência de possíveis problemas relativamente a este tema.

WebSockets é um protocolo que junta um pedido HTTP com a abertura de um canal em TCP, para construir uma *ponte* de comunicação bidirecional entre o cliente e o servidor. Em TCP, o maior problema que o grupo poderá encontrar neste contexto será o bloqueio da comunicação em caso de packet loss, já que o protocolo se encarrega da organização dos pacotes enviados. Isto significa que a utilização do protocolo TCP deverá existir em situações como o envio dos inputs do jogador para o servidor e mensagens de chat, sendo que o envio de broadcasts se pode tornar problemático, contudo o grupo ainda não encontrou problemas com tal.

É de notar também que as mensagens enviadas entre as conexões estarão descritas em formato JSON. Ou seja, para cada mensagem enviada existe *serialization* do lado de quem envia e *deserialization* por quem recebe. É possível que isto venha a criar alguma lentidão no envio de cada mensagem, contudo é a melhor abordagem a seguir para o momento, tendo em conta tudo o que já foi e será descrito neste relatório.

Falando em específico do contexto do jogo, e como pode ser visto pelo diagrama abaixo, a comunicação tem foco principal a partir do Cliente, uma vez que cada Cliente envia informação sobre os eventos que irá efetuar a cada turno de jogo. O Servidor tem então a tarefa de os

executar e, a cada final de turno, enviar informação atualizada para todos os clientes conectados àquela sala.

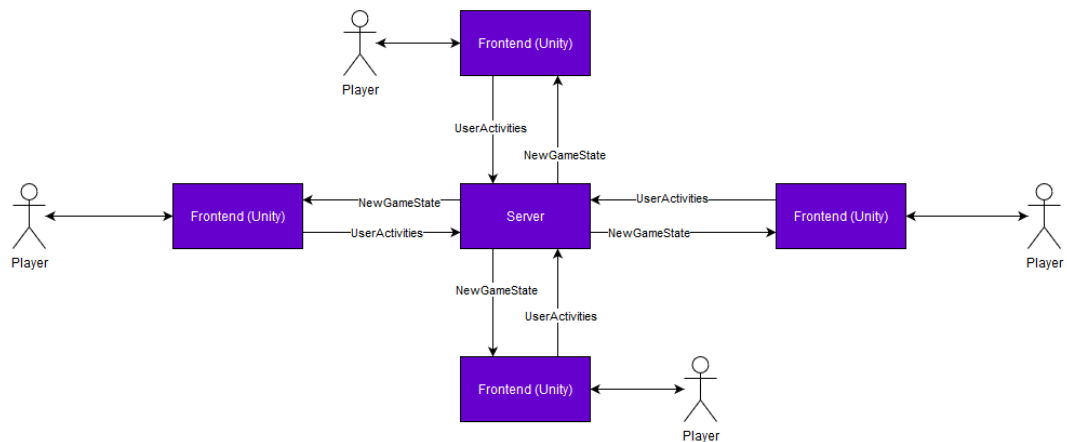


Figura 4 – Comunicação para processamento e atualizações do Jogo

4.3. Lógica de jogo e serviço web correspondente

Para a segunda *milestone*, foi importante haver já um foco grande na lógica de jogo, em servidor, uma vez que seria o core do projeto enquanto produto. Foi, por isso, considerado crucial que se começasse pelos métodos que efetivamente formam o conjunto mais fulcral do projeto, que nós consideramos como a “Lógica de Mapa”, ou a Gestão de Mapa.

De um ponto de vista programático, este Mapa definido no servidor é um Grafo implementado com recurso a uma Matriz, sendo gerado através da leitura de um ficheiro em formato JSON contendo as informações referentes a cada região, isto é, a unidade atómica do mapa. Cada vértice é, portanto, uma ligação do grafo e a lógica de mapa é uma iteração baseada em princípio de abstração sobre este facto. O grafo de mapa é um Grafo Não Pesado e Não orientado, contendo apenas as ligações lógicas, ou seja, as fronteiras, entre as diversas regiões.

Além do Mapa, foi também considerado crucial o desenvolvimento referente às componentes para Exército, Unidades e Fações, incluindo a gestão de recursos disponíveis a cada fação. Assim, sendo, foram implementados os métodos que permitissem a que estes interagissem entre si e com as regiões já previamente implementado no 1º Sprint. Utilizando a Coleção “List” existente em C#, um gestor para a Instância de jogo, que guardará todos os diversos dados referentes ao presente estado do jogo no servidor são criados. Tal como descrevem os requisitos, foram implementados métodos que potenciem a criação de exércitos, recrutamento de unidades para estes, movimento, combate, anexações e conquistas de região, além de métodos mais específicos que se encontram detalhados nos requisitos do projeto, no

documento SRS anexo. Detalhes sobre o seu funcionamento são encontrados no mesmo documento.

Como forma de tratar a informação recebida dos clientes, o servidor está sempre “à escuta” de “Eventos” (anterior designados de pedidos) causados pelos jogadores, eventos esses que guarda numa Queue, ordenando de forma natural pela ordem em que chegam ao servidor. Esses pedidos são depois encaminhados para um “**Protocolo**” que separa e executa as ações que são necessárias para cada específico evento, chamando o método designado na Instância de jogo. *Detalhes sobre o protocolo mencionado podem ser analisados no documento SRS em anexo.*

O que acontece é que, do ponto de vista de servidor, o estado de jogo poderá estar em constante atualização, no entanto, do ponto de vista do cliente, as coisas estão se a manter iguais durante o período de duração do turno. Por isso mesmo, garantimos que desta maneira não existem “conflitos” uma vez que todas as ações são executadas com um certo *flow*. O que acontece é que certas ações que o jogador não conseguiria antever poderão, de facto, acontecer, uma vez que não está a verificar o conjunto de ações que os outros jogadores estarão a tomar. Uma vez que estas ações são inesperadas, significa que um conjunto de atributos poderá ter sido completamente alterado no lado do servidor, o que compromete a integridade da solução. Para tal, um conjunto forte de verificações é aplicado para garantir que os dados ainda se encontram inalterados. Caso haja operações que não estejam necessariamente dependentes destes dados, é necessário tornar a sua execução o mais generalizada possível. Em casos menos extremos, o que poderá acontecer é que uma certa ação será bloqueada.

De um ponto de vista estrutural, a solução foi implementada recorrendo a um conjunto de “Managers” que tratam de cada componente em específico, guardando em Lista as informações pertinentes ao Estado de jogo. Quando um evento é chamado no Protocolo, a Instância de jogo pertencente ao Room, à qual a informação é enviada, conforme previamente mencionado.

4.4.Client-side Unity e Integração com o serviço web

No cliente Unity, houve um grande foco no sistema de comunicação com o servidor responsável pela gestão do jogo e na interação do jogador com a interface.

Relativamente a comunicação com o servidor foram implementados duas threads, uma responsável por receber os dados do servidor e outra responsável por enviar os inputs dos utilizadores para o servidor. Foi implementado deste modo para que estas tarefas não interferiam com a performance do jogo, ou seja, que não atrasem a execução de outras operações. Com esta implementação não se pode colocar a thread responsável por receber os dados do servidor também responsável por processá-los e atualizar o jogo em si, porque existem

certas ações do Unity que não são “thread-safe”. Com isto em mente a thread em questão apenas recebe a informação e coloca-a numa ConcurrentQueue que outros objetos podem aceder e por sua vez processar a informação.

Também é importante salientar que a Classe responsável pela comunicação com o servidor é um singleton, isto quer dizer que a instância da classe é a mesma onde quer que seja acedida na “Scene” do jogo.

Quanto à interação do utilizador com a interface foram impostas as devidas restrições para impedir o congestionamento do servidor com pedidos repetidos ou pedidos que já se sabe que o servidor vai “recusar”. Por exemplo quando se cria um exército numa região é verificado se o utilizador é o dono, se já não contem um exército, o tipo de região, se já não colocou um exército a mover-se para essa região na mesma ronda, entre outras restrições.

Outro caso é a criação de “dummies”, e por “dummies” quer-se dizer objetos que ainda não foram processados pelo servidor, mas que são necessários para o utilizador poder executar outras ações que os envolve. Como no caso de criar um exército, esse input só vai ser processado para a próxima ronda, mas na ronda em que é criado o utilizador já pode adicionar unidades a esse exército, isto também bloqueia a opção de criar exército nesta região o que, como referido anteriormente, previne o congestionamento do servidor. Neste caso é necessário criar um “dummies” tanto para o exército como para as unidades adicionadas.

Estes “dummies” quando se recebe o update são eliminados e substituídos pelo objeto “real”, o que é recebido pelo servidor.

Nesta milestone o cliente é que está responsável por pedir as atualizações ao servidor, mas no próximo milestone este comportamento irá ser alterado e o servidor é que ficará responsável por enviar os updates para os clientes.

4.5. *Client-Side* Angular e serviço web de Gestão de Salas

Toda a implementação no *front-end* para o componente *Game Client* está desenvolvida recorrendo à framework Angular. Igualmente, para garantir que a implementação respondesse aos requisitos do domínio, foram construídos mockups como representação base do produto final. Contudo, estes mockups têm como objetivo demonstrar visualmente as funcionalidades requeridas para o projeto e não tanto dar como final a aparência cosmética, ou seja, cores e alguns posicionamentos do conteúdo poderão sofrer modificações durante a implementação.

No entanto, é de notar que nem todas as funcionalidades deste componente ficaram planeadas para esta milestone, visto que a implementação do Webservice de armazenamento e acesso aos dados dos utilizadores está planeada para a terceira milestone. Com isto, do ponto

de vista funcional o *Game Client* atualmente contém implementado toda a organização das páginas web, a inicialização de *lobbies* e, em parte, a transição para o jogo, sendo que esta última terá que ser manualmente ativada. *Mockups e outros detalhes podem ser vistos no documento SRS em anexo.*

Então, para que seja possível o relacionamento entre salas de lobby criadas pelo componente *Game Client*, descrito acima, e salas relacionadas ao processamento do próprio jogo, está implementado no servidor um sistema capaz de gerir um conjunto de salas de forma concorrente. O sistema é utilizado inicialmente pelo endpoint que gere as conexões vindas do cliente no browser e depois acedido pelo endpoint de gestão de conexões vindas do cliente de interação com o jogo, em Unity.

Resumidamente, a lógica do sistema passa pela partilha de uma coleção de *salas* entre todas as conexões ligadas a qualquer um dos endpoints. Para isto ser possível, foi seguida uma abordagem baseada no conceito de *Singleton*, uma classe que contém e gere a referida coleção e é instanciada uma única vez. Dito isto, é absolutamente necessário existir um controlo de acesso concorrente aos dados, visto que a cada nova conexão, atualização ou fecho essa coleção sofre algum tipo de modificação.

Outro ponto importante a notar é o conjunto de verificações relativamente a cada sala em si. Isto é, cada sala terá no máximo até quatro jogadores. A transição entre o *lobby* e o *jogo* implica que seja necessário existir algum tipo de relação entre as duas salas. Para resolver este problema, cada instância de uma sala é a mesma para um *lobby* como para um *jogo*. No fundo, quando um jogador passa de um *lobby* para o *jogo*, após o evento de início de jogo, a sua instância no servidor, guardada dentro da sala, mantém-se e só se remove a referência do socket que representa a sua conexão. Após o início do jogo, o jogador volta a conectar-se e o servidor fica responsável por atualizar a nova referência dentro da sala onde o jogador estiver.

Pode ser visto no documento SRS, em anexo, um diagrama de classes que descreve todo o Sistema de Gestão de Salas em maior detalhe.

Concluindo, tendo em conta a abertura de um canal constante de comunicação entre clientes, em lobby, e o servidor é inevitável a criação de um sistema que organize as mensagens trocadas entre os dois lados. A ideia aqui será um seguimento do trabalho desenvolvido para o protocolo de comunicação entre os clientes em jogo, neste caso entre os clientes em lobby, no browser, e o servidor. Este **protocolo** tem como objetivo definir um conjunto de regras para as mensagens JSON trocadas, definindo no fundo um conjunto de *eventos* que representam operações com inputs e outputs. *Para mais detalhes sobre que tipos de eventos estão definidos no protocolo, veja-se o documento SRS em anexo.*

4.6. Planeamento e incrementos para a próxima Milestone

Para a próxima *milestone*, o planeamento preliminar implica que os métodos e processos começados nesta Milestone sejam efetivamente terminados e otimizados, além de abordar todos os outros requisitos que ainda necessitam de ser implementados.

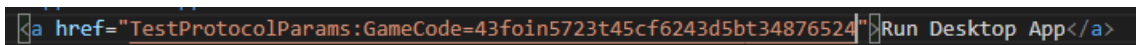
Assim sendo, **para a próxima Milestone**, o principal foco será de facto terminar todos os métodos referentes às vertentes do jogo, tanto no Cliente como no Servidor, assim como a implementação dos métodos que permitam a utilização da Plataforma que faça o suporte ao jogo. No entanto, é de esperar que se implementem alterações que corrijam qualquer tipo de problema, mau planeamento ou má execução de etapas da Segunda *Milestone*.

Anexos

Anexo A – *Start-Up* do jogo a partir do Browser

Resumidamente, será possível executar deste modo se registarmos um costum url protocol no windows. Podemos fazer este registo de forma manual, mas a ideia seria utilizar a aplicação externa “Inno” para gerar o instalador do jogo, nesta aplicação é possível alterar o script de instalação para que faça o registo do protocolo no Windows, no momento da instalação.

Relativamente à implementação nas tecnologias envolvidas é necessário da parte do browser criar um elemento html do tipo href em que o valor seria "(nomeDoProtocolo):(parâmetros)".




```
<a href="TestProtocolParams:GameCode=43foin5723t45cf6243d5bt34876524" Run Desktop App</a>
```

Figura 5 – Exemplo URL de início da aplicação pelo browser

Isto é um exemplo de teste no qual o parâmetro “GameCode” é estático, numa situação real seria um valor dinâmico que dependeria da sessão de jogo à qual se refere.

Do lado da aplicação desktop é preciso aceder aos argumentos da linha de comandos do processo atual para obter os parâmetros que o browser envia. Para testar isto foi feito uma aplicação em Unity básica cuja única função é mostrar os argumentos da linha de comandos.



```
testprotocolparams:GameCode=43foin5723t45cf6243d5bt34876524
```

Figura 6 – Argumento enviado pelo browser e recebido pela aplicação do Unity

Depois de receber este argumento só temos de extrair a informação necessária, neste caso seria o GameCode.