

## **Terceiro Trabalho Prático**

Alunos            Manuel Francisco Dias Marques, nº36836  
                     Oxana Dizdari, nº39278  
                     Beatriz Patusco Neto, nº39320

Engenheiro    Luís Assunção

Relatório do terceiro trabalho prático, realizado no âmbito de Sistemas Distribuídos,  
do curso de licenciatura em Engenharia Informática e de Computadores  
Semestre de Verão 2014/2015

Maio de 2015

# Índice

LISTA DE FIGURAS .....	V
<b>1. INTRODUÇÃO .....</b>	<b>1</b>
1.1 PROBLEMA .....	1
1.2 PONTOS IMPORTANTES.....	1
1.3 TEMAS DA UNIDADE CURRICULAR .....	1
<b>2. SOLUÇÃO IMPLEMENTADA.....</b>	<b>2</b>
2.1 SERVIÇO DE JOGO.....	2
2.1.1 Criação de Jogo.....	3
2.1.2 Regras e funcionamento de Jogo.....	4
2.1.3 Registo do Player e como se joga .....	5
2.2 SERVIÇO DE NOTIFICAÇÃO.....	7
2.3 SERVIÇO DE TRADUÇÃO .....	8
<b>3. CONCLUSÕES.....</b>	<b>9</b>

# Lista de Figuras

Figura 1 - Início de jogo com o tabuleiro de 20:10 .....	3
Figura 2 - Registo do utilizador.....	5
Figura 3 - Jogada feita em 1:1 .....	6

# 1. Introdução

Neste documento, irá ser descrito o problema proposto neste terceiro trabalho de Sistemas Distribuídos. Irá também ser explicado, de forma mais explícita possível, as decisões tomadas perante cada problema enfrentado e a sua respectiva solução.

## 1.1 Problema

Neste trabalho foi proposta a implementação de um serviço *Jogo*. Este jogo consiste na procura de um tesouro num tabuleiro  $N \times N$ . Este tabuleiro é preenchido por mortes, vidas e casa vazias, para além do tesouro. Este último é único e a sua descoberta dá a vitória ao jogador que o encontrou. O jogo é partilhado entre um ou mais participantes e após o tesouro ser descoberto todos os jogadores são notificados que o jogo acabou e que perderam, caso não tenham sido eles a descobrir o tesouro. Esta notificação é feita através de um serviço cujo contrato é disponibilizado pelo *Jogo*, tendo cada *player* que se reger pelo conjunto de métodos presente nesse contrato.

## 1.2 Pontos importantes

Para além do objecivo principal explicado na secção anterior, irá ser referido como é feita a criação do jogo, nomeadamente o estado inicial do tabuleiro e de cada jogador presente nele. Quanto aos jogadores será explicado mais detalhadamente como estes se irão registar no jogo. Irá ser explicada a importância de cada uma das casas do tabuleiro, sendo o tesouro a casa principal, e vida, morte e *nothing* as secundárias. Relativamente à afetação do jogo irão também ser explicados aspectos importantes sobre esse tema, como os acontecimentos após cada jogada.

## 1.3 Temas da unidade curricular

Neste trabalho prático serão abordados e consolidados alguns dos temas de Sistemas Distribuídos que foram leccionados nas aulas. Alguns deles são: Arquitetura orientada aos Serviços (SOA); Windows Communication Foundation (WCF); Alguns aspetos fundamentais do WCF como Endpoints, Protocolos de Comunicação, Service Operation, Service Contract, Operation Contract, Bindings, Ficheiros de Configuração, entre outros.

## 2. Solução Implementada

### 2.1 Serviço de Jogo

O Jogo é o elemento central do trabalho, sendo a parte mais importante. Está alojado num serviço *GameService* e suporta um jogo de procura de um tesouro em que podem participar múltiplos jogadores. É este que faz a gestão dos *players* e que os notifica quanto a resultados do jogo e publicidade neste presente (acho que não é este, é mesmo o *GameManager*). A publicidade é emitida usando um serviço intermédio que irá ser explicado posteriormente.

*GameService* disponibiliza dois contratos. O primeiro é o *IGameManager* cujas operações permitidas são *StartGame* – responsável por iniciar o jogo, especificando o tamanho do tabuleiro, *EndGame* – que serve para fechar o jogo e o *SetAdv* – que é utilizado para enviar publicidade aos jogadores. O segundo é *IGamePlayer* que permite ao jogador juntar-se ao jogo, especificando o seu nome e a sua língua (esta última irá servir para a publicidade enviada pelo *Manager* ser traduzida para a língua do jogador em questão e só depois ser enviada) – *JoinGame*. Permite também fazer jogadas – *MakeMove*, especificando a casa onde o mesmo quer jogar, e sair do Jogo – *ExitGame*.

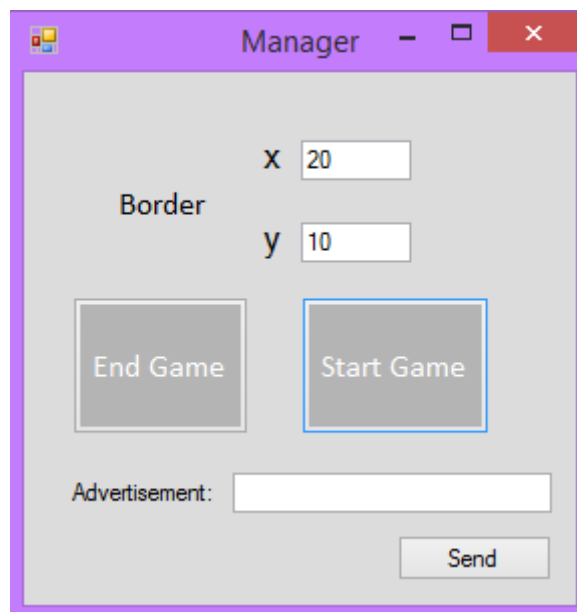
O contrato *IGamePlayer* do *GameService* disponibiliza também um *CallbackContract* do tipo *IGamePlayerReceiverCallback*, constituído por dois métodos, o *NewAnnounce* – utilizado para notificar os jogadores, por exemplo sobre o fim do jogo caso alguém já tenha ganho, e o *NewAdvertisement* – para enviar a publicidade traduzida aos jogadores.

Para o *GameService* foi escolhido o *InstanceContextMode* como *Single*, visto que o pretendido é que a mesma instância seja partilhada por vários clientes, neste caso múltiplos jogadores. O *ConcurrencyMode* escolhido é *Multiple*. No sentido crítico deste ponto, apesar de não termos locks, sabemos que a sua utilização é muito importante de maneira a evitar problemas de concorrência.

### 2.1.1 Criação de Jogo

Uma das regras do jogo é que é o Gestor que o deve iniciar. Para isso, foi criada uma aplicação *GameManager* que tem como responsabilidade indicar o tamanho do tabuleiro e começar o jogo em si. Este *GameManager* utiliza o contrato *IGameManager*, específico para utilizadores com privilégio sobre o jogo. Estes privilégios abrangem o início e o fim de um jogo tal como a gestão de anúncios. Ou seja, tendo controlo sobre qual o anúncio inicial e podendo mudar e enviar vários anúncios ao longo do jogo.

O jogo decorre sobre um tabuleiro NxN definido pelo *GameManager*. Aqui é também definido o anúncio inicial que vai estar presente na *user interface* de cada participante. Ao clicar no botão *StartGame* (Figura 1), depois de definido o tamanho do tabuleiro, é invocado o método *StartGame* do respectivo contrato e a partir deste momento, o serviço de jogo está pronto a ser utilizado.



**Figura 1 - Início de jogo com o tabuleiro de 20:10**

### 2.1.2 Regras e funcionamento de Jogo

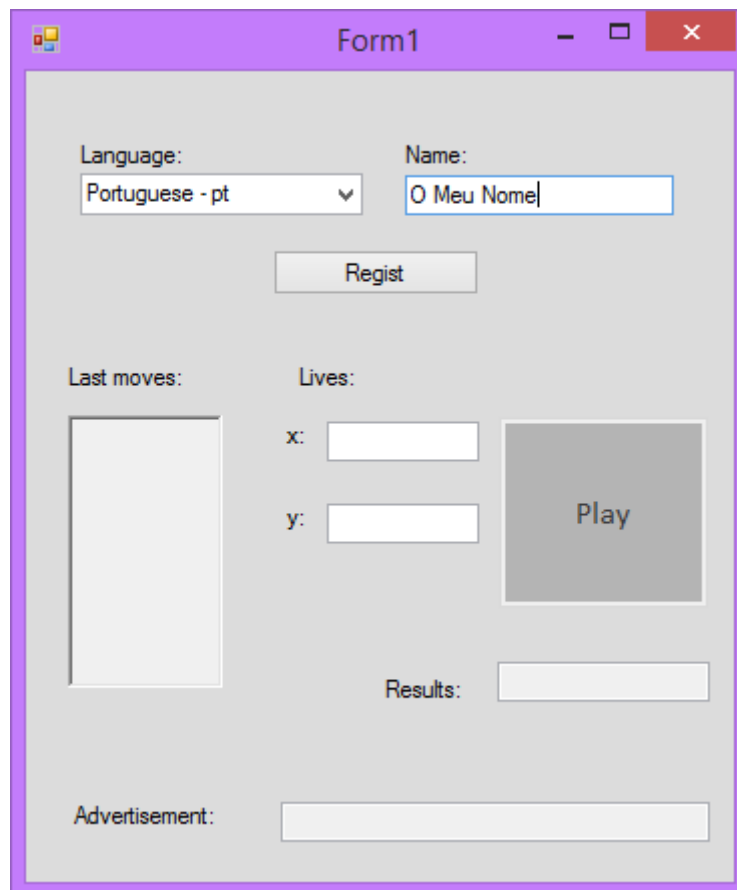
Cada jogo é composto por um Board com  $n$  por  $n$  Cells. Cada Cell tem características que influenciam a duração do jogo. Uma destas possui o tesouro, que ao ser descoberto por um dos jogadores faz com que seja vencedor, e de seguida o jogo notifica todos os outros participantes que perderam. Além do tesouro as Cells ainda podem ser zero, vida ou morte. Estas, ao contrário do tesouro, podem estar repetidas por todo o tabuleiro de uma maneira aleatória.

Cada jogador tem uma contagem inicial de três vidas, que quando chega a zero significa a derrota do participante. A casa morte decrementa o contador de vidas, enquanto que a casa vida o incrementa. A característica zero não influencia nada no jogo.

Depois de feita a jogada pelo *Player*, é modificado o conteúdo da *cell* (casa) de uma forma aleatória, de maneira a não permitir ao jogador coleccionar um número muito elevado de vidas, caso ele descubra que lá estejam. Assim, se o jogador apostou na casa 1:1 e ganhou uma vida, é feito reset a esta casa e na próxima jogada pode conter morte, outra vida ou nada.

### 2.1.3 Registo do *Player* e como se joga

Foi criada uma aplicação *GamePlayer* que utiliza o contrato *IGamePlayer* do *GameService* e é usada pelos jogadores. Quando esta aplicação corre pela primeira vez, o jogador não fica imediatamente apto a jogar, visto que necessita de se registar no jogo. Para isso deve inserir o seu nome e a sua língua (Figura 2) e clicar no botão *Regist*.

The image shows a Windows-style application window titled 'Form1'. Inside the window, there is a registration form. At the top, there are two labels: 'Language:' and 'Name:'. Under 'Language:', there is a dropdown menu currently showing 'Portuguese - pt'. Under 'Name:', there is a text input field containing 'O Meu Nome'. Below these fields is a button labeled 'Regist'. Further down, there are two more labels: 'Last moves:' and 'Lives:'. Under 'Last moves:', there is a large, empty rectangular box. Under 'Lives:', there are two small text input fields labeled 'x:' and 'y:'. To the right of these is a large, grey button labeled 'Play'. Below the 'Lives:' section, there is a label 'Results:' followed by a text input field. At the bottom of the form, there is a label 'Advertisement:' followed by a wide text input field.

**Figura 2 - Registo do utilizador**

Depois de ter sido feito o registo, o jogador está apto a jogar. Deve indicar o valor do x e do y que corresponderão à casa onde o jogador quer apostar, podendo clicar assim no botão *Play*. À esquerda serão registadas as últimas jogadas e na *TextBox* de *Results* o resultado da jogada feita. Poderá ver um exemplo de uma jogada feita na figura 3. Neste caso o jogador apostou em 1:1 e não lhe calhou nada.



**Figura 3 - Jogada feita em 1:1**

Caso o jogador efectuar uma jogada sem o Gestor ter iniciado o jogo, é lançada uma exceção. Essa exceção é do tipo *GameNotStartedException*, como é definido no método *MakeMove* através do atributo `[FaultContract(typeof(GameNotStartedException))]`. Caso o board não estiver inicializado, essa exceção é lançada pelo Jogo. O *Player* neste caso deve capturar a exceção e avisar o jogador em questão sobre a sua ocorrência.

Esta aplicação aloja também um serviço de notificações, criando uma classe *Receiver* que implementa a interface *IGamePlayerCallback* definida no contrato. Esta parte irá ser explicada em detalhe na Secção 2.2.

## 2.2 Serviço de Notificação

Para que fosse possível notificar os jogadores quando o jogo termina, foi desenvolvido um serviço de notificação baseado em *callbacks*.

Foi necessário criar um contrato *Duplex Service*, que permite uma troca de mensagens em que ambos os *endpoints* (neste caso, jogador e jogo) podem enviar mensagens um para o outro de forma independente.

Para criar este contrato foi necessário criar duas interfaces. A primeira (*IGamePlayer*) é a interface do contrato do serviço que descreve as operações que um cliente (jogador) pode invocar. No atributo desta interface é especificado o contrato de *callback* através da propriedade *ServiceContractAttribute.CallbackContract*.

A segunda interface (*IGamePlayerReceiverCallback*), especifica o contrato de *callback* e define as operações que o serviço pode chamar no *endpoint* do cliente.

O jogador para ser notificado pelo jogo tem de implementar a interface de *callback* (*IGamePlayerReceiverCallback*) do contrato duplex referido anteriormente. No momento em que se cria uma instância do jogo, é necessário enviar no construtor uma classe *InstanceContext*. Esta classe é usada para lidar com as mensagens enviadas do serviço Jogo para o Jogador na interface de *callback*.

A configuração do serviço jogo foi configurada para proporcionar um *binding* que suporta a comunicação duplex. O *wsDualHttpBinding* suporta este tipo de contrato fornecendo conexões http duplas, uma para cada direção.

```
<endpoint address="" binding="wsDualHttpBinding"
    contract="GameService.IGamePlayer" />
```

## 2.3 Serviço de Tradução

Relativamente aos anúncios dispostos na interface do jogador, estes são organizados pela *GameManager*. Estes anúncios são originalmente introduzidos numa dada língua. Mas tem de haver uma tradução para a língua que é seleccionada pelo jogador na *GamePlayer*.

Esta tradução acontece no serviço *Game*. Este, utilizando o Microsoft Translator, é encarregue de identificar a linguagem em que o anúncio está escrito, e recebendo a linguagem do jogador como parâmetro, faz a tradução do anúncio para dispor na interface do jogador.

Para a utilização deste serviço são criados um *biding*, um *endpoint address* e um canal utilizando este dois. Após a criação do canal de ligação, consegue-se ter acesso aos métodos do Microsoft Translator.

Tendo a tradução concluída, é utilizado a serviço *IGamePlayerReceiverCallback* para fazer notificar o jogador com o anúncio traduzido.

### 3. Conclusões

Concluimos que este trabalho foi determinante para a compreensão dos serviços WCF. Foi importante com base no que era pedido, decidirmos que tipo de configuração era ideal o serviço ter para suportar os diferentes pontos do trabalho. Por exemplo, ter contratos diferentes para os diferentes tipos de clientes do serviço, o facto de termos de configurar o serviço com diferentes tipos de binding conforme o modo de comunicação dos clientes com o serviço, e até mesmo a utilização de *web services* externos para fazer as traduções de linguagem.

As maiores dificuldades que encontramos no trabalho foram as decisões ao nível da configuração do serviço conforme os problemas que iam surgindo.