# Sistemas Distribuídos

**Projecto Prático:**

Sistema de Gestão de Jogos *Frogger* Distribuídos

**Rui S. Moreira & Ivo Pereira**

**@ FCT/UFP**

**Março 2022**

**Versão 2.0**

# 1. Sistema de Gestão de Jogos *Frogger* Distribuídos

## 1.1. Descrição do Projecto e Requisitos

Este projecto tem dois objectivos principais: primeiro, enriquecer os conhecimentos e a familiaridade dos alunos em relação aos modelos e algoritmos utilizados na gestão de sistemas distribuídos; segundo, aprofundar o conhecimento e a prática dos alunos em relação a algumas das ferramentas de desenvolvimento e gestão de sistemas distribuídos.

Em particular este projecto propõe o desenvolvimento de um sistema para gerir a criação e a execução de vários jogos multijogador distribuídos semelhantes ao tradicional *Frogger*.

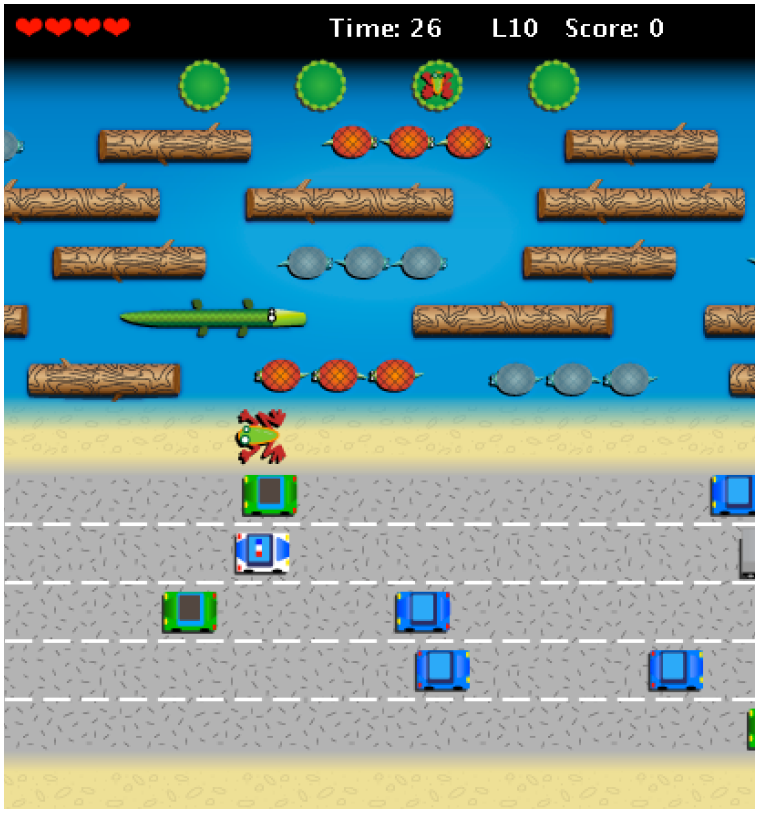


**Figura 1 – Exemplo do *FroggerGame*.**

O *Frogger* é um jogo arcade criado pela SEGA na década de 80. O jogador controla os movimento de um Sapo (*Frog*) que tem que subir o ecrã, atravessando uma estrada onde circulam vários veículos e paralelamente múltiplos objetos que percorrem um rio. O objetivo do jogo é fazer o Sapo chegar ao topo onde o está o seu ninho, evitando os veículos e sem cair ao rio, dentro de um tempo limite (Figura 1). Sempre que o jogador completa 5 travessias, o jogo sobe de nível de dificuldade, de modo a aumentar a competição.

O sistema distribuído proposto deverá permitir a gestão e sincronização de vários jogos multijogador *Frogger* a decorrer simultaneamente, de acordo com os requisitos abaixo enumerados:

[R1] Cada jogador deve utilizar uma GUI local para contactar uma ***GameFactory*** remota de modo a poder-se registar no sistema com *email* e *password*. Posteriormente usará o mesmo serviço para entrar (login) no sistema com as credenciais definidas no registo;

[R2] Depois dos jogadores efectuarem a autenticação (*login*), acedem ao ***GameSession*** de entrada do sistema. Devem considerar a utilização de JWT na autenticação. Nesta área poderão visualizar os jogos ***FroggerGame*** já criados, os que estão a decorrer ou até criar novos jogos. Na criação de um novo jogo deve ser possível definir o nível (dificuldade) inicial. Poderão ainda obter uma referência (proxy) específica para um dos ***FroggerGame*** existentes;

[R3] Através do proxy para um ***FroggerGame*** já criado, o jogador pode realizar um conjunto de acções:

- Cada jogador pode associar-se (*attach*) apenas a um jogo ***FroggerGame*** de cada vez. Cada jogo exige a presença de 2+ jogadores (clientes) para se iniciar;
- Quando um jogador se junta a um ***FroggerGame***, deverá ser criada uma instância local do jogo para que esse jogador possa jogar e visualizar as acções dos outros jogadores. Cada jogador terá como personagem um Sapo com uma cor distinta dos outros jogadores;
- Depois de se juntar a um jogo, as acções de cada jogador (e.g., posição e movimentação, estado da pista, pontuação, etc.) devem ser sincronizadas com a instância do ***FroggerGame*** no servidor. Deve ser criada uma implementação usando o padrão *observer* com RMI e outra implementação usando o padrão *publish/subscribe* com RabbitMQ. **NB**: **é obrigatória a utilização de ambas as tecnologias.**

[R4] O sistema deverá contemplar pelo menos 3 instâncias do servidor para conseguir garantir tolerância a falhas. A informação dos utilizadores registados e ligados, bem como o estado dos diferentes ***FroggerGames*** deve ser replicado e sincronizado entre os servidores. O sistema deverá gerir automaticamente a persistência do estado dos ***FroggerGames***;

[R5] Deverá existir um front server que re-encaminha os jogadores (em *Round Robin*) para um dos 3 serviços de backend. Se um servidor falhar então os jogadores ligados a esse servidor deverão ser re-encaminhados/ligados a outro servidor possuidor de uma réplica actualizada do jogo, de modo a conseguirem continuar a jogar. Um servidor que recupera a sua operação após um crash, deve conseguir sincronizar o seu estado a partir da informação local e dos outros servidores.

[R6] O sistema deverá ter a sua arquitectura descrita com base em diagramas UML: i) diagramas de classes identificando as interfaces e classes necessárias; ii) os diagramas de sequências de mensagens para os cenários de utilização do sistema.

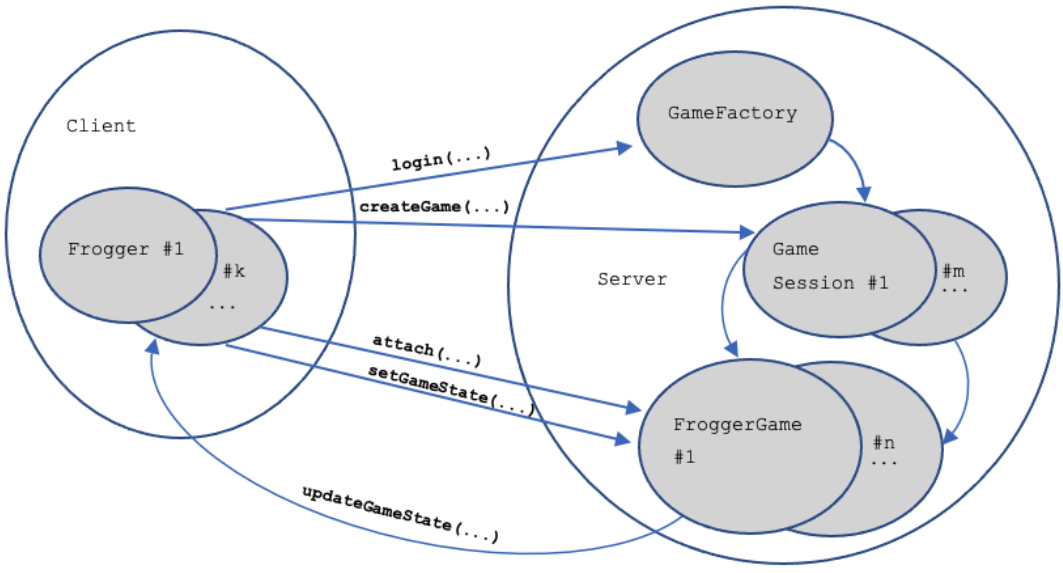


**Figura 2 - Esquema simplificado de instâncias do sistema distribuído.**

De acordo com o descrito, o principal objectivo será o desenvolvimento do jogo de forma distribuída. O sistema deve facilitar a organização, coordenação e persistência dos dados referentes aos vários grupos de jogadores que acedem de forma concorrencial aos mesmos recursos. A Figura 2 ilustra um esquema simplificado de instâncias do sistema distribuído. Os requisitos serão pesados de acordo com a Tabela 1.

**Tabela 1 - Pesos atribuídos a cada requisito**

| Requisitos | R1 | R2 | R3 | R4 | R5 | R6 |
|---|---|---|---|---|---|---|
| Cotação | 2 | 2 | 8 | 4 | 3 | 1 |

## 1.2. Tecnologia

Há vários padrões de design (***design patterns***[1]) que podem ajudar na organização da estrutura e do comportamento da aplicação distribuída solicitada. Por exemplo, o *observer design pattern* é muito utilizado para sincronizar vários *observers* que subscrevem um mesmo *subject*. Alternativamente o padrão *publish/subscribe* permite que vários *producers* comuniquem de forma assíncrona com vários *consumers*. através de filas de mensagens (*Message Queues*). O *factory method design pattern*, é também uma solução adequada para criar sessões de acordo com o perfil dos utilizadores. Outro padrão também muito útil é o *visitor design pattern*, que permite o envio/execução remota de operações numa estrutura de dados pré-definida (e.g., árvore de um sistema de ficheiros). Existem muitos outros padrões que poderão ser utilizados ou combinados na criação, organização e implementação do sistema.

O sistema proposto deve ser implementado recorrendo às tecnologias RMI (*Remote Method Invocation*) e RabbitMQ. **É obrigatória a utilização de ambas as tecnologias.** Os alunos são convidados a explorar soluções de *design* que melhor se coadunem à implementação dos algoritmos de coordenação, sincronização e gestão da aplicação distribuída. Devem dar atenção especial às questões de autenticação, segurança,

[1] URL: http://pages.cpsc.ucalgary.ca/~kremer/patterns/; URL: http://www.fluffycat.com/java/patterns.html

sincronização, coordenação e tolerância a falhas. Sugere-se que utilizem soluções ou algoritmos semelhantes aos abordados nas aulas.

## 2. Grupos, agenda e relatórios

Os alunos devem organizar-se em grupos de 3 elementos. Devem planear e dividir bem o trabalho de modo que todos participem na elaboração do mesmo. Estão previstos dois momentos de avaliação agendados na plataforma de *elearning*. Numa primeira fase deverá ser entregue um relatório preliminar descrevendo a arquitectura proposta para o sistema/serviço e os diagramas UML: i) diagramas de classes identificando as interfaces e classes necessárias; ii) os diagramas de sequências de mensagens para os cenários de utilização do sistema. Numa segunda fase deverá ser entregue um relatório final indicando a arquitectura adoptada, os diagramas UML finais e a implementação efectuada, discriminando os objectivos que foram cumpridos e aqueles que ficaram por resolver (indicando, eventualmente, a razão). Toda a documentação partilhada em ambas as fases de entrega deve ser efecuada através de um **projecto git** partilhado com os docentes.

Deverão adoptar uma arquitectura modular, faseando o trabalho e permitindo uma implementação incremental, resolvendo inicialmente os problemas mais fáceis e posteriormente atacando as situações mais complexas. Os alunos devem focar-se nos aspectos essenciais: aspectos de registo e autenticação, segurança, controlo e sincronização, replicação/tolerância a falhas e consistência/consenso.

Os alunos devem colocar o seu projecto IntelliJ no Github[2] de modo a facilitar a colaboração no desenvolvimento, de todos os elementos do grupo. Devem ainda partilhar o projecto do Github com os professores através dos seus emails (ivopereira@ufp.edu.pt, rmoreira@ufp.edu.pt).

[2] Github: https://github.com/