**Faculdade de Engenharia da Universidade do Porto**



**Biblioteca de Jogos**

**Trabalho AEDA – Parte 2 - 2018/19 - MIEIC**

**Coordenadores:** Luís Paulo Reis e Ana Paula Rocha

**Turma 2MIEIC05 Grupo 5**

**Professor Práticas:** Luís Paulo Gonçalves dos Reis

**Autores**

Gonçalo Oliveira, up201705494 ([up201705494@fe.up.pt](mailto:up201705494@fe.up.pt))

Manuel Coutinho, up201704211 ([up201704211@fe.up.pt](mailto:up201704211@fe.up.pt))

Mário Mesquita, up201705723 ([up201705723@fe.up.pt](mailto:up201705723@fe.up.pt))

Porto, 09 de janeiro de 2019

**Índice**

[**Descrição do tema do trabalho** 3](#_Toc530774202)

[**Solução Implementada** 4](#_Toc530774203)

[**Classes Implementadas** 8](#_Toc530774204)

[**Diagrama UML de classes** 10](#_Toc530774205)

[**Casos de Utilização** 11](#_Toc530774206)

[**Principais Dificuldades** 13](#_Toc530774207)

[**Esforço de Cada Membro** 15](#_Toc530774208)

# **Descrição do tema do trabalho**

O objetivo da segunda parte deste trabalho é o de acrescentar novas funcionalidades ao projeto previamente desenvolvido durante a primeira parte, a biblioteca virtual e multiplataforma de jogos de vídeo. Pretende-se que sejam usadas **estruturas de dados não lineares**, das quais árvores de pesquisa binária, filas de prioridade e tabelas de dispersão, assim como ilustrar operações básicas CRUD e listagens várias.

Em primeiro lugar, a biblioteca de jogos deve agora manter um registo de todas as **empresas** criadores de títulos que constam nela. Esta informação encontra-se guardada numa **árvore binária de pesquisa** organizada pelo número de títulos na plataforma e alfabeticamente. Cada empresa deverá possuir dados sobre a própria e estar associada aos seus títulos.

Seguidamente, pretende-se a criação de um sistema de **wishlists**, baseado numa **fila de prioridade** organizada por interesse. Os utilizadores passarão a guardar referências para os títulos que mais desejam comprar, sendo estas compostas pelo título em questão, o interesse do utilizador (de 1 a 10) e uma probabilidade representativa do quão provável se considera que o utilizador adquira o jogo se incitado a tal. Deve também ser possível selecionar, para cada utilizador, qual o item da sua wishlist que terá mais interesse em ser mostrada publicidade.

Finalmente, é pedida a implementação de uma **lista de ‘utilizadores adormecidos’** para cada título, baseada numa **tabela de dispersão**. Nas listas de cada título deverão constar os utilizadores que possuem uma probabilidade de compra acima de um valor residual mínimo. Estes utilizadores poderão ser removidos e readicionados conforme a sua atividade na biblioteca de jogos.

# **Solução Implementada**

**Registo de Empresas**

De modo a registar a informação de cada empresa na biblioteca de jogos, foi criada a classe “**Company**”. Esta classe possui o nome, NIF e contacto da entidade, assim como uma lista dos títulos criados pela mesma, que estão armazenados sob a forma de apontadores para referenciar as instâncias presentes na “**GameLibrary**”.

A listagem de todas as empresas encontra-se na classe “GameLibrary” sob a forma de uma árvore binária de pesquisa, organizada por número de títulos e alfabeticamente. Para este efeito foi usada a classe “**set**” da STL, que implementa uma forma de Árvore Vermelho-Preto.

É possível aplicar sobre esta lista de empresas as operações de criação, pesquisa, atualização e remoção, entre outras, cumprindo as operações básicas CRUD.

Implementamos também uma função que permite procurar pelo nome de *companies* recorrendo a uma wildstring, ou seja, uma palavra constituída por letras, \* (que correspondem a qualquer letra incluindo nenhuma ou várias) e ? (que faz de placeholder para qualquer letra), conseguindo assim fazer vários tipos de listagens e procuras alfabéticas.

**Sistema de Wishlists**

No sistema implementado cada utilizador possui a sua própria fila de prioridade representativa da sua wishlist. Esta contem objetos do tipo “**WishlistEntry”**, que possuem uma referência para o título que o utilizador deseja, um valor para o interesse de 1 a 10 e uma probabilidade de compra.

Esta probabilidade é calculada com base na similaridade dos outros jogos do utilizador a esse título, nos amigos que têm esse jogo na sua biblioteca, no número de cliques em publicidade do título e no número de pesquisas. A cada um destes fatores é atribuído um coeficiente, obtendo assim uma função linear de várias variáveis. Após calcular o respetivo valor da função, o resultado passa por uma função *sigmoid* para a distribuir no intervalo 0 < p < 1. Esta parte foi inspirada na estrutura de uma rede neuronal com uma função de ativação (neste caso a *sigmoid* porque pretendíamos uma probabilidade) com a diferença de que os parâmetros têm de ser ajustados manualmente, quer os coeficientes da função com diferentes variáveis quer da função ativação (esta normalmente é centrada na origem com ordenada a 0.5 e sobe rapidamente de 0 para 1 fazendo destes valores assintotas, a nós não nos interessava uma inclinação tão grande logo o *e* também é dividido por um e é somada uma constante para a deslocar para a direita no eixo dos xx).

As filas de prioridade são organizadas pelo interesse do utilizador e, em segundo caso, pela probabilidade de compra. É possível aplicar todas as operações básicas CRUD, assim como obter o próximo título que deverá ser apresentado ao utilizador de modo a conduzi-lo a uma transação. Visto que a probabilidade é calculada internamente, não pode ser alterada pelo user.

**Lista de ‘Utilizadores Adormecidos’**

Este foi o ponto mais complicado do trabalho, não por ser tecnicamente difícil, mas por causa do seu enunciado e da compatibilização de tudo o que nos era pedido (prova disso é que as funções para cada objetivo específico foram todas feitas rapidamente, tendo demorado apenas a interface que as teria de juntar e fazer delas sentido).

Tendo cada user o seu endereço de email, a hash function para a tabela de dispersão é feita tendo este por input. Começamos por tentar desenvolver nós a nossa própria função, mas, após lermos sobre o assunto, chegamos à conclusão de que esta não era a melhor ideia e, depois de alguma pesquisa, acabamos por escolher um algoritmo chamado djb2 segundo o nome Daniel J. Bernstein, que consiste em multiplicar cada letra por 33 e somar uma constante. O número 33 deve-se não só ao facto ser ímpar (que permite uma melhor distribuição que pares), mas também pela facilidade de o computar (bastam um shift e uma adição) tornando um algoritmo com uma rapidez bastante razoável em strings.

Como nada é diretamente dependente destas tabelas, e como o seu custo de atualização é alto (tem de se atualizar probabilidades, rever os users que ficaram ativos…) esta é como o gato de Schrodinger: até se abrir a caixa (até se quiser consultar a tabela que é o único propósito dela) ele está morto e vivo ao mesmo tempo. Com a melhoria de que quando a nossa caixa é aberta o nosso gato está sempre vivo.

Como estas tabelas são algo calculável a partir dos dados que temos optámos por não ocupar espaço desnecessário a guardá-la quando se faz save. Também consideramos que as operações CRUD não se aplicam aqui de forma direta, ou seja, o user não pode nem deve ser deixada a possibilidade de ele alterar a tabela. Existem funções de remoção claramente indicadas e transcritas a parte do enunciado a que dizem respeito e de adição, mas nunca para o utilizador.

**Outras Mudanças**

Além disto, foi também preciso adaptar algumas das coisas já existentes: adicionar um contador de publicidades vistas e pesquisas feitas (através de um tuplo) que vai sendo incrementado consoante se pesquise ou veja uma publicidade de certo jogo; tivemos de adicionar uma data interna à biblioteca para poder andar com ela e respeitar a condição de um jogador ser retirado se passado X meses não comprar um jogo; para verificar se algum jogo foi comprado nesse mesmo tempo foi ainda adicionado o ID do título às transações.

**Conceitos Utilizados**

* Classes
* Polimorfismo (overloading de funções, métodos virtuais, entre outros)
* Membros e métodos Static
* Exceções (lançamento e tratamento de diversos tipos)
* Árvores binárias de pesquisa
* Filas de prioridade
* Tabelas de dispersão
* Documentação do código usando Doxygen
* Leitura e escrita de dados em ficheiros

# **Classes Implementadas**

**Company**

A classe Company representa uma empresa na Biblioteca de Jogos, contento informações sobre a mesma e uma lista dos seus títulos.

**WishlistEntry**

A classe WishlistEntry representa uma entrada numa lista de desejos, contendo um apontador para o título desejado, um valor representativo do interesse do utilizador no título (de 1 a 10) e um float que representa o quão provável é o utilizador comprar o título se incitado a tal.

**Exceções**

Para o funcionamento da GameLibrary foram também implementadas várias classes representativas de exceções:

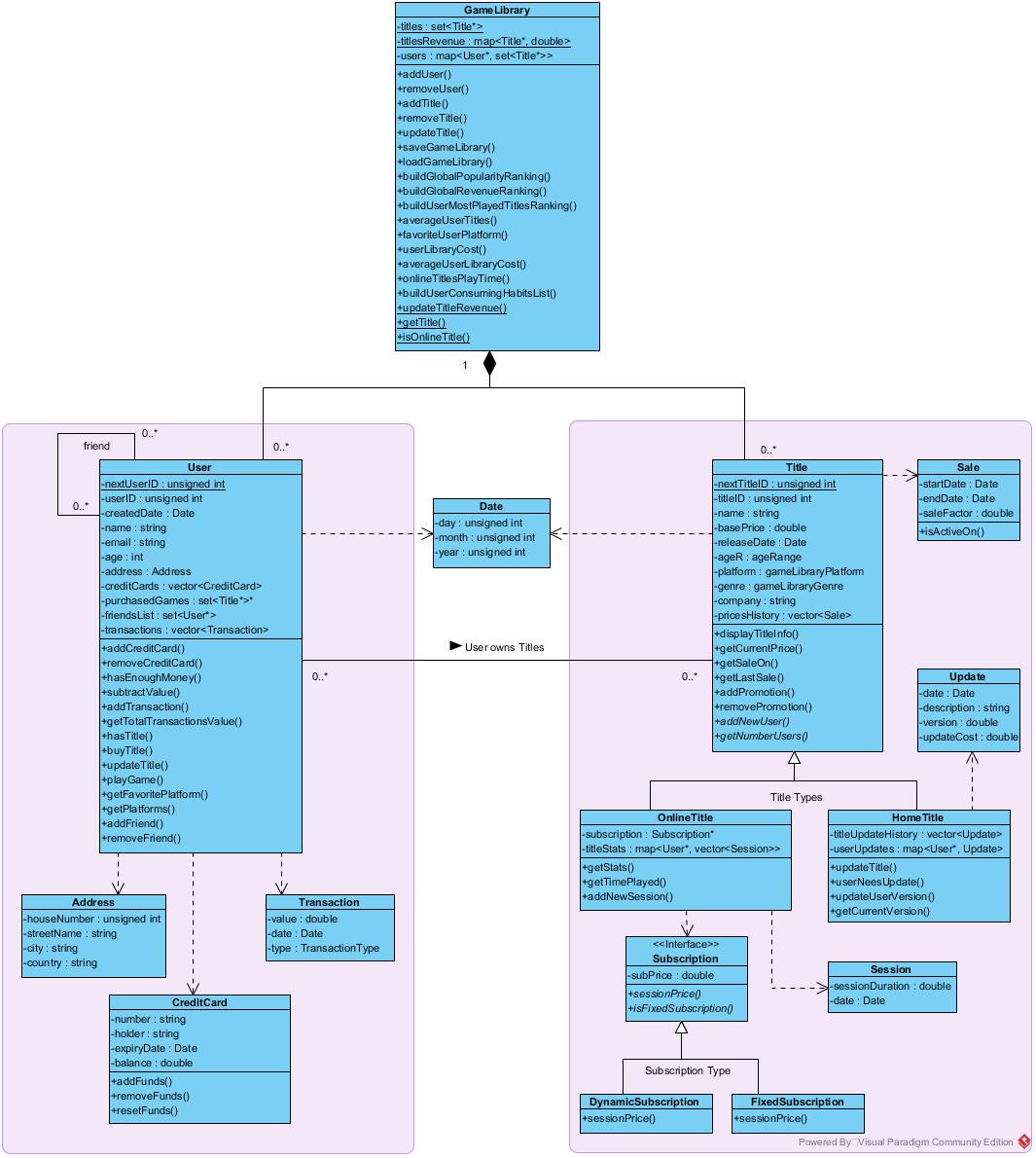
* NoMatchingWishlistEntry: lançada quando uma WishlistEntry com probabilidade de compra acima de um determinado valor não existe
* InvalidProbability
* NoMatchingWishlistEntryUser

**CompareObject**

Para o funcionamento da GameLibrary foram também implementadas várias classes representativas de exceçõ:

* CompareUsr:
* CompareCompany
* CompareCompanyByName

# **Diagrama UML de classes**



**Nota**: Foram implementadas várias exceções, especificadas na secção acima, que não constam no diagrama UML, de modo a não o sobrecarregar demasiado. Com o mesmo objetivo em mente, apesar da classe Data ser usada por muitas outras classes, apenas está indicada no diagrama como dependência de User e Title.

# **Casos de Utilização**

# **Principais Dificuldades**

# **Esforço de Cada Membro**

As contribuições individuais de cada membro foram as seguintes:

**Manuel Coutinho**

**Mário Mesquita**

* Criação do diagrama UML de classes
* Estruturas de dados implementadas:
  + Árvore de pesquisa binária
  + Fila de prioridade
* Classes implementadas:
  + Company
  + WishlistEntry
* Contribuições adicionais:
  + Exceções relativas às classes implementadas
* Documentação Doxygen do projeto
* Escrita do relatório

**Gonçalo Oliveira**

* Criação do diagrama UML de classes
* Contributos para a classe WishlistEntry
* Implementação de uma máquina de estados para escrita e leitura de ficheiros que representam o estado da biblioteca de jogos num dado instante.
* Desenvolvimento da interface entre o utilizador e o programa.
* Documentação Doxygen do projeto
* Escrita do relatório