U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Relatório 2º Projeto

Algoritmos e Estruturas de Dados

Biblioteca de Jogos

Turma 4 - Grupo 9

João Paulo Gomes Torres Abelha- up201706412@fe.up.pt

João Rafael Gomes Varela -up201706072@fe.up.pt

Vítor Hugo Pereira Barbosa -up201703591@fe.up.pt

Data de entrega: 9 / 01 /2019

Índice

1.Descrição sucinta do trabalho

1.1 Objetivo

A segunda parte do trabalho desenvolvido ao longo da segunda metade do primeiro semestre teve como objetivo de fazer a gestão das várias empresas criadoras de títulos, de "*wish lists"* e ainda de utilizadores "adormecidos". Para este efeito, são usadas estruturas de dados não lineares, nomeadamente árvores binárias, filas de prioridade e tabelas de dispersão.

1.2 Tema

O trabalho consiste numa extensão do primeiro, sendo o tema a gestão das várias bibliotecas de jogos, sendo, a esta, adicionada a gestão de informação sobre empresas, mediante determinados critérios. É criada uma *wishList* que contem títulos ordenados segundo o grau de interesse e de probabilidade de compra. Por último, ainda é gerido um conjunto de utilizadores adormecidos, isto é, aqueles que têm uma probabilidade de compra acima de um determinado valor e que não tenham comprado um título durante meses.

2.Solução implementada

2.1 Descrição sucinta da solução implementada

Com o intuito de continuar a fazer uma gestão eficiente e estruturada da Biblioteca de Jogos foram adicionados vários atributos a classes já existentes, por exemplo uma árvore binária na classe Sistema ou o número de cliques e de anúncios na classe "Titulo". Para além disto, foram ainda acrescentadas classes que consideramos relevantes, por exemplo a "*wishList"* que encapsula informação sobre o título, interesse e probabilidade de compra. Ainda são apresentadas no ecrã várias publicidades com base nos valores de métricas internas como a probabilidade e o interesse. Para cada título é usada uma *hash table* de utilizadores que serão lá adicionados sempre que forem considerados "adormecidos".

2.2 Alterações realizadas nas classes existentes

As classes existentes que foram alteradas foram a "Titulo", "Sistema", "Utilizador" e "Erro". Nestas foram acrescentadas atributos e novas funções para implementar aquilo que de novo era pedido.

Em primeiro lugar, na classe "Titulo", foram acrescentados o número de cliques e o número de anúncios, e funções que manipulem estes atributos como por exemplo a "adcionaAnuncios" que adiciona ao atributo "anuncios" um número de anúncios. Estes valores têm especial importância no cálculo aritmético da probabilidade de compra de um determinado título presente na *wishList* de um determinado utilizador. Nesta é criada a função hash, através da struct "UtilizadorPtr" para ser usada na hash table criada- typedef std::unordered_set<Utilizador*, UtilizadorPtr, UtilizadorPtr> UserHashTable. São ainda criadas funções que manipulam esta estrutura de dados, como adicionar e remover um utilizador.

Em seguida, na classe "Sistema", foi usada a libraria set para implementar a *binary tree,* BSTEmpresa, *(*que utiliza a struct EmpresasComp, para que as empresas sejam inseridas pela ordem pedida no enunciado: pelo número de títulos e, em caso de empate, pelo nome da empresa (alfabeticamente) -typdef std::set<Empresa *, EmpresasComp> BSTEmpresa). Foram implementadas funções para manipular e gerir esta estrutura de dados: adicionar títulos a uma dada empresa, adicionar uma empresa, procurar uma empresa e ainda optamos por criar ficheiros para cada empresa, onde a informação é atualizada (nome, email, contatos, NIF e títulos respetivos) quando terminamos o programa e onde é lida e passada para o sistema quando o compilamos. Nesta, ainda é de destacar a atualizaAsleepUsers(), que atualiza a hash table, mediante as possíveis mudanças que foram efetuadas ao longo do programa. Um utilizador será adicionado à tabela de dispersão correspondente caso a probabilidade de compra desse título seja superior a 60% e não compre um título há mais de 3 meses.

Na classe "Utilizador" foi adicionado um vetor constituído por uma fila de prioridade de objetos do tipo *wishTitle*, bem como funções que permitem o tratamento desta estrutura de dados, como adicionar, remover objetos deste tipo e ainda foram feitas funções que atualizam o interesse de

um utilizador num titulo e a sua probabilidade de compra, bem como uma função que mostra a próxima publicidade que aparecerá para esse utilizador, tendo em conta diferentes métricas internas. Note-se que a adição ou remoção de um título é feito com dados internos pelo que a *WishList* poderá estar a ser constantemente mudada, sendo que a remoção é realizada se o título for comprado ou se o utilizador assim o pretender. Nesta classe, foi ainda adicionado como atributo a última data de compra de um título, fator a ter em conta para saber se devemos inserir ou não um utilizador na hash table, sendo que foi considerado que um jogador sem quaisquer títulos não deverá ser adicionado a esta. Nas funções relativas a ficheiros foi adicionado a leitura e escrita das variáveis cliques e número de anúncios visualizados.

Por último, na classe "Erro" foram adicionadas funções que tratam os erros de forma apropriada, tornando não só o código mais robusto, mas também capaz de tratar casos que realisticamente não devem acontecer (como, por exemplo, tentar adicionar uma empresa que já existe).

2.3 Novas classes implementadas

Na segunda parte do projeto, foi implementada a classe "WishTitle" que tem o intuito de encapsular informação que serve para gerir as filas de prioridades que são usadas para incitar o consumo, uma vez que a partir de dados obtidos através desta são geradas publicidades. Esta classe possui uma referência para um título, um inteiro que representa o interesse do utilizador por esse título e ainda um *float* que representa a probabilidade de compra desse título. A informação contida nesta classe é guardada num ficheiro de texto que contem, para além destes dados, a restante informação do utilizador. É ainda de destacar que a probabilidade é atualizada sempre que pesquisamos um título e sempre que aparecem anúncios.

Para além disso, foi implementada a classe "Empresa" que possui como atributos um nome, um número de títulos, um NIF, um vetor de referências para os títulos e, por último, uma *struct* contactos que possui o email e o número de telemóvel da respetiva empresa. A gestão das empresas existentes é feita com o auxílio a uma árvore binária de pesquisa.

2.4 Outras considerações

No *main source file* são adicionadas novas opções ao menu com o objetivo de demonstrar novas potencialidades do programa, desenvolvidas na segunda parte do projeto.

Para o cálculo da probabilidade foi tido em conta o interesse, o número de anúncios visualizados pelo respetivo utilizador, o número de cliques (número de visitas) feitas ao título e ainda o saldo disponível. Para este cálculo os parâmetros referidos têm um peso distinto, sendo que consideramos 20% para o interesse, 20 % para a diferença relativa entre custo e saldo, 30% para o número de cliques e ainda 30% para o número de anúncios visualizados. A fórmula que foi construída é:

P("compra") = (Interesse * 2 + 30 *A / (A+10) + 30 * C / (C+5) + S / (5*S + 5*P)) / 100

A: número de anúncios

C: número de cliques

S: saldo do utilizador

P: preço do produto a comprar

Uma breve análise e explicação da fórmula: Como o interesse varia de 0 a 10 basta multiplicar-se por 2 para obtermos uma probabilidade até 20%. Tanto a parte da fórmula que trata do número de cliques e de utilizadores apresentam uma assintota horizontal em que a função se aproxima de 30% para valores muito elevados e repara-se que para zero cliques e zero anúncios esse valor será, naturalmente zero. Conclui-se, portanto lim(A->inf) = lim(C->inf) = 30 (pela parte inferior). Note-se que a constante no denominador para estes é diferente, resultando num aproximar mais rápido de 30 daquela que tem o número de cliques, sendo, deste modo, considerado que o número de cliques tem uma influência mais rápida que o número de publicidades, uma vez que se aproxima mais rapidamente de 30. Isto é feito desta forma, porque os cliques são voluntários enquanto que as publicidades refletem apenas uma estratégia de marketing.

Por último, a fórmula para cálculo da percentagem que relaciona o saldo com o preço do valor a pagar aproxima-se mais dos 20% quanto maior a amplitude entre o saldo e preço do produto a comprar.

Outro algoritmo que merece alguma atenção é aquele que é usado na função hash. É usada a função Bernstein hash djb2. Esta usa o número 33 (que acaba por ser, neste caso, melhor que outras constantes, sejam elas números primos ou não). A fórmula recorrente que este algoritmo usa sobre uma cadeia de caracteres é hash = (hash << 5) + hash + c, sendo que h = 5381 (note-se que se trata de um número primo) e que c corresponde ao ASCII code da letra atual no ciclo, sendo esta a variável responsável pelos diferentes valores (repare-se ainda que parte da singularidade na função hash é conseguida também pelo facto de os emails não poderem ser iguais). Esta função garante que as colisões são raras e que há uma boa distribuição.

No decorrer do programa, para tornar a situação realista, mal o código é compilado aparecem várias publicidades, correspondentes ao topo da *wishList* dos utilizadores do sistema com uma probabilidade de compra acima dos 50%. Quando é pesquisado um jogador específico, aparece a publicidade do título que está no topo da sua *wishList*. É ainda importante notar que o aumento do número de cliques está diretamente relacionado com o número de pesquisas que são feitas a esse título. Esta mecânica feita para as publicidades serve para instigar os utilizadores a comprar na época natalícia, fazendo com que a sua probabilidade de compra aumente.

3.Listagem de casos de utilização

3.1 Listagens adicionadas que recorrem à árvore binária

As listagens que recorrem a esta estrutura de dados armazena as várias empresas existentes, ordenando-as, em primeiro lugar, pelo número crescente de títulos e, em seguida, pelo seu nome.

No menu principal foram acrescentadas as seguintes opções:

7. “Adicionar Empresa”

8. “Pesquisar Empresa”

9. “Visualizar Empresas”

Na opção 8 é lançada uma exceção, caso não exista, senão é aberto um outro menu onde é possível adicionar títulos, visualizar contactos e visualizar os títulos.

Na opção 9 é possível visualizar as empresas, pela ordem em que se apresentam na *binary tree,* pelo número de títulos na plataforma, pelo número de títulos numa plataforma e ainda por género. Por último, é possível no menu dos títulos (primeira opção do menu principal) ver as empresas ordenadas por ordem alfabética (de forma crescente e decrescente).

3.2 Listagens adicionadas para à tabela de dispersão

As listagens que recorrem a esta estrutura de dados armazena os vários utilizadores adormecidos, sendo a posição de cada uma determinada pela função de dispersão.

No menu principal, foi adicionada a opção:

10. “Visualizar Utilizadores Adormecidos”

Nesta é aberto um segundo menu que permite ver os utilizadores presentes nessa tabela, os utilizadores lá presentes para uma determinada plataforma, para um dado título e ainda género. Os utilizadores são removidos sempre que a probabilidade de compra seja menor que 60 % ou que o título para o qual tem uma probabilidade de compra superior a 60% seja adicionado à sua biblioteca.

3.3 WishList

Para as wishList não foram geradas listagens, uma vez que não são pedidas, contudo, é possível ao utilizador no menu utilizador (opção 4 do menu principal):

2: adicionar uma wishList

3: remover uma wishList

4: alterar o interesse de um título

4.Dificuldades encontradas no decorrer do trabalho

No planeamento do projeto e na sua implementação não houve grandes dificuldades, contudo, é de destacar o último ponto do enunciado, uma vez que consideramos estar um pouco confuso, gerando dificuldades em entender aquilo que realmente é pedido. Tivemos de ponderar e discutir vários métodos e algoritmos a ser implementados, bem como decidir quando deveríamos criar ou adicionar novas funções ou atributos a classes já existentes.

5.Esforçado dedicado por cada elemento do grupo

O grupo tentou distribuir de forma igualitária o trabalho por todos os elementos, sendo que trabalhamos sempre que possível em conjunto de forma a discutir vários métodos, tendo todos contribuido para todos os módulos criados. Deste modo, todos os elementos participaram de forma ativa, escrevendo não só o código, mas discutindo várias ideias, por exemplo, aquilo que manter, mudar ou acrescentar à primeira parte do projeto.

No geral, consideramos que o resultado foi positivo, uma vez que todos os membros conseguiram cooperar, havendo uma entreajuda que permitiu terminar de forma bem-sucedida o produto final.