

**MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA E COMPUTAÇÃO**

**CASTINGS TV**

Antero Gandra, 201607926 up201607926@fe.up.pt

Bruno Vale Fernandes, 201 up201606214@fe.up.pt

Margarida Silva, 201606214 up201606214@fe.up.pt

**RESUMO**

Com este trabalho pretendemos demonstrar como é possível implementar um sistema de castings de uma empresa usando os diversos tipos de estruturas de dados.

De modo a abordar corretamente e organizadamente o jogo, separamo-lo em vários módulos diferentes para implementar: a empresa, as pessoas e as audições (cada um com os seus respetivos ﬁcheiros) .

A nossa maior diﬁculdade foi em saber como implementar xxxxxxxxxxxxxx sendo que o que nos deu mais problemas foi xxxxxxxxxxxxxxxxxxxxxxx. Acabamos por optar por xxxxxxxxxxxxxxxxxxxxxxxxxxxx.

Outro grande desaﬁo foi xxxxxxxxxxxxxxxxxxxxxxxxxxxxx para xxxxxxxxxxxxxxxxxxx. Perante este problema, a solução passou por xxxxxxxxxxxxxxxxxxxxxxxxxxx.

Deste modo conseguimos planificar o funcionamento correto da empresa e implementá-lo, de modo a que a experiência do utilizador com o nosso programa seja simples, organizada e o mais eficiente possível.

**ÍNDICE**

**DESCRIÇÃO DO PROBLEMA:**

Uma empresa do ramo televisivo pretende criar um sistema de planeamento de castings.

De modo a fazer um casting organizam-se sessões com 3 júris, sendo que cada sessão é baseada numa só arte performativa, sendo que o jurado responsável se considera especialista nessa área.

Por sessão são necessárias duas fases sendo que na primeira, a pontuação dos participantes é decidida com igualdade pelos 3 jurados. Já na segunda fase o peso da pontuação do júri responsável da sessão vai valer o dobro. Nesta fase só participam os 5 melhores concorrentes.

**REQUISITOS/RESTRIÇÕES:**

As pontuações variam entre um a dez.

O número mínimo de participantes por sessão é seis (de modo a, por questões logísticas, haverem duas fases).

O tempo por participante na primeira fase são quinze minutos e na segunda fase meia hora.

**DADOS:**

Por participante e por Júri -> id, nome, morada, número de telemóvel, morada, especialidade e participações anteriores.

Por sessão -> id, início, fim, especialidade, júri, líder.

**DIAGRAMA UML**

**GESTÃO DE PESSOAS**

A organização de um casting exige um controlo sobre quer quem é avaliado quer quem avalia. Deste modo criámos duas classes, Judge e Contestant (subclasses de Person).

De modo a gerenciar os jurados usamos as addJudge(), addNewJudge(), updateJudge(), removeJudge(), readJudgesFile() e writeJudgesFile() que tratam, como o próprio nome indicia, da adição, modificação e remoção de jurados na base de dados, acompanhadas da sua atualização em ficheiros de texto (ou não, consoante a vontade do utilizador!). Nota: addNewJudge() chama a função addJudge() após a determinação do id a atribuir a um candidato. A utilização de addJudge() por si só é destinada essencialmente à leitura de ficheiros de judges.

Para gerenciar os candidatos usamos um processo muito idêntico, com as funções addContestant(), addNewContestant(), updateContestant(), readContestantsFile() e writeJudgesFile(), embora vhkbjln

Para além de befeferfeefe usamos as as funções xxx e xxxxxx, com as quais acedemos e alteramos brgergerfvgefrvre.

**GESTÃO DE TEMPO**

**PROCESSAMENTO DAS FASES**

**SISTEMA DE CLASSIFICAÇÃO DOS CANDIDATOS**

**OUTRAS FUNÇÕES UTEIS**

**INTERFACE**

**DIFICULDADES NO DESENVOLVIMENTO**

A grande dificuldade deste trabalho foi a implementação da estrutura *Audition* (Sessão) e todas a operações relacionadas com ela.

Em primeiro lugar, o armazenamento dentro de ficheiros. Estas classes, constituídas por uma quantidade grande de atributos, alguns deles de tipos relativamente complexos (como é o caso das estruturas que representavam a Primeira Fase e a Segunda Fase da sessão), que por si só tinham uma representação em ficheiro própria. A leitura e escrita de Sessões foi trabalhosa, mas, julgamos, concluída com sucesso.

Por outro lado, a própria marcação de sessões, que envolve a verificação de todo um conjunto de critérios, desde um mínimo e máximo de candidatos, de um número exato de jurados, tendo, por nossa opção, o jurado responsável de ser obrigatoriamente da especialidade da sessão, de não haver duas sessões da mesma especialidade por dia, entre outros. Este processo complica-se bastante na criação manual de sessões.

Por fim, a implementação das classes *FirstFase* e *SecondFase* foi desafiante na questão a reflexão sobre como armazenar classificações, e se o deveríamos fazer ou apenas guardar a classificação final obtida pelo jurado responsável na informação da participação de cada Jurado. Por fim concluímos que seria proveitoso pois assim, a qualquer altura, a empresa tem esses dados prontos para analisar, se for o caso.

**FUNCIONAMENTO DO GRUPO E CONTRIBUÍÇÃO DE CADA ELEMENTO**

Antes da realização do código propriamente dito, foram realizadas várias reuniões entre os elementos do grupo em que o funcionamento genérico do programa ficou estruturado, tendo sido analisadas questões desde que classes incluir e sua futura interação, a como organizar os dados em estruturas práticas e fáceis de aceder. Todos os elementos se esforçaram por debater estas questões e chegar a um consenso, embora o Bruno Fernandes e a Margarida se tenham destacado na tomada de opções. Ao longo deste processo foi construído inicialmente um diagrama UML simplista, apenas para servir de guia à implementação do código, e por forma a garantir um resultado final coerente.

Posteriormente, passando à parte “prática” da realização deste trabalho, os elementos passaram a contribuir de forma mais individual, unindo o trabalho desenvolvido na plataforma *github*, que auxiliou o desenvolvimento independente dos elementos. Repositório: <https://github.com/Antero-Gandra/AEDA>

De qualquer modo apresentam-se listas das tarefas executadas por cada elemento do grupo tendo em conta o sucesso do projeto.

Antero Gandra:

* Geração de dados para os ficheiros de teste (contestants.dat e judges.dat).
* Estruturação do Relatório, e preenchimento das secções relacionadas com os dados do tema do trabalho e especificações.
* Implementação de métodos “set” e “get” de algumas classes.

Bruno Vale Fernandes:

* Aplicação de alguns testes ao código para deteção de bugs.
* Contribuição no aspeto visual de alguns menus.

Margarida Silva:

* Estruturação e desenvolvimento da totalidade da codificação presente no projeto, excetuando, como é óbvio, as contribuições dos outros elementos mencionadas acima.
* Contribuição no preenchimento do corpo do Relatório, nomeadamente na secção da contribuição dos elementos do grupo, das dificuldades na implementação,

**CONCLUSÃO**

Com este projeto conseguimos implementar de uma maneira interativa e interessante os conceitos aprendidos em aula. Aplicamos várias dos diversos tipos de algoritmos na resolução de problemas mais complexos e empregamos estruturas de dados ensinadas nas aulas. Deste modo adquirimos conhecimentos na área da eficiência algorítmica e na área de planeamento envolvido na organização de castings.

**RECURSOS**

* Slides das aulas teóricas
* http://www.cppreference.com/