

**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 (“Company”), o gestor da empresa (“CompanyMS”), as pessoas e as audições (cada um com os seus respetivos ﬁcheiros).

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 -------------------------------------------------------------------------- **4**

Requisitos/Restrições -----------------------------------------------------------------------------**4**

Lista de casos de utilização-----------------------------------------------------------------------**5**

Diagrama UML ----------------------------------------------------------------------------------- **7**

Gestão de Pessoas -------------------------------------------------------------------------------- **8**

Gestão de Tempo --------------------------------------------------------------------------------- **9**

Processamento das fases -------------------------------------------------------------------------10

Sistema de Classificação de Candidatos -------------------------------------------------------10

Outras Funções Uteis ---------------------------------------------------------------------------- **10**

Interface ------------------------------------------------------------------------------------------- **10**

Dificuldades no Desenvolvimento ------------------------------------------------------------- **11**

Funcionamento do Grupo e Contribuição de cada elemento-------------------------------- **12**

Conclusão ----------------------------------------------------------------------------------------- **14**

Recursos ------------------------------------------------------------------------------------------- **14**

**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 (especialidade).

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 face os seus colegas, na avaliação dos concorrentes. Nesta fase só participam os 5 melhores concorrentes da primeira fase.

**REQUISITOS/RESTRIÇÕES:**

* As pontuações variam entre um a dez.
* O número mínimo de participantes na primeira sessão é seis (de modo a, por questões logísticas, haverem duas fases).
* Na segunda fase só podem participar os 5 melhores candidatos da primeira sessão.
* Não podem haver sessões da mesma categoria no mesmo dia, devendo variar dia sim dia não.
* O tempo por participante na primeira fase são quinze minutos e na segunda fase meia hora.
* Uma pessoa não pode alterar a sua especialidade, quer seja júri quer seja candidato.

**LISTA DE CASOS DE UTILIZAÇÃO**

Desenvolvemos uma aplicação onde é possível:

- Adicionar candidaturas (de candidatos já existentes ou novos);

- Modificar um candidato;

- Eliminar um candidato;

- Eliminar as candidaturas de um candidato;

- Ver listagens parciais de candidatos ordenados por Id, ou a listagem total;

- Ver a lista de candidatos ordenados por nome;

- Ver os candidatos de uma especialidade;

- Ver a lista de candidaturas;

- Procurar um candidato por id;

- Adicionar (empregar) um jurado;

- Modificar um jurado;

- Ver listagens parciais de jurados ordenados por Id, ou a listagem total;

- Ver os júris ordenados por nome;

- Ver os jurados de uma especialidade;

- Procurar um jurado por id;

- Sincronizar todas a informações com a base de dados (lendo e escrevendo em ficheiros);

- Marcar uma audição de uma especialidade (automaticamente);

- Marcar, dadas as candidaturas existentes, o máximo de audições de uma especialidade;

- Marcar, dadas as candidaturas existentes, o máximo de audições de todas as especialidades;

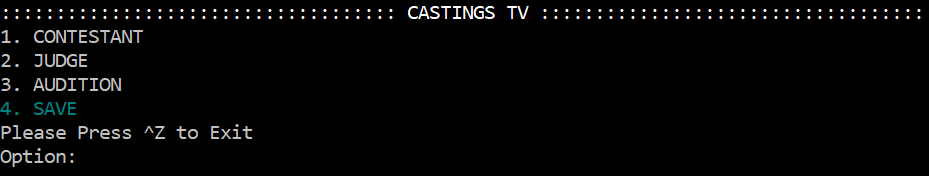
-Apagar uma audição;

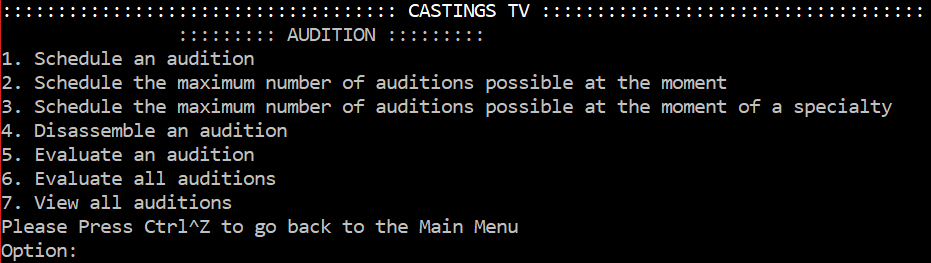
-Avaliar uma audição (automaticamente);

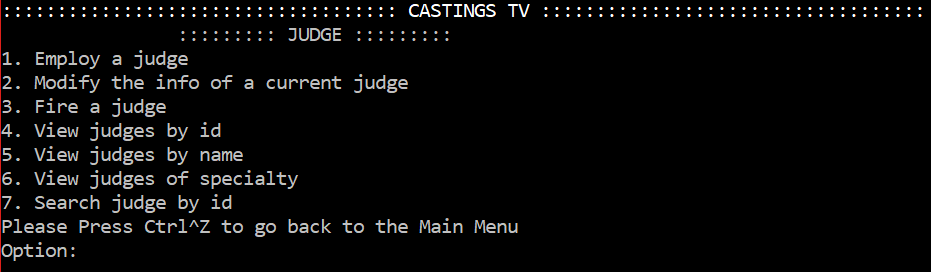
-Avaliar todas as audições

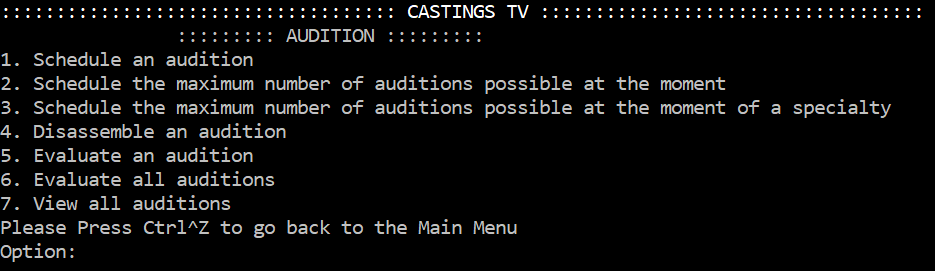
-Ver todas as audições

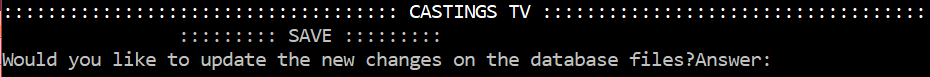
Tal como se vê nas seguintes imagens:











**DIAGRAMA UML:**



Este é o diagrama que, julgamos, melhor ilustra a nossa implementação.

Visto que as ligações não estão muito percetíveis, o ficheiro .png segue em anexo na pasta que foi entregue na submissão.

**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 (Júri) e Contestant (candidato), ambas subclasses de Person.

**Classe Judge**

De modo a gerenciar os jurados usamos as funções addJudge(), addNewJudge(), que adicionam jurados, e removeJudge(), para os remover.

Além disso criámos também updateJudge(), para que seja possível modificar as caraterísticas dos juízes, e os métodos readJudgesFile() e writeJudgesFile() que tratam, como os nomes indicam, da atualização em ficheiros de texto dos ficheiros da base de dados do programa.

**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.

**Classe Contestant**

No processo de gerir os candidatos usamos funções que atuam de maneira semelhante, uma vez que quer candidatos quer jurados variam da mesma classe pessoa.

Para isso criaram-se as funções correspondentes: addContestant(), addNewContestant(), updateContestant(), readContestantsFile() e writeJudgesFile() .

Com a adição de contestants, é normalmente adicionar candidaturas. Para isso usam-se as funções addAplication().

**GESTÃO DE TEMPO**

O tempo tem sempre grande influência na gestão dos projetos. Para isso vamos fragmentar este tema pelo seu uso nas classes:

**Classe Calendar:**

De modo a haver uma correta manipulação do tempo, no geral, de uma maneira mais básica, foram criados os métodos set (setYear, setMonth, setDay, setHour, setMinute) e get (getYear, getMonth, getDay, getHour, getMinute) que basicamente definem e acedem ás variáveis correspondentes.

Também foi criada nesta classe o método isValidDate, que testa se uma data tem os elementos correspondentes devidamente aplicados/válidos.

**Classe Company:**

O problema que surge na manipulação das sessões é: como é que existe um agendamento correto das sessões se o número de candidatos varia de sessão em sessão?

Para isso criou-se a função getDurationofAudition() que calcula a duração necessária para uma sessão que tem n candidatos (sendo n passado como argumento).

**PROCESSAMENTO DAS FASES**

Uma audição começa por inicializar a sua primeira Fase com os contestants que estão inscritos. Só com o posterior processo de avaliação é que a segunda fase vê os seus contestants inicializados. No final da primeira fase, todas os participantes que não passaram têm o seu vetor de participações aumentado, indicando que na sessão em questão ficaram na posição “0”. No final da segunda fase, o mesmo é feito, sendo que a posição varia de 1 a 5, neste caso.

**SISTEMA DE CLASSIFICAÇÃO DOS CANDIDATOS**

Optámos por gerar classificações aleatórias aquando da avaliação das sessões.

Guardamos a classificações finais como doubles, embora as participações dos Contestants depois só tenham o id da sessão, a classificação do jurado principal, e a posição em que ficaram. Se não conseguiram chegar à segunda fase, assumimos que o participante ficou no lugar “0”.

**OUTRAS FUNÇÕES UTEIS**

Na remoção de audições foi uma preocupação nossa que todas as participações dos jurados e candidatos envolvidos fossem retiradas. Como se nada tivesse acontecido.

**INTERFACE**

O utilizador pode “saltar” de menu em menu, ora selecionando o próximo ora escrevendo “^Z” (que pode ser feito com cntrl+Z) para voltar ao menu anterior. Em qualquer altura o utilizador pode escrever ^Z, mesmo quando aquilo que lhe é solicitado inserir (Ex:nome) não é isso.

**DIFICULDADES NO DESENVOLVIMENTO**

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

* O armazenamento 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.
* A 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.
* 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**

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.

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>

**CONTRIBUÍÇÃO DE CADA ELEMENTO**

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.
* Documentação Doxygen

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, interface, gestão de pessoas, lista de casos de utilização, processamento de fases, sistema de avaliação dos candidatos e outras funções úteis.
* Diagrama UML

**CONCLUSÃO**

Com este projeto conseguimos implementar de uma maneira interativa e interessante os conceitos aprendidos em aula. Aplicamos vários 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/