AIAD - MIEIC - FEUP

Alocação de alunos de 2ª fase em turmas conforme paridade e ocupação da sala

Ano letivo 2020/2021

Profª Ana Paula Rocha

Alunos:
Henrique Freitas, 201707046
José Gomes, 201707054
Liliana Almeida, 201706908

Descrição do problema

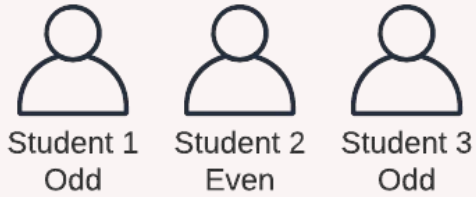
Atribuição de alunos que entraram no ensino superior na 2ª fase de colocação em turmas já existentes.

O problema inclui variáveis como a ocupação da turma, a capacidade da sala e a paridade do aluno (considerando o problema que surgiu com a pandemia do Covid-19)

Para onde vai cada aluno?

Agentes - *Student*

Responsável por contactar todas as turmas, receber a *utility* de cada uma delas e fazer um *request* para ser atribuído à que mais se adequa. Tem como único atributo relevante a sua paridade (Par ou Ímpar).



Agentes - *CUClass*

Responsável por calcular a *utility* tendo em conta a paridade do aluno que a contacta, e aceitar/recusar os *requests* de atribuição colocados pelos alunos e informar os alunos sempre que há alteração da *utility*. A fórmula de cálculo da *utility* é a seguinte:

$$if(even) \rightarrow \frac{roomCapacity - occupiedSeats}{\frac{evenStudents}{occupiedSeats}}$$

$$if(odd) \rightarrow (roomCapacity - occupiedSeats) * \frac{evenStudents}{occupiedSeats}$$

Class 1	Class 2
30 Seats	25 Seats
12 Occupied	13 Occupied
5 Even Students	7 Even Students

Interação e protocolos

As interações entre estudantes e turmas dividem-se em **duas** partes principais:

- **Subscrição da *Utility*:** os alunos mantém informação atualizada da utility de cada turma através do protocolo ***FIPA-Subscribe*** implementado através das classes *SubscriptionInitiator* e *SubscriptionResponder*.
- **Atribuição de estudantes às turmas:** Cada estudante inicia um ***FIPA-Request*** à turma na qual pretende entrar enviando a sua **paridade e a *utility*** que julga a turma ter.
  - Se a turma **concordar** com a *utility* que o aluno enviou, envia uma mensagem com a performativa **"*Agree*"** e inicia o processo de o **colocar na turma** (notificando depois os restantes alunos da mudança na *utility* na mesma).
  - Caso a *utility* enviada **não esteja de acordo** com a real, a turma envia uma mensagem com a performativa **"*Refuse*"** e **cancela a operação**. O aluno assim, espera pela **atualização da *utility* e tenta de novo**.

Estratégias utilizadas

O protocolo *FIPA-Subscribe* facilitou bastante a execução do algoritmo, pois garante que após a atualização de uma turma (devido a alocação de um aluno), todos os alunos não-alocados serão informados da alteração da *utility*, de modo a que estejam sempre atualizados em relação à melhor turma para ser alocado

Descoberta de agentes

A descoberta de agentes é feita através do *Directory Facilitator Agent* (*DFAgent*). Cada agente *CUClass* publica um serviço do tipo “Curricular Units” aquando da sua inicialização.

Quando um agente *Student* é inicializado, procura no *DFAgent* todos os agentes que publicitaram o serviço “Curricular Units” e assim obtém uma lista de todas as turmas disponíveis.

Análise de resultados

Case1 (classe em src/MainCases):

- 2 turmas iguais (rácio de paridade e ocupação)
- 5 alunos pares

Resultado: Cada aluno escolheu, de forma alternada, as turmas 1 e 2. O aluno 1 ao escolher a turma 1 diminuiu a *utility*, fazendo com que o aluno 2 escolhesse a turma 2, voltando a empatar a *utility*. O processo repete-se até todos os alunos estarem alocados.

Case2:

- 1 turma com rácio elevado de pares
- 1 turma com rácio elevado de ímpares
- 1 aluno ímpar
- 4 alunos pares

Resultado: O aluno ímpar escolheu a turma que tinha um rácio elevado de pares, e os alunos pares escolheram a turma com muitos ímpares. O esforço para equilibrar os rácios par/ímpar das turmas leva a que os alunos tenham preferência pela turma com rácio elevado da paridade oposta à sua.

Análise de resultados

Case3:

- 1 turma cheia
- 1 turma com ocupação média
- 1 turma com ocupação baixa
- 2 alunos ímpares
- 3 alunos pares

Resultado: Todos os alunos entraram na turma com menor ocupação, uma vez que todas tinham o mesmo rácio de paridade e assim o fator da capacidade pesou mais na decisão.

Case4:

- 1 turma com rácio elevado de pares
- 1 turma com rácio elevado de ímpares
- 1 turma com rácio equilibrado
- 2 alunos ímpares
- 3 alunos pares

Resultado: Semelhante ao teste Case2 (slide anterior), os alunos tiveram preferência pelas turmas extremas, numa tentativa de equilibrar os rácios das paridades das mesmas.

Conclusão

Tendo em conta os resultados previamente expostos, é possível verificar que o algoritmo funciona e aloca corretamente os alunos nas turmas.

O cálculo da *utility* não considera relevância entre o rácio de paridade e a ocupação da turma.

A execução é simples e não considera muitos fatores presentes na realidade, como várias UCs para cada aluno, turmas incompatíveis entre UCs, ou mesmo a possibilidade de alterar o peso das variáveis no cálculo da *utility*.

AIAD - MIEIC - FEUP

Alocação de alunos de 2ª fase em turmas conforme paridade e ocupação da sala

Informações Adicionais

Ano letivo 2020/2021

Profª Ana Paula Rocha

Alunos:
Henrique Freitas, 201707046
José Gomes, 201707054
Liliana Almeida, 201706908

Exemplos detalhados de execução

Tomando como exemplo o *Case1,* ao iniciarem, os alunos procuram as turmas e subscrevem à sua *utility* (linha 5). Quando obtém a subscrição de todas as turmas imprimem uma mensagem de confirmação (linha 7).

```
5: Agent uc2: Agrees Student3 subscription
6: Agent Student3 : uc2 agreed utility
7: Agent Student3 :  confirmed all subscriptions
```

De seguida, os alunos iniciam o processo de escolha das turmas, selecionando a que tem melhor *utility* (linha 10) e iniciando o processo de atribuição (linha 11). Caso tudo aconteça sem problemas, o output será parecido ao seguinte:

```
10: Agent: Student4 best Class: uc1 utility -> 40.0
11: Agent uc1: ASSIGN received from Student4
12: Agent Student4 : uc1 agreed to add
13: Agent uc1: Action successfully performed
14: Agent Student4 : uc1 added this student
```

Nota: As referências às linhas utilizadas nestes slides não correspondem exatamente à ordem da execução do código dada a sua natureza assíncrona

Exemplos detalhados de execução

No caso descrito em baixo, a *utility* que o aluno julgava a turma ter foi alterada a meio do processo de atribuição (linha 32). Assim, a turma recusa o pedido de atribuição (linha 33) e o aluno recomeça o processo com a nova *utility* (linha 34).

```
30: Agent: Student2 best Class: uc1 utility -> 40.0
31: Agent uc1: ASSIGN received from Student2
32: Agent Student2 : uc1 updated utility
33: Agent Student2 : uc1 said invalid utility
34: Agent: Student2 best Class: uc2 utility -> 40.0
```

Nota: As referências às linhas utilizadas nestes slides não correspondem exatamente à ordem da execução do código dada a sua natureza assíncrona

Exemplos detalhados de execução

Ao terminar a execução do programa de forma graciosa, cada aluno imprime a turma na qual entrou, a sua paridade e a *utility* que a mesma tinha.

```
40: Agent: Student2 best Class: uc2 utility -> 40.0
41: Student Student2 (EVEN) sent to class uc2
42: Agent: Student1 best Class: uc2 utility -> 34.833332
43: Student Student1 (EVEN) sent to class uc2
44: Agent: Student3 best Class: uc1 utility -> 30.857143
45: Student Student3 (EVEN) sent to class uc1
46: Agent: Student4 best Class: uc1 utility -> 40.0
47: Student Student4 (EVEN) sent to class uc1
48: Agent: Student5 best Class: uc1 utility -> 34.833332
49: Student Student5 (EVEN) sent to class uc1
```

Nota: As referências às linhas utilizadas nestes slides não correspondem exatamente à ordem da execução do código dada a sua natureza assíncrona

Classes Implementadas - Agentes

Student

Responsável por contactar todas as turmas, receber a *utility* de cada uma delas e fazer um *request* para ser atribuído à que mais se adequa. Tem como atributos a sua paridade (Par ou Ímpar), a *utility* das turmas existentes e o *behaviour* utilizado para subscrever à turma de forma a poder cancelar essa subscrição posteriormente.

Classes Implementadas - Agentes

CUClass

Responsável por calcular a *utility* tendo em conta a paridade do aluno que a contacta, e aceitar/recusar os *requests* de atribuição colocados pelos alunos e informar os alunos sempre que há alteração da *utility*. Contém a capacidade da sala alocada, o número total de alunos e o número de alunos pares e ainda o *behaviour* do protocolo *FIPA-Subscribe*, para posteriormente poder notificar os alunos interessados de que a sua *utility* foi alterada.

Classes Implementadas - Behaviours

UtilitySubInitiator

Implementa o papel de *Initiator* do protocolo *FIPA-Subscribe*, sendo responsável pelo envio das mensagens de subscrição para as turmas e por receber as notificações recebidas por estas e tratar as informações que as acompanham.

UtilitySubResponder

Implementa o papel de *Responder* do protocolo *FIPA-Subscribe*, sendo responsável por aceitar as subscrições dos alunos e de lhes comunicar sempre que a *utility* for alterada, ou seja, quando um aluno for adicionado à turma.

Classes Implementadas - Behaviours

AssignInitiator

Implementa o papel de *Initiator* do protocolo *FIPA-Request*, sendo responsável por pedir à turma que o coloque na mesma. Se o pedido for recusado escolhe uma nova turma após um *timeout* aleatório (entre 1 a 2.5 segundos - tempo necessário para ser notificado de alterações na subscrição da *utility*), caso contrário cancela a subscrição.

AssignResponder

Implementa o papel de *Responder* do protocolo *FIPA-Request*, sendo responsável por comparar a *utility* enviada pelo aluno e a sua própria *utility*. No caso de os valores coincidirem, o aluno é adicionado à turma e é informado que o pedido foi aceite. Caso contrário, o pedido é recusado.

Classes Implementadas - Messages

ParityMessage

Mensagem utilizada para o aluno comunicar com as turmas e contém a *parity* do aluno.

UtilityMessage

Mensagem utilizada pela turma para informar um aluno qual a sua *utility*, tendo em conta a paridade do mesmo. Contém o valor da *utility*.

AssignMessage

Mensagem utilizada para o aluno solicitar à turma que seja colocado na mesma. Contém o valor da paridade do aluno e a *utility* da turma que o aluno tem no momento do envio da mensagem.

Outras observações

A pasta “src/MainCases” inclui 4 casos de teste para possível demonstração do projeto.

O agente *Student* recebe como argumento a sua paridade (“odd” ou “even” - case insensitive).

O agente *CUClass* recebe como argumento (numa string entre aspas com cada argumento separado por espaços) a capacidade da sala, o número de alunos na turma e o número de alunos pares nessa turma. Por exemplo: “30 10 3”.