INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Licenciatura em Engenharia Informática

Simulador de pandemia (Recurso)

Relatório

Laura Melissa Bernardo Correia n.º17179

2020

INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Licenciatura em Engenharia Informática

Simulador de pandemia (Recurso)

Relatório

Elaborado por:

Laura Melissa Bernardo Correia n.º17179

Orientado por:

João Paulo Barros; Diogo Pina Manique; João Amarante
, IPBeja

Relatório de projeto para a Unidade Curricular de Programação Orientada a Objetos apresentado
na
Escola Superior a Tecnologia e Gestão do Instituto Politécnico de Beja

2020

Índice

Capítulo 1

Introdução

O presente trabalho está inserido na disciplina de Programação Orientada a Objetos da Licenciatura em Engenharia Informática e tem como objetivo a elaboração de um simulador de pandemia.

Capítulo 2

Funcionamento do Programa

O programa tem como principal objetivo simular uma pandemia. Dessa forma, terá pessoas infetadas, saudáveis e imunes. No princípio do programa, o utilizador põe como parâmetros de entrada a quantidade de pessoas que quer de cada tipo e põe também o valor de tempo mínimo e máximo que uma pessoa pode ficar doente. Dentro desses valores, mínimo e máximo, será gerado aleatóriamente um valor, este será o tempo que a pessoa doente levará a se curar. De seguida, inicia-se a simulação, em que é possível ver os movimentos aleatórios das pessoas, representadas por retângulos, no ecrã. Cada retângulo tem uma cor correspondente: verde para a pessoa saudável, vermelho para a doente, e azul para a imune. Cada vez que uma pessoa saudável se encontra numa posição justaposta a uma pessoa doente, a primeira passa também a estar doente e a sua cor muda. Dependendo do valor aleatório calculado para a recuperação das pessoas doentes, esta eventualmente passará a estar saudável.

Estes são os processos visíveis do programa.

Capítulo 3

Explicação das classes e métodos

3.1 *Package pt.ipbeja.po2.pandemic.gui*

As classes pertencentes ao *package pt.ipbeja.po2.pandemic.gui* são as responsáveis pelo que se vê no programa, mais propriamente, pela *interface* e é constituido por três classes.

3.1.1 Classe *GuiStart*

É desta classe que se inicia todo o programa.

3.1.2 Classe *ContagiousBoard*

Esta classe extende uma *VBox* e é responsável por pedir ao utilizador todos os parâmetros de entrada. Nesta classe criam-se as *TextFields* para receberem os valores que se pretende como *input* no início do programa e também se verifica se os valores são compatíveis através da função *verifyInputs()*. É também aqui que é criado o gráfico.

Métodos *getInputs1()* e *getInputs2()*

Este dois métodos recebem os parâmetros colocados pelo utilizador incialmente, as linhas, colunas, número de pessoas saudáveis, doentes e imunes, o tempo mínimo e máximo que uma pessoa pode estar doente e por fim a probabilidade de contágio e a probabilidade de ficar imúne.

Método *verifyInputs()*

Este método recebe como parâmetros o número de pessoas saudáveis, doentes e imunes. Recebe também o tempo mínimo e máximo que uma pessoa pode estar doente e as probabilidades de contágio e de uma pessoa passar a imúne. As verificações feitas neste método é se nenhum dos campos está vazio, se o número total de pessoas não ultrapassa o número máximo definido ((max = linhas * colunas) /2), se os valores mínimo e máximo são compatíveis, ou seja, se o valor mínimo é maior que zero e se é menor que o valor máximo e por fim, se as probabilidades passadas pelo utilizador não ultrapassam o valor 100. Se tudo isto se verificar, o método retorna *true*.

Método *reader()*

Este método têm a função de ler a partir do ficheiro escolhido pelo utilizador, o conteúdo do mesmo e guardá-lo na *string read* e é complementado com a função *saveValues()*.

Método *saveValues()*

Este método tem como objetivo através da *string read* ir buscar os valores das posições que se quer e guardá-las em arrays associados ao sei tipo de pessoa para depois passá-lo para o *world*.

Método *writer()*

O método *writer* é chamado quando o botão de guardar num ficheiro é carregado. A classe *world* tem um método que guarda as posições das pessoas em *string* e de seguida junta-as numa só apenas e retorna essa *string*.

Método *graph()*

Este método é responsável pelo gráfico visualizado durante a simulação. Os dados são atualizados através da classe *world*, onde são criadas três variáveis globais que guardam a quantidade de cada tipo de pessoas. Alterou-se a cor das barras também para a cor respetiva dos retângulos.

3.1.3 Classe *WorldBoard*

A classe *WorldBoard* extende a *Pane* e é a responsável por colocar e alterar tudo o que seja necessário na *interface*, segundo as intruções do *model*. Tem como pro-

priendades: as cores dos retângulos, o número de linhas e número de colunas, uma referência à classe *world* e tem um *Array* de *Arrays* de retângulos.

Método *updatePosition()*

Este método é responsável por alterar a posição dos retângulos, ou seja, é responsável por se conseguir visualizar a sua movimentação. Para isto, utilizou-se a teoria dos números, dada na cadeira de Matemática Discreta em que a = b, b = c então a = c. Assim guardou-se as propriedades do retângulo antigo e colocou-se na nova posição, pondo a anterior vazia.

Método *addRectangle()*

Este método tem como objetivo colocar a cor correta nos respetivos retângulos. Ele recebe uma *cell position* e consoante o *cell type* dessa *cell* a cor é colocada através do *array STATE_COLORS*.

Método *setCells()*

Este método é chamado depois de se percorrer o tabuleiro com o objetivo de alterar as cores consoante o estado da pessoa. O funcionamento é o seguinte, remove-se todos os retângulos que estavam anteriormente no tabuleiro e desenha-se os novos retângulos consonte o *array* de *Cells* que o método recebe.

3.2 *Package pt.ipbeja.po2.pandemic.model*

As classes pertencentes ao *package model* são responsáveis por toda a lógica do simulador e não tratam de nada do que seja *interface*.

3.2.1 Classe *Cell*

A classe *Cell* é uma classe abstrata que têm como principal função devolver a nova posição de uma pessoa.

3.2.2 Classe *Cell Position*

A classe *Cell Position* contém, como o próprio nome indica, a posição (linha e coluna) da *cell*.

3.2.3 Classe *CellType*

A classe *CellType* é uma classe do tipo *enum* que contém quatro tipos: *EMPTYCELL*, *HEALTHYPERSON*, *IMMUNEPERSON* e *SICKPERSON*.

3.2.4 Classe *Person*

A classe *Person* é uma classe do tipo abstrata que estende a classe *Cell*.

3.2.5 Classe *EmptyCell*, *HealthyPerson*, *ImmunePerson* e *SickPerson*

A classe *EmptyCell* estende a classe *Cell*. As classes restantes estendem a classe *Person*.

3.2.6 Classe *World*

A classe *world* tem toda a lógica do jogo.

Método *randomPosition()*

Este método chama a função *personType()* passando-lhe com parâmetros os valores dados pelo utilizador no ínicio do jogo. Este chama também o método *sickTime()* que é o responsável por gerar aleatóriamente o tempo que cada pessoa fica doente.

Método *personType()*

Este método recebe como parâmetros um *celltype*, o número de pessoas a colocar e um *array list* que irá servir para guardar as posições geradas. Este método contém um *switch case* que segundo o tipo de pessoa e a quantidade dada pelo utilizador, insere o número de pessoas no tabuleiro com posições aleatórias. O método *sickTime()* é explicado abaixo.

Método *sickTime()*

Este método recebe o tempo mínimo e máximo definido pelo utilizador e gera um número aleatório através da função *rand*.

Método *setPersons()*

Este método recebe os *arrays* que guardam as posições colocadas através do ficheiro. O que o método faz é pôr as pessoas consoante as posições passadas pelo utilizador, através de um ciclo *for* que percorre o *array*.

Método *simulate()*

O método *simulate* adormece a *thread* durante alguns segundos e executa o código, percorrendo o tabuleiro e chamando os métodos necessários para alterar as peças de lugar, atualizar o tabuleiro entre outros que irão ser explicados.

Método *samePosition()*

Este método tem como objetivo verificar se uma posição está contida no *array* ou não.

Método *updateModel()*

Este método recebe como parâmetros a linha e coluna antigas e a linha e coluna novas. Aqui utilizou-se uma variável temporária para mudar a pessoa de posição. No final retorna-se a nova *cellPosition* para ser adicionada ao *array lastMove* criado no método *simulate*.

Método *checkBoardState()*

O método seguinte percorre o tabuleiro e verifica se existe alguma pessoa doente, caso exista chama o método *makeSick()* que será explicado a seguir. É neste método também que entram as probabilidades, neste caso a de uma pessoa ficar imune. É gerado um número aleatório entre 0 e 100 e feita uma condição com base no valor que o utilizador inseriu inicialmente. Caso essa condição se aplique, a pessoa doente passa a estar imune se não passa apenas a saudável.

Método *makeSick()*

Este método é chamado quando é encontrada uma pessoa doente. O que faz é percorrer todas as posições à volta dessa pessoa doente e verificar se existe alguma pessoa saudável numa posição justaposta, caso exista, esta muda o seu estado para infetada. Isto depende também da probabilidade de contágio dada pelo utilizador.

Método *decreaseSickTime()*

O seguinte método percorre o mapa que guarda o tempo correspondente a cada pessoa doente e diminui-lhe um valor consoante cada iteração que ocorre.

Método *isInside()*

Este método complementa o método *makeSick()* ao verificar se as coordenadas passadas como parâmetros se encontram dentro dos limites do tabuleiro.

Método *stopGame()*

Este método retorna a variável *stopGame* a *true*, e serve para parar a *thread* quando o utilizador decide carregar no botão de *stop*.

Método *writeFile()*

O método *writeFile* é o método que percorre o tabuleiro e guarda as posições das pessoas nas *strings* correspondentes ao seu tipo, doente, saudável ou imúne. Concatena tudo numa só e retorna essa mesma *string* que irá ser usada na classe *ContagiouBoard* para ser escrita no ficheiro *.txt*.

Método *printBoard()*

Por fim, este método tem como objetivo imprimir o tabuleiro para a consola pois durante o trabalho ficava mais fácil de perceber se as coisas estavam a funcionar. Foram também adicionadas as variáveis que fazem a contagem da quantidade de pessoas no tabuleiro, estas variáveis são os dados que aparecem no grafico e que são atualizados a cada iteração.

3.3 Código de teste de movimento

No construtor da classe de testes criou-se um tabuleiro próprio para a realização dos mesmos. Iniciou-se todo o tabuleiro com posições vazias e depois preencheu-se o tabuleiro conforme foi necessário para a realização dos testes.

3.3.1 Movimento de uma pessoa para um célula vazia

moveToEmptyCellTest()

O primeiro teste de movimento necessário foi o movimento de uma pessoa para uma célula vazia, para isso verificou-se que realmente havia uma pessoa na posição xy, e que a posição para a qual se pretencia mover estava vazia. De seguida moveu-se a pessoa através do método *updateModel()* e por fim verificou-se que esta realmente foi movida.

3.3.2 Tentativa de movimento de uma pessoa para fora do tabuleiro em quatro casos:

Demasiado para a esquerda, para a direita, para cima e para baixo

A sequência dita acima corresponde aos métodos *outOfBoardLeft()*, *outOfBoardRight()*, *outOfBoardAbove()* e *outOfBoardDown()*. Estes quatro métodos funcionam da mesma maneira, as pessoas usadas e os valores que lhes são passados é que diferem. No primeiro exemplo de mover uma pessoa demasiado para a esquerda usou-se uma pessoa que já estava no limite esquerdo do tabuleiro. De seguida com o método *isAPossibleMove()* passa-se umas coordenadas que consoantes as coordenadas iniciais da pessoa, fiquem fora do tabuleiro. De seguida verifica-se que a pessoa continua no mesmo sitio que estava no inicio. Os outros métodos funcionam da mesma maneira mas com coordenadas diferentes.

3.4 Código de teste de contágio

No teste de contágio foi necessário ter em atenção o *setup* pedido no enunciado, assim colocou-se as pessoas nas devidas posições para que este *setup* fosse respeitado. Primeiro verificou-se que existiam aquele tipo de pessoas naquelas posições, depois alterou-se a posição da pessoa doente, e de seguida verificou-se que as pessoas imúnes continuavam no mesmo estado, as pessoas doentes também se mantinham doentes e que a pessoa saudável passava também a estar doente. Fez-se isto para dois tipos de movimentos, vertical e horizontal.

3.5 *Interface*

A *inteface* é composta por várias *textfields* que recebem os valores como parâmetros inicias. Contém o *menu item* para abrir o ficheiro caso o utilizador assim o pretenda e contém outro botão *start* para dar início ao jogo. Antes de passar à simulação, todos os campos são verificados, quer para ver se não estão vazios, quer para verificar se os valores colocados são possíveis. Depois de iniciada a simulção é possível ver dois botões na parte superior esquerda da janela, *start* e *stop*, estes quando são carregados param e iniciam a simulação com o número de iterações do início. A interface contém também um *menu item* que quando escolhido permite guardas o estado atual do tabuleiro num ficheiro *.txt*. Contém também um gráfico que mostra no decorrer da simulação a quantidade de pessoas e o seu tipo.

3.6 Forma como é calculado o contágio e a recuperação das pessoas

O contágio é feito cada vez que é encontrada uma pessoa doente no tabuleiro, verifica-se se nas suas posições justaposta existe alguma pessoa saudável, caso sim, esta passa a doente dependendo da probabilidade dada inicialmente pelo utilizador.

A recuperação das pessoas é feita através de uma mapa. Quando uma pessoa fica doente, adiciona-se essa pessoa ao mapa, correspondente do seu tempo de recuperação. A cada iteração o método *decreaseTime()* é chamado, e diminui o tempo que a pessoa fica doente. No método *checkBoardState* é verificado se esse tempo já chegou a 0 e caso sim, esta pessoa passa então a estar saudável. Dependedo do valor que o utilizador colocou na probabilidade de ficar imúne, esta pode vir a ficar imúne.

3.7 Ficheiro *.jar*

Por fim, seguindo as instruções foi criado um ficheiro *.jar*. Para correr o programa através da linha de comandos foi introduzido: *java -jar myJARFile.jar*.

Capítulo 4

Conclusões

Considerou-se que a parte mais dificil do trabalho foi a inicial para colocar os retangulos todos a mexer e a alterar as suas cores. Depois de ultrapassada essa dificuldade, foi-se avançando como tempo.

Bibliografia