Instituto Politécnico de Beja

Escola Superior de Tecnologia e Gestão

Engenharia Informática

Trabalho Prático de Avaliação 2

Programação Orientada a Objetos

Realizado por:
André Dâmaso 15307

Docentes: João Paulo Barros, Diogo Pina Manique e João Amarante

2020

Índice

1. Introdução

Este trabalho prático consiste em aprimorar o trabalho prático 1, onde este implementa um programa que permite simular o contágio numa epidemia/pandemia realizado em linguagem JAVA para a unidade curricular de Programação Orientada a Objetos.

Para este segundo trabalho foi utilizado o repositório GIT do primeiro trabalho para atualizar as versões do projeto.

Foram realizados os seguintes requisitos:

- **Requisitos Essenciais:** Destes requisitos foram desenvolvidos todos incluindo o seguinte relatório.
- **Requisitos Não Essenciais:** Destes requisitos foram desenvolvidos Req. NE1 e parte NE3.

1.1. O Jogo

As regras gerais a que o programa simulador obedece neste trabalho prático são as seguintes:

- Cada pessoa (Person) será representada por uma célula (Cell) numa grelha invisível;
- Existem três tipos de pessoa (Person), a saúdavel (HealhyPerson), a doente (SickPerson) e a imune à doença (ImunePerson);
- Cada pessoa é visualizada como um pequeno quadrado numa janela (Pane) de fundo liso; caso seja uma pessoa saudável, a pessoa tem a cor azul, caso seja uma pessoa doente, a pessoa tem uma cor vermelha, caso seja uma pessoa imune, a pessoa tem uma cor verde;
- Esses quadrados deslocam-se aleatóriamente na janela;
- O programa fornece ao utilizador a opção de escolher o número de pessoas de cada tipo;
- O programa fornece ao utilizador a opção de escolher o número de linhas e colunas;
- O utilizador pode iniciar o programa por um ficheiro, e por linha de comandos;
- O programa fornece ao utilizador a opção de gravar a simulação, sendo esta podendo ser utilizada;
- O programa inicia sempre com pessoas em posições em aleatórias;
- O programa fornece ao utilizador iniciar, parar e terminar a simulação;
- O programa fornece ao utilizador visualizar um gráfico correspondente ao número de pessoas;
- No final de cada passo de execução, as pessoas saudáveis que estão ao lado de uma pessoa doente passam a estar doentes antes do passo de execução seguinte;
- Ao fim de um tempo as pessoas doentes ficam curadas (ImunePerson);
- O programador também pode verificar o jogo pelo modo de testes.

2. Estrutura do Trabalho

Este projeto contém dois packages iniciais, o src e test. O src é onde está presente o código do trabalho que inclui o package gui e model, enquanto o test é referente ao teste do programa. A estrutura do projeto está presente na seguinte figura:

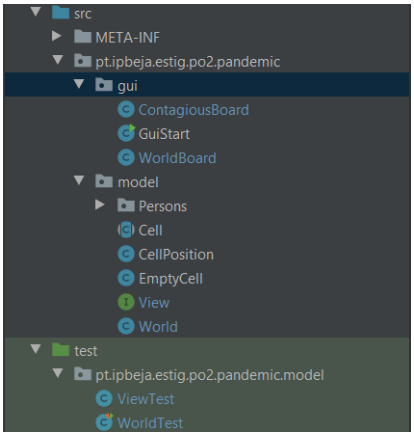

Figura 1: Estrutura do projeto

A classe ContagiousBoard é onde está desenhado o programa, onde os resultados são apresentados em modo gráfico.

A classe WorldBoard é onde estão desenhados os quadrados representados pelas pessoas.

A classe Cell, é a classe abstrata que define o tipo de dados do programa, onde ou são Pessoas ou Céculas vazias.

A classe EmptyCell, extende da classe Cell representando uma célula vazia.

A classe Person, extende da classe Cell, e esta é tambem uma classe abstrata, devido ao facto de poder conter classes filhas, representadas pelo tipo de pessoas.

As classes ImunePerson, HealhyPerson, SickPerson são as classes que extendem da classe Person.

A interface View é utilizada para especificar um comportamento que as classes devem implementar.

A classe word, é a classe que define o modelo do programa.

Para além do que foi demonstrado anteriormente, o projeto abrange mais ficheiros, isto é, o ficheiro jar, o presente relatório e os ficheiros de texto que interagem com o funcionamento do programa, quer para iniciar uma simulação ou um que seja guardado. A seguinte figura mostra a restante estrutura:

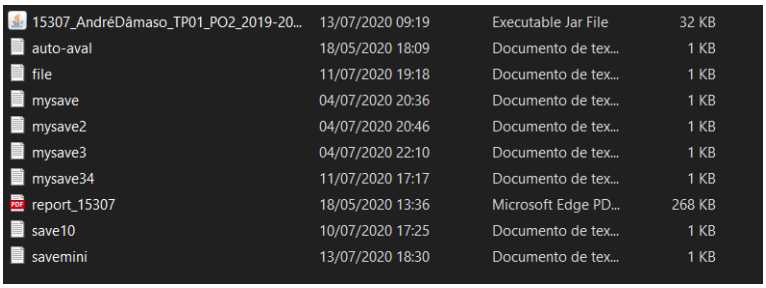

Figura 2: Estrutura do projeto

3. Início da Simulação

Para iniciar a simulação do contágio, pode-se realizar perante três maneiras diferentes. Pode-se iniciar em abrir o programa e preencher as caixas de texto. Pode iniciar em abrir o programa e selecionar a opção .ºpen"que abre um ficheiro ".txt". Ou pode abrir o programa por linha de comandos.

Começando pela última opção, utilizando um tutorial do moodle da disciplina, foi criado um ficheiro jar dentro da pasta do projeto, como está demonstrado a cima. Então, na classe GuiStart foi adicionado ao código a seguinte listagem:

Listing 1: GuiStart

```
1
2  @Override
3  public void start(Stage primaryStage) throws Exception {
4
5      ContagiousBoard board = new ContagiousBoard();
6      Scene scene = new Scene(board);
7      primaryStage.setScene(scene);
8      primaryStage.setTitle("Simulador de Pandemia");
9
10     if(System.console() != null) {
11         List<String> arguments = getParameters().getRaw();
12         String fileName = arguments.get(0);
13         System.out.println(fileName);
14         File file = new File(fileName);
15         board.openFile(file);
16     }
17
18
19     primaryStage.setOnCloseRequest((e) -> {
20         System.exit(0);
21     });
22     primaryStage.show();
23 }
```

No método *start* foi adicionado a verificação da linha 10, para quando é iniciado o programa verificar se está a ser corrido em linha de comandos ou não. Caso esteja a correr, é buscado um argumento (ficheiro .txt), em que vai ser iniciado o programa a correr como se fosse um open.

Para realizar a segunda opção de início, ou seja, iniciar a partir da abertura de um ficheiro de um menu "open", é feito primeiramente a construção de um menu "File", um item "Open", um item "Save"para o requisito de gravar, e um item "Exit" para sair. A seguinte listagem, na classe ContagiousBoard, mostra a criação das suas interfaces gráficas:

Listing 2: Método drarMenuBar

```
1
2  private void drarMenuBar() {
3
4      menuBar = new MenuBar();
5      Menu menu1 = new Menu("File");
6      submenu1 = new MenuItem("Open");
7      MenuItem submenu2 = new MenuItem("Save As...");
8      MenuItem submenu3 = new MenuItem("Exit");
9      this.menuBar.getMenus().add(menu1);
10     menu1.getItems().addAll(submenu1, submenu2, submenu3);
```

Quando é clicado no item "Open" é feito a seguinte ação:

Listing 3: Open action

```
submenu1.setOnAction(e -> {

    System.out.println("Menu Item 1 Selected");
    FileChooser fileChooser = new FileChooser();
    fileChooser.setInitialDirectory(new File("C:\\Users\\usuario\\Documents\\EI\\PO2 1
    fileChooser.setTitle("Open Resource File");
    fileChooser.getExtensionFilters().addAll(
            new FileChooser.ExtensionFilter("Text Files", "*.txt"));
    File file = fileChooser.showOpenDialog(null);
    if (file != null) {
        openFile(file);
    }
});
```

Após abrir o ficheiro é chamado o seguinte método:

Listing 4: Método openFile

```
public void openFile(File file) {

    List<String> strings = new ArrayList<>();

    try {
        BufferedReader bufferedReader = new BufferedReader(new FileReader(file));
        String availalbe;
        while((availalbe = bufferedReader.readLine()) != null) {
            strings.add(availalbe);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Este método inicialmente le o ficheiro linha a linha e guarda cada linha na lista *strings*.
De seguida, percorrendo o método:

Listing 5: Método openFile

```
int line = Integer.parseInt(strings.get(0));
int col = Integer.parseInt(strings.get(1));

List<String> listHealhy = new ArrayList<>();
List<String> listImune = new ArrayList<>();
List<String> listSick = new ArrayList<>();

for (int i = 0; i < strings.size() - 1; i++) {
    switch (strings.get(i)) {
        case "healthy":
            for (int j = i + 1; j < strings.size(); j++) {
                if ((!strings.get(j).equals("immune")) && (!strings.get(j).equals("si
                    listHealhy.add(strings.get(j));
                } else {
```

```
                    break;
                }
            }
            break;
        case "immune":
            for (int j = i + 1; j < strings.size(); j++) {
                //System.out.println(strings.get(j));
                if ((!strings.get(j).equals("sick")) && (!strings.get(j).equals("healt
                    listImune.add(strings.get(j));
                } else {
                    break;
                }
            }
            break;
        case "sick":
            for (int j = i + 1; j < strings.size(); j++) {
                if ((!strings.get(j).equals("immune")) && (!strings.get(j).equals("hea
                    listSick.add(strings.get(j));
                } else {
                    break;
                }
            }
            break;
    }

}
```

Na listagem a cima é feito como são lidos os ficheiros de texto para traduzir no que é preciso para iniciar o programa. Nas primeiras duas linhas são retiradas oo valores relativos à quantidade de linhas e colunas. Depois são criados 3 listas, sendo que cada uma representa os dados afetos às posiçoes de cada tipo de pessoa. No loop em que está presente um switch é percorrido as linhas para o caso de cada tipo de pessoa, até à última linha do ficheiro. Caso encontre uma string com o texto da pessoa seguinte, os dados que estão a ser guardados na pessoa em questão são parados, tratando este processo para cada vez que encontra uma pessoa.

No seguimento seguinte, é verificado se os dois primeiros valores do ficheiro, ou seja, as linhas e colunas se são maior ou igual a 3. Caso não sejam, é devolvido um alerta de erro. Caso contrário, é iniciado a simulação, levando as pessoas consigo.

Listing 6: Método openFile

```

if(line <= 3 || col <= 3){
            Alert alert = new Alert(Alert.AlertType.ERROR,
                    "N mero de linhas ou colunas curto");
            alert.showAndWait();
        }else {
            disabledButtons(true, false, false);

            world = new World(this, line, col);
            this.pane = new WorldBoard(this.world, 10);
            chart.getData().add(series);
            left.getChildren().add(pane);
            world.setPersons(true, listHealhy, listImune, listSick);
            world.start();
        }
```

Para iniciar o jogo diretamente pelo programa, primeiramente no construtor está presente uma VBox *leftBox*:

Listing 7: VBox leftBox

```
1
2  VBox leftBox = new VBox(
3          this.textLine,
4          this.initLine,
5          this.textCol,
6          this.initCol,
7          new Separator(Orientation.HORIZONTAL),
8          this.textHealhy,
9          this.initHealhy,
10         this.textSick,
11         this.initSick,
12         this.textImmune,
13         this.initImmune,
14         new Separator(Orientation.HORIZONTAL),
15         startBtn
16 );
```

As variáveis da VBox são do tipo TextField e TextLine, para ser escrito vários campos para o utilizador introduzir. O número de linhas, número de colunas, número de pessoas saudáveis, número de pessoas doentes e número de pessoas imunes. Por fim, está presente um botão para iniciar a simulação.

No construtor está presente a construção do topBar que contém a construção do menu "File", a construção de um HBox com dois botões (stop e restart), e um gráfico inicialmente vazio. A seguinte figura ilustra a representação gráfica inicial do programa.

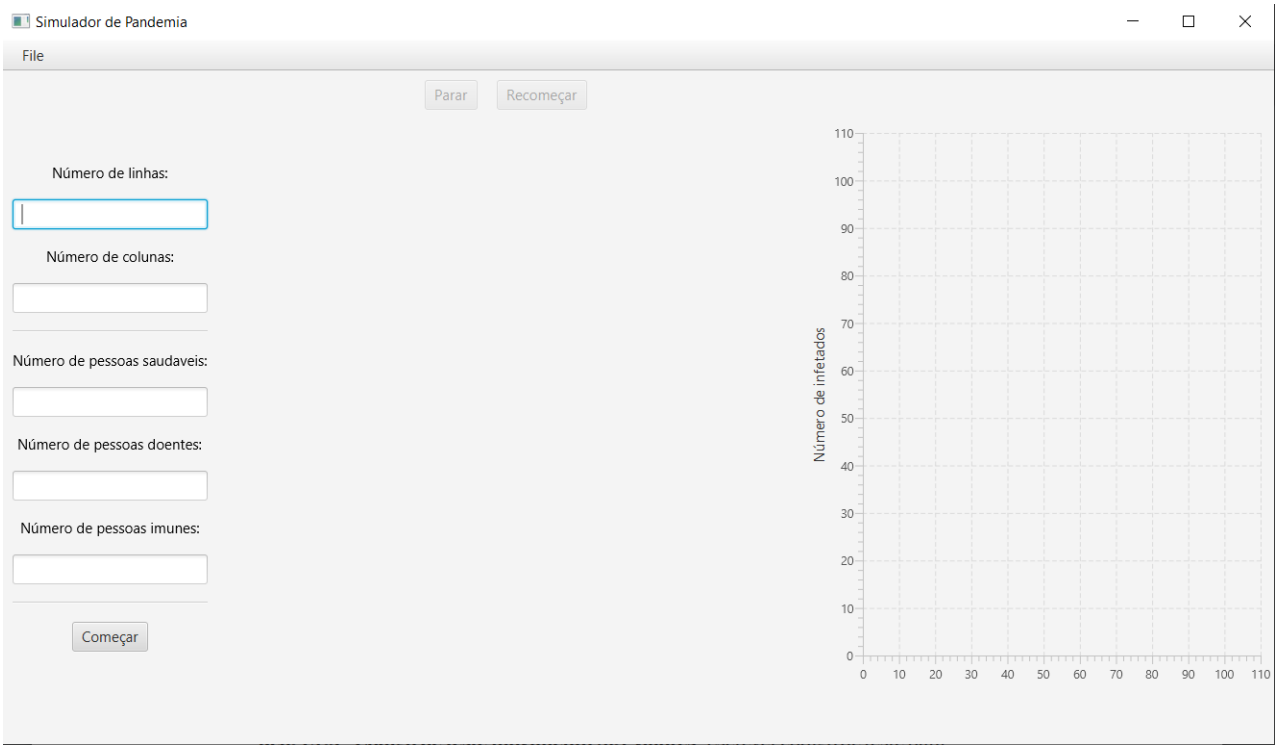

Figura 3: Cena inicial do programa

A seguinte listagem inclui o método relativo ao handler do botão "Começar":

Listing 8: Método StartButtonHandler

```
1
2  private class StartButtonHandler implements EventHandler<ActionEvent> {
3
4      @Override
5      public void handle(ActionEvent event) {
6
7              if(initCol.getText().equals("")){
8                  Alert alert = new Alert(Alert.AlertType.ERROR,
9                          "N mero de linhas inv lido");
10                 alert.showAndWait();
11
12             }else if(initLine.getText().equals("")){
13                 Alert alert = new Alert(Alert.AlertType.ERROR,
14                         "N mero de colunas inv lido");
15                 alert.showAndWait();
16
17             }else if(initHealhy.getText().equals("")){
18                 Alert alert = new Alert(Alert.AlertType.ERROR,
19                         "Tem de selecionar quantidade de pessoas saudaveis");
20                 alert.showAndWait();
21
22             }else if(initSick.getText().equals("")){
23                 Alert alert = new Alert(Alert.AlertType.ERROR,
24                         "Tem de selecionar quantidade de pessoas doentes");
25                 alert.showAndWait();
26
27             }else if(initImmune.getText().equals("")){
28                 Alert alert = new Alert(Alert.AlertType.ERROR,
29                         "Tem de selecionar quantidade de pessoas imunes");
30                 alert.showAndWait();
31
32             }else {
33
34                 disabledButtons(true, false, false);
35                 //world = new World(ContagiousBoard.this, 60, 60);
36                 int line = Integer.parseInt(initLine.getText());
37                 int col = Integer.parseInt(initCol.getText());
38                 restart(line, col);
39                 left.getChildren().add(pane);
40                 world.setSize(Integer.parseInt(initHealhy.getText()), Integer.parseInt
41                         Integer.parseInt(initImmune.getText()));
42                 world.start();
43
44             }
45
46     }
47 }
```

Após ser clicado no botão é verificado se todos os campos são preenchidos, para o caso que não sejam, é retornado alertas para o utilizador preencher. Caso contrário é chamado o método *restart* primeiro, e de seguida é começado a simulação.

O método *restart*, tem como objetivo verificar se o programa irá conter mais de 3 linhas e menos de 60. Restringiu-se o número máximo de linhas e colunas a 60, para que o tabuleiro fique dentro do espaço que está mais ou menos calculado a área que possa caber na interface gráfica (ficando entre as caixas de diálogos e o

gráfico). Caso esteja tudo correto é atribuído ao World as linhas e colunas, e iniciado o painel.

Listing 9: VBox leftBox

```
1
2  private void restart(int line, int col){
3
4      if(line <= 3 || col <= 3){
5          Alert alert = new Alert(Alert.AlertType.ERROR,
6                  "N mero de linhas ou colunas curto. \nLinha e Coluna t m de ser mai
7          alert.showAndWait();
8          disabledButtons(false, false, false);
9      }else if(line > 60 || col > 60){
10         Alert alert = new Alert(Alert.AlertType.ERROR,
11                 "N mero de linhas ou colunas muito longo. \nLinha e Coluna t m de s
12         alert.showAndWait();
13         disabledButtons(false, false, false);
14     }else {
15         world = new World(this, line, col);
16         this.pane = new WorldBoard(this.world, 10);
17         chart.getData().add(series);
18     }
19     //pane.autosize();
20 }
```

Na classe World foi optado por cirar um array de arrays de Cell, sendo esta a grelha geral do tabuleiro, e também criar um arrayList de Person, para receber as pessoas. Também foram criadas 3 listas para associar às listas das pessoas do ficheiro de texto. A variável boleana *isFile* está inicializada a false para dar a condição de qual forma é iniciado o programa (quando é iniciado pelo ficheiro fica a true) Por fim foi declarada a thread que vai correr a simulação.

Listing 10: Construtor World

```
1
2      public Cell[][] map;
3      public ArrayList<Person> persons;
4
5      private boolean isFile = false;
6      private List<String> healhyPeople;
7      private List<String> immunePeople;
8      private List<String> sickPeople;
9
10     Thread t1;
11
12     public World(View view, int nLines, int nCols) {
13         this.view = view;
14         this.nLines = nLines;
15         this.nCols = nCols;
16         this.map = new Cell[nLines][nCols];
17         this.persons = new ArrayList<>();
18
19     }
```

Na listagem seguinte está exibido o método start, que inicia a thread *t* com dois métodos, no qual o *populate* inicia as posições das pessoas, e o *simulate* realiza as simulações (500 vezes neste caso).

Listing 11: Método start

```
public void start() {

   t1 = new Thread( () -> {
       this.populate();
       this.simulate(500);
    });

   t1.start();

}
```

O método *populate*, demonstrado na seguinte listagem, contém duas formas de iniciar as pessoas. Para o caso que o programa foi iniciado por um ficheiro de texte, na condição (true) é percorrido de cada lista de pessoa, trazida do ficheiro. Em cada lista, ou seja, em cada linha, é retirado o primeiro campo para a linha, e o segundo para a coluna que vai ser iniciado a pessoa.

Listing 12: Método populate

```
if(isFile){

    for(String hp : healhyPeople){

        String[] lc = hp.split(" ");
        int line = Integer.parseInt(lc[0]);
        int col = Integer.parseInt(lc[1]);
        Person healhyPerson = new HealhyPerson(new CellPosition(line, col), this, 1);
        persons.add(healhyPerson);
        map[line][col] = healhyPerson;

    }

    for(String sp : sickPeople){

        String[] lc = sp.split(" ");
        int line = Integer.parseInt(lc[0]);
        int col = Integer.parseInt(lc[1]);
        Person sickPerson = new SickPerson(new CellPosition(line, col), this, 2);
        persons.add(sickPerson);
        map[line][col] = sickPerson;

    }

    for(String ip : immunePeople){

        String[] lc = ip.split(" ");
        int line = Integer.parseInt(lc[0]);
        int col = Integer.parseInt(lc[1]);
        Person imunePerson = new ImmunePerson(new CellPosition(line, col), this, 3);
        persons.add(imunePerson);
        map[line][col] = imunePerson;

    }

```

```
37 }
```

Para o caso que não seja iniciado por um ficheiro, é iniciado em posições aleatórias cada tipo de pessoa, pela quantidade que o utilizador introduz inicialmente.

Listing 13: Método populate

```
1
2  for (int i = 0; i < healhySize; i++) {
3
4      int line = (int) (Math.random() * this.nLines);
5      int col = (int) (Math.random() * this.nCols);
6
7      persons.add(new HealhyPerson(new CellPosition(line,
8              col), this, 1));
9      map[line][col] = persons.get(i);
10
11 }
12
13 for (int i = 0; i < sickSize; i++) {
14
15     int line = (int) (Math.random() * this.nLines);
16     int col = (int) (Math.random() * this.nCols);
17
18     persons.add(new SickPerson(new CellPosition(line,
19             col), this, 2));
20     map[line][col] = persons.get(i);
21
22 }
23
24 for (int i = 0; i < immuneSize; i++) {
25
26     int line = (int) (Math.random() * this.nLines);
27     int col = (int) (Math.random() * this.nCols);
28
29     persons.add(new ImmunePerson(new CellPosition(line,
30             col), this, 3));
31     map[line][col] = persons.get(i);
32
33 }
```

Percorrendo o destino, na classe WorldBoard, primeiramente é criado um ArrayList de quadrados, o mesmo que já teria sido feito para as pessoas na classe World. Entãp o método *populateWorld* adiciona um recângulo ao arraylist por cada vez que percorre uma pessoa

Listing 14: Método populateWorld

```
1
2      public void populateWorld(){
3
4          for(Person person : ContagiousBoard.world.persons){
5              rectangles.add(addRectangle(person));
6          }
7      }
```

O método que adiciona um recângulo, adiciona-o na posição da pessoa e com a cor representante ao método *color*.

Listing 15: Método addRectangle

```
1
2     private Rectangle addRectangle(Person person) {
3
4         int line = person.cellPosition().getLine() * CELL_SIZE;
5         int col = person.cellPosition().getCol() * CELL_SIZE;
6
7         Rectangle r = new Rectangle(line, col, CELL_SIZE, CELL_SIZE);
8
9         r.setFill(color(person));
10
11        Platform.runLater( () -> {
12            this.getChildren().add(r);
13        });
14        return r;
15    }
```

No método *color* este devolve a cor conforme seja o tipo de pessoa:

Listing 16: Método addRectangle

```
1
2     private Color color(Person person){
3
4         if (person.getState() == 1) { //normal
5             return Color.BLUE;
6         } else if (person.getState() == 2) { //infected
7             return Color.RED;
8         } else if (person.getState() == 3) { //recovered
9             return Color.GREEN;
10        }
11
12        return null;
13    }
```

4. Classes Mundo

A classe Cell, como dito anteriormente, é uma classe abstrata, e contem uma posição de linha e coluna, e o modelo.

Listing 17: Classe Cell

```
1
2       public abstract class Cell {
3
4       protected CellPosition cellPosition;
5       private World model;
6
7
8       public Cell(CellPosition cellPosition, World model) {
9           this.cellPosition = cellPosition;
10          this.model = model;
11      }
```

A classe Person, extende da classe cell a posição e o modelo, e inicia uma variável inteira *state*, no qual esta representa o estado que uma pessoa está. Se está saúdavel era irá ser iniciada a 1, se está doente ela irá ser inicada a 2, se está imune ela irá ser inicada a 3. As variáveis dx, dy, são as variáveis que representam o caminho que a pessoa vai percorrer.

Listing 18: Classe Person

```
1
2       private int dx;
3       private int dy;
4       private WorldBoard pane;
5       private int state;
6
7       public Person(CellPosition cellPosition, World model, int state) {
8           super(cellPosition, model);
9           this.state = state;
10      }
11
12      public abstract TypeContagious typeContagious();
13      public abstract TypeImmune typeImmune();
14      public abstract TypeSick typeSick();
```

Esta classe contém métodos abstratos que retorna, uma característica que uma pessoa pode ter, ou seja, pode ser contagiousa ou não, a pessoa que esteja recuperada (classe imune) pode ficar imune à doença ou não, e a pessoa doente pode ser sintomática ou assintomática.

Listing 19: Classe Person

```
1
2
3   public enum TypeContagious {
4
5       CONTAGIOUS, NO_CONTAGIOUS;
6
7       public static TypeContagious getContagious() {
8           Random random = new Random();
9           return values()[random.nextInt(values().length)];
10      }
11
12  }
```

Na listagem anterior está um exemplo do enum do tipo de contágio, onde retorna um valor aleatório para uma pessoa entre ser contagiosa ou não.

A classe EmptyCell é a classe das células vazias que herda de Cell.

Listing 20: Classe EmptyCell

```
1
2  public class EmptyCell extends Cell {
3
4      public EmptyCell(CellPosition cellPosition, World model) {
5          super(cellPosition, model);
6      }
7  }
```

A seguinte listagem, está exemplificada para um tipo de pessoa.

Listing 21: Classe SickPerson

```
1
2  public class SickPerson extends Person {
3
4      private TypeSick typeSick;
5      private TypeContagious typeContagious;
6
7      public SickPerson(CellPosition cellPosition, World model, int state) {
8          super(cellPosition, model, state);
9          this.typeContagious = TypeContagious.getContagious();
10         this.typeSick = TypeSick.getSymptom();
11     }
12
13     @Override
14     public TypeContagious typeContagious() {
15         return typeContagious;
16     }
17
18     @Override
19     public TypeImmune typeImmune() {
20         return null;
21     }
22
23     @Override
24     public TypeSick typeSick() {
25         return typeSick;
26     }
27
28
29 }
```

O tipo de pessoa retorna um valor aleatório para se é contagiosa ou não, retorna null para se ficou imune ou não, pelo facto de uma pessoa doente não tem essa característica, e retorna um valor aleatório se tem sintomas ou não.

5. Movimento

O início foi feito na classe Person, o método que move a pessoa de forma aleatório sem esta sair da tela, aperfeiçoando a rapidez do movimento. Este método é utilizado sempre que é chamado a pessoa, pode lhe retornar para onde se irá movimentar.

Listing 22: Método randomMove

```
1
2     public boolean randomMove() {
3
4         if(Math.random()<.1) {
5             final int[] v = {-1, 0, 1};
6             this.dx = v[World.rand.nextInt(3)];
7             this.dy = v[World.rand.nextInt(3)];
8             if (dx == 0 && dy == 0) {// to force a move
9                 dx = 1;
10            }
11        }
12
13        int line = cellPosition.getLine();
14        int col = cellPosition.getCol();
15
16        if (line + dy < 0 || line + dy > ContagiousBoard.world.nLines() - 1) {
17            dy = -dy;
18        }
19        if (col + dx < 0 || col + dx > ContagiousBoard.world.nCols() - 1) {
20            dx = -dx;
21        }
22
23
24        this.cellPosition = new CellPosition(
25                this.cellPosition.getLine() + dy,
26                this.cellPosition.getCol() + dx);
27
28
29        return true;
30
31
32    }
```

Para realizar uma movimentação mais realistica, para ela não mudar constantemente de direções, visto que a célula move-se para cima, baixo e nas diagonais, foi criada a condição de realizar esses movimentos numa percentagem de 10 porcento. Após isso, é verificado se a posição atual somado para onde se irá movimentar está dentro dos limites do painel. Caso o movimento tende a sair, é forçado o movimento contrário. E então é dada uma nova posição à pessoa.

Voltando à classe World, o que realiza o movimento das diversas pessoas, está presente no método *simulate* e adiciona sempre uma célula vazia, para onde não existe pessoas, e realiza os movimentos.

Listing 23: Método simulate

```
1
2 private void simulate(int nIter) {
3
4     for (int i = 0; i < nIter; i++) {
5
6         try {
```

```
            Thread.sleep(200);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        this.checkEmptyCell();
        this.move();
```

Listing 24: Método checkEmptyCell e método move

```

private void checkEmptyCell() {

    for (int line = 0; line < nLines; line++) {
        for (int col = 0; col < nCols; col++) {

            if(map[line][col] == null){
                map[line][col] = new EmptyCell(new CellPosition(line,col), this);
            }
        }
    }

}

private void move() {

    for(Person person : persons){
        if (person.randomMove()){
            this.view.updatePosition(persons.indexOf(person), person.dx(), person.dy()
        }
    }

}
```

Após verificar o movimento de cada pessoa, na classe WorldBoard, o método *updatePosition* leva o destino da posição da pessoa e leva o indice da pessoa em questão, para ser comparado com o respetivo quadrado, onde é feito no método *updatePosition* que move o quadrado para uma posição.

Listing 25: Método updatePosition

```

   public void updatePosition(int dx, int dy, int index) {
        TranslateTransition tt =
        new TranslateTransition(Duration.millis(200), rectangles.get(index));
        tt.setByX(dx * CELL_SIZE);
        tt.setByY(dy * CELL_SIZE);
        tt.play();
    }
```

Este programa oferece ao utilizador a possibilidade de poder pausar e retomar a simulação do contágio. Para tal estão presente dois botões, o botão "stopBtn" e o botão "restartBtn".

Quando o botão "stopBtn" é clicado é feito o seguinte handler:

Listing 26: StopButtonHandler

```

private class StopButtonHandler implements EventHandler<ActionEvent> {
```

```
    @Override
    public void handle(ActionEvent event) {

        disabledButtons(true, false, false);
        world.stop();
        System.out.println("stop");


    }
}
```

Este botão simplestemte chama o método *stop* da classe world:

Listing 27: Método stop

```
public void stop() {

    t1.suspend();

}
```

Já o método stop faz com que a thread pare.
Quando o botão "restartBtn" é clicado é feito o seguinte handler:

Listing 28: RestartButtonHandler

```
private class RestartButtonHandler implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent event) {

        disabledButtons(true, false, false);
        world.restart();
        System.out.println("recome ar");

    }
}
```

Já este botão simplestemte chama o método *restart* da classe world:

Listing 29: Método restart

```
public void restart(){

    try {
        Thread.sleep(200);
        t1.resume();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

}
```

Já o método restart faz com que a thread antes de recomeçar, aguente 200 milesegundos.

6. Recuperação

Na classe Person é iniciado duas variáveis; *healtime* e *sickTime*. A primeira é o tempo aleatório que as pessoas doentes têm para ficarem recuperadas. A segunda é o tempo inicial.

Listing 30: Inicialização das variáveis de tempo

```
1
2       public static int healtime = World.rand.nextInt(5 * 50);
3       private int sickTime = 0;
```

O método *cureSick* verifica para as pessoas doentes quando antigirem o tempo de ficarem recuperadas, essas pesseoas ficam imunes e é chamado o método que irá aleterar a cor delas para verde (mesmo processo do contágio).

Listing 31: Método cureSick

```
1
2       public void cureSick(){
3
4           if(this.getState() == 2){
5               sickTime++;
6               if (sickTime > healtime){
7                   this.setState(3);
8                   super.getModel().changePerson(this);
9               }
10          }
11      }
```

Este processo é realizado durante a simulação na classe World, após cada movimento realizado, e antes de cada colisão realizada, então foi adicionado ao método *simulate* o método *resolveSick*.

Listing 32: Método simulate

```
1 ...
2 this.checkEmptyCell();
3 this.move();
4 this.resolveSick();
5 ...
```

O método *resolveSick* percorre cada pessoa e faz a verificação de eventual cura.

Listing 33: Método resolveSick

```
1
2       private void resolveSick(){
3
4           for(Person person : persons){
5
6               person.cureSick();
7
8           }
9       }
```

7. Contágio

Como foi feito para o caso anterior, foi construído na classe world um método *testCollision*, sendo este que recebe duas pessoas:

Listing 34: Método testCollision

```
1
2  public void testCollision(Person person, Person otherPerson){
3
4      Rectangle r1 = new Rectangle(otherPerson.cellPosition().getLine() * 10, otherPerso
5      Rectangle r2 = new Rectangle(person.cellPosition().getLine() *10, person.cellPosit
6
7      //collision check
8      if(r1.intersects(r2.getLayoutBounds())){
9
10         if(person.getState() == 2 && otherPerson.getState() == 1) { //case person 1 is
11             otherPerson.setState(2);
12             changePerson(otherPerson);
13             System.out.println("tocaram");
14         }else if(person.getState() == 1 && otherPerson.getState() == 2) { //case perso
15             person.setState(2);
16             changePerson(person);
17             System.out.println("tocaram");
18         }
19
20     }
21
22 }
```

Neste método é associado a cada pessoa um quadrado, sendo que o quadrado r1 destina-se a otherPerson e o quadrado r2 a person. Quando se verificar uma colisão por parte dos quadrados, caso a person esteja doente e a otherPerson não esteja doente, a otherPerson fica doente, ficando com o estado 2 (Representa doente). Caso, seja ao contrário, a person fica doente. Se existir colisão entre outro tipo de pessoas, como por exemplo uma pessoa saudavel e uma pessoa imune, não existe contágio, o mesmo se uma pessoa doente intersectar com uma pessoa imune.

Sempre que exista uma colisão é chamado o *changePerson* na classe World. Este método chama a view para atualizar a cor da pessoa em questão, levando tambem o seu índice.

Listing 35: Método updateColor

```
1
2    public void changePerson(Person person) {
3        this.view.updateColor(person, persons.indexOf(person));
4     }
```

Na classe WorldBoard o método *updateColor* atualiza a cor do quadrado com o mesmo índice da pessoa.

Listing 36: Método updateColor

```
1
2    public void updateColor(Person person, int index){
3
4        rectangles.get(index).setFill(color(person));
5
6     }
```

Este processo é realizado durante a simulação na classe World, após cada movimento realizado e cada verificação de recuperação, então foi adicionado ao método *simulate* o método *resolveCollisions*.

Listing 37: Método simulate

```
...
this.checkEmptyCell();
this.move();
this.resolveSick();
resolveCollisions();
...
```

O método *resolveCollisions* é onde se percorre as pessoas para a colisão.

Listing 38: Método resolveCollisions

```

   public void resolveCollisions(){

       for(int i =0; i < persons.size();i++) {
            for(int j = i+1 ; j < persons.size();j++){
                //for each unique pair invoke the collision detection code
                testCollision(persons.get(i), persons.get(j));
            }
        }


    }
```

8. Gráfico

Para atualizar o gráfico, que está presente na interface gráfica, no qual realiza a contagem do número de pessoas infetadas, foi adicionado no método *simulate* da classe World a seguinte função:

Listing 39: Método simulate

```
...
this.checkEmptyCell();
this.move();
this.resolveSick();
this.updateCountSick();
...
```

Como se pode verificar na listagem a cima, foi adicionado o método *updateCountSick* que vai ter a função de contar em cada iteração o número de infetados. E é declarado depois do contágio, para ter uma contagem atualizada. Na listagem seguinte está presente o método *updateCountSick*

Listing 40: Método updateCountSick

```

public void updateCountSick(){

    for(Person person : persons){
        if(person.getState() == 2) {
            counterSick++;
        }
    }

    this.view.updateGraph(counterSick);
    counterSick = 0;

}
```

Nesta função está percorrido todas as pessoas em cena e verificado quem está doente. Caso aconteça é contado pela variável *counterSick*, onde é inicializada primeiro a 0 e em cada iteração conta o número de doentes. Após contar o número de doentes é chamado o método da view que faz o update do número de infetados. Após contar, o *counterSick* fica a 0 para se poder contar novamente para as iterações seguintes.

O método updateGraph, demonstrado na seguinte listagem, recebe o valor relativo ao número de infetados vindo do método anterior, e atualiza a posição y do gráfico, sendo que a posição x foca-se por cada iteração.

Listing 41: Método updateGraph

```

@Override
    public void updateGraph(int numInfects) {
        Platform.runLater( () -> {

            series.getData().add(new XYChart.Data<>(x, numInfects));
            x++;

        });
    }
```

9. Gravar Simulação

Para realizar este passo, é utilizado o item "Save As.." do menu "File", criado no início da simulação. O objetivo é como gravar um jogo, mas num ficheiro de texto, ou seja, guardar as posições da simulação a correr, e escrever num ficheiro com a estrutura do ficheiro que é para ser aberto no programa. Assim o utilizador tem a possibilidade de parar a simulação, guarda-la, e continuar num momento depois, podendo abrir a gravação noutra altura.

Quando é clicado no item, é feito o seguinte evento:

Listing 42: Save action

```
1
2  submenu2.setOnAction(e -> {
3
4      world.stop();
5
6      System.out.println("Menu Item 2 Selected");
7      FileChooser saveFile = new FileChooser();
8      saveFile.setInitialDirectory(new File("C:\\Users\\usuario\\Documents\\EI\\PO2 202
9      saveFile.setTitle("Open Resource File");
10     saveFile.setInitialFileName("mysave");
11     saveFile.getExtensionFilters().addAll(
12             new FileChooser.ExtensionFilter("Text Files", "*.txt"));
13     File file = saveFile.showSaveDialog(null);
14
15     if(file != null)
16     {
17         saveFile(file);
18     }
19
20 });
```

Quando é clicado, primeiro é parado a simulação, para enquanto se efetua a ação não estar a correr o programa, e de seguida é feito a mesma forma de abrir um ficheiro, mas para ser criado um ficheiro, na pasta do programa e com a inicial "mysave". De seguida é chamado o método *saveFile*:

Listing 43: Método saveFile

```
1
2  private void saveFile(File file) {
3
4      data = world.setData();
5
6      BufferedWriter wr = null;
7      try {
8          wr = new BufferedWriter(new FileWriter(file));
9          for (String var : data) {
10             wr.write(var);
11             wr.newLine();
12         }
13         wr.close();
14
15     } catch (Exception e) {
16         //TODO: handle exception
17     }
18
19 }
```

Esta função primeiro chama o método *setData* da classe World, que contém os dados guardados relativo as pessoas, sendo que a variável *data* é uma lista de strings e em cada sting está o conteúdo a escrever no ficheiro a guardar.

Listing 44: Método simulate

```
public List<String> setData(){

    return listData;

}
```

O método setData retorna uma lista de string *listData* em que cada conteúdo é a linha a escrever no ficheiro. Essa lista é atualizada no método *getData* que é declarado no método *simulate* em último lugar.

Listing 45: Método simulate

```
...
this.checkEmptyCell();
this.move();
this.resolveSick();
this.resolveCollisions();
this.updateCountSick();
this.getData();
```

Na listagem seguinte está presente o método *getData*:

Listing 46: Método getData

```
private void getData() {

    listData.removeAll(listData);
    healhyData.removeAll(healhyData);
    sickData.removeAll(sickData);
    imuneData.removeAll(imuneData);

    listData.add(String.valueOf(this.nLines));
    listData.add(String.valueOf(this.nCols));

    for(Person person : persons){

        if(person.getState() == 1){
            int line = person.cellPosition().getLine();
            int col = person.cellPosition().getCol();
            String position = line + " " + col;
            healhyData.add(position);
        }

        if(person.getState() == 2){
            int line = person.cellPosition().getLine();
            int col = person.cellPosition().getCol();
            String position = line + " " + col;
            sickData.add(position);
        }

        if(person.getState() == 3){
            int line = person.cellPosition().getLine();
            int col = person.cellPosition().getCol();
```

```
30              String position = line + " " + col;
31              imuneData.add(position);
32          }
33
34      }
35
36      listData.add("healthy");
37      listData.addAll(healhyData);
38
39      listData.add("sick");
40      listData.addAll(sickData);
41
42      listData.add("immune");
43      listData.addAll(imuneData);
44
45
46  }
```

A função anterior trabalha por 3 listas, no qual cada lista é correspondente aos dados de cada tipo de pessoa. Primeiro começa por remover o conteúdo de todas as listas, devido ao facto de estar a ser sempre chamado em cada iteração da simulação, então para não conter dados da iteração anterior resolveu-se eliminar os dados para depois ser introduzido e quando for chamado a opção de gravar, fica com os últimos registos da lista. Caso contrário, como já foi dito é eliminado conteúdo sempre antes de registar.

A *listData* adiciona primeiramente o número de linhas e colunas para os dois primeiros conteúdos (duas primeiras linhas do ficheiro).

De seguida é percorrido todas as pessoas, onde para cada tipo de pessoa é registado as suas posições para cada linha que vai conter o ficheiro. Antes do conteúdo é escrito a string relativa ao tipo de pessoa.

10. Testes do programa

Antes da realização dos testes ao programa, a seguinte figura, mostra gráficamente a interface do programa a correr:

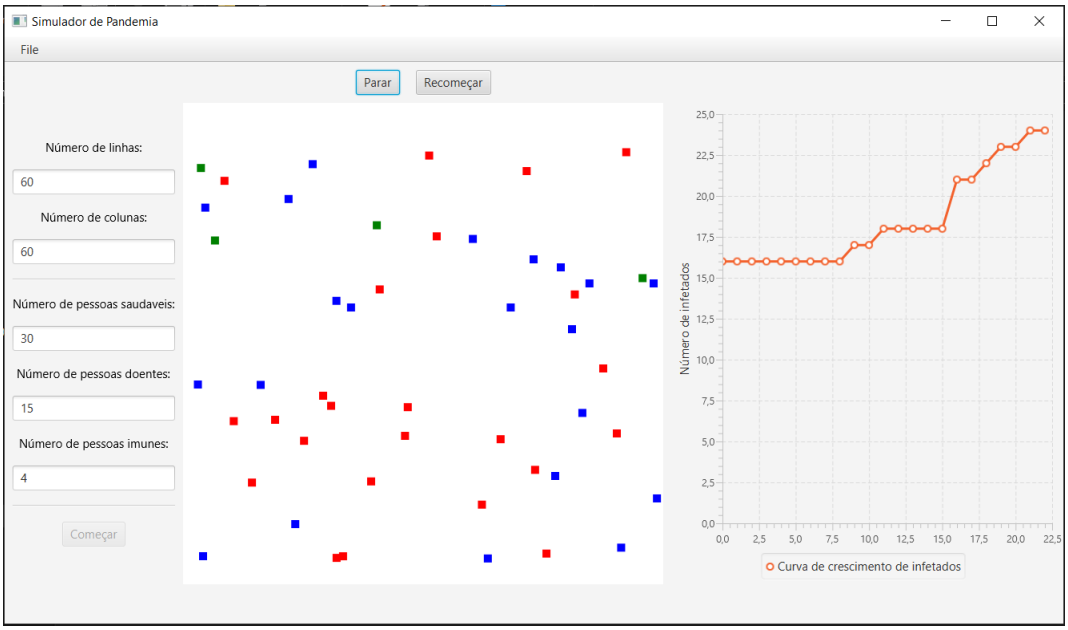

Figura 4: Interface do programa

De seguida está demonstrado o código de teste para o cenário de movimento de uma pessoa para uma célula vazia:

Listing 47: Método movePersonToEmptyCell

```
1
2       @Test
3       void movePersonToEmptyCell(){
4
5           World world = new World(new ViewTest(), 10, 10);
6
7           Person healhyPerson =  new HealhyPerson(new CellPosition(5,5), world, 1);
8
9           boolean validMove = world.movePerson(healhyPerson,5,4);
10
11          assertTrue(validMove);
12          assertEquals(5, healhyPerson.cellPosition.getLine());
13          assertEquals(4, healhyPerson.cellPosition.getCol());
14
15
16      }
```

Código de teste para tentativa de demovimento de uma pessoa para fora do tabuleiro em demasiado para a esquerda:

Listing 48: Método movePersonToInvalidPositionLeft

```
1
2       @Test
```

```
3       void movePersonToInvalidPositionLeft(){
4
5           World world = new World(new ViewTest(), 10, 10);
6
7           Person healhyPerson =  new HealhyPerson(new CellPosition(3,0), world, 1);
8
9           boolean invalidMove = world.movePerson(healhyPerson, 3, -1);
10
11          assertFalse(invalidMove);
12          assertEquals(3, healhyPerson.cellPosition.getLine());
13          assertEquals(0, healhyPerson.cellPosition.getCol());
14
15      }
```

Código de teste para tentativa de demovimento de uma pessoa para fora do tabuleiro em demasiado para a cima:

Listing 49: Método movePersonToInvalidPositionUp

```
1
2       @Test
3       void movePersonToInvalidPositionUp(){
4
5           World world = new World(new ViewTest(), 10, 10);
6
7           Person healhyPerson =  new HealhyPerson(new CellPosition(0,5), world, 1);
8
9           boolean invalidMove = world.movePerson(healhyPerson, -1, 5);
10
11          assertFalse(invalidMove);
12          assertEquals(0, healhyPerson.cellPosition.getLine());
13          assertEquals(5, healhyPerson.cellPosition.getCol());
14
15      }
```

Código de teste para tentativa de demovimento de uma pessoa para fora do tabuleiro em demasiado para a direita:

Listing 50: Método movePersonToInvalidPositionRight

```
1
2       @Test
3       void movePersonToInvalidPositionRight(){
4
5           World world = new World(new ViewTest(), 10, 10);
6
7           Person healhyPerson =  new HealhyPerson(new CellPosition(0,9), world, 1);
8
9           boolean invalidMove = world.movePerson(healhyPerson, 0, 10);
10
11          assertFalse(invalidMove);
12          assertEquals(0, healhyPerson.cellPosition.getLine());
13          assertEquals(9, healhyPerson.cellPosition.getCol());
14
15
16      }
```

Código de teste para tentativa de demovimento de uma pessoa para fora do tabuleiro em demasiado para a baixo:

Listing 51: Método movePersonToInvalidPositionDown

```
1
2     @Test
3     void movePersonToInvalidPositionDown(){
4
5         World world = new World(new ViewTest(), 10, 10);
6
7         Person healhyPerson =  new HealhyPerson(new CellPosition(9,0), world, 1);
8
9         boolean invalidMove = world.movePerson(healhyPerson, 10, 0);
10
11        assertFalse(invalidMove);
12        assertEquals(9, healhyPerson.cellPosition.getLine());
13        assertEquals(0, healhyPerson.cellPosition.getCol());
14    }
```

Os cenários de teste de contágio envolvem primeiramente iniciar duas pessoas, onde de seguida elas movem-se para a mesma posição e verificar como fica o estado de cada uma.

Código de teste para contacto de uma pessoa doente com uma pessoa saudável:

Listing 52: Método CollisionSickToHealhy

```
1
2     @Test
3     void CollisionSickToHealhy(){
4
5         World world = new World(new ViewTest(), 10, 10);
6
7         Person sickPerson = new SickPerson(new CellPosition(4, 5), world, 2);
8         Person healhyPerson =  new HealhyPerson(new CellPosition(6,5), world, 1);
9
10        world.movePerson(sickPerson,5,5);
11        world.movePerson(healhyPerson, 5,5);
12
13        world.testCollision(sickPerson, healhyPerson);
14
15        assertEquals(sickPerson.getState(), healhyPerson.getState());
16
17
18    }
```

Código de teste para quando as pessoas se movem mas não existe nenhum contacto:

Listing 53: Método NoCollision

```
1
2     @Test
3     void NoCollision(){
4
5         World world = new World(new ViewTest(), 10, 10);
6
7         Person sickPerson = new SickPerson(new CellPosition(4, 5), world, 2);
8         Person healhyPerson =  new HealhyPerson(new CellPosition(6,5), world, 1);
9
```

```
10          world.testCollision(sickPerson, healhyPerson);
11
12          assertEquals(healhyPerson.getState(), 1);
13
14
15      }
```

Código de teste quando duas pessoas doentes se intersectam:

Listing 54: Método CollisionSickToSick

```
1
2       @Test
3       void CollisionSickToSick(){
4
5           World world = new World(new ViewTest(), 10, 10);
6
7           Person sickPerson = new SickPerson(new CellPosition(4, 5), world, 2);
8           Person sickPerson2 = new SickPerson(new CellPosition(6, 5), world, 2);
9
10
11          world.movePerson(sickPerson,5,5);
12          world.movePerson(sickPerson2, 5,5);
13
14          world.testCollision(sickPerson, sickPerson2);
15
16          assertEquals(sickPerson.getState(), sickPerson2.getState());
17          assertEquals(sickPerson.getState(), 2);
18          assertEquals(sickPerson2.getState(), 2);
19
20
21
22      }
```

Código de teste de colisão de uma pessoa doente com uma pessoa imune:

Listing 55: Método CollisionSickToImune

```
1
2       @Test
3       void CollisionSickToImune(){
4
5           World world = new World(new ViewTest(), 10, 10);
6
7           Person sickPerson = new SickPerson(new CellPosition(4, 5), world, 2);
8           Person imunePerson = new SickPerson(new CellPosition(6, 5), world, 3);
9
10
11          world.movePerson(sickPerson,5,5);
12          world.movePerson(imunePerson, 5,5);
13
14          world.testCollision(sickPerson, imunePerson);
15
16          assertEquals(sickPerson.getState(), 2);
17          assertEquals(imunePerson.getState(), 3);
18
19      }
```