Logo

Description automatically generated

Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e Multimédia

Ano Letivo 2021/2022

Modelação e Simulação de Sistemas Naturais

Trabalho Pratico 1

Henrique Pereira 48571

Docente: Arnaldo Abrantes

Novembro 2021

Índice

[1. Jogo da Vida 3](#_Toc87637921)

[1.1. CellJDV 3](#_Toc87637922)

[1.1.1. Atributos 3](#_Toc87637923)

[1.1.2. Métodos 3](#_Toc87637924)

[1.2. JogoDaVida 3](#_Toc87637925)

[1.2.1. Atributos 3](#_Toc87637926)

[1.2.2. Métodos 3](#_Toc87637927)

[1.3. JDV23por2 4](#_Toc87637928)

[1.3.1. Métodos 4](#_Toc87637929)

[1.4. RunJDV 4](#_Toc87637930)

[1.4.1. Atributos 4](#_Toc87637931)

[1.4.2. Métodos 4](#_Toc87637932)

[2. DLA 5](#_Toc87637933)

[2.1. Walker 5](#_Toc87637934)

[2.1.1. Métodos 5](#_Toc87637935)

[2.2. DLA 5](#_Toc87637936)

[2.2.1. Atributos 5](#_Toc87637937)

[2.2.2. Métodos 6](#_Toc87637938)

[2.3. RunDLA 6](#_Toc87637939)

[2.3.1. Atributos 6](#_Toc87637940)

[2.3.2. Métodos 6](#_Toc87637941)

[2.4. WalkerInvertido 7](#_Toc87637942)

[2.4.1. Métodos 7](#_Toc87637943)

[2.5. DLAInvertido 7](#_Toc87637944)

[2.5.1. Métodos 7](#_Toc87637945)

[2.6. RunDLAInvertido 7](#_Toc87637946)

[2.6.1. Métodos 7](#_Toc87637947)

1. Jogo da Vida

Implementei o jogo da vida clássico (23/3) na classe JogoDaVida que herda a classe CellularAutomata desenvolvida na aula. Implementei a classe CellJDV que herda a classe Cell desenvolvida na aula. Implementei a classe RunJDV que implementa a classe IProcessingApp desenvolvida na aula.

Ainda completei os seguintes pontos facultativos:

* Implementar variantes (variante 23/2 na classe JDV23por2 que herda a classe JogoDaVida)
* Atribuir múltiplas cores às células de acordo com um dado critério
* Escolher a configuração inicial do autómato através de GUI e/ou leitura de ficheiro (apenas através da GUI)
  1. CellJDV
     1. Atributos

O atributo static deadColor armazena a cor das células mortas.

O atributo aliveColor armazena a cor da célula quando esta fica viva e é inicializado com uma cor aleatória.

* + 1. Métodos

O método getAliveNeigh() devolve um array de CellJDV com todas as células vizinhas que estão vivas, excluindo a célula que chamou o método do array retornado.

O métdodo getCommonNeighborColor() devolve o valor da cor aliveColor mais comum entre os vizinhos vivos da célula que chama o método. Caso a célula não tenha vizinhos vivos então devolve o valor da cor aliveColor da própria célula.

* 1. JogoDaVida
     1. Atributos

O atributo aliveCellsPercentage armazena o ratio de celulas vivas num valor entre 0 e 1.

* + 1. Métodos

O método createCells(Cell[][]) dá overwrite ao método na classe CellularAutomata e apenas cria células da classe CellJDV.

O método initRandom() dá overwrite ao método na classe CellularAutomata e coloca o estado da célula como viva aleatóriamente dependendo do valor do atributo aliveCellsPercentage.

O método update() cria um novo array de células onde guardará as células no seu estado seguinte. Para cada célula cálcula o seu próximo estado tendo em conta o seu número de vizinhos vivos. No final do processo atualiza o atributo cells com o novo array das células no seu estado seguinte.

* 1. JDV23por2
     1. Métodos

O método update() dá overwrite ao método na classe JogoDaVida e apenas altera as regras de mudança de estado.

* 1. RunJDV
     1. Atributos

O atributo running “ativa” e “desativa” a evolução do JogoDaVida.

O atributo aliveCellsPercentage define o ratio de celulas vivas num valor entre 0 e 1.

Os atributos nCols e nRows definem o tamanho do array de células.

* + 1. Métodos

O método setup(PApplet) cria um objeto da classe JogoDaVida ou da classe JDV23por2 dependendo de qual linha de código esteja descomentada. Após criar o objeto chama o método initRandom() do objeto criado e mostra o estado do jogo.

O método draw(PApplet) atualiza o estado do jogo para o seguinte apenas se o atributo running estiver a true, e mostra o estado do jogo.

O método mousePressed(PApplet) muda o estado da célula clicada para o estado contrário ao que se encontra.

O método keyPressed(PApplet) “ativa” ou “desativa” a evolução do jogo clicando na tecla de espaço, e evolui para o próximo estado do jogo clicando na tecla N.

1. DLA

Implementei o Diffusion-limited Agregation na classe DLA desenvolvida parcialmente na aula. Implementei a classe Walker desenvolvida parcialmente na aula. Implementei a classe RunDLA que implementa a classe IProcessingApp desenvolvida na aula.

Completei o algoritmo fazendo com que o número de partículas em movimento fosse sempre constante ao longo do tempo e também atribuí cor às partículas paradas.

Ainda completei os seguintes pontos facultativos:

* Implementar uma variante invertida em que as partículas se alastram do centro para fora. A classe DLAInvertido que herda a classe DLA. A classe WalkerInvertido herda a classe Walker. A classe RunDLAInvertido herda a classe RunDLA.
* Implementar o parâmetro stickiness que define a probabilidade da partícula parar quando entra em contacto com outra partícula parada.
  1. Walker
     1. Métodos

O construtor único Walker(DLA, State, PVector) apenas cria uma partícula com os atributos recebidos nos parâmetros.

O método setState(State) atualiza o estado da partícula e também a sua cor. A cor da partícula é branca caso esteja a vaguear, ou caso esteja parada então é uma cor específica dependendo da sua distância ao centro.

O método updateStateToStopped(List<Walker>) testa se a partícula deve passar para o estado parado. Altera o estado da partícula para parada e devolve true caso a condição de paragem se afirme, ou apenas devolve false caso contrário.

* 1. DLA
     1. Atributos

A lista WALKERS armazena todas as partículas presentes no DLA. As listas STOPPED e WANDERING armazenam apenas as partículas que se encontram no estado parado ou a vaguear, respetivamente.

O atributo N\_STOPPED\_COLORS define quantas cores diferentes as partículas paradas podem ter, e os atributos STOPPED\_COLOR\_CHANGE\_DIST e COLORS\_STOPPED servem para atribuir as cores às partículas paradas.

O atributo N\_WANDERING define quantas partículas estão a vaguear, WALKER\_RADIUS define o raio de cada partícula e PULL\_FORCE define a força de atração de cada partícula.

O atributo STICKINESS define a probabilidade de uma partícula que esteja a vaguear mude de estado quando entra em contacto com outra partícula parada.

* + 1. Métodos

O método inicializarWalkers() cria uma partícula parada no centro da tela e chama o método criarWanderingWalker() N\_WANDERING vezes.

O método criarWanderingWalker() cria uma partícula a vaguear numa posição aleatória da circunferência limite.

O método newWalker(Walker) coloca o Walker recebido na lista WALKERS e também na lista WANDERING ou STOPPED, dependendo do seu estado.

O método updateWanderers() evolui o estado do DLA para a próxima iteração. Trata de atualizar a posição de cada partícula que está a vaguear, atualizar o seu estado e criar novas partículas no lugar das que mudaram de estado.

O método updateDisplay() desenha o background, a circunferência limite e cada partícula na lista WALKERS.

O método limitReached() define a condição de paragem (uma partícula parada estar em contacto com a circunferência limite) e devolve true ou false dependendo se essa condição é cumprida ou não.

* 1. RunDLA
     1. Atributos

O atributo N\_WANDERING define quantas partículas estão a vaguear.

O atributo running “ativa” e “desativa” a evolução do DLA.

* + 1. Métodos

O método draw(PApplet) chama o método update() apenas se o atributo running estiver a true e a condição de paragem ainda não tenha sido verificada, e mostra o estado do jogo.

O método mousePressed(PApplet) cria uma nova partícula a vaguear na posição clicada caso a condição de paragem ainda não tenha sido verificada.

O método keyPressed(PApplet) “ativa” ou “desativa” a evolução do jogo clicando na tecla de espaço, e evolui para o próximo estado do jogo clicando na tecla N caso a condição de paragem ainda não tenha sido verificada.

O método update() atualiza o estado do DLA para a próxima iteração caso a condição de paragem ainda não tenha sido verificada.

* 1. WalkerInvertido
     1. Métodos

O método updateStateToStopped(List<Walker>) dá overwrite ao método na classe Walker e adiciona a condição de trocar de estado caso a partícula entre em contacto com a circunferência limite.

* 1. DLAInvertido
     1. Métodos

O método getPULL\_FORCE() dá overwrite ao método na classe DLA e retorna o valor negativo de PULL\_FORCE, para dessa forma repelir as partículas do centro ao invés de as atrair.

O método inicializarWalkers() dá overwrite ao método na classe DLA e já não cria a partícula parada no centro da tela.

O método criarWalkerWandering() dá overwrite ao método na classe DLA e cria a partícula no centro da tela.

O método limitReached() dá overwrite ao método na classe DLA e redefine a condição de paragem para quando uma partícula parada se encontra no centro da tela.

* 1. RunDLAInvertido
     1. Métodos

O método mousePressed() dá overwrite ao método na classe RunDLA e apenas cria uma da classe WalkerInvertido.