

**Trabalho Prático 1**

**Jogo da Vida / DLA**

**Licenciatura:**

Engenharia Informática e Multimédia

**Unidade Curricular:**

Modelação e Simulação de Sistemas Naturais

**Docente:**

Arnaldo Abrantes

**Alunos:**

Henrique Matos nº 48608

João Gonçalves nº 47507

**Turma:**

33D

Contents

[Introdução 3](#_Toc87727507)

[Diffusion-Limited Aggregation (DLA) 4](#_Toc87727508)

[- Classe Walker 4](#_Toc87727509)

[- Classe DLA 6](#_Toc87727510)

# Introdução

Este primeiro trabalho prático, dividido em duas partes (Jogo da Vida e DLA), pretende simular virtualmente processos de evolução de elementos em comunidades e a agregação de objetos caóticos num determinado espaço. Tais simulações foram feitas em Java, utilizando as ferramentas de Processing.

A primeira parte pretende representar o autómato celular, Jogo da Vida (criado em 1970 pelo matemático *John Horton Conway*), cuja finalidade seria estudar a evolução de comunidades ecológicas, aplicando as regras e princípios de ações genéticas e os seus processos de desenvolvimento, chegando à conclusão que estes são inseparáveis no que toca à sustentabilidade dum sistema biológico. Tal jogo explora as hipóteses do desenvolvimento de um sistema com base nos princípios de sobrepopulação e subpovoamento, sendo que em ambos os casos a célula não sobreviverá, necessitando duma vizinhaça sustentável, nem muito sufocante nem muito escassa.

A segunda parte do trabalho representa o processo de DLA (*Diffusion-Limited Aggregation*) cuja finalidade será representar a popular *Brownian Tree*, estudando a agregação de partículas com movimento caótico, num determinado espaço/ambiente cujas forças permitem tal agregação a partículas vizinhas num estado parado e agregado. O modelo de DLA é frequentemente utilizado para o estudo de crescimento dendrítico (ramificado), sendo que com a simulação da agregação de partículas é possível melhor compreender a criação de estruturas como, por exemplo, um floco de neve.

# Diffusion-Limited Aggregation (DLA)

Dissecando o nome *Diffusion-Limited Aggregation*, entendemos que *Diffusion* refere-se ao movimento difuso e caótico das particulas envolventes, até ao momento em que se agregam umas às outras (*Aggregation*). A junção *Diffusion-Limited*, refere-se ao facto das particulas estarem em baixas concentrações de modo a que não entrem as particulas em contacto umas com as outras, assegurando que a estrutura cresce num todo em vez de ser uma junção de aglomerados, permitido que seja formada a ramificação da mesma.

Para estudar este processo de agregação, criou-se duas classes: a classe **Walker** e a classe **DLA**.

# - Classe Walker

Começaremos então com a classe Walker. Esta classe é responsável pelo comportamento individual da particula (Walker), possuindo um construtor e tratando métodos que atualizam o estado do Walker em si e da colisão.

Definimos as variáveis **pos** (posição), **state** (estado), **colour** (cor), **radius** (raio), **num\_wanders** (numero de walkers em movimento), **num\_stopped** (numero de walkers parados).

Graphical user interface, text, chat or text message, website

Description automatically generatedNo construtor, definimos a posição inicial do primeiro **Walker**, sendo esta no meio da janela, e definimos o estado como “Wander” para que possa haver movimento nos adjacentes. Definimos também um vetor para orientar a forma da força da agregação, atribuindo o vetor à posição inicial dos walkers, para que se desloquem com o formato de movimento pretendido. Este construtor, tal como os métodos descritos abaixo, passam o argumento **Papplet**, já que esta é a classe responsável pelo tratamento gráfico de todos os elementos em Processing.

Definimos um segundo construtor **Walker**, passando-se como argumento o Papplet e o vetor **pos**. Este construtor é responsável pelo Walker inicial, que se inicia parado, tal como se define com a função *setState*, que será explicada abaixo. Este Walker inicial será o catalizador para a formação ramificada da estrutura simulada.

Definindo então o método **setState**, passamos como argumento **PApplet**, e o enum **State**, que define o estado do Walker. Este estado, usa uma das duas constantes (já que um **enum** é uma coleção de constantes definidas pelo utilzador). Dentro deste método também se estabeleceu um limite através de intervalos de Walkers parados, em que a cor muda a cada intervalo. O métodod também conta o número de Walkers parados e em movimento.

Desenvolveu-se também um método **getState** de modo a obter o estado do Walker.

Para atuailizar o estado, criou-se o método **updateState**, passando dois argumentos em que um é o **PApplet** e o outro é uma lista de Walkers. Este método irá atualizar o estado do Walker através de deteção de colisão. Através de um ***if***, caso o estado seja já parado, não será necessário fazer nada. Após esta verificação, cria-se um ***for loop***que cria e percorre a lista de Walkers, adicionado cada Walker à lista, adicionando dentro do loop um ***if*** que define a distância necessária para a deteção de colisão. Caso o Walker esteja em movimento e toque num já parado, muda de estado para **STOPPED**, daí o nome de updateState do método.

Existe também o método **wander** que passa o argumento **PApplet**, responsável pelo movimento dos Walkers, definindo as direções e posições de todos os elementos. Este método faz uso do vetor de forças, usando uma classe do processing chamada **lerp()** que permite modificar o centro de ação e a velocidade de aglomeração (*stickiness*).

Por fim, o último método da classe **Walker** é o método **display** que passa o argumento **PApplet**, que serve para demonstrar os walkers criados, que terão uma forma circular e a cor que será definida pelo programador.

# - Classe DLA

Esta classe é responsável pela aglomeração dos Walkers, definidos na classe anterior. Passa as variáveis duma lista de walkers, o número de walkers (**NUM\_WALKERS**) e o numero de ações por frame de evento (**NUM\_STEPS\_PER\_FRAME**).

Text

Description automatically generatedEstabelece-se o método **setup** que passa o argumento **PApplet** em que se define a lista de walkers a ser usada e adiciona-se um walker, criado com o objeto Walker, passado através da classe **Walker**. Cria-se e enche-se esta lista de walkers (passada através dum ArrayList), baseando-se no número de walkers, definido na variável **NUM\_WALKERS**.

O método responsável pela concatenação de todos os métodos, dando origem à visualização da simulação das particulas é o método **draw** que passa or argumento **PApplet** e um float **ft**. Este método define a cor de fundo da janela, que neste caso é preto, definida através da classe background em PApplet, com o valor de RGB 0. Aqui definimos uma variável “parados” que será explicada de seguida.

Possui um ***for loop*** com um outro ***nested for loop*** que serve para, caso o estado seja em movimento (WANDER), damos update ao estado do walker. Caso o estado seja parado (PARADO), criamos um segundo ***if*** que caso o numero de walkers parados seja menor que 1200, continuar-se-à a contar o numero de walkers parados, sendo para isso que serve a variável definida anteriormente “parados”.

Fora deste ***for loop*** principal, criamos um novo ***for loop*** que, por cada walker parado, adiciona-se um novo walker à lista de Walkers, para que por cada walker parado, adicione-se um novo, tendo sempre um número de walkers em movimento constante.

Usamos também um outro ***for loop*** que percorre todos os walkers na lista de walkers, para que possamos dar display a todos os walkers, através do método **display** da classe **Walker**.

Por fim, damos ***print*** ao numero de walkers em movimento e parados na consola.

Assim sendo, eis o resultado final:

A picture containing outdoor object

Description automatically generated

Temos então acima o resultado final da simulação de particulas, formando uma ***Brownian Tree***, finalizando assim o DLA.