Projeto Final

Simulação de Ecossistema

Licenciatura:

Engenharia Informática e Multimédia

Unidade Curricular:

Modelação e Simulação de Sistemas Naturais

Docente:

Arnaldo Abrantes

Alunos:

Henrique Matos nº48608

João Gonçalves nº 47507

Turma:

33D

Índice

Tabela Imagens

1. Introdução

O projeto em si realizado tem como base a aplicação e concatenação das matérias leccionadas na unidade curricular de Modelação e Simulação de Sistemas Naturais, juntando teoria de aula à prática fornecida pelos diversos materiais presentes em Moodle.

A simulação realizada permite a verificação de comportamentos de espécies que fazem parte duma determinada cadeia alimentar, em que os predadores dominam as presas e a reprodução das espécies apresenta a evolução da própria a nível físico e cognitivo.

Estando presente um ecossistema diverso em termos de seres vivos e de modalidades de terreno, permite que seja observada a luta pela sobrevivência e por alimento que leva à reprodução e consequente sobrevivência dos mais aptos e dos seus descendentes.

2. Desenvolvimento do Projeto

2.1 Narrativa

O ecossistema demonstra comportamentos entre três espécies diferentes pertencentes à mesma cadeia alimentar, que não só interagem entre si, como também interagem com o ambiente à sua volta, levando a que este último seja alterado conforme as ações das espécies.

Possuimos nesta narrativa, 3 espécies diferentes:

- Presa, representada por uma imagem de coelho

- Predador, representada por uma imagem de lobo

- (Predador) Apex, representada por uma imagem de um ser humano

O ambiente em que se inserem é representado por uma mistura de terreno fértil, com alimento e de obstáculos, nomeadamente obstáculos regenerativos (lagos e rios) e obstáculos perigosos (terreno hostil).

A cadeia alimentar descreve-se da seguinte maneira:

- As presas comem o alimento presente em terreno fértil, ou seja, são animais herbívoros
- Os predadores comem as presas
- Os predadores Apex comem as presas e os predadores comuns, sendo que o valor nutritivo dos predadores comuns (lobos) é superior aos das presas (coelhos)
- Podemos descrever o coelho (presa) como sendo herbívoro e ambos os predadores e predadores Apex sendo carnívoros

Quando o alimento do terreno é consumido pelos coelhos, demora 20 segundos a regenerar de modo a que haja uma limitação na quantidade de energia por iteração do programa que pode ser obtida por parte desta espécie, já que tais mudanças para menos tempo poderiam levar a uma sobre-população por parte dos coelhos. Tal não traria nenhum problema específico para as outras espécies já que haveria mais alimento mas dum ponto de vista de simulação, não seria ideal para observar os variados comportamentos e interações entre as espécies.

Integrado no ecossistema está também a possibilidade de criar árvores, por parte do utilizador, sendo esta meramente ilustrativas, não interagindo com o ecossistema.

Dum modo geral, já que as presas se alimentam do terreno, os predadores das presas e os predadores Apex dos predadores comuns e das presas, sabemos que caso as presas todas morram, automaticamente o ecossistema deixa de ser sustentável, levando à morte de todas as espécies presentes e à persistência permanente dum terreno fértil e cheio de alimento. Eis a cadeia:

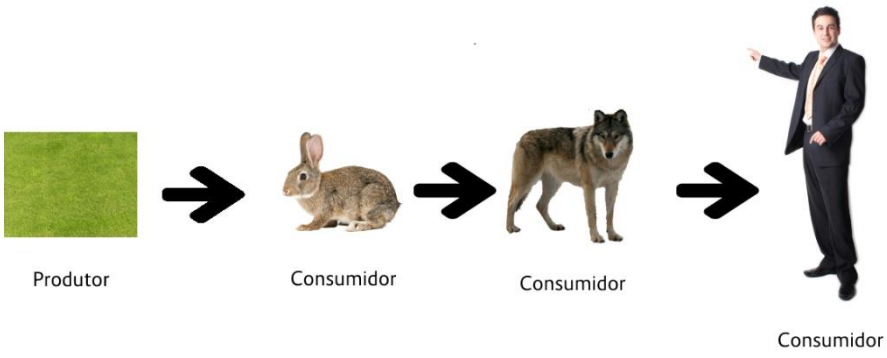


Figura 1- cadeia alimentar

2.2 Comportamentos Observados

Ao longo da simulação observou-se os seguintes comportamentos particulares:

- Inicialmente os predadores atacam em grupo, enquanto que as presas vagueiam simplesmente pelo ambiente em que se encontram todos
- Caso haja um lago, onde a reprodução das presas se acentua devido à regeneração energética, os predadores mantêm-se na mesma região devido à abundância de alimento, o que levará consequentemente a um aumento da população de predadores e a um ciclo vicioso na mesma região do mapa
- Caso seja inserido um obstáculo, por parte do utilizador, na zona de alimento abundante referida acima, automaticamente as espécies fogem já que estão pre-determinadas a afastarem-se dos obstáculos (será explicado mais à frente juntamente com o desenvolvimento do código) – comportamento '*avoidObstacle*'
- Confome o tempo passa e as espécies sofrem mutações, evoluem de forma a tornarem-se mais inteligente e independentes, sendo que os predadores começam a caçar de forma mais solitária, mais rápida e mais inteligente, sendo que o mesmo também se observa nas presas em que conseguem fugir dos obstáculos e dos predadores de forma mais eficaz

Estes comportamentos que se observam devem-se a um condicionamento comportamental inicial que é definido pelo código, sendo que à medida que as espécies se reproduzem, as mutações da espécie levam a uma evolução cognitiva e física na simulação o que as leva a fugir dos obstáculos de modo mais rápido e eficaz, sabendo que todas as espécies estão condicionadas a evoluir de forma a melhorar a sobrevivência num ambiente maioritariamente hostil.

2.3 Explicação do código desenvolvido

Duma forma inicial, apresentamos que as classes que definem a presa, predador e predador Apex são as classes Prey, Predator e Apex, que extendem a classe Animal, a classe mãe.

Cada animal cria um objeto DNA na sua classe que permite caracterizar os seus comportamentos, sendo que esta dna passa para os seus filhos, mutando e simulando a reprodução de espécies, levando à evolução das espécies.

Tendo em conta que a simulação apresentada é baseada em comportamentos evolutivos e mutações de espécies, alterou-se o método responsável pelas mutações dos comportamentos na classe *Boid*.

Para tal, programou-se mudanças nos pesos das forças comportamentais de modo a que os filhos das espécies pudessem perceber melhor como agir conforme as ameaças do ambiente. Assim sendo, usou-se um *for loop* que percorre todos os comportamentos dos Animais, sendo que ao encontrar uma instância pertencente ao animal em questão, mudar-se ia o comportamento associado.

```
public void mutateBehaviors() {
    for(Behavior behavior : behaviors){
        if(this instanceof Prey){
            if (behavior instanceof AvoidObstacle) {
                behavior.weight += DNA.random(-0.5f, 0.5f);
                behavior.weight = Math.max(0, behavior.weight);
            }
            updateSumWeights();
        }

        if(this instanceof Predator){
            if(behavior instanceof Pursuit){
                behavior.weight += DNA.random(-0.2f, 0.2f);
            } else {
                behavior.weight += DNA.random(0f, 0.1f);
            }
            behavior.weight = Math.max(0, behavior.weight);

            if(behavior instanceof Wander){
                behavior.weight += DNA.random(0.5f, 0.9f);
                behavior.weight = Math.max(0, behavior.weight);
            }

            if(behavior instanceof AvoidObstacle){
                behavior.weight += DNA.random(-0.5f,0.5f);
                behavior.weight = Math.max(0, behavior.weight);
            }
```

Figura 2- exemplo de mutação comportamental

Para que tal pudesse acontecer, mudou-se no DNA o construtor relativo à mutação, em que se passou um parâmetro *string* estando relativo ao método de mutação da classe DNA, em que a velocidade máxima muda conforme as mutações, sendo que caso a string no construtor corresponda à classe, extendida de Animal (Prey, Predator, Apex), usada em questão.

```
private void mutate(String str) {
    if(str == "Prey"){
        maxSpeed += random(-0.1f, 0.1f);
        maxSpeed = Math.max(0, maxSpeed);
    }
    if(str == "Predator"){
        maxSpeed += random(-0.2f, 0.2f);
        maxSpeed = Math.max(0, maxSpeed);
    }
    if(str == "Apex"){
        maxSpeed += random(-0.4f, 0.4f);
        maxSpeed = Math.max(0, maxSpeed);
    }

}
```

Figura 3- atribuição de parâmetros à classe

Assim, com este *mutate*, toda as classes derivadas de Animal inclusive, possuem um parâmetro no seu construtor responsável pela identificação da sua classe.

É relevante referir que como tanto as presas como os dois tipos de predadores são classes derivadas da superclasse Animal, temos presente o facto de os seus métodos de comer, reproduzir e movimentar sejam derivados da mesma.

No entanto, os predadores possuem uma diferença. Como não se alimentam do terreno, criou-se um novo método *eat* responsável pelo consumo de animais, em que cada predador identifica a presa, calcula a distância de si ao alimento, dirige-se a ela e come.

```java
public List<Animal> eat (List<Animal> animals){
    PVector mePos = this.getPos().copy();

    for(Animal animal : animals){
        if(animal instanceof Prey){
            PVector preyPos = animal.getPos().copy();
            float distance = PVector.dist(mePos, preyPos);
            if(distance < this.radius*4){
                energy += WorldConstants.ENERGY_FROM_PREY;
                animals.remove(animal);
                return animals;
            }
        }
    }
    return animals;
}
```

Figura 4- método para comer presas

Como é importante que os alvos dos predadores se atualizem, certificou-se que foi criado um método para que mude o alvo (presa), em que se vai buscar a presa mais próxima, calculando a distância com base no parâmetro de *safeDistance* da classe DNA, em que para cada animal caso a distância esteja muito longa, o predador dirige-se à próxima presa mais perto.

```java
public void updateTargetPrey(PApplet p, List<Body> obstacles, List<Animal> allPrey) {
    float dist = 0;
    if (!obstacles.contains(this.target) && this.target != null) {
        //distancia entre o animal e a prey
        dist = PVector.dist(this.getPos(), this.target.getPos());
        if (dist > this.getDNA().visionSafeDistance) {
            for (Animal animal : allPrey) {
                float newDist = PVector.dist(this.getPos(), animal.getPos());
                if (newDist < dist && this.target != animal && animal instanceof Prey) {
                    this.target = animal;
                    //System.out.println("alvo alterado");
                }
            }
            List<Body> tempTrackingBodies = new ArrayList<~>();
            //adicionar o target e obstaculos a lista
            tempTrackingBodies.add(this.target);
            tempTrackingBodies.addAll(obstacles);
            //criar um novo Eye com essa lista
            Eye eye = new Eye( me: this, tempTrackingBodies);
            this.setEye(eye);
        }
    }
}
```

Figura 5- método de mudança de alvo

É importante referir que, como foi explicado acima, existem 5 variações do mesmo terreno, dentro do mapa:

- *EMPTY*
- *OBSTACLE*
- *FERTILE*
- *FOOD*
- *WATER*

Respetivamente, terreno vazio, obstáculos hostis, terreno fértil responsável pelo aparecimento de comida, comida das presas, água.

De acordo com o código desenvolvido, o terren vazio simplesmente não possui nenhum qualidade, é terreno neutro.

O terreno de obstáculos define zonas hostis que removem energia às espécies, levando à sua morte caso atravessem, dependendo da sua quantidade de energia inicial/corrente.

O terreno fértil serve para gerar comida, demorando 20 segundos após as presas se aliementarem do mesmo.

O terreno comida é alimento para as presas, fornecendo-lhes a quantidade máxima de energia; é a fonte primária de energia para os coelhos.

O terreno água possui um comportamento contrário ao obstáculo, já que regenera energia, embora em pouca quantidade comparado com o alimento, servindo apenas como meio para as espécies se sustentarem por um maior período de tempo.

Definiu-se também que cada espécie possui uma quantidade inicial de energia diferente, sendo que também cada espécie recebe maior energia dependendo do alimento em questão. Por exemplo, o predador Apex recebe maior energia se alimentar-se de predadores comuns, sendo que ao alimentar-se de presas coelhos recebe menos energia por serem mais pequenas e num estágio mais inicial da cadeia alimentar.

Cada espécie também possui uma quantiade diferente de energia necessária para se reproduzir sendo que os coelhos reproduzem-se muito mais rapidamente que os lobos, e os homens são os mais lentos.

```
public final static float PREY_SIZE = .1f;
public final static float PREY_MASS = 1f;
public final static int INI_PREY_POPULATION = 25;
public final static float INI_PREY_ENERGY = 10f;
public final static float ENERGY_FROM_PLANT = 4f;
public final static float PREY_ENERGY_TO_REPRODUCE = 60f;

public final static float PREDATOR_SIZE = .12f;
public final static float PREDATOR_MASS = 1.2f;
public final static int INI_PREDATOR_POPULATION = 10;
public final static float INI_PREDATOR_ENERGY = 60f;
public final static float ENERGY_FROM_PREY = 25f;
public final static float PREDATOR_ENERGY_TO_REPRODUCE = 80f;

public final static float APEX_SIZE = .20f;
public final static float APEX_MASS = 1.2f;
public final static int INI_APEX_POPULATION = 4;
public final static float INI_APEX_ENERGY = 60f;
public final static float APEX_ENERGY_FROM_PREDATOR = 10f;
public final static float APEX_ENERGY_FROM_PREY = 3f;
public final static float APEX_ENERGY_TO_REPRODUCE = 200f;
```

Figura 6- constantes do mundo relativo às espécies

Responsável pela gestão dos recursos em conformidade com os comportamentos das populações, o método Population descreve como se devem comportar as espécies face a perigo ou alimento.

Neste método, o seu construtor possui uma inicialização às listas de animais em que por cada conjunto de animais, definimos os seus comportamentos (Wander para vaguear, AvoidObstacle para fugir aos obstáculos e Pursuit para perseguir as presas).

Neste método também se definem os olhos, derivados da classe Eye, para que cada espécie possa identificar os alimentos ou perigos de modo a desviar-se ou aproximar-se.

Possuimos um método para que cada alvo possa ser atualizado já que é necessário que cada animal procure por vários animais, identificando-os.

```
for(int i = 0; i < WorldConstants.INI_PREDATOR_POPULATION; i++){
    PVector pos = new PVector(parent.random((float)window[0],(float)window[1]),
            parent.random((float)window[2],(float)window[3]));

    Animal a = new Predator(pos, WorldConstants.PREDATOR_MASS, WorldConstants.PREDATOR_SIZE, color: 0, parent, plt, imgPredator);
    a.addBehavior(new Wander( weight: 1));
    a.addBehavior(new Pursuit( weight: 1));;
    a.addBehavior(new AvoidObstacle( weight: 3));


    List<Body> allTrackingBodies = new ArrayList<>();

    Prey target = (Prey) allAnimals.get((int)parent.random( low: 0, allAnimals.size()));

    allTrackingBodies.add(target);
    a.target = target;
    allTrackingBodies.addAll(obstacles);

    Eye eye = new Eye(a, allTrackingBodies);
    a.setEye(eye);

    allPredator.add(a);
}
```

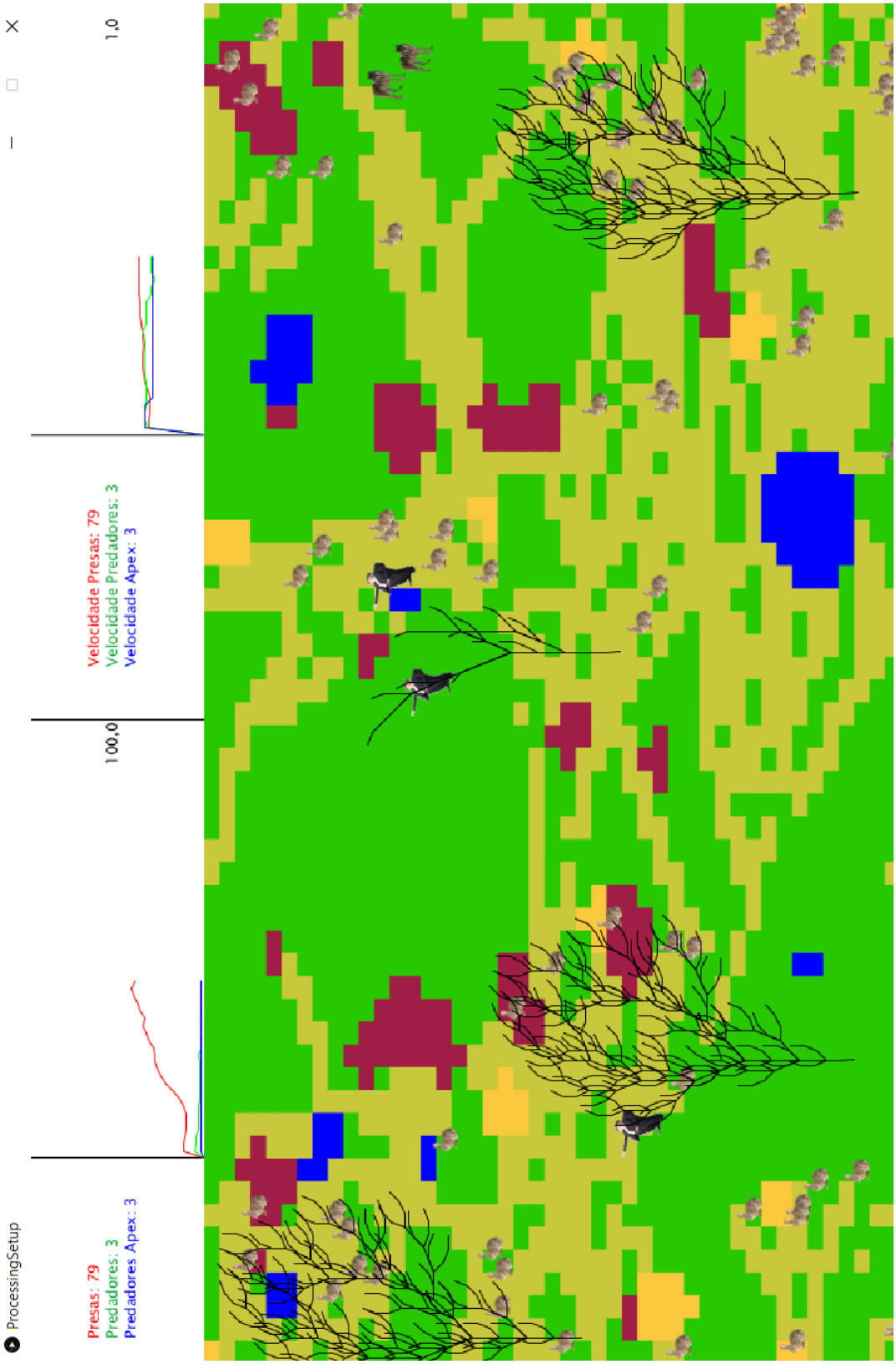
Figura 7- exemplo de atribuição comportamental à espécie

Para que se pudesse concatenar o trabalho todo numa só aplicação, criou-se a classe TestEcosystemApp de modo a que se observe a simulação por completo, tendo presente também gráficos que representam duas variáveis: a velociadade média da espécie e o número de indivíduos presente por cada população de espécies.

Assim permite ao utilizador melhor observar os parâmetros de simulação, sabendo o que está a ocorrer enquanto observa.

É nesta classe também que definimos a interação por parte do utilizador de modo a que possa ter instruções para a interação e que saiba a narrativa descrita pela simulação.

3. Produto Final



```
Bem vindo à simulação dum ecossistema!
Neste ecossistema estão presentes 3 espécies diferentes: presa, predador e o predador Apex!
Como presa temos um coelho, como predador um lobo e como apex é o próprio ser humano!
Ao longo do tempo observamos como os predadores buscam comida e como as presas pretendem encontrar comida.
Se tocam nos obstáculos a vermelho, os seres morrem. Se tocam na água ganham mais energia.
As presas comem pastagem, os predadores comem as presas e os apex comem os predadores!
De acordo com a quantidade de alimento ingerido, ganham mais energia o que leva à reprodução, criando filhos
que herdam as características dos pais, aprendendo também com os seus erros!
Mas atenção, pouco alimento e os seres morrem!
É completamente interativo:
   tecla 1 e arrastar clique de rato para criar lagos
   tecla 2 e arrastar clique de rato para cirar obstáculos
   tecla 3 e apenas clique de rato na posição pretendida para criar árvores. (meramente ilustrativo)
Divirta-se com esta história!


Está a criar lagos!
Está a criar obstáculos!
Está a criar árvores!
```

Figura 8- tutorial e apresentação à simulação (para utilizadores)

4. Diagrama UML

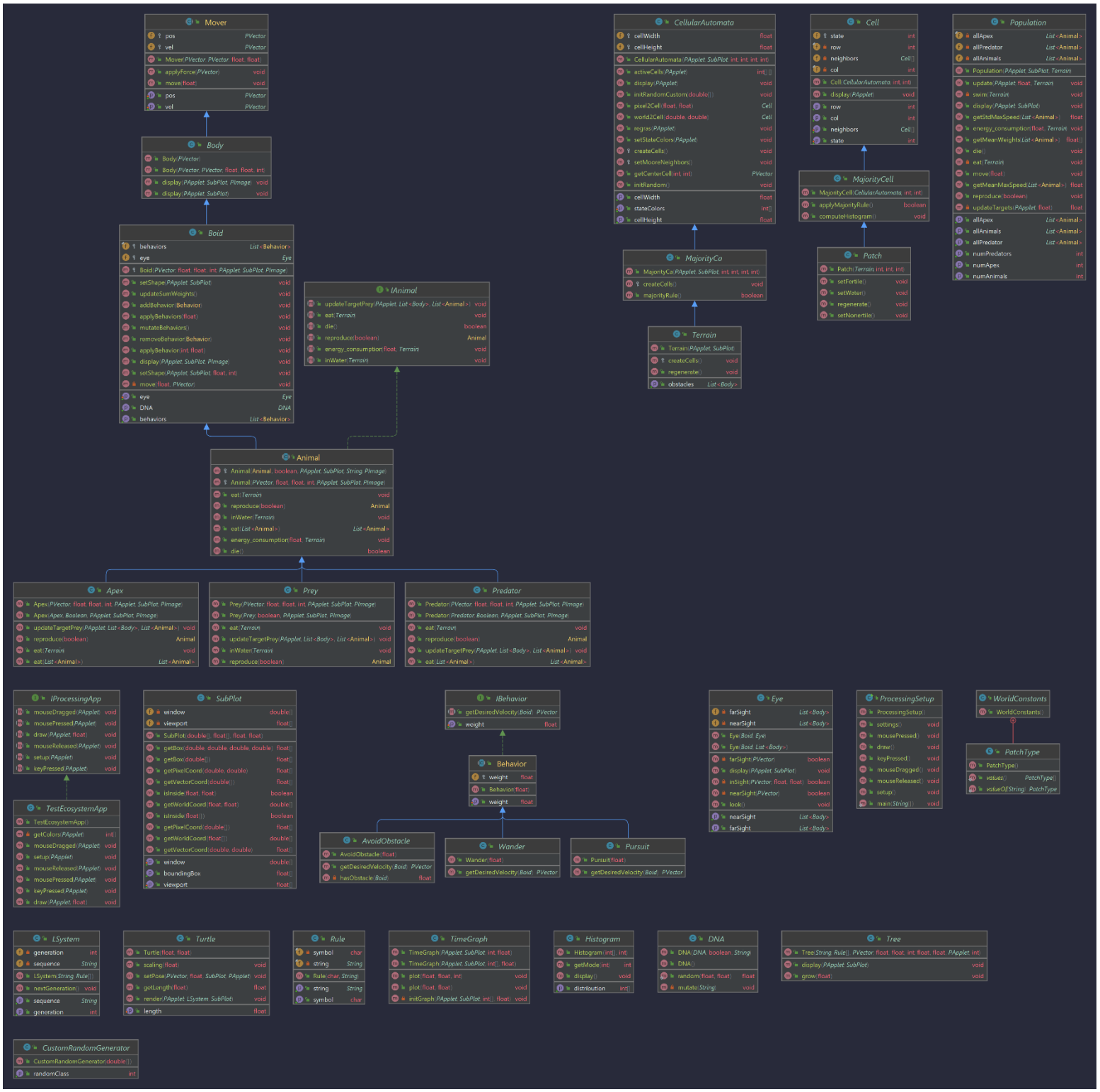


Figura 9- diagrama UML

5. Conclusão

Pondo em prática dum modo final a todos os assuntos abordados em aula e às matérias aplicadas de modo prático, poderemos concluir a importância da simulação em computador dos diversos comportamentos de espécies, já que nos providencia com métodos de análise empírica em situações que abrangem o considerado 'ideal' ao não-ideal.

A evolução das espécies observa-se que ocorre não de modo linear. É caótica já que não existe um padrão observável, caindo sob o tópico estudado na unidade curricular da teoria do caos já que, até por input do utilizador, a mínima perturbação causada no ecossistema é suficiente para alterar por completo o padrão de comportamentos, podendo até mesmo levar à extinção das espécies.

Aliás, podemos mesmo observar que basta que uma espécie intermédia (predadores) diminua que automaticamente existe uma abundância larga de presas, no entanto basta uma pequena divergência de comportamentos em que se inserem mais obstáculos ou lagos no mapa e automaticamente é aproximadamente impossível analisar o que poderia se desenrolar futuramente na evolução destas.

Foi de facto um projeto bastante interessante a desenvolver, já que pode-se observar a quantidade de fatores que entram na sustentabilidade e resiliência dum ecossistema, sabendo que o sistema em questão é frágil – qualquer alteração pode desencadear uma sucessão de acontecimentos que levará a alterações de grande escala.