****

ADDETC – Área Departamental de Engenharia Eletrónica e Telecomunicações e de Computadores

LEIM -Licenciatura Engenharia informática e multimédia

Modelação e Simulação de Sistemas Naturais

Relatório Trabalho Final

**Turma:**

LEIM-51N

**Trabalho realizado por:**

Vasco Cruz Nº42837

Pedro Henriques Nº45415

**Docente:**

Vítor Almeida

**Data:**24/02/2021

# Introdução

Este relatório pretende explicar o processo de desenvolvimento do trabalho final da disciplina de Modelação e Simulação de Sistemas Naturais (MSSN).

O trabalho final de MSSN, consistiu na criação e desenvolvimento de um jogo que aplica os conceitos lecionados em aula tais como modelação de agentes (boids), forças, sistemas de partículas e sistemas complexos.

O jogo é um clone, implementado em processing do antigo jogo Space Invaders, com algumas alterações que ajudam a refletir a matéria aprendida.

Devido á natureza do jogo, não foi possível implementar explicitamente certas partes da matéria, no entanto estas poderão ser mencionadas neste relatório por terem sido utilizados como fundamento teórico de outras partes do trabalho.

## Objetivos

De um ponto de vista mais técnico, este jogo surge no culminar da unidade curricular de MSSN e como tal irá abordar temas tais como:

-Modelação baseada em agentes (inimigos e jogador serão agentes)

-Forças (aplicadas aos agentes, levam a movimento)

-Sistemas complexos. (Interação entre os agentes)

Desta forma iremos usar código elaborado por nós, classes fornecidas ao longo do semestre pelos docentes e outras partes pesquisadas online para resolver bugs pontuais que foram surgindo.

## O Jogo

Como mencionado em cima o jogo é um clone de Space Invaders. Neste jogo um “exército” de aliens ataca a Terra e um Super soldado foi criado para combater a invasão.

O nosso soldado, movimenta-se no solo e vai apanhando e disparando os ossos dos seus camaradas que morreram antes dele contra os inimigos que cada vez mais se aproximam dele e colocam em risco a sua vida e a de todos os habitantes nesse planeta.

O jogo acaba, quando o soldado vença o inimigo final (depois de destruir todos os minions), ou com a morte do jogador e consequente destruição do seu planeta.

O jogo tem de possuir forma de:

- Verificar de vitoria/derrota

- Atribuir pontos ao jogador

- Movimentar e aplicar forças a objetos e entidades

- Mostrar menus que correspondem ao estado do jogo (menu inicial, menu jogo, pausa, etc…).

- O jogador atacar

- Morrer (tanto para o jogador como os inimigos) e verificação visual do ato.

- Animações para mover os objetos para originar

- Música de fundo

## Introdução teórica

### Agentes ou Individuo

Um agente, no contexto do nosso trabalho é um elemento capaz de interagir com o ambiente e com outros agentes, com autonomia podendo ou não efetuar decisões com alguma inteligência.

No nosso trabalho, tanto o jogador como o *boss* final são agentes que tomam decisões baseados no seu ambiente. O jogador é um humano por isso a sua inteligência depende somente do utilizador. O *boss* é um agente “semi-inteligente” que tenta destruir o jogador acertando-lhe com bombas, ajustando a sua posição para lançar as mesmas em cima do jogador.

### Forças

Num ambiente não computacional os elementos movem-se seguindo leis físicas bem definidas. Nomeadamente, o movimento ocorre após a aplicação de uma ou mais forças. Tendo em conta a segunda lei de Newton em que , é correto dizer que . Ou seja, aplicando uma força F a um corpo de massa m o corpo irá ter uma aceleração a. Assim através das equações do movimento e , podemos parametrizar o movimento do corpo segundo qualquer um dos eixos, após termos fornecido uma força, F.

Ou seja, se for necessário calcular qual a posição do corpo x, no instante t, é necessário resolver a equação diferencial em que e .

No entanto o nosso programa, não consegue ser executado em todos os instantes pelo que estas equações de movimento não são as mais adequadas. É necessário proceder a uma aproximação das posições, através do Método de Euler

### Método de Euler

O método de Euler é um processo através do qual, aproximamos uma equação diferencial através de iterações sucessivas seguindo a equação.

Adaptando ao nosso contexto a posição x(n) é dada por

Em que é o intervalo entre frames, é a velocidade inicial do corpo e a aceleração corresponde á força F, a dividir pela massa do corpo. Desta forma obtemos a melhor aproximação possível para o movimento dos nossos objetos e mantemos uma relação de proximidade com a realidade.

### Sistemas de Partículas

Um sistema de partículas é um objeto especial composto por outros objetos muito pequenos (partículas). O objetivo deste sistema é a simulação de comportamentos de corpos que se estejam a desfazer ou cujo movimento das suas muitas partes ofereça um custo computacionalmente elevado e como tal não seja eficiente a representação partícula a partícula.

# Desenvolvimento

## Diagrama de classes

Antes de começar a implementar o trabalho foi realizada um diagrama de classes que iria reger o processo de desenvolvimento esta classe não foi seguida a 100% pelo que existe objetos presentes no código final que não estão aqui representados, nomeadamente a classe abstrata entidade que foi criada por conveniência.

O diagrama representa por alto a maneira como o código será estruturado e quais as classes que irão ser utilizadas na sua realização. No entanto no desenrolar do desenvolvimento foi necessário a utilização de outras classes e inclusive eliminação de algumas.

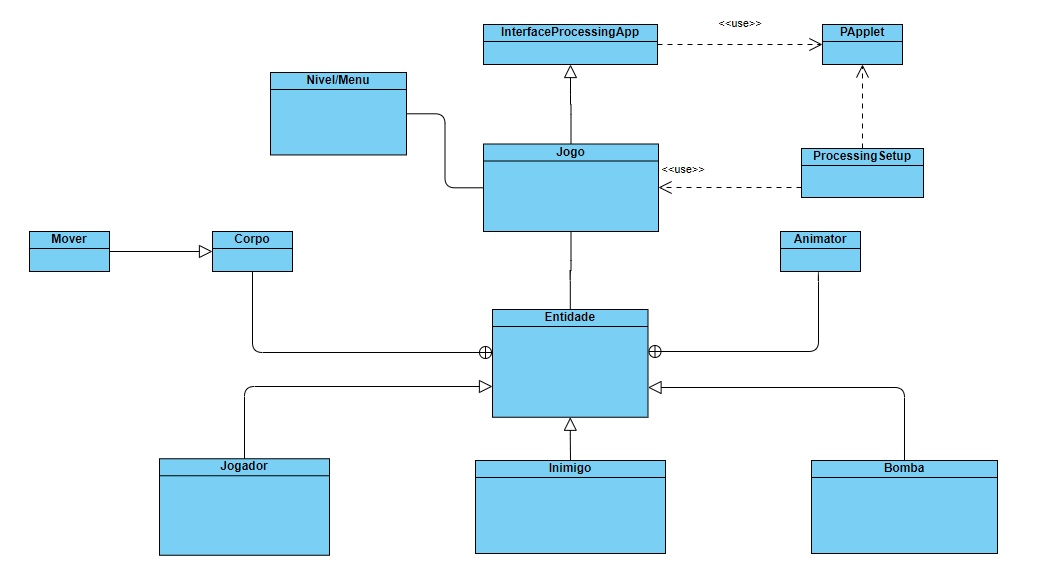


Figura 1 - Diagrama de classes inicial

A classe Processing Setup, tal como elaborada na aula será o ponto de partida de todo o programa. No entanto o Jogo será a parte fundamental deste.

### Classe Jogo

A classe Jogo foi concebida com o intuito de possuir toda a parte logica deste, e delegar às classes que implementem um nível (ou menu), e às entidades, o desenho na própria GUI dos seus elementos. Desta forma obtemos um código altamente modular, pouco propenso a erros e caso existam alguns erros, fáceis de corrigir e identificar.

Esta classe Jogo ficaria também responsável por toda a parte logica do mesmo, tais como instanciar inimigos, processar eventos do jogador, e as condições de vitoria/derrota.

A classe Jogo foi implementada de maneira sequencial tendo em conta as necessidades desta e utilizando sempre o melhor possível as suas classes derivadas. Esta classe possui um método draw onde os métodos draw das implementações de Nivel e Entidade são chamados, sempre nesta ordem de modo a sobrepor os agentes ao fundo, (Nivel).

Este método draw, foi nos muito conveniente pois é chamado pelo Processing, com o executar de cada frame, desta forma tínhamos um ponto no código que sabíamos sempre que seria executado pelo que a logica de jogo foi também implementada dentro deste.

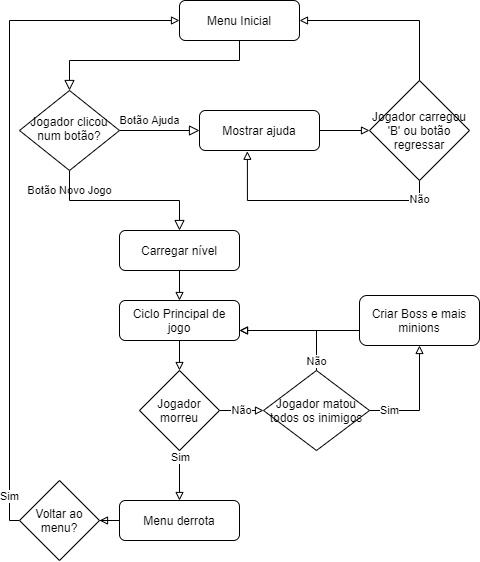
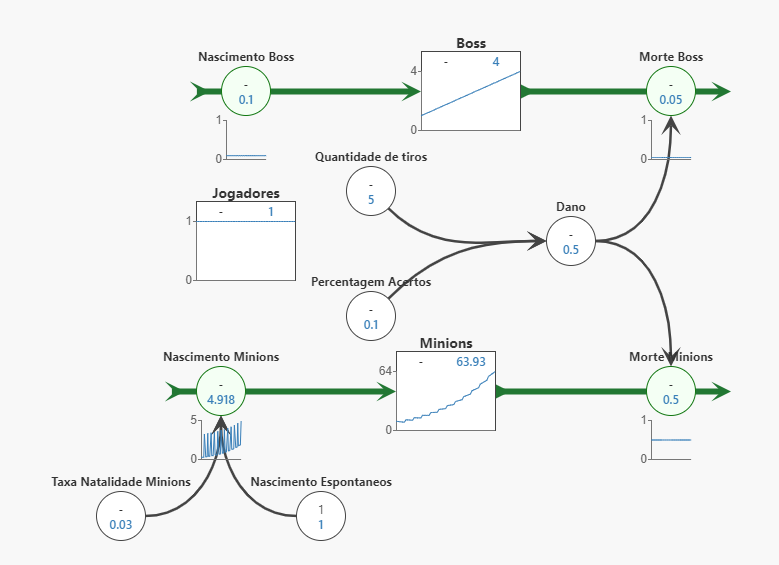


Figura 2 - Fluxo do jogo aplicação

O diagrama, presente na figura 2, representa o fluxo de atividades que um jogador experiência aquando da execução da aplicação. Este ciclo corresponde por traços grossos ao fluxo do programa tendo em conta os pressupostos mencionados em cima.

#### Reprodução dos Inimigos (Stocks e Flows)

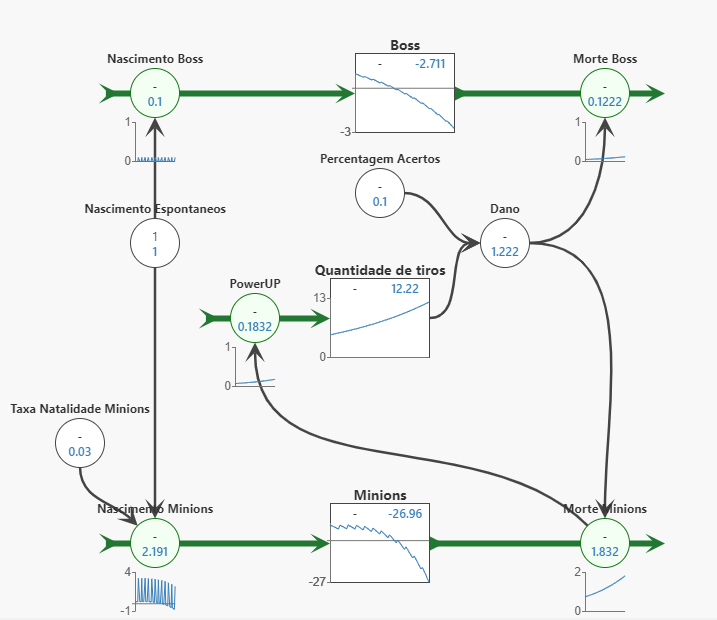
De um ponto de vista muito grosseiro, podemos dizer que os inimigos são presas e o jogador um predador. Assim sendo podemos modelar o comportamento do sistema através de um diagrama de stocks e flows. Diagrama este muito útil para tentar equilibrar a complexidade do jogo, mantendo o desafio, mas não permitindo uma explosão na população de inimigos.



Este modelo tem algumas variáveis fixas, nomeadamente os nascimentos espontâneos e a taxa de natalidade, ambos os valores incluídos no código, apesar de não explicitamente. Como o nosso projeto é um jogo, o predador não foi incluído no diagrama apesar de estar representado com a quantidade de tiros e a percentagem de acertos, em média que o jogador acerta.

Através deste diagrama é possível ver facilmente que se nada mudar, depressa o número de inimigos aumenta exponencialmente, chegando eventualmente a despoletar a condição de perda por “perda do planeta”.

Assim sendo, para equilibrar o jogo, decidimos que seria uma boa ideia implementar power ups para o jogador, dando um sentido de progresso e possibilitando um equilíbrio dos números numa fase mais tardia do jogo. Power ups este incluído no diagrama seguinte.



Desta forma somos capazes de controlar a quantidade de dano debitada pelo jogador e assim levar a uma vitoria mais rápida. De notar que a quantidade de tiros é também um stock, que é influenciado pela quantidade de mortes de *minions*.

Ou seja, cada vez que o jogador mata um *minion* existe uma pequena probabilidade de o *minion* largar um *power up* que aumente a quantidade de tiros, e assim aumenta a quantidade de dano dada por este que contraria a tendência de crescimento da população de minions.

Finalmente a variável percentagem de acertos é um valor determinado empiricamente entre os dois autores do projeto, e que resultou em gráficos que permitem interpretar o resultado pretendido, a percentagem de acertos do leitor, pode ser maior ou menor dependendo da sua habilidade.