



FACULDADE DE  
CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE DE  
**COIMBRA**

*Departamento de Engenharia Informática*

Fundamentos de Inteligência Artificial  
Introdução à Inteligência Artificial  
2023/2024 - 2º Semestre

Trabalho Prático Nº1:  
*Pac-Man:*  
*Ghost Unity*

**Nota:** A fraude denota uma grave falta de ética e constitui um comportamento inadmissível num estudante do ensino superior e futuro profissional licenciado. Qualquer tentativa de fraude levará à anulação da componente prática tanto do facilitador como do prevaricador, independentemente de ações disciplinares adicionais a que haja lugar nos termos da legislação em vigor. Caso haja recurso a material não original, as **fontes** devem estar explicitamente indicadas.

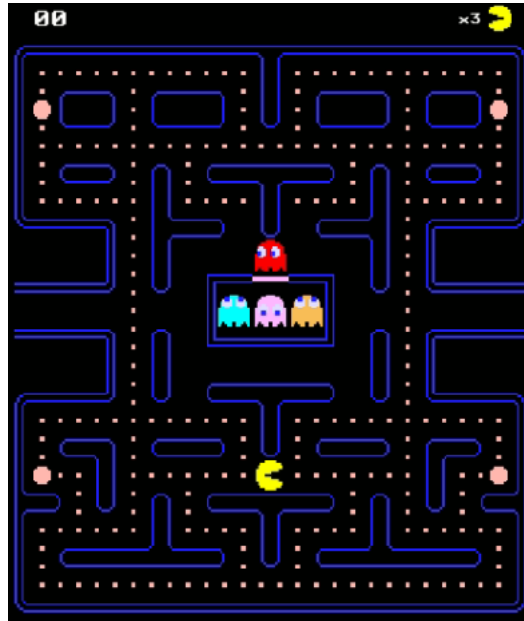


Figura 1: Pac-Man

## 1 Introdução

O Pac-Man é um popular jogo de arcade, cuja versão original foi lançada pela Namco no início da década de 1980. Neste jogo, o jogador controla o Pac-Man (um *character* redondo e amarelo) que se move num labirinto, onde existem bolas de comida (*Pellets*) e quatro fantasmas. O seu objetivo é comer toda a comida, sem ser apanhado pelos fantasmas. Por fim, existem quatro bolas de comida especiais (*PowerPellets*), que se denotam pelo seu tamanho maior, e que permitem ao Pac-Man comer os fantasmas durante um período de tempo reduzido. Neste projeto, será utilizada uma implementação simplificada do Pac-Man desenvolvida em Unity. O Unity é um motor de jogos desenvolvido pela Unity Technologies, especialmente notável pela sua portabilidade entre plataformas. Entre as suas especificações é de distinguir os motores de física e gráfico, a extensa biblioteca e respetiva documentação, assim como suporte para as linguagens C# e JavaScript. Apesar da sua finalidade ser o suporte ao desenvolvimento de jogos, estas características permitem também que o Unity seja utilizado como ferramenta de simulação, com aplicabilidade na Inteligência e Vida Artificial, capaz de funcionar como ambiente para simulações realistas.

## 2 Objectivos Genéricos

O presente trabalho tem como objetivos genéricos:

1. Aprender a desenhar um agente reativo adequado a um problema e ambiente específico.
2. Aprender a especificar e formalizar os seguintes aspetos de um agente reativo:
  - (a) Perceções
  - (b) Ações
  - (c) Sistema de produções
  - (d) Memória
3. Aprender a fazer uma descrição de alto-nível do comportamento desejado implementável através de um agente reativo.
4. Aprender a criar um agente reativo a partir de uma descrição de alto nível.

## 3 Pac-Man

Este trabalho prático tem como objetivo principal a aquisição de competências relacionadas com a análise, desenvolvimento, implementação e teste de sistemas de produções para o controlo de agentes autónomos. Para tal, usar-se-á o jogo Pac-Man como plataforma de aprendizagem. Importa assim descrever quais os componentes fundamentais deste jogo, nomeadamente o ambiente, os fantasmas e o Pac-Man.

### 3.1 Ambiente

O ambiente deste jogo (Figura 2) consiste num mundo em grelha com uma dimensão de 28 x 36 células. Cada célula pode conter um *Pellet* (comida), obstáculo, ou ser uma passagem ou entroncamento de passagens. Tal como já foi referido, os *Pellets* podem ser normais, contribuindo apenas para o aumento da pontuação do Pac-Man, ou serem *PowerPellets* que lhe permite comer fantasmas. Existe um total de 240 *Pellets* e 4 *PowerPellets* no ambiente, que vão sendo removidos à medida que são comidos. É importante referir que apesar do ambiente ser baseado numa grelha, os movimentos do Pac-Man e fantasmas são contínuos, sendo que em cada momento movem-se

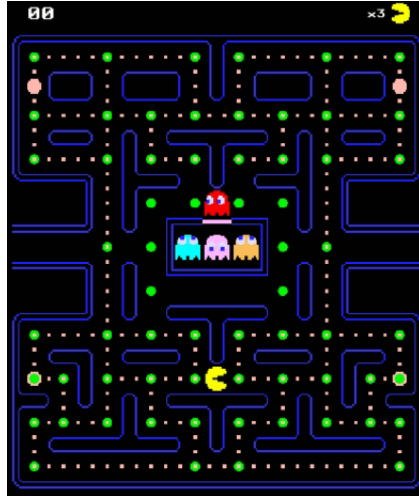






Figura 2: Labirinto com locais de tomada de decisão visíveis (círculos verdes).

na direção em que estão voltados com velocidade constante. Enquanto que o Pac-Man é controlado pelo jogador (podendo mudar de direção a qualquer instante) os fantasmas apenas têm a possibilidade de decidir uma nova direção de movimento quando se encontra em células específicas (entroncamento, cruzamento, ou esquina) denotadas pelos círculos verdes da Figura 2. Na implementação fornecida, estes círculos têm triggers associados, fazendo executar a função de tomada de decisão do fantasma correspondente.

### 3.2 Fantasmas

Na versão original, cada fantasma caracteriza-se por ter uma alcunha, bem como cor e comportamento distintos:

- Inky 
- Blinky 
- Pinky 
- Clyde 

Apesar de distintos, os comportamentos dos fantasmas podem ser divididos em três fases (*home*, *chase* e *frightened*), sendo que as discrepâncias surgem apenas no modo *chase*. Tal como referido na Secção 3.1, os fantasmas decidem qual a direção de movimento sempre que colidem com um dos círculos verdes, sendo a decisão dependente do fantasma em causa, bem como o seu modo de comportamento ativo de momento.

O modo *home* está ativo enquanto os fantasmas estão dentro da sua base e consiste apenas em movimentos verticais. Os fantasmas podem estar dentro

da sua base em dois momentos:

- No início de cada ronda, todos os fantasmas à exceção do Blinky começam dentro da base (Figura 1), sendo soltos periodicamente, de modo a dar algum espaço de manobra ao Pac-Man.
- Após um fantasma ser comido, renasce na base, adotando este comportamento.

Assim que os fantasmas saem da sua base, ativam o modo *chase*, iniciando a perseguição ao Pac-Man com comportamentos distintos:

- Inky - Este fantasma exhibe o comportamento mais errático de todos, seleccionando uma direcção aleatória sempre que tem de tomar uma decisão.
- Blinky - Procura mover-se para a posição actual do Pac-Man escolhendo, em cada intersecção, a direcção da célula adjacente que minimiza a distância ao Pac-Man.
- Pinky - A personalidade deste fantasma prima pelas tentativas de emboscadas, tentando mover-se de forma a intersectar o movimento do Pac-Man. Para tal, o Pinky tenta mover-se para a célula quatro unidades à frente do Pac-Man.
- Clyde - Este é o fantasma mais paradoxal de todos, tendo uma fobia de outros fantasmas. Desta forma, sempre que está no modo *chase*, este fantasma tenta mover-se de forma a maximizar a distância ao fantasma mais próximo de si.

Importa ainda salientar que, durante a execução do modo *chase* os fantasmas não podem inverter o sentido de marcha, i.e., numa intersecção, não podem seleccionar a direcção oposta à que têm actualmente.

Por fim, o modo *frightened* é activado assim que o Pac-Man come uma das *PowerPellets*. Neste modo, os fantasmas alternam de cor entre branco e azul e movem-se de forma a maximizar a sua distância ao Pac-Man. Para tal, sempre que chegam a uma intersecção, os fantasmas calculam a distância ao Pac-Man em cada uma das células adjacentes, seguindo na direcção da que maximiza essa distância. Caso um fantasma seja comido pelo Pac-Man, regressa à sua base onde irá permanecer por 8 segundos, sendo representado apenas pelos seus olhos.

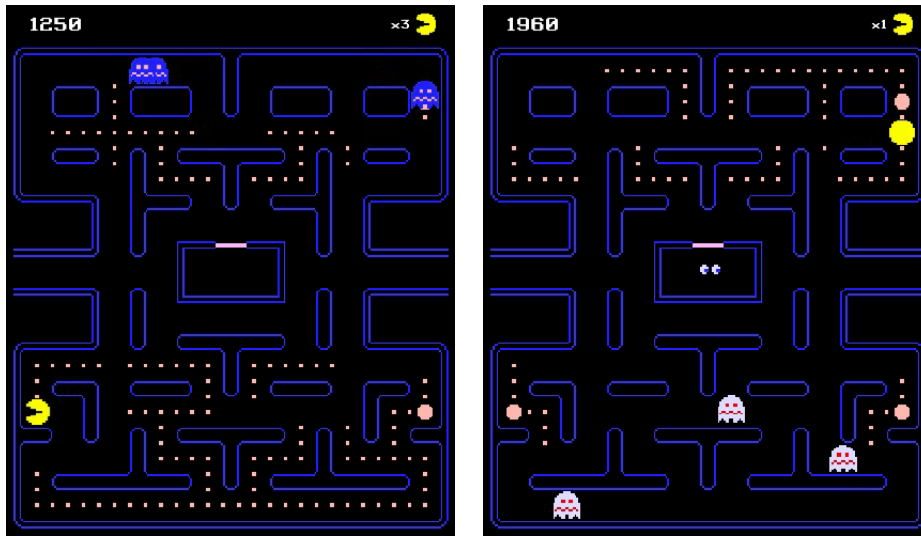


Figura 3: Fantasmas em modo *Frightened*.

### 3.3 Pac-Man

No que toca ao Pac-Man, os seus movimentos são controlados pelo jogador, através das teclas WASD ou das setas. O nível é ganho quando o Pac-Man come todos os *Pellets*, sendo que a pontuação é incrementada sempre que se come *Pellets*, *PowerPellets* ou fantasmas. Deste modo, o objetivo secundário do jogo é terminar com a maior pontuação possível.

Tal como já foi referido, o nível termina quando o Pac-Man come todos os *Pellets*, começando um novo nível com a pontuação e vidas atuais. Por outro lado, o jogo termina quando o Pac-Man é comido pela terceira vez pelos fantasmas, gastando todas as suas vidas. Neste caso, pressionando uma tecla aleatória, será iniciado um novo jogo.

## 4 Tarefas

O objetivo deste trabalho consiste no **desenvolvimento do sistema de produções que modela o comportamento *chase*** de cada um dos fantasmas. Deverá começar por descarregar o projeto do *UCStudent* e abri-lo no Unity. Para tal, deverá **criar um novo projeto 2D** (Figura 4) e aceder à opção de **importar uma package custom** (Figura 5 - esquerda). De seguida será aberta uma janela onde deve escolher a package que descarregou do *UCStudent* e, uma vez selecionada, será aberta uma nova janela que lhe permite escolher quais os componentes a importar (Figura 5 - direita). **Deverá certificar-se que todos estão selecionados.** Por fim, deverá aceder



Figura 4: Criação de um projeto 2D.

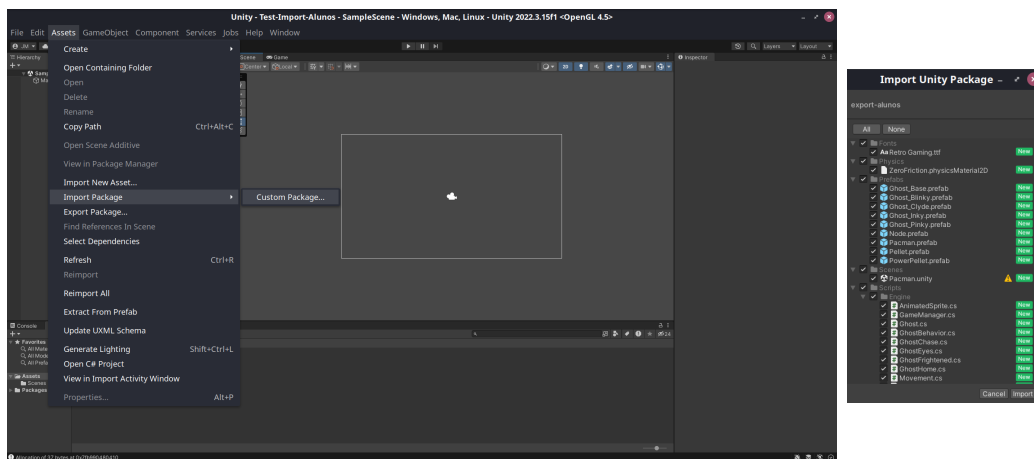


Figura 5: Importação do projeto.

à pasta *Scenes* e abrir a intitulada *Pacman*. Neste momento deverá ter o projeto corretamente importado (Figura 6) e pode apagar a *Sample Scene*.

O projeto apresenta já uma implementação funcional do jogo, seguindo uma abordagem orientada a objetos. Na pasta Scripts/Engine existem vários scripts que asseguram as funcionalidades básicas do jogo e que **não deverão ser alterados**.

Os comportamentos de cada fantasma estão divididos em três scripts distintos (*GhostHome.cs*, *GhostChase.cs* e *GhostFrightened.cs*), sendo que cada um deles estende o script *GhostBehaviour.cs*. Dado que o objetivo deste trabalho se foca no desenvolvimento dos comportamentos do modo *chase*, foi já criado um **script específico para cada fantasma, intitulados com o nome do fantasma seguido do modo de comportamento (por exemplo, BlinkyChase.cs)**. Estes scripts herdam as funcionalidades de *GhostChase.cs*, redefinindo apenas a função ***OnTriggerEnter2D***, sobre a qual incidirá a sua implementação. De modo a facilitar o seu trabalho, a **implementação do comportamento do Inky é já fornecida**, sendo também apresentada na Listagem 1. Por outro lado, os restantes fantasmas

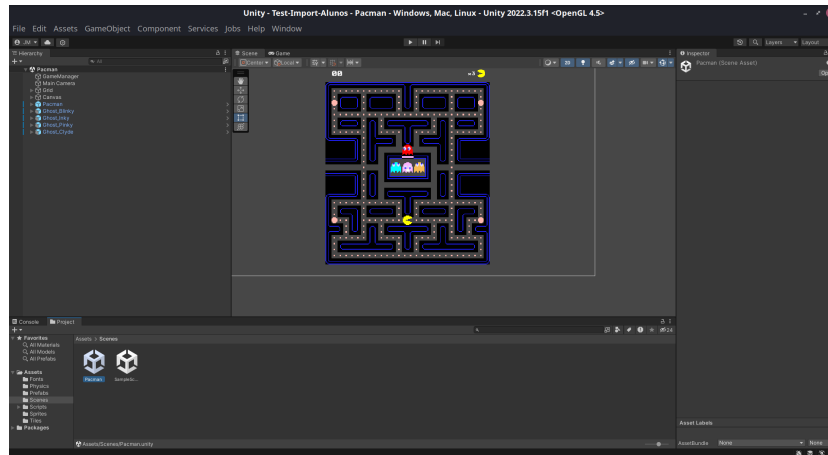


Figura 6: Visão do Unity com o projeto importado.

adotam um comportamento base, implementado no *GhostChase.cs* que consiste em escolher a primeira direção disponível que não inverte o sentido de marcha, segundo a ordem Norte, Sul, Este e, por fim, Oeste.

#### Listagem 1: Excerto do script InkyChase.cs

```

1 protected override void OnTriggerEnter2D(Collider2D other){
2     //Inky's chase behavior
3     //We only have to select the next direction to move to
4
5     //instantiating the intersection node object
6     Node node = other.GetComponent<Node>();
7
8     //First check if this behaviour is enabled
9     //and the ghost is not frightened
10    if (node != null && isChasing() && !isFrightened()){
11        //Get the available directions in this intersection
12        List<Vector2> dirs = getAvailableDirections(node);
13        int count = dirs.Count;
14
15        //Choose a random direction and avoid going back
16        //the same direction it came from
17        int i = Random.Range(0, count);
18        if (count > 1 && dirs[i] == -currentDirection()){
19            i = (i + 1) % count;
20        }
21        setDirection(dirs[i]);
22    }
23 }
```

Tal como referido acima, este script redefine a função *OnTriggerEnter2D* que é responsável pelo processo de tomada de decisão de cada comporta-



mento. Esta função é ativada sempre que o fantasma entra numa intersecção, colidindo com um nó invisível (círculos verdes apresentados na Figura 2). Nesse momento é gerado um evento que ativa esta função. No entanto, uma vez que é possível que existam outros elementos que ajam como trigger, é necessário começar por verificar se o evento foi gerado por um nó. É ainda verificado se o modo *chase* está ativo e se o fantasma não está no modo *frightened*. Caso todas as condições se verifiquem, faz-se então o processo de tomada de decisão que, no caso do Inky, consiste em escolher a nova direção de movimento do fantasma de forma aleatória.

Para resolver este projeto, deverá focar-se nos scripts BlinkyChase.cs, PinkyChase.cs e ClydeChase.cs. Mais propriamente, **deverá desenvolver as funções *OnTriggerEnter2D* destes scripts, substituindo os placeholders pela implementação dos seus sistemas de produções de forma a emular o comportamento descrito para cada fantasma.**

No desenvolvimento do seu sistema de produções será necessário perceber o ambiente. Para tal, foram definidos vários sensores que poderá utilizar e que são apresentados na Listagem 2. Note que estes sensores foram definidos na classe GhostBehaviour, de onde todos os scripts de comportamentos herdam as suas propriedades. Deste modo, poderá invocar estes sensores diretamente no seu script (tal como exemplificado na linha 10 da Listagem 1). Os sensores providenciados podem ser agrupados em cinco grupos, sendo que o primeiro grupo permite verificar se um determinado comportamento está ativo (linhas 1 a 3); o segundo grupo permite obter a posição e direção de movimento do Pac-Man, bem como calcular a distância a que ele se encontra deste fantasma (linhas 5 a 7); o terceiro grupo (linhas 9 a 14) permite obter informações relativas ao estado do jogo, tais como o nível actual, o número de vidas restantes, a quantidade e posição das *Pellets* ativas no ambiente e a distância à *Pellet* mais próxima, bem como a sua posição; o quarto grupo (linhas 16 a 18) permite calcular a distância ao fantasma mais próximo, bem como obter a sua posição e direção de movimento; o último grupo (linhas 20 a 26) permite obter a lista de direções válidas na intersecção atual (i.e., direções não obstruídas), verificar se uma determinada direção de movimento é válida e obter a posição e direção de movimento atual. Por fim, a linha 28 permite fazer uma ação, que consiste em definir a próxima direção de movimento (na forma de um Vector2).

Listagem 2: Percepções e ações dos fantasmas

```
1 protected bool atHome();
2 protected bool isChasing();
3 protected bool isFrightened();
4
5 protected Vector3 getPacmanPosition();
6 protected Vector2 getPacmanDirection();
7 protected float getPacmanDistance();
8
9 protected int getCurrentLevel();
10 protected int getRemainingLives();
11 protected int countRemainingPellets();
12 protected Vector3[] getRemainingPellets();
13 protected Vector3 getClosestPelletPosition();
14 protected float getClosestPelletDistance();
15
16 protected Vector3 getClosestGhostPosition();
17 protected Vector2 getClosestGhostDirection();
18 protected float getClosestGhostDistance();
19
20 protected List<Vector2> getAvailableDirections(Node node);
21 protected bool canGoUp(Node node);
22 protected bool canGoDown(Node node);
23 protected bool canGoLeft(Node node);
24 protected bool canGoRight(Node node);
25 protected Vector2 currentDirection();
26 protected Vector3 currentPosition();
27
28 protected void setDirection(Vector2 direction);
```

## 5 Metas

O presente trabalho prático encontra-se dividido em 2 metas distintas:

### Meta 1 – Modelação e desenvolvimento do Sistema de Produções

Nesta meta, deverá modelar os comportamentos descritos para cada fantasma (**incluindo o Inky**) descrevendo as **percepções**, **ações** e **sistema de produções**. Deve ainda implementar pelo menos um dos fantasmas (Blinky, Pinky ou Clyde).

### Meta 2 – Implementação do Sistema de Produções em Unity

Deve implementar o comportamento de todos os fantasmas. Como desafio suplementar, modele e implemente comportamentos que considere interes-

santes utilizando as percepções disponibilizadas e, caso necessite, memória.

## 6 Datas e Modo de Entrega

Os grupos têm uma dimensão máxima de 3 alunos. A defesa é obrigatória, bem como a presença de todos os elementos do grupo na mesma.

A entrega da meta 1 é opcional, chama-se no entanto a atenção dos alunos para a importância de concluir atempadamente esta meta. Para efeitos de nota apenas será considerada a entrega final e a defesa.

### 6.1 Meta 1 – Modelação e desenvolvimento do Sistema de Produções

**Material a entregar:**

- Package de Unity com o projeto, incluindo os scripts onde implementaram e/ou alteraram código, que deve estar devidamente comentado.
- Um breve documento (max. 3 páginas), em formato pdf, com a seguinte informação:
  - Identificação dos elementos do grupo (Nomes, Números de Estudante, e-mails, Turma(s) Prática(s)).
  - Modelação dos comportamentos de *chase* dos **quatro fantasmas** através de um sistema de produções.
  - Outra informação que considere pertinente relativamente a esta meta.

**Modo de Entrega:**

Entrega eletrónica através do Inforestudante.

**Data Limite: 3 de Março de 2024**

### 6.2 Meta 2 – Implementação do Sistema de Produções em Unity

Tal como indicado anteriormente, esta entrega será a única que tem um impacto direto na nota. O relatório deve conter informação relativa a **todo** o trabalho realizado. Ou seja, o trabalho realizado no âmbito das metas 1 e 2 deve ser **inteiramente descrito**, por forma a possibilitar a avaliação.

### **Material a entregar:**

- Package de Unity com o projeto, incluindo os scripts onde implementaram e/ou alteraram código, que deve estar devidamente comentado.
- Um relatório (max. 10 páginas), em formato pdf, com a seguinte informação:
  - Identificação dos elementos do grupo (Nomes, Números de Estudante, e-mails, Turma(s) Prática(s)).
  - Informação pertinente relativamente à globalidade do trabalho realizado. Incluindo os sistemas de produções, perceções e ações dos fantasmas.

Num trabalho desta natureza o relatório assume um papel importante. Deve ter o cuidado de descrever detalhadamente todas as funcionalidades implementadas, dando particular destaque aos problemas e soluções encontradas. Deve ser fácil ao leitor compreender o que foi feito e ter por isso capacidade de adaptar / modificar o código.

O relatório deve conter informação relevante tanto da perspetiva do utilizador como do programador. Não deve ultrapassar as 10 páginas, formato A4. Todas as opções tomadas deverão ser devidamente justificadas e explicadas.

### **Modo de Entrega:**

Entrega eletrónica através do Inforestudante.

**Data Limite: 17 de Março de 2024**

## **7 Bibliografia**

1. **Inteligência Artificial: Fundamentos e Aplicações**  
*Ernesto Costa, Anabela Simões*
2. **Artificial Intelligence: A Modern Approach**  
*Stuart Russel, Peter Norvig*