



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE
COIMBRA

Departamento de Engenharia Informática

Introdução à Programação e Resolução de Problemas
2025/2026 - 1^o Semestre

Mini-Projeto N^o2: *Invasores do Espaço*

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. Caso use ferramentas de IA na produção deste trabalho (e.g. ChatGPT), deverá identificar de forma clara todas as partes em que a ferramenta esteve envolvida. Note que, durante a defesa, deverá demonstrar ter conhecimento profundo dos conteúdos gerados pela ferramenta, sendo esse conhecimento objeto de avaliação.

João Correia, Carlos Fonseca, Maria José Marcelino, João Macedo, Tiago Baptista, Telmo Neves, Noé Godinho – 2025/2026

1 Introdução

Lançado em 1978 pela empresa japonesa *Taito* e criado por **Tomohiro Nishikado**, o *Space Invaders* é amplamente reconhecido como um dos marcos fundadores da história dos videogames. O título popularizou o género dos jogos de disparo e estabeleceu muitas das convenções que definiriam a indústria nas décadas seguintes: ondas progressivas de inimigos, pontuação acumulativa e uma sensação de tensão crescente à medida que o jogador defende a Terra de uma invasão alienígena. O seu sucesso massivo — cultural e comercial — marcou o início da “era dourada” dos *arcades* e transformou o videogame num fenómeno global, servindo até hoje como referência pedagógica e inspiração para recriações e projetos educativos de programação interativa. Pretende-se que neste trabalho implemente uma versão deste jogo semelhante ao apresentado na Figura 1.



Figura 1: Space Invaders IPRP edition!

2 Objetivos Genéricos

O presente projeto tem como principal objetivo consolidar, através da implementação prática de um jogo interativo, os conhecimentos adquiridos ao longo da unidade curricular de Introdução à Programação e Resolução de Problemas. Pretende-se que os estudantes desenvolvam competências de análise, decomposição e estruturação de problemas, aplicando os conceitos fundamentais da programação em **Python**, nomeadamente:

- Interpretação e adaptação de base de código fornecida.
- Definição e utilização de funções para modularizar o código e promover a reutilização;
- Manipulação de estruturas de dados para representar as diversas componentes do jogo;
- Desenho e manipulação da parte gráfica do jogo através de *turtle graphics*.
- Leitura e escrita de ficheiros de texto para guardar e restaurar informação persistente (pontuações e estado do jogo);
- Controlo de eventos de teclado e atualização contínua de um ciclo principal (*game loop*);
- Organização do programa em componentes lógicos independentes, promovendo a clareza, a legibilidade e a manutenibilidade do código.

Para além dos objetivos técnicos, o projeto visa ainda fomentar a criatividade, a atenção ao detalhe e o rigor na implementação, incentivando os estudantes a compreender o funcionamento interno de um jogo clássico e a aplicar boas práticas de programação estruturada.

3 Enunciado e Objetivos

Como referido na introdução, o **Space Invaders** é um jogo de ação em que o jogador controla uma nave espacial posicionada na parte inferior do ecrã e deve defender a Terra de uma frota de invasores alienígenas que descem lentamente em formação. O objetivo é destruir todos os inimigos antes que estes alcancem o fundo do ecrã ou colidam com a nave do jogador. O jogador move-se horizontalmente (esquerda e direita) e dispara projéteis verticais para eliminar os inimigos, ganhando pontos por cada acerto. Os inimigos movem-se gradualmente para baixo e por vezes também horizontalmente, podendo também disparar projéteis. O jogo termina se o jogador for atingido, se um inimigo chegar ao fundo do ecrã ou em caso de colisão direta entre um inimigo e a nave.

Objetivo geral: completar a implementação do jogo de forma a:

- Apresentar a nave do jogador, um exército de inimigos em **grelha configurável** (linhas/colunas/espacamento) alinhado **no topo visível do ecrã**, e projéteis do jogador e dos inimigos;
- Permitir movimento da nave (*Left/Right*) e disparo (*Space*);
- Atualizar o estado do jogo (**STATE**) em ciclo contínuo: mover balas, mover inimigos (queda + *drift*), disparo aleatório dos inimigos, e deteção de colisões;
- Implementar guardar e carregar estado do jogo em ficheiro texto, incluindo: pontuação, posição do jogador, posições dos inimigos, projecteis do jogador e balas dos inimigos;
- Manter um conjunto de pontuações máximas (e.g. top-10) em ficheiro;
- A tecla **g** guarda o estado; a tecla **Esc** termina o jogo e atualiza os highscores.

4 Estrutura e Ficheiros Fornecidos

Foram disponibilizados os seguintes ficheiros para desenvolvimento do projeto:

- `space_invaders_alunos.py` — ficheiro base com `STATE`, globais, ciclo principal e assinaturas das funções (`def ...`).
- `player.gif`, `enemy.gif` — imagens para a nave do jogador (*player*) e inimigos.

5 Funções a Implementar

As funções listadas abaixo estão declaradas no ficheiro base do projeto e deverão ser completadas pelos estudantes. Por cada subsecção temos um grupo de funções que corresponde a uma componente funcional do jogo *Space Invaders*. A ordem de apresentação destes grupos segue a mesma ordem pela qual as funções aparecem no código. Contudo, **não se recomenda a implementação do projeto pela ordem apresentada**, devendo ser analisado a zona *main* do código fornecido para tomar a decisão quanto à ordem de implementação.

O normal funcionamento do jogo fará uma série de actualizações de estado do jogo através de verificações e alterações à variável de estado (i.e. *state*):

Listagem 1: Estrutura base do dicionário `state`

```
1 # Estado base
2 state = {
3     "screen": screen,
4     "player": None,
5     "enemies": [],
6     "enemy_moves": [],
7     "player_bullets": [],
8     "enemy_bullets": [],
9     "score": 0,
10    "frame": 0,
11    "files": {"highscores": HIGHSCORES_FILE, "save":
              SAVE_FILE}
12 }
```

5.1 Top Resultados (Highscores)

Funções que devem gerir os melhores resultados alcançados. Deverá existir um ficheiro que contém o registo das melhores pontuações por jogador que deve ser lido na altura de actualizar os resultados. O actualizar deverá verificar se a pontuação atingida é suficiente para fazer parte do topo. Se sim, deverá pedir ao utilizador o nome para se poder registar o mesmo e o seu respectivo score alcançado. Caso o score registado no jogo não seja suficiente para o top definido na variável *TOP_N* não deverá ser pedido ao utilizador o nome nem deverá ser registado no ficheiro de *highscores*.

1. `ler_highscores(filename)`

Lê o ficheiro de pontuações e devolve uma lista com os melhores resultados (top-N), ordenada de forma decrescente pela pontuação.

2. `atualizar_highscores(filename, score)`

Atualiza o ficheiro de pontuações, pedindo o nome do jogador se a nova pontuação pertencer ao top-N.

5.2 Guardar / Carregar Estado (texto)

A informação sobre o estado do jogo deve ser exportada para ficheiro. Deve ser possível carregar o estado do jogo em pleno aquando o carregamento do ficheiro gravado. Deve ter atenção à forma como exporta os dados ser possível carregar na totalidade o estado do jogo: posição do jogador, score, posição dos inimigos, direcção dos mesmos, balas do jogador e balas dos inimigos. Poderá incluir informação adicional nos ficheiros se achar pertinente.

1. `guardar_estado_txt(filename, state)`

Guarda o estado completo do jogo num ficheiro de texto, incluindo: pontuação, posição do jogador, posições dos inimigos, direcções de movimento (`enemy_moves`) e balas (do jogador e dos inimigos).

2. `carregar_estado_txt(filename)`

Lê o ficheiro de texto e devolve um dicionário com o estado do jogo ou `None` caso o ficheiro não exista.

5.3 Criação de Entidades (Jogador, Inimigos e Balas)

As funções que se seguem lidam com a parte gráfica do jogo, gerem a criação do jogador, da grelha de inimigos e da gestão das balas que são criadas quer pelo utilizado quer pelos inimigos.

1. `criar_entidade(x, y, tipo="enemy")`
Cria e devolve uma entidade do tipo indicado ("player" ou "enemy") com a imagem correspondente (player.gif ou enemy.gif) posicionada nas coordenadas dadas.
2. `criar_bala(x, y, tipo)`
Cria e devolve uma bala ("player" ou "enemy").
3. `spawn_inimigos_em_grelha(state, posicoes_existentes, dirs_existentes=None)`
Gera os inimigos em grelha no topo do ecrã ou repõe posições e direções a partir de um ficheiro guardado. O número de linhas, colunas e espaçamento entre inimigos é configurável através de variáveis definidas no topo do ficheiro providenciado (e.g. *ENEMY_ROWS*, *ENEMY_COLS*).
4. `restaurar_balas(state, lista_pos, tipo)`
Restaura as balas guardadas, recriando-as e adicionando-as às listas do estado (player_bullets ou enemy_bullets).

5.4 Handlers para Teclado

O jogo deve contar com a implementação dos seguintes controlos por teclado:

- **Setas direcionais *Left* e *Right*:** mover a nave do jogador horizontalmente para a esquerda e para a direita, respectivamente;
- **Space:** disparar;
- **g:** guardar estado para ficheiro;
- **Esc:** terminar jogo e atualizar highscores.

Desta forma espera-se que se implementem as seguintes funções:

1. `mover_esquerda_handler()`
Move a nave do jogador para a esquerda, respeitando os limites do ecrã.
2. `mover_direita_handler()`
Move a nave do jogador para a direita, respeitando os limites do ecrã.
3. `disparar_handler()`
Cria uma nova bala do jogador e adiciona-a à lista de balas ativas.

4. `gravar_handler()`

Invoca a função de gravação do estado atual do jogo (`guardar_estado_txt`).

5. `terminar_handler()`

Termina o jogo, apresenta a pontuação final e atualiza o ficheiro de `highscores`.

5.5 Atualizações e Colisões

A lógica do jogo, as regras e condições de término serão geridas por estas funções. A cada ciclo de jogo as posições dos inimigos são actualizadas incluindo as funções que gerem as chances de disparo e movimento horizontal dos inimigos. Um aspecto importante para o funcionamento do jogo é a deteção de colisões entre as entidades do jogo, player, inimigos e balas. Considera-se colisão se duas figuras se interceptarem mediando um raio de colisão definido na variável `COLLISION_RADIUS`.

1. `atualizar_balas_player(state)`

Move as balas do jogador para cima e remove as que saem do ecrã.

2. `atualizar_balas_inimigos(state)`

Move as balas dos inimigos para baixo e remove as que ultrapassam o limite vertical inferior.

3. `atualizar_inimigos(state)`

Move os inimigos verticalmente para baixo, aplicando um *drift* horizontal aleatório e individual para cada inimigo. Deverão gerir aqui o movimento vertical e horizontal. O vertical deve ser actualizado a cada chamada desta função de acordo com a variável `ENEMY_FALL_SPEED`. Os movimentos horizontais ocorrem de acordo com `ENEMY_DRIFT_CHANCE` em que deverá ser feito movimento horizontal sobre uma direção. Deve existir também a possibilidade de inverter essa direção gerida pela probabilidade de `ENEMY_INVERT_CHANCE`. As diferentes direções em *x* resultantes dos mecanismos de movimentos aleatórios horizontais (`enemy_moves`) devem ser registadas a cada chamada da função e para cada inimigo, bastando o registo de -1 ou 1 dependendo se está invertida ou não. Este dados fazem parte do estado do jogo e portanto fazem parte da informação a ser guardada e gerida pela variável `state`.

4. `inimigos_disparam(state)`

Cada inimigo tem uma probabilidade definida pela variável `ENEMY_FIRE_PROB`

de disparar uma bala que deve partir do inimigo que dispara e seguir verticalmente até chegar ao limite vertical inferior do jogo.

5. `verificar_colisoes_player_bullets(state)`

Verifica e trata colisões entre as balas do jogador e os inimigos, removendo ambos e incrementando a pontuação caso exista colisão.

6. `verificar_colisoes_enemy_bullets(state)`

Verifica colisões entre balas inimigas e o jogador. Se o jogador for atingido o jogo deverá terminar de forma limpa chamando a função *terminar_handler*.

7. `inimigo_chegou_ao_fundo(state)`

Deve verificar se algum inimigo ultrapassar a linha inferior de segurança.

8. `verificar_colisao_player_com_inimigos(state)`

Deve verificar se o jogador colidiu diretamente com um inimigo.

5.6 Ir mais além

Os pressupostos base do jogo ficarão assegurados com a execução correta das funções descritas anteriormente. Após concluírem o projeto, os estudantes são desafiados a ir mais além e a explorar melhorias ou novas funcionalidades que expandam o comportamento do jogo. Por exemplo, poderão implementar mecanismos de seleção de dificuldade, configurar a grelha de inimigos, introduzir novos comportamentos, efeitos visuais ou artefactos de jogo. Tais implementações **serão consideradas como “extras” e, dependendo da sua profundidade, poderão compensar parcialmente aspetos do projeto base que não estejam totalmente corretos**. Caso optem por incluir extras, **é obrigatório submeter duas versões do ficheiro principal**: uma versão base, correspondente ao projeto solicitado, e uma versão separada contendo as extensões adicionais. Deverão também evidenciar no código que blocos extra foram adicionados para fácil discussão na defesa.

6 Entrega, Defesa e Datas

O trabalho deverá ser realizado parcialmente durante as aulas TP e por grupos com uma dimensão máxima de 2 alunos pertencentes à mesma turma.

O trabalho está sujeito a uma defesa obrigatória, em que todos os elementos do grupo têm de estar presentes. A não realização da defesa resultará numa classificação de 0 valores no trabalho.

A entrega deve ser feita de forma eletrónica através do Inforestudante e deve conter um ficheiro .zip com todo o conteúdo que faça parte do projeto.

Data Limite: 23:59 de 5 de Dezembro de 2025