# Sistemas Operativos 2

2017/18

DEIS – Engenharia Informática

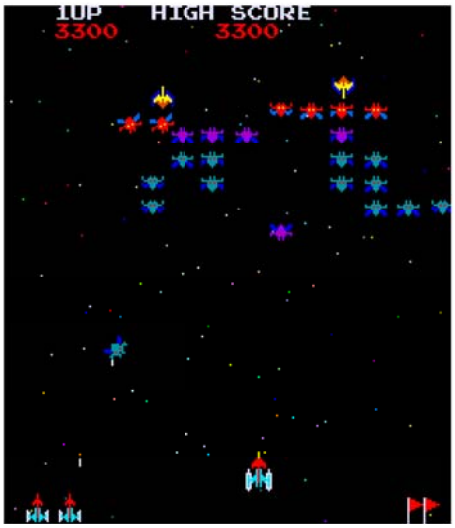
## Trabalho prático

As regras que regem o trabalho prático foram divulgadas nas primeiras aulas e encontram-se descritas na ficha de unidade curricular (*moodle*), chamando-se a atenção para a sua leitura.

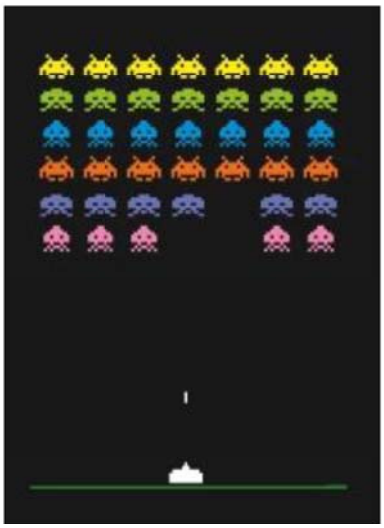
### 1. Descrição geral do tema - *Phoenix multiplayer*

Pretende-se que implemente na plataforma Win32 um jogo de habilidade de condução naves espaciais e destruição de frotas inimigas, semelhante aos clássicos Phoenix, Galaxians, Invaders, etc., popularizados em máquinas de arcada. Abaixo apresentam-se imagens desses jogos, mas apenas para referência – o aspecto gráfico pretendido não será tão elaborado. Apesar do aspecto gráfico dos exemplos, um jogo deste tipo é extremamente simples e resume-se a movimentar rectângulos no ecrã, sendo os aspectos relacionados com os mecanismos de sistemas operativos a parte mais difícil.


Phoenix




Galaxians



Invaders

O jogo poderá ser jogado com apenas um jogador ou com vários (modo cooperativo), mas cada jogador conduz apenas uma nave. É importante identificar já as entidades para simplificar o texto daqui para a frente: existem as naves *defensoras* (dos jogadores), as naves *invasoras*, as *bombas* (atirados pelos invasores), os *tiros* (atirados pelos defensores), e os *powerups* (objetos que vão caindo e que podem ser apanhados pelos defensores despoletando um determinado efeito).

O jogo será controlado por um programa central e a interface gráfica será feita por um outro programa independente.

**Modelo de dados envolvido**

Todas as entidades existem num plano (campo do jogo). As naves (invasoras e defensoras) são conceptualmente rectângulos com uma certa dimensão e que estão numa dada posição (coordenada no plano). Os tiros e bombas são meros pontos (sem dimensão – ou dimensão "1") e são descritos apenas pelas suas coordenadas (posição). Os *powerups* são também retângulos mas mais pequenos que as naves.

A mecânica relativa ao jogo em si limita-se ao seguinte:

Dados:

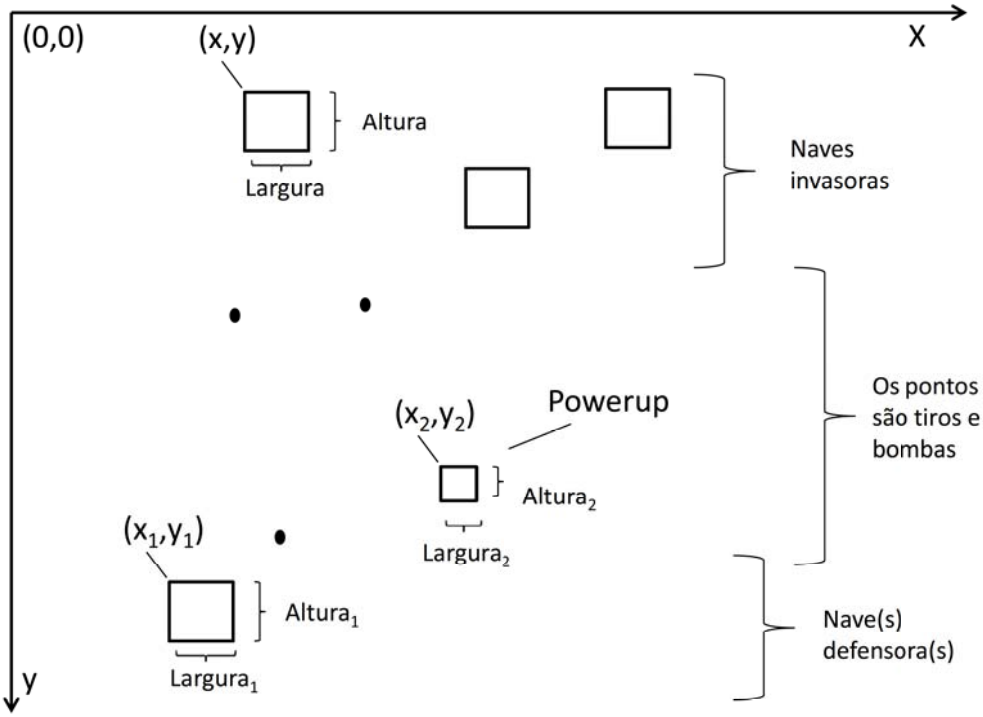
- Estruturas de dados capazes de representar vários rectângulos (naves defensoras, invasoras e *powerups*) para que seja possível saber qual o espaço ocupado por cada rectângulo.
- Estruturas de dados capazes de armazenar as coordenadas dos vários tiros e bombas (pontos).

"Algoritmos":

- Para verificar se um ponto se encontra dentro de um retângulo, com o objetivo de saber se um tiro ou bomba atingiu uma nave.
- Para ver se dois rectângulos se sobrepõem (se um deles tem pelo menos um canto dentro do outro – ver "algoritmo" anterior) para saber se duas naves estão a tentar ocupar a mesma posição. O significado disso depende do caso em questão (nave invasora com nave defensora → ambas destruídas, nave inimiga com nave inimiga ou nave defensora com nave defensora → não pode e o jogo deve impedir isso).
- Fazer descer um ponto (bomba inimiga vai caindo) ou fazer subir um ponto (tiro defensor vai subindo).
- Deslocar rectângulos: nave defensora (conforme as ordens dos jogadores), nave invasora (consoante um algoritmo a ver mais adiante), *powerups* (estes, em princípio, vão simplesmente caindo).

As estruturas de dados resumem-se a isto. O resto do trabalho é, essencialmente, sistemas operativos. Quanto à visualização, há que dizer que em vez de se desenhar rectângulos, vão-se colocar pequenos *bitmaps* no ecrã, segundo a matéria que ainda vai ser dada e essa parte também é muito simples. O tamanho das naves não tem que ser igual para todas as naves, e os tiros e bombas poderão ser representados com mais de que um pixel.

A figura seguinte representa as ideias descritas acima, em particular o esquema conceptual do jogo. A representação usará *bitmaps* e vez de retângulos e pontos. A coordenada (0,0) no sistema Windows é canto superior esquerdo.




As coordenadas dos vários elementos em jogo são descritas de acordo com as coordenadas no plano. Não há necessariamente uma relação 1:1 entre pontos e pixéis (mas pode haver). Isto facilita o programa, removendo dependências entre o *código que controla as coisas* e o *código que desenha as coisas*. Um exemplo: o plano onde se desenrola o jogo pode ter, por exemplo, dimensões 1000x1000 e a janela onde se representa pode ter apenas (exemplo) 500x500 – e nesse caso cada 2x2 pontos seria representado num único pixel. Conselho: assuma que o plano tem uma dimensão sempre igual e suficientemente grande para que possa ser posteriormente desenhado, tanto em janelas grandes como pequenas.

**Regras do jogo**

- O jogo pode ser *singleplayer* ou *multiplayer*. No caso de ser *multiplayer*, os jogadores agem de forma cooperativa.
- Os invasores são naves de diversos tipos. Cada tipo tem um comportamento específico que define o seu movimento, a forma como lançam bombas, e a sua resistência. As bombas deslocam-se desde a nave que a disparou, na vertical até ao fundo do plano do jogo ou até colidir com um defensor.
- As naves defensoras podem disparar a qualquer momento. Os tiros deslocam-se sempre para cima e desaparecem quando colidem com um invasor, ou quando chegam ao topo do plano do jogo.
- Periodicamente aparecem *powerups* no topo do plano, que vão caindo a uma determinada velocidade (igual à das bombas). Se um defensor colidir com um *powerup*, apanha-o, despoletando o seu efeito. Um *powerup* que chegue ao fim do plano sem ser apanhado desaparece.
- O jogo é constituído por sucessivos níveis de dificuldade crescente. Os factores que aumentam a dificuldade são: número de naves invasoras, velocidade das naves invasoras, comportamento mais "complicado" das naves invasoras, frequência com que disparam bombas, etc. Este aspecto deve ser configurável.
- O nível termina quando a última nave invasora é destruída, ou quando a última nave defensora é destruída.

- Cada jogador pode ter mais do que uma vida. Esse aspecto deve ser configurável.
- A pontuação do jogador depende do número de invasores destruídos e a forma como isso se converte em pontos é deixado completamente em aberto (mas tem que haver uma contabilização).
- As naves inimigas têm um número de pontos de resistência. Cada tiro defensor retira-lhes um ponto. Quando chega a zero, o invasor é destruído. Algumas naves invasoras mais difíceis podem ter mais do que um ponto de resistência.
- As naves defensoras são destruídas assim que colidem com uma bomba, excepto se estiverem sob a protecção temporária de algum *powerup*. Os invasores podem deslocar-se para qualquer parte do plano. As naves defensoras só podem deslocar-se nos 20% inferiores do plano. As bombas descem sempre. Os tiros defensores sobem sempre.
- Nenhuma nave pode ocupar o mesmo espaço que outra nave. No caso de invasor x invasor ou defensor x defensor, o jogo simplesmente impede que isso aconteça. No caso de invasor x defensor, ambas as naves são destruídas (excepto se o defensor estiver sob alguma protecção temporária de um *powerup*, caso em que apenas o invasor é destruído).

A velocidade dos vários elementos é independente uns dos outros: tiros, bombas, naves defensoras, naves invasoras podem ter a sua velocidade intrínseca diferente dos restantes. Inclusivamente, as próprias naves invasoras podem exibir diversas velocidades.

O tipo de *powerups* é dado na tabela abaixo

| Powerup | Efeito | Duração | Ocorrência |
|---|---|---|---|
| Escudo | Durante N segundos o defensor fica imune às bombas inimigas. | Temporário | Vulgar |
| Gelo | Bloqueia os movimentos de todas as naves invasoras durante um determinado período de tempo (configurável). No entanto, continuam a disparar. | Temporário | Invulgar |
| Bateria | Aumenta a velocidade dos tiros dos defensores (de todos). | Temporário | Invulgar |
| Mais | Aumenta a velocidade de movimento das naves inimigas. | Temporário | Vulgar |
| Vida | Uma vida extra para o jogador. | Permanente | Raro |
| Álcool | Inverte o sentido das teclas desse jogador. | Temporário | Invulgar |
| ? | Invente os seus próprios objectos adicionais. Seja criativo. | ? | ? |

O tipo de invasores é indicado na tabela seguinte

| Invasor | Comportamento |
|---|---|
| Básica | Desloca-se horizontalmente. Quando atinge o limite, desce o equivalente à sua altura e desloca-se horizontalmente no sentido oposto. A atingir o limite inferior, reaparece no topo.<br>Desloca-se N pontos a cada X milissegundos. Esta é a nave mais lenta que existe e a sua velocidade é referida como velocidade base dos invasores.<br>Taxa de disparo: dispara a cada N movimentos (configurável). Esta é a taxa de disparo base.<br>Tem um factor de resistência de 1 (um tiro mata-a). |
| Esquiva | Desloca-se aleatoriamente, sendo muito complicado fazer-lhe pontaria. É 10% mais rápida que a nave básica. Esta nave dispara menos 40% que a básica, mas tem um factor de resistência de 3. |
| ? | Invente as suas naves e divirta-se. |

Algumas notas acerca das regras de jogos e outros pormenores.

- Todas as velocidades, taxas de disparo, etc. e efeitos dos *powerups* devem ser configuráveis. Pode-se assumir que todos os *powerups* têm a mesma duração.
- A constituição dos níveis (número de naves, de que tipo, etc.) é deixada em aberto.
- Variações de regras são toleradas desde que não removam complexidade ou dimensão.
- Os professores não têm necessidade de micro-gerir os trabalhos. Se pensar em regras novas (ou modificações a estas) que façam sentido, siga em frente (sem retirar dimensão nem complexidade).

Opcionalmente, o plano do jogo pode conter também obstáculos fixos para tornar o jogo mais interessante. Esses obstáculos podem ser, por exemplo, obstáculos que impedem o movimento do defensor, dificultando-lhe a fuga às bombas. Ou então, podem ser obstáculos que existem acima dos defensores, servindo de escudo contra bombas (podem inclusivamente ir sendo destruídos). Isto são meras hipóteses opcionais. Podem ser acrescentados elementos novos que tornem o trabalho mais interessante.

**Interface com o utilizador**

O jogo será representado graficamente com os recursos do Windows (*bitmaps*, *icons*, mecanismos de desenho, etc.). A representação gráfica deverá manter-se actualizada e reflectir todas as acções de todos os jogadores envolvidos. Os diversos elementos em jogo devem ter uma representação que os permita distinguir (inclusive distinguir entre tiros e bombas). Os *powerups* devem ter um efeito visível nas naves defensoras afectadas.

O controlo do movimento é feito por teclas. O jogador pressiona teclas (configuráveis) e o movimento da nave será feito na direção associada a essa tecla enquanto a tecla estiver premida.

Os movimentos devem ser fluidos: saltos do tipo "quadrícula em quadrícula" são de evitar. A actualização do ecrã deve evitar o efeito de cintilação.

**Arquitetura**

O jogo é constituído pelos 3 seguintes programas:

- **Servidor**: controla todos os dados, implementa as regras e valida e concretiza todas as ações. É acessível por memória partilhada, nas condições descritas já a seguir. O servidor será uma aplicação gráfica com uma interface que permitirá configurar todos os aspectos configuráveis mencionados no enunciado. O servidor deverá ter várias *threads* da seguinte forma:
  - Uma *thread* por cada tipo de invasor em jogo. Pode ser uma *thread* por cada nave invasora, mas pelo menos uma por cada tipo.
  - Uma *thread* para gerir os tiros dos defensores, as bombas dos invasores e a queda dos *powerups*.
  - Uma *thread* para gerir a temporização e dos efeitos dos *powerup* que foram apanhados.
  - Pelo menos uma *thread* para gerir os jogadores (pode ser só uma *thread*).

  Poderá haver mais *threads*, mas esta enumeração ser o propósito duplo de: ajudar a estruturar o servidor e especificar um conjunto de requisitos mínimos a cumprir.
- **Cliente**: atua como mero interface com o utilizador. Não define nem gere regra nenhuma de jogo. Recolhe as teclas (configuráveis) e envia-as ao jogo, e obtém a informação do estado actual do jogo e representa-o graficamente no ecrã. O cliente é gráfico e suporta apenas um jogador de cada vez.

O cliente pode ser executado na máquina do servidor ou numa máquina diferente. Em qualquer dos casos, o mecanismo de comunicação é apenas o *named pipe*. Cada utilizador usa o seu próprio cliente. A configuração das teclas é local ao cliente e não tem nada a ver com o servidor.

- **Gateway**: faz a ponte entre a o cliente e o servidor. Trata-se de uma aplicação de consola que interage com o servidor através de memória partilhada, para acesso aos dados do jogo, e com os clientes através de *named pipes*. O seu objetivo é encaminhar a informação dos clientes para o servidor e vice-versa.

Outros elementos estruturantes

- **Memória partilhada**. Para uso entre servidor e o *gateway* e organizada em duas zonas:
  - **Zona de mensagens** (para leitura e escrita de ambos – servidor e *gateway*): esta zona será organizada como um *buffer* em que o *gateway* coloca mensagens oriundas dos clientes para que o servidor as leia e atenda. O exemplo dos produtores-consumidores das aulas poderá ser útil neste caso.
  - **Zona de dados do jogo** (para leitura do *gateway*, leitura e escrita do servidor). Nesta zona o servidor manterá as estruturas de dados descritivas do jogo. Quando houver alterações, o servidor notificará o *gateway* através de um mecanismo se sincronização apropriado para que este vá consultar o novo estado do jogo e informe os clientes.
- **DLL**. As funções para gerir a zona de mensagens da memória partilhada deverão estar numa DLL, a ser usada tanto pelo servidor como pelo *gateway*.

**Ciclo de vida do jogo**

O jogo decorre através de uma sequência fixa de etapas controlada pelo servidor:

- **Criação do jogo**. O jogo é primeiro criado no servidor através da sua interface gráfica (com recurso a componentes gráficos adequados - *edit boxes*, *check boxes*, etc.) onde são definidos:
  - O número máximo de jogadores humanos (assuma um limite superior fixo, por exemplo, 20).
  - Os parâmetros configuráveis (número de naves inimigas, número de powerups, duração, probabilidades de ocorrência de powerups, número de vidas iniciais, etc.).
  - Estes dados são especificados pelo utilizador do servidor.
- **Associação ao jogo**. Os jogadores, através dos seus clientes podem associar-se ao jogo, desde que este ainda não se tenha iniciado e não se ultrapasse o limite de jogadores estabelecido por quem o criou. É fornecido um nome para registo de pontuações.
- **Início do jogo**. É dado no servidor, através da sua interface, e desde que já haja pelo menos um jogador associado. Após este momento não são aceites mais jogadores nesse jogo.
- **Decorrer do jogo**. O jogo decorre – todos os clientes participantes no jogo recebem as atualizações devidas, mesmo que o seu jogador humano não faça nada. Os jogadores que vão perdendo deixam de controlar a sua nave (que, entretanto, desaparece da área de jogo), mas continuam a ver os outros jogadores (a não ser que se desliguem do servidor, o que poderão fazer a qualquer altura, mesmo antes de perder).
- **Final do jogo**. Quando se atingem as condições de fim de jogo (o último jogador humano morreu, ou foram destruídas todas as naves inimigas). O jogo encerra, as *threads* no servidor dedicadas aos clientes e ao controlo deste jogo em particular terminam ou são suspensas (no caso de *worker*

*threads*), e os clientes terão que voltar a ligar a um novo jogo (nota: "ligar a um jogo" ≠ "ligar ao servidor"). No final do jogo, o cliente obtém a informação do "top-10" de pontuações relativo aos últimos jogos.

**Características do cliente e da interação com o jogador**

- O cliente tem dois modos de interação com o jogador:
  - Modo de início: para o estabelecimento de uma ligação ao servidor. A identificação do utilizador é pedida ao utilizador (via, ex., *dialog box*) e enviada ao servidor (via gateway).
  - Modo do jogo: para o decorrer do jogo. O jogo em si é essencialmente controlado por teclas (configuráveis) e com recurso a imagens, não havendo aqui os componentes usados no modo de início.
- É suposto que a totalidade da superfície do jogo seja visível ao mesmo tempo no cliente. Isto pode obrigar ao uso de janelas que ocupem a quase totalidade do ecrã, ou a fazer corresponder a cada N pontos do plano 1 pixel na janela (fator de escala).
- O cliente deve utilizar elementos gráficos adequados, com uma qualidade visual aceitável. Cintilação não será considerada aceitável. As naves dos jogadores e/ou inimigos devem ter um aspeto diferente do habitual quando estão sob o efeito de um objeto temporário. As naves de jogadores distintos devem ter uma cor/aspeto que as distingam.

Nota: O cliente não comunica directamente com o servidor, apenas com o gateway.

**Aspetos adicionais relacionados com a funcionalidade**

- O servidor deverá ser responsável pela animação das naves inimigas, pela gestão dos objetos e seus efeitos. Para tal utilizará *threads*, objetos de sincronização e de comunicação, e demais recursos do sistema que forem apropriados. Deve ser dada primazia ao API Win32 sobre biblioteca C, exceto em questões periféricas.
- A identificação e pontuação dos jogadores vão sendo memorizadas em chaves do *registry* da máquina onde está a correr servidor, sendo a sua gestão exclusivamente do servidor.
- Deve ser dada especial atenção à questão da sincronização de forma a garantir as regras quanto às colisões e sobreposições de naves e gestão correta de acontecimentos do jogo.

**Aspectos extra**

Poderá realizar funcionalidades opcionais com valorização extra. Esta valorização poderá compensar outros aspectos menos bons no trabalho, mas não permitirá ultrapassar os 100%.

- Efeitos sonoros/musicais (sem se limitar a meros "beeps") – 5%
- Animações gráficas (por exemplo, explosões) – 10%

**Algumas chamadas de atenção**

- Não coloque ponteiros em memória partilhada (pelas razões explicadas/a explicar nas aulas teóricas). Isto abrange ponteiros seus e também objetos biblioteca que contenham internamente ponteiros (por exemplo, objetos *STL String*, *Vector*, etc.).

- Pode usar C++ se quiser. Os elementos do jogo são relativamente diretos e pouco precisam de polimorfismo. Se usar C++ na mira de usar polimorfismo, não se esqueça que o polimorfismo em C++ requer ponteiros e que não pode colocar ponteiros na memória partilhada.
- Planeie cuidadosamente as estruturas de dados, a interação entre os diversos programas, e arquitetura dos mecanismos de comunicação antes de avançar no código. Siga o modelo dado no início do enunciado que é o mais simples.
- Vá testando os seus programas de forma incremental. Não comece a testar apenas na véspera da data de entrega.
- Faça cópias de segurança do trabalho ou use um repositório do tipo *git*.
- Grupos de dois alunos. É importante ver o que a ficha de unidade curricular diz acerca do TP.

## 2. Prazos

Existem duas entregas: Meta 1 e Meta final.

### Meta 1 – 12 de Maio

Aplicação servidor com o lançamento de *threads* e utilização de memória partilhada. Esta meta envolve essencialmente:

- Lançamento de *threads* para controlo das naves inimigas.
- Uso de memória partilhada e eventuais problemas de sincronização que surjam.
- Definição clara das estruturas de suporte ao jogo em memória partilhada.
- DLL.

Material a entregar:

- Relatório: muito breve a explicar os pontos essenciais da implementação da memória partilhada, as estruturas de dados que contém e a sua utilidade, os aspectos de sincronização que haja e como foram resolvidos.
- O projeto do servidor.

### Entrega final – 14 de Junho

A entregar:

- Trabalho completo, com toda a funcionalidade, envolvendo os vários projetos: cliente, servidor e *gateway*.
- Relatório completo, com a descrição detalhada dos programas que constituem o trabalho.

## 3. Avaliação

A avaliação e defesas são feitas, essencialmente, na entrega final. Na meta 1 avalia-se o cumprimento dos objetivos estabelecidos. Pode haver apresentação na meta 1, mas a defesa será essencialmente feita na entrega final.

Tal como descrito na ficha da unidade curricular:

Nota da meta 1 = valor entre 0,8 e 1,0

Nota da meta 2 = valor entre 0 e 6

Nota final do trabalho = nota obtida na meta 2 x nota obtida na meta 1 (e influenciada pela defesa).

A meta 1 sem meta 2 não terá qualquer valor.