Sistemas Operativos 2

2018/19

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.

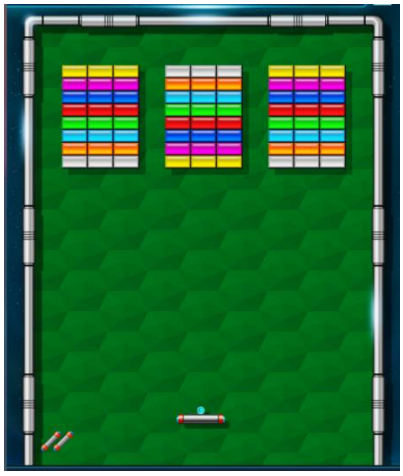
1. Descrição geral do tema – *Arkanoid / Breakout*

Pretende-se que implemente na plataforma Win32 um jogo de habilidade em que conduz uma barreira que servirá para impedir que uma bola em movimento saia do ecrã. A bola está sempre em movimento e no seu percurso vai colidindo com tijolos de vários tipos e cores, que são destruídos na colisão (conforme o seu tipo). O objectivo é manter a bola no ecrã ("ecrã" → "área de jogo") e destruir os tijolos todos. Este jogo é extremamente simples e as frases anteriores descrevem o seu essencial. Existem diversas implementações *online* que se aconselha a ver para entender o conceito. É preciso ter em atenção que o que é pedido neste enunciado tem algumas diferenças face às versões que se encontram *online* (por exemplo: neste enunciado vamos ter multi-player, e um aspecto gráfico diferente).

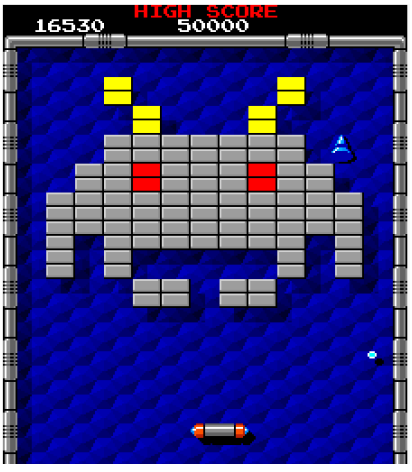
Alguns *links* para implementações online para melhor entendimento do conceito do jogo:

- http://www.agame.com/game/arkanoid
- https://www.crazygames.com/game/atari-breakout

Algumas imagens para exemplo:



Exemplo 1



Exemplo 2

Exemplo 3

Características do jogo

As entidades envolvidas no jogo são: Bola, Barreira, Tijolo, Brinde

Bola: Existe uma bola em movimento contínuo, sempre na diagonal com inclinação de 45 graus face aos extremos da área de jogo. Isto significa que sempre que a bola se desloca uma determinada distância na horizontal, também se desloca a mesma distância na vertical. A bola inverte o movimento sempre que colide com algo. Essa inversão ocorre apenas no eixo segundo o qual ocorreu a colisão. Exemplo: a bola estava a deslocar-se *para cima e para a direita* e embateu contra o limite direito da área de jogo. Após a colisão a bola desloca-se agora para a *esquerda* mas *continua a deslocar-se para cima*.

Barreira: O jogador controla uma barreira que se encontra na parte inferior da área de jogo e serve para "aparar" a bola, impedindo-a de sair da área de jogo. A barreira tem uma determinada dimensão. Quanto maior mais fácil será o jogo. A dimensão pode variar consoante o nível e consoante determinados acontecimentos durante o jogo. A velocidade da barreira e da bola são inicialmente iguais.

Tijolo: Pequena área rectangular com uma determinada posição na área de jogo. Existem diferentes tipos com representações visuais distintas. Existe inicialmente uma determinada quantidade de tijolos numa configuração que pode ser aleatória ou predefinida, consoante o nível. Os tipos de tijolos são:

- ***Normal***: é destruído numa colisão com a bola.
- ***Resistente***: precisa de mais do que uma colisão para ser destruído. O número de colisões vai de 2 a 4, e esse número (assim como quantos falta para o destruir) deve reflectir-se visualmente.
- ***Mágico***: Ao ser destruído larga um brinde que irá caindo a uma velocidade fixa (aproximadamente a mesma velocidade da bola e da barreira). Os tijolos mágicos precisam apenas de uma colisão para serem destruídos.

Brinde: são pequenos objetos (mesmo tamanho que a bola) que aparecem como resultado da destruição de tijolos mágicos. Aparecem no local onde se encontrava o tijolo que lhes deu origem e vão caindo a uma velocidade constante da mesma ordem de grandeza que a velocidade da barreira. Podem ser do tipo:

- ***Speed-up***: aumenta a velocidade da bola em 20% até ao máximo do dobro da velocidade normal.
- ***Slow-down***: diminui a velocidade da bola em 20% até ao mínimo de 60% da velocidade normal.

Os efeitos de *speed-up* e *slow-down* têm uma duração fixa (configurável, ex: 1 minuto).

- ***Vida-extra***: Tem o efeito que o seu nome indica.
- ***Triple***: Faz com que apareçam duas novas bolas em movimento a partir de locais aleatórios e com direcções aleatórias (na diagonal) à mesma velocidade que a bola já existente. Não acumula efeitos com outros brindes deste tipo.
- ***Outros***?: tem a total liberdade de inventar outros brindes, se o desejar (exemplos: aumentar a barreira, diminuir, etc.).

Os brindes produzem efeitos apenas se forem apanhados pela barreira. Se não forem apanhados, desaparecem pelo fundo da área de jogo e perdem-se. Os brindes podem colidir com a bola, fazendo com que a bola mude de direcção em conformidade, mas a trajetória do brinde não é modificada. Os brindes podem atravessar tijolos (é como se os tijolos não estivessem lá).

A velocidade do movimento dos vários elementos em jogo não é necessariamente a mesma para todos.

Modelo de dados envolvido

Apesar do aspecto gráfico, um jogo deste tipo é extremamente simples e resume-se a movimentar e gerir rectângulos e quadrados no ecrã. Os tijolos e a barreira são áreas retangulares, a bola e os brindes são, no fundo, áreas quadradas, e muita da lógica resume-se a detetar se duas áreas se sobrepõem (colisão). A parte gráfica é também relativamente direta – é muito simples desenhar imagens no ecrã.

À primeira vista, o modelo de dados poderia ser baseado na ideia de uma tabela bidimensional como se a área de jogo fosse um tabuleiro (uma célula representa uma posição possível para a bola e tijolos). No entanto, o movimento resultante seria "aos saltos" (a não ser que as células fossem muito pequenas, mas então a tabela seria enorme, gastando muita memória). Outro aspecto negativo associado à estratégia do tabuleiro é o fato dos tijolos terem dimensões diferentes das bolas e brindes, assim como a barreira ter uma dimensão diferente de todos os outros elementos, fazendo a gestão de "células ocupadas" bastante mais complexa do que inicialmente seria de supor. Assim, a solução adequada será simplesmente cada elemento em jogo estar associado às suas coordenadas e dimensões e não haver tabela bidimensional nenhuma.

Os dados relativos à mecânica do jogo (e apenas estes) resumem-se a estruturas de dados capazes de representar as áreas retangulares ocupadas pelos tijolos, bolas, barreiras e brindes. Os algoritmos necessários ao jogo em si têm apenas a ver com detecção de colisões e alteração de coordenadas para concretizar os movimentos.

O principal do trabalho é, essencialmente, sistemas operativos. Quanto à visualização, há a assinalar 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 é muito simples.

Lógica do jogo:

O jogo pode ser organizado em níveis ("pode" é uma sugestão). Cada nível terá uma determinada configuração de tijolos (que lhe dará maior/menor dificuldade) e que pode ser aleatória, predefinida ou lida de um ficheiro.

O nível termina quando todos os tijolos são destruídos (jogador ganhou) ou quando a bola sai da área de jogo e o jogador não tem mais "vidas" (jogador perdeu). O jogador tem inicialmente 3 vidas. Começar um novo nível não repõe o número de vidas.

O jogo deve permitir vários jogadores, havendo então uma barreira por cada jogador. As barreiras não se podem sobrepor e os jogadores agem de forma cooperativa. No caso de existirem vários jogadores, o tamanho de cada barreira será deverá ser menor do que se houver apenas um jogador.

As regras do jogo são de importância menor. O que interessa mais é o uso dos mecanismos do sistema, que decorre da arquitectura da solução e do tipo de recursos usados. Se quiser mudar algumas regras, faça-o desde que isso não tenha impacto na estrutura, recursos e API usados.

Algumas notas acerca das regras de jogo e outros pormenores:

- Todas as velocidades, probabilidades de aparecimento de brindes, etc. devem ser configuráveis.
- A constituição dos níveis (configuração dos tijolos) é deixada em aberto e tem várias formas de fazer isso.
- 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).

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 visual que os permita distinguir.
O controlo do movimento é feito por teclas e rato. O rato permite fazer movimentos mais rápidos (mas mais difíceis de controlar), enquanto as teclas são mais fáceis mas eventualmente menos rápidas. O rato permitirá posicionar rapidamente a barreira numa determinada posição enquanto as teclas indicam que se deseja movimentar numa determinada direcção enquanto a tecla estiver premida.
O cliente recebe os *inputs* (teclas/rato) e traduz para a informação correspondente a enviar ao servidor. Ou seja, o servidor não tem que receber directamente a informação "crua" respeitante a códigos de teclas e afins, mas sim informação relativa ao que deve fazer (códigos tais como MOVER_DIREITA em vez de código *ascii* de uma tecla em particular). As teclas devem ser configuráveis no cliente (mais uma vez: o servidor não precisa de saber que tecla foi premida: apenas precisa de saber o que o cliente pretende fazer).
Os movimentos devem ser fluídos: saltos do tipo "quadrícula em quadrícula" são de evitar (evite usar o conceito de "tabuleiro"). A actualização do ecrã deve evitar o efeito de cintilação.

Arquitetura

O jogo será gerido por um programa central e a interface com o utilizador será feita por um outro programa. A lógica de interação entre ambos os programas é a de cliente-servidor. O programa central será o servidor e a interface será o cliente. Haverá tantos programas cliente quanto jogadores envolvidos. Haverá apenas um servidor numa determinada máquina. Os clientes podem estar em máquinas deferentes da do servidor.
Os mecanismos de comunicação serão: *named pipes* e memória partilhada. Quanto a mecanismos de sincronização, existirão aqueles que forem necessários. Os clientes não interagem diretamente uns com os outros. A funcionalidade do cliente (ótica do cliente) de interacção entre o cliente e o servidor estará maioritariamente codificada numa DLL.
O jogo é então constituído pelos 3 seguintes programas (projetos): Servidor, Cliente, DLL

- **Servidor**: controla todos os dados, implementa as regras e valida e concretiza todas as ações. É acessível por memória partilhada ou por *named pipe*, nas condições descritas já a seguir. O servidor será uma aplicação com interface consola. A configuração é feita a través de um ficheiro de texto descrito mais adiante. O servidor irá gerir uma lista de jogadores mais pontuados no *Registry* (top 10).
  O servidor deverá ter várias *threads* da seguinte forma:
  - Uma *thread* para gerir o movimento dos brindes.
  - Uma *thread* para gerir a(s) bola(s).
  - Pelo menos uma *thread* para gerir os jogadores (pode ser só uma *thread*).

  Poderá haver mais *threads*. A lista indicada acima tem o propósito duplo de ajudar a estruturar o servidor e especificar um conjunto de requisitos mínimos a cumprir.
  A duração dos efeitos dos brindes pode ser gerida por uma *thread*, ou então por outra forma qualquer.

- **Cliente**: atua como mero interface com o utilizador. Não define nem gere regra nenhuma de jogo. Inicialmente pede um *username* ao utilizador e envia-o ao servidor (sem *password* – é apenas para identificação e atribuição da pontuação); durante o jogo recolhe o *input* do utilizador e comunica com o servidor; e obtém a informação do estado atual do jogo representando-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. Dependendo de onde corre o programa cliente, o mecanismo de comunicação será memória partilhada ou *named pipe,* respetivamente (ver no ponto a seguir). Cada utilizador usa o seu próprio cliente. A configuração das teclas é local ao cliente e não tem nada a ver com o servidor.

- **DLL**: encapsula toda a comunicação com o servidor na ótica do cliente e implementa a funcionalidade de enviar/receber informação ao/do servidor. Esta funcionalidade internamente é encaminhada para uma forma se o cliente estiver a correr na mesma máquina do servidor (cliente local); ou outra forma se o cliente estivera a correr numa máquina diferente da do servidor (clientes remotos). A funcionalidade exportada aos clientes deverá ocultar o mais possível estas diferenças.

  As funções a implementar na DLL devem incluir pelo menos as seguintes:
  - *Login(...)* : envia um pedido de login dado o *username* introduzido pelo utilizador ao servidor.
  - *ReceiveBroadcast(...)*: recebe informação atualizada do servidor (enviada a todos os clientes).
  - *SendMessage(...)*: envia pedido de movimentação da barreira, ver top10, e sair do jogo.
  - *ReceiveMessage(...)*: recebe a resposta de um pedido feito pelo cliente (*Login()* ou *SendMessage())*

  Estas funções permitem ao programa cliente abstrair a forma de ligação e não se preocupar com os pormenores da ligação (memória partilhada ou named pipes). Pode haver mais funções na DLL.

Outros elementos estruturantes:

- **Memória partilhada**. Para uso entre servidor e o cliente *local.* É organizada em duas zonas:
  - **Zona de mensagens** (para leitura e escrita de ambos – servidor e cliente *local*): esta zona será organizada como um *buffer* em que o cliente coloca as suas mensagens para que o servidor as leia e atenda. O exemplo dos produtores-consumidores das aulas poderá ser útil neste caso. Pode envolver mecanismos de sincronização para gerir o buffer.
  - **Zona de dados do jogo** (para leitura do cliente *local*, leitura e escrita do servidor). Nesta zona o servidor manterá as estruturas de dados descritivas do jogo. Quando houver alterações, o servidor notificará o cliente através de um mecanismo se sincronização apropriado para que este vá consultar o novo estado do jogo e atualize a informação visível na interface.

  As funções para gerir a zona de mensagens e de dados do jogo da memória partilhada estarão implementadas pelo servidor (ótica do servidor) e na DLL (para a ótica do cliente)

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 com base na configuração dada pelo ficheiro indicado na linha de comandos, e onde são definidos:
    - O número máximo de jogadores humanos (assuma um limite superior fixo, por exemplo, 20).
    - Os parâmetros configuráveis (número de níveis, número de *speed-ups* e *slow-downs*, duração, probabilidades de ocorrência de *speed-ups* e *slow-downs*, número de vidas iniciais, quantidade inicial de tijolos, velocidades de movimentos, etc.).
        - O nome do ficheiro de configuração é dado no arranque do servidor como um parâmetro de linha de comandos.
        - O formato do ficheiro de configuração é deixado em aberto mas deve obedecer aos seguintes requisitos:
            - Deve poder ser editado de forma simples (exemplo, ficheiro de texto/notepad).
            - O formato deve ser explicado no relatório.
- **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 máximo de jogadores estabelecido no servidor. É fornecido um nome para registo de pontuações.
- **Início do jogo**: É desencadeado no servidor, através da sua interface, e desde que já exista pelo menos um jogador associado. Após este momento, os clientes que se ligarem já só poderão ver, mas não jogar.
- **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.
- **Final do jogo**: Quando se atingem as condições de fim de jogo (o último jogador morreu, ou foram ultrapassados todos os níveis). 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 associar a um novo jogo (nota: "associar a um jogo" ≠ "ligar ao servidor" e "jogo" ≠ "servidor").

  No final do jogo, o cliente obtém a informação do "top-10" de pontuações relativo aos últimos jogos. A pontuação é baseada no número de tijolos partidos. Em caso de *multi-player*, considera-se que o tijolo foi partido pelo jogador cuja barreira aparou a bola em último lugar antes desta tocar no tijolo. Pode sugerir e usar outros critérios de pontuação mais criativos desde que melhorem o jogo (exemplo: considerar brindes apanhados, numero de vezes que a bola foi aparada, etc.)

Características do cliente quanto à forma de 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 (*username*) é pedida ao utilizador (via, ex., *dialog box*) e enviada ao servidor.
    - Modo do jogo: para o decorrer do jogo. O jogo é essencialmente controlado por rato e teclas e a interface é construida com recurso a imagens, não havendo aqui os componentes usados no modo de início.
- A totalidade da superfície do jogo deve ser visível ao mesmo tempo no cliente. Isto pode obrigar ao uso de janelas que ocupem a quase totalidade do ecrã. Não pode assumir um ecrã maior que *Full HD*.

- O cliente deve utilizar elementos gráficos adequados, com uma qualidade visual aceitável. Cintilação não será considerada aceitável. As barreiras dos jogadores e bolas devem ter um aspeto diferente do habitual quando estão sob o efeito de um brinde. As barreiras de jogadores distintos devem ter uma cor/aspeto que as distingam.

Aspetos adicionais relacionados com a funcionalidade

- O servidor deverá ser responsável pela animação das bolas e brindes, pela gestão dos objetos e seus efeitos. Para tal utilizará *threads*, objetos de sincronização e de comunicação, e os demais recursos do sistema que forem apropriados. Deve ser dada primazia ao API Win32 sobre bibliotecas de C, exceto em questões periféricas.
- A identificação e pontuação dos jogadores são 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 barreiras e a 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 dos tijolos) – 10%
- Tijolos com movimento – 10 %
- Implementação do servidor como Serviço NT – 5 %

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 de 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 a arquitetura dos mecanismos de comunicação antes de avançar no código.
- 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* **que seja privado** (se fizer um repositório publico que depois seja copiado por terceiros, será considerado culpado de partilha indevida de código e terá a devida penalização na nota).
- Grupos de dois alunos. Não existem situações que permitam grupos maiores e não vale a pena perguntar se podem ou não exceder esse número.

- Partilha de código. O código de um grupo não pode ser partilhado a outros grupos. Uma coisa é ajudar dando ideias e explicando dúvidas, outra coisa bem diferente é dar código. As partilhas de código detectadas terão impacto negativo em quem recebe e também em quem disponibiliza.
- É natural obter pequenas "receitas" e exemplos com código *online*. No entanto, todo o código apresentado poderá ser questionado na defesa e os alunos têm obrigatoriamente que o saber explicar.

Convém ver o que a ficha de unidade curricular diz acerca do TP.

2. Prazos

Existem duas entregas: Meta 1 e Meta final.

Meta 1 – 5 de Maio

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

- Lançamento de *threads* para controlo da bola (ainda não há outras bolas nem brindes).
- Uso de memória partilhada e solução de eventuais problemas de sincronização que surjam.
- Definição clara das estruturas de suporte ao jogo em memória partilhada.
- DLL – funcionalidade a ser usada pelo cliente local (memória partilhada).
- Armazenamento e recuperação do top10 no *registry* (pode usar valores fictícios apenas para começar e testar)
- Um cliente muito simples em consola que invoca cada funcionalidade da DLL através de uma sequência pré-definida (exemplo: cliente consola pede o *username* ao utilizador; envia ao servidor; recebe confirmação/rejeição; e entra em ciclo a receber novas posições da bola até uma tecla ser premida pelo utilizador).

Material a entregar:

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

Entrega final – 4 de Junho

A entregar:

- Trabalho completo, com toda a funcionalidade, envolvendo os vários projetos: cliente, servidor e DLL (funcionalidade memória partilhada e named pipes).
- 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 através de uma apresentação obrigatória. A defesa será essencialmente feita na entrega final. Tal como descrito na ficha da unidade curricular:

- Nota da meta 1 = fator multiplicativo entre 0,8 e 1,0
    - A não comparência na apresentação obrigatória da meta 1 fará com que a mesma não seja considerada. Ou seja, o fator multiplicativo obtido será o mesmo que obteria se não entregasse a meta (0,8).
- Nota da meta 2 = valor entre 0 e 6 (influenciado pela defesa)
    - O contributo de cada elemento do grupo vai ser individualmente analisado. Isto significa que dois alunos do mesmo grupo podem obter notas diferentes em função do seu contributo para o trabalho e da qualidade da defesa.
- Nota final do trabalho = nota obtida na meta 2 * fator multiplicativo obtido na meta 1
- A meta 1 não tem qualquer valor sem a meta 2. Ou seja, grupos que não entregarem a meta 2 ou que não compareçam na sua defesa terão uma nota final de 0 valores.
- A falta da meta 1 não impede a entrega da meta 2. Apenas prejudica a nota final (0,8 * nota meta 2).

O trabalho está planeado para ser feito ao longo do resto do semestre e a entrega do trabalho prático é única para todo o ano letivo. Isto significa que não existirá trabalho prático na época especial ou noutras épocas extraordinárias que os alunos possam ter acesso.