Sistemas Gráficos Interativos
Aulas Práticas

M.EIC – 2023/2024

Trabalho prático 3
Desenvolvimento de um jogo

Versão 0.1, 18/11/2023

1. Introdução

Pretende-se, com este trabalho, desenvolver um pequeno jogo 3D. O jogo consiste na corrida entre dois carros, um deles controlado pelo jogador, e o outro movido/animado autonomamente.

No contexto deste trabalho é esperado que a linguagem YASF continue a ser usada pelo menos parcialmente, de modo a possibilitar um mais ágil desenvolvimento do jogo. Fica ao critério dos autores quais os tópicos do trabalho em que a mesma não seja utilizada; não se prevê a partilha de ficheiros XML, pelo que é possível introduzir novas funcionalidades pelos autores ou alterar os elementos existentes na linguagem.

Alguns elementos da cena 3D podem ser modelados em outras ferramentas e importados para o jogo (por exemplo em formato obj, gltf…).

2. Mecânica do jogo

Independentemente da criatividade colocada no desenvolvimento do jogo, é previsto o seguinte:

- No início, o jogador escolhe um carro para si e um carro autónomo que será o seu adversário.
- Dada a partida, compete ao jogador "guiar" o seu carro na pista, evitando a colisão com os obstáculos e com o carro adversário, assim como evitando saídas de pista; é de interesse do jogador colidir com *power-ups*.
- Os dois carros devem dar 3 voltas à pista (ou outro valor configurável).
- No final deve ser apresentado, em um *display*, o nome do jogador, o tempo realizado e o lugar obtido (1º ou 2º lugar).

Durante a prova, a mecânica do jogo baseia-se nos pontos seguintes, suportadas em mecanismos de Computação Gráfica / ThreeJS já conhecidos ou a estudar durante a realização do trabalho:

- O jogador deve controlar o seu carro usando as teclas para percorrer a pista sem sair da mesma, realizando três voltas completas num tempo limitado.
- Quando o carro sai da pista, a sua velocidade máxima (em valor absoluto) deve ser reduzida (por exemplo, passando para 70% da velocidade máxima normal) até que regresse à pista.
- Quando o carro colide com o carro adversário, a sua velocidade máxima (em valor absoluto) deve ser reduzida (por exemplo, passando para 70% da velocidade máxima normal) durante um tempo de ***N*** segundos (configurável).
- Sempre que o carro colide com um *power-up*, ganha o benefício associado, de acordo com a secção 3.3.
- Se colide com um obstáculo, sofre penalizações associadas, referidas na secção 3.4.
- Os power-ups e os obstáculos permanecem em pista após serem sujeitos a uma colisão.
- O tempo decorrido e outras informações devem ser mostradas num HUD (*Heads-Up Display*), descrito na secção 3.5.

3. Elementos de jogo

3.1 Base do jogo

A base do jogo constitui-se de uma pista onde se movimentarão dois carros, um concorrente e um adversário; espalhados pela pista poderão existir pontos de aquisição de benefícios (*power-ups*) e pontos de dificuldades (*obstacles*).
A base de jogo deve por isso incluir pelo menos quatro camadas (*layers*), de acordo com o seguinte texto e como exemplificado na figura 1:

- ***Layer Track***
  - Contém o trajeto da pista; o respetivo eixo de via é definido por uma curva de Catmull-Rom (traço interrompido na figura 1) da qual o primeiro e o último pontos são unidos automaticamente por um segmento de reta; o primeiro ponto da curva deve ser interpretado como o ponto de partida/meta; a largura da pista é constante ao longo do percurso.
- ***Layer Power-ups***
  - Contém a lista de pontos *(x,z)* do mapa onde cada *power-up* deve ser colocado/gerado (pontos marcados com quadrados verdes, na figura 1). Podem existir vários tipos de *power-ups*, a definir pelos autores.
- ***Layer Obstacles***
  - Contém a lista de pontos *(x,z)* do mapa onde cada obstáculo deve ser colocado/gerado (pontos marcados com triângulos vermelhos, na figura 1). Podem existir vários tipos de *obstacles*, a definir pelos autores.
- ***Layer Routes***
  - Cada rota (*route*) corresponde a uma lista de "chaves" (pontos marcados com pequenos círculos ciano, na figura 1) que serão utilizadas na animação de um carro "autónomo" ao longo da pista (podem existir uma ou mais rotas).

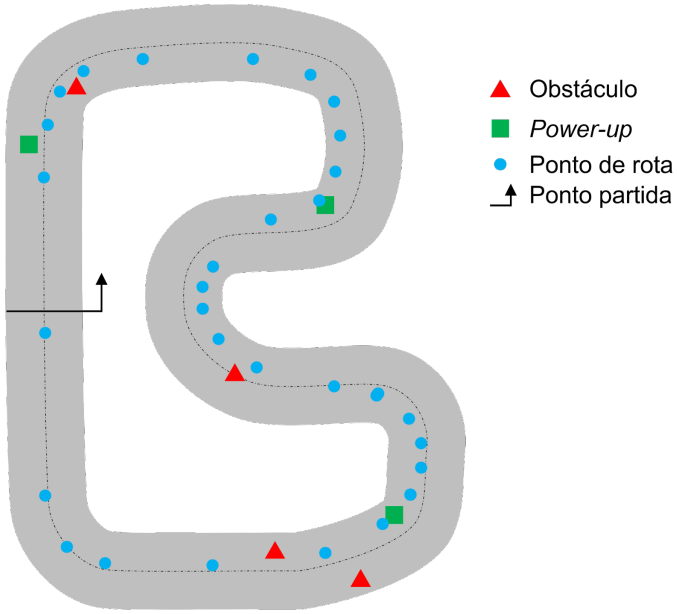

Figura 1: Ilustração da base de jogo com pista, obstáculos, power-ups e uma rota.

Pretende-se assim:

- Criar a classe ***MyTrack*** para o desenho da pista.
- Criar a classe ***MyRoute*** para armazenar os pontos-chave de uma rota.
- Criar as classes ***MyObstacle***, ***MyPowerUp***, ***MyVehicle*** com representações 3D dos respectivos elementos.
- Criar uma classe ***MyReader*** para interpretar a informação anterior e instanciar objetos das classes acima, nas posições ***(x, z)*** correspondentes.

3.2 Carros

Podem existir um ou mais modelos de carro. A mesma base geométrica pode ser decorada com diferentes texturas.

Alguns detalhes sobre um carro:

- Deve ser controlado por meio das teclas WASD, respetivamente para acelerar, travar, virar à esquerda e virar à direita.
- Quando em curva, as rodas da frente devem virar e o *pivot* de rotação do carro deve situar-se no centro do eixo traseiro.
- Deve ser possível observar as quatro rodas girando em torno do seu eixo, em função da velocidade do carro.
- Pode ter velocidade “negativa” (ou seja, andar em “marcha-atrás”).
- Deverá ser dotado de uma velocidade máxima.

No início do jogo, o utilizador pode escolher o modelo de carro que deseja. Pode também escolher um carro autónomo com o qual concorrerá; a cada carro autónomo deve corresponder uma rota.

NOTA: na fase inicial do desenvolvimento, é aconselhável que seja utilizado um modelo geométrico muito simples para cada carro, por exemplo um retângulo com textura (vista de cima).

3.3 Power-ups

Desenvolva pelo menos dois tipos distintos de power-ups, sendo que cada tipo oferece um benefício diferente. Sugestões (outras poderão ser definidas) possíveis de benefícios a implementar:

- Permitir uma velocidade máxima igual a 200% da velocidade máxima normal durante ***N*** segundos (configurável), sendo que a cada momento deve apresentar-se o tempo restante na interface. Terminado esse tempo, a velocidade máxima retorna ao valor normal.
- Redução, no tempo total, de uma parcela a definir (configuração).

NOTAS:

- É obrigatório, em complemento a um benefício, que o jogador escolha um novo obstáculo de entre os disponíveis num parque próprio e o coloque no ponto da pista que pretender (para o efeito, o jogo fica em pausa). Mais detalhes podem ser encontrados na secção 4.1.
- A forma geométrica 3D de representação dos *power-up* é livre.
- Os carros autónomos não são influenciados por *power-ups*.

3.4 Obstáculos

Desenvolva pelo menos dois tipos distintos de obstáculos, sendo que cada tipo oferece uma penalização diferente. Sugestões (outras poderão ser definidas) possíveis de penalizações a implementar:

- Reduzir a velocidade máxima para 70% durante alguns segundos.
- Adicionar um número limitado de segundos ao tempo total já passado.
- Alterar o funcionamento dos controlos (p.ex. inverter esquerda e direita).

NOTAS:

- A forma geométrica 3D de representação dos obstáculos é livre.
- Os carros autónomos não são influenciados por obstáculos.

3.5 Outdoor Display

Um *outdoor display* devidamente enquadrado no cenário 3D deve manter sempre visíveis vários elementos informativos, no mínimo:

- Tempo decorrido
- Número de voltas completadas
- Velocidade máxima, podendo esta ter aspecto diferente se:
  - aumentada por power-up
  - reduzida por obstáculo
  - reduzida por o carro estar fora da pista
- Tempo restante de benefício/penalização (decorrente de power-up/obstáculo)
- Estado em curso/pausa do jogo.

3.6 Cenário

A pista deverá ser incluída num cenário desenhado a critério dos autores, com elementos vários de paisagem, e que se sugere ser dotado de relevo.

Além da pista devem existir três parques de estacionamento de carros:

- Parque de concorrentes: local onde o jogador seleciona o carro com que irá concorrer.
- Parque de adversários: local onde o jogador seleciona o adversário com que irá concorrer.
- Parque de obstáculos: local onde o jogador seleciona um obstáculo a colocar em pista, de acordo com o especificado na secção 3.3.

Nota: a seleção dos elementos será realizada com a técnica de picking, descrita na secção 4.1. Na presente secção deverão apenas criar os elementos visuais para o cenário.

4. Técnicas a implementar

Resumem-se de seguida as principais técnicas a implementar neste trabalho.

4.1 Seleção de objetos / Picking

A seleção de objetos é uma facilidade importante das tecnologias de Computação Gráfica 3D. No jogo em causa, um objeto, depois de selecionado, é alvo de uma ação dependente do próprio objeto e do estado do jogo. Exemplos:

- Início da prova: o jogador seleciona o seu carro e o carro concorrente; ambos devem ser colocados no ponto de partida.
- Durante a prova, no processamento de um *power-up*:
  - São atribuídos os benefícios associados ao *power-up;*
  - O jogo fica em pausa;
  - O jogador seleciona um obstáculo do respetivo parque; deve selecionar seguidamente um ponto na pista; o obstáculo selecionado é colocado nesse ponto; o jogo prossegue;
- Objetos 3D são usados como dispositivos lógicos de interação, em alternativa a elementos da interface 2D do ThreeJS (embora esta última possa ser mantida, ver secção 5).

4.2 Animação por chaves

Técnicas de animação podem ser usadas em vários objetos do jogo. Uma técnica de animação em particular deve ser usada nos carros autónomos: animação por chave.

Genericamente, a animação por chaves passa pela definição prévia de um conjunto de instantes chave (muitas vezes conhecidos por imagens chave) e de, para cada um desses instantes, de um conjunto de propriedades (transformações geométricas ou outras). O software deve, nos intervalos de tempo entre chaves, efetuar uma interpolação (linear ou outra) dessas propriedades.

Uma rota (secção 3.1) constitui-se de um conjunto de instantes chave e respectivas transformações geométricas que efetuam a animação 3D de um carro adversário ao longo de uma volta à pista. Um fator de escala temporal (configurável) pode ser usado de forma a, com a mesma animação base, poderem obter-se diferentes tempos por volta.

Entre imagens chave, interpolação de transformações geométricas pode começar por ser linear, evoluindo posteriormente para uma interpolação cúbica.

4.3 Deteção de colisões

A deteção de colisões em aplicações de Computação Gráfica pode ser uma tarefa complexa, envolvendo, em alguns casos, um motor de física para a introdução de conceitos como a gravidade, objectos rígidos, etc. No entanto, para o efeito do jogo proposto, será usada uma versão simplificada para determinar se o carro se encontra ou não na pista e se colide com um obstáculo, com um power-up ou com o carro adversário.

4.3.1 Deteção de carro fora de pista

Para detetar se o carro está dentro ou fora de pista podem usar-se técnicas diferentes. A medida da distância do ponto pivot do carro (ponto situado no centro do eixo traseiro, segundo secção 3.2) é uma delas, mas pode ser algo ineficiente.

4.3.2 Deteção de colisão de carro com outros objetos

Para efeitos de deteção de colisão, cada objeto (carro, obstáculo ou *power-up*) deve ser dotado de um volume envolvente (*enclosing volume*) que, por simplificação, poderá ser uma esfera. A colisão entre dois objetos acontece quando a distância entre os seus centros é inferior ou igual à soma dos raios das esferas envolventes respetivas.

4.4 Texto 3D / Spritesheets

A afixação de texto no jogo pode ser efetuado por meio de bibliotecas de texto 3D associadas ao ThreeJS. No entanto, sem prejuízo dessa possibilidade e por se tratar de uma técnica muito usual em jogos de computador, deve implementar-se a técnica *Spritesheets* que se descreve de seguida.

- Uma *spritesheet* é uma textura que contém a representação visual de todos os caracteres possíveis de utilizar (ver exemplo da figura 2);
- A partir do código ASCII do caracter a desenhar, obter as coordenadas (***u, v***) de mapeamento na textura (eventualmente manipulando o valor de *offset*).

Figura 2: *Spritesheet* com “Oolite font” (16x16 caracteres)

4.5 Shaders

As placas gráficas atuais são programáveis em várias etapas diferentes sendo o *vertex shaders* e *fragment shaders* das etapas mais comuns de programação. Os *vertex shaders* atuam ao nível da geometria, por exemplo alterando coordenadas de vértices e das normais de uma superfície. Por outro lado, os *fragment shaders* atuam sobre a cor a atribuir a um fragmento (aproximadamente um *pixel*).
Os *shaders* constituem-se de pequenos processadores, de baixa complexidade (e por isso limitados quanto ao que podem fazer), mas que existem em grande número, funcionando em paralelo, em modo SISD (*Single Instruction Multiple Data*). Esta arquitetura atribui às placas gráficas um poder computacional assinalável que justifica a sigla GPU com duplo sentido (*Graphics Processor Unit* e *General Processor Unit*).

No presente trabalho os *shaders* devem ser usados, no mínimo, para:

- Dar um aspecto pulsatório à dimensão radial de um ou mais obstáculos: o diâmetro da forma cilíndrica utilizada varia no tempo, com frequência visível; para efeitos de deteção de colisões, deve ser considerado o diâmetro original do objeto.
- Realizar um display (*outdoor*) adicional cujas imagens (estáticas) sejam "tridimensionais", em baixo relevo:
    - Selecionar uma imagem RGB qualquer.
    - Obter, usando ferramentas adequadas, uma imagem LGray equivalente à anterior mas em níveis de cinzento, do tipo das obtidas do *depth buffer*: o nível de cinzento em cada pixel significa a distância entre a câmara hipotética e o objeto representado no pixel.
    - A imagem LGray deve ser usada pelos *vertex shaders* para deslocar pontos do objeto na direção da normal respetiva (pode ser necessária a utilização de um fator atenuador da distância, dado que se trata de um baixo relevo).
    - A imagem RGB deve ser usada pelos *fragment shaders* para "pintar" o baixo relevo anterior.

Em segunda prioridade, alterar o sistema anterior para:

- A cada minuto, adquirir a imagem RGB da cena, apresentada no *canvas* e capturada a partir da câmara ativa no programa.
- Simultaneamente, adquirir a imagem LGray, em níveis de cinzento, a partir do *depth-buffer* da câmara em uso.
- Proceder com estas duas imagens como anteriormente.

4.6 Sistema de partículas

Os sistemas de partículas são utilizados em Computação Gráfica para efetuarem a simulação visual de vários fenómenos. Constituem-se vulgarmente de um número assinalável de partículas, sendo cada partícula um elemento gráfico dotado de:

- Um momento de nascimento
- Um período de vida
- Um momento de desaparecimento
- Propriedades que podem variar ao longo do tempo de vida:
    - Cor
    - Transparência
    - Trajetória
    - …

No presente trabalho pretende-se simular um fogo de artifício através de um sistema de partículas:

- A partir de pontos aleatórios numa base retangular no chão, as partículas são lançadas para o ar em direções também aleatórias.
- Cada partícula deve efetuar uma trajetória balística, atendendo à aceleração da gravidade.
- Atingindo o fim de vida (com variação aleatória), a partícula "explode", sendo substituída por um conjunto pré-definido de partículas com um movimento radial a partir do ponto de explosão (aspeto visual de esfera crescente de partículas).

5. Interação

Podem manter-se numa interface 2D semelhante às dos trabalhos anteriores os controlos para ativar e desativar as fontes de luz, assim como para selecionar a vista/câmara ativa. Esta é uma interface técnica/complementar e não substitui a interface de jogo descrita seguidamente.

A interface do jogo pode incluir algumas das facilidades de interação 2D já conhecidas dos trabalhos anteriores. É no entanto incentivada a interação baseada nos elementos visuais de *input* e *output* no próprio cenário de jogo, ou seja, através de objetos 3D, alguns dos quais selecionáveis.

A técnica de seleção/*picking* deverá ser usada para identificar se e qual botão (elemento visual da interface) foi tocado/pressionado com o cursor do rato. Os botões podem ser visualizados como objetos simples (retângulos, cubos, cilindros…) com texturas.

Dependendo do botão tocado deverá ser executada a lógica correspondente, por exemplo:

- Quando o botão *Start* é tocado, o programa deve passar ao estado de “iniciar jogo” realizando as acções necessárias, tais como, por exemplo, colocar os carros na partida, iniciar geral de contador de tempo, vigiar por teclas premidas para movimentar o carro, etc.
- Quando o botão "Nivel de dificuldade 3" é tocado, deixa a posição de descanso e mantém-se pressionado (para baixo) indefinidamente até que outro nível de dificuldade seja selecionado.

Consideram-se três estados distintos de interface:

- o programa está a apresentar e tratar as opções relativas à situação inicial;
- o programa está a executar o jogo;
- o programa apresenta o resultado final.

5.1 Estado: Inicial

O menu inicial deve no mínimo mostrar botões e áreas de *display* para:

- Nome do jogo e dos autores, com alusão à FEUP
- Input (teclado) do nome do jogador
- Identificação do carro selecionado (null, no início)
- Identificação do carro adversário selecionado (null, no início)
- Nível de dificuldade, de entre duas ou mais alternativas (afeta o fator de escala temporal referido na secção 4.2); duas implementações alternativas são:
  - Usando um botão único em que a opção selecionada/mostrada vai mudando de cada vez que é pressionado
  - Usando vários botões para as diferentes opções, com feedback visual (o botão atualmente selecionado apresenta-se com um aspeto distinto)
- Botão *Start*, inicia o jogo se informação anterior está completa.

5.2 Estado: Jogo em execução

A interação durante o jogo resume-se essencialmente à vigilância de teclas pressionadas para controlo do carro (WSAD) e de teclas para funções adicionais (por exemplo "Esc" terminar o jogo, "Space" para pausa, etc.). Como output deve ser mantido um heads-up display conforme descrito na secção 3.5.
Devem ser previstas as funcionalidades necessárias para o processamento de saídas de pista e de colisões com outros objetos (secção 4.3).

5.3 Estado: Resultados finais

Neste estado devem apresentar-se em *display(s)* próprio(s), alguns resultados da prova:

- Nível de dificuldade utilizado
- Carros utilizados (próprio e adversário)

- Tempo total do jogador
- Tempo total do carro adversário
- Indicação de jogador ganhador (indicação festiva)
- Indicação de jogador perdedor

Simultaneamente deverão ser visíveis efeitos especiais "festivos", por exemplo baseados em sistemas de partículas (secção 4.6).

Botões complementares devem ainda possibilitar:
- Reiniciar outra prova,
- Voltar ao Estado Inicial

5.4 Elementos informativos

À responsabilidade criativa dos autores, devem ser apresentadas informações adicionais ao longo de toda a aplicação.

Notas sobre a avaliação do trabalho

A classificação máxima a atribuir a cada alínea corresponde a um desenvolvimento ótimo da mesma, no absoluto cumprimento com todas as funcionalidades enunciadas. Sem perda da criatividade desejada num trabalho deste tipo, **não serão contabilizados**, para efeitos de avaliação, **quaisquer outros desenvolvimentos** além dos que são pedidos, **como forma de compensar componentes em falta**.

No item **Criatividade da cena** será avaliada a forma como a **utilização de técnicas** definidas **neste trabalho e nos trabalhos anteriores** permitem um **maior grau de inovação, originalidade, complexidade e o cuidado visual do jogo**.

Os critérios a usar na avaliação e respectivos pesos são os seguintes:

Elementos de jogo (secção 3.0)	4.0
Técnicas a implementar (secção 4.0)	8.0
Interação (secção 5.0)	2.5
Criatividade da cena (inovação, originalidade, complexidade e cuidado visual)	3.0
Estruturação, legibilidade e comentários/documentação do código	2.5

De acordo com a formulação constante na ficha de disciplina, a avaliação deste trabalho conta para a classificação final com um peso de:

80% * 40% = 32%

Planeamento do Trabalho

Planeamento Semanal

- Semana 1 (início em 20/11/2023): pontos 3.1 a 3.6
- Semana 2 (início em 27/11/2023): pontos 4.1 e 4.2
- Semana 3 (início em 04/12/2023): pontos 4.3 a 4.6
- Semana 4 (início em 11/12/2023): pontos 5.1 a 5.4
- Restante tempo (até 3/01/2024): complet./, refinam./ e verificações finais
- Sessões a calendarizar (início em 03/01/2024); avaliação de grupo.

Entrega

- Via moodle (oportunamente serão dadas instruções)
- Data limite de entrega: 03/01/2024
