Exercício 18 – Dig Dug

Temática

- Estruturas de decisão
- Estruturas de repetição
- *Timers*
- *Arrays* bidimensionais

Na sua pasta de trabalho crie uma nova pasta com o nome “Exercicio_18”. Nesta pasta devem ser guardados os ficheiros desenvolvidos nesta aula.

Problema:

Pretende-se criar um jogo *online*, onde o jogador terá como objetivo apanhar diamantes que se encontram no subsolo. Para tal terá de escavar e desviar-se dos obstáculos.
O objetivo é recriar uma versão simplificada de clássicos como:

Digger: https://www.youtube.com/watch?v=M6V4AoafGbU
Dig Dug: https://www.youtube.com/watch?v=313csrAS8Pk

Com base na página “index.html”, e no ficheiro “jogo.js”, desenvolva o código javascript necessário para implementar as alíneas seguintes.

NOTA: os seguintes passos orientaram para uma possível solução para este jogo. No entanto, existem outras soluções igualmente válidas.

Consulte o resultado final em: http://labmm.clients.ua.pt/LM3/LM3-p/ex18/

PARTE 1

Nesta primeira versão do jogo vamos construir dinamicamente a área de jogo e controlar o movimento da personagem.

1. Desenvolver a função **iniciar** a qual deve:
    a. Esconder a div “score”;
    b. criar um evento “onclick” na div “splash” que irá executar a função **startgame**;

2. Desenvolver a função **startgame** na qual deve:
    a. Esconder a div “splash”;
    b. Mostrar a div “score”;

3. Criar o mapa de jogo:
    a. Para criar o mapa do jogo, recorrendo a *tiles, iremos* armazenar os seguintes valores numa estrutura adequada (12 linhas e 16 colunas):

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 1, 1, 1, 1

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

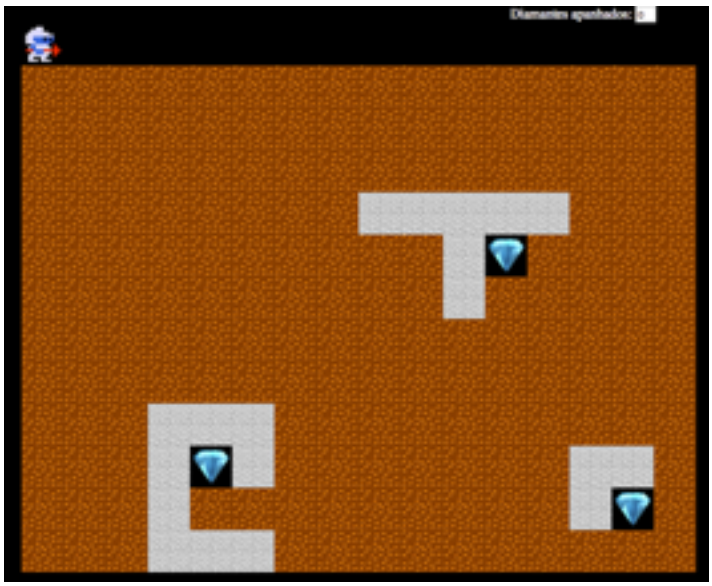
1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

1, 1, 1, 0, 2, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1

1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 1

1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

    b. invocar a função **`construirMapa`**;
    c. na função **`construirMapa`** e com base na estrutura do ponto a), deve ser adicionado ao conteúdo da div “tiles” uma imagem:
        i. “earth-tile.png” por cada 1;
        ii. “rock-tile.png” por cada 0;
        iii. “diamond.png” por cada 2;
    d. ainda na função **`construirMapa`**, adicione a imagem do jogador “player_right.png”, a -50px do topo e a 0px da esquerda;
    e. O resultado final deve estar de acordo com o seguinte exemplo:



4. Desenvolver a função **`processaTecla`** na qual deve:
    a. atualizar a imagem do jogador de acordo com o sentido do movimento horizontal efetuado (anim`_left.gif, anim_right.gif, anim_up.gif, anim_down.gif`);
    b. a função **`processaTecla`** deve ser invocada sempre que um evento de tecla pressionada é gerado.
    c. Restringir o movimento do jogador à área de jogo;
    d. calcular o movimento do jogador, 50 pixéis em cada movimento, através das setas do teclado e, utilizando os valores calculados (left e top), executar a função **detetaColisao**, passando-os como parâmetros;
    e. implemente a função detetaColisao, recebendo os dois parâmetros (*posLeft* e *posTop*) e movimentando o jogador para essas posições;

Antes de avançar para a parte seguinte garanta que o mapa e o movimento do jogador estão corretamente implementados.

PARTE 2

Nesta parte do exercício deve ser desenvolvido o código necessário para verificar a colisão do jogador com cada um dos *tiles*.
ATENÇÃO: A lógica desta verificação deve utilizar não a posição atual do jogador, mas sim verificar se a posição **para a qual** pretendemos mover o jogador é uma posição válida e efetuar a ação correspondente.

5. Na função **`detectaColisao`** deve:
    a. percorrer todos os tiles (com recurso à estrutura criada no ponto 3a)) e verificar se a posição de algum deles coincide com a todos valores passados como parâmetros;
    b. se coincidir, deve verificar o tipo de *tile* dessa posição e:
        i. se for do tipo 1, o jogador pode movimentar-se para essa posição e o *tile* deve ser escondido;
        ii. se for do tipo 0, o jogador **não** se pode movimentar para essa posição;
        iii. se for do tipo 2, o jogador pode movimentar-se para essa posição, o tile deve ser escondido e contar +1 nos diamantes apanhados;

PARTE 3

Nesta parte do exercício pretende-se efetuar alguns melhoramentos:

6. Desenvolva o código necessário para que o jogador não possa apanhar várias vezes o mesmo diamante, mesmo depois de este ter sido escondido;
7. Utilizando as imagens “player_down.png”, “player_up.png”, “player_left.png” e “player_right.png”, implemente uma solução que faça com que o jogador fique parado na sua posição correspondente, depois de ter sido movimentado;