Trabalho Prático

Trabalho realizado por:

Bruno Freitas Ferreira a13226

Márcio Rocha a13218

Carlos Costa a13673

Introdução:

A ideia do jogo a concretizar neste trabalho da disciplina foi um jogo de ritmo musical em que o jogador pudesse escolher a música que quisesse do seu telemóvel, externa ao jogo, e que o jogo gerasse um "mapa" para esta música com objetos para serem clicados ao ritmo da música, pois o jogo ao criar o mapa observa a onda de som desta, e, baseado na sua amplitude, detetaria os momentos de cada batida gerando o mapa.

Gerador do Mapa:

A criação dos mapas foi das partes que demorou mais tempo a concluir. Devido a incapacidade do sistema android não conseguir fazer certas coisas tivemos de ir buscar uma Framework externa. Depois de muita procura e várias tentativas, descobrimos uma Framework chamada TarsosDSP. Esta Framework não só nos deixava decodificar a musica como fazer tanta deteção de melodia como deteção de percussão, entre outros. Optamos por só usar a deteção de percussão pois e a que precisávamos para deter o "beat" da musica.

Devido ao decodificador que esta Framework usa tivemos de ser obrigados a diminuir a versão de android que estávamos a criar o jogo para, pois, o FFMPEG (o tal decodificador) não funciona com versões novas do android (mais especificamente 7.0) só funciona entre Kitkat 4.4 e Marshmallow 6.0.

Depois para enviar os dados desta framework para o libgdx e vice-versa, tivemos de usar interfaces, que explicámos mais á frente, pois também as utilizámos para a Firebase.

UI:

A ideia ao construir o UI foi mante-lo simples e compacto, mas também familiar ao famoso jogo "Osu". Grande parte dos seus elementos são imagens que formam botões, *backgrounds* ou mesmo imagens permitindo total customização sem a necessidade de skins.

Menu Inicial:

Menu com os 3 típicos botões de uma aplicação de telemóvel: *Play*, *Options* e *Exit*, sendo *Play* para prosseguir ao próximo menu representado em baixo e *Exit* para sair da aplicação,

Options de momento não tem qualquer função pois veio-se a chegar a conclusão que não era preciso de todo.

É neste menu onde se acede á *Firebase* partir do botão no canto superior direito, quando preto o utilizador encontra-se desconectado e quando verde conectado.

Menu Play:

Menu para selecionar a música a ser jogada, quando não existe musica selecionada o botao de *play* permanece cinzento como na imagem e preto quando há musica selecionada. Depois de escolhida a musica, carregar no botão com a seta para confirmar e começar o jogo.

Jogo:

Ecrã do jogo. Aqui vão aparecendo vários círculos aonde o jogador terá que carregar por ordem neles no momento da batida. Este momento é representado pela circunferência á volta do círculo, que ao longo do tempo, vai diminuindo de raio até ter o mesmo raio que o círculo, sendo esse o momento ideal para clicar, pois é o momento da batida.

No canto inferior esquerdo do ecrã também se pode ver a pontuação total do jogador e o multiplicador que representa o número de círculos clicados sucessivamente com sucesso, ou seja, o combo.

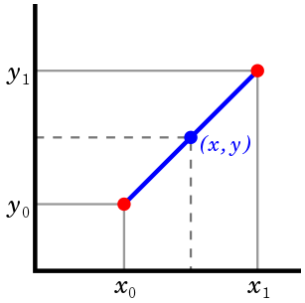
Os pontos recebidos dependem de quatro situações, o jogador não clica ou clica muito antes do tempo e não recebe pontos, quebrando o combo. Ele clica, mas é bastante antes do tempo e recebe 50. Clica ligeiramente antes ou depois do tempo e recebe 100. Clica perfeitamente ou muito próximo do tempo ideal e recebe 300 pontos, sendo este o máximo.

Quando os círculos são clicados ou morrem de não serem clicados após bastante tempo, estes deixam para trás na sua posição o valor de pontos que deram ao jogador. Neste caso pode-se ver o valor 0 onde já esteve um círculo.

Círculos:

Tamanho da auréola:

Esta linha circular, diminui o seu raio com o tempo para igualar-se ao raio do círculo. Este raio é calculado todos os *frames* do jogo e é feito através de interpolação linear. Ou seja, calcula-se uma função linear que ligue dois pontos num espaço bidimensional. Um ponto sendo o raio do círculo como y e o tempo que esse raio deve ser atingido como x e o outro ponto sendo o raio inicial da auréola e o tempo inicial 0.



$$y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0} = \frac{y_0(x_1 - x) + y_1(x - x_0)}{x_1 - x_0}$$

Com isto, concluímos os nossos cálculos para o que precisamos que é uma simples aplicação da fórmula acima:

```
haloSize = (circleSize * (initHaloTime - haloTime) + initHaloSize * (haloTime - 0)) / (initHaloTime - 0);
```

Deteção de input dentro do círculo:

Aqui a nossa lógica foi calcular a distância entre o centro do círculo e a posição do input do jogador. Depois de ter esta distância calcular a magnitude do vetor e verificar se é inferior ao raio. Se for, é porque está dentro do círculo, senão, está fora.

```
public boolean CheckIfClicked(int xInput, int yInput, double timeWasClicked, double diff)
{
    yInput = Gdx.graphics.getHeight() - yInput;
    if (Math.abs(Math.sqrt(Math.pow(x - xInput, 2) + Math.pow(y - yInput, 2))) < circleSize) {
        isActive = false;

        double playerTime = (clickTime + diff / 1.85) - timeWasClicked;
        if (playerTime <= 0.15 && playerTime >= -0.15)
            score = 300;
        else if (playerTime <= 0.3)
            score = 100;
        else if (playerTime <= 0.5)
            score = 50;
        else if (playerTime <= 0.7)
            score = 0;

        wasClicked = true;

        return true;
    }
    return false;
}
```

Aqui tivemos um pequeno problema, pois a coordenada de input no eixo do Y é o inverso do sistema de coordenadas do *libgdx*. Por isso, invertemos a coordenada como se pode ver em cima na primeira linha de código da função.

Também é verificado a diferença de tempo entre o tempo ideal e o tempo do input para atribuir um valor ao score.

Gerador de círculos:

Os círculos são gerados numa classe á parte, pois implementámos padrões de círculos onde em vez de ser sempre gerado um círculo por batida, existe uma percentagem onde irá gerar dois círculos, um em cima do outro, ou então três círculos numa forma triangular numa batida, com o objetivo de tornar os mapas mais desafiantes e divertidos.

Esta classe também ordena todos os círculos por cores e números para facilitar a leitura de um mapa pelo jogador, assim como a posição do próximo círculo a ser gerado, tendo em conta onde o último esteve posicionado, dando um certo "*flow*" ao mapa em vez de ser círculos em posições completamente aleatórias no ecrã. Também se certifica que o círculo está contido dentro do ecrã.

Esta posição é calculada criando um ponto a uma distância do ponto anterior e, depois, é rodada em torno do ponto anterior com um ângulo aleatório:

```
do
{
    double angle = random.nextInt( i: 360);
    pos.X = (int)(lastX + distance * (float)Math.cos(angle));
    pos.Y = (int)(lastY + distance * -(float)(Math.sin(angle)));
} while ((pos.X > Gdx.graphics.getWidth() - circleSize || pos.X <
```

Este cálculo é sempre repetido enquanto a posição calculada não estiver entre os limites do ecrã.

Guardar mapa:

Quando o jogador joga uma vez uma música pela primeira vez, o mapa gerado é guardado em ficheiro para que o jogador possa repetir o mapa e tentar melhorar o seu score. Isto é feito guardando a lista dos círculos, guardando para cada círculo o tempo em que aparece, o tamanho, a posição, a cor e o número da ordem.

```
writer.write( str: c.getTime() + "," +
        c.getX() + "," +
        c.getY() + "," +
        c.getColor().r + "," +
        c.getColor().g + "," +
        c.getColor().b + "," +
        c.getNum() + "\n");
```

Ao ler o ficheiro faz-se exatamente o mesmo, mas para criar novos círculos que vão ser exatamente iguais e adiciona-se á lista do mapa. Isto só acontece se for encontrado um ficheiro existente correspondente á música.

```
/*time,x,y,circleSize,color,num*/
Circle circle = new Circle(Double.parseDouble(circleInfo[0]),
        Integer.getInteger(circleInfo[1]),
        Integer.getInteger(circleInfo[2]),
        (int)(Gdx.graphics.getHeight() * 0.10f),
        new Color(Integer.getInteger(circleInfo[3]), Integer.getInteger(circleInfo[4]), Integer.getInteger(circleInfo[5]), a: 1),
        circleInfo[6]);
list.add(circle);
```

Classe do jogo:

Esta classe faz todos os cálculos para o score, input e percorre a lista de círculos para os atualizar. Ao percorrer, primeiro verifica se já chegou o tempo de o círculo ser desenhado, senão não faz nada a não ser ver se é um círculo que já morreu, para o apagar da lista. Caso seja tempo de ser desenhado, é atualizado e é verificado se todos estes círculos são tocados pelo jogador. Neste último caso, a lista é percorrida da ordem inversa aos outros casos, pois é necessário desenhar os primeiros da lista em último para ficarem por cima dos outros na ordem correta, mas é necessário correr a lista da forma habitual para detetar qual é tocado pelo jogador primeiro quando dois círculos se situam na mesma posição (um por cima do outro), fazendo-se break no ciclo para não chegar a testar o input com os restantes círculos.

Tivemos alguns problemas com isto, nomeadamente o facto de estarmos a percorrer uma lista que pode ser enorme dependendo da música e da sensibilidade do detetor de percussão, duas vezes, embora a segunda seja parcialmente por causa do break, mas, mesmo assim, reparámos em várias perdas de *fps* em vários momentos por causa do tamanho desta lista. Pensámos em tentar utilizar outra maneira qualquer de fazer isto para evitar este problema, mas simplesmente não é possível, ou pelo menos, não seria simples o suficiente para poder ser feito.

Firebase:

Para correr código que é próprio do android e não do libgdx, foi necessário utilizar interfaces. Com estas interfaces, conseguimos enviar dados para os aplicar noutro sítio, a Firebase.

Na Firebase, é guardado o utilizador quando é registado e o score total que o jogador vai obtendo à medida que vai jogando.

Conclusão:

Com este trabalho pudemos desenvolver as nossas capacidades ligadas à criação de projetos *Android Studio* e *libgdx*. E, apesar das dificuldades sentidas, especialmente na criação do *decoder* das músicas, conseguimos produzir o pretendido com sucesso. Pôde-se melhorar e aprender mais sobre como organizar e gerir as classes e métodos de forma a funcionarem de forma eficiente e organizada, aprender a trabalhar com *FireBase,* escrever em ficheiros e ter um *inside* de como funciona os jogos online de android.

Bilbiografia:

Framework de processamento de música em tempo real:

https://github.com/JorenSix/TarsosDSP

Github:

https://github.com/washinima/Uso