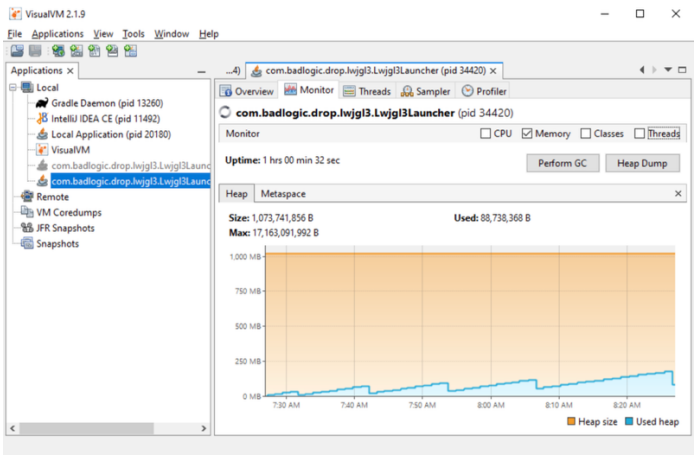
Jogos Digitais
Professor Lozano

Laboratório 05

Passo 01 – Essas gotas cairão da tela e nunca mais serão vistas. Mas o Java não esquece. Essas gotas permanecerão na memória para sempre. Se verificar o uso de memória, verá que temos um vazamento de memória.



Passo 02 – Se o jogador deixar o jogo ligado por muito tempo, ele travará. Portanto, devemos remover o sprite de queda da lista quando ele cair da tela. Precisamos fazer algumas modificações consideráveis no loop for dentro do método logic(). Apague seu loop no método logic:

```
//delete these lines
for (Sprite dropSprite : dropSprites) {
    dropSprite.setY(dropSprite.getY() - 2f * delta);
}
```

Passo 03 – Substitua-o pelo loop indicado abaixo:

```
private void logic() {  1 usage
    float worldWidth = viewport.getWorldWidth();
    float bucketWidth = bucketSprite.getWidth();
    bucketSprite.setX(MathUtils.clamp(bucketSprite.getX(),  min: 0,  max: worldWidth - bucketWidth));

    float delta = Gdx.graphics.getDeltaTime(); // retrieve the current delta

    // loop through each drop
    for (int i = dropSprites.size - 1; i >= 0; i--) {
        Sprite dropSprite = dropSprites.get(i); // Get the sprite from the list
        float dropWidth = dropSprite.getWidth();
        float dropHeight = dropSprite.getHeight();

        dropSprite.translateY( yAmount: -2f * delta);

        // if the top of the drop goes below the bottom of the view, remove it
        if (dropSprite.getY() < -dropHeight) dropSprites.removeIndex(i);
    }

    dropTimer += delta;
    if (dropTimer > 1f) {
        dropTimer = 0;
        createDroplet();
    }

}
```

O objetivo do laço é atualizar a posição das gotas e remover as que saíram da tela:

for (int i = dropSprites.size - 1; i >= 0; i--) { itera de trás pra frente na lista dropSprites. Fazer isso evita bugs ao remover itens durante a iteração, pois se você removesse indo pra frente, os índices mudariam e elementos poderiam ser pulados.

Sprite dropSprite = dropSprites.get(i); pega a gota (um Sprite) no índice atual.

float dropWidth = dropSprite.getWidth();

float dropHeight = dropSprite.getHeight(); lê a largura e a altura do sprite. Aqui só a altura é utilizada no teste de saída da tela; a largura foi lida mas, neste trecho, não é utilizada (poderia ser removida, porém podemos usar depois para colisão/centralização).

dropSprite.translateY(-2f * delta); Move a gota para baixo no eixo Y. translateY soma ao y atual; como é -2f * delta, o valor diminui. Delta é o tempo entre frames, então a velocidade é 2 unidades por segundo (suave em diferentes FPS).

if (dropSprite.getY() < -dropHeight) dropSprites.removeIndex(i); Esse trecho é o que garante que a gota desapareça da lista quando sai totalmente da tela.

O que significa dropSprite.getY()? O getY() do Sprite retorna a posição Y da base (parte de baixo) do sprite na tela, ou seja, é a coordenada vertical do "pé" da gota.

O que significa dropHeight? dropHeight = dropSprite.getHeight() é a altura do sprite, em pixels (ou unidades de mundo).

A lógica que acontece quando a gota começa, ela tem:

Topo: getY() + dropHeight

Base: getY()

A tela tem o zero no fundo (y=0). Queremos remover a gota quando o topo dela passar do fundo da tela.

Passo 04 – As gotas ainda não interagem com o balde. É aqui que incorporamos alguma detecção rudimentar de colisão. Isso pode ser feito com a classe Rectangle. Precisamos de dois retângulos para fazer comparações. Um para o balde e outro para ser reutilizado a cada gota. Vamos declarar dois retângulos:

```
public class Main implements ApplicationListener {  2 usages
    Texture bucketTexture;  2 usages
    SpriteBatch spriteBatch;  7 usages
    FitViewport viewport;  10 usages
    Sprite bucketSprite;  9 usages

    Texture backgroundTexture;  2 usages
    Vector2 touchPos;  4 usages
    Texture dropTexture;  2 usages

    Array<Sprite> dropSprites;  6 usages
    float dropTimer;  3 usages

    Rectangle bucketRectangle;  no usages
    Rectangle dropRectangle;  no usages
```

Passo 05 – Agora vamos gerar as instâncias no método create:

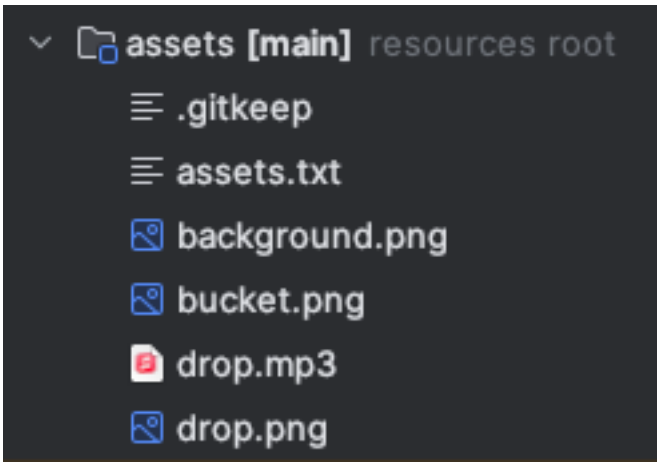
```
public void create() {
    bucketTexture = new Texture( internalPath: "bucket.png");
    backgroundTexture = new Texture( internalPath: "background.png");
    dropTexture = new Texture( internalPath: "drop.png");

    spriteBatch = new SpriteBatch();
    viewport = new FitViewport( worldWidth: 8,  worldHeight: 5);

    bucketSprite = new Sprite(bucketTexture);
    bucketSprite.setSize( width: 1,  height: 1);

    touchPos = new Vector2();

    dropSprites = new Array<>();

    bucketRectangle = new Rectangle();
    dropRectangle = new Rectangle();
}
```

Passo 06 – Vamos alterar algumas coisas no método logic para que o código defin os retângulos para a posição e as dimensões dos Sprites:

```
private void logic() {  1 usage
    float worldWidth = viewport.getWorldWidth();
    float bucketWidth = bucketSprite.getWidth();
    float bucketHeight = bucketSprite.getHeight(); //implementação do retangulo
    bucketSprite.setX(MathUtils.clamp(bucketSprite.getX(),  min: 0,  max: worldWidth - bucketWidth));

    float delta = Gdx.graphics.getDeltaTime();

    //implementação dos retangulos
    bucketRectangle.set(bucketSprite.getX(), bucketSprite.getY(), bucketWidth, bucketHeight);


    for (int i = dropSprites.size - 1; i >= 0; i--) {
        Sprite dropSprite = dropSprites.get(i);
        float dropWidth = dropSprite.getWidth();
        float dropHeight = dropSprite.getHeight();

        dropSprite.translateY( yAmount: -2f * delta);
        //implementação do retangulo
        dropRectangle.set(dropSprite.getX(), dropSprite.getY(), dropWidth, dropHeight);

        if (dropSprite.getY() < -dropHeight) dropSprites.removeIndex(i);
        else if (bucketRectangle.overlaps(dropRectangle)) { // implementação do retangulo
            dropSprites.removeIndex(i);
        }
    }

    dropTimer += delta;
    if (dropTimer > 1f) {
        dropTimer = 0;
        createDroplet();
    }

}
```

O bucketRectangle.overlaps(dropRectangle) verifica se o balde se sobrepõe ao drop. Se isso acontecer, o Sprite será removido da lista de Sprites dropados. Isso significa que ele não será mais desenhado ou usado. Ele simplesmente não existe mais, fazendo parecer que o balde o coletou. Certifique-se de que seu jogo funcione conforme o esperado.

Passo 07 – É muito fácil adicionar uma linha para reproduzir um efeito sonoro agora que chegamos ao final do nosso fluxo de trabalho. Queremos que o som da gota seja reproduzido quando o balde colidir com a gota. Ele não deve ser reproduzido quando a gota sair da tela. Baixe o arquivo drop.mp3 e arraste o mesmo para a pasta assets do projeto.



Passo 08 – Agora crie uma variável do tipo Sound:

```
public class Main implements ApplicationListener {  2 usages
    Texture bucketTexture;  2 usages
    SpriteBatch spriteBatch;  7 usages
    FitViewport viewport;  10 usages
    Sprite bucketSprite;  12 usages

    Texture backgroundTexture;  2 usages
    Vector2 touchPos;  4 usages
    Texture dropTexture;  2 usages

    Array<Sprite> dropSprites;  7 usages
    float dropTimer;  3 usages

    Rectangle bucketRectangle;  3 usages
    Rectangle dropRectangle;  3 usages

    Sound dropSound;  no usages
```

Passo 09 – Os objetos do tipo Sound servem para lidar com o arquivo de áudio, como por exemplo a gota de chuva em nosso projeto. Os sons são carregados completamente na memória para que possam ser reproduzidos de forma rápida e repetida. Vamos agora instanciar a classe Sound passando como parâmetro o nome do arquivo do som da gota, dentro do método create().

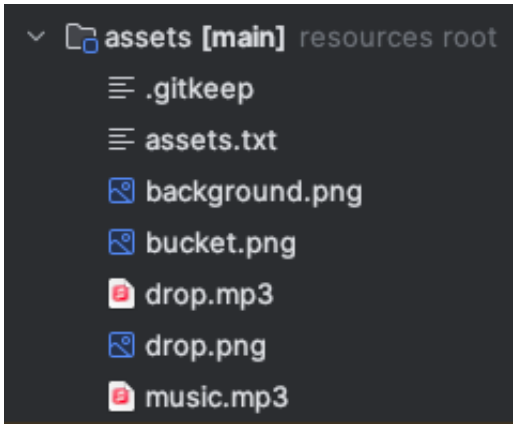
```
public void create() {
    bucketTexture = new Texture( internalPath: "bucket.png");
    backgroundTexture = new Texture( internalPath: "background.png");
    dropTexture = new Texture( internalPath: "drop.png");

    spriteBatch = new SpriteBatch();
    viewport = new FitViewport( worldWidth: 8,  worldHeight: 5);

    bucketSprite = new Sprite(bucketTexture);
    bucketSprite.setSize( width: 1,  height: 1);

    touchPos = new Vector2();

    dropSprites = new Array<>();

    bucketRectangle = new Rectangle();
    dropRectangle = new Rectangle();

    dropSound = Gdx.audio.newSound(Gdx.files.internal( s: "drop.mp3"));
}
```

Passo 10 – No método logic(), logo após a detecção da colisão entre o Retângulo do balde com a gota (no local onde definimos que quando houver essa colisão a gota some), vamos inserir uma chamada ao método play().

```
if (dropSprite.getY() < -dropHeight) dropSprites.removeIndex(i);
else if (bucketRectangle.overlaps(dropRectangle)) { // implementação do retangulo
    dropSprites.removeIndex(i);
    dropSound.play();
}
```

Agora, ao testar o jogo, você vai ver que quando há colisão o som é emitido.

Passo 11 – Vamos agora inserir uma música em nosso jogo. Para isso, arraste o arquivo music.mp3 para a pasta assets.



Passo 12 – Agora crie uma variável do tipo Music. A música, diferente do Som, é muito grande para ser mantida inteiramente na memória. Ela é transmitida do arquivo em partes. Não há uma regra precisa sobre o que deve ou não ser um som versus uma música, mas você pode considerar qualquer áudio com menos de 10 segundos como um som.

```
public class Main implements ApplicationListener {  2 usages
    Texture bucketTexture;  2 usages
    SpriteBatch spriteBatch;  7 usages
    FitViewport viewport;  10 usages
    Sprite bucketSprite;  12 usages

    Texture backgroundTexture;  2 usages
    Vector2 touchPos;  4 usages
    Texture dropTexture;  2 usages

    Array<Sprite> dropSprites;  7 usages
    float dropTimer;  3 usages

    Rectangle bucketRectangle;  3 usages
    Rectangle dropRectangle;  3 usages

    Sound dropSound;  2 usages
    Music music;  no usages
```

Passo 13 – No método create(), gere a instância da classe Music:

```
public void create() {
    bucketTexture = new Texture( internalPath: "bucket.png");
    backgroundTexture = new Texture( internalPath: "background.png");
    dropTexture = new Texture( internalPath: "drop.png");

    spriteBatch = new SpriteBatch();
    viewport = new FitViewport( worldWidth: 8,  worldHeight: 5);

    bucketSprite = new Sprite(bucketTexture);
    bucketSprite.setSize( width: 1,  height: 1);

    touchPos = new Vector2();

    dropSprites = new Array<>();

    bucketRectangle = new Rectangle();
    dropRectangle = new Rectangle();

    dropSound = Gdx.audio.newSound(Gdx.files.internal( s: "drop.mp3"));
    music = Gdx.audio.newMusic(Gdx.files.internal( s: "music.mp3"));
}
```

Passo 14 – A música deve tocar no início do jogo. Ela precisa ficar em loop contínuo até que o jogador termine de jogar. O arquivo também está um pouco alto, então deve tocar na metade do volume. O volume é um valor de 0f a 1f, sendo 1f o volume normal do arquivo. No final do método create() insira o seguinte trecho:

```
music.setLooping(true);
music.setVolume(.5f);
music.play();
```

Continuando o jogo da nave que você iniciou na última aula, faça as seguintes funcionalidades:

- Faça com que ao sua nave encostar em algum objeto criado o objeto suma da tela.
- Insira uma musica e sons de colisões.