

INSTITUTO FEDERAL
Rio Grande do Sul
Campus Farroupilha

DESENVOLVIMENTO DE GAMES VIRTUAIS EDUCATIVOS EM LIBGDX

Gabriel Lovato Vianna
Rafael Vieira Coelho
Cassiano José Sobierai

PENSE 2022

Clique [aqui](#) para baixar



O que é o libGDX?

Um framework open-source

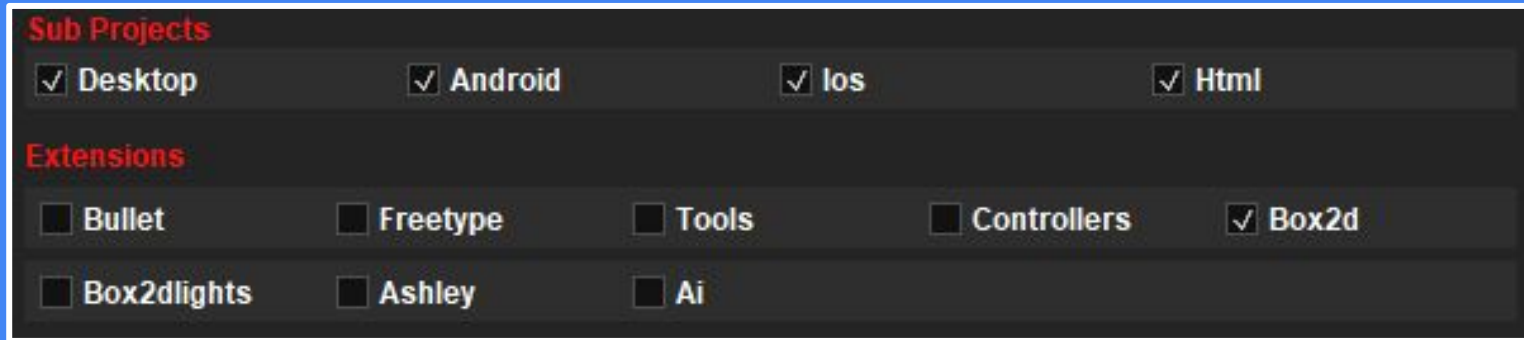
Configurando o projeto



The screenshot shows the 'Libgdx Project Generator' window. At the top, the title bar reads 'Libgdx Project Generator'. Below the title bar, the 'libGDX' logo is displayed in a large, stylized font, with 'lib' in white and 'GDX' in red. Underneath the logo, the text 'PROJECT SETUP' is visible. The main area of the window contains five configuration fields, each with a label on the left and a text input field on the right. To the right of each input field, there is a black box with white text providing a description of the field. The fields are: 'Name' with 'my-gdx-game' and description 'Nome do jogo'; 'Package' with 'com.mygdx.game' and description 'Nome da pasta das sources'; 'Game class' with 'MyGdxGame' and description 'Nome da classe principal do jogo'; 'Destination' with 'C:\Windows\system32\test' and description 'Pasta onde o projeto será salvo'; and 'Android SDK' with 'C:\Path\To\Your\Sdk' and description 'Kit para dev de jogos para Android'.

Name:	my-gdx-game	Nome do jogo
Package:	com.mygdx.game	Nome da pasta das sources
Game class:	MyGdxGame	Nome da classe principal do jogo
Destination:	C:\Windows\system32\test	Pasta onde o projeto será salvo
Android SDK	C:\Path\To\Your\Sdk	Kit para dev de jogos para Android

Configurando o projeto



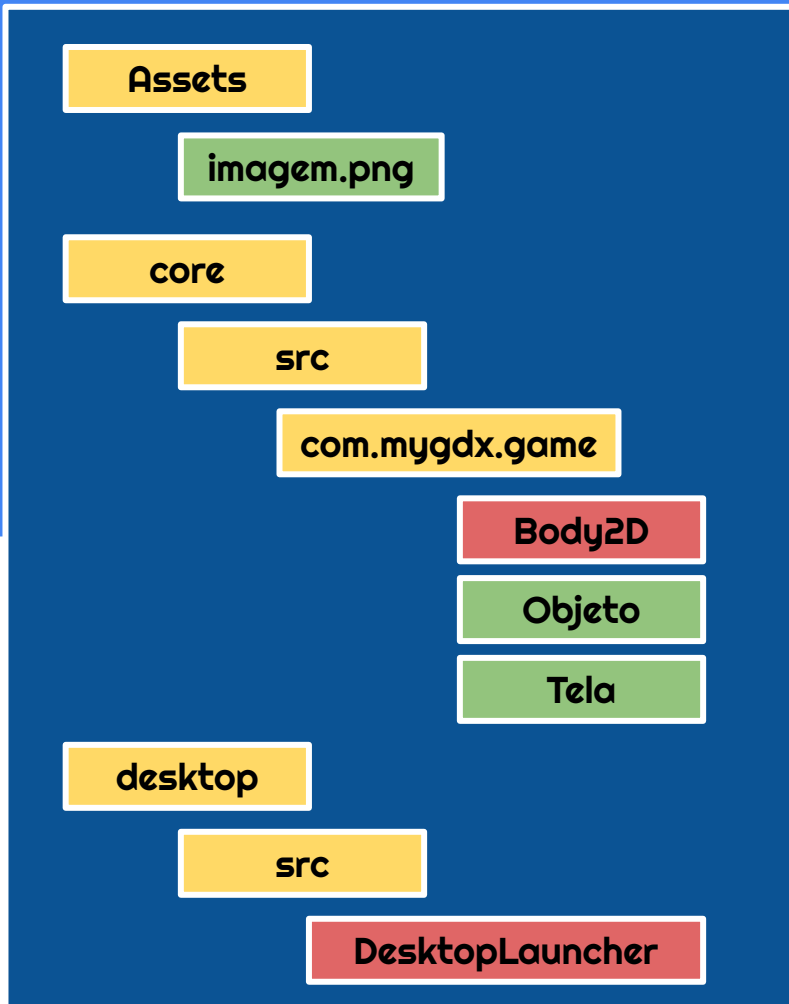
Sub Projects: quais plataformas o projeto será desenvolvido;

- 1. Desktop: Para computadores. O qual será usado neste curso;**
- 2. Android: Para smartphones com android;**
- 3. Ios: Para smartphones com ios;**
- 4. Html: Para ambientes WEB.**

Configurando o projeto

Extensions: quais extensões do libGDX o projeto utilizará;

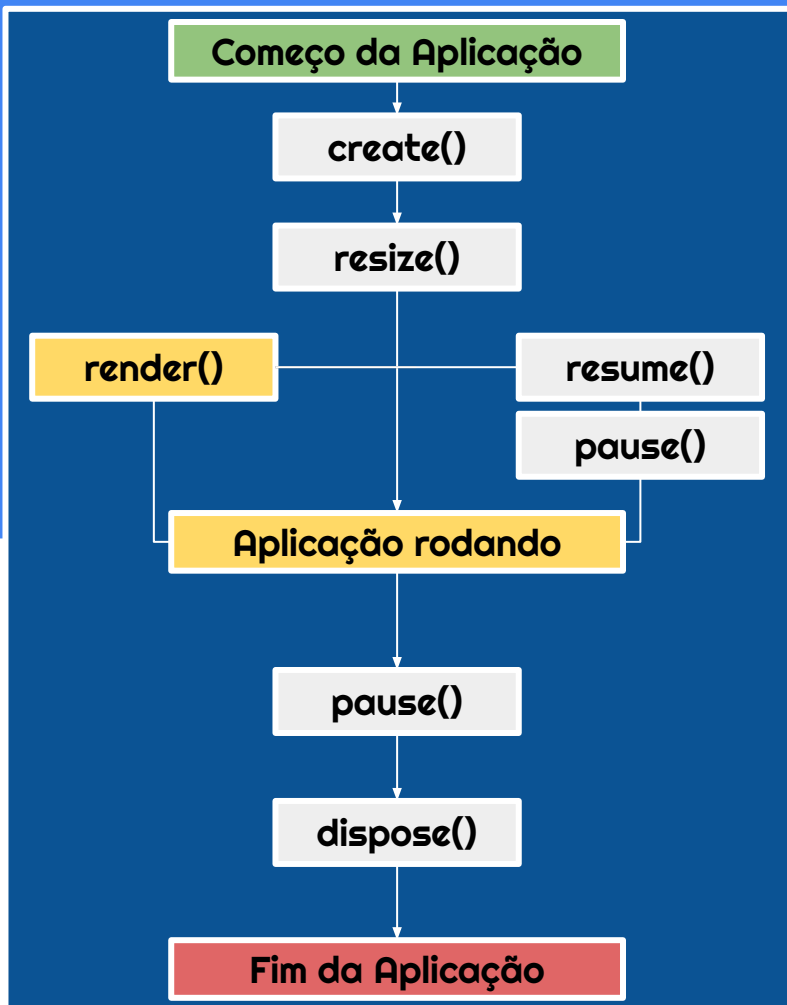
- 1. Bullet:** Uma biblioteca que detecta colisões 3D e possui uma mecânica de bodys.
- 2. FreeType:** Manipulador de tamanho de fontes;
- 3. Tools:** Inclui editor de partículas (2d/3d): matriz de fontes e pacotes de texturas e imagens;
- 4. Controller:** Uma biblioteca que lida com controles (Xbox 360);
- 5. Box2d:** Uma biblioteca com físicas 2D;
- 6. Box2dlights:** Usa a biblioteca box2D para tratar a imagem e o OpenGL ES 2.0 para renderizar;
- 7. Ashley:** Um framework pequeno de entidades;
- 8. Ai:** Um framework de inteligência artificial.



PASTAS/ARQUIVOS

O libGDX já organiza algumas pastas e arquivos para o desenvolvedor, na aula de hoje criaremos mais alguns arquivos.

ESTRUTURA



LOOP

Quando o projeto for criado, apenas haverá um arquivo no core e ele começará com poucas funções e todas elas seguirão em forma de loop, dependendo, da herança e interface, devemos adicionar mais funções para fazer parte do loop.

ESTRUTURA

Começando o projeto

Usando (config.XXX) podemos atribuir algumas características a classe do jogo

```
public class DesktopLauncher {  
    public static void main (String[] arg) {  
        Lwjgl3ApplicationConfiguration config = new Lwjgl3ApplicationConfiguration();  
        config.setForegroundFPS(60);  
        config.setTitle("Aula_1");  
        new Lwjgl3Application(new Body2D(), config);  
    }  
}
```

Função isolada que dá início ao projeto
Cria e configura o objeto do jogo
É acessada apenas uma vez

Physics1



GRAVIDADE

Com a extensão Box2d do libGDX tem-se acesso a uma gravidade pronta, porém podemos criar uma gravidade do zero com algumas linhas de código.

Exemplo gravidade sem libGDX:

- `posição.y += 18 - acumY;`
- `acumY += 0.4;`

BOX2D

2 usages

```
public class Objeto extends ApplicationAdapter {
```

1 usage

```
SpriteBatch batch;
```

11 usages

```
Sprite sprite;
```

2 usages

```
Texture img;
```

13 usages

```
Body body;
```

1 usage

```
public Objeto(Tela screen){  
    create(screen);  
}
```

Declarando as variáveis

CORPO/BODY

Com a extensão Box2d do libGDX pode-se criar um corpo com gravidade e colisões. Existem algumas maneiras de implementá-la.

Iniciando o construtor

BOX2D

1 usage

```
public void create(Tela screen) {  
  
    batch = new SpriteBatch();  
  
    img = new Texture(internalPath: "ball.png");  
    sprite = new Sprite(img);  
  
    sprite.setPosition(x: 400, y: 200);  
  
    BodyDef bodyDef = new BodyDef();  
    bodyDef.position.set(sprite.getX(), sprite.getY());  
    bodyDef.type = BodyDef.BodyType.DynamicBody;  
    body = screen.getWorld().createBody(bodyDef);  
}
```

Iniciando as variáveis

BOX2D

```
PolygonShape shape = new PolygonShape();  
shape.setRadius(30);  
  
FixtureDef fixtureDef = new FixtureDef();  
fixtureDef.shape = shape;  
  
body.createFixture(fixtureDef).setUserData(this);  
  
}
```

Iniciando as variáveis

BOX2D

3 usages

```
public class Tela extends InputAdapter implements Screen {
```

4 usages

```
private World world;
```

8 usages

```
private Objeto obj;
```

4 usages

```
private SpriteBatch batch;
```

1 usage

```
public Tela(Body2D body2D) {
```

```
    world = new World(new Vector2(x: 0, y: -100f), doSleep: true);
```

```
    obj = new Objeto(screen: this);
```

```
    batch = new SpriteBatch();
```

```
}
```

Função da tela principal

Iniciando as variáveis

BOX2D

Atualizando e renderizando o jogo

```
@Override
public void render(float delta) {

    world.step( timeStep: 1 / 60f, velocityIterations: 6, positionIterations: 2);


    obj.sprite.setPosition( x: obj.body.getPosition().x - obj.sprite.getWidth() / 2,
                           y: obj.body.getPosition().y - obj.sprite.getHeight() / 2);

    batch.begin();

    obj.render(batch);

    batch.end();

}
```



1 usage

```
public void render(SpriteBatch batch) {

    batch.draw(sprite, sprite.getX(), sprite.getY());

}
```

BOX2D

INPUT

Através do libGDX, pode-se saber se algo foi ou está pressionado e usar isso para operações, como a movimentação de um objeto(jogador).

Exemplos:

- `Gdx.input.isKeyPressed(Input.Keys.TECLA)` - verifica se alguma tecla está sendo pressionada, retorna um boolean;
- `Gdx.input.getX()` ou `Gdx.input.getY()` - retorna a posição do mouse na coordenada X ou Y;
- `Gdx.input.isButtonPressed(Input.Buttons.BOTÃO);` - verifica se o algum botão do mouse está sendo pressionado, retorna um boolean.

Listas de botões e teclas:

- Botões;
- Teclas.

SONS E MÚSICAS

Para carregar o arquivo de som devemos declará-lo desta forma.

- **`Sound sound = Gdx.audio.newSound(Gdx.files.internal("data/mysound.mp3"));`**

Após declarado existem várias maneiras de trabalhar com o mesmo.

- **`long id = sound.play(1.0f);`** // Começa o som em um long para manipulação
- **`sound.stop(id);`** // Para o som imediatamente
- **`sound.setPitch(id, 2);`** // Seta a velocidade do som para 2x
- **`sound.setPan(id, -1, 1);`** // Seta o volume do lado esquerdo em 100%
- **`sound.setLooping(id, true);`** // Deixa o som em looping
- **`sound.stop(id);`** // Para o som

Em algumas aplicações, para deixarmos o jogo mais atraente devemos adicionar sons e músicas, o libGDX fornece várias ferramentas para mexer com isso.

Clique [aqui](#) para baixar código inicial

ATIVIDADE: CRIANDO UM CORPO

Criar um corpo com forma circular, através do Body2D que tenha gravidade, se movimente e a cada movimento produza um som.

Não esqueça de usar as funções de input e sound. Para fazer o boneco se movimentar mude a velocidade do corpo, chamando uma função!

Revisão: Aula 1

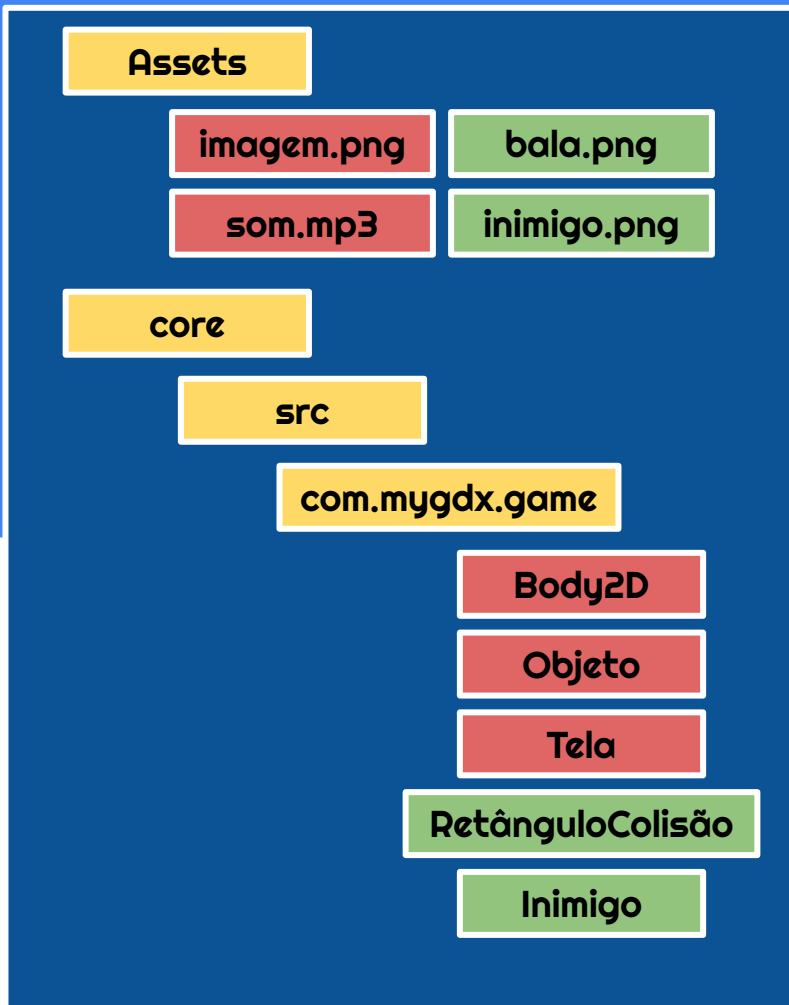
Usar a função applyLinearImpulse para movimentar o Body

```
public class DesktopLauncher {  
    public static void main (String[] arg) {  
        Lwjgl3ApplicationConfiguration config = new Lwjgl3ApplicationConfiguration();  
        config.setForegroundFPS(60);  
        config.setTitle("Aula_1");  
        new Lwjgl3Application(new Body2D(), config);  
    }  
}
```

Conferir a proporção do sprite e o Body

Mudar a posição do Sprite em relação ao Body e não o contrario

Criamos um Body
Aprendemos a movimentar um Body
Manipulamos arquivos de som

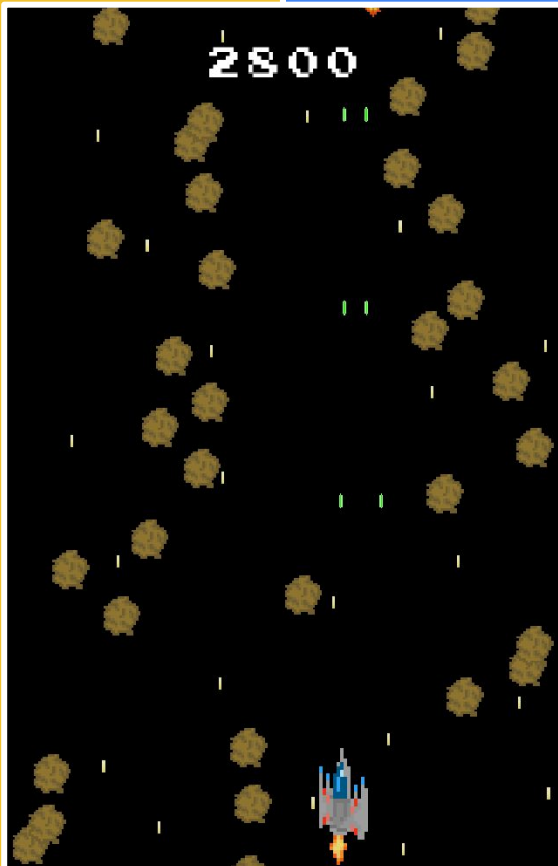


PASTAS/ARQUIVOS

Na última aula criamos 3 arquivos (imagem.png, Objeto.java, Tela.java), na aula de hoje criaremos mais 4 arquivos.

ESTRUTURA

BULLET: TIROS DENTRO DO JOGO



Para deixar os jogos mais interessantes e mecânicos, existem ferramentas dentro do libGDX que cria corpos que possivelmente colidem e geram consequências para o jogador.

7 usages

```
public class Bala {
```

1 usage

```
public static final int VELOCIDADE = 500;
```

1 usage

```
public static final int LARGURA = 3;
```

1 usage

```
public static final int ALTURA = 12;
```

3 usages

```
private static Texture TEXTURA;
```

3 usages

```
float x, y;
```

3 usages

```
RetanguloColisao retangulo;
```

2 usages

```
public boolean remove = false;
```

Declarando as variáveis

BULLET

1 usage

```
public Bala (float x, float y) {  
    this.x = x;  
    this.y = y;  
    this.retangulo = new RetanguloColisao(x, y, LARGURA, ALTURA);  
  
    if (TEXTURA == null)  
        TEXTURA = new Texture( internalPath: "bullet.png");  
}
```

Iniciando o construtor

1 usage

```
public void update (float deltaTime) {  
    y += VELOCIDADE * deltaTime;  
    if (y > Gdx.graphics.getHeight())  
        remove = true;  
  
    retangulo.mover(x, y);  
}
```

Atualização das balas

BULLET

1 usage

```
public void render (SpriteBatch batch) {  
    batch.draw(TEXTURA, x, y);  
}
```

Renderizando a bala

```
public RetanguloColisao getColisao () {  
    return retangulo;  
}
```

Função para usar na colisão

```
}
```

BULLET

Arquivo RetanguloColisão

```
public class RetanguloColisao {
```

6 usages

```
float x, y;
```

3 usages

```
int width, height;
```

2 usages

```
public RetanguloColisao(float x, float y, int width, int height) {
```

```
    this.x = x;
```

```
    this.y = y;
```

```
    this.width = width;
```

```
    this.height = height;
```

```
}
```

2 usages

```
public void mover (float x, float y) {
```

```
    this.x = x;
```

```
    this.y = y;
```

```
}
```

Declarando as variáveis

Atualizando o retângulo

BULLET

Parte do código na função render do arquivo da tela

```
ArrayList<Bala> balasParaRemover = new ArrayList<>();  
for (Bala bala : obj.balas) {  
    bala.update(delta);  
    if (bala.remove)  
        balasParaRemover.add(bala);  
}  
  
obj.balas.removeAll(balasParaRemover);  
  
batch.begin();  
  
for (Bala bala : obj.balas) {  
    bala.render(batch);  
}  
  
batch.end();
```

Atualizando as balas

**Chamando a função para
renderizar as balas**

BULLET

INIMIGOS

Um jogo deve ter adversários para os jogadores e esse é o nosso próximo tópico. Para criar um inimigo apenas deve se criar um Body, seguindo a mesma lógica do personagem principal, cria-se um Body que recebe uma forma e pode ser desenhado a partir de um Sprite.

inimigo.java



criar o Body

tela.java



atribuir
sprite body

Clique [aqui](#) para saber como criar um body

Clique [aqui](#) para baixar o código inicial

ATIVIDADE: CRIANDO TIROS

Para esta atividade. Não esqueça de primeiro atualizar o Body e depois o Sprite e tente ao máximo utilizar as funções prontas do libGDX.

Fazer que o Body criado na aula passada, atire na horizontal, a bala deve sumir após alguma coordenada. Além de criar um inimigo que quando atingido aconteça algo com o mesmo.

tela.java

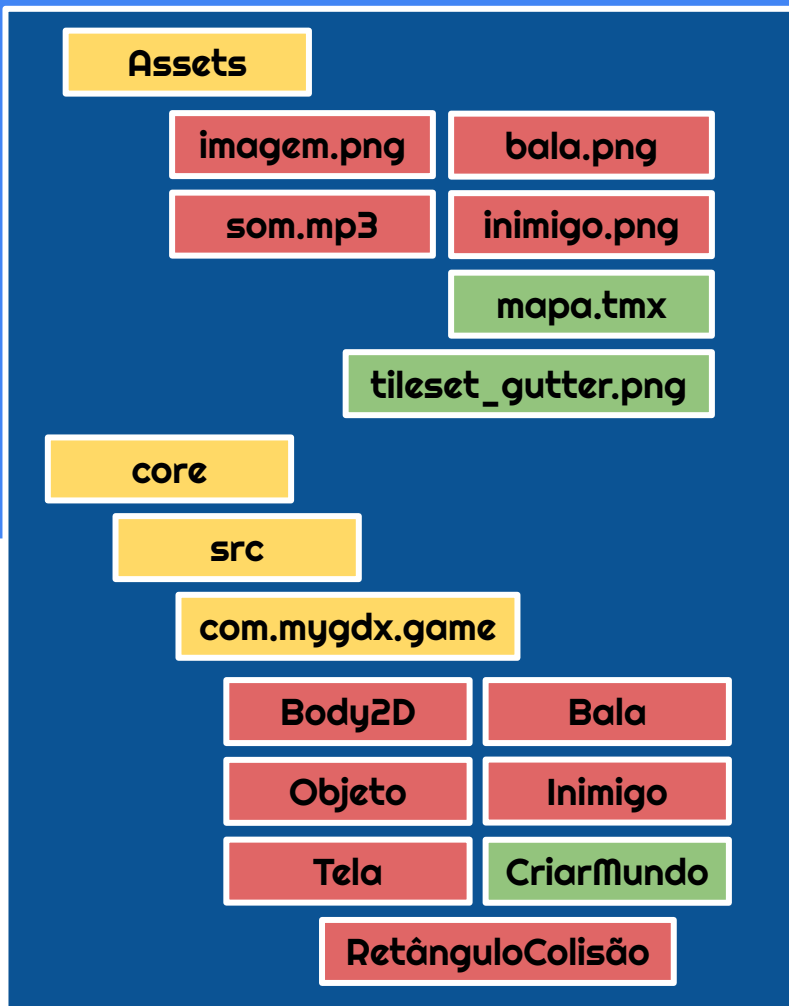


verificar
colisão

tela.java



atribuir
consequência



PASTAS/ARQUIVOS

Na última aula criamos 5 arquivos (bala.png, inimigo.png, bala.java, inimigo.java, RetanguloColisao.java), na aula de hoje criaremos mais alguns arquivos.

ESTRUTURA

Clique [aqui](#) para baixar o Tiled.

TILED: CRIANDO O MAPA DO JOGO

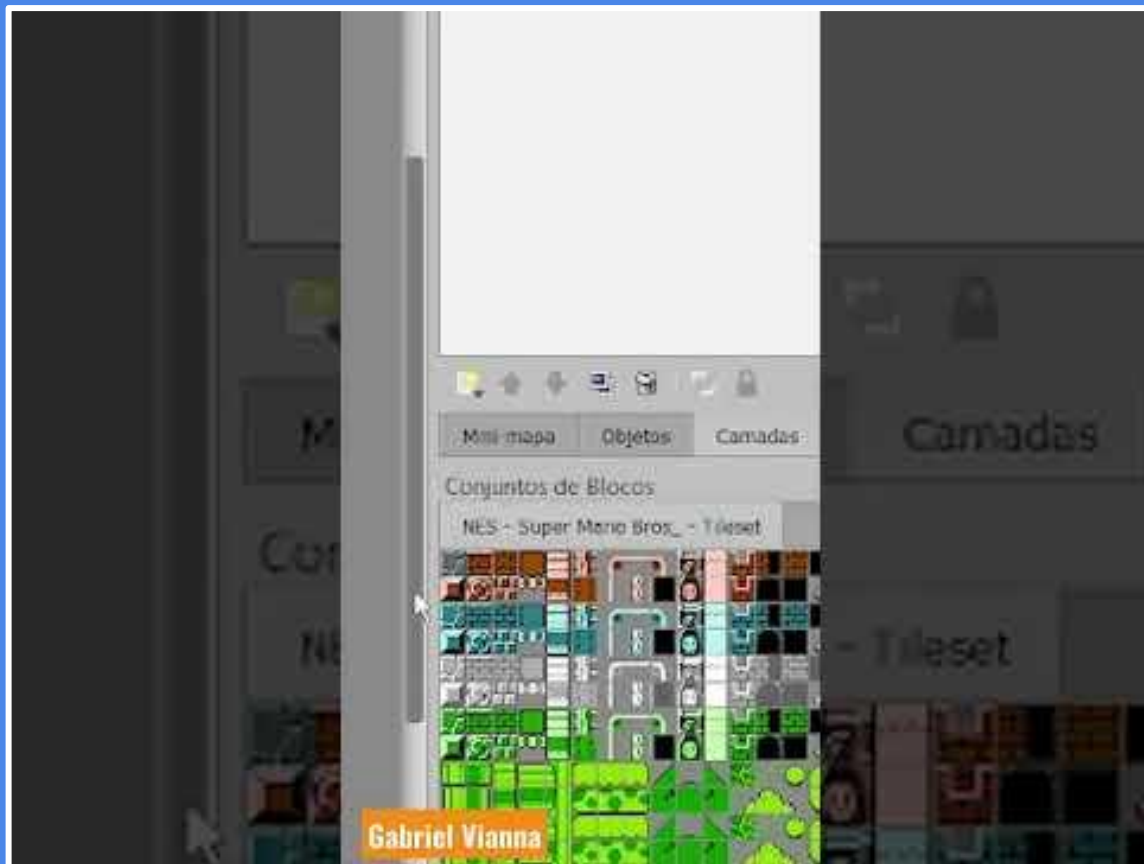
Tiled cria uma matriz com o mapa, dividindo o mesmo em vários quadrados, que devem ser preenchidos com camadas.

Para usar o design, imagens devem ser criadas e adicionadas dentro do Tiled.

Para criar o jogo usaremos dois tipos de camadas.

- **Camada de tile:** Camada onde será desenhado o jogo, uma camada estática.
- **Camada de objeto:** Camada onde serão criadas as colisões, ela é invisível e pode gerar uma reação quando um objeto do jogo entra em contato.

Clique [aqui](#) para baixar o Tiled.



TILED: VÍDEO PARA ENSINO

TILED: DENTRO DO INTELLIJ

Declarando as variáveis;

- **private TmxMapLoader CarregarMapa;**
- **private TiledMap mapa;**
- **private OrthogonalTiledMapRenderer carregador;**

Carregando o mapa no construtor;

- **CarregarMapa = new TmxMapLoader();**
- **mapa = CarregarMapa .load("NOMEDOARQUIVO.tmx");**
- **carregador = new OrthogonalTiledMapRenderer(mapa);**

Renderizando o mapa:

- **carregador.render();**

Clique [aqui](#) para baixar o Tiled.

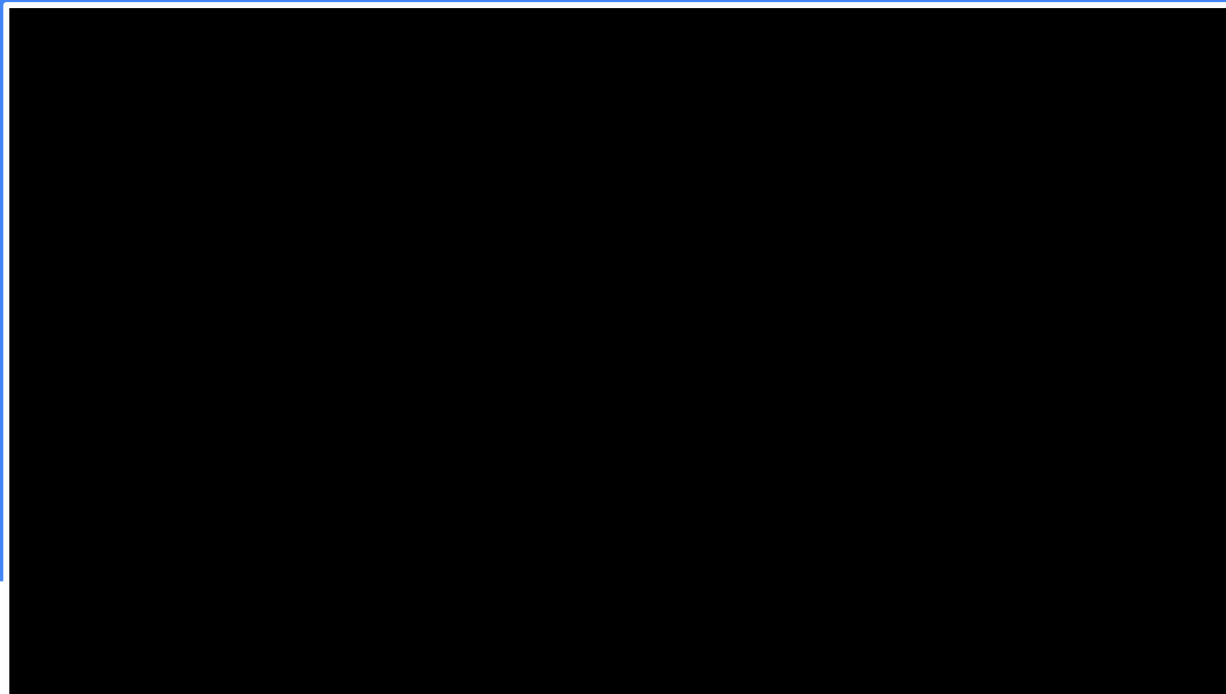
ATIVIDADE: CRIE UM MAPA

Com todo aprendizado adquirido até aqui, você deve usar o seu último código como base, para fazer esse. Adicione um mapa, sem colisões e tente criar uma consequência para quando o personagem encostar no inimigo.

- 1. Não esqueça de declarar todas as variáveis;**
- 2. Lembre-se do loop que o libGDX percorre;**
- 3. Descubra qual é o erro na colisão do personagem do código cedido e resolva-o.**

Câmera: centralizando o objeto

**Para seguirmos o boneco, precisamos
essencialmente de uma câmera**



Parte do código no construtor do arquivo principal

```
//Cria a camera
```

```
gamecam = new OrthographicCamera();
```

```
//Cria um Viewport para mandar a proporção da tela
```

```
gamePort = new FitViewport(Gdx.graphics.getWidth(), Gdx.graphics.getHeight(), gamecam);
```

```
gamecam.position.x = obj.sprite.getX();
```

```
gamecam.position.y = obj.sprite.getY();
```

Declarando a câmera

CÂMERA

Parte do código na função update do arquivo principal

```
gamecam.position.x = obj.sprite.getX();
```

```
//atualiza o objeto da camera
```

```
gamecam.update();
```

```
//diz para renderizar apenas o que está dentro da visão da camera
```

```
renderer.setView(gamecam);
```

**Atualizando a
posição da câmera**

CÂMERA

Parte do código na função render do arquivo principal

```
update(delta);
```

```
b2dr.render(world, gamecam.combined);
```

```
batch.setProjectionMatrix(gamecam.combined);
```

**Configuração da
câmera**

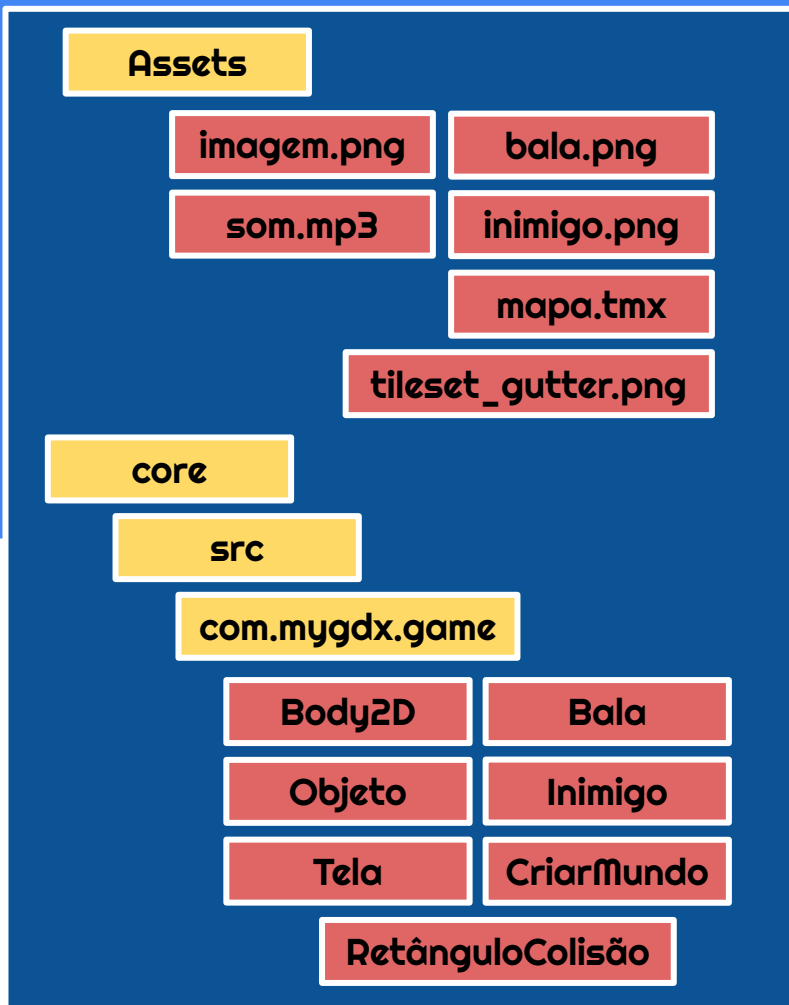
CÂMERA

Clique [aqui](#) para baixar o código inicial.

ATIVIDADE: CRIE UMA CÂMERA

Utilize o código disponibilizado para criar uma câmera que siga o nosso personagem principal e deixe-o no centro da tela.

- 1. Não esqueça de declarar todas as variáveis;**
- 2. Lembre-se do loop que o libGDX percorre;**
- 3. Não se preocupe com o nosso inimigo a ideia é apenas configurar a câmera.**

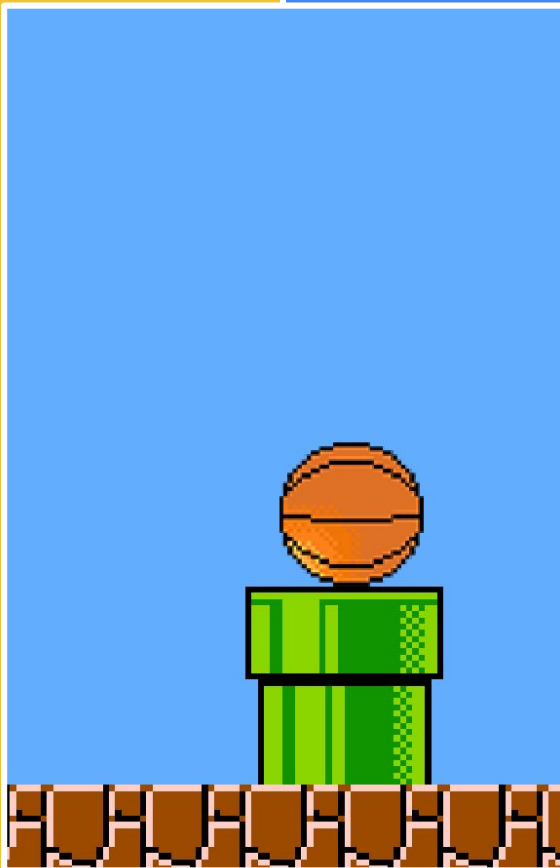


PASTAS/ARQUIVOS

Na última aula criamos 3 arquivos (`tileset_gutter.png`, `CriarMundo.java`, `mapa.tmx`), na aula de hoje não criaremos nenhum, apenas faremos mudanças.

ESTRUTURA

BOX2D: COLISÃO COM TILED



Após introduzir o mapa devemos criar colisões com o mesmo, para isso temos que percorrer a lista de objetos do Tiled e criarmos um Body para cada quadrado que necessita de colisão.

Parte do código do arquivo CriarMundo

```
1 usage
public CriarMundo(Tela screen){
    World world = screen.getWorld();
    TiledMap map = screen.getMap();

    BodyDef bdef = new BodyDef();
    PolygonShape shape = new PolygonShape();
    FixtureDef fdef = new FixtureDef();
    Body body;

    for(MapObject object : map.getLayers().get(2).getObjects().getByType(RectangleMapObject.class)){
        Rectangle rect = ((RectangleMapObject) object).getRectangle();

        bdef.type = BodyDef.BodyType.StaticBody;
        bdef.position.set((rect.getX() + rect.getWidth() / 2) * Body2D.TAMANHO, (rect.getY() + rect.getHeight() / 2) * Body2

        body = world.createBody(bdef);

        shape.setAsBox(hx: rect.getWidth() / 2 * Body2D.TAMANHO, hy: rect.getHeight() / 2 * Body2D.TAMANHO);
        fdef.shape = shape;
        body.createFixture(fdef);
    }
```

MAPA

INIMIGOS

Um jogo deve ter adversários para os jogadores e esse é o nosso próximo tópico. Para criar um inimigo apenas deve se criar um Body, seguindo a mesma lógica do personagem principal, cria-se um Body que recebe uma forma e pode ser desenhado a partir de um Sprite.

inimigo.java



criar o Body

tela.java



atribuir
sprite body

Clique [aqui](#) para saber como criar um body

Não precisamos criar o objeto inimigo na tela

1 usage

```
public void create(Tela screen, float cordX, float cordY)
```

```
batch = new SpriteBatch();
```

```
img = new Texture(internalPath: "inimigo.png");
```

```
sprite = new Sprite(img);
```

```
sprite.setPosition(cordX, cordY);
```

```
this.retangulo = new RetanguloColisao(sprite.getX()
```

```
BodyDef bodyDef = new BodyDef();
```

```
bodyDef.position.set(sprite.getX(), sprite.getY())
```

```
bodyDef.type = BodyDef.BodyType.DynamicBody;
```

```
body = screen.getWorld().createBody(bodyDef);
```

```
CircleShape shape = new CircleShape(sprite.getWidth() / 2);
```

```
shape.setRadius(sprite.getWidth() / 2);
```

**No arquivo inimigo.java
devemos receber as posições
dos objetos**

MODIFICAR

Quando adicionarmos inimigos devemos modificar todo o código para que os mesmos tenham colisão e sejam renderizados da forma certa.

**Temos que criar uma lista
como as balas**

INIMIGOS

Clique [aqui](#) para baixar o Tiled.

ATIVIDADE: COLISÕES E INIMIGOS

Com todo aprendizado adquirido até aqui, você deve usar o seu último código como base, para fazer esse. Com o mapa e as colisões adicionados, crie mais de um inimigo a partir das camadas de objeto do Tiled e alguma consequência para caso o personagem for atingido

- 1. Não esqueça de declarar todas as variáveis;**
- 2. Lembre-se do loop que o libGDX percorre;**
- 3. Crie também alguma consequência para quando algum inimigo for atingido**