libGDX

https://github.com/libgdx/libgdx/wiki/

Rafael Vieira Coelho

rafael.coelho@farroupilha.ifrs.edu.br

Tópicos

- The Application Framework
- A Simple Game
- File Handling
- Networking
- Preferences
- **Input Handling**
- Memory Management
- Audio
- Graphics

https://github.com/libgdx/libgdx/wiki/

Entrada de Dados

- Cada plataforma (android, desktop, etc.) tem suas peculiaridades no que diz respeito ao recebimento de dados:
    - Desktop e WEB: teclado e mouse
    - Android: touch screen
- O Libgdx abstrai estes dispositivos de entrada e trata alguns deles como iguais. Mouse e touch screens são tratados como a mesma coisa. O mouse é como se fosse um touch screen com um dedo somente.
- Podemos verificar o estado do dispositivo **periodicamente** ou definir um **listener** que capture qualquer interação.
- Vocês podem verificar mais informações sobre o **Módulo Input**:
  https://github.com/libgdx/libgdx/blob/master/gdx/src/com/badlogic/gdx/Input.java

Entrada de Dados

- Podemos descobrir quais características estão disponíveis no dispositivo

```
boolean hardwareKeyboard = Gdx.input.isPeripheralAvailable(Peripheral.HardwareKeyboard)
boolean multiTouch = Gdx.input.isPeripheralAvailable(Peripheral.MultitouchScreen);
```

- E no caso do Android, como forma de economia de energia, podemos debilitar algumas delas:

```
public class MyGameActivity extends AndroidApplication {
   @Override
   public void onCreate (Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      AndroidApplicationConfiguration config = new AndroidApplicationConfiguration();
      config.useAccelerometer = false;
      config.useCompass = false;
      initialize(new MyGame(), config);
   }
}
```

Entrada de Dados: Teclado

- Todo momento que o usuário clica em uma tecla, é gerado um evento.
- Cada objeto de evento carrega consigo um código da tecla (**key-code**).
- O ato de verificar se uma tecla foi pressionada é chamada de **pooling:**

```
// Check if the A key is pressed
boolean isPressed = Gdx.input.isKeyPressed(Keys.A);
```

- https://libgdx.badlogicgames.com/nightlies/docs/api/com/badlogic/gdx/Input.Keys.html

Fields

Modifier and Type	Field and Description
static int	A
static int	ALT_LEFT
static int	ALT_RIGHT
static int	ANY_KEY
static int	APOSTROPHE
static int	AT
static int	B
static int	BACK
static int	BACKSLASH
static int	BACKSPACE
static int	BUTTON_A
static int	BUTTON_B
static int	BUTTON_C
static int	BUTTON_CIRCLE
static int	BUTTON_L1
static int	BUTTON_L2

Entrada de Dados: Touch

- Para verificar se está sendo pressionada a tela:

```
boolean isTouched = Gdx.input.isTouched();
```

- Para verificar quantos dedos estão pressionando:

```
boolean firstFingerTouching = Gdx.input.isTouched(0);
boolean secondFingerTouching = Gdx.input.isTouched(1);
boolean thirdFingerTouching = Gdx.input.isTouched(2);
```

- Para verificar se algum foi pressionado:

```
boolean justTouched = Gdx.input.justTouched();
```

- Para obter a posição de onde foi pressionada a tela (o padrão é zero):

```
int firstX = Gdx.input.getX();
int firstY = Gdx.input.getY();
int secondX = Gdx.input.getX(1);
int secondY = Gdx.input.getY(1);
```

- A posição inicial é no canto esquerdo superior (0,0)
- E podemos obter a pressão dada pelo usuário (valor de 0 a 1):

```
Gdx.input.getPressure()
```

Entrada de Dados: Mouse

- Em aplicações desktop, podemos verificar qual botão do mouse foi pressionado:

```
boolean leftPressed = Gdx.input.isButtonPressed(Input.Buttons.LEFT);
boolean rightPressed = Gdx.input.isButtonPressed(Input.Buttons.RIGHT);
```

- Em android podemos emular apenas o botão esquerdo.

- https://libgdx.badlogicgames.com/ci/nightlies/docs/api/com/badlogic/gdx/Input.Buttons.html

Fields

Modifier and Type	Field and Description
static int	**BACK**
static int	**FORWARD**
static int	**LEFT**
static int	**MIDDLE**
static int	**RIGHT**

Entrada de Dados: Tratamento de Eventos

- Para realizar o tratamento de eventos, precisamos implementar a interface **InputProcessor**

- Métodos de teclado: **keyDown** (tecla pressionada), **KeyUp** (tecla liberada) e **KeyTyped** (tecla digitada). Retornam o Key Code da classe Input.Keys.

- Métodos de mouse/touch: **touchDown**, **touchUp**, **touchDragged**, **mouseMoved** e **scrolled**. Retornam a coordenada (pointer index) e qual botão do mouse foi pressionado ou liberado (em dispositivos touch, é sempre Buttons.LEFT)

- Assim que se define a sua classe de tratamento de eventos, precisamos informar ao LibGDX que ela fará o tratamento:

```
MyInputProcessor inputProcessor = new MyInputProcessor();
Gdx.input.setInputProcessor(inputProcessor);
```

- Os eventos são repassados ao seu objeto InputProcessor antes de renderizar a tela

```
ApplicationListener.render()
```

```
public class MyInputProcessor implements InputProcessor {
   public boolean keyDown (int keycode) {
      return false;
   }

   public boolean keyUp (int keycode) {
      return false;
   }

   public boolean keyTyped (char character) {
      return false;
   }

   public boolean touchDown (int x, int y, int pointer, int button) {
      return false;
   }

   public boolean touchUp (int x, int y, int pointer, int button) {
      return false;
   }

   public boolean touchDragged (int x, int y, int pointer) {
      return false;
   }

   public boolean mouseMoved (int x, int y) {
      return false;
   }

   public boolean scrolled (int amount) {
      return false;
   }
}
```

Entrada de Dados: Tratamento de Eventos

- Também é possível implementar uma Inner Class (classe anônima)

```
Gdx.input.setInputProcessor(new InputAdapter () {
   @Override
   public boolean touchDown (int x, int y, int pointer, int button) {
      // your touch down code here
      return true; // return true to indicate the event was handled
   }

   @Override
   public boolean touchUp (int x, int y, int pointer, int button) {
      // your touch up code here
      return true; // return true to indicate the event was handled
   }
});
```

Entrada de Dados: Tratamento de Eventos

- E podemos ter múltiplos tratadores de evento através do **InputMultiplexer**.

- Por exemplo, um para o UI (User Interface) que deve tratar primeiro e outro para os eventos de entrada que manipulam o jogo (game's world).

```
InputMultiplexer multiplexer = new InputMultiplexer();
multiplexer.addProcessor(new MyUiInputProcessor());
multiplexer.addProcessor(new MyGameInputProcessor());
Gdx.input.setInputProcessor(multiplexer);
```

- Caso o primeiro InputProcessor não consiga tratar o evento (retorna falso), ele é repassado para o próximo InputProcessor.

Entrada de Dados: Tratamento de Eventos

- Para mover um personagem (ator) usando o InputProcessor, precisamos adicionar uma **flag** para sabermos o estado atual do sprite em sua própria classe.

```
public class Bob
{
    boolean leftMove;
    boolean rightMove;
    ...
    updateMotion()
    {
            if (leftMove)
            {
                    x -= 5 * Gdx.graphics.getDeltaTime();
            }
            if (rightMove)
            {
                    x += 5 * Gdx.graphics.getDeltaTime();
            }
    }
    ...
    public void setLeftMove(boolean t)
    {
            if(rightMove && t) rightMove = false;
            leftMove = t;
    }
    public void setRightMove(boolean t)
    {
            if(leftMove && t) leftMove = false;
            rightMove = t;
    }
```

Entrada de Dados: Tratamento de Eventos

- E no InputProcessor, usar esta **flag**.

```
...
    @Override
    public boolean keyDown(int keycode)
    {
            switch (keycode)
            {
                case Keys.LEFT:
                        bob.setLeftMove(true);
                        break;
                case Keys.RIGHT:
                        bob.setRightMove(true);
                        break;
            }
            return true;
    }
    @Override
    public boolean keyUp(int keycode)
    {
            switch (keycode)
            {
                case Keys.LEFT:
                        bob.setLeftMove(false);
                        break;
                case Keys.RIGHT:
                        bob.setRightMove(false);
                        break;
            }
            return true;
    }
```

Entrada de Dados: Texto do Usuário

- Podemos abrir uma caixa de diálogo (desktop) ou um pop up (android) para receber uma entrada de dados em forma textual.

- Para isto, precisamos implementar a interface **MyTextInputListener:**

```
public class MyTextInputListener implements TextInputListener {
   @Override
   public void input (String text) {
   }

   @Override
   public void canceled () {
   }
}
```

- O método **input()** será chamado quando o usuário informar um texto e o método **canceled()** será chamado quando o usuário fechar a caixa de diálogo ou apertar o botão voltar no android.

- Para que apareça a caixa de diálogo, deve-se instanciar o listener e chamar o método **getTextInput():**

```
MyTextInputListener listener = new MyTextInputListener();
Gdx.input.getTextInput(listener, "Dialog Title", "Initial Textfield Value", "Hint Value");
```

Tópicos

- The Application Framework
- A Simple Game
- File Handling
- Networking
- Preferences
- Input Handling
- **Memory Management**
- Audio
- Graphics

https://github.com/libgdx/libgdx/wiki/

Gerenciamento de Memória

- Jogos costumam consumir bastante recursos da máquina.
- Imagens e efeitos sonoros podem usar uma quantidade grande de memória RAM
- A maioria destes recursos não são gerenciados pelo Garbage Collector de Java.
- Desta forma, precisamos decidir quando liberar recursos no LibGDX.
- Estes recursos são objetos cujas classes implementam a interface **Disposable**, o que força a implementação do método **dispose()**.
- Ao lado, são apresentadas as principais classes cujos objetos precisam ser liberados em memória após seu uso.

- AssetManager
- Bitmap
- BitmapFont
- BitmapFontCache
- CameraGroupStrategy
- DecalBatch
- ETC1Data
- FrameBuffer
- Mesh
- Model
- ModelBatch
- ParticleEffect
- Pixmap
- PixmapPacker
- Shader
- ShaderProgram
- Shape
- Skin
- SpriteBatch
- SpriteCache
- Stage
- Texture
- TextureAtlas
- TileAtlas
- TileMapRenderer
- com.badlogic.gdx.physics.box2d.World
- all bullet classes

Object Pooling

- Trata-se do reuso de objetos inativos ao invés de criar novos objetos em memória.
- Cria-se então um pool de objetos e quando se necessita de um, basta obtê-lo do pool.
- Quando o objeto é liberado (método **reset()**), automaticamente ele retorna ao pool de objetos disponíveis para reuso.
- Para isto, sua classe precisa implementar a interface **Pool.Poolable**

```java
/**
 * Callback method when the object is freed. It is automatically
 * Must reset every meaningful field of this bullet.
 */
@Override
public void reset() {
    position.set(0,0);
    alive = false;
}
```

```java
public class Bullet implements Pool.Poolable {

    public Vector2 position;
    public boolean alive;

    /**
     * Bullet constructor. Just initialize variables.
     */
    public Bullet() {
        this.position = new Vector2();
        this.alive = false;
    }

    /**
     * Initialize the bullet. Call this method after getting a bulle
     */
    public void init(float posX, float posY) {
        position.set(posX,  posY);
        alive = true;
    }

     /**
      * Method called each frame, which updates the bullet.
      */
     public void update (float delta) {

         // update bullet position
         position.add(1*delta*60, 1*delta*60);

         // if bullet is out of screen, set it to dead
         if (isOutOfScreen()) alive = false;
     }
}
```

Object Pooling

- E na classe do mundo do jogo (código ao lado).
- A classe Pools provê métodos estáticos para criar dinamicamente pools de qualquer objeto (generics):

```
private final Pool<Bullet> bulletPool = Pools.get(Bullet.class);
```

- Os objetos são obtidos do Pool através do método **obtain()**
- E liberados para retornar ao Pool com o método **free()**

```
public class World {

    // array containing the active bullets.
    private final Array<Bullet> activeBullets = new Array<Bullet>();

    // bullet pool.
    private final Pool<Bullet> bulletPool = new Pool<Bullet>() {
        @Override
        protected Bullet newObject() {
                return new Bullet();
        }
    };

    public void update(float delta) {

        // if you want to spawn a new bullet:
        Bullet item = bulletPool.obtain();
        item.init(2, 2);
        activeBullets.add(item);

        // if you want to free dead bullets, returning them to the poo
        Bullet item;
        int len = activeBullets.size;
        for (int i = len; --i >= 0;) {
            item = activeBullets.get(i);
            if (item.alive == false) {
                activeBullets.removeIndex(i);
                bulletPool.free(item);
            }
        }
    }
}
```

Tópicos

- The Application Framework
- A Simple Game
- File Handling
- Networking
- Preferences
- Input Handling
- Memory Management
- **Audio**
- Graphics

https://github.com/libgdx/libgdx/wiki/

Módulo Audio

- O acesso as facilidades proporcionadas para áudio é feita através do módulo abaixo:

```
Audio audio = Gdx.audio;
```

- Quando a aplicação é pausada/retomada LibGDX automaticamente gerencia o áudio utilizado.
- Efeitos sonoros são pequenos sons que não excedem alguns segundos e colocados no fundo de um jogo para proporcionar dinâmicas como o pulo de um personagem ou o disparo de uma arma.
- Vários formatos são suportados: **MP3**, **OGG** e **WAV**

Efeitos Sonoros

- São representados pela Interface **Sound**
- Para carregar um efeito sonoro, um exemplo é:

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

- O arquivo de áudio se chama **mysound.mp3** e está localizado em uma pasta interna do projeto chamada
- Assim que tiver sido carregado, podemos chamar o método **play()** para tocá-lo:

```
sound.play(1.0f);
```

- Existem outros métodos possíveis:

```
long id = sound.play(1.0f); // play new sound and keep handle for further ma
sound.stop(id);             // stops the sound instance immediately
sound.setPitch(id, 2);      // increases the pitch to 2x the original pitch

id = sound.play(1.0f);      // plays the sound a second time, this is treate
sound.setPan(id, -1, 1);    // sets the pan of the sound to the left side at
sound.setLooping(id, true); // keeps the sound looping
sound.stop(id);             // stops the looping sound
sound.dispose();
```

Efeitos Sonoros

- Quando se tratar de um efeito sonoro pequeno, podemos usar um objeto Music que não carrega em memória e transmite diretamente do disco:

```
Music music = Gdx.audio.newMusic(Gdx.files.internal("data/mymusic.mp3"));
```

- E para tocar e usar a música, basta:

```
music.play();

music.setVolume(0.5f);                  // sets the volume to half the maximu
music.setLooping(true);                 // will repeat playback until music.s
music.stop();                           // stops the playback
music.pause();                          // pauses the playback
music.play();                           // resumes the playback
boolean isPlaying = music.isPlaying(); // obvious :)
boolean isLooping = music.isLooping(); // obvious as well :)
float position = music.getPosition();  // returns the playback position in s

music.dispose();
```