

Carregando tela inicial antes do jogo começar

Para fazer isso de forma simples:

1) Você pode criar três novos atributos no jogo:

- `jogo_iniciou` - um atributo verdadeiro/falso para controlar se o jogo já começou ou não;
- `tempo_inicio_jogo` - um atributo do tipo `double` para controlar o tempo decorrido até o início do jogo;
- `tela_inicial_jogo` - uma textura para carregar a tela inicial do jogo.

```
bool jogo_iniciou;  
double tempo_inicio_jogo;  
Texture2D tela_inicial_jogo;
```

2) No método de inicialização do jogo (`Initialize`), inicialize os atributos `jogo_iniciou` e `tempo_inicio_jogo` com os valores `false` e `0`.

```
jogo_iniciou = false;  
tempo_inicio_jogo = 0;
```

3) No método de carga de conteúdo (`LoadContent`), coloque o código abaixo para carregar a imagem `tela_inicial_jogo.png` que será a tela inicial do jogo:

```
// Carregando a tela inicial do jogo  
tela_inicial_jogo = Texture2D.FromFile(graphics.GraphicsDevice,  
    "../../Content/Imagens/tela_inicial_jogo.PNG");
```

4) Programe o método de atualização do jogo para `aguardar o início do jogo`. O seu método de atualização do jogo (`Update`) ficará mais ou menos assim:

```
protected override void Update(GameTime gameTime)  
{  
  
    if (jogo_iniciou)  
    {  
        // Coloque aqui o seu código do método Update do jogo  
    }  
    else  
    {  
        // Atualizando o tempo de início do jogo  
        tempo_inicio_jogo +=  
            gameTime.ElapsedGameTime.Milliseconds;  
  
        // Aguardando 10 segundos para iniciar o jogo  
        if (tempo_inicio_jogo > 10000)  
        {  
            jogo_iniciou = true;  
        }  
    }  
}
```

```
    }  
  }  
}
```

4) Programe o método de exibição do jogo (**Draw**), exibindo a tela inicial do jogo antes do jogo ser iniciado:

```
protected override void Draw(GameTime gameTime)  
{  
    if (jogo_iniciou)  
    {  
        // Coloque aqui o seu código do método Draw do jogo  
    }  
    else  
    {  
        // Carregando a tela inicial do jogo  
        spriteBatch.Begin();  
        spriteBatch.Draw(tela_inicial_jogo, new Vector2(0,0),  
                          Color.White);  
        spriteBatch.End();  
    }  
    base.Draw(gameTime);  
}
```

Limitando os disparos do inimigo

Para limitar os disparos dos inimigos, podemos criar 2 atributos novos na classe Espaçoave: `_tempo_tiro` e `_pode_atirar`.

O atributo `_tempo_tiro` vai medir quanto já passou desde que o inimigo disparou o último tiro.

O atributo `_pode_atirar` controla se já o inimigo já pode fazer um novo disparo de tiro;

Para criar os atributos, acrescente as linhas abaixo dentro da sua classe Espaçoave:

```
private double _tempo_tiro;
private bool _pode_atirar;
```

Também precisaremos criar 2 novos métodos: `TempoTiro` e `PodeAtirar`. Serão usados para acessar e atualizar os valores dos atributos criados.

```
public double TempoTiro()
{
    return _tempo_tiro;
}

public bool PodeAtirar()
{
    return _pode_atirar;
}

public void TempoTiro(double p_tempo_tiro)
{
    _tempo_tiro = p_tempo_tiro;
}

public void PodeAtirar(bool p_pode_atirar)
{
    _pode_atirar = p_pode_atirar;
}
```

O código da Aula 11 para fazer o inimigo atacar era este:

```
// Atirando contra o jogador
if (NaveInimiga.Atacar(NaveJogador.PosicaoX(),
                      NaveJogador.PosicaoY()))
{
    NaveInimiga.Atirar(3,graphics.GraphicsDevice,
                      "../..../Content/Imagens/tiro.png", TirosDisparados);
}
```

Precisaremos modificar um pouco o código, acrescentando uma verificação para atirar apenas se o inimigo já puder disparar um novo tiro. Veja em azul o que foi acrescentado:

```
// Atirando contra o jogador
if (NaveInimiga.Atacar(NaveJogador.PosicaoX(),
                      NaveJogador.PosicaoY()) && NaveInimiga.PodeAtirar())
{
    NaveInimiga.Atirar(3, graphics.GraphicsDevice,
                      "../..../Content/Imagens/nave.png", TirosDisparados);
}
```

```
    NaveInimiga.PodeAtirar(false);  
}  
  
// Controlando o tempo decorrido após o disparo do tiro  
NaveInimiga.TempoTiro(NaveInimiga.TempoTiro() +  
    gameTime.ElapsedGameTime.Milliseconds);  
  
// Liberando o disparo de um novo tiro após 100 milissegundos  
if (NaveInimiga.TempoTiro() > 100)  
{  
    NaveInimiga.TempoTiro(0);  
    NaveInimiga.PodeAtirar(true);  
}
```

Repare que também acrescentamos um controle de tempo decorrido após o disparo do tiro, de forma a poder liberar um novo disparo a cada 100 milissegundos de jogo.

Rotacionando a figura da espaçonave

Para rotacionar a figura da espaçonave:

Vamos acrescentar um método Desenhar a mais para a classe Espaçonave, que conterà **mais um parâmetro `p_rotacao`**, para fazer a rotação da textura. A classe Espaçonave agora terá dois métodos Desenhar diferentes. Veja como ficou:

```
// Método Desenhar
public void Desenhar(SpriteBatch p_sprite)
{
    p_sprite.Begin();
    p_sprite.Draw(_textura, new Rectangle(_posicao.x, _posicao.y, _textura.Width,
    _textura.Height), Color.White);
    p_sprite.End();
}

// Método Desenhar com rotação da textura
public void Desenhar(SpriteBatch p_sprite, float p_rotacao)
{
    p_sprite.Begin();
    p_sprite.Draw(_textura, new Rectangle(_posicao.x, _posicao.y, _textura.Width,
    _textura.Height), null, Color.White, p_rotacao, new Vector2(0f,0f),
    SpriteEffects.None, 0f);
    p_sprite.End();
}
```

Os valores para o parâmetro `p_rotacao` são:

0f - Não rotaciona a imagem

1.57f - Rotaciona a imagem para a direita

3.14f - Rotaciona a imagem para baixo

4.71f - Rotaciona a imagem para a esquerda

Para desenhar a nave do jogador virada para a direita, por exemplo, basta colocar o seguinte comando:

```
NaveJogador.Desenhar(spriteBatch,1.57f);
```

Produzindo números aleatórios

Para produzir números aleatórios em C# é bastante fácil. Segue abaixo um exemplo para gerar aleatoriamente as posições x e y de um inimigo:

1) No projeto de jogo, coloque os seguintes comandos para criar um gerador de números aleatórios e posições x e y do inimigo:

```
Random geradorNumeros;  
int posicaoX_inimigo, posicaoY_inimigo;
```

2) No método de inicialização do jogo (Initialize), acrescente os comandos abaixo para inicializar o gerador de números e gerar as posições x (número de 1 a 800) e y (1 a 600) do inimigo:

```
geradorNumeros = new Random();  
posicaoX_inimigo = geradorNumeros.Next(1, 800);  
posicaoY_inimigo = geradorNumeros.Next(1, 600);
```

Criando uma lista de paredes para o labirinto

Para criar as paredes para um jogo do tipo labirinto, experimente fazer o seguinte:

- 1) Coloque como cenário de fundo a imagem do labirinto;
- 2) Crie uma imagem modelo de parede para o labirinto. De preferência uma imagem mais ou menos com a espessura (width) e altura (height) máximas que uma parede do seu labirinto possa ter (veja imagem anexo);
- 3) Crie uma classe Parede para o seu jogo com os atributos posicaoX, posicaoY, altura e largura e um método Desenhar (igual o da classe Espaconave). Crie também uma lista de paredes para o seu jogo:
`List<Parede> ListadeParedes;`
- 4) No método de atualização do jogo (Update), após o jogador se movimentar, verifique se houve colisão entre alguma parede da lista e o personagem do jogador. Neste caso, impeça (desfaça) o movimento do jogador. Pode fazer isso com o comando foreach, assim como fizemos para os tiros disparados.

```
    // Caso haja, desfaça o movimento do jogador, impedindo ele de atravessar a
    parede
}
```

Abraços,

```
foreach (Parede oParede in ListadeParedes)
```

```
{
```

```
    // Verifique se houve colisão entre o objeto oParede e o personagem do jogador
```

André

Leitura de arquivo texto binário (Enviado pelo Christian: 2009-1)

```
using System;

using System.IO;

Em seguida:

using (StreamReader sr = File.OpenText("seu_arquivo_texto_vai_aqui"))
{
    /// Aloca 255 bytes para leitura do arquivo de fase 15x15 (15
    linhas por 15 colunas)

    /// a variavel receberá 225 bytes lidos do arquivo desejado

    char[] buffer = new char[225];

    /// Faz a leitura dos dados do arquivo desejado começando na
    posição 0 até a posicao 224

    /// 225 bytes no total

    sr.Read(buffer, 0, 224);
}
```

Controlando o disparo de tiros do jogador

Para gerar apenas um único tiro por vez, você precisa criar um atributo para controlar se a barra de espaços já foi pressionada. Para tal:

Acrescente aos atributos do jogo o atributo abaixo:

```
bool apertou_tiro;
```

Em seguida, acrescente ao método de inicialização do jogo (Initialize) o comando para colocar o valor "falso" dentro desse atributo:

```
apertou_tiro = false;
```

Agora, no método de atualização do jogo (Update), coloque uma verificação adicional para testar se o botão de tiro já foi apertado, de forma que o tiro só seja gerado uma única vez e o valor do atributo `apertou_tiro` seja atualizado para "verdadeiro" após o tiro ter sido gerado:

```
//Apertou o tiro
```

```
if (teclado.IsKeyDown(Keys.Space) && !apertou_tiro)  
{
```

```
NaveJogador.Atirar(graphics.GraphicsDevice, "../../Content/Imagens/nave.png",  
TirosDisparados);
```

```
apertou_tiro = true;
```

```
}
```

Por fim, acrescente um teste para verificar se a barra de espaços já foi solta pelo jogador. Neste caso o atributo `apertou_tiro` deve retornar o seu valor para "falso":

```
if (teclado.IsKeyUp(Keys.Space) && apertou_tiro)
```

```
{
```

```
apertou_tiro = false;
```

```
}
```

Abraços,

André

Adicionar várias cópias de um mesmo personagem

Você poderia simplesmente **retirar o atributo `_textura`** da sua classe de personagem e alterar o método desenhar para que trabalhe com uma **textura que seja passada como parâmetro**. Segue abaixo como ficaria o método desenhar da sua classe com o parâmetro adicional de textura:

```
// Desenha o objeto na tela utilizando um sprite
public void Desenhar(SpriteBatch p_sprite, Texture2D p_textura)
{
    p_sprite.Begin();
    p_sprite.Draw(p_textura, new Rectangle(_posicaoX, _posicaoY,
        p_textura.Width, p_textura.Height), Color.White);
    p_sprite.End();
}
```

Desta forma você cria uma única textura e utiliza ela para desenhar várias cópias de personagens.

Você pode criar uma lista de personagens utilizando o comando `List`, supondo que você criou uma classe chamada `Personagem`:

```
// Criando uma lista de personagens para usar no jogo
List<Personagem> ListadePersonagens;
```

É necessário inicializar a lista de personagens no método de inicialização do jogo (`Initialize`):

```
// Inicializando a lista de personagens do jogo
ListadePersonagens = new List<Personagem>();
```

Para acrescentar personagens à esta lista, utilize o comando `Add`, assumindo que você tenha criado um objeto da classe `Personagem` chamado `oPersonagem`:

```
// Acrescenta um personagem à lista de personagens
ListadePersonagens.Add(oPersonagem);
```

Criando uma classe para carregar várias texturas no jogo

Você pode criar uma classe chamada **Textura**, contendo esses dois métodos: **Carregar** e **Desenhar**. Ficaria assim:

```
public class Textura
{
    private Texture2D _textura;
    private int _posicaoX;
    private int _posicaoY;

    // Carrega a textura a partir de um arquivo de imagem
    public void Carregar(GraphicsDevice p_dispositivo, string p_local)
    {
        _textura = Texture2D.FromFile(p_dispositivo, p_local);
    }

    // Desenha a textura na tela utilizando um sprite
    public void Desenhar(SpriteBatch p_sprite, int p_posicaoX, int
        p_posicaoY)
    {
        p_sprite.Begin();
        p_sprite.Draw(_textura, new Rectangle(p_posicaoX, p_posicaoY,
            _textura.Width, _textura.Height), Color.White);
        p_sprite.End();
    }

    public void PosicaoX(int p_posicaoX)
    {
        _posicaoX = p_posicaoX;
    }

    public void PosicaoY(int p_posicaoY)
    {
        _posicaoY = p_posicaoY;
    }
}
```

Em seguida você pode criar uma lista de texturas para usar dentro do seu jogo:

```
public class Game1 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;

    // Definindo a espaçonave do jogador no jogo
    Espaconave NaveJogador;

    // Definindo um leitor de teclas do teclado
    KeyboardState teclado;

    // Criando uma lista de texturas para usar no jogo
    List<Textura> ListadeTexturas;
```

Depois você precisa inicializar a lista de texturas no método de inicialização do jogo (**Initialize**):

```
protected override void Initialize()
{
```

```

// Criando efetivamente a espaçonave do jogador
NaveJogador = new Espaconave();

// Colocando um nome para a nave do jogador
NaveJogador.Nome("Falcão Justiceiro");

// Inicializando a lista de texturas do jogo
ListadeTexturas = new List<Textura>();

base.Initialize();
}

```

Para acrescentar uma nova textura à lista de texturas, você precisa usar os comandos abaixo:

```

// Cria uma textura
Textura oTextura;
oTextura = new Textura();

// Define as posições x e y da textura
oTextura.PosicaoX(10);
oTextura.PosicaoY(10);

// Carrega uma imagem para a textura
oTextura.Carregar(this.GraphicsDevice, "../../../imagens/nave.png");

// Acrescenta a textura na lista de texturas
ListadeTexturas.Add(oTextura);

```

No método de exibição do jogo (Draw) você pode exibir todas as Texturas da lista com o seguinte comando:

```

protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here
    NaveJogador.Desenhar(spriteBatch);

    foreach (Textura oTextura in ListadeTexturas)
    {
        oTextura.Desenhar(spriteBatch);
    }

    // TODO: Add your drawing code here

    base.Draw(gameTime);
}

```

Parando de tocar a música do jogo

Para parar de tocar a música do jogo não é difícil. Siga os 3 passos abaixo:

1) No seu projeto de jogo, crie um atributo chamado musicajogo, do tipo Cue:

```
Cue musicajogo;
```

2) No método de carga de conteúdo do jogo (LoadContent), substitua o comando que carrega a música do jogo:

```
soundBank.PlayCue("musica");
```

pelos seguintes comandos:

```
musicajogo = soundBank.GetCue("musica");  
musicajogo.Play();
```

3) Em qualquer lugar do método de atualização do jogo (Update) quando quiser parar a música, escreva o comando:

```
musicajogo.Pause();
```

Digitando o nome do jogador dentro do jogo

Segue abaixo um passo-a-passo para permitir que o jogador digite seu nome dentro no jogo:

1) Criar o atributo **NomeJogador**, do tipo string (conjunto de caracteres) dentro do seu projeto de jogo, usando o comando abaixo em azul:

String NomeJogador;

2) Dentro do método de construção do jogo (**Game1**), acrescentar a linha abaixo que está em azul:

```
public Game1()
{
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";
    Components.Add(new GamerServicesComponent(this));
}
```

Após acrescentar esse componente ao seu jogo, o jogador poderá receber mensagens utilizando a interface do XNA

3) Dentro do método de inicialização do jogo (**Initialize**), acrescentar o comando abaixo em azul para exibir a mensagem "Digite seu nome:" ao iniciar o jogo:

```
protected override void Initialize()
{
    // TODO: Add your initialization logic here
    base.Initialize();

    Guide.BeginShowKeyboardInput(PlayerIndex.One, "Jogador", "Digite seu nome:", "", RetornarNomeJogador, null);
}
```

4) Criar o método **RetornarNomejogador**, que será automaticamente chamado quando o jogador terminar de digitar o seu nome e que serve para guardar o nome do jogador dentro do atributo **NomeJogador**:

```
private void RetornarNomeJogador(IAsyncResult resultado)
{
    //Console.WriteLine(Guide.EndShowKeyboardInput(result));
    NomeJogador = Guide.EndShowKeyboardInput(resultado).ToString();
}
```

Exibindo a pontuação do jogo e outros textos na tela:

Para exibir a pontuação do jogador e outros textos na tela, basta:

1) Criar no seu jogo (classe Game1) o atributo fonte, do tipo SpriteFont:

SpriteFont fonte;

2) No Solution Explorer (menu direito dentro do Visual Studio) clicar com o botão direito em cima da pasta Content e escolher a opção Add/New Item e em seguida selecionar SpriteFont. Dar o nome "Arial" para essa fonte.

3) No método de carga do conteúdo do jogo (LoadContent), incluir o comando abaixo para carregar a fonte Arial:

```
fonte = Content.Load<SpriteFont>("Arial");
```

4) No método de exibição do jogo (Draw), incluir o comando abaixo para exibir o texto desejado na tela:

```
spriteBatch.DrawString(fonte, "Score Geral: 100 pontos", new Vector2(10, 20), Color.Gold);
```

Repare que:

- Em azul aparece o nome da fonte que você usou para escrever;
- Em vermelho está o texto que você deseja exibir na tela do jogo;
- Em verde são as posições X e Y da tela onde vai aparecer o texto;
- Em rosa está a cor do texto exibido.

Alternando a velocidade conforme a movimentação do Personagem

Para incrementar a velocidade do Personagem conforme a movimentação, basta:

- 1) Criar um atributo **_direcao** na classe do Personagem;
- 2) O valor da velocidade do Personagem se inicia em 0, ou seja, o personagem fica estático;
- 3) Ajustar o método Mover para armazenar a direção do movimento atual do jogador em **_direcao**, sempre que a velocidade chegar a zero (0);
- 4) Ajustar o método Mover para comparar se a direção do movimento atual realizado pelo jogador coincide com a direção para onde ele estava se movimentando (frente frente).

Caso coincida, incrementar o valor do atributo **_velocidade**.

Caso seja a direção contrária, vá reduzindo o valor do atributo **_velocidade** até que chegue a zero (0).

Quando este valor chegar a zero, você armazena a nova direção do movimento do jogador em **_direcao** e passa a incrementar novamente atributo **_velocidade**.

Outra opção é configurar duas teclas a mais, uma para acelerar e outra para reduzir a velocidade do Personagem. É mais fácil, mas não fica tão legal.