TUTORIAL PASSO A PASSO

GameUtil – XNA

Este tutorial tem por objetivo mostrar todas as classes do pacote “GameUtil”, desenvolvido para a plataforma XNA , com o objetivo de “facilitar” o desenvolvimento de jogos.

1) A biblioteca “game.util”

Esta biblioteca contém as classes básicas para o desenvolvimento de um jogo. Vamos conhecer agora as classes dessa biblioteca:

1.1) A classe Image

Essa classe tem somente uma única finalidade, que é exibir imagens na tela. Vejamos os métodos que existem nela :

Método	Descrição
`public Image(ContentManager content, String path, int x, int y, int w, int h)`	Método construtor, que carrega a imagem na memória.

`public override void Draw(SpriteBatch spriteBatch)`
 	
`public void Draw(SpriteBatch spriteBatch, bool flip_horizontal)`
	Este método é responsável por “Desenhar” a imagem na tela. O atributo “flip_horizontal” , quando assumido como **true**, inverte “na horizontal” a imagem original.

`public void SetX(int x)`	Neste método definimos a coordenada “X” da imagem na tela.
`public void SetY(int y)`	Neste método definimos a coordenada “Y” da imagem na tela.
`public void SetWidth(int width)`	Neste método definimos a largura da imagem.
`public void SetHeight(int height)`	Neste método definimos a altura da imagem.
`public void SetBounds(int x, int y, int width, int height)`	Neste método definimos as posições x e y, incluindo também largura e altura da imagem.
`public void SetTag(string tag)`	Neste método definimos uma “tag” (rótulo) para o objeto (imagem).
`public int GetX()`	Este método “retorna” a coordenada x do objeto na tela.
`public int GetY()`	Este método “retorna” a coordenada Y do objeto na tela.

`public int GetWidth()`	Este método “retorna” a largura da imagem.
`public int GetHeight()`	Este método “retorna” a altura da imagem.
`public int GetTag()`	Este método “retorna” a tag do objeto (imagem)
`public bool IsClicked(int posx, int posy)`	Esse método retorna verdadeiro (**true**) quando a imagem “teoricamente” está sendo clicada, e falso (**false**) caso contrário.
`public void MoveByX(int x)`	Este método move o objeto na coordenada “x”, partindo da posição dele atual na tela. Se o valor do parâmetro “x” for positivo, move o objeto para direita, senão, para esquerda. O deslocamento do objeto é dado em “pixels”. Exemplo: supondo um objeto na posição x 450: Objeto.MoveByX(5) O código acima move o objeto até a posição x 455.
`public void MoveByY(int y)`	Este método move o objeto na coordenada “y”, partindo da posição dele atual na tela. Se o valor do parâmetro “y” for positivo, move o objeto para baixo, senão, para cima. O deslocamento do objeto é dado em “pixels”. Exemplo: supondo um objeto na posição y 20: Objeto.MoveByY(5) O código acima move o objeto até a posição y 25.

1.2) A classe AnimationSprites

Nessa classe podemos adicionar várias imagens (sprites), para podermos realizar uma animação. Veja os métodos que existem nela :

Método	Descrição
`AnimationSprites(ContentManager c, int x, int y, int width, int height)`	Método construtor, que carrega o objeto na memória.
`public void Add(String pathName)`	Neste método podemos adicionar as imagens que farão parte de uma animação
`public void Start(int frames, bool loop)`	Neste método iniciamos a animação do objeto. O intervalo de troca de imagens da animação é dado em “frames”. No parâmetro loop, definimos se a animação ficará em loop (se **true**) ou não (se **false**)
`public void Stop()`	Encerra uma animação.

`public override void Draw(SpriteBatch spriteBatch)`
 	
`public void Draw(SpriteBatch spriteBatch, bool flip_horizontal)`
	Este método é responsável por “Desenhar” a animação na tela. O atributo “flip_horizontal” , quando assumido como **true**, inverte “na horizontal” a animação original.

`public void SetX(int x)`	Neste método definimos a coordenada “X” da animação na tela.
`public void SetY(int y)`	Neste método definimos a coordenada “Y” da animação na tela.
`public void SetWidth(int width)`	Neste método definimos a largura do objeto.
`public void SetHeight(int height)`	Neste método definimos a altura do objeto.
`public void SetBounds(int x, int y, int width, int height)`	Neste método definimos as posições x e y, incluindo também largura e altura do objeto.
`public void SetTag(string  tag)`	Neste método definimos uma “tag” (rótulo) para o objeto (animação).
`public int GetX()`	Este método “retorna” a coordenada x do objeto na tela.
`public int GetY()`	Este método “retorna” a coordenada Y do objeto na tela.
`public int GetWidth()`	Este método “retorna” a largura da animação.
`public int GetHeight()`	Este método “retorna” a altura da animação.
`public int GetTag()`	Este método “retorna” a tag do objeto (animação)
`public bool IsClicked(int posx, int posy)`	Esse método retorna verdadeiro (**true**) quando a animação “teoricamente” está sendo clicada, e falso (**false**) caso contrário.
`public void MoveByX(int x)`	Este método move o objeto na coordenada “x”, partindo da posição dele atual na tela. Se o valor do parâmetro “x” for positivo, move o objeto para direita, senão, para esquerda. O deslocamento do objeto é dado em “pixels”. Exemplo: supondo um objeto na posição x 450: Objeto.MoveByX(5) O código acima move o objeto até a posição x 455.
`public void MoveByY(int y)`	Este método move o objeto na coordenada “y”, partindo da posição dele atual na tela. Se o valor do parâmetro “y” for positivo, move o objeto para baixo, senão, para cima. O deslocamento do objeto é dado em “pixels”. Exemplo: supondo um objeto na posição y 20: Objeto.MoveByY(5) O código acima move o objeto até a posição y 25.

1.3) A classe GameElement

Essa é a classe base de todas as outras que foram citadas. Todos os objetos (classes) do jogo, desse pacote, são uma GameElement. Essa classe não pode ser instanciada diretamente (pois ela é uma classe abstrata, modelo para a criação de outras). Vejamos os métodos desta classe:

`public override void Draw(SpriteBatch spriteBatch)`	Este método é responsável por "Desenhar" a animação na tela.
`public void SetX(int x)`	Neste método definimos a coordenada "X" do objeto na tela.
`public void SetY(int y)`	Neste método definimos a coordenada "Y" do objeto na tela.
`public void SetWidth(int width)`	Neste método definimos a largura do objeto.
`public void SetHeight(int height)`	Neste método definimos a altura do objeto.
`public void SetBounds(int x, int y, int width, int height)`	Neste método definimos as posições x e y, incluindo também largura e altura do objeto.
`public void SetTag(string tag)`	Neste método definimos uma "tag" (rótulo) para o objeto.
`public int GetX()`	Este método "retorna" a coordenada x do objeto na tela.
`public int GetY()`	Este método "retorna" a coordenada Y do objeto na tela.
`public int GetWidth()`	Este método "retorna" a largura da animação.
`public int GetHeight()`	Este método "retorna" a altura da animação.
`public int GetTag()`	Este método "retorna" a tag do objeto (animação)
`public bool IsClicked(int posx, int posy)`	Esse método retorna verdadeiro (**true**) quando o objeto "teoricamente" está sendo clicada, e falso (**false**) caso contrário.
`public void MoveByX(int x)`	Este método move o objeto na coordenada "x", partindo da posição dele atual na tela. Se o valor do parâmetro "x" for positivo, move o objeto para direita, senão, para esquerda. O deslocamento do objeto é dado em "pixels". Exemplo: supondo um objeto na posição x 450: Objeto.MoveByX(5) O código acima move o objeto até a posição x 455.
`public void MoveByY(int y)`	Este método move o objeto na coordenada "y", partindo da posição dele atual na tela. Se o valor do parâmetro "y" for positivo, move o objeto para baixo, senão, para cima. O deslocamento do objeto é dado em "pixels". Exemplo: supondo um objeto na posição y 20:

	Objeto.MoveByY(5) O código acima move o objeto até a posição y 25.

1.4) A classe Collision

Essa classe foi desenvolvida para “detectar” colisões entre os objetos. Ela possui o seguinte método:

```
public static bool Check(GameElement obj1, GameElement obj2)
```

O método acima “detecta” colisões entre dos objetos. Ele retorna **true** se houve uma colisão e **false** caso contrário.

2) A biblioteca “game.util.character”

Esta biblioteca contém somente uma única classe , vejamos qual classe é.

2.1) A classe Character

Esta classe é muito útil para criar um “personagem” com várias animações sem muito esforço. Ela é bem completa, já possui um sistema de colisões (por lado e por queda), física (pulo e queda), dano (pisca quando recebe um dano) e etc. Vejamos os métodos que ela possui:

Método	Descrição
`public Character(ContentManager content, int x, int y, int w, int h)`	Método construtor, que carrega o objeto na memória.
`public void AddNewSpriteIdle(string pathName)`	Neste método podemos adicionar as imagens referentes à animação dele em “repouso” (traduzindo: parado)
`public void AddNewSpriteWalking(string pathName)`	Neste método podemos adicionar as imagens referentes à animação dele andando.
`public void AddNewSpriteRunning(string pathName)`	Neste método podemos adicionar as imagens referentes à animação dele correndo.
`public void AddNewSpriteAttack1(string pathName)`	Neste método podemos adicionar as imagens referentes à animação de ataque (correspondendo a primeira animação de ataque).
`public void AddNewSpriteAttack2(string pathName)`	Neste método podemos adicionar as imagens referentes à animação de ataque

	(correspondendo a segunda animação de ataque).
`public void AddNewSpriteAttack3(string pathName)`	Neste método podemos adicionar as imagens referentes à animação de ataque (correspondendo a terceira animação de ataque).
`public void AddNewSpriteAttack4(string pathName)`	Neste método podemos adicionar as imagens referentes à animação de ataque (correspondendo a quarta animação de ataque).
`public void AddNewSpriteAttack5(string pathName)`	Neste método podemos adicionar as imagens referentes à animação de ataque (correspondendo a quinta animação de ataque).
`public void AddNewSpriteJumping(string pathName)`	Neste método podemos adicionar as imagens referentes à animação dele pulando
`public void AddNewSpriteDamage(string pathName)`	Neste método podemos adicionar as imagens referentes à animação dele sofrendo dano.
`public void Idle(int frames, bool loop)`	Neste método iniciamos a animação do personagem em estado de repouso (parado). O intervalo de troca de imagens da animação é dado em “frames”. No parâmetro loop, definimos se a animação ficará em loop (se **true**) ou não (se **false**)
`public void WalkingToRight(int frames, bool loop)`	Neste método iniciamos a animação do personagem andando para direita. O intervalo de troca de imagens da animação é dado em “frames”. No parâmetro loop, definimos se a animação ficará em loop (se **true**) ou não (se **false**)
`public void WalkingToLeft(int frames, bool loop)`	Neste método iniciamos a animação do personagem andando para esquerda. O intervalo de troca de imagens da animação é dado em “frames”. No parâmetro loop, definimos se a animação ficará em loop (se **true**) ou não (se **false**)
`public void RunningToRight(int frames, bool loop)`	Neste método iniciamos a animação do personagem correndo para direita. O intervalo de troca de imagens da animação é dado em “frames”. No parâmetro loop, definimos se a animação ficará em loop (se **true**) ou não (se **false**)
`public void RunningToLeft(int frames, bool loop)`	Neste método iniciamos a animação do personagem correndo para esquerda. O intervalo de troca de imagens da animação é dado em “frames”. No parâmetro loop, definimos se a animação ficará em loop (se **true**) ou não (se **false**)
`public void Attack1(int frames)`	Neste método iniciamos a animação do personagem realizando um ataque (referente ao primeiro ataque). O intervalo de troca de imagens da animação é dado em “frames”. Terminada a animação, o personagem volta a animação anterior, se o método Update estiver sendo executado.
	Neste método iniciamos a animação do personagem realizando um ataque

`public void Attack2(int frames)`	(referente ao segundo ataque). O intervalo de troca de imagens da animação é dado em “frames”. Terminada a animação, o personagem volta a animação anterior, se o método Update estiver sendo executado
`public void Attack3(int frames)`	Neste método iniciamos a animação do personagem realizando um ataque (referente ao terceiro ataque). O intervalo de troca de imagens da animação é dado em “frames”. Terminada a animação, o personagem volta a animação anterior, se o método Update estiver sendo executado
`public void Attack4(int frames)`	Neste método iniciamos a animação do personagem realizando um ataque (referente ao quarto ataque). O intervalo de troca de imagens da animação é dado em “frames”. Terminada a animação, o personagem volta a animação anterior, se o método Update estiver sendo executado
`public void Attack5(int frames)`	Neste método iniciamos a animação do personagem realizando um ataque (referente ao quinto ataque). O intervalo de troca de imagens da animação é dado em “frames”. Terminada a animação, o personagem volta a animação anterior, se o método Update estiver sendo executado
`public void Jump(int frames, bool loop)`	Neste método iniciamos a animação do personagem pulando. O intervalo de troca de imagens da animação é dado em “frames”. No parâmetro loop, definimos se a animação ficará em loop (se **true**) ou não (se **false**). Além da animação, o personagem se desloca para cima e para baixo (realizando o pulo), caso o método Update seja executado e o método SetEnableFall esteja definido como true (como já é por padrão).
`public void Update(Scene scene)`	Este método “processa” o pulo, as animações de ataque, dano e colisões que o personagem pode vir a sofrer. Como parâmetro possui o objeto scene (do tipo Scene, ver tópico 3.1 para mais detalhes sobre a classe), onde nele existem todos os elementos do jogo, para o processamento das colisões de queda.
`public void AddCollisionElementOfFallByTag(String tag)`	O processamento das colisões do personagem em uma cena, é feito pelas tags dos objetos presentes em na mesma. Nesse método adicionamos as “tags” dos objetos que funcionaram como pontos de colisão de queda do personagem, para que quando o personagem “colidir” com o objeto que possui essa “tag”, o mesmo parar e ficar em cima dele.
	O processamento das colisões do personagem em uma cena, é feito pelas

`public void AddCollisionElementOfSideByTag(String tag)`	tags dos objetos presentes em na mesma. Nesse método adicionamos as “tags” dos objetos que funcionaram como pontos de colisão “pelos lados” do personagem, para que quando o personagem “colidir” com o objeto que possui essa “tag”, o mesmo parar e ficar imóvel, sem poder avançar.
`public bool CollisionBySide(Scene scene)`	Este método retorna verdadeiro (**true**) se houver alguma “colisão” pelos lados , por parte do personagem, e falso (**false**) caso contrário.
`public void WalkingToLeft(int frames, bool loop)`	Neste método iniciamos a animação do personagem andando para esquerda. O intervalo de troca de imagens da animação é dado em “frames”. No parâmetro loop, definimos se a animação ficará em loop (se **true**) ou não (se **false**)
`public void WalkingToRight(int frames, bool loop)`	Neste método iniciamos a animação do personagem andando para direita. O intervalo de troca de imagens da animação é dado em “frames”. No parâmetro loop, definimos se a animação ficará em loop (se **true**) ou não (se **false**)
`public void RunningToLeft(int frames, bool loop)`	Neste método iniciamos a animação do personagem correndo para esquerda. O intervalo de troca de imagens da animação é dado em “frames”. No parâmetro loop, definimos se a animação ficará em loop (se **true**) ou não (se **false**)
`public void RunningToRight(int frames, bool loop)`	Neste método iniciamos a animação do personagem correndo para direita. O intervalo de troca de imagens da animação é dado em “frames”. No parâmetro loop, definimos se a animação ficará em loop (se **true**) ou não (se **false**)
`public void SetEnableFall(bool fall)`	Neste método podemos definir se o processamento do pulo e da queda será processado pelo método Update (se **true**) ou não (caso **false**).
`public void SufferDamage(int frames)`	Inicia a animação dele sofrendo dano e pisca o personagem, indicando a situação. Se o método Update estiver sendo usado, ao terminar a animação, o personagem volta ao seu estado de repouso (animação dele parado).
`public bool IsDamaged()`	Retorna verdadeiro (**true**) se o personagem estiver em estado de dano e falso (**false**) caso contrário.
`public bool IsAttacking()`	Retorna verdadeiro (**true**) se o personagem estiver “atacando” e falso (**false**) caso contrário.
`public void SetMaxFramesDamage(int max_frames)`	Neste método definimos em frames o tempo máximo de dano no personagem. O efeito será executado se o método Update estiver em execução.
`public void TurnToLeft()`	Este método “vira” o personagem para direita (invertendo as imagens dele).
`public void TurnToRight()`	Este método “vira” o personagem para esquerda (invertendo as imagens dele).

`public override void Draw(SpriteBatch spriteBatch)`	Este método é responsável por "Desenhar" a personagem na tela.
`public void SetX(int x)`	Neste método definimos a coordenada "X" do personagem na tela.
`public void SetY(int y)`	Neste método definimos a coordenada "Y" da animação na tela.
`public void SetWidth(int width)`	Neste método definimos a largura do objeto.
`public void SetHeight(int height)`	Neste método definimos a altura do objeto.
`public void SetBounds(int x, int y, int width, int height)`	Neste método definimos as posições x e y, incluindo também largura e altura do objeto.
`public void SetTag(string  tag)`	Neste método definimos uma "tag" (rótulo) para o objeto (personagem).
`public int GetX()`	Este método "retorna" a coordenada x do objeto na tela.
`public int GetY()`	Este método "retorna" a coordenada Y do objeto na tela.
`public int GetWidth()`	Este método "retorna" a largura do personagem.
`public int GetHeight()`	Este método "retorna" a altura do personagem.
`public int GetTag()`	Este método "retorna" a tag do objeto (personagem)
`public bool IsClicked(int posx, int posy)`	Esse método retorna verdadeiro (**true**) quando o objeto "teoricamente" está sendo clicada, e falso (**false**) caso contrário.
`public void MoveByX(int x)`	Este método move o objeto na coordenada "x", partindo da posição dele atual na tela. Se o valor do parâmetro "x" for positivo, move o objeto para direita, senão, para esquerda. O deslocamento do objeto é dado em "pixels". Exemplo: supondo um objeto na posição x 450: Objeto.MoveByX(5) O código acima move o objeto até a posição x 455.
`public void MoveByY(int y)`	Este método move o objeto na coordenada "y", partindo da posição dele atual na tela. Se o valor do parâmetro "y" for positivo, move o objeto para baixo, senão, para cima. O deslocamento do objeto é dado em "pixels". Exemplo: supondo um objeto na posição y 20: Objeto.MoveByY(5) O código acima move o objeto até a posição y 25.

3) A biblioteca “game.util.scene”

Esta biblioteca contém apenas uma classe, vejamos qual classe é:

3.1) A classe Scene

Está classe representa a cena do jogo, onde todos os elementos que vão ser visualizados e processados no jogo, devem estar dentro dela. Vejamos quais são seus métodos.

Método	Descrição
`public Scene()`	Método construtor, que carrega o objeto na memória.
`public void Add(GameElement element)`	Adiciona um elemento no objeto scene, podendo ser ele qualquer elemento do tipo GameElement (uma Image, AnimationSprites, Character e etc.).
`public GameElement Get(int index)`	Esse método retorna um objeto da cena através do seu índice. Se você quiser obter o primeiro elemento da cena, o seu índice é 0, o segundo elemento da cena, seu índice 1 e assim por diante.

`public void Remove(int index)`
 	
`public void Remove(GameElement element)`
	Este método “deleta” um objeto da cena, através do seu índice ou através de sua instância.

`public int GetCount()`	Retorna o total de elementos presentes na cena (objeto scene)
`public List<GameElement> Elements()`	Este método retorna em um “array”, todos os elementos presentes no objeto scene.
`public void Draw(SpriteBatch spritebatch)`	Desenha todos os objetos presentes no objeto scene na tela.
`public void SetX(int x)`	Neste método definimos a coordenada “X” de todos os elementos do objeto scene na tela.
`public void SetY(int y)`	Neste método definimos a coordenada “Y” de todos os elementos do objeto scene na tela.
`public void MoveByX(int x)`	Este método move todos os objetos da cena na coordenada “x”, partindo da posição atual na tela de cada um.
`public void MoveByY(int y)`	Este método move todos os objetos da cena na coordenada “y”, partindo da posição atual na tela de cada um.