# Simulações de Movimentos Físicos em Duas Dimensões, Utilizando Framework XNA

Luiz Filipe F. Carneiro, Wietske Ineke Meyering (orientadora)
*Faculdade Farias Brito*
*{luizfilipefc, wietske.meyering}@gmail.com*

## Resumo

*Para que jogos 2D sejam produzidos com qualidade é necessário simular a realidade. Para isso conceitos de física mecânica e matemática devem ser aplicados no processo de codificação de um jogo, principalmente na emulação de circunstâncias reais, como uma bola batendo na borda da mesa de sinuca, oléo em uma pista de corrida como fator de redução de atrito ou um objeto em um espaço com gravidade desprezível. Os objetivos deste artigo são explanar os conceitos de física mecânica e matemática para a simulação de movimentos em duas dimensões de um determinado objeto e a aplicação destes em simuladores construídos em um ambiente na plataforma (ou Framework) XNA Game Studio.*

## 1. Introdução

Para simular movimentos em 2D é necessário considerar conceitos físicos como inércia, força, ação e reação, movimento acelerado, movimento retardado, bem como o uso do plano cartesiano e vetorização na matemática computacional para a interação de um objeto com o meio. Além disso, para lidar com a plataforma XNA é necessário ser proficiente com a liguagem de programação C#.

## 2. Aspectos Básicos da plataforma XNA

Ao se criar um novo projeto na plataforma XNA deve-se estar ciente de quais componentes foram criados e para que cada um deles serve. Logo após, verifica-se a função de cada um dos métodos que compõem a estrutura inicial do projeto.

Um novo projeto contém uma pasta chamada *Content* e dois arquivos, *Game1.cs* e *Program.cs*.

*Game1.cs* é o arquivo de código que carrega a lógica de um jogo e inicializa as variavéis de instância [4].

*Program.cs* é o arquivo de código que instancia um objeto de *Game1.cs* e executa o código do jogo [4].

*Content* guarda todo o conteúdo de mídia (Imagens, Fontes) que será utilizado durante o processo de desenvolvimento do jogo em XNA [4].

### 2.1. Sprites

Um *sprite* é uma imagem em duas dimensões que pode ser manipulada independentemente do cenário do jogo [4]. No XNA ela possui propriedades como largura, altura e velocidade.

Segundo Cawood [5], *sprite* é um conjunto de imagens guardadas em um único arquivo de imagens com o propósito de criar animação. Em jogos 2D *sprites* são freqüentemente usados em visão de terceira pessoa. Como componente, um *sprite* apresenta propriedades especiais de imagem como filtros *alpha* que torna possível o *alpha blending*, que é definido como o processo de combinação de cor de um objeto em primeiro plano translúcido, com a cor do fundo.

### 2.2. O "Game Loop"

O *game loop* é a repetição das rotinas cíclicas de um jogo, como entradas dos jogadores, cálculos que serão determinados pela inteligência artificial do jogo [4], detecções de colisão, etc. O *game loop* só encerra quando um critério de finalização do jogo é executado dentro de seu escopo [4]. Ele é constituído de dois métodos: *Update()*, que contém toda a lógica do jogo, e o método *Draw()*, que adiciona os objetos em suas possíveis novas posições. Estes métodos estão localizados na classe *Game1* [4].

### 2.3. Carregando o Conteúdo

Todo e qualquer conteúdo multimídia usado no jogo, além de ser alocado na pasta *Content,* deve ser instanciado na classe *Game1* no método *LoadContent()* para ser desenhado no método *Draw()* que faz parte do *game loop* [4]. O método *Inicialize()* é utilizado para inicializar as váriaves de instância da classe *Game1* [4].

Para descarregar os objetos que não serão mais utilizados faz-se o uso do método *UnLoadContent()* [4].

## 3. Fundamentos Matemáticos aplicados à plataforma XNA

### 3.1. Plano Cartesiano

Um plano cartesiano é um espaço bidimensional constituido por dois eixos, onde um ponto é representado por um par ordenado na forma (x,y).

Todo plano cartesiano possui uma coordenada especial utilizada para definir o ponto de origem [1]. A plataforma XNA trata o canto superior esquerdo da tela do jogo como origem O (0,0). Nele, o eixo Y é orientado de cima para baixo e o eixo X da esquerda pra direita, como na definição de imagens em Computação Gráfica.

O campo visual do plano cartesiano computacional é delimitado pela largura e pela altura da janela do jogo fazendo com que este possua um ponto máximo, entretanto pode-se utilizar coordenadas fora do campo visual da janela do jogo.

### 3.2. Vetores em duas dimensões

Algumas grandezas, expressas por variáveis de instância no código de um jogo em XNA, devem possuir módulo, direção e sentido, portanto podemos definir força e velocidade como grandezas vetoriais que são diferentes de uma grandeza escalar como a massa de um objeto [2]. Um vetor em 2D ($\vec{v}$) pode ser representado por um segmento de reta que parte da origem e se encerra nas suas componentes ($v_x$, $v_y$).

**3.2.1. Algumas operações básicas com vetores.** Negar um vetor [1] ou a Inversão da direção [3] significa mudar o sentido do vetor multiplicando as suas componentes cartesianas por menos um para obter um vetor com mesma direção e módulo, mas de sentido oposto. Um objeto com velocidade ($\vec{v}$) construído em uma simulação na plataforma XNA se desloca para direita em determinado momento e após a Negação de seu vetor ($\vec{v}$) o objeto passa a se deslocar para esquerda.

O módulo de um vetor 2D ($\vec{v}$) tal que seus componentes sejam ($v_x$, $v_y$) é dado pela expressão a seguir:

$$|v| = \sqrt{v_x^2 + v_y^2}$$

Na prática vetores não são medidos pelo seu módulo, mas são expressos pela medida de suas coordenadas. Na plataforma XNA é possível obter esse resultado apenas utilizando um método *Length()* de um objeto da classe *Vector2* [5].

Multiplicar um vetor 2D por um escalar (i) é multiplicar o seu módulo por (i) ou de modo semelhante multiplicar todas as componentes desse vetor por (i) [1].

Segundo Dunn [1] existem situações mais fáceis de lidar quando se conhece o caminho do vetor. O sentido de um determinado vetor 2D é representada pela sua normalização.

Seja ($\vec{a}$) a normalização de um vetor 2D ($\vec{v}$), onde este é formado por componentes ($\vec{v}_x$, $\vec{v}_y$), é representada pela expressão abaixo:

$$\vec{a} = \frac{\vec{v}_x + \vec{v}_y}{|\sqrt{v_x^2 + v_y^2}|}$$

Contudo na Plataforma XNA o valor da normalização de um vetor pode ser obtido com o método *Normalize()* da classe *Vector2* [5].

A soma e a subtração de vetores consistem em realizar estas operações básicas nas componentes de dois ou mais vetores de duas dimensões.

## 4. Fundamentos da Física Mecânica Aplicados à Plataforma XNA

### 4.1. Tempo

O tempo é uma medida escalar fisicamente expressa em horas, minutos, segundos e suas frações.

As definições físicas do tempo ficam explícitas se o programador conseguir vincular o tempo ao *game loop*, imaginando que cada iteração é uma unidade de tempo.

### 4.2. Matéria

Para que um determinado *sprite* no plano cartesiano interaja com as bordas da janela ou com algum outro *sprite* ou obedeça a queda gravitacional é necessário atribuir à ele, massa e solidez. Dessa forma ele pode colidir com outros *sprites* e ser afetado por grande

parte dos fenômenos físicos importantes para a movimentação em duas dimenções.

Para tanto, deve-se delimitar o espaço ocupado por esses objetos e realizar testes de interseção. No XNA isso é feito usando o método *Intersects()* [4] de alguma das classes que implementam detecção de colisão. Existem dois tipos de testes a serem feitos em ambientes computacionais, os testes estáticos que retornam um valor booleano, verdadeiro se houver colisão ou falso caso contrário, e os testes de interseção dinâmicos que além de retornarem valores booleanos têm uma terceira componente temporal que indica quando os objetos colidem. O teste de colisão dinâmico é mais utilizado para colisão de raios e raramente é aplicado na simulação de movimento dos objetos [1].

Existem três tipos básicos de colisões estáticas na plataforma XNA que são amplamente utilizadas em desenvolvimento de jogos 3D e 2D são eles: *BoundingBox*, *BoundingSphere*, e colisão por pixel.

**4.2.1. Colisões.** Colisões por *BoundingBox* são utilizadas na edificação de um jogo em duas dimensões, quando o *sprite* em questão tem o formato retangular [4], para que não haja colisão entre as partes em *alpha* do *sprite*.

De forma similar à colisão por *BoundingBox*, as colisões por *BoundingSphere* [5] em duas dimensões, apenas podem ser realizadas em *sprites* de forma circular. Para colisões por *BoundingSphere* é necessário ter informações do centro e do raio da circunferência.

A colisão por pixel de simulações em 2D é a mais eficiente, complexa e custosa computacionalmente de se edificar [4]. A sua utilização se dá por meio de jogos que empregam o conceito de *sprites* animados com formatos complexos como um *sprite* em formato humanóide. O método desta colisão consiste em guardar a informação dos pixels da matriz que formam a imagem em um array e comparar se há colisão entre aqueles pixels que possuem informações *alpha* diferente de zero.

## 4.3. Leis de Newton Aplicadas à Simulação computacional

**4.3.1. Inércia.** Na iminência de uma força, um corpo tende a permanecer imóvel devido a inércia, que é uma propriedade da máteria. Um corpo permanece inerte em dois casos: quando se encontra em repouso ou em movimento uniforme (um exemplo disso é o movimento no vácuo), quando a força resultante atuando sobre o objeto é igual a zero, logo o corpo não sofre aceleração [2].

**4.3.2. Força.** Em XNA outro aspecto que pode ser abordado através do *game loop* é a força. A aplicação da simulação de movimento funciona de forma que se existe uma força resultante atuando sobre um determinado corpo na mesma direção da velocidade, a aceleração pode aumentar ou diminuir, dependendo do sentido da velocidade deste corpo (movimento acelerado, movimento retardado) em relação ao sentido da força aplicada [2]. Se o corpo se move para direita com uma força apontando para a esquerda e a velocidade do corpo é maior que zero, então o corpo sofre uma frenagem pois os sentidos dos vetores força e velocidade são contrários. Se o corpo se move para esquerda com uma força para esquerda, e a velocidade é maior que zero, o corpo sofre uma aceleração pois os vetores velocidade e força apontam para o mesmo sentido.

Se houver uma força constante atuando sobre este corpo, então a aceleração gerada por essa força é constante. Ao aplicar uma força resultante que mude de intensidade no decorrer do tempo haverá uma aceleração variando de acordo com a força, pois para toda e qualquer força, a massa do corpo independe do movimento na física mecânica [2].

Ao definir a força constante no simulador de movimentos em duas dimensões no XNA é possível tratá-la através da entrada do jogador para fazer com que essa força seja aplicada a determinado objeto, mudando assim seu vetor velocidade.

**4.3.3. Ação e Reação.** Uma força gerada pela reação do choque entre esferas muda o sentido da que causou a ação mantendo a direção e a intensidade se considerarmos que o sistema não apresenta forças externas [2]. Todo corpo atingido por outro com determinada força reage com força de mesma intensidade no sentido contrário.

Através da ação e reação, desprezando forças externas, é possível obter colisões elásticas como o choque entre duas esferas de bilhar.

## 4.4. Força de Atrito

A resistência que atua sobre um corpo que está deslizando sobre uma superfície denomina-se força de atrito cinético [2], definida pela expressão abaixo:

$$\vec{f}_{at} = m * \vec{N}$$

Onde $m$ é o coeficiente de atrito da superfície, portanto, quanto mais lisa é a superfície em que o objeto desliza menor será a força de atrito [2] e $\vec{N}$ é a força normal, perpendicular à força de atrito, fazendo

com que a força de atrito seja proporcional ao peso total do conjunto [2].

Em XNA é possível definir a força de atrito, ou suas componentes, para que os objetos interajam com o ambiente.

## 5. Resultados

Foram construídos dois simuladores físicos que utilizam os conceitos abordados neste artigo. O primeiro deles apresenta diversas aplicabilidades nas simulações de situações reais em duas dimensões, como resistência à força de atrito, movimento acelerado, movimento retardado e as três leis de Newton.

O componente principal deste simulador é um *sprite* que obtém velocidade através de entradas dos usuários. Cada tecla definida no algoritmo implica em uma força para que o *sprite* possa percorrer o plano cartesiano, parando apenas com a ação da força de atrito do ambiente ou outras entradas de usuário com o sentido da força em oposição ao deslocamento corrente.

A figura 1 apresenta o fluxograma do *Game Loop* para as entradas de usuário no simulador 1.

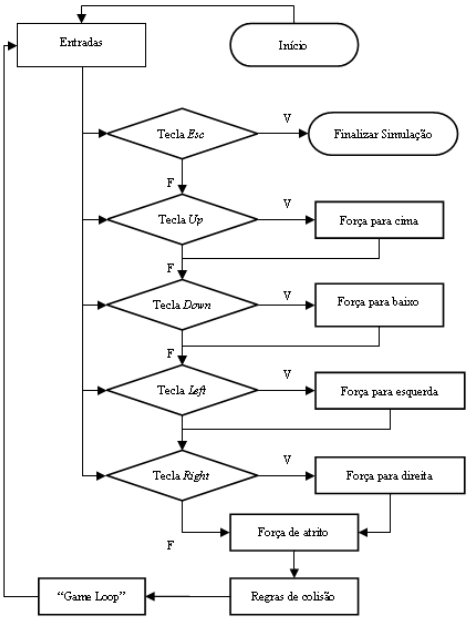


**Figura 1. Fluxograma do algoritmo de game loop para o simulador 1.**

O segundo simulador é um jogo completo de "pong", que consiste em um jogo de tênis em duas dimensões. Este simula todas as situações físicas de velocidade e efeito ocasionado pelo choque, definindo a direção resultante de uma esfera. A figura 2 mostra uma seqüência de imagens do simulador 2.

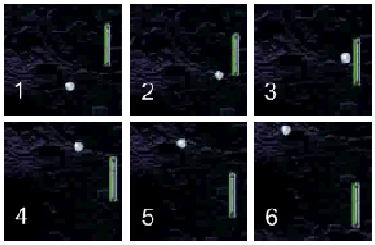


**Figura 2. Seqüência de imagens representando o choque elástico no simulador 2.**

Na sequência de imagens pode-se constatar que a trajetória inicial descrita pela esfera, antes do choque com o anteparo, representa uma função linear que cresce no eixo x e no eixo y a cada repetição do *Game Loop*.

Quando a esfera entra em contato com o anteparo verifica-se a relação de forças existentes entre o anteparo e a esfera, (o atrito entre os dois objetos e a força resultante do movimento do anteparo são levados em consideração no momento do choque), causando uma mudança na trajetória da esfera.

## 6. Conclusão

Este trabalho se dedicou ao estudo da plataforma XNA, a simulação de movimentos físicos em duas dimensões, englobando força de aceleração, força de atrito, inércia, choques elásticos e alguns conceitos matemáticos como, o cálculo do módulo, a normalização, a multiplicação, a soma e a subtração de vetores.

Como resultado foram edificados dois simuladores que aplicam os processos físicos estudados na plataforma XNA, com resultados satisfatórios.

Como trabalho futuro, pretende-se desenvolver simulações em ambientes de três dimensões utilizando a física e matemática como ferramenta de auxílio.

## 7. Referências


[1]Dunn, F., Parberry, I., *3D math primer for graphics and game development*, Wordware Publishing, 2002.

[2]Young, H.D., Freedman, R.A., *Sears e Zemansky Física I Mecânica*, 10 ed São Paulo, Addison Wesley, 2004.

[3]Tremblay, C., *Mathematics for game Developers*, Thomson, Boston, 2004.

[4]Lobão, A., Evangelista, B., Farias, J.A.L., *Beginning XNA 2.0 Game Programming*, Apress, New York, 2008.

[5]Cawood, S., McGee, P., *Microsoft XNA Game Studio Creator´s Guide*, McGraw Hill, 2007