# Inteligência Artificial para Jogos

## Aula 04 - Probabilidade

# Apresentação da Aula

Olá, seja bem-vindo(a) à aula 04! Nas aulas anteriores, você desenvolveu o cenário de um campo de futebol e os jogadores com alguns comportamentos e navegação. E como um bom técnico, mandou eles se aquecerem, preparando-os antes de uma partida. 😉

Mas, cá pra nós, não dá para ficar só correndo de um lado para o outro. Como é futebol, tem de ter uma bola... para chutar! Hum... O jogador ainda nem aprendeu a chutar.

Então, prepara-se para ensiná-lo!

## Objetivos

Conhecer noções básicas de estatística, probabilidade e de uma distribuição normal;

Compreender a definição de P;

Entender a utilização de probabilidades em um jogo usando o Unity;

Aplicar probabilidade para reconhecer padrões do usuário e para gerar ações em um personagem.

# 1. Chutes e incertezas

Antes de prosseguir, pense por um momento na palavra **chute**. Até agora ela foi utilizada no sentido literal, mas como seria no sentido figurativo? Quando você diz, por exemplo “Irei **chutar** algumas questões da prova”, quer dizer que você vai escolher uma opção que você acredita ser a melhor, mesmo que não seja necessariamente verdade. Pois é, para se ter uma ideia, hoje existem técnicas para isso. Como assim? Estudar para chutar? Foi-se o tempo em que **chutar** era somente uma forma de não entregar uma prova com questão em branco.

Se você vai adivinhar, então está falando de **incerteza**. Praticamente todos os jogos (há algumas exceções) possuem **incertezas** embutidas nas suas mecânicas. Em muitos casos, é ela que permite aumentar a jogabilidade, por exemplo, de um nível em um jogo. Como também, entre outros fatores, permite ao jogador uma certa tensão positiva ou apreensão por eventos inesperados e comportamentos imprevisíveis. A **incerteza** é, portanto, um mecanismo essencial no design de jogos e os desenvolvedores, incluindo você, precisam saber como modelá-la.


Você já deve ter visto a seguinte situação: quando o árbitro de futebol chama os dois capitães para o meio do gramado e decide o lado do campo em um "cara ou coroa". Pois é, a expressão é utilizada, na maioria das vezes, para a escolha de alguma coisa que tenha duas possibilidades (ou isso ou aquilo). Ou seja, existe a **probabilidade** de um dos times perder a posse da bola para iniciar o jogo. Você acabou de usar o termo cujo conceito começou a ser usado no século XVI para modelar incertezas.

## Curiosidade

Cara ou coroa já era! Árbitros personalizam moedas e dão brinde aos atletas...

Confira aqui

**Probabilidade** é uma medida ou fator potencial de algo acontecer entre todas as possíveis opções. Essa é uma definição simples, mas bastante rica porque envolve dois outros conceitos que precisam também ser modelados: **quando um evento ocorre e todas as opções possíveis de eventos**.


Voltando à situação da moeda, quero, por exemplo, saber qual a probabilidade de tirar "cara" ao lançá-la para o alto. Como a moeda possui 2 lados, ao lançá-la haverá um total de 2 possíveis resultados (todas as opções possíveis de eventos). Assim, a probabilidade, representada pela letra P, de "cara" ocorrer é de:

$$P = \frac{evento\ específico}{todas\ opções\ de\ eventos\ possíveis} = \frac{1}{2} = 0.5$$

A probabilidade de sair "cara" é, portanto, de 0.5. Como o número de eventos específicos será sempre menor ou igual ao número de todos os eventos possíveis, o valor de uma probabilidade estará sempre entre 0 e 1. Para simplificar a representação, usa-se então um valor em percentual, ou seja, um valor que dividido por 100 resulta no valor da probabilidade. No caso da moeda, a probabilidade é de

50%. Ótimo! Você já sabe modelar incertezas! Basta definir, para cada evento específico, um valor entre 0 e 1, que define a probabilidade de o evento ocorrer dentre todas as possibilidades possíveis.

Mas existem algumas restrições. Não se pode simplesmente dar qualquer valor entre 0 e 1 a todas as possibilidades. Não é possível que um dos lados de uma moeda tenha 90% de chance de sair após seu lançamento. Haveria algo incoerente aí. A soma das probabilidades de todos os eventos possíveis deve ser 1 (ou 100%).

Bom... mas onde esses conceitos entram no jogo?

Primeiro, comece ensinando o jogador a dar chutes (no sentido literal).

😉 Depois, ensine-o a escolher em qual canto da trave ele chutará. Em seguida, para promover as incertezas das quais você viu aqui, defina as probabilidades para onde ele chutará.

Havendo chutes em direção à trave, você precisará de um goleiro para defender o gol. Da mesma forma, o goleiro pode escolher aleatoriamente um lado para pular no momento do chute, ou seja, ter uma probabilidade de pulo para cada um dos lados. Mas será que ele não pode aprender para onde o jogador vai chutar? Se o jogador tiver uma tendência (ou maior probabilidade) para chutar no lado direito, será que o goleiro não pode se adaptar a isso? Você verá como fazê-lo se adaptar, aprendendo as tendências em função dos eventos que ocorrem.

Por fim, você irá usar o conceito de **Probabilidade** visto anteriormente para também modelar os erros nos chutes. Sim, claro. A bola nem sempre vai na direção que o jogador quer. Aliás, não me atrevo a desafiar nenhum jogador ou time para não ferir (possivelmente) sua honra de torcedor, mas tem jogador que raramente faz com que a bola vá onde ele quer. 😁

Mas e se você quer gerar um erro no chute, ele será de quanto? Outro ponto, o jogador vai chutar a bola lá nas arquibancadas ou ela vai sair “tirando a tinta” do travessão? Assim, você precisa também de uma forma para modelar valores contínuos, que não sejam simplesmente “à direita” ou “à esquerda”. Para isso, faz-se necessário ter noção sobre curvas de distribuição e como usá-las para definir o erro dos chutes.

# Atividade 01 – Penalidade máxima !!!

Até o momento, tudo que você fez foi o jogador ir de um lado para o outro. Todo o comportamento do jogo encontra-se em um único *script*, associado ao jogador. Agora, você tem mais objetos e pode melhorar a organização de mais *scripts*.

Comece colocando o jogador para cobrar um pênalti. Ele deve escolher um dos lados da trave para chutar, seguindo uma probabilidade, tanto para a esquerda quanto para a direita. Para isso, reestruture o jogo, deixando-o mais organizado. Já que você vai modelar a cobrança de pênalti, têm de ter no mínimo o cobrador, o goleiro e o gol. Além disso, precisa saber quando o chute fez gol, quando o goleiro conseguiu defendê-lo, bem como fazer com que o goleiro pule para um lado quando o jogador chutar. É bastante coisa! Vai ser necessária uma completa remodelagem do que se tem até agora.

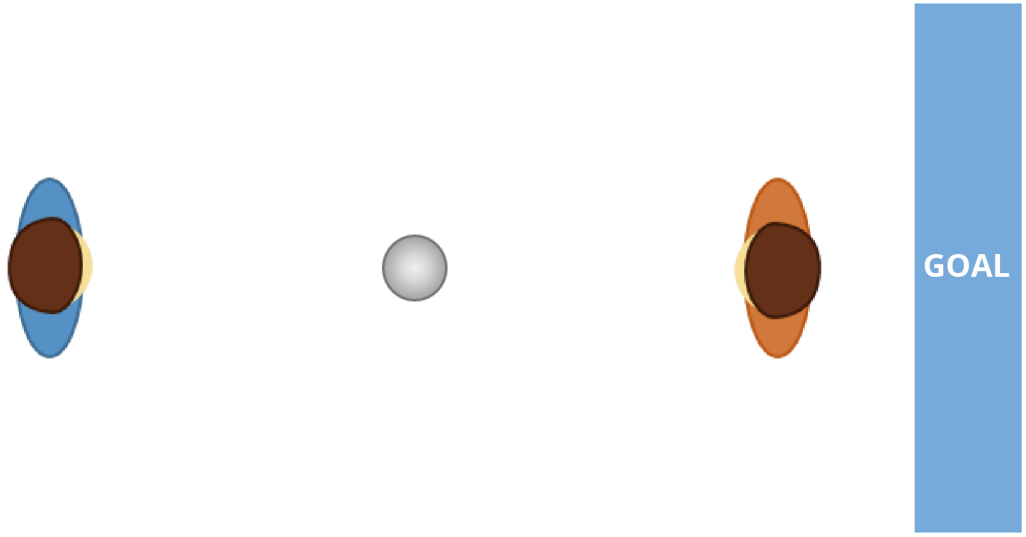
Normalmente, os desenvolvedores de jogos usam um objeto que coordena ou sincroniza elementos com os demais. Faça o mesmo, crie o objeto *GameManager*. Ele vai ser responsável por contar os pênaltis realizados, o número de gols feitos, assim como por reinicializar o cenário após cada pênalti. Como esse objeto é um "coordenador" do que acontece no jogo, ele precisa ser único (só existir uma instância da classe) e facilmente encontrado pelos demais objetos do jogo.

Essas são características comuns presentes em alguns objetos de muitos sistemas. Por isso, alguns programadores já criaram "padrões" que facilitam a criação de objetos com essas características. Inclusive, cada padrão tem um nome para que os programadores possam se entender mais facilmente. Você utilizará o que se chama de *Singleton*, usado para restringir uma classe para que ela tenha apenas um objeto e que ele seja facilmente encontrado pelos demais. Veja como se implementa um *Singleton* no Unity.

Além do *GameManager*, crie também um *script* para associar a uma área chamada *Goal* (a área de gol). O objeto representando essa área será responsável por avisar ao *GameManager* quando um gol ocorre, no momento que houver uma colisão da bola com a área de gol delimitada. Por isso, será necessário ter um *BoxCollider* associado a ele, de uma forma que quando a bola bater nesta área, você poderá sair gritando Gooooooooooool!!!

A Figura 01 mostra a representação visual dos objetos que foram descritos até o momento, inclusive do *GameManager* e do *Goal*, que são "invisíveis", ou seja, nem um nem o outro possuem visual (não há *SpriterRenderer* associado). Porém, *Goal* possui um *BoxCollider* para especificar a região em que, se a bola colidir, será considerado como gol. Além desses, o jogo deverá ter *scripts* tanto para o cobrador de pênalti quanto para o goleiro. O cobrador precisa escolher de forma aleatória um dos lados do gol para chutar. Por sua vez, o goleiro precisa esperar o chute do batedor para poder saltar para um dos lados, escolhido também de forma aleatória.

**Figura 01** - Objetos a serem criados no cenário da atividade




Após o chute, o objeto *GameManager* reiniciará os objetos que precisarão voltar às suas posições, ou seja, o cobrador, o goleiro e a bola. Por essa razão, você tem de ter nos *scripts* associados aos três objetos um método chamado *Reset()*, com o objetivo de reiniciar a posição dos três. O Código 01 mostra como a reinicialização pode ser feita no *script* a ser associado à bola. Códigos similares devem ser igualmente implementados nos *scripts* do cobrador do pênalti e do goleiro. Esse *script* guarda a informação da posição inicial da bola e, quando for solicitado, através do método *Reset()*, sua posição é restaurada e o seu deslocamento é zerado. Assim, o *script* tem como único objetivo reinicializar a posição e velocidade para um novo pênalti.

**Código 01** – *Script* a ser associado a bola

```
1   public class Ball : MonoBehaviour {
2
3       private Rigidbody2D body;
4       private Vector2 initialPosition;
5
6       void Start () {
7           body = GetComponent<Rigidbody2D>();
8           initialPosition = body.position;
9           Reset();
10      }
11
12      public void Reset() {
13          body.position = initialPosition;
14          body.velocity = Vector2.zero;
15      }
16  }
17
```

Ótimo! Agora você tem ideia de todos os objetos que irão compor o cenário e seus respectivos *scripts*. Comece a trabalhar pelo *GameManager*. Como já foi dito, ele implementa um padrão de projeto chamado *Singleton*. Esse padrão permite que exista apenas uma instância (objeto) da classe e que ela seja facilmente acessível por todos os objetos do jogo.

Primeiramente, como se tem apenas uma instância, guarde-a na própria classe. Ou seja, ela deve ser um atributo da classe (para isso, use o termo *static* na definição do atributo). Você irá também defini-la como uma propriedade, tendo os métodos *get* e set específicos para cada uma. Além disso, para evitar que objetos de outras classes alterem quem é a instância que você está guardando, altere o acesso do método set para que apenas a própria classe *GameManager* possa alterá-la. Por fim, se o *script* de *GameManager* for associado a mais de um objeto do jogo, como a instância deve ser a mesma, se ela existir, a antiga deve ser desalocada da memória, e a instância válida passa a ser a última criada. Coloque esse teste no método *Awake()* do objeto. Essas observações vão resultar no trecho de código apresentado abaixo.

**Código 02** – Trecho do código do *GameManager* assegurando o padrão de projeto *Singleton* na Unity

```
1   public class GameManager : MonoBehaviour {
2      ...
3      public static GameManager instance { get; private set; }
4
5      void Awake() {
6         if (instance != null && instance != this) {
7            Destroy(gameObject);
8         }
9         instance = this;
10     }
11    ...
12  }
13
```

Para as suas estatísticas, o *GameManager* precisa armazenar quantos chutes foram dados e quantos deles resultaram em gol. Você irá, então, definir dois atributos com esse propósito (*score* e *shots*) e dois métodos que serão chamados para registrar as mudanças nas variáveis - *OnShot()* e *Goal()*. O primeiro será registrado como um evento do jogo, pois quando o jogador chutar, todos têm de saber que houve o chute (goleiro, defesa e o *GameManager!*). O segundo também pode ser um evento do jogo, mas como só está modelando as cobranças de pênalti no momento, apenas o *GameManager* precisa saber do gol para incrementar a variável score e reiniciar os objetos para uma nova cobrança de pênalti. Assim, o *GameManager* pode ser implementado como no Código 03. O atributo *OnShot* de *soccer* é um evento em que você cadastrou o método *OnShot()* do *GameManager* quando ele ocorrer.

**Código 03** – Código completo do *GameManager*

```
public class GameManager : MonoBehaviour {

    public int score = 0;
    public int shots = 0;

    public GameObject soccer;
    public GameObject goalKeeper;
    public GameObject ball;
    public GameObject goal;

    public static GameManager instance { get; private set; }

    void Awake() {
        if (instance != null && instance != this) {
            Destroy(gameObject);
        }
        instance = this;
    }

    void Start () {
        soccer = GameObject.Find("soccerPlayer");
        goalKeeper = GameObject.Find("goalKeeper");
        ball = GameObject.Find("ball");
        goal = GameObject.FindWithTag("Goal");

        // registra “escutador” para evento de chute
        SoccerPlayer sp = soccer.GetComponent<SoccerPlayer>();
        sp.OnShot += OnShot;
    }

    public void OnShot() {
        shots++;
    }

    public void Reset() {
        soccer.GetComponent<SoccerPlayer>().Reset();
        goalKeeper.GetComponent<GoalKeeper>().Reset();
        ball.GetComponent<Ball>().Reset();
    }

    public void Goal() {
        score++;
        Reset();
    }
}

```

O *script* do goleiro usará os comportamentos de navegação para “pular” para direita ou esquerda (escolhido de forma aleatória) quando o jogador chutar a bola. Você precisará, então, de um atributo *target*, que armazenará a posição para onde ele pulará, e um atributo para indicar quando ele pulou: *jumped*. Sim, porque só usará o comportamento de navegação quando ele pular.

Como no *GameManager*, você ficará “prestando atenção” quando o evento de chute ocorrer. O goleiro deve optar aleatoriamente para um dos lados do gol e defini-lo como alvo do comportamento de navegação. Coloque também um *collider* verificando quando ocorrer o evento dele (tocar na bola). Caso isso ocorra, chame a reinicialização do *GameManager* porque a bola foi agarrada pelo goleiro e um novo pênalti deve ser cobrado.

O Código 04 apresenta a implementação do goleiro. A posição para a qual o goleiro irá “pular” é definida no método *OnShot()*, baseada no objeto da área do gol. Consulte o retângulo que envolve o objeto (bounds) e defina dois pontos. O mais acima (max.y) estará à direita do goleiro e o mais abaixo (min.y) estará à esquerda dele.

**Código 04** – Código completo do goleiro (GoalKeeper)

```
public class GoalKeeper : MonoBehaviour {

    private SteeringBehavior steer;
    private Vector2 target;
    private bool jumped;

    void Start() {
        steer = GetComponent<SteeringBehavior>();

        // registra “escutador” para evento de chute
        SoccerPlayer.OnShot += OnShot;

        Reset();
    }

    void OnShot() {
        GameObject goal = GameManager.instance.goal;
        Bounds bounds = goal.GetComponent<BoxCollider2D>().bounds;
        Vector2[] sides = {
            new Vector2(transform.position.x, bounds.max.y), // direita
            new Vector2(transform.position.x, bounds.min.y)  // esquerda
        };
        int sideId = Random.value < 0.5? 0 : 1;
        target = sides[sideId];
        jumped = true;
    }

    public void Reset() {
        Rigidbody2D body = GetComponent<Rigidbody2D>();
        body.rotation = 180;
        steer.Reset();
        jumped = false;
    }

    void FixedUpdate () {
        if (jumped) {
            Vector2 steering = steer.Seek(target);
            steer.ApplySteering(steering);
        }
    }

    void OnTriggerEnter2D(Collider2D coll) {
        if (coll.gameObject.name == "Ball") {
            GameManager.instance.Reset();
        }
    }
}

```

Em relação ao cobrador de pênalti, precisa-se inicialmente definir o evento de chute. Quando o cobrador colidir com a bola, ele disparará o evento, fazendo com que o *GameManager* e o *GoalKeeper* "escutem" o evento. Além disso, você precisará também de um atributo para controlar quando o cobrador vai chutar a bola. Isso porque se você quiser que ele use o comportamento de navegação *Arrival* para ir em direção à bola (alvo do comportamento), deve fazer com que ele pare de correr atrás dela depois do chute. Por fim, para facilitar a configuração da força do chute, coloque um atributo público com essa finalidade. As demais escolhas são similares ao *GoalKeeper*.

**Código 05** – Código completo do cobrador de pênalti (SoccerPlayer)

```
public class SoccerPlayer : MonoBehaviour {

    public delegate void ShotAction();
    public event ShotAction OnShot;

    private bool kicked;
    public float kickForce;
    private SteeringBehavior steer;
    private Vector2 target;

    void Start () {
        steer = GetComponent<SteeringBehavior>();
        Reset();
    }

    public void Reset() {
        kicked = false;
        steer.Reset();
    }

    void FixedUpdate () {
    if (!kicked) {
            GameObject ball = GameManager.instance.ball;
            target = ball.transform.position;
        }
        Vector2 steering = steer.Arrival(target);
        steer.ApplySteering(steering);
    }

    void OnTriggerEnter2D(Collider2D coll) {
        if (coll.gameObject.name == "Ball") {
        GameObject goal = GameManager.instance.goal;
        Bounds bounds = goal.GetComponent<BoxCollider2D>().bounds;

        Vector2[] sides = {
            new Vector2(bounds.center.x, bounds.max.y), // direita
            new Vector2(bounds.center.x, bounds.min.y)  // esquerda
        };
        int sideId = Random.value < 0.5? 0 : 1;
        Vector2 side = sides[sideId];

        Vector2 force = side - coll.attachedRigidbody.position;
            coll.attachedRigidbody.AddForce(force * kickForce);
            kicked = true;
            OnShot(); // dispara o evento
        }
    }
}

```

# 2. Descobrindo padrões

Você conhece o ditado popular “a beleza está nos olhos de quem vê?” Pois bem, você pode adaptar essa frase para a área da inteligência artificial dizendo que:

**A inteligência está nos olhos de quem vê**

Ou será que é **mais do que os olhos podem ver? :)**

**Vídeo 01** - *Transformers: Generation* 1

**Fonte:** Disponível em: https://www.youtube.com/watch?v=5zFLm8bpAN8. **Acesso em:** 26 abr 2018.

O que se quer dizer com isso?

Inicialmente, dizer que o conceito de inteligência é muito relativo. Por exemplo, você pode dizer que um cachorro é inteligente? Sim. Alguns cachorros parecem que entendem o que a gente diz. Por outro lado, eles não conseguem jogar xadrez... mas isso não coloca em dúvida a inteligência do cachorro porque não se espera ver cachorros jogando xadrez! O cachorro é considerado inteligente quando a gente pede algo e ele atende, quando nos acompanha sem precisar usar a coleira, quando sente que você está triste e fica sempre perto de você, ou seja, quando ele se comporta como um “cachorro inteligente”. Se ele começar a jogar xadrez, pelo menos eu não vou pensar “Olha! Que cachorro inteligente!”. Vou imaginar que se trata de um dos alienígenas do filme *Homens de Preto...* Ou não?

**Fonte**: Disponível em https://imgur.com/gallery/THN7ejn Acesso em: 26 abr. 2018.

Enfim, o conceito de inteligência está relacionado a uma capacidade ou a um comportamento esperado. Por isso, adapte o ditado popular para dizer que a inteligência está nos olhos de quem vê.

As ferramentas estatísticas, entre elas, o uso de **Probabilidade**, são muito importantes para criar personagens “inteligentes aos nossos olhos” porque nos ajuda a medir o que o usuário “está esperando”.

Como assim?

Vou dar um exemplo através do jogo que você criou. Sendo você um goleiro e fosse tentar segurar um pênalti de um jogador que bateu praticamente todos seus pênaltis no lado direito do gol, para qual lado você pularia?

Como você é “inteligente”, pularia para o lado direito, não é mesmo? 😁 Espera-se, então, que, no jogo, um personagem aja da mesma forma. Caso ele não pule, aos nossos olhos, o personagem não será “inteligente”. O objetivo da IA nos jogos é simular inteligência nos personagens em função do comportamento que o jogador espera deles. Precisa-se, então, fazer com que o goleiro aprenda para qual lado pular em função do padrão de chutes do jogador, e pode-se fazer isso com o uso do conceito de probabilidades.

## 2.1 Descobrindo valores numéricos através da Probabilidade

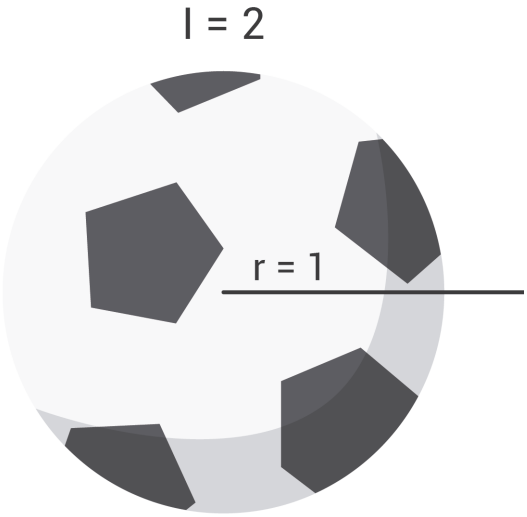
---

Nesta seção, irei exemplificar como a descoberta de padrões pode funcionar. O mecanismo é similar, tanto para padrões de comportamento em jogos, como padrões numéricos. Veja a seguir como se faz a descoberta do valor de PI por esse mecanismo.

**PI** (ou $\pi$) é uma constante numérica que representa a relação de proporção entre o comprimento de uma circunferência e seu diâmetro. Essa constante é utilizada em vários cálculos, entre eles o da área de um círculo, cuja fórmula é $\pi . r^2$, em que $r$ é o tamanho de seu raio. Se você não souber o valor de $\pi$, como poderá usar a probabilidade para calcular seu valor?

A ideia é simples. Sabe-se pela fórmula que um círculo de raio igual a 1 tem área igual a $\pi . r^2 = \pi . 1^2 = \pi$, e que a área de um quadrado envolvendo esse círculo é 4, pois, como o lado do quadrado é igual ao diâmetro do círculo, a área do quadrado é $I^2 = 2^2 = 4$. Na Figura 02 você pode ver a descrição desses elementos. Agora, pode-se achar o valor de $\pi$ respondendo a seguinte questão: se for escolhido um ponto aleatório dentro do quadrado, qual a probabilidade de ele estar também dentro do círculo?

**Figura 02** - Círculo (bola de futebol) de raio igual a 1 envolvida em um quadrado de lado igual a 2




Como não se tem o valor de $\pi$, pois o objetivo é calculá-lo, não se tem por enquanto uma resposta direta para o questionamento feito anteriormente. É preciso calcular essa probabilidade de forma dinâmica. Há uma relação entre a área do círculo e a área do quadrado que será útil nesse cálculo. Então, gere vários pontos aleatórios dentro do quadrado e conte quantos deles caem dentro do círculo. Quanto mais pontos gerar, mais precisa será a probabilidade calculada.

Para entender a relação, imagine que o número total de pontos gerados aleatoriamente seja $T$ e o número de pontos que caíram dentro do círculo seja $C$. A proporção definida pelo número de pontos que caíram no círculo e o total de pontos gerados $\frac{C}{T}$ está diretamente associada à proporção entre a área do círculo e a área do quadrado. Denominando a área do círculo de $A_c$ e a área do quadrado de $A_q$, temos a relação:

$$\frac{C}{T} = \frac{A_c}{A_q}$$

Sabendo que é $\frac{C}{T}$ a probabilidade de o ponto gerado estar dentro do círculo (relação entre número de eventos específicos e o total de eventos possíveis), a área do círculo é $\pi$ e a do quadrado é 4, tem-se então a seguinte relação:

$$P_c = \frac{\pi}{4} \Rightarrow \pi = 4.P_c$$

Ou seja, você pode calcular o valor de $\pi$ multiplicando a probabilidade de o ponto estar dentro do círculo por 4. Não acredita? Então faça a atividade a seguir para conferir.

## Atividade 02 – Cálculo de PI

Inicialmente, você vai alterar qualquer um dos objetos de jogo previamente definidos, inserindo alguns atributos públicos e alterando também o método *Update()*. Pode-se fazer a alteração, por exemplo, no objeto *GameManager*.

Insira os atributos **pi**, **c** e **t** para armazenarem respectivamente o valor de $\pi$, o número de pontos dentro do círculo e o número total de pontos gerados. Deixe-os públicos para poder ver seus valores enquanto a simulação é executada.

Você precisará usar também o mecanismo de geração de números aleatórios da linguagem C#. Opte em usar essa linguagem, e não o que o Unity fornece, para usar o tipo *double* e ter uma precisão maior.

No método *Update()*, você gerou duas coordenadas (x e y) com valores entre -1 e 1. Ou seja, as coordenadas definem um ponto dentro de um quadrado de lado de tamanho 2. Verifique depois se a distância desse ponto para o centro do quadrado

(0, 0) é menor ou igual a 1. Se for, é porque o ponto encontra-se dentro do círculo inscrito dentro do quadrado, pois a distância é menor ou igual ao seu raio. Nesse caso, o contador c precisa ser incrementado. O contador **t** é incrementado a cada atualização, uma vez que um novo ponto é testado a cada chamada do método, e **pi** é calculado a partir dos contadores mencionados. O Código 06 mostra os detalhes da solução.

**Código 06** – Alterações em *GameManager* para calcular o valor de PI usando o conceito de probabilidade

```
1   public class GameManager : MonoBehaviour {
2      ...
3      public double pi;
4      public int c = 0;
5      public int t = 0;
6      private System.Random rnd;
7
8      void Start () {
9         ...
10     rnd = new System.Random();
11     }
12
13     public void Update() {
14        double x = 2 * rnd.NextDouble() - 1;
15        double y = 2 * rnd.NextDouble() - 1;
16        if (Math.Sqrt(x*x + y*y) <= 1)
17           c++;
18        t++;
19        pi = 4 * (double) c / t;
20     }
21  }
22
```

Execute o programa com o objeto *GameManager* selecionado e você verá o valor de **pi** convergindo paulatinamente para 3,14. Note que, para convergir, é necessário realizar milhares de chamadas do método *Update()* (no meu teste, o valor convergiu para 3,14 quando o contador **t** estava na casa de 15.000). Quanto mais pontos gerar, mais precisa será a probabilidade calculada e, consequentemente, mais casas decimais definidas você encontrará para o valor de $\pi$ .

# 2.2 Descobrindo padrões de comportamento no jogo

Da mesma forma que você fez a descoberta do valor de $\pi$, também pode descobrir padrões de comportamento, seja do jogador ou de um outro personagem virtual. Isso é muito importante para vários aspectos do jogo. Por exemplo, descobrindo padrões de comportamento do jogador, você pode ajustar o nível de dificuldade do jogo de forma a manter o jogador em estado de *flow* ((Você estudou o conceito na disciplina de Design)).

De acordo com essa abordagem, é importante apresentar desafios conforme as habilidades do jogador, de forma que eles não sejam nem muito difíceis, a ponto de desestimular o jogador, nem muitos fáceis, a ponto de perder a graça por não ter algo a ser “conquistado”.

A descoberta de padrões facilita, então, a implementação de mecanismos que ajudem o jogo a se adaptar ao jogador. Se o jogador for muito bom, você irá, por exemplo, melhorar a precisão dos passes ou a capacidade de dar chutes certeiros, lá no cantinho do gol, e assim por diante.

Nessa aula, você irá aprimorar o desempenho do seu goleiro na penalidade máxima. Ele pulará para a direita ou para a esquerda em função de como outro personagem jogador cobra seus pênaltis. Se o jogador chutar muitas vezes para o lado direito, você tem de adaptar o goleiro para pular mais para o lado direito. Na atividade 1, você colocou um padrão de chute 50-50, ou seja, há 50% de probabilidade de o jogador chutar para a esquerda e 50% de chutar para a direita. Se mudar o padrão para 90-10, ele chutará muito mais para um lado do que para o outro. Você quer, então, que o goleiro vá aprendendo, ou seja, vá descobrindo o padrão à medida que ele for chutando.

Uma forma para fazer isso é começar com o padrão de pulo do goleiro de 50-50 e ir ajustando à medida que o jogador for chutando. O ajuste é feito através da fórmula da probabilidade descrita anteriormente.

$$P = \frac{evento\ específico}{todas\ opções\ de\ eventos\ possíveis}$$

Ou seja, você terá a probabilidade de o goleiro pular para a esquerda e direita descrita pelo padrão $P_e - P_d$ por meio das fórmulas:

$$P_e = \frac{número\ de\ chutes\ à\ esquerda}{número\ de\ chutes\ total} P_d = \frac{número\ de\ chutes\ à\ direita}{número\ de\ chutes\ total}$$

Bom, como no cálculo de $\pi$, a descoberta dos valores de probabilidade pode demorar a convergir. Ou seja, não é necessariamente depois de dois ou três eventos que os padrões são descobertos. Dependendo do jogo ou da aplicação que você estiver desenvolvendo, o número de eventos necessário pode ser bastante grande. Mas como esse caso apresenta apenas duas possibilidades, a convergência é bem mais rápida do que o cálculo do $\pi$.

## Atividade 03 – Ajuste do pulo

Altere o código da atividade 01 para que o padrão de pulo do goleiro vá se ajustando em função dos chutes do cobrador de pênaltis. Execute uma simulação de cobranças de pênaltis e veja se o goleiro começa a segurar mais os pênaltis depois de um certo número cobrado.

Agora é com você!

**Código 07** – Código para o goleiro aprender o lado mais chutado

```
1   public class GoalKeeper : MonoBehaviour {
2       ...
3       //Variável para guardar a tendência de chutes para a direita, começa com 50%
4       public float tend = 0.5
5
6       ...
7       void OnShot () {
8           ...
9           //tend controla a porcentagem da escolha agora
10          int sideId = Random.value < tend ? 0 : 1;
11      ...
12      }
13
14      public void Reset() {
15      ...
16          //Cálculo da tendência de chute, feito a cada três cobranças
17          if (GameManager.instance.shots > 0 && GameManager.instance.shots % 3 == 0) {
18              tend = (float)GameManager.instance.rights / (float)GameManager.instance.shots;
19          }
20      ...
21      }
22  }
23
```

# 3. Detalhando a ação a partir de um erro

Muito bem! O jogador está optando por um lado para chutar e o goleiro está aprendendo para onde pular, mas estão faltando alguns detalhes. Mesmo que o jogador chute para direita, há maneiras diferentes de fazer isso. Ele pode chutar bem colocado no cantinho, pode dar um tiro forte, pode ser rasteiro, pode ser por cima e assim por diante. Além disso, se você quiser manter um nível de realismo aceitável, a precisão do chute pode ir desde o “chute perfeito” ao “chute à la Roberto Baggio” na copa de 94, quando o Brasil foi tetracampeão, ou os chutes da seleção brasileira na Copa América de 2011, tanto faz, foram no mesmo nível (Confira no *YouTube*).

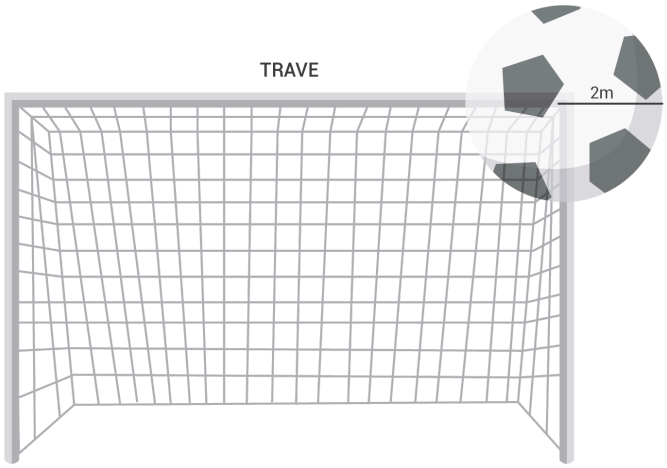
Ou seja, você precisa ter um nível de erro associado às ações dos jogadores. Que podem ser de qualquer tipo, como por exemplo, a cobrança de pênaltis, passes para companheiros, cobrança de escanteio, tiro de meta, alvo do deslocamento, entre outros.

Note que a escolha que a IA precisa fazer, a princípio, não são valores dentro de um conjunto nominal de possibilidades, como “direita” ou “esquerda”. A escolha pelo nível de erro é um parâmetro dentro de uma faixa contínua de valores, como por exemplo o intervalo de números reais de 0 a 10. Existem infinitas possibilidades dentro desse intervalo e você precisa escolher uma delas. Assim, devemos tratar de forma diferente as escolhas em conjuntos discretos (ex.: “direita” e “esquerda”) das escolhas em conjuntos contínuos (ex.: números reais de 0 a 10).

O primeiro passo a ser tomado para fazer essa escolha é estabelecer qual a faixa de valores que definem os limites da ação. Ou seja, é necessário definir dois parâmetros: um referente à “ação perfeita” e o segundo à “ação desastrosa” (à la Roberto Baggio). Por exemplo, imagine que o cobrador de pênalti queira chutar no canto superior direito da trave. A ação perfeita corresponde a exatamente onde ele quer colocar a bola, enquanto a ação mais desastrosa corresponde, por exemplo, a bola ir numa direção a 2 metros de onde ele quer colocar.

A Figura 03 ilustra esse exemplo através da área que delimita um possível chute no intervalo de 0 a 2 metros de erro.

**Figura 03** - Faixa de erro de 2m de um chute no canto superior direito da trave



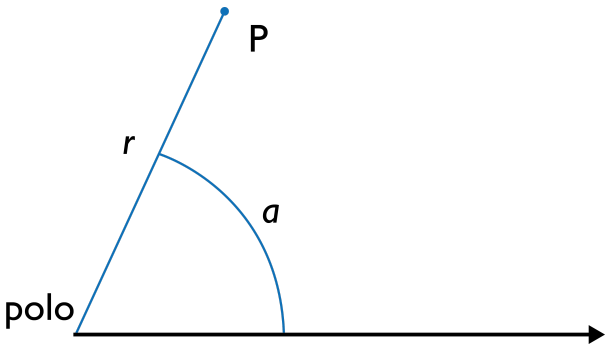

O exemplo da figura não corresponde exatamente à cobrança de pênaltis do jogo porque ele encontra-se numa “visão de cima”. Então, a faixa de erro não é um círculo como demonstrado. Porém, decidi representar esse tipo de situação porque ela é similar a diversas que são encontradas no jogo. Por exemplo, se um jogador vai

dar um passe para seu companheiro de time, ele pode mirar a bola em um determinado alvo, mas errar de um certo valor que corresponde à área de um círculo.

A partir do intervalo definido, você pode usar diferentes técnicas para escolher qual o nível de erro. A técnica mais simples é gerar um valor qualquer dentro do intervalo. No exemplo dado, seria um número real entre 0 e 2. Nesse exemplo, o nível de erro em um círculo pode ser em qualquer ponto dele. Use esse valor em um sistema de coordenadas polares, de forma que possa definir exatamente o ponto onde a bola iria no chute.

O **sistema de coordenadas polares** tem o mesmo propósito do sistema de coordenadas cartesianas, que é definir pontos em um plano. A diferença encontra-se na forma como os pontos são definidos. Enquanto no sistema cartesiano um ponto é definido por valores em dois eixos (x e y), no sistema polar um ponto é definido por uma distância à origem e um ângulo de rotação. Por exemplo, o ponto **P** da Figura 04 é um ponto do plano definido através dos valores de **r** e **α**, que são respectivamente sua distância ao polo (origem do sistema de coordenada), denominada de **raio**, e o ângulo que ele faz com um eixo fixo em relação ao polo, chamado de **coordenada angular** ou **azimute**.

**Figura 04** - Sistema de coordenadas polares




O sistema de coordenadas polares é ideal para esse caso porque ele permite gerar facilmente um ponto aleatório dentro de um círculo. O raio corresponderia ao nível de erro escolhido aleatoriamente, cujo valor é a distância em relação ao alvo do chute, e o ângulo pode ser qualquer valor (entre 0 e 360). Quaisquer que forem os valores aleatórios gerados, você terá obrigatoriamente um ponto dentro da área desejada.

Depois que você tiver o ponto representado através de um sistema de coordenadas polares, precisa transformá-lo para o sistema cartesiano, uma vez que o Unity utiliza este último. Isso é feito através das funções trigonométricas a seguir.

$$x = r.\cos \propto y = r.\sin \propto$$

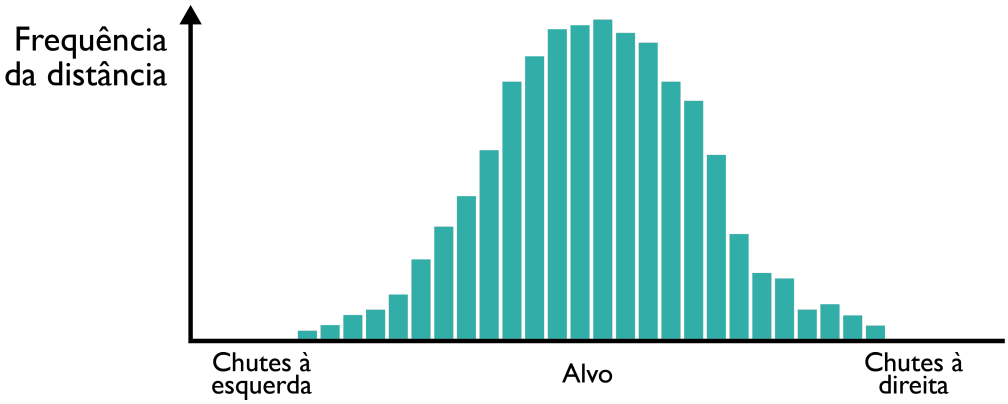
Muito bem! Você tem o ponto onde a bola irá quando o jogador chutar ou quando ele der um passe, fazer um lançamento etc. Esse ponto vai ser diferente a cada vez que a ação for realizada devido a um nível de erro inserido na ação. O erro é gerado aleatoriamente dentro de uma faixa de valores, com a mesma probabilidade de ocorrência. Isso pode ser suficiente para a maioria dos casos. Mas, para os mais atentos, essa estratégia pode ser um problema, que você terá de corrigir na próxima seção.

## Distribuição de probabilidades

O fato de gerar um valor qualquer dentro do intervalo de erro (na seção anterior) faz com que todos os valores dele tenham a mesma probabilidade de serem escolhidos. Isso faz sentido para você? Faz sentido que o maior nível de erro (aquele do chute de Roberto Baggio!) tenha a mesma probabilidade de ocorrer do que um chute mais próximo do alvo desejado?

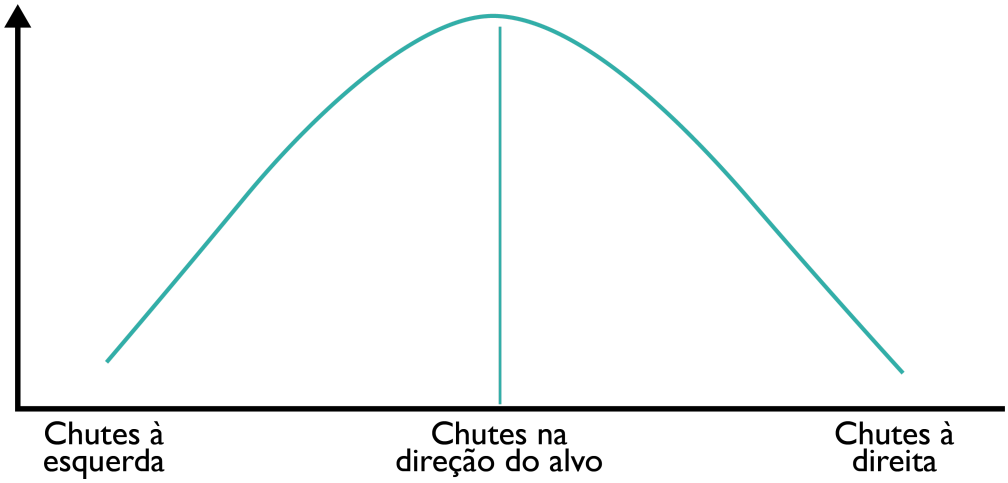
Normalmente, não é isso que acontece. Os chutes desastrosos ocorrem com menos frequência do que os chutes próximos ao alvo. Se você pedir para um jogador cobrar centenas de pênaltis, descobrirá uma distribuição nos níveis de erros dos chutes (distância do alvo) que não é igual para todos os valores. Em geral, se para cada chute dado fosse medida a distância do alvo e somadas suas frequências, provavelmente teríamos um gráfico como o histograma apresentado na Figura 05. Ou seja, os chutes desastrosos, sejam para a direita ou para a esquerda, apareceriam com menor frequência, enquanto as maiores frequências ocorreriam mais próximas do alvo.

**Figura 05** - Histograma ilustrando uma frequência da distância dos chutes em relação ao alvo




Esse histograma pode ser simplificado através de uma curva que indica uma tendência do comportamento dos chutes do jogador. É claro que ela pode ser diferente para cada um. Para alguns, a média estará mais próxima do chute perfeito, enquanto para outros (e eu me incluo nesse grupo 😁 ) a média estará bem mais próxima do chute desastroso. Como essa curva representa a frequência de ocorrências de um evento (no seu caso, o erro do chute), ela está diretamente relacionada à probabilidade de esse evento ocorrer. Por exemplo, se após 100 chutes, 90 foram a menos de 50 cm do alvo, pode-se inferir que a probabilidade de o jogador chutar nesse intervalo é de 90%, enquanto a de chutar a bola a uma distância maior ou igual a 50 cm é de 10%. Veja na Figura 06 a curva de distribuição de probabilidade das tentativas dos chutes do jogador.

**Figura 06** - Curva de distribuição de probabilidade




Por isso, diz-se que essa é uma curva de **distribuição de probabilidades**. Para simplificar o método, considere-a como uma **curva de distribuição normal**, também conhecida como **distribuição de Gauss** ou **Gaussiana**. Essa curva possui

características muito interessantes para esse caso.

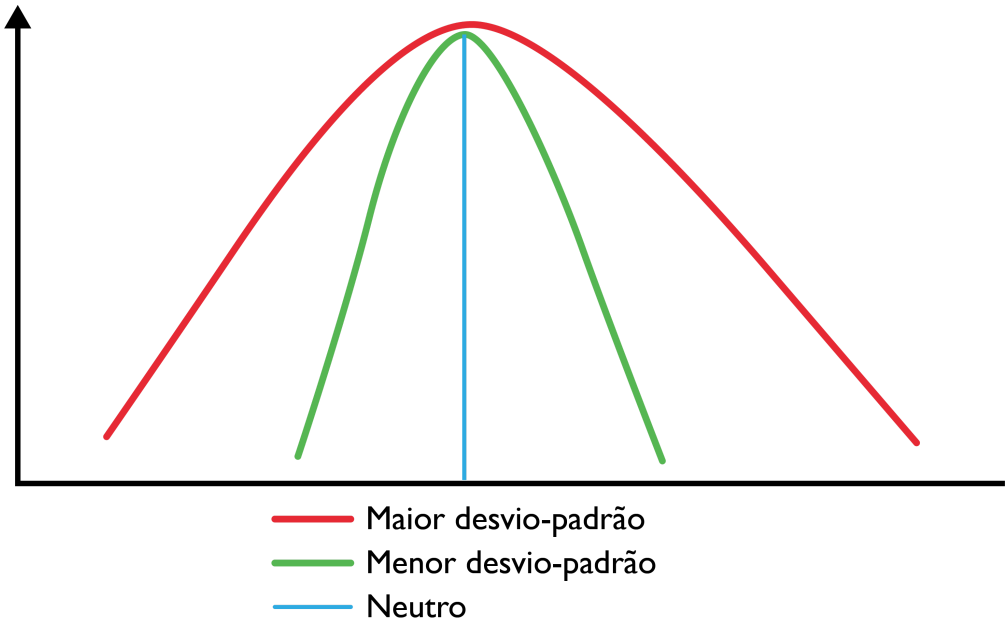
A **distribuição normal** é muito importante em várias áreas do conhecimento, seja para descrever fenômenos físicos ou até da economia. Porém, nesse caso, ela será útil para descrever de forma mais realista as ações dos jogadores sem precisar pedir que eles chutem centenas de vezes para ficar medindo seus erros. Uma das características dessa curva é que ela pode ser descrita completamente através de dois parâmetros: a **média** e o **desvio padrão**. Conhecendo esses dois valores, você pode determinar qual a probabilidade de qualquer intervalo de erro ocorrer.

Claro que você sabe o que é a média, mas o que é desvio padrão?

**Desvio padrão** é uma medida para representar o quão a boca do sino (fiz essa analogia, pois a forma da curva lembra um sino) da Figura 06 é fechada ou aberta. Ou seja, **desvio padrão** é uma medida de dispersão de dados. Quanto mais dispersos são os dados, mais a boca do sino estará aberta. Consequentemente, quanto menos dispersos, mais fechada ela estará. Por exemplo, compare os seguintes conjuntos { 1, 2, 3, 4, 5, 6, 7, 8, 9 } e { 4, 4, 4, 5, 5, 5, 6, 6, 6 }. Ambos, possuem a mesma média: 5. Porém, os dados do primeiro conjunto estão mais dispersos que o do segundo. Assim, o **desvio padrão** do primeiro é maior do que o do segundo e, por conseguinte, a “boca do sino” estará mais aberta no primeiro.

A Figura 07 ilustra duas curvas com mesma média, mas com desvios padrões diferentes, e, como consequência, a amplitude da “boca do sino” também.

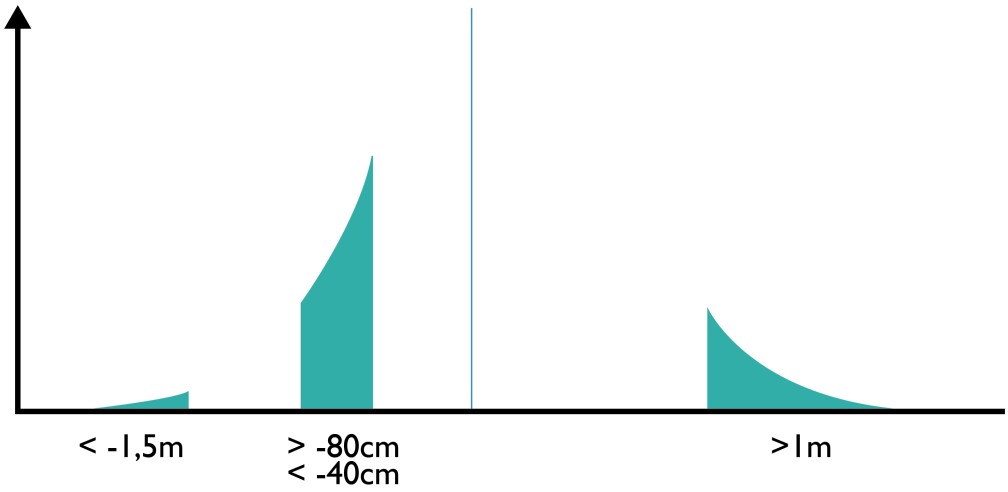
**Figura 07** - Exemplo de duas curvas de distribuição normal com mesma média, mas com desvios padrões diferentes




Assim, se considerar que os chutes dos jogadores seguem uma distribuição normal, usando a **média** e o **desvio padrão** de um jogador, você poderá extrair a probabilidade do nível de erro de um chute dele ocorrer. Bom, na verdade, como você está lidando com valores contínuos, não dá para saber a probabilidade de um erro exatamente, mas de um intervalo de erro. Por exemplo, não dá para saber a probabilidade exata do chute a 40 cm do alvo, mas se pode extrair a probabilidade de um chute sair a mais de 1 m de distância à direita do seu alvo (intervalo definido pelos valores maiores do que 1), bem como a probabilidade de seu chute sair a mais de 1,5 m à esquerda do alvo (intervalo dos valores menores que -1,5), ou de a distância do alvo estar entre 80 e 40 cm à esquerda do alvo (intervalo dos valores maiores que -80 e menores que -40).

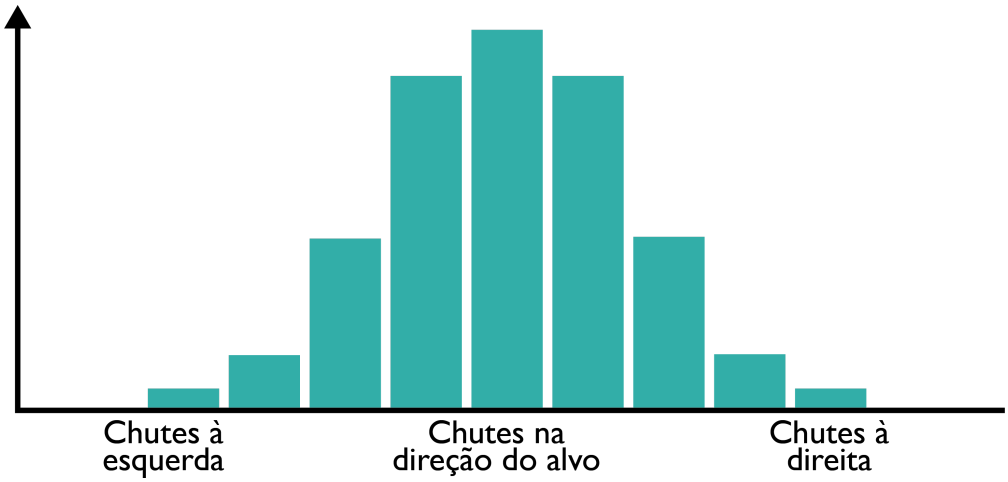
A Figura 08 ilustra a probabilidade desses intervalos. É importante ressaltar que, na curva de **distribuição normal**, a probabilidade de um intervalo é definida pela área da curva delimitada pelo intervalo. Assim, a área total da curva é de 100%.

**Figura 08** - Exemplos de intervalos em que se pode extrair a probabilidade em uma curva de distribuição normal




Para calcular a área de um intervalo, consulte uma tabela que contém os valores de **distribuição normal**, para simplificar o processo e tentar uma abordagem mais direta ao problema que você está resolvendo (simular o erro das ações dos jogadores). A simplificação é usar sua própria tabela, porém seguindo as características de uma **distribuição normal**. Por exemplo, você pode criar uma tabela que corresponda à curva de distribuição presente na Figura 09. Assim, você teria intervalos de erros e a probabilidade associada a cada intervalo.

**Figura 09** - Simplificação da distribuição de probabilidades




Como você percebeu, foi transformada a curva de distribuição em valores discretos, definidos pelos seus respectivos intervalos, como exemplificado no Quadro 01. Porém, você fez isso baseado nos conceitos de **distribuição normal**, há

muito tempo utilizados com sucesso para representar fenômenos estatísticos. Então, espere que o comportamento dos personagens virtuais tenha o mesmo sucesso.

**Quadro 01 -** Exemplo de quadro de distribuição de probabilidade sobre os erros

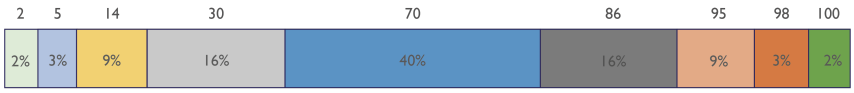
| Distância (cm) | -200 | -150 | -100 | -50 | 0 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|---|---|---|---|---|
| Probabilidade | 2% | 3% | 9% | 16% | 40% | 16% | 9% | 3% | 2% |

Lembre-se de que os valores negativos do Quadro 01 representam a distância da bola que foi à esquerda, enquanto os valores positivos representam a distância à direita. Dado o quadro, o que você precisa fazer é aplicar o mesmo procedimento das escolhas entre direita e esquerda feito anteriormente. Só que, agora, a escolha é uma sequência de intervalos, cada um tendo uma probabilidade associada. Note que, da mesma forma que a escolha entre direita e esquerda, a soma das probabilidades das possíveis opções necessita ser obrigatoriamente igual a 1 (ou seja, 100%).

Mas como tirar um valor dentro de um conjunto de possibilidades, cada um com uma probabilidade diferente, uma vez que as linguagens de programação em geral nos dão um comando para tirar um valor sem probabilidade associada (ou seja, todos os valores têm a mesma probabilidade de serem sorteados)?

A maneira mais simples é reescrever o Quadro, usando o somatório das probabilidades, como ilustrado na Figura 10. Usando essa estratégia, você pode usar o comando de geração de números aleatórios da linguagem e verificar em qual intervalo um valor aleatório se encontra. O tamanho do intervalo é que definirá o valor da probabilidade associado. No exemplo dado, um valor entre 0 e 2 tem 2% de chance de ocorrer, um valor entre 2 e 5 tem 3% de chance, um valor entre 5 e 14 tem 9%, um valor entre 14 e 30 tem 16% e assim por diante. O novo Quadro terá, portanto, um novo dado contendo o limite do intervalo, tal como apresentado no Quadro 02.

**Figura 10** - Reescrita das probabilidades na faixa de 0 a 100 através do somatório da probabilidade de cada intervalo




**Quadro 02 -** Comparação entre os métodos de partida de motores elétricos.

| Distância (cm) | -200 | -150 | -100 | -50 | 0 | 50 | 100 | 150 | 200 |
|---|---|---|---|---|---|---|---|---|---|
| Probabilidade | 2% | 3% | 9% | 16% | 40% | 16% | 9% | 3% | 2% |
| Lim. intervalo | 2 | 5 | 14 | 30 | 70 | 86 | 95 | 98 | 100 |

# Resumo

Ufa, quanta incerteza para uma aula só!

O uso de probabilidade é um recurso essencial na construção de jogos! Ela nos permite criar simulações que são mais próximas da realidade dando um aspecto de maior inteligência para os personagens. Através dela, também, você poderá criar eventos de forma randomizada, adicionando um elemento de surpresa interessantíssimo para a jogabilidade. :)

Afinal, ninguém acerta ou erra 100% das vezes!

Os pontos principais dessa aula foram:

- Conceito de probabilidade e como calcular a probabilidade de um evento ocorrer;
- Como usar a probabilidade dentro de um jogo, implementando-a no projeto do Unity;
- Como calcular padrões de valores e comportamentos a partir das fórmulas de probabilidade;
- Como adicionar a probabilidade para adicionar incerteza (possibilidade de erros) nas ações dos personagens;
- Conhecer as distribuições de probabilidade e como elas podem ajudar a controlar o resultado de um determinado evento probabilístico.

Você teve uma boa quantidade de conteúdo, não é mesmo? Aproveita para revisar tudo e se preparar para a próxima aula!

Até lá! 😉

# Referências

KYAW, Aung Sithu. **Unity 4. x Game AI Programming**. Packt Publishing Ltd, 2013.

BOURG, David M.; SEEMANN, Glenn. **AI for game developers.**. "O'Reilly Media, Inc.", 2004.

DAGRACA, Micael. Practical Game AI Programming. 2017.