# MÁQUINA DE ESTADOS FINITA EM JOGOS 2D

## A Inteligência Artificial do comportamento autônomo para um personagem

GONÇALVES, Rafael José[1]
JÚNIOR, Renée de Araújo[1]
ROQUE, Ionara Antônia[1]
SANTOS, Jhonattan Lucas dos[1]
SANTOS, Flávia Aparecida Oliveira[2]

[1]Acadêmico do curso de Ciência da Computação - Unifenas / Alfenas
[2]Profa. Dra. do curso de Ciência da Computação - Unifenas / Alfenas

**RESUMO**

Esse artigo científico apresenta uma das diversas técnicas de inteligência artificial para jogos 2D e possui como objetivo mostrar os elementos básicos para desenvolvimento do comportamento autônomo de um personagem em um jogo. Embora existam diversas técnicas para criação de comportamentos autônomos quando se trata de jogos, a pesquisa busca mostrar à eficiência de uma das técnicas mais tradicionais utilizadas. A elaboração da pesquisa ocorreu por intermédio de levantamento teórico em livros e documentação online na área e o desenvolvimento da aplicação foi realizado com à engine de jogos Unity. Esses materiais foram importantes para compreensão e entendimento no assunto. A Máquina de Estados Finita desenvolvida apresenta ao todo sete estados, que funcionam levando em conta o nível de saúde no qual o personagem se encontra, tendo este um comportamento mais agressivo em relação ao jogador, conforme seu nível de saúde fica menor. Foi possível identificar a facilidade em se desenvolver comportamentos autônomos com a classe State Machine Behaviour do Unity, além da possibilidade de combinação com outras técnicas de inteligência artificial para geração de um comportamento mais interativo. Esses pontos observados servirão de base para novos projetos e aprofundamento de pesquisas em oportunidades futuras.

**ABSTRACT**

This paper presents one of several techniques of artificial intelligence for 2D games and aims to show the basic elements to develop the autonomous behavior of a character in a game. Although there are several techniques for creating autonomous behaviors when it comes to games, the research seeks to show the efficiency of one of the most traditional techniques used. The elaboration of the research was done by means of theoretical research in books and online documentation in the area and the development of the application was carried out with the Unity game engine. These materials were important for understanding and understanding the subject. The developed Finite States Machine has seven states, which work by taking into account the health level in which the character is, which has a more aggressive behavior towards the player, as his health level is lower. It was possible to identify the ease in developing autonomous behaviors with the Unity State Machine Behavior class, as well as the possibility of combining with other artificial intelligence techniques to generate a more interactive behavior. These points will serve as the basis for new projects and further research on future opportunities.

**PALAVRAS-CHAVE**

Inteligência artificial, Comportamento autônomo, Máquina de estados finita, Jogos 2D.

**KEYWORDS**

Artificial Intelligence, Autonomous Behaviour, Finite State Machine, 2D Games.

## 1 INTRODUÇÃO

A Máquina de Estados Finita (FSM –Finite State Machine), é uma das técnicas de inteligência artificial mais utilizadas por desenvolvedores de jogos eletrônicos. É uma das formas de representação mais comum para comportamento de personagens em um jogo. Uma Máquina de Estados Finita é composta por um conjunto de estados e um conjunto de regras de transição entre esses estados, refletindo em certo momento, algum evento no mundo do jogo.

A utilização dessa técnica para definir o comportamento dos personagens em um jogo é bastante tradicional devido ao fato da necessidade de pouco processamento e de ser de fácil compreensão, principalmente com o auxílio de ferramentas visuais, como é o caso da engine de jogos Unity. Atualmente jogos como: Doom e Call of Duty, utilizam Máquina de Estados Finita em diversos personagens nos seus respectivos ambientes de jogos, além de clássicos como: Medal of Honor, Killzone, Super Mario World, Pacman, entre outros utilizaram essa técnica em seus personagens, quando foram desenvolvidos.

O objetivo é mostrar a técnica da Máquina de Estados Finita, desenvolvendo uma pequena aplicação em cima do assunto abordado, onde será construído uma estrutura com o comportamento inteligente, ou seja, a inteligência artificial que um chefe de fase possuí em um jogo. À estrutura básica desenvolvida deverá oferecer possibilidades para novas abordagens e utilização em novos projetos, além da integração com outras técnicas de inteligência artificial para jogos eletrônicos.

Existem vantagens e desvantagens na Máquina de Estados Finita, como toda técnica de inteligência artificial, sendo está apropriada em determinadas situações, que ficaram visíveis ao longo deste presente artigo.

## 2 METODOLOGIA

### 2.1 O que é Inteligência Artificial?

De acordo com PRADO (2016), para entender o que exatamente é esse termo, precisamos saber à origem da Inteligência Artificial, que ocorreu em meados de 1956. A Inteligência Artificial surgiu como uma área da Ciência da Computação onde os pesquisadores buscavam realidades diferentes de um modelo simplesmente programado. Com o intuito de não existir somente para resolver problemas simples, mas, para criar uma espécie de pensamento na computação, através do conhecimento acumulado.

TECHOPEDIA (2018), Inteligência Artificial ou Inteligência Computacional é uma área da Computação Natural que estuda a natureza do homem de resolver problemas com o objetivo de desenvolver um sistema computacional capaz de resolver problemas complexos, sendo essa uma área de pesquisa repleta de desafios, abrangendo as redes neurais artificiais, teoria dos conjuntos nebulosos, computação evolutiva , dentre outras.

Espera-se que a Inteligência Artificial encontre respostas em grande parte dos problemas, com a capacidade de encontrar a melhor solução sem cometer erros, o que auxilia e agiliza a realização de diversas tarefas.

Segundo PRADO (2016), a Inteligência Artificial se ramifica em diversas áreas, indo de games até à filosofia e podemos imaginar essa ciência sendo aplicada basicamente em tudo.

## 2.2 Inteligência Artificial em jogos

Para YANNAKAKIS (2014), à Inteligência Artificial para jogos não é considerada a mesma que a estudada por pesquisadores do meio acadêmico. No começo do desenvolvimento de jogos eletrônicos, a programação da IA era mais era conhecida por "programação de jogabilidade", pois não havia nada de inteligente sobre os comportamentos exibidos pelos personagens controlados pelo computador.

Segundo SELECT GAME (2010), muitos programadores no início da era de jogos eletrônicos implementavam padrões de movimentos ou movimentos repetitivos e/ou aleatórios para os personagens controlados pelo computador como sendo a inteligência existente no jogo. Isso ocorria principalmente pela falta de memória e limitação existente na velocidade de processamento da época. Os jogos de estratégia estão entre os pioneiros em IA para jogos, uma vez que tais jogos necessitam de uma boa IA para que o computador controle grupos de personagens com estratégias e táticas. Em jogos de tiro de primeira pessoa, conhecidos pelo termo FPS (First Person Shooter), a IA ficou conhecida pelo excelente nível tático dos inimigos, desenvolvida através do uso de Máquina de Estados Finita e Scripts que determinam como um agente inteligente deve agir em várias situações.

A Inteligência Artificial no caso dos jogos, nem sempre é utilizada com o intuito direto de encontrar a melhor solução para vencer o jogador, pois, nesse caso, o computador pode se tornar invencível. Nos jogos é importante dar a impressão de que o computador está realmente competindo com o jogador, podendo haver uma margem de erros para que a jogabilidade seja mais realista. A Inteligência Artificial também pode ser muito útil na área de jogos educativos, fazendo com que o jogo vá percebendo as dificuldades do usuário e focando os problemas nessas dificuldades.

O mercado de desenvolvimento de jogos no Brasil cresce mais a cada dia, devido a várias parcerias feitas com interessados em investir na área. Atualmente essa área exporta seus produtos nacionais para cerca de 200 tipos de mercados, dentre os mais variados públicos. A área de jogos atualmente supera a indústria da música e do cinema juntos, e é uma área com grandes possibilidades para o Brasil futuramente (ABRA GAMES, 2018).

## 2.3 Diferença entre IA acadêmica e IA para jogos

De acordo com TOGELIUS (2018), a principal diferença entre a IA acadêmica e a IA para jogos é o objetivo que cada uma busca. No primeiro caso, o objetivo é buscar a solução para problemas extremamente difíceis, como imitar o reconhecimento que os humanos são capazes de realizar, entender e construir agentes inteligentes. No segundo caso, o objetivo de usar inteligência artificial é a diversão. Sua importância é quanto aos resultados que o sistema irá gerar, e não como o sistema chega até os resultados, ou seja, o problema não é como o sistema pensa, mas sim como ele age.

Para YANNAKAKIS (2014), isso acontece pelo fato que jogos eletrônicos são negócios e os consumidores desses produtos os compram em busca de diversão, e não lhes interessa como a inteligência de um personagem no jogo foi criada, desde que ela transforme o jogo divertido e desafiador.

## 2.4 O que é o Unity?

A Unity é um motor de jogo multiplataforma que permite a criação de jogos 2D ou 3D. Possui uma interface gráfica que permite desenvolver jogos com facilidade, e muitos serviços integrados que aceleram o processo de desenvolvimento. As linguagens de programação que podem ser usadas são UnityScript e C# (UNITY USER MANUAL, 2018a).

O editor da Unity também é extensível, possibilitando implementar funcionalidades ainda não existentes. Um exemplo de extensão é o Rival Theory, que implementa vários conceitos de Inteligência Artificial, desde funcionalidades básicas como pathfinding, condutas de patrulha, esconder, atacar, seguir, vaguear, até conceitos mais complicados como percepção e árvore de condutas.

**2.5 O que é o Mono Develop?**

O MonoDevelop é um ambiente de desenvolvimento integrado, que permite que os desenvolvedores gravem rapidamente aplicativos da área de trabalho e da Web no Linux, Windows e MacOS. Também torna mais fácil para os desenvolvedores portar aplicativos .NET criados com o Visual Studio para Linux e o MacOS mantendo uma única base de código para todas as plataformas (MONO DEVELOP, 2018).

**2.6 O que é o Clip Studio?**

Clip Studio Paint, é um software gráfico apoiado por criadores de mangás, quadrinhos e desenhos animados. Ele oferece recursos especializados para desenhar quadrinhos e desenhos animados, além de recursos aprimorados para colorir trabalhos (CLIP STUDIO, 2016).

**2.7 Animações em um jogo**

Segundo MILHOMEN (2016), o conhecimento dos ciclos de animação são essenciais para a criação do movimento dos diversos personagens inseridos em um jogo, sendo possível através desse conhecimento criar quase todas as movimentações que vemos no mercado, desde um simples andar até uma complexa animação de luta entre personagens. Os ciclos de animação são criados através de desenhos estáticos com uma sequência de poses que quando animadas transmitem a sensação de animação. Esses desenhos estáticos são conhecidos como “sprites”, sendo estes pequenas imagens com tamanhos aproximados que quando animados quadro a quadro passam a ideia de movimento e animação. A principal função dos sprites é economizar os recursos gráficos de computadores mais lentos pois a sua aplicação exige menos do poder de processamento das máquinas. Em um jogo também existem diversos efeitos gráficos que podem ser utilizados para as mais diversas funções como: explosões, fogos, raios, entre outros. Todos esses efeitos são produzidos com uma sequência de animações que tornam o visual dos jogos mais atraente e personalizada.

**2.8 Controlador de animações**

Um controlador de animações permite organizar e manter um conjunto de clipes de animação e transições de estado de animação associadas à um personagem ou um objeto. Na maioria dos casos, é normal ter várias animações e alternar entre elas quando certas condições do jogo ocorrem (UNITY USER MANUAL, 2018b).

Na engine de jogos Unity, mesmo se você tiver apenas um clipe de animação, ainda precisará colocá-lo em um controlador de animação para usá-lo na cena do jogo. O Unity cria automaticamente um

controlador de animações quando se começa a animar um personagem ou um objeto. O Animator Controller tem referências aos clipes de animação usados em um personagem ou objeto que esteja na cena do jogo, com isso ele gerencia os vários clipes de animação e as transições entre eles usando uma Máquina de Estado. Durante o modo de reprodução do jogo, o Animator Controller reproduz a visualização para que o estado atual que está sendo reproduzido esteja sempre visível (UNITY USER MANUAL, 2018b).

### 2.9 O que é Máquina de Estados Finita?

Para BONATO (2010), uma Máquina de Estados Finita (FSM – Finite State Machine) é composta por um conjunto de estados e um conjunto de regras de transição entre estes estados, refletindo em certo momento algum evento no mundo do jogo. É uma das técnicas de inteligência artificial mais utilizadas por desenvolvedores de jogos eletrônicos, onde é utilizada para definir o comportamento dos personagens no ambiente de um jogo, sendo bastante tradicional devido ao fato de ser de fácil compreensão e gastar pouco processamento.

Uma Máquina de Estados Finita é limitada quando há um aumento da complexibilidade no ambiente de jogo, pelo aumento do número de estados e de transições e uma máquina de estados tem que prever todos os casos e situações possíveis no ambiente de jogo. Outro problema encontrado são os comportamentos previsíveis e repetitivos, pois uma máquina de estados possuí um conjunto fixo de estados e de transições e se uma mesma situação acontecer duas vezes, o comportamento ativado será o mesmo em ambas as situações.

### 2.10 Comportamento na Máquina de Estados Finita

Na engine de jogos Unity, para definir o comportamento dos personagens através de uma máquina de estados, é utilizada a classe “State Machine Behaviour”. É uma classe especial de script, que funciona de maneira semelhante à anexação de scripts regulares do Unity (MonoBehaviours) para um objeto de jogo em cena. Anexando um script State Machine Behaviour à um estado individual dentro de uma máquina de estados permitirá que se escreva um código que será executado quando a máquina de estados entrar, sair ou permanecer dentro de um determinado estado, isso significa que não se precisa escrever sua própria lógica para testar e detectar alterações no estado (UNITY USER MANUAL (2018c).

Alguns exemplos para utilização desse recurso em jogos eletrônicos:

- Reproduzir sons à medida que os estados são iniciados ou terminados.
- Realizar determinados testes, de detecção de solo, por exemplo, somente quando em estados apropriados.
- Ativar e controlar efeitos especiais associados a estados específicos.

Os scripts de comportamento da máquina de estados possuem acesso a vários eventos que são chamados quando o Animator entra, atualiza e sai de estados diferentes (UNITY SCRIPTING API, 2018).

State Machine Behaviour possui algumas funções pré-definidas: OnStateEnter, OnStateUpdate, OnStateExit, descritas abaixo:

- OnStateEnter: Chamado no primeiro quadro de atualização, quando a máquina de estados avalia esse estado.
- OnStateUpdate: Chamado em cada quadro de atualização, exceto o primeiro e o último quadro.
- OnStateExit: Chamado no último quadro de atualização quando a máquina de estados avalia esse estado.

State Machine Behaviour possui também métodos públicos, métodos estáticos, operadores e outras funções pré-definidas que não serão abordadas nesse artigo.

### 2.11 Combinação com outras técnicas de IA

Para SOUZA (2014), a Lógica Fuzzy ou lógica difusa é a forma de lógica multivalorada na qual os valores lógicos das variáveis podem ser qualquer número real entre 0, correspondente ao valor falso, e 1, correspondente ao valor verdadeiro. Essa lógica permite em jogos utilizar valores mais precisos para definição de estados de um personagem, ao contrário da lógica booleana que alterna somente em valores de verdadeiro ou falso.

A Lógica Fuzzy combinada com uma máquina de estados permiti resolver parcialmente problemas de comportamento previsíveis e repetitivos, onde com valores Fuzzy nas transições de estado, pode-se suportar múltiplos estados ativos.

## 3 RESULTADOS E DISCUSSÃO

Para o desenvolvimento da aplicação primeiramente foi criado um game object vazio na cena, chamado Boss, em seguida com o game object criado selecionado, foi montado uma estrutura hierárquica com game objects para ser o corpo do chefe de fase, ficando na seguinte ordem: Head, Leg Right, Leg Left e dentro de Head foi adicionado: Eye Right, Eye Left, Arm Right e Arm Left. Todos contendo o componente Transform e o componente Sprite Renderer, com seu respectivo sprite em cada um. Para criação dos sprites foi utilizado o software Clip Studio. Em seguida, com o Boss selecionado foi adicionado o componente Animator Controller.

Após isso, com a guia Animation aberta foram criados os clips de animação, contendo os sete estados do nosso personagem: introStageOne, jumpOne, idleOne, introStageTwo, jumpTwo, idleTwo e dead. O Unity automaticamente criará uma instância para cada um desses estados na guia Animator. Foram criados quatro parâmetros na guia Animator: idle, jump, stageTwo e death, esses parâmetros servirão para fazer referência quando for preciso realizar uma transição de estado. Logo após, com algum estado selecionado e com o botão direito do mouse sendo pressionado, foi escolhida à opção “Make Transition” e apontado para o estado desejado. No final a Máquina de Estados Finita deverá ficar conforme a FIGURA 1.

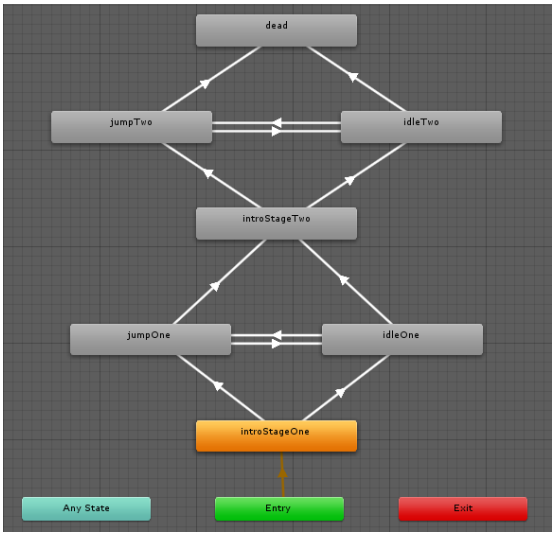


FIGURA 1 – Máquina de Estados Finita do chefe de fase

Voltando a cena de jogo com o objeto Boss selecionado, foram adicionados os components Rigidbody 2D e Circle Collider 2D. Após isso, foi criado um Script chamado Boss, sendo adicionado ao mesmo. Ao clicar nesse Script a Unity automaticamente abrirá um ambiente de desenvolvimento integrado para escrita de códigos, nesse caso foi utilizado o Mono Develop. Este Script conterá os atributos do chefe de fase, como: health, damage, healthBar, dentro outros, conforme representado na FIGURA 2.

```
7  public class Boss : MonoBehaviour {
8
9      public int health; //guarda a saúde do chefe de fase
10     private int damage; //guarda o dano que o chefe de fase causa ao jogador
11     private float timeBtwDamage = 1.5f; //guarda o tempo que o chefe de fase poderá causar outro dano
12
13     public Animator camAnim; //referência ao animator controller da câmera
14     public Slider healthBar; //mostra visualmente o nível de saúde do chefe de fase
15     private Animator anim; //referência ao animator controller do chefe de fase
16     public bool isDead; //indica se o chefe de fase está morto
17
18     private void Start()
19     {
20         //buscando o componente animator controller do chefe de fase
21         anim = GetComponent<Animator>();
22
23         //dano inicial ao jogador
24         damage = 5;
25     }
26
27     private void Update()
28     {
29         //comportamento que o chefe de fase terá de acordo com o seu nível de saúde
30         if (health <= 50) {
31             anim.SetTrigger("stageTwo");
32             damage = 15;
33         }
34
35         if (health <= 0) {
36             anim.SetTrigger("death");
37         }
38
```

FIGURA 2 – Script do chefe de fase

Logo após, voltando a guia Animator, e com algum dos estados selecionado, na guia Inspector foi adicionado o comportamento do nosso chefe de fase, através do botão “Add Behaviour”, ao clicar nesse botão será criado um Script da classe State Machine Behaviour. Sendo vinculado por padrão do Unity algumas funções pré-definidas ao Script criado, que deverão ser utilizadas conforme determinadas situações. E dentro desse Script são colocados atributos que sofreram alteração na transição para outro estado. A FIGURA 3 mostra o Script de comportamento adicionado para o estado de pulo do chefe de fase.

```
public class JumpBehavior : StateMachineBehaviour {

    private float timer; //guarda o intervalo de tempo em que a animação permanecerá ativa
    public float minTime; //guarda o tempo mínimo
    public float maxTime; //guarda o tempo máximo

    private Transform playerPos; //guarda a posição do jogador
    public float speed; //guarda a velocidade de movimentação do chefe de fase

    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex) {
        //buscando a posição do jogador
        playerPos = GameObject.FindGameObjectWithTag("Player").GetComponent<Transform>();

        //sorteando o tempo que a animação jump permanecerá ativa
        timer = Random.Range(minTime, maxTime);
    }

    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex) {
        //atualizando para encerrar a animação ou apontar outra animação
        if (timer <= 0)
        {
            animator.SetTrigger("idle");
        }
        else {
            timer -= Time.deltaTime;
        }

        //definindo o jogador como alvo do chefe de fase
        Vector2 target = new Vector2(playerPos.position.x, animator.transform.position.y);

        //movendo o chefe de fase para mais perto do jogador
        animator.transform.position = Vector2.MoveTowards(animator.transform.position, target, speed * Time.deltaTime);
    }

```

FIGURA 3 – Script do estado de pulo do chefe de fase

O Script conterá um atributo chamado “timer” para guardar o intervalo de tempo que o estado da animação de pulo permanecerá ativada, além de um atributo chamado “playerPos”, buscando a posição do nosso jogador, pois o chefe de fase deverá pular sempre na direção do nosso jogador. Também foi criado um atributo chamado “speed”, para guardar a velocidade de movimentação do chefe de fase em cena. Foi utilizado no desenvolvimento dessa aplicação somente as funções OnStateEnter, OnStateUpdate e OnStateExit.

Foi feito o mesmo procedimento para adicionar um comportamento no momento de entrada do chefe de fase no estágio um e no estágio dois, ambos com um Script da classe State Machine Behaviour chamado “introBehaviour”, além de terem sido criados Scripts de comportamento para os estados de “idle”, chamado “IdleBehaviour” e “dead”, chamado “DeathBehaviour”.

O atributo “health” do chefe de fase determinará a mudança para um comportamento mais agressivo. Foi criado uma barra contendo a saúde do chefe de fase, através do componente “Slider”, essa barra diminuíra conforme o jogador o atinja com os disparos dos projéteis de sua arma. Sendo o atributo “health” ligado a barra de sáude que foi criada, para o usuário do jogo ter uma melhor visualização. O chefe de fase possui o estágio um, se à sua saúde for maior ou igual que cinquenta, onde terá aparência conforme a FIGURA 4.

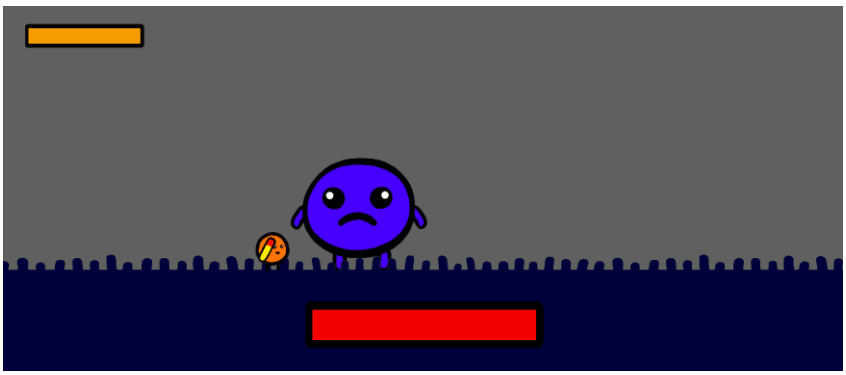

FIGURA 4 – Chefe de fase no estágio um

O chefe de fase também possui o estágio dois, se à sua saúde estiver entre zero e cinquenta, onde terá aparência conforme a FIGURA 5.

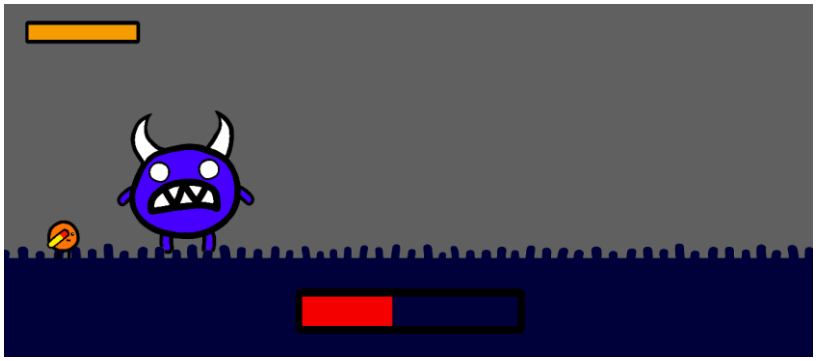

FIGURA 5 – Chefe de fase no estágio dois

Estando no estágio dois o chefe de fase, terá um comportamento mais agressivo, onde terá uma transição de estado mais rápida, além de sua velocidade de movimentação em cena ser maior também, assim o chefe de fase vai perseguir o jogador de uma maneira muito mais rápida.

Nessa mudança de comportamento para mais agressivo, também aumenta o dano causado do chefe de fase ao jogador, possuindo o valor de cinco no estágio um e passando para o valor de quinze no estágio dois.

O jogador também possuí uma barra de saúde que diminuíra conforme o chefe de fase o atinja, se a barra de saúde chegar a zero então será fim de jogo para o jogador, mas, se à barra de saúde do chefe de fase chegar a zero, então entrará no estado de morto e o jogador terá ganho.

Os resultados obtidos foram satisfatórios, onde no final do desenvolvimento da aplicação se obteve um jogo 2D, no qual à inteligência artificial do chefe de fase terá por objetivo perseguir o jogador ao longo da cena onde ele vá, com o intuito de acertá-lo para diminuir sua saúde.

## 4 CONCLUSÃO

Concluímos que foi possível identificar a facilidade em se desenvolver comportamentos autônomos com a classe State Machine Behaviour da engine de jogos Unity, onde foi possível criar uma fácil transição de estados, na Máquina de Estados Finita do chefe de fase que foi desenvolvido, ao longo deste presente artigo. Foi observado também a possibilidade de combinação com outras técnicas de inteligência artificial, para a geração de um comportamento mais interativo. Quando se combina uma Máquina de Estados Finita com a Lógica Fuzzy, por exemplo, podemos então definir valores do nível de saúde do chefe de fase de uma forma mais precisa.

Entretanto, foi notado que uma Máquina de Estados Finita é limitada, principalmente quando se tem explosões combinatórias, ou seja, com o aumento da complexibilidade no ambiente de jogo, também crescem o número de estados e de transições, pois essa técnica tem que prever todos os casos e situações possíveis no ambiente de jogo. Outro problema encontrado, são os comportamentos potencialmente repetitivos, pois essa técnica possuí um conjunto fixo de estados e de transições e se uma mesma situação acontecer duas vezes, o comportamento ativado será o mesmo em ambas as situações.

Uma abordagem para resolver esses problemas seria a criação de uma Máquina de Estados Hierárquica, onde cada estado seria uma nova Máquina de Estados Finita. Essa abordagem é bastante poderosa, permitindo uma melhor organização e modularização dos comportamentos, de modo a minimizar o impacto do crescente número de estados, porém não elimina o problema e também não trata o problema dos comportamentos repetitivos previsíveis.

Esses pontos observados servirão de base para novos projetos e aprofundamento de pesquisas em oportunidades futuras.

## REFERÊNCIAS


ABRA GAMES (2018). Brazilian Game Developers Export Program: O que é o BGD?. Associação Brasileira das Desenvolvedoras de Jogos Eletrônicos. 2018 [acesso em 11 nov 2018]. Disponível em: http://www.abragames.org/sobre-o-bgd.html

BONATO, Vanderlei (2010). Máquinas de Estados. Elementos de lógica digital II. 2010 [ acesso em 12 nov 2018]. Disponível em: http://wiki.icmc.usp.br/images/6/60/Aula_3_-_StateMachineSSC0110_2010.pdf

CLIP STUDIO (2016). Clip Studio Paint. 2016 [acesso em 12 nov 2016]. Disponível em: https://www.clipstudio.net/en

MILHOMEN, Marcelo (2016). Uso da animação em jogos. 2018 [atualizada em 10 nov 2016, acesso em 12 nov 2018]. Disponível em: http://www.formuladejogos.com.br/single-post/2016/11/09/5---ANIMA%C3%87%C3%83O---Uso-da-anima%C3%A7%C3%A3o-em-jogos

MONO DEVELOP (2018). Cross platform IDE for C#, F# and more. 2018 [acesso em 12 nov 2016]. Disponível em: https://www.monodevelop.com/

PRADO, J. (2016). A Inteligência Artificial é mais antiga do que você imagina. 2016 [acesso em 10 nov 2018]. Disponível em: https://tecnoblog.net/195106/inteligencia-artificial-historia-dilemas/

SELECT GAME (2010). Estado da Arte da Inteligência Artificial para jogos eletrônicos. 2010 [acesso em 11 nov 2018]. Disponível em: https://www.selectgame.com.br/artigos-e-tutoriais/estado-da-arte-da-inteligencia-artificial-para-jogos-eletronicos/

SOUZA, W. (2014). Lógica Fuzzy Conceitos e Aplicações. 2014 [acesso em 16 nov 2018]. Disponível em: https://pt.slideshare.net/ToniEsteves/logica-fuzzy-conceitos-e-aplicaes

TECHOPEDIA (2018). Artificial Intelligence (AI). 2018 [acesso em 10 nov 2018]. Disponível em: https://www.techopedia.com/definition/190/artificial-intelligence-ai

TOGELIUS, J. (2018). Artificial Intelligence and Games. 2018. [ acesso em 12 nov 2018]. Disponível em: http://gameaibook.org/book.pdf

UNITY SCRIPTING API (2018). Script Reference - State Machine Behaviours. San Francisco: Unity Technologies. 2018 [atualizada em 06 nov 2018, acesso em 14 nov 2018]. Disponível em: https://docs.unity3d.com/ScriptReference/StateMachineBehaviour.html

UNITY USER MANUAL (2018a). Unity User Manual. San Francisco: Unity Technologies. 2018 [atualizada em 06 nov 2018, acesso em 12 nov 2018]. Disponível em: https://docs.unity3d.com/Manual/UnityManual.html

UNITY USER MANUAL (2018b). Animation Reference – Animator Controller. San Francisco: Unity Technologies. 2018 [atualizada em 06 nov 2018, acesso em 12 nov 2018]. Disponível em: https://docs.unity3d.com/Manual/class-AnimatorController.html

UNITY USER MANUAL (2018c). State Machine Behaviours. San Francisco: Unity Technologies. 2018 [atualizada em 06 nov 2018, acesso em 14 nov 2018]. Disponível em: https://docs.unity3d.com/Manual/StateMachineBehaviours.html

YANNAKAKIS, G. N.; TOGELIUS, J. (2014). A panorama of artificial and computational intelligence in games. IEEE, n. 1943-068X, 2014.