CENTRO PAULA SOUZA
GOVERNO DO ESTADO SÃO PAULO

FACULDADE DE TECNOLOGIA DE AMERICANA
Bacharelado em Sistemas e Tecnologia da Informação

Lucas Vieira de Miranda

APLICAÇÃO DE MÁQUINA DE ESTADOS EM JOGOS DIGITAIS

AMERICANA - SP
2014

FACULDADE DE TECNOLOGIA DE AMERICANA
Bacharelado em Sistemas e Tecnologia da Informação

Lucas Vieira de Miranda

APLICAÇÃO DE MÁQUINA DE ESTADOS EM JOGOS DIGITAIS

Monografia apresentada à Faculdade de Tecnologia de Americana – FATEC Americana, como requisito parcial para obtenção do título de Bacharel em Sistemas e Tecnologia da Informação

Orientador: Prof. MSc. Rossano Pablo Pinto

Coorientador: Prof. MSc. Kleber de Oliveira Andrade

AMERICANA - SP
2014

MIRANDA, Lucas Vieira de. **Aplicação de Máquina de Estados em Jogos Digitais**. Bacharelado em Sistemas e Tecnologia da Informação. Faculdade de Tecnologia de Americana. Americana, SP. (Trabalho de Conclusão de Curso) 54 pág.

RESUMO

O presente trabalho tem como objetivo estudar a aplicação do algoritmo de máquina de estados, uma das técnicas proveniente de um campo de estudo chamado de inteligência artificial, como ferramenta para modelar o comportamento de personagens em jogos digitais, tornando-os agentes autônomos.

Para esse estudo, realizou-se, primeiramente, um levantamento bibliográfico, em seguida, utilizou-se o método investigativo a partir da implementação do protótipo de um jogo denominado *Dragon's Den,* inspirado no clássico *Bomberman*, aplicando o algoritmo de máquina de estados como ferramenta para modelar o comportamento dos personagens, documentando todo o processo de desenvolvimento.

Essa pesquisa torna-se relevante, pois personagens inteligentes fazem com que o jogo seja mais complexo e atrativo para seus jogadores.

Palavras-chave: Jogos Digitais, Inteligência Artificial, Máquina de Estados.

MIRANDA, Lucas Vieira de. **Aplicação de Máquina de Estados em Jogos Digitais**. Bacharelado em Sistemas e Tecnologia da Informação. Faculdade de Tecnologia de Americana. Americana, SP. (Trabalho de Conclusão de Curso) 54 pág.

ABSTRACT

The present work aims to study the application of the state machine algorithm, one of the techniques from a field of study called artificial intelligence, as a tool for modeling the behavior of characters in digital games, making them autonomous agents.

For this study, it was held, first, a bibliographic survey then used the investigative method from the implementation of a prototype game called Dragon's Den, inspired by the classic Bomberman, applying the algorithm state machine as a tool to model the behavior of the characters, documenting the entire process of development.

This research is relevant because smart characters make the game more complex and attractive for their players.

Keywords: Computer Games, Artificial Intelligence, State Machine.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

IA	Inteligência Artificial
FSM	*Finite State Machine*
MEF	Máquina de Estados Finitos
MEFD	Máquina de Estados Finitos Determinísticos
MEFND	Máquina de Estados Finitos Não Determinísticos
MEFH	Máquina de Estados Finitos Hierárquica
MEFFu	Máquina de Estados Finitos *Fuzzy*
UML	*Unified Modelling Language*

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de jogos digitais é uma atividade relativamente jovem que vêm crescendo rapidamente ano após ano devido à aceitação do público e à difusão das tecnologias da informação. Gregory (2009), define como jogo digital todo e qualquer software capaz de simular um mundo imaginário (virtual), no qual uma pessoa (chamada de jogador), controla um personagem capaz de interagir com esse ambiente e seus personagens, sejam eles outros jogadores ou entidades controladas por computador.

Tendo isso em mente, pode-se observar que o desenvolvimento de entidades autônomas é uma tarefa importante e recorrente na produção de jogos digitais. A construção desse tipo de entidade é possível devido a um amplo conjunto de técnicas fornecidas por uma área de estudo da ciência da computação chamada de inteligência artificial (IA). Segundo Schwab (2009), a IA pode ser dividida em duas vertentes devido aos seus objetivos, estas vertentes são chamadas de IA clássica e IA para jogos digitais. Para Mellington e Funge (2009), a IA para jogos digitais se difere da IA clássica, pois a primeira busca criar comportamentos que causem ilusão de inteligência, enquanto a segunda busca compreender o conceito de inteligência e criar programas que simulem o comportamento e o pensamento humano que são vistos como o padrão de inteligência ideal.

Uma das técnicas proveniente da IA clássica que é utilizada na IA para jogos digitais na modelagem do comportamento de objetos e personagens é a máquina de estados finitos. A máquina de estados finitos é uma estrutura simples composta de estados e transições que permite a definição de padrões lógicos que representarão as ações que serão desempenhadas pelos personagens durante a execução do jogo.

Dentro desse contexto, este trabalho tem como objetivo implementar o protótipo de um jogo chamado *Dragon's Den*, no qual os personagens possuam uma inteligência artificial baseada em máquina de estados. Para tal, realizou-se um levantamento bibliográfico sobre os diferentes tipos de máquina de estados

existentes e suas aplicações em jogos digitais.

No segundo capítulo serão abordados conceitos fundamentais como inteligência artificial, sua história e a inteligência artificial para jogos digitais.

Já no terceiro capítulo descreve-se uma técnica de inteligência artificial conhecida como máquina de estados finitos e suas variantes.

O quarto capítulo aborda o conceito, mecânica e controle do jogo *Dragon's Den*.

No quinto capítulo é abordado a modelagem da inteligência artificial baseada em máquina de estados e os resultados obtidos com a implementação.

No sexto capítulo aborda-se a metodologia de desenvolvimento e as ferramentas utilizadas durante o desenvolvimento da máquina de estados.

O sétimo e último capítulo se reserva às conclusões obtidas durante o desenvolvimento desse trabalho.

2 INTELIGÊNCIA ARTIFICIAL

Este capítulo tem como objetivo introduzir o conceito de inteligência artificial e inteligência artificial para jogos digitais.

Sendo assim, a seção 2.1 destina-se a apresentação do conceito de inteligência artificial e sua história de modo geral. Já a seção 2.2, foca-se na inteligência artificial para jogos digitais.

2.1 Inteligência Artificial

Segundo Russel e Norvig (2009) a inteligência artificial (IA), é uma área multidisciplinar, que envolve: filosofia, matemática, economia, neurociência, psicologia, computação, robótica e linguística. Russel e Norvig (2009), também dividem as áreas de estudo da IA em quatro abordagens principais, duas delas concentram-se nos processos de pensamento e raciocínio e outras duas no comportamento, duas delas defendem uma abordagem humanista que toma como sucesso a fidelidade ao desempenho humano, já as outras duas defendem uma abordagem racionalista, e tomam como sucesso o conceito ideal de inteligência, que é chamado de racionalidade. Diz-se que um sistema é racional quando desempenha suas tarefas corretamente com os dados que possui. As abordagens referidas, são:

- Sistemas que pensam como seres humanos: Essa abordagem baseia-se na ciência cognitiva, e envolve neurociência, psicologia e computação no esforço de compreender os processos por trás do pensamento humano e criar modelos computacionais que possam representá-los. Nessa abordagem utiliza-se de técnicas psicológicas e da introspecção como ferramentas para a coleta e análise de dados, a partir disso são criados modelos mentais. Esses modelos são, então, transformados em programas de computador e caso o

programa de computador corresponda com o modelo mental, encontram-se evidências de que o cérebro humano opera de forma semelhante a encontrada no programa;

- Sistemas que atuam como seres humanos: Essa abordagem, também chamada de “Abordagem do Teste de Turing”, recebeu esse nome em homenagem ao cientista da computação Alan Turing, que em 1950, propôs um teste que hoje chamamos de “Teste de Turing”. Nesse teste submete-se um computador a um interrogatório realizado por um ser humano, caso o interrogador não consiga distinguir se o interrogado é uma máquina ou ser humano, o computador terá obtido êxito no teste. Programar uma máquina para superar o “Teste de Turing”, é uma tarefa complexa e envolve a aplicação de várias técnicas que atualmente compõe a IA, como: processamento de linguagem natural, representação de conhecimento, raciocínio automatizado, aprendizado de máquina, visão computacional e robótica;

- Sistemas que pensam racionalmente: Essa abordagem, também chamada de “Abordagem Logicista”, se concentra no estabelecimento do raciocínio válido ou correto, e utiliza-se, principalmente, da lógica clássica durante o processo de avaliação;

- Sistemas que atuam racionalmente: Essa abordagem, também chamada de “Abordagem do Agente Racional”, tem como foco o estudo e construção de agentes racionais. Um agente é qualquer coisa que possa perceber seu ambiente por meio de sensores e agir sobre o mesmo através de atuadores. Chama-se de agente racional um agente que realiza todas as suas tarefas de maneira correta considerando os dados que possui, e que em momentos de incerteza ou imprecisão, tenta realizá-las da melhor maneira possível.

Para Mellington e Funge (2009), a história da IA pode-se resumir a três períodos principais, que são: “Os Primeiros Dias”, “A Era Simbólica” e “A Era Contemporânea”. Essa divisão ocorre, pois em cada um desses períodos houve uma abordagem predominante.

Os primeiros dias da IA começam antes do surgimento dos computadores. Nesta época a humanidade fazia perguntas, como “O que é o pensamento?”, “O que é a inteligência?” e “Máquinas podem vir a pensar ou a ser inteligentes?”. Estas perguntas que hoje são atribuídas à filosofia, junto ao surgimento dos computadores programáveis na década de 1940, foram responsáveis pelo surgimento do que hoje chama-se de IA.

A era simbólica ocorreu de 1950 até 1980, e recebe esse nome, pois a maior parte das pesquisas na área de IA se concentrou no que se chama de sistemas simbólicos. Os cientistas da computação Ian Mellington e John Funge definem sistemas simbólicos, como:

> Um sistema simbólico é um sistema no qual o algoritmo é dividido em dois componentes principais: um conjunto de conhecimentos (representado por símbolos, como: palavras, números, sentenças, imagens, etc…) e um algoritmo que manipula esses símbolos para solucionar um problema ou gerar um novo conhecimento. (MELLINGTON, Ian; FUNGE, John. Artificial Intelligence For Games. Tradução Nossa. 2. ed. Burlington: Elsevier, 2009. 5 p)

Entre as técnicas que foram desenvolvidas nessa época, pode-se citar: os sistemas baseados em regras, técnicas de busca em grafos, árvores de decisões, máquina de estados, entre outras.

A era contemporânea vem ocorrendo desde 1980 até os dias atuais, seu início foi marcado pelo surgimento de problemas maiores e mais complexos e pela crescente frustração com as abordagens simbólicas uma vez que essas, pareciam não mais escalar. A solução foi buscar novas opções inspiradas na biologia ou em outros sistemas encontrados na natureza. Essa época é marcada pelas redes neurais e pelos algoritmos genéticos, bem como o surgimento de novos métodos probabilísticos, como por exemplo: redes de bayes, máquina de suporte de vetores e processos gaussianos (MELLINGTON e FUNGE, 2009).

Atualmente, tanto as abordagens simbólicas quanto as abordagens

contemporâneas encontram-se em uso, pois a grande maioria dos cientistas percebeu que dependendo do problema que se deve solucionar, uma abordagem pode se sair melhor do que a outra.

2.2 Inteligência Artificial para Jogos Digitais

Para Schwab (2009), a IA acadêmica difere-se da IA para jogos digitais quanto aos seus objetivos. Na IA acadêmica busca-se compreender o conceito de inteligência e construir programas que simulem o pensamento e o comportamento humano, bem como o pensamento e o comportamento racional. Já a IA para jogos digitais foca-se na questão comportamental, pois busca construir programas que simulem os altos e baixos encontrados no comportamento humano, objetivando criar a ilusão de inteligência, de forma que os jogos se tornem mais imersivos, competitivos e divertidos para seus jogadores.

Para Mellington e Funge (2009), na construção de uma IA para jogos pode-se mesclar três abordagens distintas, que são:

- *Hacks*: Artimanhas, geralmente soluções simples que ajudam a produzir uma certa ilusão de inteligência. Um exemplo dessa técnica, é a adição de informações em objetos presentes no cenário com o objetivo de facilitar a interação da inteligência artificial com o ambiente;
- Heurísticas: Soluções aproximadas e não formais, que podem funcionar em diversas situações, mas que não possuem garantias de funcionamento em outros cenários. Um exemplo dessa técnica, é o ataque automático preventivo realizado por unidades controladas por jogadores em jogos de estratégia quando um inimigo se encontra muito próximo, esse tipo de estratégia pode funcionar algumas vezes, mas não em todos os casos;
- Algoritmos: Nesse caso, são técnicas formais já testadas e comprovadas, e constituem um amplo ferramental, muito útil para a resolução de problemas já

conhecidos. Um exemplo desse tipo de técnica, é a busca de caminho que já possui diversas soluções testadas e comprovadas.

Mellington e Funge (2009), consideram que entre as três abordagens citadas, os algoritmos são as peças mais importantes para a construção de uma IA para jogos, e propõe que os algoritmos sejam agrupados em três categorias distintas, segundo suas funções, estas categorias são:

- Estratégia: Agrupa técnicas utilizadas para coordenar o comportamento de diversos agentes simultaneamente. A função principal desses algoritmos é controlar todos os agentes que compõem um determinado grupo para alcançar um único objetivo;

- Decisão: Agrupa técnicas utilizadas para coordenar o comportamento de um único agente. A função principal desses algoritmos é decidir qual ação será executada e quando será executada;

- Movimentação: Agrupa técnicas utilizadas para coordenar o movimento de um único agente, considerando o mundo e os obstáculos presentes no mesmo.

Bourg e Seeman (2004) alertam para duas características importantes encontradas nos algoritmos utilizados para a construção de IA em jogos, eles podem ser determinísticos ou não determinísticos. Um algoritmo determinístico é aquele que não apresenta nenhum grau de incerteza, ou seja, será executado sempre da mesma forma, já um algoritmo não determinístico é o oposto, pois apresenta um certo grau de incerteza, o que faz com que sua execução não seja realizada sempre da mesma forma.

Ainda segundo Bourg e Seeman (2004), é fundamental saber como e quando escolher entre uma abordagem e outra ao enfrentar um problema, porque o sucesso do jogo todo pode depender dessas escolhas.

3 MÁQUINA DE ESTADOS APLICADAS A JOGOS DIGITAIS

Uma máquina de estados finitos (MEF), do inglês *Finite State Machine (FSM)*, consiste basicamente de um conjunto de estados finitos e um conjunto de regras de transição, no qual um estado representa uma etapa de um processo a ser executado e uma regra de transição compreende as condições necessárias para se mudar de um estado para outro (MELLINGTON e FUNGE, 2009).

O funcionamento de uma máquina de estados ocorre da seguinte forma: ao ser iniciada, a máquina, automaticamente, assume o estado definido como inicial, e a partir desse momento ela passa a receber dados, que serão utilizados pelas funções de transição no processo de decisão. Após tomada uma decisão poderá ou não ocorrer uma mudança de estados. Esse processo ocorrerá a cada iteração até que a máquina atinja um estado definido como final.

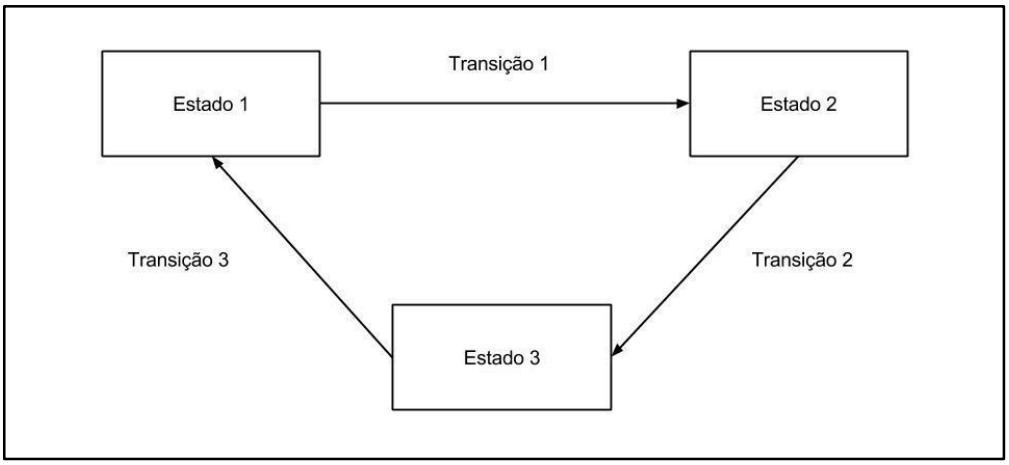
Pode-se representar uma máquina de estados através de um diagrama de gráfos dirigidos, no qual os vértices representam os estados, os valores sobre as arestas representam as entradas que serão submetidas a função de transição e o sentido da seta indica qual será o próximo estado (MELLINGTON e FUNGE, 2009). A figura 1 demonstra um exemplo do diagrama de gráfos dirigidos.

Outra opção para a representação, seria o diagrama de máquina de estados presente na Linguagem de Modelagem Unificada, do inglês *Unified Modelling Language* (*UML*), no qual os quadrados representam os estados, os valores sobre as arestas representam a condição de transição e o sentido da seta indica qual será o próximo estado (MELLINGTON e FUNGE, 2009). A figura 2 demonstra um exemplo do diagrama de máquina de estados da UML.

Ainda, para Melligton e Funge (2009), as máquinas de estados são ferramentas muito úteis para modelagem de comportamento tanto para personagens quanto para objetos em jogos digitais. Já Schwab (2009) e Buckland (2004), afirmam que nenhuma outra técnica para controle de comportamento em jogos foi mais utilizada que as máquinas de estado. Buckland (2004), atribui esse sucesso as seguintes características: rapidez e simplicidade de codificação, facilidade de

depuração, pouco uso do processador, intuitividade e flexibilidade.

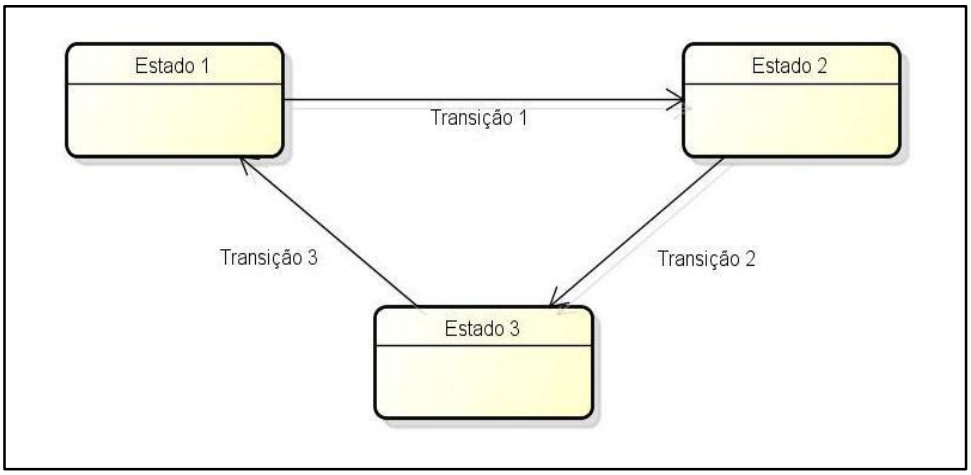
Figura 1. Diagrama de gráfos dirigidos.



Fonte: Elaborada pelo autor.

Existem quatro tipos de máquinas de estado utilizadas no desenvolvimento de jogos digais: a máquina de estados finitos determinísticos, a máquina de estados finitos não determinísticos, máquina de estados hierárquicas e a máquina de estados finitos *fuzzy*.

Figura 2. Diagrama de máquina de estados da UML.



Fonte: Elaborada pelo autor.

3.1 Máquina de Estados Finitos Determinísticos

A máquina de estados finitos determinísticos (MEFD), recebe esse nome pois as transições entre os estados ocorrem de forma ordenada, ou seja, dado o estado atual e uma entrada, seu próximo estado é pré-estabelecido (HOPCROFT ET AL, 2006).

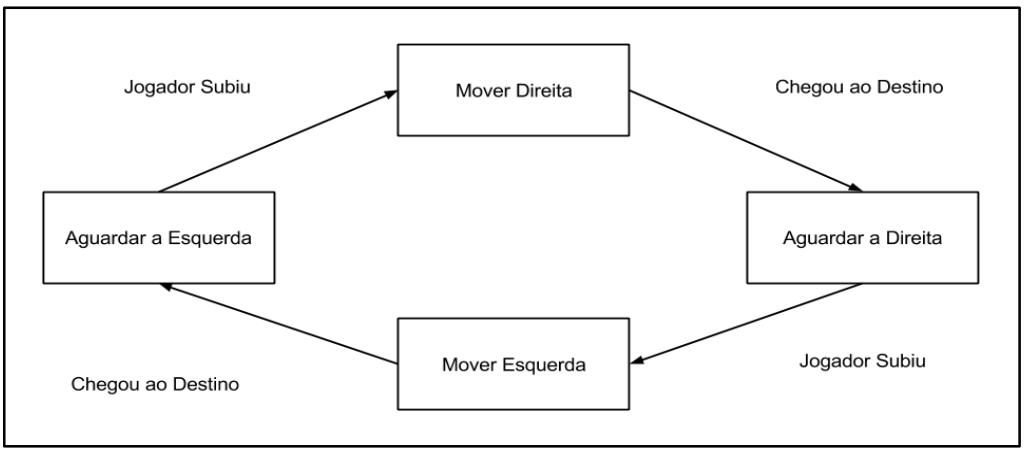
Hopcroft et al (2006) também define, formalmente, uma MEFD como uma quíntupla ordenada $(S, \Sigma, \delta, s_0, F)$, na qual:

- S representa um conjunto finito, não vazio, de estados;
- Σ representa um conjunto finito, não vazio, de dados de entrada, chamado de alfabeto;
- δ representa a função parcial $\delta: S \times \Sigma \rightarrow P(S)$, chamada de função de transição;
- s_0 , no qual $s_0 \subseteq S$ e $s_0 \neq \emptyset$, representa o estado inicial;
- F, no qual $F \subseteq S$, representa o conjunto dos estados finais.

O funcionamento de uma máquina de estados finitos determinísticos pode ser entendido da seguinte forma: dada uma máquina $\mathrm{M} = (\{s_0, s_1\}, \{0, 1\}, \delta, s_0, \{s_1\})$, ao iniciar a máquina imediatamente assume o estado s_0, desse momento em diante a máquina passa a receber dados, quando uma entrada de dados é detectada, dado o seu estado atual e o valor da entrada, a máquina poderá atingir o próximo estado pré-definido ou caso receba uma entrada inválida, se manterá no estado atual ou retornará para o estado inicial. A máquina continuará a operar dessa forma até o momento em que uma transição a leve para o estado final s_1.

Um exemplo simples desse tipo de máquina aplicado a jogos digitais poderia ser o controle de uma plataforma móvel. Suponha que uma plataforma deve mover-se da esquerda para a direita e da direita para a esquerda quando um jogador sobe na plataforma. A figura 3 ilustra uma máquina de estados que, possivelmente, resolveria esse problema.

Figura 3. Possível máquina de estados para plataforma móvel.



Fonte: Elaborada pelo autor.

Nesse exemplo, o estado inicial da plataforma é “Aguardar à Esquerda”, quando o jogador subir na plataforma, a condição de transição “Jogador Subiu” será disparada e o estado “Mover para Direita” será executado, fazendo com que a plataforma se mova para a direita. Quando a plataforma chegar ao ponto de destino,

a condição de transição “Chegou ao Destino” será disparada, fazendo com que a plataforma assuma o estado “Aguardar à Direita”, o que fará a plataforma parar. O mesmo processo ocorrerá da direita para a esquerda.

3.2 Máquina de Estados Finitos Não Determinísticos

Máquina de estados finitos não determinísticos (MEFND) recebe esse nome pois as transições entre os estados ocorrem de forma desordenada, ou seja, dado o estado atual e uma entrada, é possível que a máquina atinja N diferentes estados, não havendo um próximo estado pré-determinado (HOPCROFT ET AL, 2006).

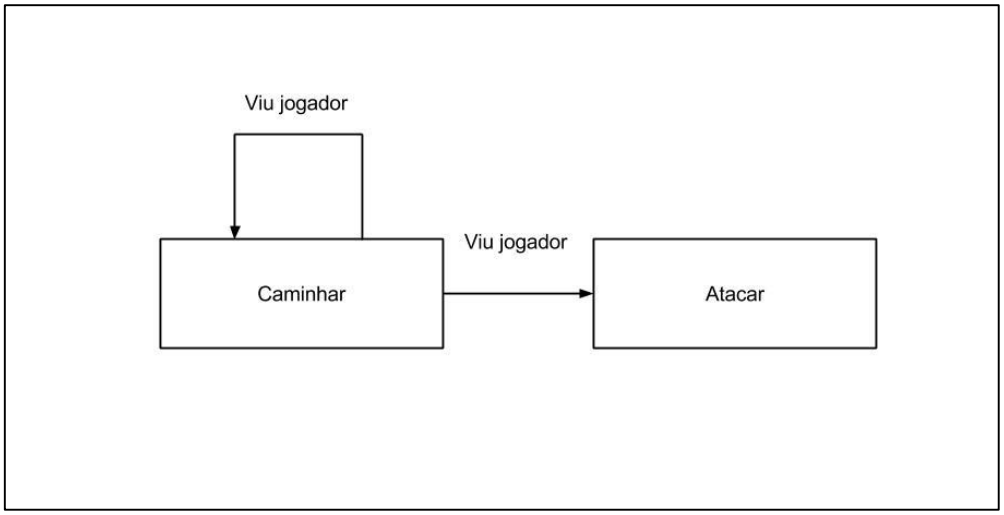
Assim como a MEFD, Hopcroft et al (2006), definem formalmente uma MEFND, como uma quíntupla ordenada $(S, \Sigma, \delta, s_0, F)$, na qual:

- S representa um conjunto finito, não vazio, de estados;
- Σ representa um conjunto finito, não vazio, de dados de entrada, chamado de alfabeto;
- δ representa a função parcial $\delta: S \times \Sigma \rightarrow P(S)$, chamada de função de transição;
- s_0 , no qual $s_0 \subseteq S$ e $s_0 \neq \emptyset$, representa o estado inicial;
- F, no qual $F \subseteq S$, representa o conjunto dos estados finais.

Basicamente, o funcionamento de uma máquina de estados finitos não determinísticos ocorre da seguinte forma: dada a máquina $M = (\{s_0, s_1, s_2\}, \{1, 2\}, \delta, s_0, \{s_1\})$, ao iniciar, a máquina, imediatamente, assume o estado inicial, chamado de s_0. A partir desse momento ela passa a receber entradas, quando uma entrada é detectada, dado seu estado atual e o valor da entrada, é possível que a mesma atinja diferentes estados. A máquina continuará a operar até o momento em que uma transição a leve para o estado final, nesse caso, s_1.

Um exemplo da utilização desse tipo de máquina de estados aplicada a jogos digitais seria a modelagem do comportamento de um jacaré. Suponha que um jacaré possa atacar ou ignorar outro personagem que se encontre em sua área de visão. A figura 4 ilustra uma máquina de estados que, possivelmente, resolveria esse problema.

Figura 4. Possível máquina de estados não determinística para um jacaré.



Fonte: Elaborada pelo autor.

Nesse exemplo, o estado inicial do jacaré é “Caminhar”, quando o jacaré percebe que outro personagem entra em sua área de visão, aplica-se um método pseudo aleatório para determinar se o jacaré continuará no mesmo estado, ou seja, “Caminhar”, ou realizará a transição para o estado “Atacar”, no qual o jacaré iniciará um ataque ao personagem percebido.

3.3 Máquina de Estados Hierárquica

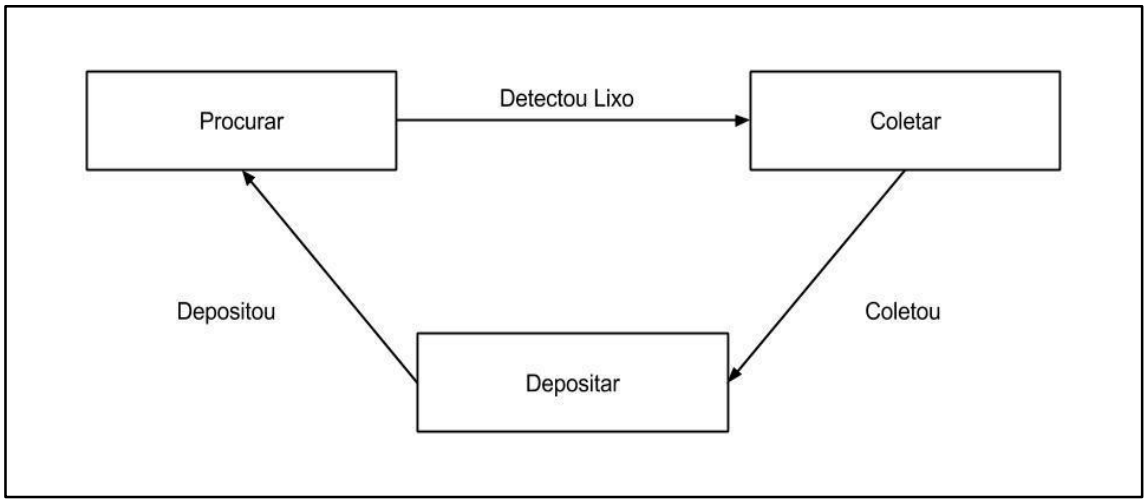
As máquinas de estado sejam determinísticas ou não determinísticas, são ferramentas poderosas para a modelagem de processos ou comportamentos tanto para objetos quanto para personagens em jogos digitais, entretanto, conforme a complexidade envolvida nos processos ou comportamentos aumenta, o número de condições de transição também aumenta (MELLINGTON e FUNGE, 2009).

Surge, então, a necessidade de se gerenciar essa complexidade e, por consequência, o número de condições de transição. Nesses casos, pode-se utilizar uma técnica conhecida como máquina de estados finitos hierárquica (MEFH), esse tipo de máquina recebe esse nome pois permite que seus estados representem novas máquinas de estado, formando uma grande máquina aninhada. A máquina mais externa passa, então, a ser chamada de super máquina e a máquina mais interna passa a ser chamada de sub máquina (MELLINGTON e FUNGE, 2009).

A capacidade de representar estados como sub máquinas torna possível o agrupamento de estados que possuam características em comum e também permitem que estados mais externos tenham maior prioridade de execução sobre os estados mais internos. Essas duas características permitem que casos complexos sejam simplificados, de forma a diminuir o número de condições de transição e a simplificar a representação do problema.

Uma maneira de ilustrar como a máquina de estados hierárquica pode simplificar o desenvolvimento de um comportamento ou processo, pode ser expresso na modelagem de um robô coletor de lixo. Um robô coletor de lixo pode possuir três estados, que são: “procurar”, “coletar” e “depositar”. Esse robô, ao ser iniciado, assumirá o estado “procurar” e irá tentar localizar a pilha de lixo mais próxima, quando a pilha for localizada, o robô passará ao estado “coletar”, nesse estado o robô irá se mover até a pilha de lixo e coletá-lo, após a coleta o robô passará para o estado “depositar”. Nesse momento, ele irá procurar a lata de lixo mais próxima e se moverá até ela, na qual, por fim, depositará o lixo. A figura 5 ilustra uma máquina de estados que, possivelmente, solucionaria esse problema.

Figura 5. Possível máquina de estados para o robô coletor de lixo (Versão 1).

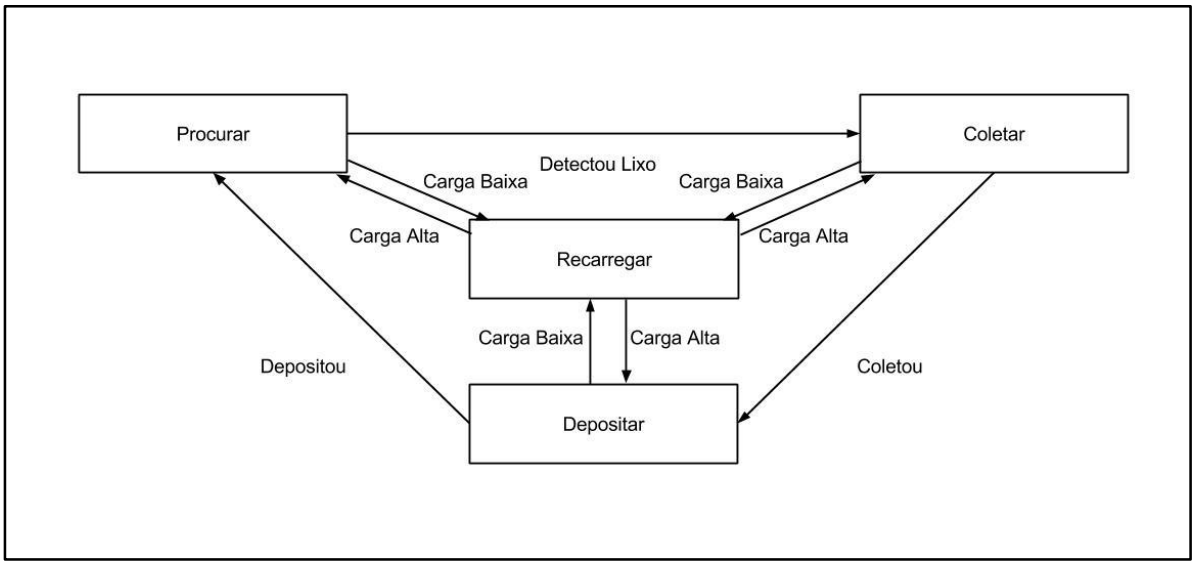


Fonte: Mellington e Funge, 2009, p. 319.

Nota-se que apesar da máquina especificada cumprir os requisitos que compreendem a procura, coleta e depósito do lixo, esse modelo não será eficiente, pois as ações só serão executadas até o momento em que a energia do robô acabe e ele seja desligado.

Surge, então, a necessidade de se refinar esse comportamento para que o robô saiba quando deve se dirigir ao ponto de recarga para recuperar sua energia e, posteriormente, retornar suas tarefas. Uma forma de incluir esse comportamento, é a adição de um estado chamado “recarregar”. Esse estado poderá ser chamado em qualquer um dos três estados já existentes quando a energia do robô estiver baixa. A figura 6 ilustra a máquina de estados proposta.

Figura 6. Possível máquina de estados para o robô coletor de lixo (Versão 2).



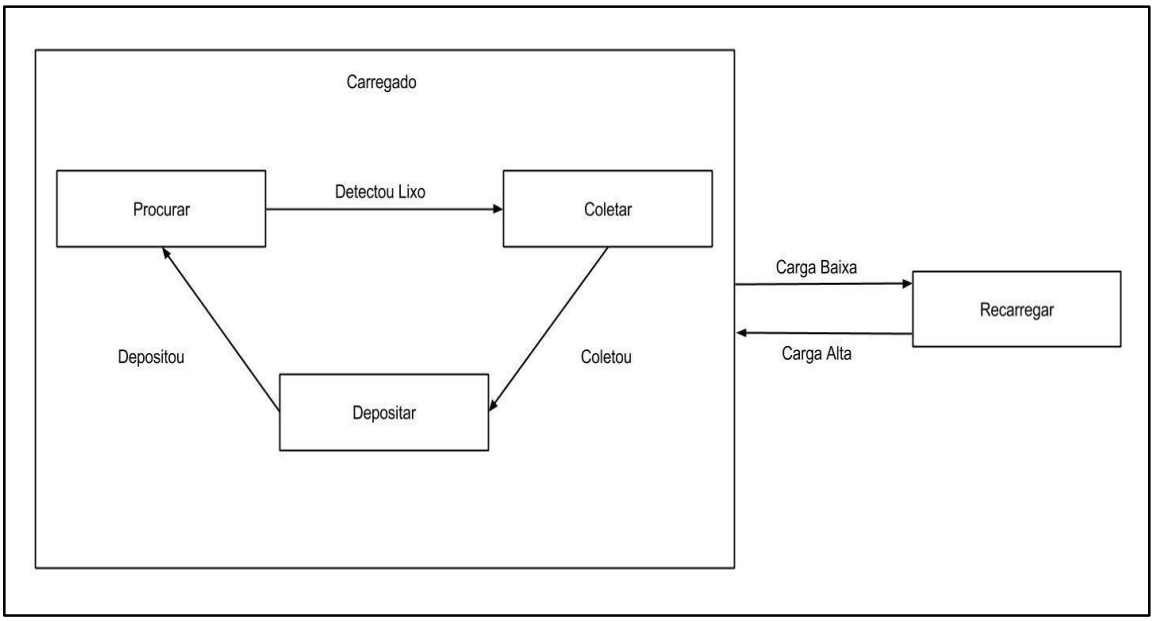
Fonte: Mellington e Funge, 2009, p. 319.

A adição do estado “recarregar” é uma saída eficiente, pois, realmente, soluciona o problema referente ao comportamento do robô. Mas a aplicação dessa abordagem também implica na criação de três novas condições de transição, o que aumenta a complexidade presente no modelo.

Ao observar atentamente o diagrama pode-se notar que os estados procurar, coletar e depositar, podem, em algum momento, atingir o estado “recarregar”. Isso demonstra que esses três estados só serão executados enquanto a carga do robô estiver alta.

Considerando o exposto acima, uma saída viável para a redução da complexidade, é a criação de um novo estado chamado de “carregado”, esse novo estado irá abrigar a máquina de estados responsável pela coleta de lixo, de forma que enquanto a energia do robô estiver alta, o mesmo continuará a desempenhar suas tarefas, e quando a energia do robô estiver baixa, ocorrerá a transição para o estado “recarregar” no qual ficará até que a sua carga esteja completa e possa retornar ao trabalho de onde parou. A figura 7 ilustra a máquina de estados com a adição do estado “carregado”.

Figura 7. Possível máquina de estados para o robô coletor de lixo (Versão 3).

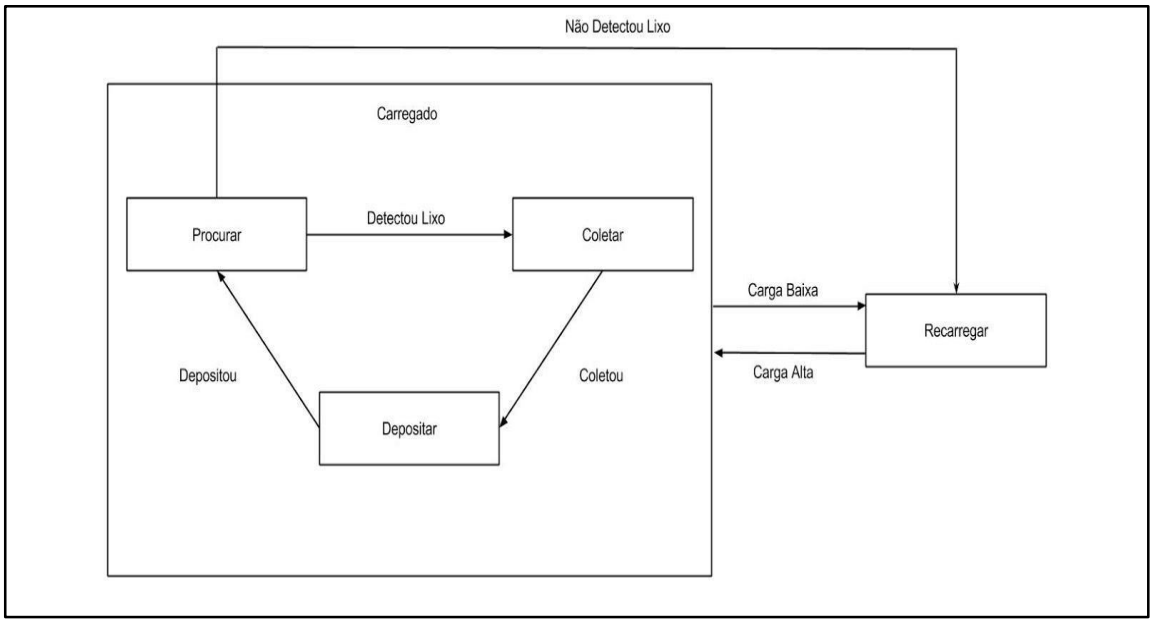


Fonte: Mellington e Funge, 2009, p. 320.

Ainda, pode-se refinar mais esse modelo adicionando uma transição entre hierarquias no estado “procurar”, para que quando não haja mais sujeira o robô se mantenha no estado “recarregar” e não gaste sua energia se locomovendo à toa. A figura 8 ilustra a máquina de estados proposta.

A implementação final do estado “carregado”, demonstra como as máquinas de estado hierárquicas podem reduzir a complexidade existente em um modelo ou solucionar problemas que envolvam estados com características comuns, nos quais um deles tenha prioridade sobre os demais.

Figura 8. Possível máquina de estados para o robô coletor de lixo (Versão 4).



Fonte: Mellington e Funge, 2009, p. 321.

3.4 Máquina de Estados Fuzzy

As máquinas de estado vistas até agora são técnicas muito úteis para a modelagem de comportamento, mas em todas elas os estados representam situações ideais de operação, ou seja, em cada estado existem apenas algumas ações a serem executadas. Mas em muitos casos, é necessário que se possa mesclar comportamentos distintos, isto é, estar em mais de um estado ao mesmo tempo, pois isso permite a produção de comportamentos complexos ou aleatórios.

Uma das formas de se implementar comportamentos complexos ou aleatórios consiste na utilização de uma máquina de estados finitos fuzzy (MEFFu), esse tipo de máquina de estados recebe esse nome porque se utiliza de princípios e conceitos existentes na lógica *fuzzy* (SCHWAB, 2009).

A lógica *fuzzy* ou lógica difusa pode ser entendida como uma extensão da lógica booleana que possibilita a existência de valores entre verdadeiro (1) e falso

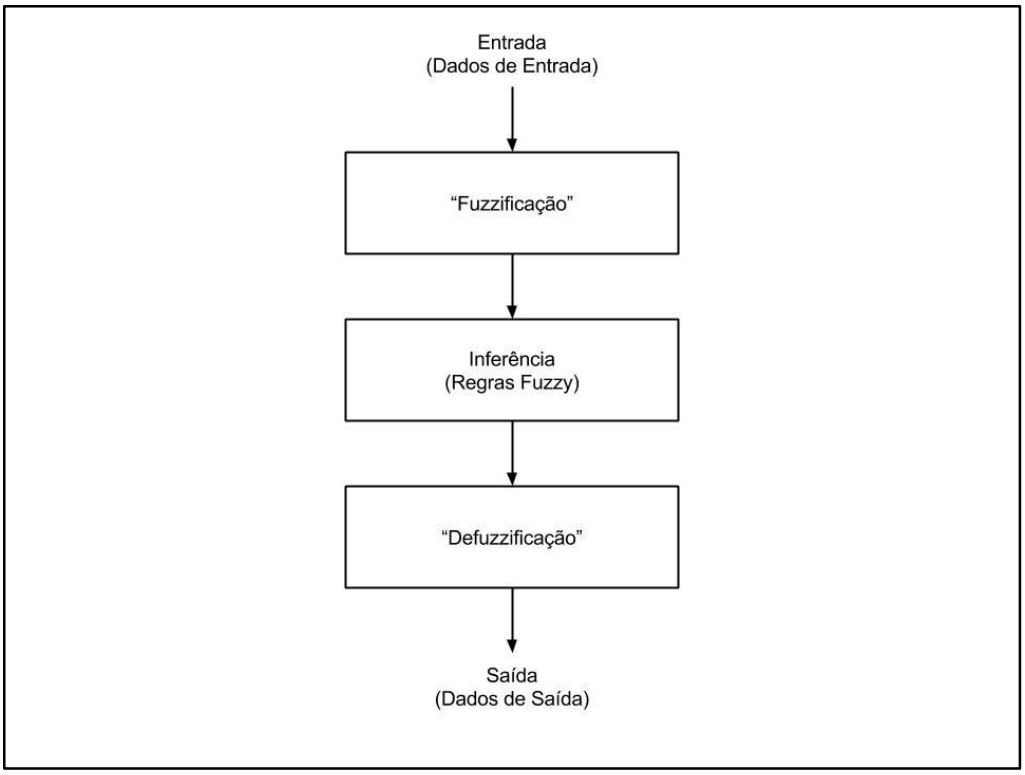
(0), isto é, são aceitos diversos graus de verdade (SCHWAB, 2009). Para Cox (1999), essa característica é muito importante, pois permite representar modelos de maneira mais precisa e semelhante ao mundo real, no qual não existe apenas condições extremas como verdadeiro e falso.

Um exemplo do poder de representação da lógica *fuzzy* em relação ao da lógica booleana poderia ser uma pessoa ferida. Na lógica booleana pode-se representar uma pessoa como estando ou não ferida, já na lógica *fuzzy* pode-se representar o quanto uma pessoa está ferida em diferentes graus, como: pouco, razoavelmente, muito, etc.

Suponha a existência de uma variável chamada ferimento, nomeia-se então, de conjunto *fuzzy* todos os valores possíveis para essa variável que se encontram no intervalo entre 0 e 1. Ainda, nomeia-se de grau de pertinência o valor numérico armazenado nessa variável em um determinado momento, ou seja, se em algum momento a variável ferimento possuir o valor 0.9, dizemos que 0.9 é seu grau de pertinência.

A utilização da lógica *fuzzy* consiste na aplicação de um processo conhecido como processo de inferência *fuzzy*, que consiste em três processos distintos que recebem o nome de: “fuzzificação”, inferência e “defuzzificação”. A figura 9 ilustra o processo de inferência *fuzzy*.

Figura 9. Processo de inferência *fuzzy*.



Fonte: Elaborada pelo autor

A "fuzzificação" é o processo no qual dados comuns de entrada são convertidos em graus de pertinência de um conjunto *fuzzy*, isso ocorre por meio de uma função chamada de função de pertinência.

A inferência *fuzzy* é o processo no qual são tomadas decisões baseadas nos valores dos graus de pertinência, isso ocorre por meio da combinação de operadores lógicos *fuzzy*. A tabela 1 exibe os operadores lógicos clássicos e seus equivalentes na lógica *fuzzy*, os operadores lógicos *fuzzy*.

A "defuzzificação" é o processo no qual os graus de pertinência modificados durante o processo de inferência são convertidos novamente para dados comuns, isso ocorre através da aplicação de uma função de conversão.

Tabela 1. Operadores Lógicos *Fuzzy*.

Operadores Lógicos Fuzzy		
Expressão	**Equivalente**	**Equação Fuzzy**
NOT A		1 - ma
A AND B		min(ma, mb)
A OR B		max(ma,mb)
A XOR B	NOT(B) AND A NOT(A) AND B	min(ma,1 - mb) min(1 - ma, mb)
A NOR B	NOT (A OR B)	1 - max(ma, mb)
A NAND B	NOT (A AND B)	1 - min(ma, mb)

Fonte: Elaborada pelo autor com base em Mellington e Funge, 2009.

Entendido o básico sobre a lógica *fuzzy*, pode-se entender o funcionamento da máquina de estados *fuzzy*. A máquina de estados *fuzzy*, como dito anteriormente, é uma técnica que permite mesclar comportamentos atribuídos a diferentes estados com a intenção de criar comportamentos complexos ou imprevisíveis. Isso é possível, porque esse tipo de máquina possui a capacidade de se manter em mais de um estado ao mesmo tempo.

Ainda, na máquina de estados *fuzzy* a ativação de cada um de seus estados dependerá do valor calculado para o grau de pertinência do mesmo, isto é, quando o valor calculado para o grau de pertinência de um determinado estado se igualar ou superar o valor requerido para ativação, as ações atribuídas a esse estado serão executadas.

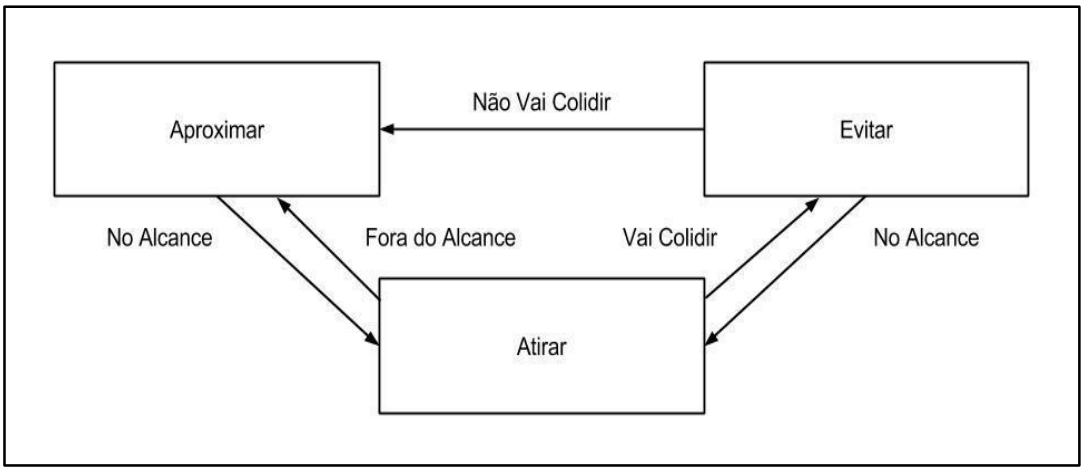
Uma forma de ilustrar o comportamento desse tipo de máquina de estados, é a modelagem de comportamento de uma nave autônoma para um jogo semelhante ao jogo Asteroids. A nave autônoma deverá possuir a capacidade de se aproximar dos asteróides, atirar nos asteróides quando estiver próximo o suficiente e evitar os asteróides vindo em sua direção. Para isso serão necessários três estados, que são: aproximar, atirar e evitar.

Quando a nave se encontrar no estado “aproximar”, deverá selecionar um

asteróide e se aproximar dele até chegar a uma distância na qual seja possível atirar; Nesse momento, a nave detectará que está na distância necessária e passará para o estado “atirar”, no qual irá disparar contra o asteróide; Caso algum asteróide esteja em rota de colisão, a nave entrará no estado “evitar”, no qual sairá do caminho do asteróide, ainda, se o asteróide estiver no alcance da nave ela passará para o estado “atirar”, caso o contrário passará para o estado “aproximar”.

Numa máquina de estados não determinística ou determinística, a nave autônoma só poderia estar em um estado por vez, isto é, em um determinado momento ela poderia estar no estado “aproximar” ou “atirar” ou “evitar”. A figura 10 ilustra uma possível máquina de estados determinística para o jogo *Asteroids*.

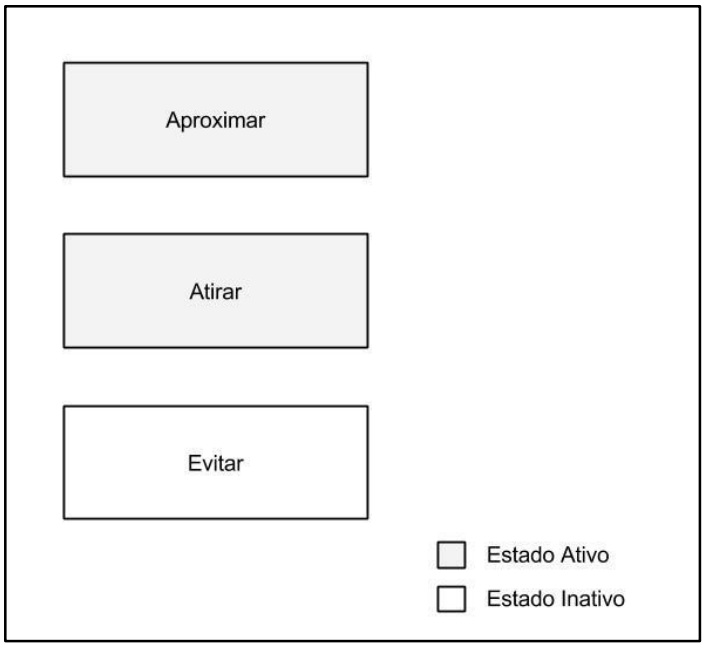
Figura 10. Possível máquina de estados determinística para *Asteroids*.



Fonte: Elaborado pelo autor.

Já na máquina de estados *fuzzy*, a nave autônoma poderia estar em qualquer número de estados ao mesmo tempo, isto é, num determinado momento ela poderia estar no estado “aproximar” ou “aproximar” e “atirar” e assim por diante. A figura 11 ilustra uma possível máquina de estados *fuzzy* para o jogo *Asteroids*.

Figura 11. Possível máquina de estados *fuzzy* para o jogo *Asteroids*.

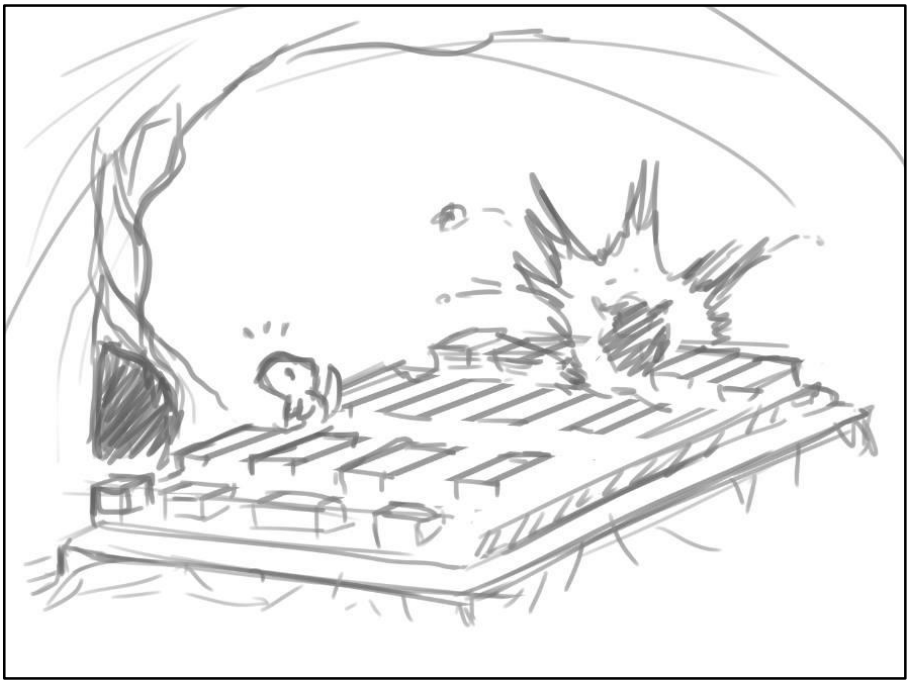


Fonte: Elaborado pelo autor.

4. O JOGO

Dragon's Den é um jogo simples de ação e estratégia que se passa dentro de um labirinto, no qual quatro filhotes de dragão competem pelo domínio de um covil. O jogo é inspirado no clássico *Bomberman*, desenvolvido originalmente pela Hudson Soft, em 1983. A figura 12 ilustra o conceito de uma partida do jogo *Dragon's Den*.

Figura 12. Conceito de uma partida do jogo *Dragon's Den*.



Fonte: FORMAGIO, Bruno, 2014.

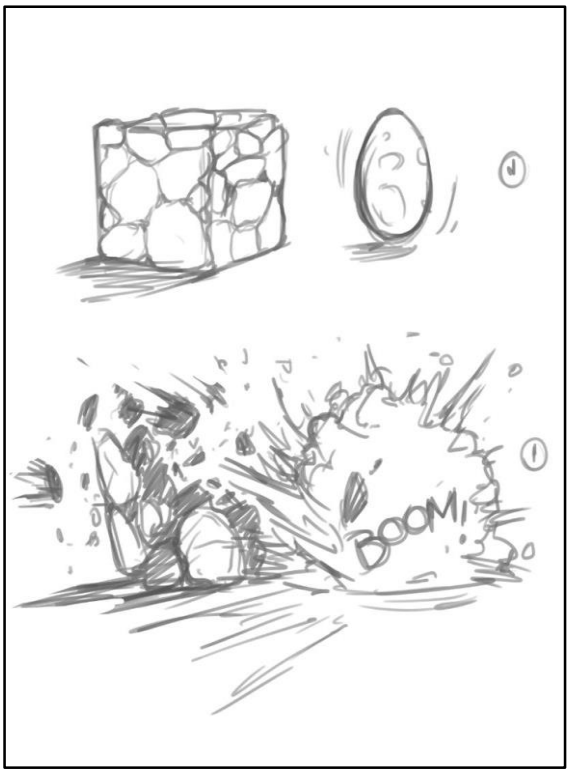
O objetivo principal do jogo é eliminar todos os inimigos controlados pelo computador que se encontram dentro do labirinto para se tornar o mestre do covil, para isso, é necessário abrir caminho destruindo os obstáculos, de forma que seja possível alcançar seus oponentes.

Os obstáculos presentes no labirinto podem ser divididos em duas categorias

distintas, que são: obstáculos destrutíveis e obstáculos indestrutíveis. Os obstáculos indestrutíveis não podem ser destruídos de nenhuma forma, já os obstáculos destrutíveis podem ser eliminados utilizando ovos de dragão explosivos.

Os ovos de dragão explosivos são as armas utilizadas pelos filhotes de dragão para quebrar obstáculos destrutíveis, destruir inimigos ao longo do caminho ou disparar outros ovos que estejam em seu raio de explosão. O raio da explosão original dos ovos é de um quadrado de distância e o tempo para explosão é de três segundos, é possível ampliar sua área de explosão por meio de amplificadores de poder. A figura 13 ilustra o conceito do ovo de dragão explosivo presente no jogo *Dragon's Den*.

Figura 13. Conceito do ovo explosivo no jogo *Dragon's Den*.



Fonte: FORMAGIO, Bruno, 2014.

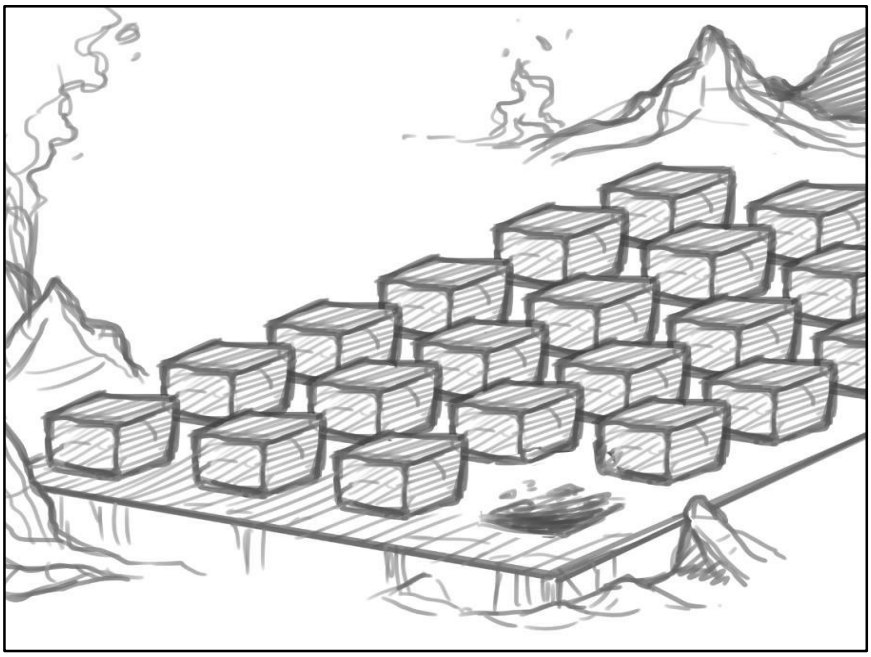
Os amplificadores de poder são objetos que surgem quando um obstáculo é destruído, esses objetos, quando coletados amplificam a capacidade do filhote de dragão. Atualmente, existem dois amplificadores disponíveis, o primeiro deles expande a área de explosão dos ovos e o segundo aumenta a velocidade de locomoção do filhote de dragão. No futuro serão adicionados novos amplificadores.

4.1 Cenário

As partidas do jogo *Dragon's Den* se passam dentro de um covil de dragões, esse covil nada mais é do que um calabouço medieval com o chão feito de pedras, no qual existem diversos obstáculos. Esses obstáculos podem ser pedras grandes ou pequenas. As pedras grandes não podem ser destruídas com os ovos explosivos e por isso pertencem ao grupo de obstáculos indestrutíveis, já as pedras pequenas podem ser destruídas com os ovos explosivos e fazem parte do grupo de obstáculos destrutíveis. A figura 14 ilustra o conceito do cenário para o jogo *Dragon's Den*.

Ainda, a cada nova partida a posição dos obstáculos no mapa é alterada aleatoriamente, de forma que a cada nova partida a aparência do mapa seja diferente da partida anteror.

Figura 14. Conceito do cenário do jogo *Dragon's Den*.



Fonte: FORMAGIO, Bruno, 2014.

4.2 Personagens

Os personagens do jogo *Dragon's Den*, são quatro filhotes de dragão recém-nascidos, que devido à natureza de sua espécie devem lutar entre si para demonstrar qual deles é o mais apto a sobreviver e a dominar o covil em que nasceram.

Ainda, pode-se descrever os filhotes de dragão como pequenos lagartos bípedes que possuem um par de asas pequenas. Eles possuem a capacidade de botar ovos explosivos e utilizam esses ovos para destruir obstáculos presentes no covil e atacar outros filhotes de dragão. A figura 15 ilustra o conceito do filhote de dragão para o jogo *Dragon's Den*.

Figura 15. Conceito do filhote de dragão do jogo *Dragon's Den*.

Fonte: FORMAGIO, Bruno, 2014.

4.3 Itens

Durante uma partida de *Dragon's Den* podem surgir diversos objetos no mapa, entretanto existe apenas um tipo de item coletável, esse tipo de item é conhecido como amplificador.

Os amplificadores podem surgir quando pedras pequenas são destruídas e como o próprio nome sugere, são utilizados para amplificar o poder dos filhotes de dragão de alguma forma. A figura 16 ilustra o conceito dos amplificadores no jogo *Dragon's Den*.

Figura 16. Conceito dos amplificadores do jogo *Dragon's Den*.

Fonte: FORMAGIO, Bruno, 2014.

Atualmente, existem apenas dois tipos de amplificadores implementados, o primeiro deles permite ampliar a área de explosão dos ovos e o segundo permite ampliar a velocidade de locomoção do filhote de dragão.

4.4 Controle

Devido ao fato de *Dragon's Den* ser um jogo para computador e possuir uma mecânica extremamente simples, para controlar um personagem basta utilizar as teclas direcionais do teclado e o botão "z" para liberar uma bomba no local desejado.

5 MODELAGEM DA MÁQUINA DE ESTADOS

Este capítulo tem como objetivo descrever o processo de modelagem da máquina de estados e os resultados obtidos com a prototipação desse modelo.

Dessa forma, na subseção 5.1, aborda-se a modelagem da máquina de estados e todo o processo de pensamento realizado para chegar a sua definição. Já na subseção 5.2 descreve-se, detalhadamente, o funcionamento de cada estado encontrado na máquina. Por último, na subseção 5.3, descreve-se os resultados obtidos com a implementação da máquina de estados proposta.

5.1 Processo de Modelagem da Máquina de Estados

O principal objetivo para o desenvolvimento do jogo *Dragon's Den*, é o estudo da técnica conhecida como máquina de estados para modelar o comportamento de um agente inteligente, ou seja, desenvolver um módulo de inteligência artificial que seja capaz de controlar personagens e propiciar um desafio divertido para um oponente humano. Em jogos digitais os módulos de inteligência artificial utilizados para controlar personagens são, geralmente, chamados de *bots* (diminutivo da palavra *robots*, que em português significa robôs).

Durante o desenvolvimento, fez-se um estudo sobre as variações existentes do algoritmo de máquina de estados que são utilizados em jogos digitais, de modo a entender seus pontos fortes e fracos, com o objetivo de se escolher a melhor variante para se implementar durante o desenvolvimento do jogo.

Em seguida, realizou-se uma análise do jogo *Bomberman*, no qual se observou, principalmente, o comportamento exibido pelos personagens controlados pelo computador, o objetivo foi extrair o máximo de informações possíveis e compilá-las num pequeno documento para ser utilizado durante o processo de modelagem

da máquina de estados para o jogo. A tabela 2 ilustra as ações encontradas nos personagens do jogo original (*Bomberman*).

Tabela 2. Ações desempenhadas pelo personagem no jogo *Bomberman*.

Ações Personagem
Encontrar parede próxima e acessível
Encontrar power-up próximo e acessível
Encontrar oponente próximo e acessível
Procurar refúgio quando em perigo
Movimentar-se até parede, power-up ou oponente
Ordem de prioridade: oponente, power-up, parede
Aleatoriedade na movimentação

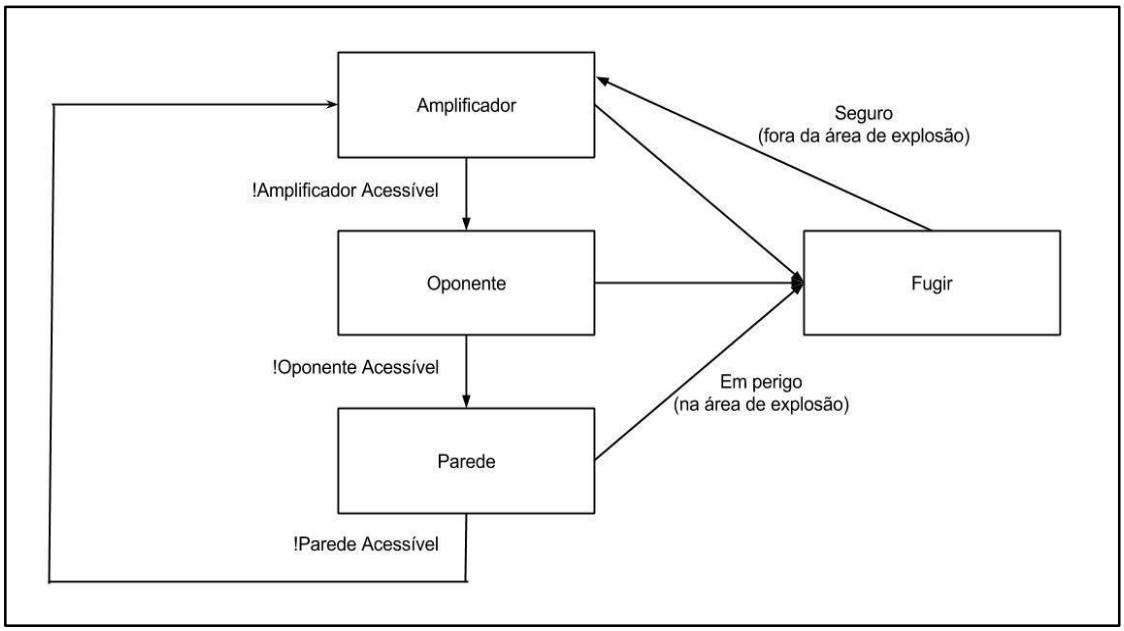
Fonte: Elaborada pelo autor.

Após a análise dos diversos algoritmos disponíveis e do comportamento apresentado pela inteligência artificial presente no jogo original, deu-se início a modelagem da máquina de estados destinada a controlar os filhotes de dragão presentes no jogo.

Na primeira versão da máquina de estados, optou-se pela utilização de uma máquina de estados determinística, na qual foram estabelecidos quatro estados, que são: “amplificador”, “oponente”, “parede” e “fuga”. Ao ser iniciada, a máquina é colocada no estado “amplificador”, fazendo com que o personagem se dirija ao amplificador mais próximo e acessível para coletá-lo, caso não existam amplificadores ou os mesmos não estejam acessíveis, a máquina fará uma transição para o estado “oponente”. No estado “oponente”, o personagem irá se dirigir ao oponente mais próximo e acessível quando houver, objetivando depositar um ovo explosivo próximo a ele, caso contrário a máquina realizará uma transição para o estado “parede”. No estado “parede”, o personagem irá se dirigir à parede mais próxima e acessível para depositar um ovo na área adjacente, caso não exista

parede ou as paredes estejam inacessíveis, o personagem retornará ao estado “amplificador”. O estado “Fugir”, é disparado a partir de qualquer um dos três estados anteriores quando o personagem detecta que está em uma área insegura, ou seja, na área de explosão de uma bomba. A figura 17 ilustra a primeira versão da máquina de estados utilizada pela IA.

Figura 17. Máquina de estados utilizada pela IA (Versão 1).

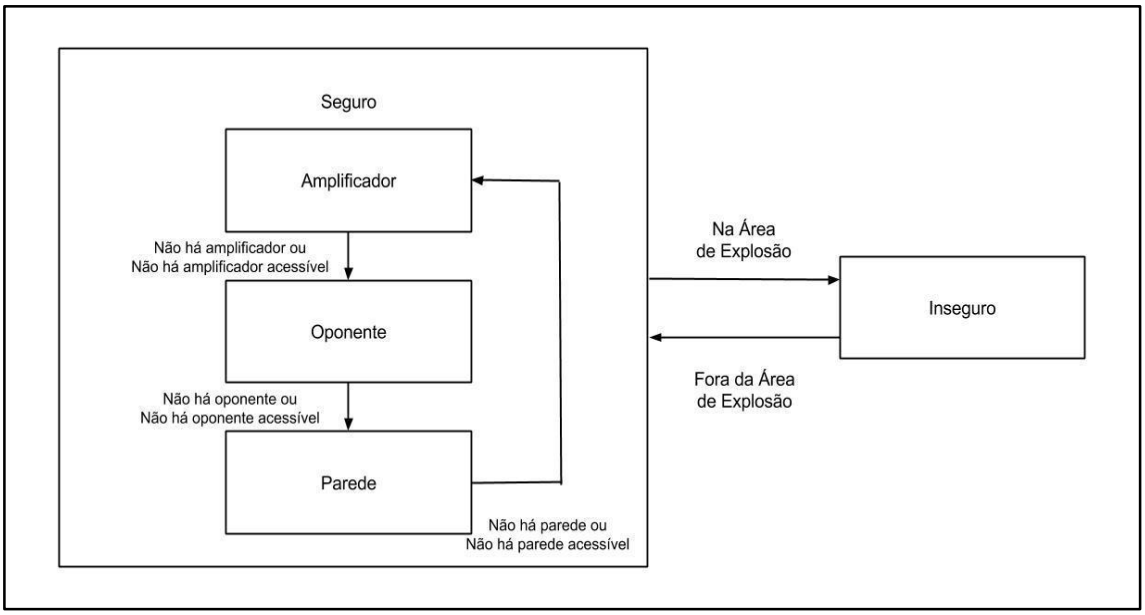


Fonte: Elaborada pelo autor.

A primeira versão apesar de experimental foi capaz de prover as funcionalidades básicas necessárias, entretanto devido à necessidade e se priorizar o instinto de sobrevivência provido através do estado “fugir”, foi necessário utilizar-se de diversas transições o que aumentou o grau de complexidade representacional dessa máquina. Optou-se então, pela remodelagem dessa máquina com o objetivo de reduzir o número de transições e simplificar o modelo, para isso foram combinados dois algoritmos distintos, que são: a máquina de estados finitos determinísticos e a máquina de estados hierárquica. O motivo por trás dessa decisão foi a percepção de que as ações referentes aos estados “amplificador”,

“oponente” e “parede”, só podem ser executadas quando o personagem estiver protegido das explosões e a ação “fugir” só deve ser executada quando o personagem não estiver protegido das explosões. Tendo isso em mente, criou-se uma nova máquina de estados contendo apenas dois estados que são “seguro” e “inseguro”. Ao estado “inseguro” foi delegada a tarefa que pertencia ao estado “fugir”, já ao estado “seguro”, foi delegada a tarefa de representar uma nova máquina de estados contendo os estados “amplificador”, “oponente” e “parede” que já existiam na primeira versão. A figura 18 ilustra a segunda versão da máquina de estados utilizada pela IA.

Figura 18. Máquina de estados utilizada pela IA (Versão 2).



Fonte: Elaborada pelo autor.

Após a remodelagem, observou-se a simplificação da máquina de estados, a redução do número de transições e tornou-se explícita a relação existente entre as tarefas e as condições necessárias para se executá-las.

5.2 Descrição Detalhada dos Estados

Após a modelagem e definição da máquina de estados, iniciou-se a etapa de implementação da mesma. No início dessa etapa, analisou-se todas as ações necessárias para o correto funcionamento de cada estado, de forma a entendê-las e abstraí-las em componentes reutilizáveis para simplificar e agilizar o desenvolvimento do jogo.

Ainda, durante essa etapa observou-se que a maioria dos estados podiam ser decompostos em quatro passos distintos, que são: busca do alvo, cálculo do caminho, movimentação ao longo do caminho calculado e implantação do ovo (bomba). Também observou-se que existiam apenas duas variações possíveis para esse fluxo.

Na primeira variação, deve-se executar a busca pelo alvo, o cálculo do caminho, a movimentação e a implantação da bomba nas devidas condições. Essa variação é utilizada pelos sub estados “oponente” e “parede”. A figura 19 ilustra o algoritmo utilizado pelos sub estados “oponente” e “parede”.

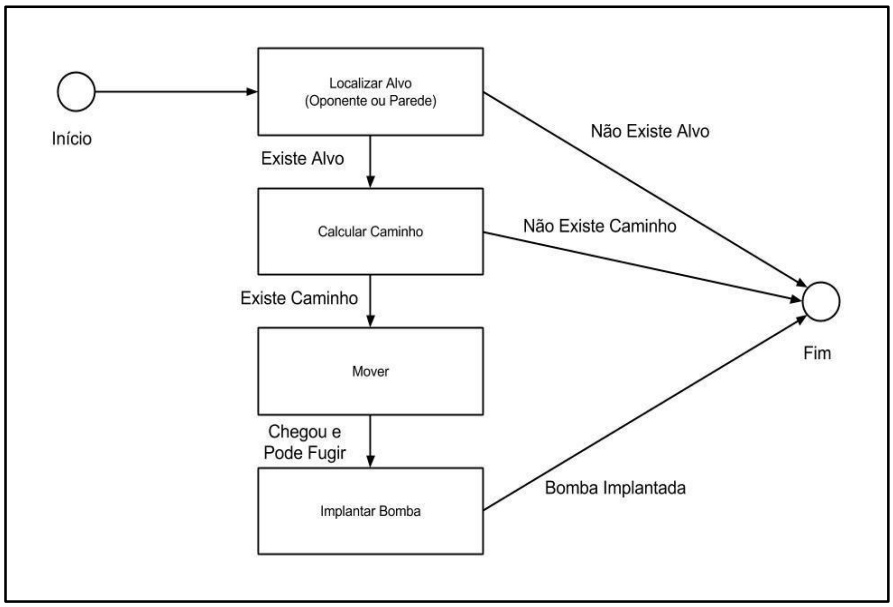
Já na segunda variação, executam-se todos os passos presentes na primeira variação, com exceção da implantação da bomba. Essa variação é utilizada pelo sub estado amplificador e pelo estado inseguro. A figura 20 ilustra o algoritmo utilizado pelo sub estado “amplificador” e pelo estado “inseguro”.

No processo “localizar alvo”, itera-se sobre todos os alvos de um determinado tipo seja ele amplificador, oponente, parede ou local seguro. Calcula-se então, a distância euclidiana entre o personagem e o alvo e por fim recupera-se a posição (coordenada cartesiana) do alvo mais próximo para se calcular o caminho até ele.

Já no processo “calcular caminho”, utiliza-se de um componente chamado *Simply A**, esse componente é uma implementação do Algoritmo A*, um algoritmo de busca em grafos baseado no Algoritmo de Dijkstra e no Algoritmo de Busca em Largura. O *Simply A** gera um grafo sobre o cenário durante a inicialização do jogo e

fornece uma interface que é utilizada para calcular o caminho entre um ponto e outro, esse cálculo retorna uma lista de pontos (coordenadas cartesianas), do ponto A ao ponto B.

Figura 19. Algoritmo utilizado pelos sub estados “oponente” e “parede”.

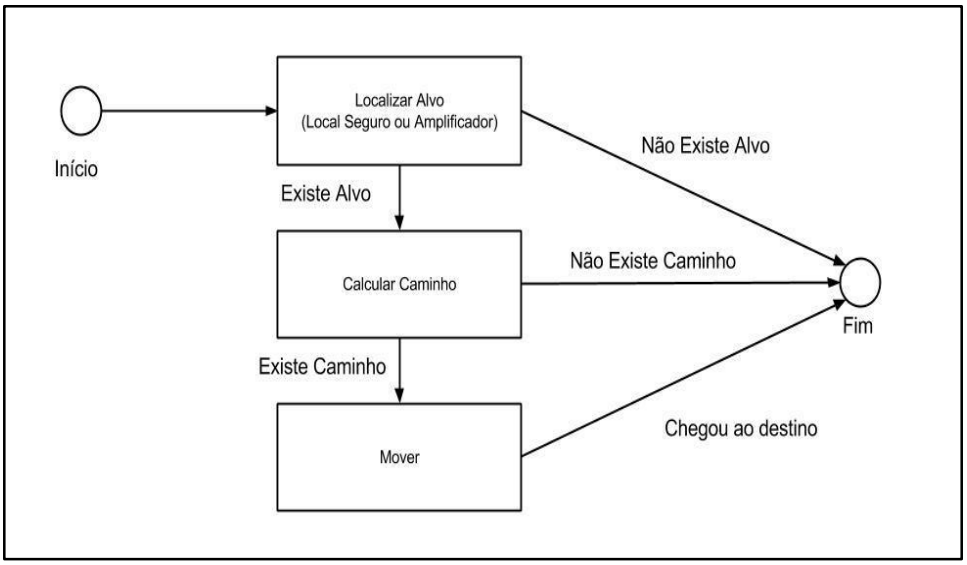


Fonte: Elaborada pelo autor.

No processo “mover”, itera-se sobre a lista de pontos retornada pelo componente *Simply A**, e move-se o personagem ponto a ponto até chegar a posição final.

E por último, no processo “Implantar Bomba”, calcula-se a distância entre o personagem e o alvo, quando a distância entre ambos for menor ou igual a um, a bomba será implantada e o personagem entrará no estado “fugir”.

Figura 20. Algoritmo utilizado pelos estados "amplificador" e "inseguro".



Fonte: Elaborada pelo autor.

5.3 Resultados

Durante o desenvolvimento do protótipo foi possível implementar com sucesso a máquina de estados proposta, entretanto encontrou-se um grande problema referente ao componente *Simply A**.

A implementação da busca de caminhos fornecida pelo *Simply A** demonstrou-se ineficiente nos casos em que o mapa continha um elevado número de obstáculos ou inimigos, pois nesses dois cenários foram registrados elevados índices de consumo de memória e tempo de processamento.

Em decorrência desse consumo, não foi possível implementar todas as funcionalidades necessárias para se realizar uma partida do jogo no tempo disponível, portanto implementou-se dois modos de demonstração, no primeiro é possível realizar uma batalha de quatro entidades de IA, e no segundo pode-se realizar uma batalha de um jogador contra quatro entidades de IA.

Realizaram-se então dez partidas no modo IA x IA, nessa partida cada

entidade de IA é representada por uma cor, as cores são: Amarelo, Azul, Verde, e Vermelho. Coletou-se então, dados sobre o tempo de jogo e o vencedor de cada uma das partidas, o objetivo era descobrir se a máquina de estado modelada iria operar corretamente e se uma batalha entre diversas entidades controladas pela IA chegaria a um final. A tabela 3 exibe os dados da partida IA x IA.

Tabela 3. Dados da partida IA x IA.

Partidas		
Partida	Tempo de Jogo (s)	Vencedor
1	98,04	Verde
2	81,22	Verde
3	68,88	Amarelo
4	24,95	Verde
5	97,32	Amarelo
6	63,67	Verde
7	55,03	Azul
8	32,65	Amarelo
9	111,66	Verde
10	152,42	Verde

Fonte: Elaborada pelo autor.

Ao analisar os dados coletados foi possível observar que as partidas tiveram um bom tempo de duração que variou de 25 à 153 segundos, o que demonstrou o bom funcionamento da máquina de estados. Observou-se, também, que todas as partidas tiveram um final, o que garante que mesmo numa batalha entre entidades controladas pela IA a partida sempre chegará a um final.

Ainda realizaram-se mais dez partidas no modo Jogador x IA, nas quais coletou-se, novamente, o tempo de jogo e o vencedor de cada uma das partidas, o objetivo, dessa vez, era descobrir se as entidades controladas pela IA poderiam ser derrotadas por um jogador. A tabela 4 exibe os dados da partida Jogador x IA.

Tabela 4. Dados da partida Jogador x IA.

Partidas		
Partida	Tempo de Jogo (s)	Vitória
1	39,66	Jogador
2	48,28	Jogador
3	40,56	Azul
4	48,59	Jogador
5	46,33	Amarelo
6	242,31	Vermelho
7	21,62	Verde
8	29,16	Amarelo
9	20,98	Amarelo
10	51,55	Azul

Fonte: Elaborada pelo autor.

Ao analisar os dados, observou-se que a IA apresenta um certo nível de dificuldade para um oponente humano, uma vez que houveram apenas três vitórias e sete derrotas.

6 DETALHES DE IMPLEMENTAÇÃO

Esse capítulo tem como objetivo descrever a metodologia e as ferramentas utilizadas no desenvolvimento do projeto.

Sendo assim, na seção 6.1, explica-se a metodologia utilizada e na seção 6.2 descreve-se as ferramentas que foram utilizadas.

6.1 Metodologia

Durante o desenvolvimento do protótipo da IA para o jogo *Dragon's Den*, optou-se pela utilização da metodologia *Scrum*. O *Scrum* é uma metodologia ágil utilizada no gerenciamento e planejamento de projetos de desenvolvimento de software que envolvem equipes grandes (LIMA, 2014), entretanto, por oferecer uma estrutura simples para organização e execução de tarefas, pode ser de grande valia para o desenvolvimento de projetos individuais.

Ao utilizar essa metodologia, os projetos são divididos em etapas ou ciclos (semanais ou mensais) que recebem o nome de *sprints* (corridas). Um *sprint* consiste num *time box* (intervalo de tempo), no qual um conjunto de tarefas devem ser realizadas e acompanhadas através de reuniões diárias que são chamadas de *daily scrum*. No final de cada *Sprint,* realiza-se a chamada *sprint review meeting* (encontro de revisão do *sprint*), no qual são apresentados as funcionalidades implementadas nesse ciclo. Após a *sprint review meeting* é realizada a *sprint retrospective* (retrospectiva do *sprint*), na qual a equipe pode expressar suas dificuldades e propor soluções para que elas não ocorram no futuro.

Ainda, no *Scrum*, as funcionalidades a serem implementadas num determinado projeto são mantidas em uma lista chamada de *product backlog* (lista de requisitos), que é especificada pelo *product owner* (dono do produto). Durante o

início de cada *sprint*, ocorre o *sprint planning meeting* (encontro de planejamento do *sprint*), no qual o cliente e a equipe se reunem para determinar quais das funcionalidades devem ser implementadas neste ciclo, essa decisão é baseada na prioridade atribuída a cada funcionalidade. Em seguida, as tarefas selecionadas são transferidas do product backlog para o *sprint backlog* (lista de atividades), para que sejam executadas durante o *sprint*. A figura 21 ilustra os processos realizados ao se utilizar a metodologia *Scrum*.

Figura 21. Processos realizados ao se utilizar Scrum.

Fonte: LIMA, Rafael, 2014.

Entendido o *Scrum*, pode-se entender a metodologia utilizada no desenvolvimento do protótipo. Inicialmente, foi criada uma lista contendo todas as funcionalidades que deveriam ser entregues ao final do projeto, ou seja, foi criado nada mais do que um *product backlog* (lista de requisitos). A tabela 5 exibe a lista de requisitos para a implementação da inteligência artificial baseada em máquina de estados.

Tabela 5. Lista de requisitos

Requisitos
Buscar Caminho
Buscar Parede
Buscar Amplificador
Buscar Oponente
Buscar Local Seguro
Movimentação
Colocar Bomba
Máquina de Estados
Testes
Polimento

Fonte: Elaborada pelo autor.

Em seguida, a lista de requisitos foi analisada em busca dos requisitos mais críticos para se estabelecer o período de tempo médio para a realização de cada *sprint*, esse período ficou estabelecido como sendo de uma semana.

Deu-se, então, o início ao desenvolvimento do protótipo do jogo tentando sempre respeitar o tempo médio estabelecido para a realização dos *sprints*, entretanto, houveram casos nos quais necessitou-se de mais tempo do que o estipulado para se executar as tarefas.

6.2 Ferramentas

Esta seção tem como objetivo apresentar as ferramentas que foram utilizadas no desenvolvimento do protótipo do jogo *Dragon's Den*.

Na subseção 6.2.1, apresenta-se a *Unity Game Engine*, uma plataforma para desenvolvimento de jogos multiplataforma desenvolvida pela *Unity Technologies*; Já,

na subseção 6.2.2, descreve a linguagem de programação *C#* que foi utilizada para desenvolver lógica do jogo; A subseção 6.2.3, apresenta o ambiente integrado de desenvolvimento utilizado, neste caso o Microsoft Visual Studio. A subseção 6.2.4, apresenta o serviço *BitBucket* que foi utilizado para hospedar e versionar o projeto.

6.2.1 Unity Game Engine

A *Unity Game Engine* é uma plataforma para desenvolvimento de jogos multiplataforma, desenvolvida pela *Unity Technologies*, em 2005. Atualmente, encontra-se na versão 4.3.4 e está disponível para as plataformas *Windows* e *Mac OS*.

A *Unity* é construída sobre uma implementação *open source* do *.NET Framework* chamada *Mono*, e suporta a escrita de programas em três linguagens de programação distintas, que: são *C#*, *JavaScript* e *Boo*. A biblioteca gráfica do *Unity* se utiliza das bibliotecas *DirectX (Windows)*, *OpenGL (Windows, Linux, Mac OS)*, *OpenGL ES (Android e IOS)* e diversas outras proprietárias. Além disso, a *Unity* permite exportar executáveis para as plataformas: *Windows*, *Linux*, *Mac OS*, *Android*, *IOS*, *Windows Phone*, *BlackBerry*, *Web* (*Flash* e *Unity Web Player*), *PS3*, *Xbox 360* e *Wii U*.

A *Unity* acelera o processo de desenvolvimento oferecendo um rápido fluxo de trabalho baseado no editor de cenas e componentização de software. Por esses e outros motivos a *Unity* foi escolhida como a ferramenta para o desenvolvimento desse projeto.

6.2.2 Linguagem de Programação C#

A linguagem de programação *C#* (lê-se *C Sharp*), é uma linguagem orientada a objetos desenvolvida pela *Microsoft* para fazer parte do *.NET Framework*. A linguagem *C#* é inspirada nas linguagens *C++* e *Java*, das quais herdou muitas de suas características.

Os programas escritos em *C#* podem ser executados sobre *.NET Framework*, Mono ou qualquer outra implementação da *Common Language Infrastructure* (*CLI*), a *CLI* é uma especificação aberta desenvolvida pela Microsoft e padronizada pela *International Organization for Standardization* (*ISO*) e *European Computer Manufacturers Association* (*ECMA*). Nessa especificação são definidos: um ambiente virtual de execução chamado de *Common Language Runtime* (*CLR*), uma linguagem intermediária para ser executada sobre a *CLR* chamada de *Common Intermediate Language* (*CIL*) e uma biblioteca de código padrão que define recursos básicos e avançados da linguagem chamada de *Standard Library* (*SL*). A existência do *CLI* é muito importante, pois torna possível a execução de programas escritos em *C#* nas mais diversas plataformas, desde que essas possuam uma implementação da especificação *CLI*.

Além disso, o *C#* encontra-se disponível entre as três linguagens suportadas pelo *Unity*, o que contribuiu para a sua utilização nesse projeto.

6.2.3 Microsoft Visual Studio

O *Visual Studio* é um ambiente integrado de desenvolvimento da *Microsoft*, que é utilizado para desenvolver aplicações para o sistema operacional *Microsoft Windows*, também é possível desenvolver aplicações para web e jogos.

O *Visual Studio* suporta um amplo conjunto de linguagens de programação, como: *C#*, *C++*, *C*, *Visual Basic*, *JavaScript*, etc. Além disso, oferece um grande conjunto de ferramentas, como: editor de código, autocompletude de código,

ferramentas de refatoração, ferramentas de depuração, e etc.

O *Visual Studio*, também integra-se a *Unity Game Engine*, o que foi um fator importante para a sua utilização nesse projeto.

6.2.4 Bitbucket

O *Bitbucket* é um serviço para a hospedagem de projetos em nuvem gratuito ou comercial, oferecido pela empresa *Atlassian*. Suporta a hospedagem para projetos que se utilizam dos sistemas de controle e revisão distribuídos (SCRD) *Mercurial* ou *Git*. Os SCRDs são utilizados para criar, compartilhar, versionar e manter backups de repositórios de código.

O *Bitbucket* oferece planos de hospodeagem gratuitos e comerciais, no plano gratuito é possível possuir repositórios privados e públicos ilimitados, porém só é possível compartilhá-los com até cinco desenvolvedores, já os planos pagos permitem que mais desenvolvedores acessem os repositórios.

Por esses e outros motivos o *Bitbucket* foi escolhido como ferramenta de backup e versionamento desse projeto.

7 CONSIDERAÇÕES FINAIS

A inteligência artificial está se tornando um tópico cada vez mais relevante no desenvolvimento de jogos digitais, pois conforme esse mercado cresce, cresce também a exigência de seus consumidores por jogos mais desafiadores e de maior qualidade. Atualmente, a maioria dos jogos utilizam técnicas de inteligência artificial, e, portanto, torna-se necessário aos desenvolvedores interessados nessa área o conhecimento de algumas delas.

Esse trabalho realizou um levantamento bibliográfico no qual se estudou uma técnica de inteligência artificial chamada de máquina de estados e suas aplicações em jogos digitais. Em seguida, aplicou-se esse conhecimento no desenvolvimento do protótipo do jogo *Dragon's Den*, objetivando a criação de um personagem que exibisse um comportamento inteligente quando observado por seus jogadores.

A partir do levantamento bibliográfico realizado, foi possível observar que o algoritmo de máquina de estados é uma estrutura muito simples que pode ser implementada utilizando apenas desvios condicionais. Observou-se, também, que apesar da simplicidade na sua implementação, esse algoritmo apresenta-se como uma forma eficiente de implementar tarefas extensas ou complexas, pois encoraja a decomposição das mesmas em tarefas menores o que simplifica a implementação e evolução das funcionalidades.

Durante o desenvolvimento do jogo, realizou-se uma análise do jogo *Bomberman* (a versão original), na qual foi possível estudar o comportamento exibido pelas entidades controladas pelo computador. Essa análise foi de suma importância, pois forneceu as informações necessárias para a modelagem da máquina de estados.

Por fim, construiu-se um protótipo, no qual foi possível observar que os personagens controlados pelo computador conseguiram desempenhar as tarefas a eles atribuídas de maneira correta.

Apesar do bom resultado obtido técnica, encontrou-se um problema com o

componente *Simply A** que é o responsável pela busca de caminhos, e portanto não foi possível implementar todas as funcionalidades previstas para o protótipo. No futuro, planeja-se substituir o *Simply A** por uma solução mais madura e implementar o jogo completamente, também haverá uma remodelagem da máquina de estados para melhorar o desempenho no que diz respeito a aleatoriedade.

REFERÊNCIAS

BOURG, David M.; SEEMAN, Glenn. **AI For Game Developers**. Sebastopol, CA: O'reilly Media, 2004.

BUCKLAND, Mat. **Programming Game AI By Example**. Plano, TX: Wordware Publishing, 2004.

COX, Earl; O'HAGAN, Michael. **The Fuzzy Systems Handbook:** A Practitioner's Guide to Building, Using, and Maintaining Fuzzy Systems. 2. ed. San Diego, CA: Academic Press, 1999.

FORMAGIO, Bruno, 2014.

GREGORY, Jason. **Game Engine Architecture.** Wellesley, Ma: A K Peters L, 2009.

HOPCROFT, John E.; MOTWANI, Rajeev; MOTWANI, Rajeev. **Introduction to Automata Theory, Languages, and Computation.** 3. ed. Boston, MA: Addison Wesley, 2006.

LIMA, Rafael. **SCRUM.** Disponível em: <http://desenvolvimentoagil.com.br/scrum/>. Acesso em: 28 maio 2014.

MELLINGTON, Ian; FUNGE, John. **Artificial Intelligence For Games**. 2. ed. Burlington, MA: Elsevier, 2009.

RUSSEL, Stuart; NORVIG, Peter. **Artificial Intelligence:** A Modern Approach. 3. ed. Upper Saddle River, NJ: Pearson, 2009.

SCHWAB, Brian. **AI Game Engine Programming**. Boston, MA: Course Technology, 2009.