

**UNIVERSIDADE ESTADUAL PAULISTA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

MATEUS GONÇALEZ ETTO

UTILIZAÇÃO DE INTELIGÊNCIA ARTIFICIAL EM JOGO RPG

**BAURU – SP
2016**

MATEUS GONÇALEZ ETTO

UTILIZAÇÃO DE INTELIGÊNCIA ARTIFICIAL EM JOGO RPG

Trabalho de Conclusão de Curso de graduação apresentado à disciplina Projeto e Implementação de Sistemas do curso de Bacharelado em Ciência da Computação da Faculdade de Ciências da Universidade Estadual Paulista "Júlio de Mesquita Filho" como requisito para obtenção do título de Bacharel em Ciência da Computação.

Orientadora: Profa. Dra. Simone das Graças Domingues Prado

MATEUS GONÇALEZ ETTO

UTILIZAÇÃO DE INTELIGÊNCIA ARTIFICIAL EM JOGO RPG

Trabalho de Conclusão de Curso de graduação apresentado à disciplina Projeto e Implementação de Sistemas do curso de Bacharelado em Ciência da Computação da Faculdade de Ciências da Universidade Estadual Paulista "Júlio de Mesquita Filho" como requisito para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA

Profa. Dra. Simone das Graças Domingues Prado

(Nome do segundo membro da Banca Examinadora)

(Nome do terceiro membro da Banca Examinadora)

Bauru, ____ de _____ de ____

RESUMO

Este projeto trata-se da criação de um protótipo de jogo no estilo RPG em turnos, em conjunto com a criação de uma aplicação de Inteligência Artificial capaz de controlar os personagens, assim como de aprender a controlá-los melhor com treinamento. Para a criação desta aplicação de Inteligência Artificial, foram usados conceitos de Redes Neurais Artificiais e Algoritmo Genético, e para a criação do jogo e seus scripts foi usado o motor de jogo Unity.

PALAVRAS-CHAVE: Inteligência Artificial, Unity, jogo RPG

ABSTRACT

This project comes to creating a turn-based RPG game prototype, in conjunction with an Artificial Intelligence application that is able to control the characters as well as to learn how to control them better with training. For the creation of the Artificial Intelligence application, concepts of Artificial Neural Networks and Genetic Algorithm were used, and for the creation of the game and its scripts, the Unity game engine was used.

KEY-WORDS: Artificial Intelligence, Unity, RPG game

LISTA DE ILUSTRAÇÕES

Figura 1 – A interface padrão da Unity	13
Figura 2 – Exemplo de classe no Visual Studio	15
Figura 3 – Funções de Ativação comuns em uma RNA	16
Figura 4 – RNA com uma camada oculta	17
Figura 5 – Importância da camada oculta	18
Figura 6 – Método da roleta para Seleção	20
Figura 7 – Passo a passo do AG	22
Figura 8 – Exemplo de <i>design</i> de carro	22
Figura 9 – RNA e AG juntos	23
Figura 10 – Rede Neural Artificial implementada	28
Figura 11 – Bônus de derrotar inimigo	33
Figura 12 – <i>Screenshot</i> do jogo	34
Figura 13 – <i>Fitness</i> durante 20 gerações	34

LISTA DE TABELAS

Tabela 1 – Descrição dos painéis na Unity	14
Tabela 2 – Representação binária de cromossomos	19
Tabela 3 – Ciclo do Algoritmo Genético	21
Tabela 4 – Atributos dos personagens	24
Tabela 5 – Tipos de danos	25
Tabela 6 – Parâmetro dos personagens	28
Tabela 7 – Valores de entrada na RNA	31
Tabela 8 – Significado dos valores de saída da RNA	31

SUMÁRIO

1	INTRODUÇÃO	10
1.1	Objetivos do Trabalho	11
1.1.1	Objetivo Geral	11
1.1.2	Objetivos Específicos	11
1.2	Organização da Monografia	12
2	FERRAMENTAS UTILIZADAS	13
2.1	Unity	13
2.1.1	Interface	13
2.2	Visual Studio	14
3	CONCEITOS	16
3.1	Redes Neurais Artificiais	16
3.1.1	Funcionamento	16
3.1.2	Tipos de aprendizado	18
3.2	Algoritmo Genético	19
3.2.1	Conceitos de AG	19
3.2.2	Processo	21
3.2.3	Exemplos de aplicação	22
3.2.4	Variação com RNA	23
4	O JOGO	24
4.1	Descrição do jogo	24
4.1.1	Descrição dos atributos	24
4.1.2	Cálculos de dano e velocidade	25
4.1.3	Comandos	26
4.1.4	Atributos dos personagens	27
4.2	Rede Neural Artificial Implementada	28
4.3	Algoritmo Genético Implementado	29
4.4	Funcionamento do jogo	30
4.4.1	Preparação da batalha	30
4.4.2	Durante a batalha	30
4.4.3	Após a batalha	33
5	RESULTADOS	34

REFERÊNCIAS 36

1 INTRODUÇÃO

Há ainda muitas pessoas que acreditam que jogos eletrônicos são para crianças ou pessoas desocupadas. Talvez isso fosse verdade no milênio passado, mas a realidade vem se mostrando ser bem diferente.

De acordo com uma pesquisa da SuperData, o mercado de jogos cresceu 8% de 2014 a 2015, com 61 bilhões de dólares circulando nesta indústria (CNBC, 2016). Em 2014, o valor da indústria de jogos já havia ultrapassado o da indústria de música em 20 bilhões, e está chegando ao da indústria do filmes (WINGFIELD, 2014).

Então é fato, a indústria de jogos está movendo bilhões de dólares pelo mundo, já passou da indústria de música e não para de crescer. Como diz o gerente de produto da Eletronic Sports League, James Lampkin, no artigo da New York Times: "Isto está se expandindo fora de controle" (WINGFIELD, 2014, tradução nossa). Tais palavras explicam muito bem o estado atual do mercado de jogos.

E não é só nas vendas de jogos e consoles, existem muitos torneios de jogos ocorrendo pelo mundo, surgindo uma nova categoria de profissionais que, em poucos anos atrás era inimaginável, senão motivo de piada, que é a categoria de jogador profissional de jogo eletrônico. A área de trabalho já existe e é chamada de Esporte Eletrônico (WINGFIELD, 2014).

Mesmo nestes torneios, não é por pouco dinheiro que os jogadores se enfrentam. No Campeonato Mundial de 2015 (o quinto da série) de *League of Legends*, foi oferecido 1 milhão de dólares para a equipe vencedora mundial do jogo, como pode ser visto nas regras do campeonato¹.

Os prêmios não param por aí. O jogo Dota 2 distribuiu 11 milhões de dólares para os 10 melhores do mundo, sendo 5 milhões para os campeões, se tornando assim o maior prêmio já oferecido em um torneio de jogo eletrônico (WINGFIELD, 2014).

Não apenas pelos prêmios, mas também tem muita gente para assistir a estes campeonatos. Nos dados mostrados pela Riot² sobre o Campeonato Mundial de 2015, houveram 334 milhões de telespectadores "únicos" durante as 4 semanas do torneio, somando 360 milhões de horas de visualizações das partidas ao vivo.

No entanto, a área de jogos não está chamando apenas a atenção do mercado, mas também a de pesquisadores. Um exemplo é o desenvolvimento e aplicação de técnicas de Inteligência Artificial (IA) em jogos, que de acordo com especialistas, existem áreas dentro de IA em jogos que ainda estão inexplorados (YANNAKAKIS; TOGELIUS, 2014).

¹ Regras: <https://riot-web-static.s3.amazonaws.com/lolesports/Rule%20Sets/2015%20Revised%20World%20Championship%20Rule%20Set%20Version%201_01.pdf>

² Dados disponíveis em: <http://www.lolesports.com/en_US/articles/worlds-2015-viewership>

Em 2007 foi montada, pela AiGameDev, uma lista dos 10 jogos com Inteligência Artificial mais influentes. Um exemplo é um jogo chamado *Creatures*, que implementou aprendizado de máquina em uma simulação interativa ao usar Redes Neurais nas criaturas do jogo. Outro exemplo é o *Halo*, o jogo que implementou pela primeira vez a "árvore de condutas", tecnologia que ficou muito popular na indústria de jogos (AIGAMEDEV, 2007).

Além dos dois jogos citados acima, a IA de F.E.A.R. é descrita em detalhes no artigo da empresa Monolith Productions (ORKIN, 2006). F.E.A.R. é um *First-Person Shooter* (FPS) que criou um sistema dinâmico, coordenado, interessante e desafiador. Isto foi feito utilizando o sistema *Goal Oriented Action Planning* (Planejamento de Ações Orientado a Objetivo), que foi construído junto de duas técnicas: Algoritmo A* e Máquina de Estados Finitos. Os *Non Player Characters* (NPCs) possuem uma lista de objetivos, então durante o jogo eles buscam o plano que irá completar o objetivo com maior prioridade. O planejamento feito é similar ao STRIPS, tendo-se a situação atual e quais são as ações necessárias para cumprir o objetivo. Além disto, foi implementado uma extensão da conduta individual dos NPCs com uma conduta em equipe. No entanto, como a conduta dos NPCs foi criada para minimizar a ameaça a si mesmo, os instintos básicos do NPC podem sobrescrever a conduta de equipe caso a segunda opção seja muito arriscada para si mesmo (ORKIN, 2006).

Percebe-se, desta forma, que a área de jogos está muito ativa e em pleno crescimento, tanto no mercado quando em pesquisas. Existem áreas inexploradas de IA em jogos, com novas fronteiras a serem exploradas. Com isto em mente este trabalho foi desenvolvido, e espera-se contribuir com a comunidade acadêmica e/ou mercado de alguma forma.

1.1 Objetivos do Trabalho

1.1.1 Objetivo Geral

Produzir um *Role Playing Game* (RPG) em turnos que implementa conceitos avançados de Inteligência Artificial, sendo esta inteligência capaz de tomar decisões de forma autônoma sobre o que deve fazer, assim como ser capaz de aprender a tomar melhores decisões por meio de treinamento.

1.1.2 Objetivos Específicos

- a. Criar um jogo RPG com muitas variáveis envolvendo os personagens.
- b. Criar uma aplicação de Inteligência Artificial capaz de simular o desempenho de uma pessoa.

- c. Criar uma aplicação de Inteligência Artificial capaz de aprender.

1.2 Organização da Monografia

Este trabalho está dividido em 5 seções, sendo esta seção (Introdução) a primeira. As outras seções são:

- Seção 2, **Ferramentas Utilizadas**: apresentação das ferramentas utilizadas para o desenvolvimento do projeto proposto no trabalho.
- Seção 3, **Conceitos**: explicação das teorias de Inteligência Artificial e Algoritmo Genético usadas para desenvolver o trabalho.
- Seção 4, **O Jogo**: descrição em detalhes do jogo, suas variáveis, e de sua implementação.
- Seção 5, **Resultados**: apresentação dos resultados obtidos no trabalho.

2 FERRAMENTAS UTILIZADAS

2.1 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, 2016a).

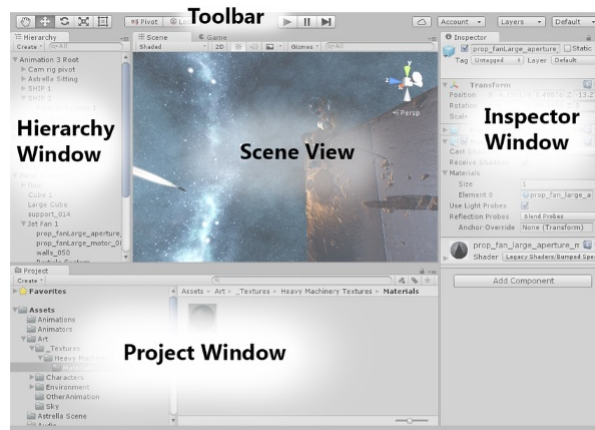
Com mais de 200 jogos na lista de jogos em destaque que foram criados na Unity, um dos exemplos que pode-se citar é o *Sky Force*, um jogo no estilo de ação, missões e *shooter*, e que está disponível na App Store e Google Play. Outro exemplo é o *The Uncertain*, do qual o jogador é um robô e deve resolver quebra-cabeças em uma aventura. O jogo está disponível na Steam (UNITY, 2016b).

O editor da Unity também é extensível, possibilitando implementar funcionalidades ainda não existentes (UNITY, 2016a). 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 (RIVALTHEORY, 2015).

2.1.1 Interface

A interface da Unity é composta por vários painéis chamados *views*, que podem ser rearranjados, agrupados, separados e fixados (UNITY, 2016d). O arranjo padrão das janelas permite o acesso às janelas mais comuns, e pode ser visto na Figura 1 a seguir:

Figura 1 – A interface padrão da Unity



Fonte: Unity (2016d)

A descrição de cada painel pode ser vista na Tabela 1 a seguir:

Tabela 1 – Descrição dos painéis na Unity

<i>Project Window</i>	Exibe a biblioteca de <i>Assets</i> que estão disponíveis para uso no projeto. Ao importar os <i>Assets</i> no projeto, eles aparecerão aqui.
<i>Scene View</i>	Permite navegar visualmente e editar a cena. A <i>Scene View</i> pode mostrar em perspectiva 3D ou 2D, dependendo do tipo de projeto que está sendo trabalhando.
<i>Hierarchy Window</i>	É uma representação de texto hierárquica de cada objeto na cena. Cada item na cena tem uma entrada na hierarquia, de forma que as duas janelas estão inerentemente conectadas. A hierarquia revela a estrutura da forma como os objetos estão ligados um ao outro.
<i>Inspector Window</i>	Permite visualizar e editar todas as propriedades do objeto selecionado. Como diferentes tipos de objetos têm diferentes conjuntos de propriedades, o layout e conteúdo da janela do <i>Inspector</i> pode variar.
<i>Toolbar</i>	Fornece acesso aos recursos de trabalho mais essenciais. À esquerda estão as ferramentas básicas para manipular a <i>Scene View</i> e os objetos dentro dela. No centro estão os controles de <i>play</i> , <i>pause</i> e <i>step</i> . Os botões à direita dará acesso aos Serviços da Unity Cloud e da Conta Unity, seguido pelo menu de visibilidade dos <i>layers</i> e, finalmente, o menu de <i>layout</i> do editor (que fornece alguns <i>layouts</i> alternativos para as janelas do editor, e permite salvar um <i>layout</i> customizado).

Fonte: Unity (2016d)

Os objetos contidos na cena do projeto são chamados de *GameObject*, sendo que suas características podem ser alteradas por *Components* anexados a ele. A Unity possui vários *Components* prontos, no entanto é possível criar novos usando as linguagens de programação do qual se dá suporte (C# e UnityScript). Tais *Components* são chamados de *Scripts*, e eles permitem disparar eventos, modificar as propriedades de outros *Components* ou do próprio *GameObject* durante o jogo, e a interação com o jogador. (UNITY, 2016c).

2.2 Visual Studio

O Visual Studio é um Ambiente de Desenvolvimento Integrado (IDE) usado para a criação de aplicativos para Windows, Android, iOS, aplicações Web e serviços de nuvem. Com ele é possível programar em C#, Visual Basic, F#, C++, HTML, JavaScript e Python, dentre outras linguagens de programação. (MICROSOFT, 2016)

A integração do Visual Studio com a Unity, usando C#, ocorre com a adição do *namespace "UnityEngine"* ao *script*, liberando a utilização da classe *MonoBehaviour*, da qual possui implementado funcionalidades e funções internas da Unity. Para utilizar a classe *MonoBehaviour* nos *scripts* criados, basta estender a classe criada com a

MonoBehaviour (UNITY, 2016c). Desta forma, o cabeçalho do *script* fica como no código a seguir:

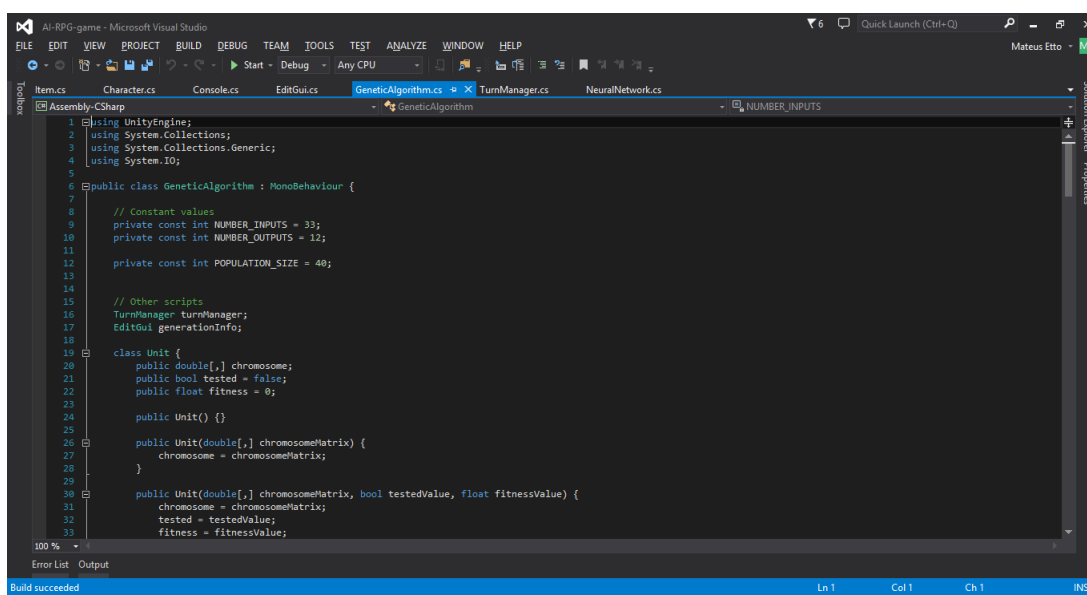
```
1 using UnityEngine ;
2
3 public class NomeDaClasse : MonoBehaviour
4 {
5     // Código da classe
6 }
```

Com isto feito, é possível utilizar uma das principais vantagens do Visual Studio, que é o *AutoComplete*. Ou seja, funções internas da Unity, como as do *GameObject*, aparecem em uma lista conforme se vai digitando, facilitando muito a programação do *script*.

Além da integração com a Unity, ainda é possível utilizar as bibliotecas próprias do C#, como bibliotecas matemáticas, acesso à arquivo e listas.

Na Figura 2 a seguir, é possível ver um exemplo de classe criada no Visual Studio que foi usada neste projeto:

Figura 2 – Exemplo de classe no Visual Studio



Fonte: Elaborado pelo autor.

Nota-se que no *script* de Algoritmo Genético, além do *UnityEngine* e outras duas bibliotecas básicas do C#, também foi utilizada o *System.IO*, uma biblioteca para Leitura e Escrita de Arquivo. Qualquer outro *script* criado para funcionar na Unity tem este padrão.

3 CONCEITOS

Nesta seção será descrito dois conceitos que foram amplamente utilizados neste trabalho, que são Redes Neurais Artificiais (RNA) e Algoritmo Genético (AG). Apesar de ser possível descrever várias variações de aplicação desses conceitos, será explicado apenas a essência deles e o necessário para desenvolver este trabalho.

3.1 Redes Neurais Artificiais

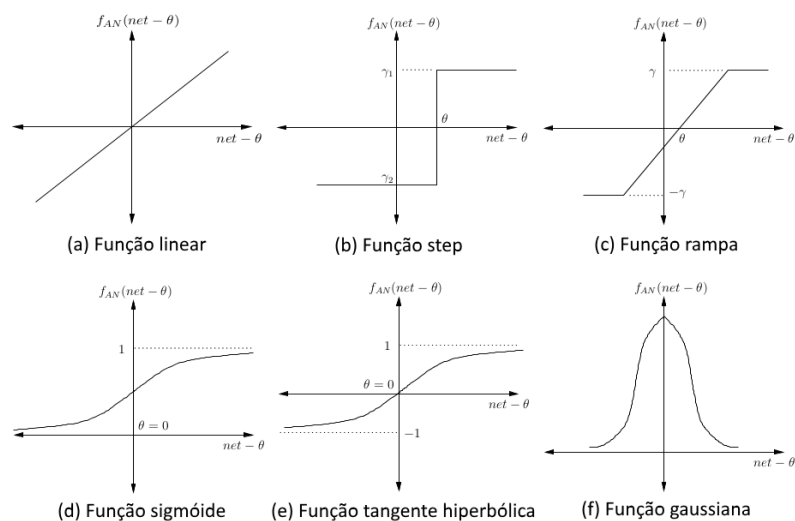
As RNAs são uma família de modelos inspirados no funcionamento do cérebro humano, sendo que a "baixo nível" procuram imitar o que acontece nos neurônios. São usadas para estimar ou aproximar funções que dependam de um grande número de entradas.

3.1.1 Funcionamento

A unidade em uma RNA é o neurônio. O neurônio recebe estímulos, que na RNA são números reais. Recebendo os vários estímulos, o neurônio faz uma operação com todos os números recebidos, que normalmente é um somatório. Esta é a função de entrada.

Após aplicar a função de entrada em todos os estímulos, é obtido um único valor. Este valor, então, é processado por uma função de ativação. Existem várias funções de ativação que podem ser utilizadas, e pode-se ver alguns exemplos na Figura 3.

Figura 3 – Funções de Ativação comuns em uma RNA



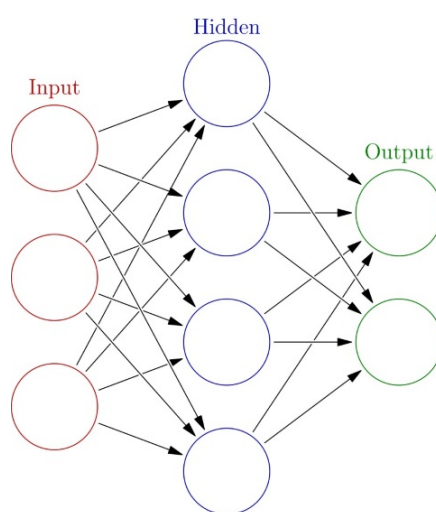
Fonte: Baseado em Reid (2013)

Diferentes funções de ativação se adequam melhor dependendo da aplicação da RNA, assim como a camada em que o neurônio se encontra.

O resultado da função de ativação é então multiplicado por um peso, e passado para o próximo neurônio como um estímulo.

A arquitetura de uma RNA é formada por uma camada de entrada (*input layer*), uma camada de saída (*output layer*), e as camadas ocultas (*hidden layers*) que poderão conter de zero a muitas camadas. Cada camada na RNA terá n neurônios, sendo n qualquer valor maior ou igual a 1. Um exemplo de RNA com uma camada oculta pode ser visto na Figura 4.

Figura 4 – RNA com uma camada oculta



Fonte: Wikimedia (2013)

Como ilustrado na Figura 4, cada neurônio de uma camada se conecta com todos os outros neurônios da camada seguinte. Existem muitas variações de arquiteturas de RNA, não sendo todas que seguem o padrão visto na Figura 4, porém esta é a arquitetura mais "tradicional".

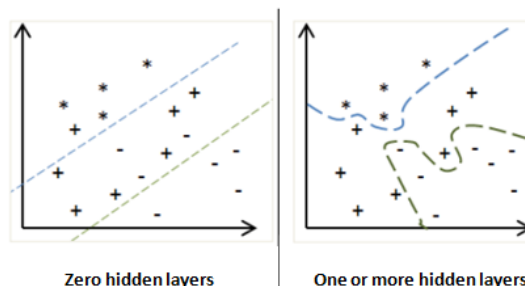
Uma RNA sempre terá uma camada de entrada e uma de saída. No entanto, o número de camadas ocultas pode variar bastante entre um problema e outro. Estas camadas ocultas são explicadas como sendo "extratoras de características" (STACKEXCHANGE, 2013b).

Então, se o número de camadas ocultas varia, de que maneira pode-se determinar o número de camadas ocultas a serem utilizadas em um determinado problema? Caso os dados sejam linearmente separáveis, não é necessário nenhuma camada oculta. Por outro lado, se forem dados não lineares, uma ou mais camadas ocultas serão necessárias (STACKEXCHANGE, 2013a).

Quanto mais camadas ocultas forem utilizadas, maior a granularidade dos resultados obtidos na RNA, como pode ser visto na Figura 5. Para garantir esta não-linearidade

dos resultados, é necessário colocar funções de ativação não-lineares nos neurônios das camadas ocultas.

Figura 5 – Importância da camada oculta



Fonte: StackOverflow (2014b)

Deve-se lembrar que, apesar de muitas camadas ocultas aumentar a granularidade dos resultados, isto vem com um custo: mais tempo de processamento e aprendizagem mais lenta. É por este motivo que deve-se buscar a arquitetura (número de camadas ocultas e número de neurônios) otimizada ao problema, com baixo tempo de processamento e alto desempenho.

Existem muitas discussões a respeito do número otimizado de camadas ocultas a serem usadas, caso ela seja necessária. Um consenso que existe é que uma única camada é o suficiente para a maioria dos problemas (STACKEXCHANGE, 2013a).

Considerando a necessidade de colocar uma camada oculta na RNA, existem regras empíricas que ajudam a escolher um número de neurônios a serem colocados na camada oculta. Uma delas é encontrar a média do número de neurônios de entrada e de saída, e usar este valor como ponto de partida, ajustando-o então com testes (STACKEXCHANGE, 2013a).

3.1.2 Tipos de aprendizado

Um dos maiores atrativos de uma RNA é sua capacidade de aprender, e atualmente existem três paradigmas de aprendizagem: Aprendizado Supervisionado, Aprendizado Não-supervisionado e Aprendizado Reforçado.

No Aprendizado Supervisionado, tem-se os dados de entrada, e infere-se os dados de saída. Sabendo-se estas duas informações, treina-se a RNA para que produza os resultados esperados, ajustando os pesos de suas conexões. Esta forma de aprendizagem é utilizada principalmente para duas finalidades: reconhecimento de padrões e regressão de funções.

Dependendo da finalidade da RNA no Aprendizado Supervisionado, diferentes funções de ativação são recomendadas na camada de saída (as funções de ativação podem ser vistas na Figura 3, página 16). Caso a tarefa da RNA seja de reconhecer

padrões, as funções recomendadas são a *step* (b), sigmoide (d) e tangente hiperbólica (e). Por outro lado, se a tarefa for de regressão de uma função, a função linear (a) é a mais recomendada (RESEARCHGATE, 2013).

No Aprendizado Não-supervisionado, tem-se os dados de entrada e uma função de custo a ser minimizada. Com o decorrer das iterações, os pesos da RNA serão adaptados para minimizar o custo. Tarefas que usam tal paradigma são os que envolvem problemas de estimativa.

No Aprendizado Reforçado, os dados de entrada são desconhecidos, e o cálculo do custo é feita de maneira dinâmica. Os dados de entrada são gerados através das interações do RNA com o ambiente. Exemplos de utilização deste paradigma são programação dinâmica e problemas de controle.

3.2 Algoritmo Genético

Algoritmo Genético (AG) faz parte da computação evolutiva, e inspirou-se na teoria de Darwin sobre a evolução das espécies. É uma busca heurística usada para encontrar soluções de otimização e resolver problemas de busca.

3.2.1 Conceitos de AG

A unidade que caracteriza um determinado indivíduo na população é o cromossomo. O cromossomo é constituído de vários genes, que carregam informações ou características de um indivíduo. O gene pode ser representado de várias formas dentro do AG, sendo a mais simples delas a representação binária, como pode ser visto na Tabela 2.

Tabela 2 – Representação binária de cromossomos

0	1	1	0	1	0	0	0	1	0	1
1	0	1	0	0	1	1	0	0	1	0

Fonte: Elaborado pelo autor.

Outras representações de cromossomos também existem. Por exemplo, existe a codificação por permutação, utilizado em problemas de ordenação. Neste caso, todo cromossomo é um vetor de números, e cada número representa uma posição.

Outro exemplo é codificação por valor. Este valor poderá ser qualquer coisa relacionada ao problema, como números reais ou sequência de caracteres. E como último exemplo existe a codificação em árvore, em que cada cromossomo é uma árvore de objetos, como funções ou comandos em uma linguagem de programação.

Um indivíduo no AG sempre possuirá um cromossomo. Este indivíduo carrega uma das possíveis soluções do problema.

O indivíduo faz parte de uma população. Para se resolver um problema usando AG, nem sempre uma população enorme irá se traduzir em uma convergência mais rápida. De acordo com algumas pesquisas, população em torno de 30 indivíduos se mostram ser eficientes, apesar de que outros valores possam ser melhores dependendo do problema (OBITKO, 1998).

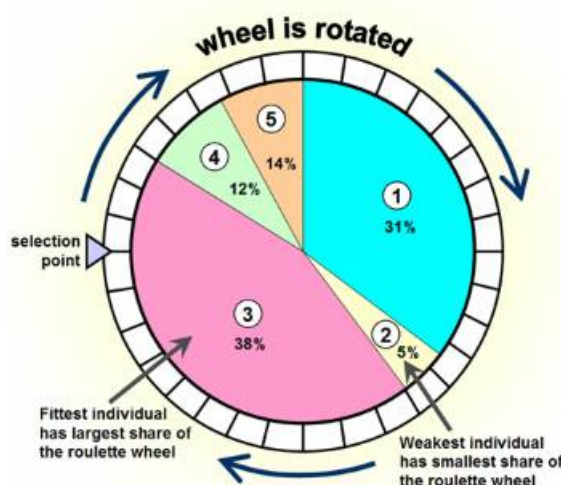
Uma geração diz respeito à iteração do qual a população se encontra. Uma população nova criada a partir de uma população antiga é uma nova geração. Tem-se, então, várias gerações de populações diferentes (ou evoluídas).

Durante o algoritmo do AG, deve-se calcular o *fitness* (nível de aptidão) de cada indivíduo da população. A forma de se calcular o *fitness* do indivíduo depende do problema, e este cálculo faz parte do processo de Avaliação da população, que pode ser visto em mais detalhes na Tabela 3 e Figura 7.

No processo de Seleção, indivíduos da população são selecionados para passarem para a próxima geração ou se reproduzirem. A Seleção usa o valor de *fitness* para dar maiores chances de selecionar os indivíduos mais adaptados.

Um método comum para se selecionar indivíduos é o método da Roleta. Neste método, um intervalo numérico relativo ao valor de *fitness* é atribuído a cada indivíduo. Então é sorteado aleatoriamente um valor no intervalo de todos os indivíduos. O indivíduo que conter o número sorteado dentro de seu intervalo é o escolhido para passar para a próxima geração ou se reproduzir. Percebe-se assim que indivíduos com maior *fitness* terão um intervalo maior e, portanto, maiores chances de serem selecionados. Uma ilustração do que foi descrito pode ser visto na Figura 6.

Figura 6 – Método da roleta para Seleção



Fonte: StackOverflow (2014a)

Outro exemplo de método de Seleção é o de *Ranking*. De forma similar ao da Roleta, é usado o valor de *fitness*. No entanto, o diferencial está no fato de ordenar os

indivíduos em um *ranking*, e atribuir um intervalo numérico dependendo do *ranking* do indivíduo (quanto maior o *ranking*, maior o intervalo atribuído).

Após a seleção existe o *Crossover* (ou cruzamento). Se for decidido que haverá um *crossover*, informações do cromossomo de 2 indivíduos serão misturados, criando "cromossomos filhos". Neste processo, são selecionados pontos nos cromossomos pais, e os cromossomos filhos copiam do primeiro cromossomo pai até aquele ponto, passando então a copiar do segundo cromossomo pai. O *crossover* pode ter um único ponto ou vários pontos de *crossover*. Também existe o *crossover* uniforme, em que é copiado vários pequenos trechos de ambos os pais, com taxa de 50%.

No entanto, o *crossover* não necessariamente ocorre. Existe uma probabilidade dele acontecer, e essa probabilidade é a taxa de *crossover*. Considerando-se, por exemplo, uma taxa de *crossover* de 90%, tem-se que 90% dos indivíduos selecionados passarão pelo processo de *crossover*, e que 10% serão copiados para a nova população.

Após o *crossover*, os cromossomos passam pelo processo de Mutação. A mutação permite a mudança de alguns genes escolhidos aleatoriamente. A chance de ocorrer uma mutação em um determinado gene é determinado pela taxa de mutação. Por exemplo, se a taxa de mutação for de 1%, isto significa que 1% dos genes terão seus valores alterados.

A fim de não se perder boas soluções por causa do *Crossover* e Mutação, uma técnica muito aplicada e eficiente é a do Elitismo. Nesta técnica, o melhor (ou melhores) indivíduos da população são copiados sem nenhuma alteração para a próxima geração.

3.2.2 Processo

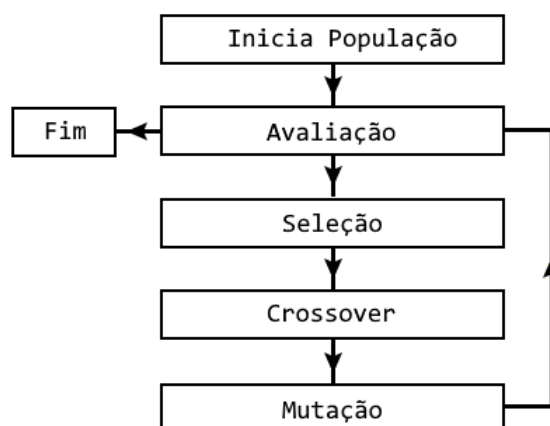
O AG mais simples de ser representado tem 5 etapas, em um processo que se repete até que uma determinada condição de parada seja satisfeita. O processo ocorre como explicado na Tabela 3 e ilustrado na Figura 7 a seguir:

Tabela 3 – Ciclo do Algoritmo Genético

Inicia População	Gera uma população aleatória de n indivíduos.
Avaliação	Calcula o valor de <i>fitness</i> de cada cromossomo na população. Se a condição de parada for satisfeita, o algoritmo acaba.
Seleção	Seleciona 2 cromossomos de acordo com o <i>fitness</i> . Quanto maior o valor de <i>fitness</i> , maior a probabilidade de ser selecionado.
Crossover	Copia partes dos cromossomos dos pais em cromossomos filhos, inserindo-os na nova população.
Mutação	Aplica a probabilidade de alterar aleatoriamente os genes dos indivíduos da nova população, e retorna ao passo de Avaliação.

Fonte: Elaborado pelo autor.

Figura 7 – Passo a passo do AG



Fonte: Baseado em Armstrong (2015)

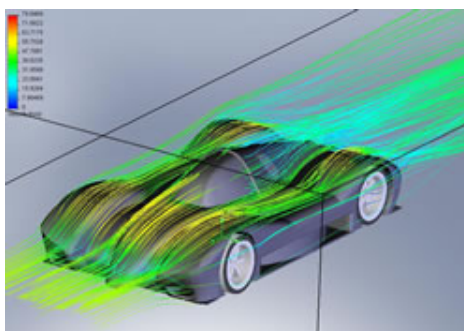
A condição de parada pode ocorrer por mais de um motivo. A melhor condição para se parar é uma boa "avaliação" (*fitness*) dos indivíduos, mas também é possível parar por causa de um limite de gerações ou iterações.

Como é possível notar, a repetição das etapas de Avaliação, Seleção, *Crossover* e Mutação melhora gradativamente as soluções encontradas, sendo esta a maior motivação de se utilizar um AG.

3.2.3 Exemplos de aplicação

Como exemplos de aplicação de AG, será citado Design Automotivo, Robótica, Otimização de rotas e Jogos.

No caso de Design Automotivo, AGs podem auxiliar na escolha de materiais e seus formatos para a produção de veículos mais rápidos, leves, econômicos e mais seguros. A Figura 8 ilustra um exemplo de design de carro. Por se tratar de um problema de otimização, o AG pode encontrar várias soluções que irão ajudar o *designer* a projetar o veículo (BRAINZ, 2010).

Figura 8 – Exemplo de *design* de carro

Fonte: Brainz (2010)

A aplicação de AG em Robótica procura os *designs* de robôs mais otimizados para as

suas tarefas. Como existem muitas "tarefas" que podem ser dadas aos robôs, diferentes *designs* são necessários. Um AG pode agilizar o processo de desenvolvimento de *design* de um robô, considerando as inúmeras variáveis e criando um *design* melhorado (BRAINZ, 2010).

AGs também são capazes de encontrar boas soluções em problemas de otimização de rotas. As rotas podem ser de, por exemplo: um navio no oceano, um carro na estrada ou cidade, e dados trafegando por roteadores na internet (BRAINZ, 2010).

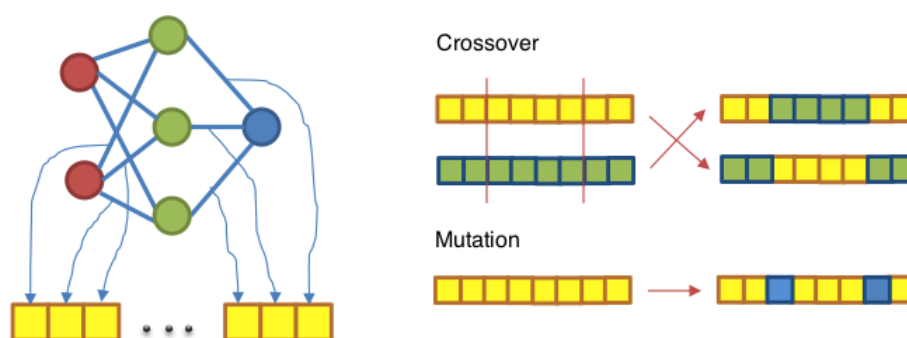
Como último exemplo, é possível utilizar AG em jogos eletrônicos. O jogo The Sims permite que o jogador crie uma família e treine-a para se relacionar uns com os outros (BRAINZ, 2010). O jogo Creatures simula a vida pequenas criaturas, e procura ser fiel ao simular a vida deles como se fossem pequenos animais, criando seu "DNA" e características passadas geneticamente (AIGAMEDEV, 2007).

3.2.4 Variação com RNA

Uma diferente forma de se trabalhar com AG é juntá-la com RNA. Desta maneira, a RNA terá como papel o processamento de dados complexos, e o AG fará a busca dos pesos otimizados na RNA.

Na Figura 9 a seguir, é possível visualizar como é implementação dos conceitos juntos.

Figura 9 – RNA e AG juntos



Fonte: Lahodiuk (2013)

Os pesos da RNA são colocados na estrutura do cromossomo, e então é tratado como um cromossomo para realizar o processo do AG. Ao finalizar, o cromossomo retorna à RNA como pesos, usa-se e repete o processo.

Este foi o método escolhido para a criação da aplicação de Inteligência Artificial neste trabalho: Algoritmo Genético otimizando os pesos de uma Rede Neural Artificial.

4 O JOGO

4.1 Descrição do jogo

O jogo desenvolvido neste projeto é um jogo RPG em turnos, do qual 2 times de 3 personagens cada se enfrentam numa batalha cujo objetivo é derrotar o time inimigo.

A complexidade do jogo se compõe de: uma lista de atributos para cada personagem, diferentes habilidades para dar dano ou curar, e itens consumíveis, tudo junto com uma verificação elemental que pode dar alguma vantagem ao dano ou não. A seguir será descrito cada uma destas partes.

4.1.1 Descrição dos atributos

A Tabela 4 a seguir define quais são os atributos dos personagens, e mostra uma breve descrição de seu papel durante o jogo.

Tabela 4 – Atributos dos personagens

HP (<i>Health Points</i>)	Pontos de vida, é o quanto de dano o personagem pode receber antes de ser incapacitado.
MP (<i>Magic Points</i>)	Pontos de magia, é consumido ao utilizar habilidades.
ATK (<i>Attack</i>)	Um valor alto de ataque permite infligir maiores danos físicos.
DEF (<i>Defense</i>)	Um valor alto de defesa reduz os danos físicos que serão infligidos no HP.
MAG (<i>Magic</i>)	Equivalente ao ATK, mas infligindo danos mágicos.
RES (<i>Resistance</i>)	Equivalente ao DEF, mas reduzindo danos mágicos que serão infligidos no HP.
SPD (<i>Speed</i>)	Velocidade do personagem, e valores altos o permite executar mais comandos em menos tempo.

Fonte: Elaborado pelo autor.

Percebe-se que existe uma diferenciação de 2 tipos básicos de danos: o físico e o mágico. Para ambos os casos existem atributos ofensivos que tendem a aumentar o dano (ATK para dano físico e MAG para dano mágico), e atributos defensivos que tendem a reduzir o dano (DEF para dano físico e RES para dano mágico).

Após um calculo de dano, aquele valor é subtraído no HP do defensor. Caso o HP deste personagem chegue a 0, ele é considerado como incapacitado e não participa mais da batalha.

A única maneira de causar danos mágicos é através de habilidades. Como visto na Tabela 4, utilizá-las consome MP. Caso o MP seja insuficiente para utilizar a habilidade,

ela não poderá ser usada.

Assim como os personagens tem atributos, eles também tem uma lista de pontos fracos e fortes. Esta lista é composta por 5 diferentes danos que é possível receber (sendo 4 delas sub-divisões do dano mágico) e seu valor (alta resistência, normal ou fraco). Mais detalhes sobre tipos de danos podem ser vistos na Tabela 5.

Tabela 5 – Tipos de danos

Ataque físico	Dano físico
Ataque mágico (elemental)	Dano de água
	Dano de fogo
	Dano de terra
	Dano de vento

Fonte: Elaborado pelo autor.

Então se um determinado personagem tiver fraqueza em dano de água e levar este tipo de ataque, o dano a ser causado em seu HP será aumentado em relação ao dano normal. Por outro lado, se este mesmo personagem for resistente à água, o mesmo ataque teria o dano no HP reduzido em relação ao dano normal.

4.1.2 Cálculos de dano e velocidade

O cálculo de danos no jogo é realizado pelas seguintes fórmulas:

$$\text{Dano} = 0,3 \text{ ATK} \times \left(\frac{\text{ATK}}{0,3 \text{ ATK} + 1,2 \text{ DEF}} \right) \quad (4.1)$$

$$\text{Dano} = 0,3 \text{ MAG} \times \left(\frac{\text{MAG}}{0,3 \text{ MAG} + 1,2 \text{ RES}} \right) \quad (4.2)$$

A fórmula (4.1) é usada para calcular danos físicos, e o valor de ATK a ser usado é do atacante, DEF é do defensor, e o dano calculado é o que será reduzido do HP do defensor.

A fórmula (4.2) é usada para calcular danos mágicos, sendo MAG atributo do atacante, RES do defensor, e como a primeira fórmula, o dano é subtraído do HP do defensor.

Após o cálculo do dano, é verificado qual é o tipo de dano a ser atribuído (detalhes dos tipos de danos estão na Tabela 5), e qual a resistência àquele tipo de dano em específico que o defensor tem. Caso o defensor tenha fraqueza neste tipo de ataque, o dano ainda passa pela Fórmula (4.3).

$$\text{Dano final} = 1,5 \times \text{dano} \quad (4.3)$$

Por outro lado, caso o defensor tenha resistência no tipo do dano, a fórmula a ser usada será a Fórmula (4.4)

$$\text{Dano final} = 0,6 \times \text{dano} \quad (4.4)$$

Se o personagem não tem fraqueza nem resistência ao dano, o dano final será o mesmo que o calculado nas Fórmulas (4.1) e (4.2).

Por último, tem-se a influência do SPD na frequência de se efetuar comandos. A partida é composta por vários turnos, e é possível que ninguém ou apenas um personagem efetue alguma ação. Os personagens tem uma variável de "preparado para efetuar comando" que inicia-se em 0 e completa-se ao chegar em 1000 (mas não se limitando a este valor). Em todos os turnos, todos os personagens ganham pontos nessa variável, e a quantidade que ganham é determinada pela Fórmula (4.5) a seguir:

$$\text{Preparado} = \text{Preparado} + \text{SPD} \quad (4.5)$$

Após este cálculo, o jogo ordena todos os personagens que ainda estão batalhando de acordo com esta variável. Se o primeiro da lista tiver 1000 pontos ou mais, um turno é atribuído a ele, a variável dele é reduzida em 1000 pontos, voltando assim ao final da lista.

As fórmulas aqui apresentadas foram usadas na implementação do jogo, mas é importante deixar claro que também foi adicionado aleatoriedade nas fórmulas, a fim de aumentar a complexidade do jogo e deixá-lo menos determinístico. Os danos finais podem variar 15% para mais ou para menos do que foi calculado. E a velocidade, no cálculo do "preparado", SPD tem um acréscimo aleatório entre 0 e 9 unidades.

4.1.3 Comandos

Quando um personagem recebe um turno, ele pode escolher um entre 4 comandos básicos: Atacar, Defender, Usar Habilidade e Usar Item.

Ataque é o comando ofensivo mais simples de todos. Não consome MP, e causa dano físico razoavelmente baixo.

Defesa é um comando defensivo que reduz o dano a ser recebido. Após todos os cálculos de dano, é verificado se o personagem está defendendo. Se estiver, aquele dano é reduzido em 50%.

Habilidade é um comando que possui 3 subcomandos, cada um deles representando uma habilidade diferente:

- **Habilidade Fraca** é uma habilidade que pode causar tanto dano físico quanto mágico, consome 50 MP, e após os cálculos de dano, o valor de dano é aumentado em 50%.
- **Habilidade Forte** é semelhante a fraca, consome 110 MP, e causa um aumento de dano de 100%.
- **Habilidade de Cura** é uma habilidade que permite recuperar o HP de algum personagem. Por ser uma habilidade de suporte, a RES do alvo é ignorada, sendo aplicada uma cura equivalente a 50% do MAG do personagem que está aplicando a cura.

Item é um comando que permite o uso de itens consumíveis durante a batalha. Existem 6 itens diferentes, um é Poção e os outros 5 são ofensivos, um para cada tipo de dano existente no jogo. A poção tem valor constante de cura, e os outros tem um valor de "ataque" semelhante a ATK ou MAG, que é independente do personagem que irá usá-lo.

4.1.4 Atributos dos personagens

A escolha dos valores de atributos, elemento das habilidades, pontos fracos e fortes foram definidos de forma a deixar a batalha estratégica e interessante.

Os personagens de ambos os times foram criados com parâmetros iguais, de forma que se tenha uma batalha de "igual para igual". Todas estas informações foram resumidas e estão ilustradas na Tabela 6 a seguir.

Percebe-se que foram criados 3 tipos diferentes de personagens, cada um com suas características.

O **Tanker** é um personagem resistente a danos, com bastante HP, mas que não é capaz de causar grandes danos.

O **Guerreiro** é um personagem especializado em ataques físicos, possui um HP razoável e é rápido em seus movimentos. No entanto, suas habilidades mágicas não são das melhores.

O **Mago** é um personagem especializado em usar magias, capaz de causar grandes danos em suas habilidades, mas é fraco em ataques físicos e tem baixo HP.

Tabela 6 – Parâmetro dos personagens

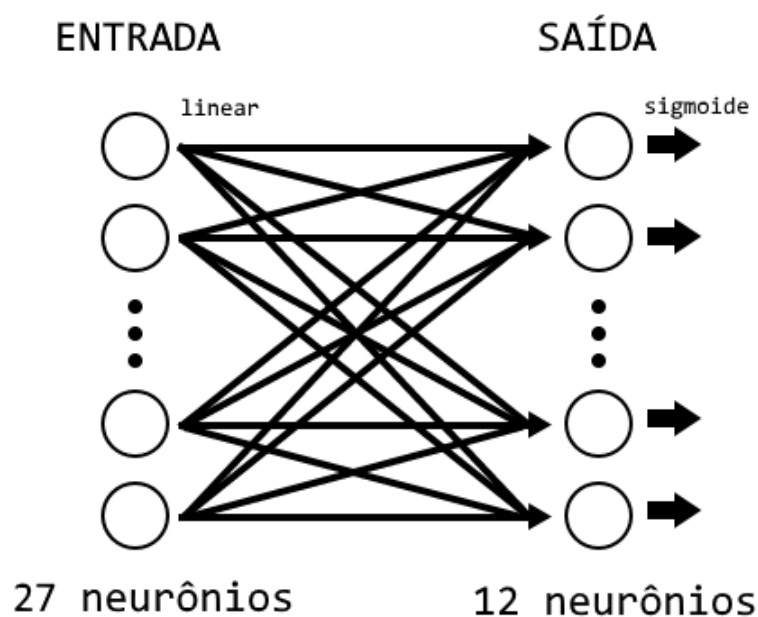
Parâmetro	Pers. 1 (<i>Tanker</i>)	Pers. 2 (<i>Guerreiro</i>)	Pers. 3 (<i>Mago</i>)
HP Max	750	500	400
MP Max	350	550	800
ATK	250	475	200
DEF	500	425	215
MAG	200	250	510
RES	400	210	450
SPD	82	98	79
Defesa Física	Alta	Alta	Normal
Defesa de Água	Normal	Baixa	Alta
Defesa de Fogo	Alta	Normal	Normal
Defesa de Terra	Normal	Normal	Alta
Defesa de Vento	Alta	Normal	Alta
Hab. Fraca	Terra	Física	Água
Hab. Forte	Física	Fogo	Vento

Fonte: Elaborado pelo autor.

4.2 Rede Neural Artificial Implementada

A RNA implementada possui 2 camadas, uma de entrada e outra de saída, e está ilustrada na Figura 10. Não foi colocada nenhuma camada oculta por dois motivos: melhor performance e porque bons resultados foram alcançados mesmo com sua ausência.

Figura 10 – Rede Neural Artificial implementada



Fonte: Elaborado pelo autor.

A camada de entrada possui 27 neurônios, que recebem valores entre -1000 e 1000. Em sua função de ativação, seus valores são multiplicados linearmente por uma constante de 0,002, reduzindo o intervalo dos valores para -2 e 2.

A camada de saída possui 12 neurônios, e na função de entrada é feito um somatório dos estímulos vindos da camada de entrada multiplicados por um peso. Na função de ativação é aplicada a função sigmoide (4.6) produzindo saídas no intervalo de 0 a 1 (ilustrada na Figura 3, gráfico **d**, página 16).

$$S(t) = \frac{1}{1 + e^{-t}} \quad (4.6)$$

A implementação foi feita com o uso de dois vetores e uma matriz, sendo que os vetores representam os neurônios da camada de entrada e saída, e a matriz é a matriz de pesos da RNA. Após o preenchimento do vetor de entrada e a matriz de pesos, são feitos os cálculos explicados acima para obter-se o vetor de saída, que é o resultado.

4.3 Algoritmo Genético Implementado

A população do AG implementado é composta por 40 indivíduos, sendo o cromossomo do indivíduo uma matriz de valores reais. Os genes do cromossomo são inicializados com valores aleatórios entre -1,0 e 1,0.

Na avaliação, dois indivíduos são selecionados e seus cromossomos (que representam a matriz de pesos na RNA) são carregados na Rede Neural Artificial. Após a execução da RNA em cada comando do jogo, é dada uma nota (*fitness*) sobre a influência que a RNA causou, sendo que esta nota pode ser positiva se o resultado foi bom, ou negativa caso o resultado tenha sido indesejável. Mais detalhes sobre os cálculos serão mostrados na próxima seção: 4.4 Funcionamento do jogo.

Após a avaliação de todos os indivíduos da população, inicia-se o processo de seleção. Nesta parte, cria-se a nova população ainda vazia, e o objetivo passa a ser preenchê-la até ficar com 40 indivíduos. Os primeiros 2 indivíduos que entram na nova população são resultado do Elitismo, que copia os 2 com melhor *fitness* da população antiga sem nenhuma modificação.

Para as 38 vagas restantes, repete-se o seguinte processo para preenchê-las: Primeiro, seleciona-se 2 indivíduos usando o método da Roleta. Após isto, é calculado na taxa de *crossover*, de 90%, se haverá *crossover* ou não. Caso haja, é aplicado o *crossover* uniforme para a geração dos cromossomos filhos, que são adicionados na nova população. Caso não haja, os indivíduos selecionados são copiados para a nova população.

Na última etapa, a de mutação, a nova população já está preenchida com 40

indivíduos. Para cada gene dos 38 cromossomos criados na seleção e *crossover*, é calculada uma chance de 1,5% de se ocorrer a mutação. Se for determinado que haverá uma mutação em um gene, haverá uma alteração de 0,3 unidades para mais ou para menos nele, escolhido aleatoriamente.

Finalizada a mutação, a nova população se torna a população atual e retorna-se na Avaliação, repetindo o processo.

4.4 Funcionamento do jogo

Visto a descrição das variáveis do jogo na seção 4.1 e a explicação da implementação da Rede Neural Artificial e Algoritmo Genético nas seções 4.2 e 4.3, será explicado aqui como cada uma destas partes foram juntadas umas nas outras para formarem o todo, que é o trabalho desenvolvido.

4.4.1 Preparação da batalha

Ao abrir o jogo pela primeira vez, o script do Algoritmo Genético cria uma população aleatória e a salva em um arquivo. Desta forma, caso o jogo seja fechado e aberto novamente, as informações da população são carregadas do arquivo. Neste arquivo são salvos o número que representa a geração atual e informações de cada indivíduo da população: os cromossomos, seu *fitness* e se o cromossomo já foi avaliado.

Após a população ter sido criada ou carregada, procura-se os dois primeiros indivíduos que ainda não foram avaliados. Ao encontrar, a informação de seus cromossomos é carregada na matriz de pesos de uma RNA. Nesta parte, cada RNA representará a inteligência de um time, que por sua vez "emula" um jogador real.

4.4.2 Durante a batalha

Ao iniciar a batalha, cada time tem uma RNA que a representa e controla. O personagem que receber o turno é o que poderá efetuar um comando, e neste momento o *script* de RNA do seu time entra em ação 6 vezes, uma para cada diferente personagem ainda em batalha.

Em cada execução da RNA, são usadas como entrada alguns parâmetros do personagem que possui o turno, e parâmetros do possível alvo. As informações usadas como entrada na RNA estão descritas na Tabela 7.

No total são 27 entradas, em que os que foram marcados com um (X), sendo X um número, significa que aquele atributo ainda foi quebrado em X neurônios diferentes. Ou seja, no caso das habilidades, é subdividido em 5 neurônios para que cada um simbolize um elemento.

Tabela 7 – Valores de entrada na RNA

Atacante	Possível alvo		Outros
ATK	HP perdido	Def. Física*	Alvo é inimigo?*
MAG	MP usado	Def. Água*	Alvo está defendendo?*
SPD	ATK	Def. Fogo*	
Elem. Hab. Fraca* (5)	DEF	Def. Terra*	
Elem. Hab. Forte* (5)	MAG	Def. Vento*	
	RES		
	SPD		

Fonte: Elaborado pelo autor.

Valores na Tabela 7 que contém um * (asterisco) representa que eles tem valor binário. Foi implementado que caso o valor seja "falso", "não" ou "fraco", terá o valor de entrada de -1000. Por outro lado, se o valor for "verdadeiro", "sim" ou "forte". terá o valor de entrada de 1000.

Os demais valores vem com intervalo de 0 a 1000, mas que são convertidos para um intervalo de -1000 a 1000, ficando no mesmo padrão que as outras entradas. Como já visto na seção 4.2, o intervalo de -1000 a 1000 é multiplicado por uma constante para se reduzir ao intervalo de -2 a 2.

Após inserir esses dados nos neurônios de entrada da RNA da equipe, é possível fazer os cálculos para obter seus dados de saída, composto por 12 neurônios.

Como já explicado, as saídas tem valores entre 0 e 1. A saída do primeiro neurônio é a que calcula a probabilidade daquele personagem se tornar alvo. Quanto mais próximo de 1, maior a probabilidade, e quanto mais próximo de 0, menor. Os outros 11 neurônios representam, cada um, a probabilidade de utilizar um determinado comando naquele personagem, caso ele vire alvo. O papel de cada neurônio de saída está ilustrada na Tabela 8.

Tabela 8 – Significado dos valores de saída da RNA

Alvo	Atacar	Defender
Prob. de virar alvo	Prob. de atacar	Prob. de defender
Habilidade	Usar item	
Prob. usar a fraca	Prob. item dano físico	Prob. item dano terra
Prob. usar a forte	Prob. item dano água	Prob. item dano vento
Prob. usar a cura	Prob. item dano fogo	Prob. item poção

Fonte: Elaborado pelo autor.

Para utilizar esses dados, foi criada uma estrutura que une "alvo" a "comando" em

uma única probabilidade, se segue a fórmula (4.7) a seguir:

$$\text{Probabilidade} = 1,5 \times \text{Prob. de virar alvo} + \text{Prob. comando} \quad (4.7)$$

Desta forma, cria-se 11 combinações de alvo + comando em um personagem. Repetindo este processo nos N personagens ainda em batalha, obtêm-se $11 \times N$ combinações de alvo e comando no final.

Com todos estes resultados em mãos, é feita uma ordenação decrescente. Neste momento, o jogo tenta executar a melhor combinação de alvo e comando. Caso falhe (por falta de item ou MP), tenta-se executar o próximo comando da lista. Repete-se as tentativas até que consiga executar algum comando com sucesso.

Após o comando ser executado pelo personagem, existe a avaliação de sua escolha. A nota inicial é 0 (zero), e a primeira verificação que se faz é atacou aliado ou curou inimigo. Caso tenha feito isto, recebe um decréscimo de 100 em sua nota. Caso contrário, se tiver atacado um inimigo ou curado um aliado, recebe um acréscimo de 50 pontos na nota.

Logo em seguida, recebe também uma nota referente ao quão bom foi o dano ou cura. Em caso de dano, a nota é a que está representada na fórmula (4.8) a seguir.

$$\text{Nota} = \text{dano}^{1,5} \quad (4.8)$$

Se este dano foi feito num inimigo, soma-se esta nota na que já foi previamente calculada. Por outro lado, se o dano foi em um aliado, subtrai-se na nota que já tem.

Algo similar acontece na cura, que usa a fórmula (4.9) a seguir.

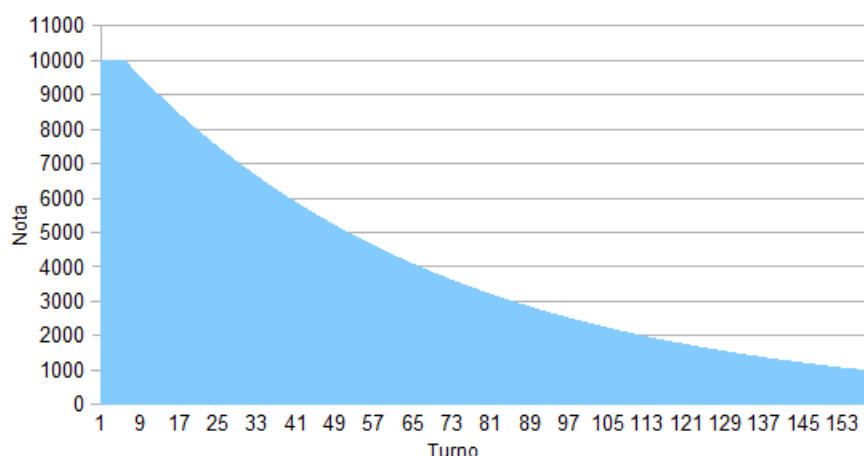
$$\text{Nota} = \text{cura}^{1,4} \quad (4.9)$$

Inversamente ao dano, esta nota é acrescida se for feita em aliado e decrescida se for feita em inimigo.

Como última etapa na escolha da nota do comando verifica-se se o alvo foi derrotado, se sim um bônus na nota será dada. O bônus é inicialmente de 10.000 (dez mil) pontos, e reduz-se em 1,5% de seu valor atual em cada turno a partir do primeiro personagem que efetuar seu segundo comando na batalha. O gráfico dos valores do bônus de derrotar está ilustrado na Figura 11. O principal motivo da existência deste bônus é encorajar a estratégia de derrotar rapidamente os adversários, dado que cada personagem a menos no time inimigo é um a menos para causar danos no time aliado.

Todas estas notas que o personagem consegue e que são somadas são convertidas no *fitness* do time (e da RNA). Uma batalha terá vários comandos executados pelo seu

Figura 11 – Bônus de derrotar inimigo



Fonte: Elaborado pelo autor.

time, e quando a batalha acaba, todas as notas são somadas para obter o *fitness* final da equipe.

Este processo de receber turno, rodar a Rede Neural Artificial, executar o comando e avaliá-la se repete até que uma das duas possíveis condições de parada sejam satisfeitas. A primeira condição é um time completo derrotado. Neste caso, tem-se um vencedor, que é a equipe que ainda tem integrantes participando da batalha. A segunda condição é um limite de turnos, e o motivo dela é a existência de RNAs que enviam comandos de defender repetidamente aos personagens. Este limite turnos é verificado no valor do bônus ao derrotar: caso ele esteja com valor menor que 1.000 (mil), a batalha é encerrada com empate.

4.4.3 Após a batalha

Declarada o final da batalha, ter-se-á o valor de *fitness* de ambas as equipes. Estes valores vão para o *script* do Algoritmo Genético, que registra em sua estrutura o *fitness* de cada um dos dois "indivíduos" e então salva no arquivo.

Para continuar o treinamento da RNA, o *script* do AG procura os dois próximos cromossomos que ainda não foram avaliados e carrega-os na RNA, repetindo todo o processo. Se todos os indivíduos da população já tiverem sido avaliados, o AG passa para a sua próxima etapa e cria a nova geração.

5 RESULTADOS

Para o visual do jogo procurou-se deixá-lo o mais agradável possível, então foram usadas imagens disponibilizadas por usuários do site OpenGameArt¹, e o resultado pode ser visto na Figura 12.

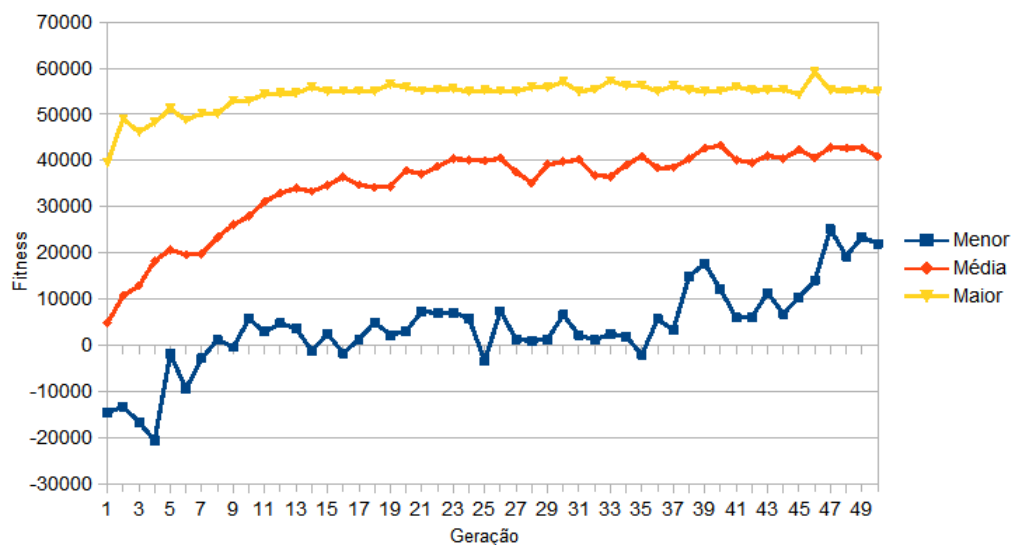
Figura 12 – *Screenshot* do jogo



Fonte: Elaborado pelo autor.

Já a respeito da Inteligência Artificial, um treinamento de 50 gerações foi feito e os resultados estão ilustrados na Figura 13.

Figura 13 – *Fitness* durante 20 gerações



Fonte: Elaborado pelo autor.

¹ Link do site: <<http://opengameart.org/>>

Percebeu-se que os personagens aprenderam a melhorar sua performance com sucesso, e com melhoras significativas nas primeiras gerações. Na primeira geração, a média de *fitness* foi muito próxima a zero, o que mostra que quase metade das equipes tinha desempenho negativo. Este quadro muda completamente já na 10^a geração, do qual a equipe de pior desempenho foi melhor que a média da primeira.

É possível notar também que, coincidentemente, logo na primeira geração já houve uma equipe com alto desempenho. Esta equipe não só se manteve, mas continuou melhorando lentamente.

Por fim, o indicador que melhor mostrou a constante evolução da inteligência no jogo é a média do *fitness*, do qual é possível perceber um constante aumento, com apenas pequenas quedas pontuais.

REFERÊNCIAS

AIGAMEDEV. *Top 10 Most Influential AI Games*. 2007. Disponível em: <<http://aigamedev.com/open/highlights/top-ai-games/>>. Acesso em: 12 Julho 2016.

ARMSTRONG, J. *The Genetic Algorithm - Explained*. 2015. Disponível em: <<https://stats.stackexchange.com/questions/63152/what-does-the-hidden-layer-in-a-neural-network-compute>>. Acesso em: 14 Julho 2016.

BRAINZ. *15 Real-World Uses of Genetic Algorithms*. 2010. Disponível em: <<http://brainz.org/15-real-world-applications-genetic-algorithms/>>. Acesso em: 20 Julho 2016.

CNBC. *Digital gaming sales hit record \$61 billion in 2015: Report*. 2016. Disponível em: <<http://www.cnbc.com/2016/01/26/digital-gaming-sales-hit-record-61-billion-in-2015-report.html>>. Acesso em: 11 Julho 2016.

LAHODIUK, Y. *Evolving of neural networks for collective agents*. 2013. Disponível em: <<https://github.com/lagodiuk/evo-neural-network-agents>>. Acesso em: 14 Julho 2016.

MICROSOFT. *Free Dev Tools - Visual Studio Community 2015*. 2016. Disponível em: <<https://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx>>. Acesso em: 07 Julho 2016.

OBITKO, M. *Introduction to Genetic Algorithms*. 1998. Disponível em: <<http://www.obitko.com/tutorials/genetic-algorithms/index.php>>. Acesso em: 13 Julho 2016.

ORKIN, J. *Three States and a Plan: The A.I. of F.E.A.R.* 2006. Disponível em: <http://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf>. Acesso em: 12 Julho 2016.

REID, S. *10 misconceptions about Neural Networks*. 2013. Disponível em: <<http://www.turingfinance.com/misconceptions-about-neural-networks/>>. Acesso em: 20 Julho 2016.

RESEARCHGATE. *How to select the best transfer function for a neural network model?* 2013. Disponível em: <https://www.researchgate.net/post/How_to_select_the_best_transfer_function_for_a_neural_network_model>. Acesso em: 13 Julho 2016.

RIVALTHEORY. *Features*. 2015. Disponível em: <<http://rivaltheory.com/rain/features/>>. Acesso em: 05 Julho 2016.

STACKEXCHANGE. *How to choose the number of hidden layers and nodes in a feedforward neural network?* 2013. Disponível em: <<https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw>>. Acesso em: 13 Julho 2016.

STACKEXCHANGE. *What does the hidden layer in a neural network compute?* 2013. Disponível em: <<https://stats.stackexchange.com/questions/63152/what-does-the-hidden-layer-in-a-neural-network-compute>>. Acesso em: 13 Julho 2016.

STACKOVERFLOW. *Genetic Programming : Difference between Roulette Rank and Tournament Selection.* 2014. Disponível em: <<https://stackoverflow.com/questions/23183862/genetic-programming-difference-between-roulette-rank-and-tournament-selection>>. Acesso em: 20 Julho 2016.

STACKOVERFLOW. *How do you decide the parameters of a Convolutional Neural Network for image classification?* 2014. Disponível em: <<https://stackoverflow.com/questions/24509921/how-do-you-decide-the-parameters-of-a-convolutional-neural-network-for-image-cla>>. Acesso em: 13 Julho 2016.

UNITY. *Game engine, tools and multiplatform.* 2016. Disponível em: <<https://unity3d.com/pt/unity>>. Acesso em: 01 Julho 2016.

UNITY. *Made with Unity - Games.* 2016. Disponível em: <<https://madewith.unity.com/games>>. Acesso em: 01 Julho 2016.

UNITY. *Made with Unity - Manual: Creating and Using Scripts.* 2016. Disponível em: <<https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html>>. Acesso em: 05 Julho 2016.

UNITY. *Made with Unity - Manual: Learning the Interface.* 2016. Disponível em: <<https://docs.unity3d.com/Manual/LearningtheInterface.html>>. Acesso em: 05 Julho 2016.

WIKIMEDIA. *File:Colored neural network.svg.* 2013. Disponível em: <https://commons.wikimedia.org/wiki/File:Colored_neural_network.svg>. Acesso em: 13 Julho 2016.

WINGFIELD, N. *In E-Sports, Video Gamers Draw Real Crowds and Big Money.* 2014. Disponível em: <http://www.nytimes.com/2014/08/31/technology/esports-explosion-brings-opportunity-riches-for-video-gamers.html?_r=1>. Acesso em: 9 Julho 2016.

YANNAKAKIS, G. N.; TOGELIUS, J. *A Panorama of Artificial and Computational Intelligence in Games.* 2014. Disponível em: <<http://julian.togelius.com/Yannakakis2014Panorama.pdf>>. Acesso em: 11 Julho 2016.