UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

ENRICO FERRARI OSSANAI

Geração Procedural de Mapas em Jogos de Estratégia em Tempo Real

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. Anderson Rocha Tavares

Porto Alegre
2022

“A wizard is never late, nor is he early,
he arrives precisely when he means to.”
— GANDALF THE GRAY

AGRADECIMENTOS

Agradeço à minha família, que me deu os recursos necessários para que toda a experiência universitária acontecesse. Agradeço ao professor Anderson pela orientação, assim como pelo apoio dado durante o tempo de realização deste trabalho. Agradeço à minha namorada, Emanueli, e meu grande amigo, Lucas, por terem percorrido este caminho comigo e me motivarem nos momentos em que pareceu difícil demais. Agradeço à UFRGS e todos os professores com quem cruzei por todo conhecimento que me proporcionaram e, claro, agradeço aos amigos que fiz durante todos esses anos no Instituto de Informática.

RESUMO

A geração procedural de conteúdo (PCG, do inglês Procedural Content Generation) é uma técnica bastante utilizada para a criação algorítmica de diversos conteúdos de jogo. Este trabalho analisa, a partir da revisão de literatura, a viabilidade de PCG em jogos de Estratégia em Tempo Real (RTS, do inglês Real-Time Strategy). Para isso, foram estabelecidas métricas de qualidade obtidas através de replays do jogo RTS StarCraft 2 e a partir de um algoritmo genético, foi possível gerar novos mapas para um jogo RTS.

Palavras-chave: Geração Procedural de Conteúdo. Algoritmo Genético. Real-Time Strategy games.

Procedural Maps Generation in Real-Time Strategy Games

ABSTRACT

Procedural Content Generation (PCG) is a well-known data creation technique. This method uses a genetic algorithm to generate game content. This work analyses, through literature review, the viability of using PCG in Real-Time Strategy (RTS) games. For that, quality metrics were obtained by replaying StarCraft 2 matches. Using a genetic algorithm, it was possible to generate new maps for an RTS game.

Keywords: Procedural generation, Genetic Algorithm, Real-Time Strategy games.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

PCG	*Procedural Content Generation* - Geração Procedural de Conteúdo
RTS	*Real-Time Strategy* - Estratégia em tempo real
CSV	*Comma-separated values* - Valores separados por vírgula
XML	*Extensible Markup Language* - Linguagem de Marcação Extensível
NPC	*Non-Playable Character* - Personagem não jogável

SUMÁRIO

1 INTRODUÇÃO

Quase todo jogo tem o potencial de tornar-se exaustivo, entediante ou, simplesmente, sem graça, com o passar do tempo. Não importando qual seja o nível de afeição pelo jogo, nem mesmo o valor nostálgico, quando insiste-se demais em certas plataformas, elas tendem se tornar menos desafiadoras - até mesmo porque, em alguns casos, pode-se já ter visto tudo o que havia para ver. Pensando nisso, este trabalho busca explorar alternativas criativas para que os jogos não acabem se tornando cansativos e repetitivos sendo, ao contrário, cada vez mais interessantes e menos previsíveis, não importando quantas horas de partida sejam acumuladas pelo jogador.

O tema do presente Trabalho de Conclusão de Curso gira em torno de uma categoria de jogo chamada Estratégia em Tempo Real (RTS, do inglês Real-time Strategy). Esses jogos baseiam-se em uma experiência na qual o jogador deve coletar e gerenciar recursos para construir e controlar múltiplas unidades e estruturas, com a finalidade de derrotar seu oponente. Para que esses jogos mantenham o interesse dos jogadores, é necessário que tenham uma variedade de mapas, recursos e conteúdo em geral. Caso isso não aconteça, o jogador vai visualizar sempre os mesmos cenários e desafios. Uma das soluções propostas para esse problema é a produção procedural de conteúdo (do inglês: Procedural Content Generation).

Figura 1.1 – Imagem do RTS Starcraft: Brood War

Fonte: *StarCraft: Brood War*(1998)

PCG é uma técnica de criação automática de conteúdo por meio de algoritmos (BROWNE, 2011). Com a utilização de PCG, desenvolvedores de jogos podem produzir novos mapas e cenários para o jogador percorrer e, desta forma, não ter uma experiência de jogo repetitiva. Parberry (2014) diz que a geração procedural de conteúdo precisa ser aleatória, mas ainda estruturada, para que crie conteúdo variado e interessante; e, também, precisa ser controlável de maneira natural e intuitiva. Porém, é relativamente complexo definir métricas de qualidade e um objetivo de qual se deseja aproximar, o que pode limitar o uso dessa técnica de geração de conteúdo.

Assim sendo, a utilização de PCG em jogos tem sido experimentada em sistemas notáveis. Um exemplo disso é Ludi, um sistema especializado em criar jogos de tabuleiro inovadores que sejam viáveis e interessantes para humanos (BROWNE, 2011). Entre os jogos digitais, a geração procedural de conteúdo tem sido explorada, com erros e acertos, desde o século passado. No entanto, mesmo que existam estudos sobre a geração de jogos de maneira procedural, os jogos na categoria RTS ainda não foram bastante estudados.

Considerando as limitações da geração procedural de conteúdo, este trabalho pretende analisar a viabilidade da implementação de técnicas de PCG em jogos RTS, por meio da aplicação de um algoritmo genético em jogos da categoria. A principal contribuição deste trabalho é estabelecer métricas de qualidade precisas com o apoio da bibliografia de browne2011evolutionary e, com isso, gerar mapas para jogos RTS. Para isso, o jogo microRTS, versão simplificada de jogos da categoria como Age of Empires e Starcraft 2, será utilizado como base.

O restante desta monografia está estruturado da seguinte forma:

- O Capítulo 2 apresentará os conceitos utilizados no trabalho, tais como: PCG, PCG em jogos digitais, algoritmos genéticos e suas características.
- No Capítulo 3, os trabalhos selecionados para a revisão de literatura serão explorados.
- O Capítulo 4 apresenta a metodologia do trabalho: o levantamento de métricas de qualidade através da análise de replays de um jogo comercial e seu uso em um algoritmo genético para geração de mapas no microRTS.
- O Capítulo 5 expõe e analisa os resultados da implementação feita utilizando o modelo explicado no capítulo prévio.
- Por fim, no Capítulo 6, o capítulo de conclusão, a avaliação do cumprimento dos objetivos será realizada. E serão apresentados possíveis perspectivas de continuidade

desse trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, são explorados conceitos essenciais para a compreensão deste projeto de pesquisa.

2.1 Geração procedural de conteúdo

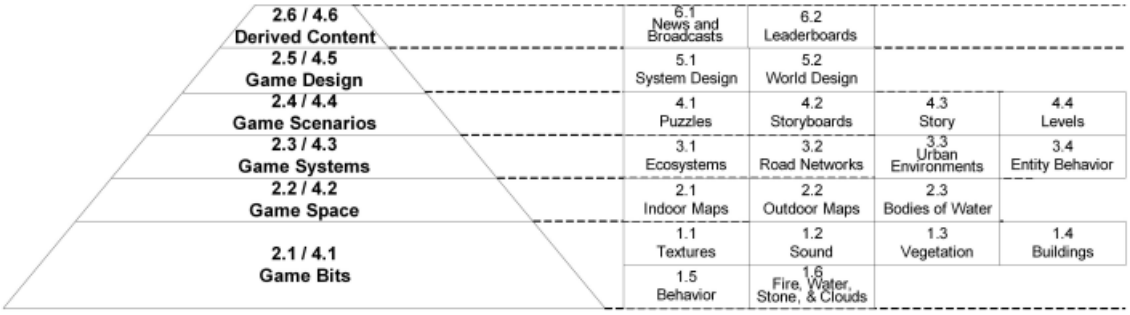
A geração procedural de conteúdo é uma técnica comum para a criação algorítmica de conteúdo de jogo (YANNAKAKIS GEORGIOS N.; TOGELIUS, 2019). Utilizando PCG, é possível automatizar a criação de determinados recursos, como mapas, cenários, sons, gráficos, níveis, mecânicas de jogo, entre outros (DORMANS, 2010).

Hendrikx(2013) propõe uma estrutura de pirâmide dividida em seis classes que podem ser criadas proceduralmente. A base da estrutura é nomeada *Game Bits* e considera elementos como sons, vegetação, comportamentos, texturas, entre outros. A segunda fase é *Game Space*, que conta com mapas internos e externos, parte do que será trabalhado neste projeto de pesquisa. A estrutura pode ser vista na Figura 2.1.

Ainda segundo Hendrikx(2013), a criação de conteúdo se tornou gargalo no processo de desenvolvimento de títulos de alta qualidade. O alto custo de produção de elementos em jogos digitais torna o desenvolvimento um desafio e algumas técnicas de geração procedural são apresentadas como solução para esse problema, devido à alta capacidade de geração de conteúdo (SILVA MARCOS VINíCIUS; MARSON, 2016).

Produzir conteúdo dinâmico de forma procedural também auxilia em um período de produção mais curto (AMATO, 2017). Isso se justifica por PCG se tratar de uma família de técnicas, algoritmos e procedimentos que gera automaticamente um conteúdo que poderia ter que ser feito manualmente (AVERSA, 2015). Além de reduzir o tempo

Figura 2.1 – Estrutura *Game Bits*



Fonte: (HENDRIKX, 2013)

de desenvolvimento do jogo digital, usar métodos procedurais para gerar conteúdo reduz os custos da produção e da mão de obra, ao mesmo tempo em que proporciona uma grande quantidade de elementos com variedades pseudo-infinitas (SILVA MARCOS VINíCIUS; MARSON, 2016).

2.1.1 PCG em jogos digitais

PCG foi inicialmente pensado como uma forma de comprimir dados (AVERSA, 2015). Desta forma ao invés de armazenar o conteúdo do jogo em arquivos, o conteúdo pode ser gerado proceduralmente à medida que o jogador joga.

Figura 2.2 – Elite

Fonte: *Elite*(1984)

Um dos primeiros casos de que se tem notícia é o jogo Elite, de Brabel e Bell, lançado em 1984 (Figura 2.2). O jogo, por meio de PCG, oferece a oportunidade de jogadores explorarem 8 galáxias com mais de 200 sistemas cada. Com isso, Elite oferece gameplays menos previsíveis (ARAÚJO WENDELL OLIVEIRA; DA SILVA ARANHA, 2018), embora haja pontos negativos na utilização de PCG em um jogo, como a geração automática de nomes impróprios para entidades do jogo. Ainda, constatou-se que alguns

sistemas solares estavam mal conectados e acabaram por prender o jogador em determinados lugares. A possibilidade de geração de resultados inconsistentes e caóticos segue sendo um dos desafios do uso de conteúdo gerado proceduralmente.

2.2 Algoritmos genéticos

Algoritmos genéticos (AG) são algoritmos de otimização concebidos por Holland(1975). Em essência, os algoritmos genéticos são mecanismos de busca inspirado nas mecânicas de seleção natural e genética natural (YANG, 2020), apresentadas em 1859 no livro A Origem das Espécies, de Darwin. Segundo essa lógica, os indivíduos mais adaptados sobrevivem e favorecem as próximas gerações com as suas características. Dessa forma, espera que "computadores que 'evoluam' de formas similares à seleção natural possam resolver problemas complexos que nem seus criadores pudessem compreender." (HOLLAND, 1992).

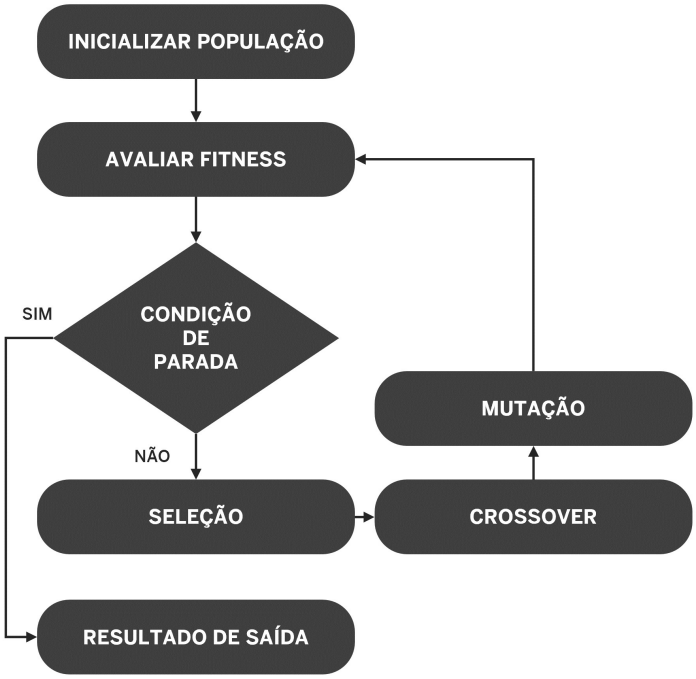
Os algoritmos genéticos apresentam um conjunto de mecanismos de busca adaptiva razoavelmente gerais, mas ainda eficientes (JONG, 1990). Um ponto a se destacar sobre esses algoritmos, ainda segundo Jong(1990), é que eles misturam estratégias como *random search, hill climbing* e *sampling* com a ideia de competição naturalmente, se provando uma boa alternativa aos problemas de busca adaptiva.

Holland(1975) foi provavelmente o primeiro a utilizar as variantes de seleção, crossover e mutação no estudo de sistemas artificiais e adaptativos (YANG, 2020). Mesmo que hoje em dia existam outras variantes para problemas de otimização, as operações de seleção, *crossover* e mutação serão utilizadas neste trabalho e, por isso, são explicadas a seguir.

2.2.1 Seleção

Os indivíduos são selecionados para a reprodução durante o processo de seleção (PACHECO, 1999). Inspirado no processo de seleção natural, o algoritmo genético seleciona os melhores indivíduos da população para gerar variantes, filhos desses indivíduos, por meio de *crossover* e mutação (LACERDA ESTEFANE; DE CARVALHO, 1999). Os indivíduos são selecionados utilizando uma *fitness* (PACHECO, 1999), a qual é exata-

Figura 2.3 – Fluxograma básico de um algoritmo genético



Fonte: O Autor

mente a função que se deseja otimizar no problema, e assim, direcionando o algoritmo genético para as melhores regiões do espaço de busca.

2.2.1.1 Seleção por Roleta

A Seleção por Roleta é um método de seleção para algoritmos genéticos. Neste operador, a chance de um indivíduo ser selecionado para a geração seguinte é diretamente proporcional a *fitness* deste indivíduo. Isso faz com que indivíduos com uma *fitness* baixa também tenham uma chance de serem selecionados para a proxima geração. Pode ser uma vantagem, pois há uma chance que estes indivíduos, que seriam descartados, tenham características que podem ser úteis após um processo de *crossover* ou mutação.

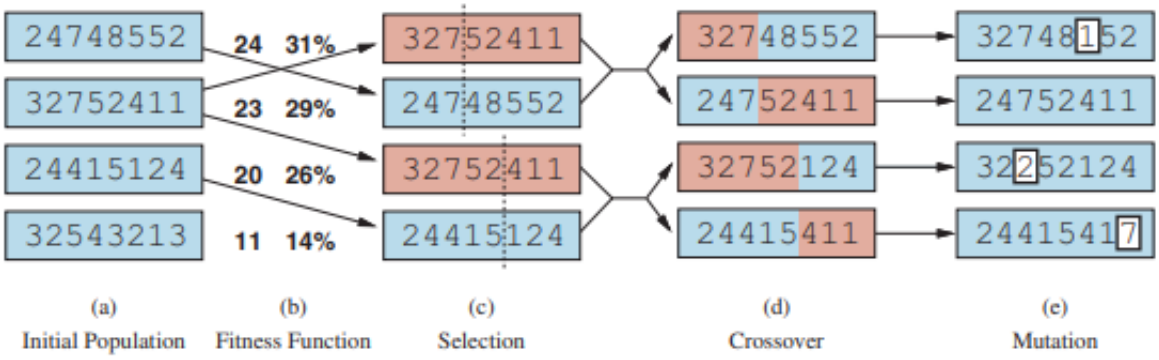
2.2.2 Crossover

Após selecionados, os indivíduos são recombinados atráves do *crossover*. A função do *crossover* é variar os indivíduos de uma geração para a próxima e essa operação é aplicada aos dois cromossomos com dada probabilidade, que é um parâmetro do algo-

ritmo genético. Não ocorrendo o *crossover*, no entanto, os filhos serão iguais aos pais.

O crossover visa combinar características boas de ambos os pais para gerar filhos diferentes de seus antecessores. Isso normalmente é feito de maneira uniforme, onde, para cada gene do filho, sorteia-se qual o pai de origem; ou com o corte em k-pontos: onde inicia-se com os genes de um pai e nos pontos de corte, troca-se o pai de origem dos genes.

Figura 2.4 – Etapas de uma geração de um algoritmo genético



Fonte: (**??**)

2.2.3 Mutação

A mutação, por sua vez, é o operador exploratório com propósito de aumentar a diversidade na população. Ele normalmente causa uma pequena modificação no individuo ao trocar o valor de algum gene aleatoriamente. Esse operador é aplicado em cada um dos filhos resultantes do *crossover*, também com dada probabilidade. Deve-se considerar uma taxa de mutação pequena pois, ao mesmo tempo que gera diversidade, a mutação destrói informação contida no cromossomo.

2.3 Resumo

Neste capítulo, foram abordadas as tecnologias utilizadas nesta pesquisa. Inicialmente, explicou-se o que é a geração procedural de conteúdo e como ela se mostra em jogos digitais. Mais adiante, foi apresentado o conceito de algoritmo genético, com base em (HOLLAND, 1975), e três de suas operações: seleção, *crossover* e mutação.

3 TRABALHOS RELACIONADOS

Geração procedural de conteúdo já é usada há muitos anos. No entanto, por conta da variabilidade de recursos e, consequentemente, dos problemas que podem ser gerados, não há uma proposta de solução genérica para as dificuldades que podem ocorrer durante utilização de PCG em jogos de RTS.

Reunem-se aqui os trabalhos relacionados à aplicação de PCG em jogos que foram considerados durante o desenvolvimento desta pesquisa.

3.1 PCG em jogos de tabuleiro

Em *Evolutionary Game Design* (BROWNE, 2011), o autor conta a história da criação do jogo Yavalath. O jogo foi desenvolvido utilizando o sistema Ludi, que é capaz de jogar, medir e gerar novos jogos de tabuleiro. O processo de geração de novos jogos acontece através de um algoritmo evolutivo, e uma das suas mais importantes tarefas é a habilidade de se avaliar o conteúdo produzido. Isso significa quantificar quão bons são os novos jogos gerados.

Browne (BROWNE, 2011) utiliza cinco critérios de qualidade: Drama (Drama), Incerteza (Uncertainty), Mudança de Liderança (Lead Change), Jogadas Excepcionais (Killer Moves) e Permanência (Permanence). Esses critérios foram utilizados como indicadores de qualidade dos mapas gerados neste trabalho e são explicados no Capítulo 4.

3.2 Geração Procedural de Conteúdo em Jogos de Estratégia em tempo Real

Em *Towards multiobjective procedural map generation* (TOGELIUS; PREUSS; YANNAKAKIS, 2010) é proposto o algoritmo SBPCG (Search-based Procedural Content Generation), um algoritmo de geração de mapas para jogos RTS. Este algoritmo evolutivo permite que dois objetivos sejam alcançados, em vez de apenas um. O presente trabalho gera mapas baseando-se em um jogo RTS imaginário, que apresenta os elementos mais comuns de jogos RTS, como localização das bases inicias e recursos. Os experimentos do SBPCG apresentam em seus resultados mapas jogáveis, porém, o algoritmo utilizado tenta alcançar apenas dois objetivos simultaneamente. O presente trabalho tenta, por sua

vez, alcançar diversos objetivos em vez de apenas dois.

Considerando que a criação de mapas equilibrados para jogos de RTS consiste em um processo complexo, *PSMAGE Balanced map generation for StarCraft* (URIARTE; ONTANON, 2013), criou um sistema chamado PSMAGE (Procedural StarCraft MAp GEnerator). O algoritmo foi desenvolvido como uma solução para gerar mapas para StarCraft.

PSMAGE utiliza diagramas Vonoroi para criar um modelo de mapa inicial, incorporando um conjunto de difrentes métricas de avaliação, que medirão diferentes propriedades de um mapa para definir, no final, se ele é equilibrado ou não.

Para a pesquisa, os autores definiram como "mapa balanceado"aquele no qual jogadores com o mesmo nível de habilidade não tem vantagem um sobre o outro dependendo da posição em que joguem.

Por fim, para garantir a geração procedural de mapas equilibrados para o domínio escolhido, StarCraft, os autores dividiram o processo do algoritmo PSMAGE em seis passos, sendo eles:

1. Geração de região: gera o modelo base do mapa;
2. Determinação de elevação: determina as elevações de todas as regiões.
3. Determinação da posição inicial: atribui regiões como posições iniciais para os jogadores.
4. Adição de localizações base: adiciona recursos a algumas das regiões do mapa, tornando-as possíveis regiões para bases adicionais dos jogadores.
5. Simetria do mapa: gera, com o uso de simetrias, um mapa final para ser equilibrado.
6. Realização: traduz o mapa para um mapa de StarCraft.

Este trabalho, porém, não apresenta a inclusão de regiões não transitáveis.

3.3 Procedural Map Generation for a RTS Game

Nessa pesquisa, utiliza-se também um algoritmo genético para gerar mapas de maneira procedural para um jogo do tipo RTS, Planet Wars (LARA-CABRERA; COTTA; FERNáNDEZ-LEIVA, 2012). Em Planet Wars, os mapas se apresentam em formato de grafos, sem nenhuma dimensão particular, onde há apenas planetas.

Para avaliar os mapas gerados e evoluídos pelo algoritmo genético, utiliza-se uma ferramenta desenvolvida pela *Google* para o *Google AI Challenge 2010*. Em sua conclu-

são, esse trabalho é capaz de gerar mapas jogáveis e balanceados, porém, sua *fitness* não cresce conforme o passar das geraçoes. Além disso, percebe-se uma tendência à geração de mapas amplos e com planetas muito distantes uns dos outros. O que causou essa tendência foi justamente o indicador de *fitness* utilizado, que deu preferência para esses mapas.

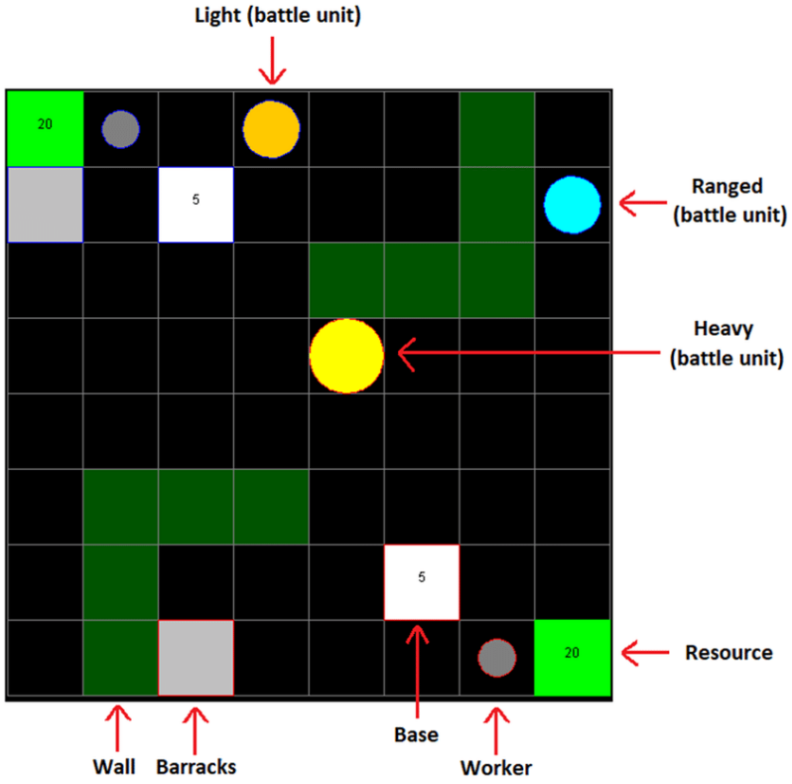
4 PROPOSTA

A proposta desta pesquisa é utilizar o algoritmo genético para realizar a geração procedural de mapas para o jogo microRTS, considerando os indicadores citados abaixo. Como visto anteriormente, o algoritmo genético é um algoritmo de otimização que trabalha em cima de cromossomos. Isso significa que precisa-se ter uma meta a qual se deseja aproximar, e também é necessário codificar nossos mapas de microRTS em cromossomos.

4.1 Ambiente Escolhido

MicroRTS é uma implementação de um jogo de RTS em pequena escala, criada com o objetivo de realizar pesquisas em IA (ONTAñON, 2013). Sua principal vantagem é que, por ser muito mais simples que jogos como StarCraft, pode ser usada para testar diferentes implementações de ideias de maneira mais rápida. Na Figura 4.1, pode-se visualizar um exemplo da interface visual de uma partida. MicroRTS foi definido neste trabalho como o jogo de escolha para a implementação de um mapa.

Figura 4.1 – Mapa explicativo de MicroRTS



Fonte: MicroRTS

4.2 Representação e Codificação do Mapa

Sendo o objetivo desta pesquisa a criação de forma procedural de um mapa para o jogo MicroRTS, é necessário representar o mapa do jogo em um formato de cromossomo com qual um algoritmo genético possa trabalhar.

Foi decidido, então, codificar o mapa do jogo MicroRTS em um *array* de números inteiros para que este possa ser utilizado como um cromossomo de um algoritmo genético. Os mapas apresentam originalmente um formato XML, exemplificado no Apêndice A.

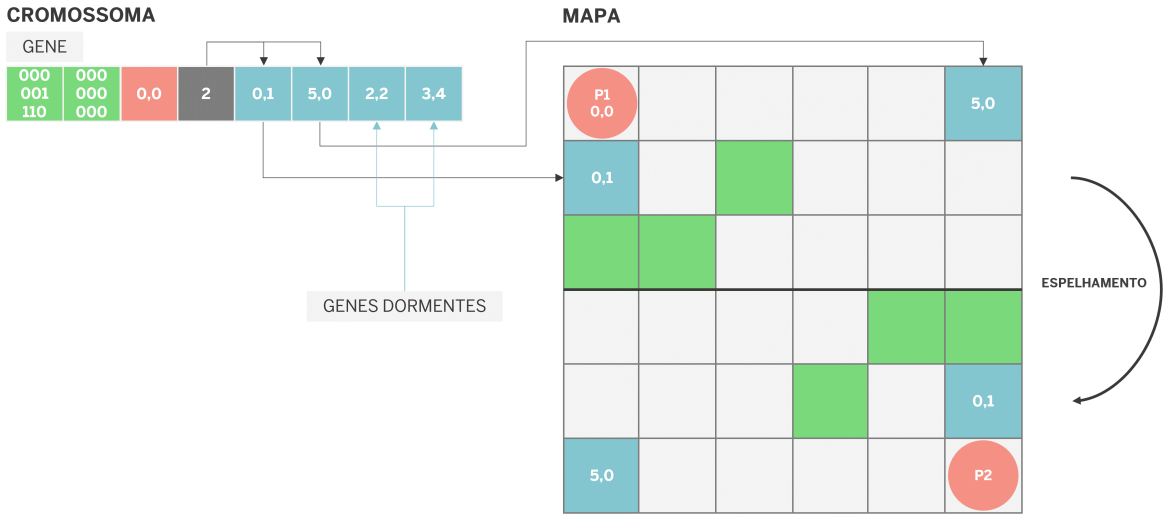
As principais características do mapa são:

1. A largura e altura do mapa.
2. O terreno do mapa, correspondente à um *array* de uns e zeros que representam terreno livre ou bloqueado.
3. Os jogadores que, no caso dos experimentos, serão sempre 2.
4. As unidades iniciais do mapa, que incluem os recursos a serem obtidos, as bases principais dos jogadores e seus trabalhadores iniciais. Todas as unidades precisam ter suas posições iniciais definidas.

Para evitar o problema da possível criação de mapas desequilibrados, os mapas gerados serão gerados de maneira espelhada. O mapa terão um espelhamento duplo, ou seja, tanto no no eixo horizontalcomo no vertical, pra que os jogadores tenham que "percorrer a diagonal. Isso é comum em mapas de dois jogadores em jogos RTS. Para tal, metade do mapa será codificado, e a outra metade será um espelhamento da primeira.

Os primeiros genes são as posições do terreno e suas barreiras, onde ’0’s representam terreno livre e ’1’s representam terreno bloqueado por uma barreira. Os terrenos foram divididos em diferentes zonas quadradas, que variam de tamanho dependendo do tamanho do mapa inicial. Essa escolha foi feita para tentar manter "zonas"características dos mapas passadas nas gerações seguintes. Em mapas de 12 de altura por 12 de largura, existem 8 genes em um cromossomo para representar as zonas do mapa. Decidiu-se manter as dimensões do mapa como constantes. Dessa forma, é mais simples codificar o terreno, que torna-se um *array* sempre com o mesmo comprimento. Na Figura 4.3 é

Figura 4.2 – Decodificação de Cromossomo para Mapa de MicroRTS



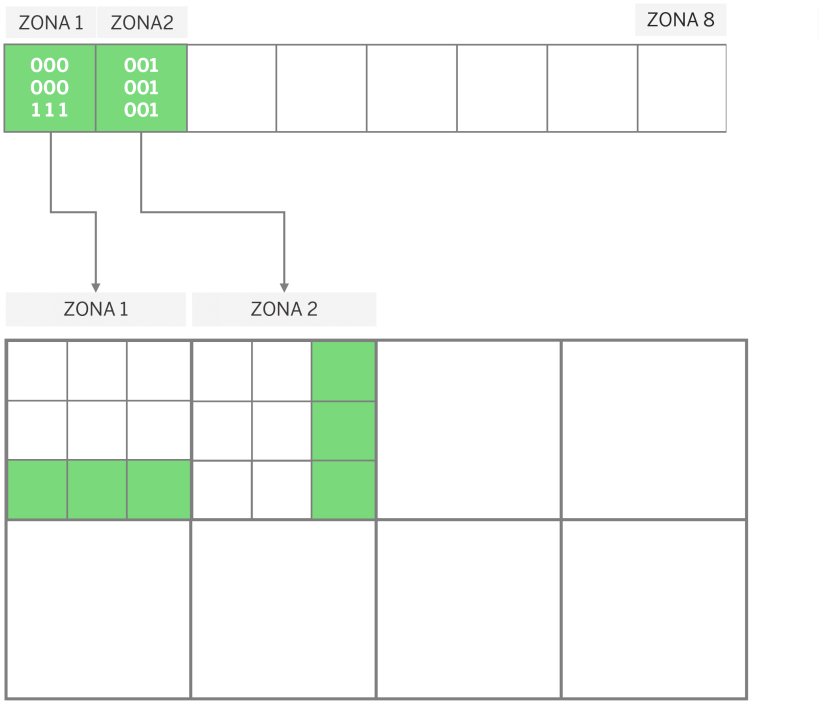
Fonte: O Autor

exemplificada a codificação do terreno.

O seguinte gene representa as posições X e Y da base inicial de um jogador (e assim, por espelhamento, do outro jogador também).O próximo valor representa o número de recursos disponíveis em cada metade do mapa, e os pares de valores subsequentes apresentam a posição dos tais recursos no mapa. Nota-se que, por o cromossomo precisar sempre ter o mesmo tamanho, os cromossomos carregam consigo os genes da posição do número máximo de recursos, ainda que a posição dos recursos não esteja presente em um mapa. Esses, então, chamam-se genes dormentes, que não influenciam na geração do mapa não decodificado, porém podem ter seus valores ativados nas gerações seguintes.

Para remover mapas inválidos, toma-se cuidado para que não sejam criados recursos ou bases em posições onde hajam barreiras no terreno, removendo-se a barreira das posições destes recursos. Os demais mapas inadequados, como aqueles em que os jogadores não conseguem atingir um ao outro, serão descartados na hora de rodar o experimento, pois partidas que não chegam à conclusão em um limite de tempo acabam por ter sua *fitness* zerada. O Apêndice B mostra a função de decodificação do cromossomo.

Figura 4.3 – Representação dos genes do terreno para zonas do mapa



Fonte: O Autor

4.3 Evolução

Para se calcular a *fitness* dos indivíduos de cada geração, é realizada a transformação dos cromossomos em mapas, e cada um deles é jogado uma vez, utilizando como jogadores um dos bots existentes no próprio repositório do MicroRTS, providos pela *microRTS AI Competition*. . Como processo de seleção, utiliza-se a *Roulette Wheel*, onde a chance de um individuo ser selecionado para a próxima geração é proporcional à sua *fitness*. O método de *crossover* é (*Two-Point Crossover*), onde dois pontos aleatórios de indivíduos são escolhidos e o material genético é trocado. O processo de mutação ocorre em 50% dos indivíduos e nele apenas um dos genes sofre mutação.

4.4 Critérios de Avaliação

Para determinar o que é uma partida ideal, foram utilizados os conceitos dos critérios de Browne(2011). Nota-se que apenas os conceitos dos critérios foram utilizados, pois as formulas foram adaptadas para poderem ser utilizadas dentro do contexto de jogos RTS. A seguir, estão apresentados os critérios utilizados como indicadores de qualidade e

suas fórmulas utilizadas:

- *Drama*: A noção de drama carrega a ideia de que deve ser possível se recuperar de uma posição ruim, para que, assim, os jogadores mantenham o interesse no jogo. O critério de drama mede o quanto o jogador que acaba por ganhar a partida fica em desvantagem durante o jogo. O cálculo de drama nessa pesquisa é feito então pela soma da desvantagem em todos os intervalos em que o eventual ganhador está em desvantagem, dividido pelo número de intervalos da partida.
  A seguir esta apresentado o pseudocódigo da Fórmula do drama, onde *N* é o número de intervalos de partida, h(Pn) é heurisitica de vantagem do jogador perdedor da partida no round n e h(Gn) é a heurisitica de vantagem do jogador ganhador da partida no round n.

Algoritmo 1: Cálculo de Drama em uma partida

```
1 S = 0
2 for i=0 in range(0, N) do
3 |   if h(Pi) > h(Gi) then
4 |   |   S += (h(Pi) - h(Gi))
5 |   end
6 end
7 return S/N
```

- *Incerteza (Uncertainity)*: Incerteza é a ideia de que o resultado do jogo deve se manter incerto pelo maior tempo possível, para que todos os jogadores mantenham o interesse no jogo. Para descobrir a incerteza, calcula-se a diferença da vantagem esperada em cada intervalo pela vantagem calculada do ganhador, onde a vantagem esperada é uma crescente linear do ganhador. No algoritmo a seguir ADVn representa a vangam esperada no round n.

Algoritmo 2: Cálculo de Incerteza em uma partida

```
1 S = 0
2 for i=0 in range(0, N) do
3 |   S += (ADVi - h(Gi))
4 end
5 return S/N
```

- *Mudança de Liderança (Lead Change)*: É a tendência em que muda a liderança de quem está ganhando o jogo ao longo da partida. O cálculo deste indicador é feito pela soma do número de vezes em que muda o jogador que está em maior

vantagem na partida, dividido pelo número de intervalos da partida. No algoritmo abaixo, *w(n)* retorna quem está ganhando no intervalo n.

Algoritmo 3: Cálculo de Mudança de Liderança

```
1 S = 0
2 for i=1 in range(1, N) do
3 |   if w(i) != w(i-1) then
4 |   |   S += 1
5 |   end
6 end
7 return S/(N-1)
```

- *Permanência (Permanence)*: Mede o grau de permanência do jogo, ou seja, quão não-variável é o posicionamento dos jogadores. Ela é calculada medindo-se o módulo da diferença da vantagem dos jogadores entre um intervalo e os intervalos anteriore e posterior.

Algoritmo 4: Cálculo de Permanência em uma partida

```
1 S = 0
2 for i=0 in range(1, N-1) do
3 |   S += |(h(Gi) - h(G(i-1)))| + |(h(Gi) - h(G(i+1)))|
4 end
5 return S/(N-2)
```

- *Jogadas Excepcionais (Killer Moves)*: São jogadas que abalam o jogo, mudando drasticamente a posição dos jogadores sobre quem está ganhando. Em jogos RTS, jogadas excepcionais são, na maior parte, combates que definem a partida. No algoritmo abaixo, v(Pn) é o valor dos exercitos do jogador perdedor da partida no round n e v(Gn) é o valor dos exercitos do jogador ganhador da partida no round n. A idéia por trás da fórmula é que, se houve perda de exercitos e a diferença da heuristica de vantagem de um jogador mudou muito (neste caso 0,2), houve então uma jogada excepional. Ele testa antes porém se no intervalo passado já houve uma jogada excepcional, pois como estamos trabalhando com intervalos, duas jogadas excepcionais uma depois da outra provavelmente significa que ocorreu uma jogada excepcional mais prolongada, e não duas.

Algoritmo 5: Cálculo de Jogadas Excepcionais

```
1  S = 0
2  Last = false
3  for i=0 in range(0, N) do
4      if Last = false AND (v(Pi) < v(P(i-1))) OR (v(Gi) < v(G(i-1))) then
5          if |(h(Pi) - h(P(i-1)))| > 0,2 then
6              S += 1
7              Last = true
8          end
9          Last = false
10     end
11     Last = false
12 end
13 return S/(N-1)
```

- *Perda (Loss)*: Esse último indicador não consta na lista de critérios de Browne, porém o autor da pesquisa o considerou importante adiciona-lo. Ele mede a frequência de intervalos em que houve uma perda de recursos maior do que ganho. Em jogos de RTS, este indicador pode ser usado para mostrar a quantidades e a frequência de combates na partida.

Algoritmo 6: Cálculo de Perda em uma partida

```
1  S = 0
2  for i=0 in range(0, N) do
3      if v(Pi) < v(P(i-1))) OR (v(Gi) < v(G(i-1)) then
4          S += 1
5      end
6  end
7  return S/(N-1)
```

Para se descobrir o valor ideal de cada um destes indicadores, optou-se pela extração de valores de partidas de StarCraft 2. Por ser o jogo de RTS de maior sucesso comercial existente, assume-se que seus valores estejam bem ajustados, e se deseja alcançar a criação de mapas que proporcionam partidas tão interessantes quanto as de StarCraft 2.

Algumas destas fórmulas utilizam para seus cálculos uma medida que representa

quão perto está de se ganhar. Porém, em uma partida de jogo RTS, há grande dificuldade de medir quem está ganhando. Essa métrica envolve diversas variáveis, como recurso disponível, estruturas construídas, trabalhadores coletando recursos e valor de exército. Por isso, decidiu-se utilizar uma aproximação um pouco mais direta e mensurável que, apesar de não ser exata, é um bom indicador. Para medir a vantagem, tanto em StarCraft 2, quanto em MicroRTS, serão utilizados apenas os valores absolutos dos exércitos dos dois times. A porcentagem de chance de ganhar a partida se baseia na diferença dos valores dos dois exércitos em determinado período, dividida pela soma dos dois valores.

Para se calcular a *fitness* foi dado uma nota variando de 1 a zero para cada um dos indicadores calculados, onde a nota 1 é recebida quando o valor obtido é igual ao valor desejado, e vai diminuindo gradualmente em 0,1 conforme o valor obtido vai se afastando do desejado. A Tabela D.2 apresenta as notas utilizadas para cada indicador. No final, é realizada então a média das notas dos indicadores calculados para se obter o valor final da *fitness*.

5 AVALIAÇÃO EXPERIMENTAL

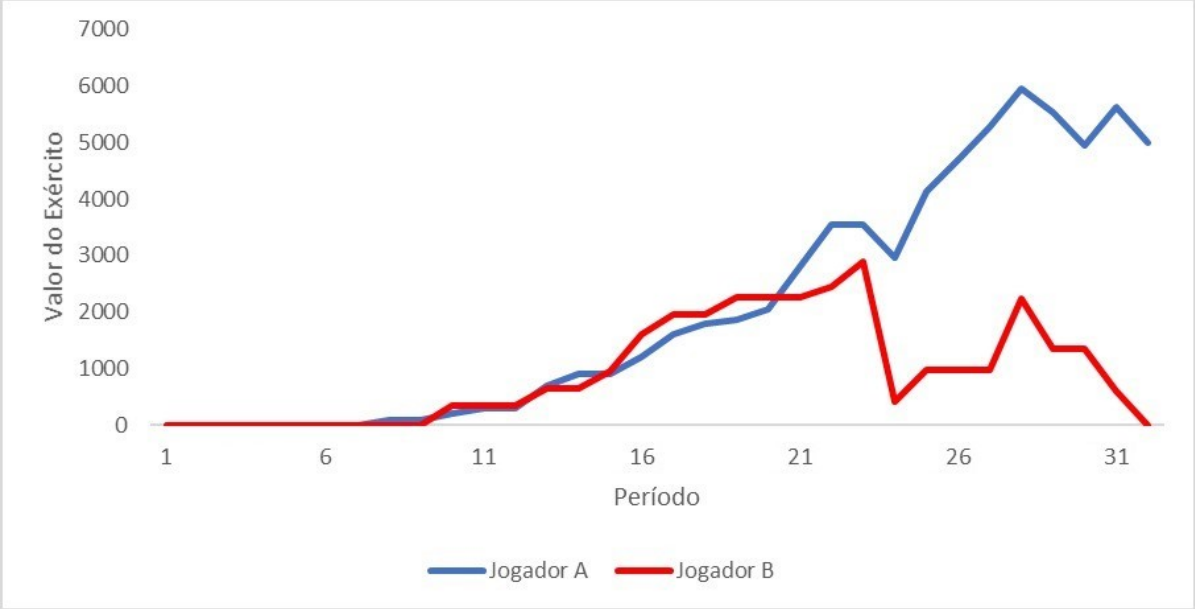
Buscando o cumprimento dos objetivos desta pesquisa, este capítulo apresenta e explica todos os experimentos realizados durante o desenvolvimento deste trabalho. Ao fim, é feita a análise de resultados dos experimentos feitos.

5.1 Busca de Valores a Serem Otimizados

A primeira etapa desta pesquisa foi composta pela coleta de dados, feita através de replays de partidas do jogo StarCraft 2 com o auxílio de *Sc2ReplayStats*, ferramenta on-line capaz de analisar jogos e estatísticas de partidas de StarCraft 2. 50 partidas do jogo foram baixadas e extraídas para um arquivo CSV, elaborado com valores dos dois exércitos, em intervalos de 20 segundos. Na Figura 5.1, pode-se ver o valor absoluto dos exércitos de cada time (A e B) em uma partida de StarCraft 2.

Visto que apenas os números absolutos dos resultados não são o suficiente para adquirir valores das métricas de Drama, Incerteza, Jogadas Excepcionais e Incerteza, houve a necessidade de mais um processo. Neste, houve a necessidade de criar uma heurística que representasse a vantagem de cada jogador na partida utilizando os números absolutos.

Figura 5.1 – Valores absolutos de dados extraídos de partida de StarCraft 2



Fonte: O Autor

Foi escolhida a proposta mais simples de estabelecer a heurística da vantagem *h(a)*, mostrada na equação 5.1. Nesta equação, *E(a)* representa o valor numérico correspondente ao custo do exército do time A, e *E(b)* representa o valor numérico correspondente ao custo do exército do time B.

Se apenas o valor numérico de determinado time dividido pelo valor numérico do exército dos dois times for utilizado como fórmula de heurística, alguns problemas começam a surgir. Um destes problemas é a instabilidade no início das partidas. Até certo momento, o valor de cada exército é igual a zero e, por isso, o primeiro jogador a criar um exército teria uma heurística igual a um, o que deveria acontecer apenas se o jogador ganhou o jogo.

Para solucionar tais problemas foram somados, sempre, o valor *k*, que é o valor da unidade de exército mínimo para os dois times, que para StarCraft 2 é 50 e para MicroRTS é 2. As tabelas resultadas dessas somas se aproximam mais da real vantagem nas partidas, apesar de, ainda assim, serem provenientes de uma proposta bastante simplificada.

$$h(a) = \frac{E(a) + k}{E(a) + E(b) + 2k} \tag{5.1}$$

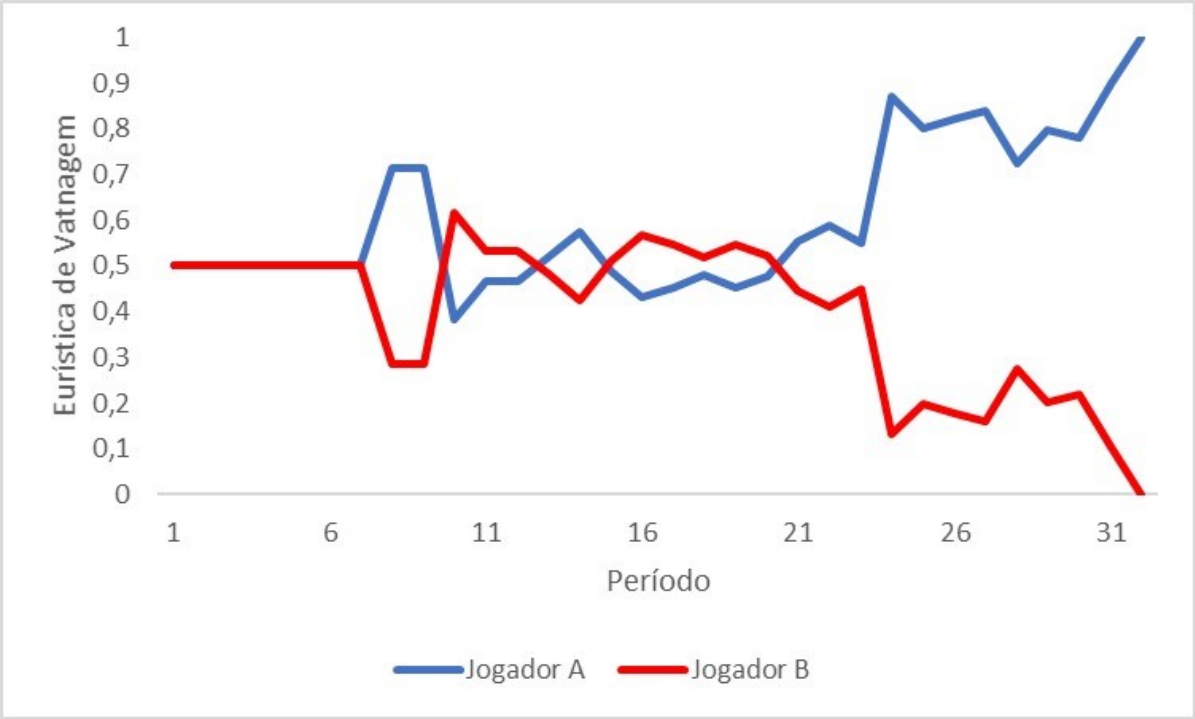
Na Figura 5.2, pode-se visualizar o gráfico da vantagem dos times A e B no decorrer da mesma partida que a Figura 5.1.

Com os dados obtidos, o valor numérico de cada indicador é calculado em cada uma das partidas investigadas, utilizando as fórmulas mostradas no capítulo anterior.

Enquanto a média serve como um ótimo indicador para encontrar um valor único para representar um conjunto de dados, ela nem sempre é o melhor estimador quando existem dados assimétricos e valores extremos. Como critério de análise deste trabalho, a média será utilizada como estimador quando os dados apresentarem bom comportamento da medida assimetria. Como regra geral, se o valor estiver entre -1 e 1, o conjunto pode ser considerado simétrico.

Além disso, o desvio padrão amostral também será utilizado como medida de auxílio para identificar se o conjunto de dados é assimétrico. Como regra de decisão final, caso a medida de assimetria esteja entre -1 e 1 e a proporção do desvio padrão amostral for menor que 40% em relação à média, o conjunto será considerado simétrico e o indicador de tendência central escolhido será a média. Caso contrário, o indicador escolhido será a mediana.

Figura 5.2 – Vantagem de partida anterior



Fonte: O Autor

Com basse nisso, foi definido que, após avaliação dos conjuntos de dados obtidos para os dados de Drama e de Jogadas Excepcionais, a mediana será utilizada como indicador de tendência central. Para os demais dados, a média será utilizada como indicador.

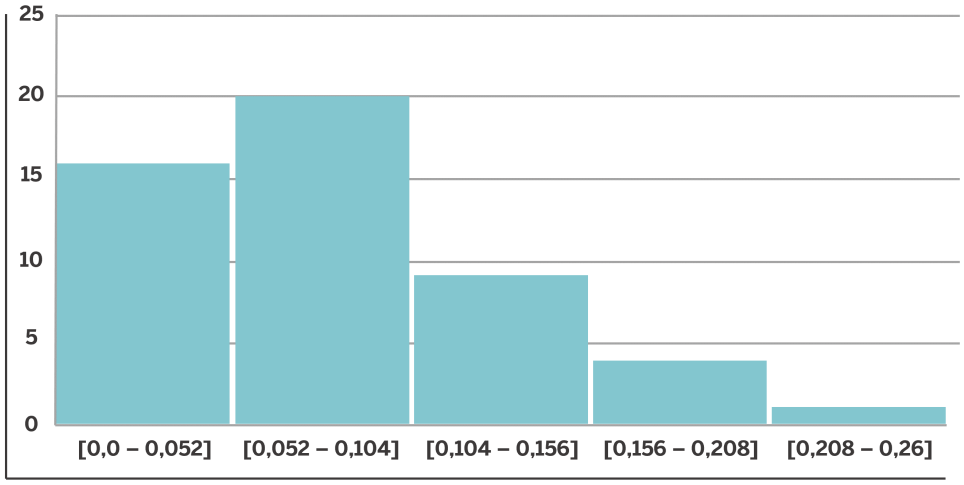
Tendo a Figura 5.3 como exemplo do histograma dos valores obtidos para o critério de Mudança de Liderança, foram considerados como desejáveis os valores apresentados na Tabela 5.2. Estes valores são uma condição que os mapas gerados com o algoritmo genético de otimização proposto por este trabalho precisam atender. A Tabela D.1 apresenta as métricas analisadas.

Tabela 5.1 – Indicadores de qualidade e seus valores ideais

Indicador de qualidade	**Valor Ideal**
Drama	0.037
Incerteza	0.203
Mudança de liderança	0.230
Jogadas excepcionais	0.065
Permanência	0.118
Perda	0.256

Fonte: O Autor

Figura 5.3 – Histograma dos valores de Drama obtidos com o dataset das partidas de StarCraft 2



Fonte: O Autor

5.2 Execução do Algoritmo Genético

Para a execução do algoritmo genético, foram utilizados como populações iniciais uma população de 20 indivíduos sendo eles os mapas providos já no repositório do microRTS. Além destes, por não haverem muitos, foi adicionados mapas desenhados manualmente pelo autor desta pesquisa.

Tabela 5.2 – Parâmetros do Algoritmo Genético

Parametro	**Valor**
Número de indivíduos	20
Número de gerações	25
Política de seleção	*Seleção por Roleta*
Chance de crossover	75%
Tipo de crossover	*Crossover de dois pontos*
Chance de mutação	50%

Fonte: O Autor

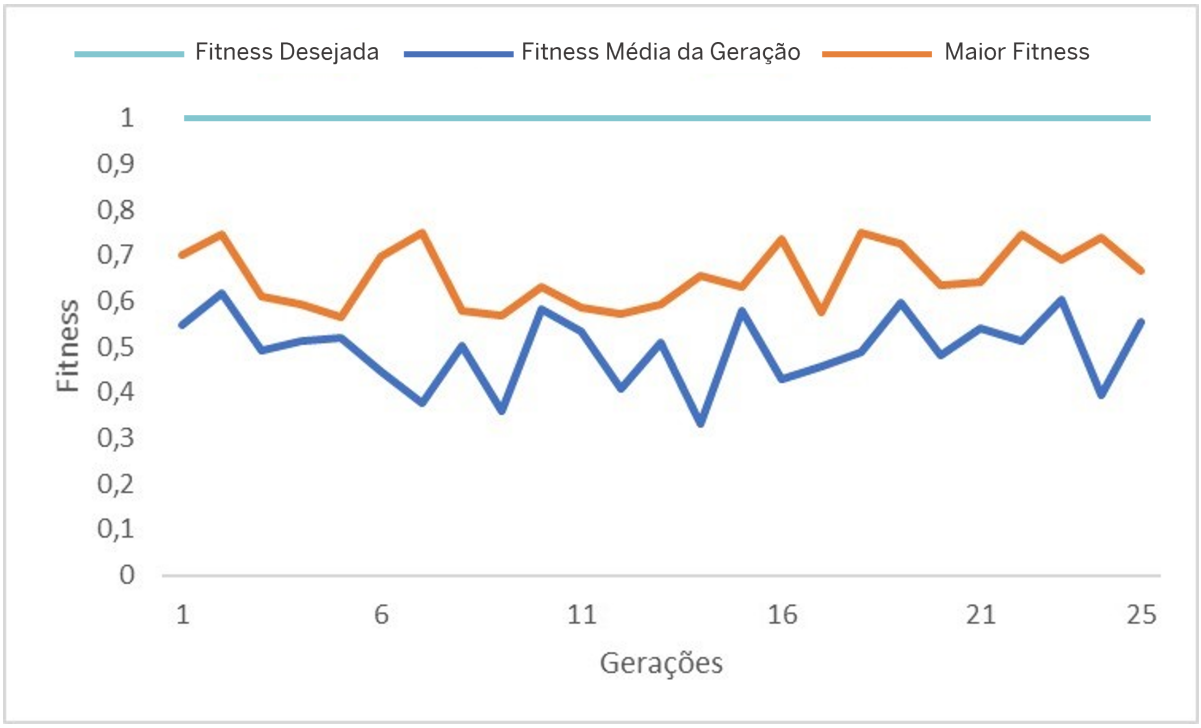
O *playtest* do jogo foi realizado utilizando como jogares o bot CoacAI, ganhador da *microRTS AI Competition 2020*. Foram jogada apenas uma partida por mapa para-se extrair as métricas destes mapas e obter suas *fitness*. Para se desconsiderar partidas em mapas onde os jogadores não conseguem se alcançar, foi usado como limite de tempo da partida um total de 4000 ciclos, que é o mesmo usado na competição de microRTS para mapas 16x16. Partidas que por ventura acabem por durar mais que esse tempo foram

consideradas inadequadas e seus mapas tiveram seus valores de *fitness* zerados.

Inicialmente, todos os indicadores de qualidade foram utilizados. Para avaliar quão bom são cada um destes indicadores, inicialmente foram medidos como estavam os indicadores dos mapas padrão já existentes e os valores resultantes foram comparados aos valores em que se deseja chegar. Ao fim, para obter a *Fitness* ideal do mapa, foi dado um peso igual para cada indicador.

De acordo com o gráfico da Figura 5.4, o algoritmo não converge como esperado, ou seja, não existe uma aproximação da *Fitness* ao valor 1 desejado conforme as gerações vão passando. Além disso, existe uma grande variação de valores entre uma geração e sua geração seguinte, tanto da *Fitness* Média da geração quanto a *Fitness* do maior indivíduo da mesma. Para investigar se o problema está relacionado à algum dos critérios de *Fitness*, o algoritmo genético é executado novamente, agora tendo apenas um dos critérios de *Fitness* de cada vez.

Figura 5.4 – Gráfico de evolução utilizando todos os 6 indicadores para fitness



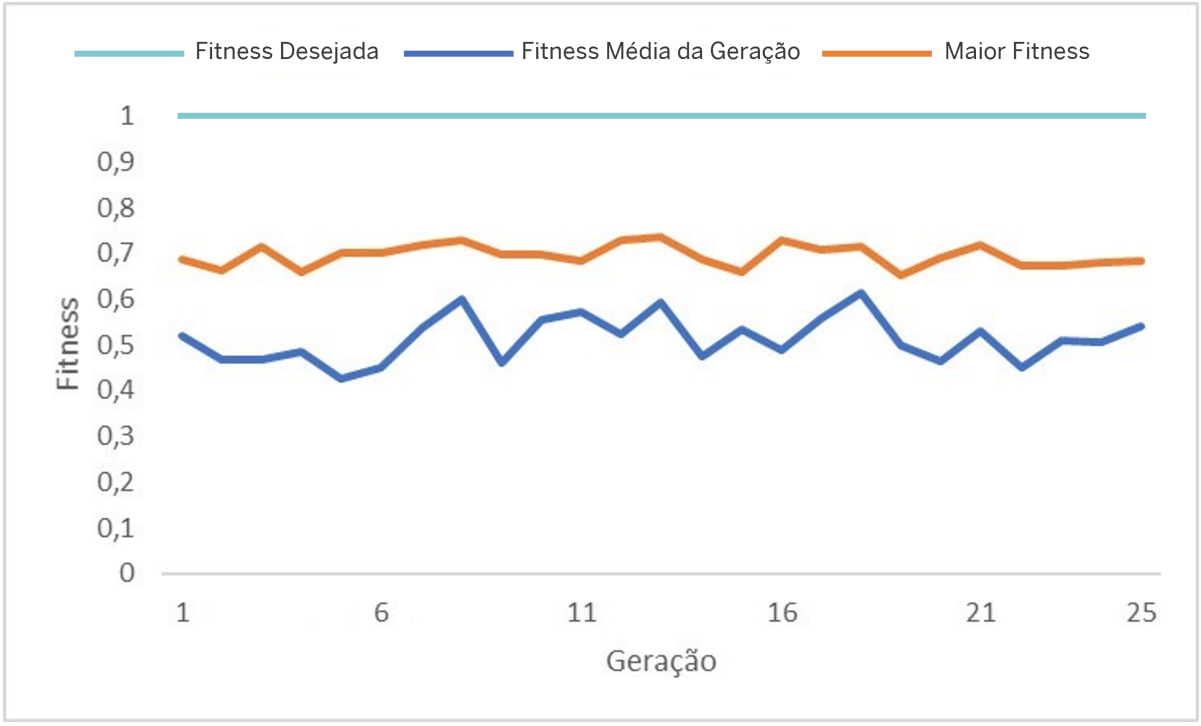
Fonte: O Autor

Então, percebe-se que os indicadores de Incerteza, Jogadas Excepcionais e Permanência têm valores aleatórios conforme as gerações passam. Em nenhum momento os

valores parecem convergir para seus ideais, e os outros três indicadores, apesar de apresentarem uma leve e não muito significativa convergência, ainda apresentam uma enorme variação.

Tendo isso em mente, o experimento foi realizado novamente, tendo como *Fitness* final apenas os indicadores de Drama, Mudança de Liderança e Perda. A Fìgura 5.5 mostra o grafico de sua fitness por gerações. Podemos perceber que ele apresenta uma estabilidade maior do que o da Figura 5.4 mas ainda sim, tanto a *Fitness* média quanto a maior, seguem apresentando variação, sem convergir.

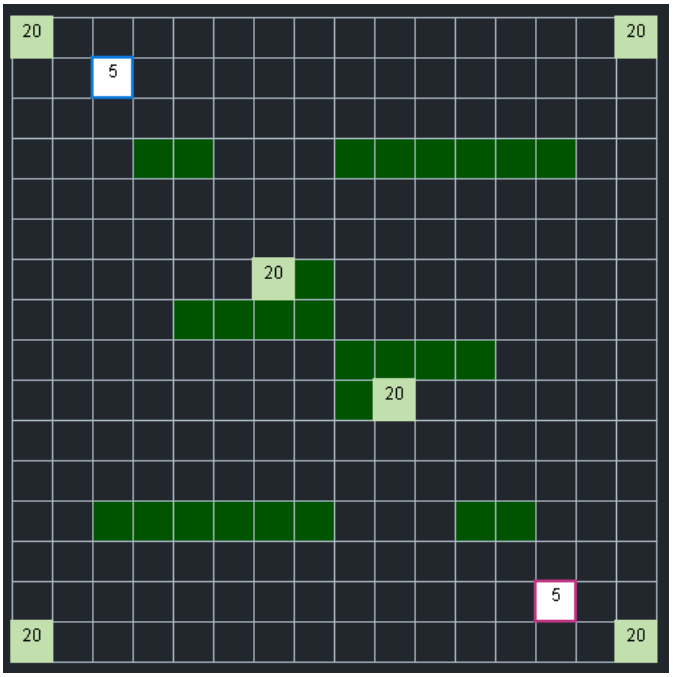
Figura 5.5 – Gráfico de evolução utilizando indicadores de Drama, Mudança de Liderança e Perda como fitness



Fonte: O Autor

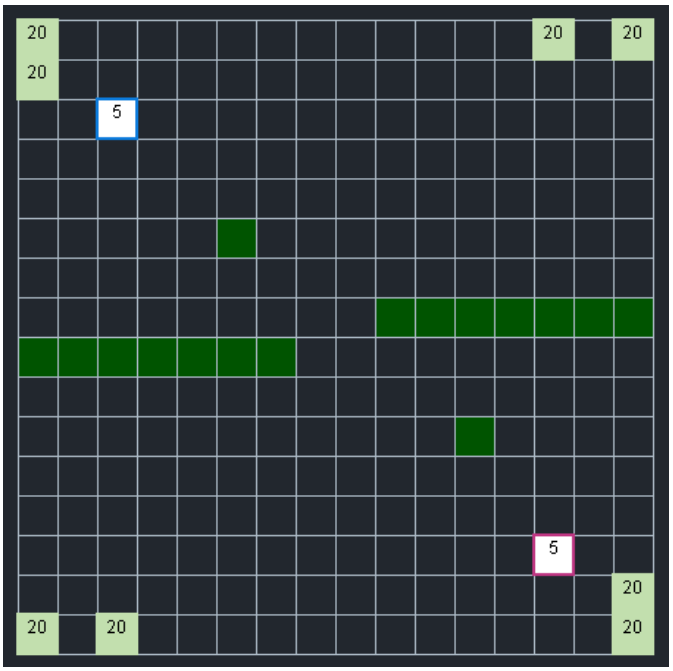
Na execução final, pode-se ver que o algoritmo foi capaz de gerar um mapa jogável e relativamente interessante, como mostrado na Figuras 5.6 e 5.7. Porém, houveram mapas na geração final que podem parecer "inadequados". Apesar disso, isso não é inesperado. Por exemplo, no processo de geração de Yavalath (BROWNE, 2011), a maior parte dos jogos gerados foram não-jogáveis.

Figura 5.6 – Exemplo 1 de Mapa Gerado



Fonte: O Autor

Figura 5.7 – Exemplo 2 de Mapa Gerado



Fonte: O Autor

5.3 Avaliação de Resultados

Mesmo removendo os indicadores que não apresentaram nenhuma convergência, pode-se ver que a melhora dos valores, tanto da "Média"quanto do "Melhor indivíduo de cada geração", não acontece em todas as gerações. Em muitas gerações, os valores pioram. Também percebe-se que a escolha dos bots escolhidos para jogar influenciou os indicadores muito mais do que a diferença entre os mapas. Dois bots diferentes jogando o mesmo mapa apresentam uma variação de critérios muito maior do que o mesmo bot jogando dois mapas diferentes.

Além disso, pode-se observar que mesmo após diversas gerações, nenhum indivíduo se aproxima muito do que foi definido como desejável. Foram destacados pelo autor alguns fatores que podem ter causado isso, sendo eles:

1. A falta de ligação direta entre o mapa gerado e os indicadores, pois indicadores são medidos a partir, não diretamente a partir destes mapas gerados, e sim de partidas jogadas sobre estes mapas.
2. O fato de apenas uma partida ter sido rodada para cada mapa gerado. Partidas com outros jogadores possivelmente resultariam em métricas diferentes.
3. O número de gerações acabou sendo baixo. Isto pela falta de tempo, pois a extração da fitness de cada geração depende dos bots jogarem as partidas, o que leva certo tempo.

6 CONCLUSÃO

Neste trabalho, foi explorada a aplicação do algoritmo genético como técnica de geração procedural de mapas de jogos RTS utilizando critérios de qualidade de Browne(2011). Como conclusão, a geração de mapas jogáveis e, em alguns casos, visualmente interessantes, foi possível.

Por fim, foram identificados alguns pontos que poderiam ser aprimorados e podem ser explorados em trabalhos futuros. São estes:

- Um estudo aprofundado para a confirmação da validade dos indicadores utilizados como critério de qualidade de jogo. Apesar de terem sido assumidos como ideais, pode ser o caso de não serem os mais apropriados.
- A extração da *Fitness* obtida no jogo é coletada de apenas uma única partida jogada no mapa. Idealmente, deveria ser coletada em diferentes partidas com diferentes jogadores.
- Outras maneiras de codificar um mapa do jogo podem ser exploradas e gerar diferentes resultados.
- Com uma Inteligência Artificial que jogue de forma mais semelhante a um ser humano, seria possível reproduzir uma partida real de forma mais verossímil.

REFERÊNCIAS

AMATO, A. Procedural content generation in the game industry. In: ____. [S.l.: s.n.], 2017. p. 15–25. ISBN 978-3-319-53087-1.

ARAÚJO WENDELL OLIVEIRA; DA SILVA ARANHA, E. H. M. C. A. G. D. Geração procedural de conteúdo aplicada a jogos digitais educacionais. **Jornada de Atualização em Informática na Educação**, v. 1, p. 1–20, 2018.

AVERSA, D. History and techniques used in the modern video-game industry. In: . [S.l.: s.n.], 2015.

BROWNE, C. **Evolutionary Game Design**. Springer London, 2011. (SpringerBriefs in Computer Science). ISBN 9781447121794. Disponível em: <https://books.google.com.br/books?id=tn2fE9USzZkC>.

DORMANS, J. Adventures in level design: Generating missions and spaces for action adventure games. 01 2010.

HENDRIKX, M. e. a. Procedural content generation for games: A survey. **ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)**, v. 9, p. 1–22, 2013.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. [S.l.]: The University of Michigan Press, 1975.

HOLLAND, J. H. Genetic algorithms. In: . [S.l.: s.n.], 1992.

JONG, K. D. Genetic-algorithm-based learning. In: . [S.l.: s.n.], 1990.

LACERDA ESTEFANE; DE CARVALHO, A. D. Introdução aos algoritmos genéticos. sistemas inteligentes: aplicações a recursos hídricos e ciências ambientais. In: . [S.l.: s.n.], 1999.

LARA-CABRERA, R.; COTTA, C.; FERNáNDEZ-LEIVA, A. Procedural map generation for a rts game. In: . [S.l.: s.n.], 2012.

ONTAñON, S. The combinatorial multi-armed bandit problem and its application to real-time strategy games. In: **Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment**. [S.l.]: AAAI Press, 2013. (AIIDE'13), p. 58–64. ISBN 1577356071.

PACHECO, M. A. C. e. a. Algoritmos genéticos: princípios e aplicações. In: . [S.l.: s.n.], 1999.

PARBERRY, I. Designer worlds: Procedural generation of infinite terrain from real-world elevation data. **Journal of Computer Graphics Techniques (JCGT)**, v. 3, n. 1, p. 74–85, March 2014. ISSN 2331-7418. Disponível em: <http://jcgt.org/published/0003/01/04/>.

SILVA MARCOS VINíCIUS; MARSON, F. C. V. D. gamebits framework: A procedural content generation framework for digital games. In: . [S.l.: s.n.], 2016.

TOGELIUS, J.; PREUSS, M.; YANNAKAKIS, G. Towards multiobjective procedural map generation. 06 2010.

URIARTE, A.; ONTANON, S. Psmage: Balanced map generation for starcraft. In: . [S.l.: s.n.], 2013. p. 1–8. ISBN 978-1-4673-5308-3.

YANG, X.-S. Nature-inspired optimization algorithms. In: . [S.l.: s.n.], 2020.

YANNAKAKIS GEORGIOS N.; TOGELIUS, J. Artificial intelligence and games. **Genet Program Evolvable Mach**, p. 143–145, 2019.

APÊNDICE A — ARQUIVO XML DO MAPA DE MICRORTS

```
<rts.PhysicalGameState width="12" height="12">
  <terrain>000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000</terrain>
  <players>
    <rts.Player ID="0" resources="5">
    </rts.Player>
    <rts.Player ID="1" resources="5">
    </rts.Player>
  </players>
  <units>
    <rts.units.Unit type="Resource" ID="16" player="-1" x="0" y="0"
        resources="20" hitpoints="1" >
    </rts.units.Unit>
    <rts.units.Unit type="Resource" ID="17" player="-1" x="1" y="0"
        resources="20" hitpoints="1" >
    </rts.units.Unit>
    <rts.units.Unit type="Resource" ID="18" player="-1" x="11" y="
       11" resources="20" hitpoints="1" >
    </rts.units.Unit>
    <rts.units.Unit type="Resource" ID="19" player="-1" x="10" y="
       11" resources="20" hitpoints="1" >
    </rts.units.Unit>
    <rts.units.Unit type="Base" ID="20" player="0" x="1" y="2"
       resources="0" hitpoints="10" >
    </rts.units.Unit>
    <rts.units.Unit type="Base" ID="21" player="1" x="10" y="9"
       resources="0" hitpoints="10" >
    </rts.units.Unit>
    <rts.units.Unit type="Worker" ID="22" player="0" x="1" y="1"
       resources="0" hitpoints="1" >
    </rts.units.Unit>
    <rts.units.Unit type="Worker" ID="23" player="1" x="10" y="10"
       resources="0" hitpoints="1" >
    </rts.units.Unit>
  </units>
</rts.PhysicalGameState>
```

APÊNDICE B — GERAÇÃO DE MAPA

```
1  import xml.etree.cElementTree as ET
2
3  MAPSIZE = 16
4
5  def gerarMapa(cromossomo):
6      """
7      gene[0-7] - zones of map, array of 1 an 0 with lenght = 9 ex:
           "000000111"
8      gene[8] - position of base, array of two numbers ex: 000000111
9      gene[9] - number of resources of map
10     gene[10-14] - position of resources
11     """
12     #exemplo decromossomo =
           ['000000000','000000000','000000000','000000000','
13     # 000000000','000000000','000000000','000000000',
14     # [0,0],5,[1,1],[2,2],[3,3],[4,4],[5,5]]
15
16     #removendo espacos ocupados por entidades
17     espaÃ§osOcupados = []
18     espaÃ§osOcupados.append(cromossomo[8])
19     for i in range(cromossomo[9]):
20         espaÃ§osOcupados.append(cromossomo[10+i])
21     for j in range(len(espaÃ§osOcupados)):
22         posX = espaÃ§osOcupados[j][0]
23         posY = espaÃ§osOcupados[j][1]
24         cromossomo[posY] = cromossomo[posY][:posX] + '0' +
               cromossomo[posY][posX + 1:]
25
26     #gerando raiz
27     root = ET.Element("rts.PhysicalGameState", width=str(MAPSIZE),
           height=str(MAPSIZE))
28
29     #gerando terreno principal
30     terrainValeu = ""
31     for i in range(4):
32         for j in range(8):
33             terrainValeu += cromossomo[j][(i*4):((i*4)+4)]
34     terrainValeu+=terrainValeu[::-1]
35
```

```
36      terrain = ET.SubElement(root, "terrain").text = terrainValeu
37
38      #gerando jogadores
39      players = ET.SubElement(root, "players")
40      ET.SubElement(players, "rts.Player", ID="0", resources="5")
41      ET.SubElement(players, "rts.Player", ID="1", resources="5")
42
43      units = ET.SubElement(root, "units")
44      #gerando bases principais
45      ET.SubElement(units, "rts.units.Unit", type="Base", ID="2",
            player="0",
46                      x=str(cromossomo[8][0]), y=str(cromossomo
                            [8][1]), resources = "5", hitpoints = "10")
47      ET.SubElement(units, "rts.units.Unit", type="Base", ID="3",
            player="1",
48                      x=str((MAPSIZE-1)-cromossomo[8][0]), y=str((
                            MAPSIZE-1)-cromossomo[8][1]), resources = "5
                            ", hitpoints = "10")
49
50      #gerando recursos
51      for i in range(cromossomo[9]):
52          ET.SubElement(units, "rts.units.Unit", type="Resource", ID=
                str(20+(2*i)), player="-1",
53                          x=str(cromossomo[10+i][0]), y=str(
                                cromossomo[10+i][1]), resources = "20",
                                hitpoints = "1")
54          ET.SubElement(units, "rts.units.Unit", type="Resource", ID=
                str(21+(2*i)), player="-1",
55                          x=str((MAPSIZE-1)-cromossomo[10+i][0]), y=
                                str((MAPSIZE-1)-cromossomo[10+i][1]),
                                resources = "20", hitpoints = "1")
56
57      tree = ET.ElementTree(root)
58      tree.write("gamemap.xml")
59      return
```

APÊNDICE C — FUNÇÕES DO ALGORITMO GENÉTICO

```
import random

#seleciona um elemento utilizando a roulette wheel selection
def seleciona(listaCromossomos, listaFitness):
    somafitness = sum([fitness for fitness in listaFitness])
    escolha = random.uniform(0, somafitness)
    valorAtual = 0
    for i in range (len(listaCromossomos)):
        valorAtual += listaFitness[i]
        if valorAtual > escolha:
            return listaCromossomos[i]


def crossover(cromossomoA, cromossomoB):
    tamanho = len(cromossomoA)-1
    #corte no primeiro ponto
    pontoA = random.randint(0,tamanho)
    tempA = cromossomoA[:pontoA]+(cromossomoB[pontoA:])
    tempB = cromossomoB[:pontoA]+(cromossomoA[pontoA:])
    #corte no segundo ponto
    pontoB = random.randint(0,tamanho)
    cromossomoA = tempA[:pontoB]+(tempB[pontoB:])
    cromossomoB = tempB[:pontoB]+(tempA[pontoB:])

    return cromossomoA,cromossomoB

def mutacao(cromossomo):
    #cromossomo escolhido para mutaÃ§Ã£o
    def mutaTerreno(terreno):
        posicaoMutada = random.randint(0,(len(terreno)-1))
        if (terreno[posicaoMutada] == '0'):
            return (terreno[:posicaoMutada] + '1' + terreno[
                posicaoMutada + 1:])
        else:
            return (terreno[:posicaoMutada] + '0' + terreno[
                posicaoMutada + 1:])

    def mutaPosicao(cordenadas):
        randomiza = random.randint(0,7)
```

```
38          if randomiza == 0:
39              return [(cordenadas[0]+1)%16, cordenadas[1]]
40          elif randomiza == 1:
41              return [abs(cordenadas[0]-1), cordenadas[1]]
42          elif randomiza == 2:
43              return [cordenadas[0], (cordenadas[1]+1)%8]
44          else:
45              return [cordenadas[0], abs(cordenadas[1]-1)]
46
47      def mutaNumero(numero):
48          valor = numero-1
49          randomiza = random.randint(0,1)
50          if randomiza == 0:
51              return ((valor+1) % 5)+1
52          elif randomiza == 1:
53              return (abs(valor-1) +1)
54
55      geneEscolhido = random.randint(0,(len(cromossomo)-1))
56      if(geneEscolhido<8):
57          return cromossomo[:geneEscolhido] + [mutaTerreno(cromossomo
                [geneEscolhido])] + cromossomo[geneEscolhido+1:]
58      elif(geneEscolhido==9):
59          return cromossomo[:geneEscolhido] + [mutaNumero(cromossomo[
                geneEscolhido])] + cromossomo[geneEscolhido+1:]
60      else:
61          return cromossomo[:geneEscolhido] + [mutaPosicao(cromossomo
                [geneEscolhido])] + cromossomo[geneEscolhido+1:]
```

APÊNDICE D — TABELAS DOS INDICADORES

Tabela D.1 – Metricas para indicadores obtidas atravéz de replays de Starcraft 2

	Drama	*Incerteza*	*Mudança de Liderança*
Média	0,05	0,20	0,23
Erro padrão	0,01	0,01	0,01
Mediana	**0,04**	**0,21**	**0,22**
Desvio padrão	0,04	0,04	0,09
Variância da amostra	0,00	0,00	0,01
Curtose	1,54	1,85	0,47
Assimetria	**1,24**	**-0,88**	**0,61**
Intervalo	**0,16**	**0,23**	**0,38**
Mínimo	0,00	0,06	0,08
Máximo	0,16	0,29	0,46
Soma	2,47	10,15	11,51
Contagem	50,00	50,00	50,00

	Jogadas Excepcionais	*Permanência*	*Perda*
Média	0,07	0,12	0,26
Erro padrão	**0,01**	**0,01**	**0,01**
Mediana	**0,07**	**0,12**	**0,25**
Desvio padrão	0,05	0,04	0,08
Variância da amostra	0,00	0,00	0,01
Curtose	**0,39**	**0,13**	**0,00**
Assimetria	**0,68**	**0,45**	**0,24**
Intervalo	0,21	0,16	0,33
Mínimo	0,00	0,05	0,10
Máximo	0,21	0,21	0,44
Soma	3,50	5,92	12,80
Contagem	50,00	50,00	50,00

Fonte: O Autor

Tabela D.2 – Nota de fitness atribuidas para indicadores

	Desv Drama	0,036	Desv Incerteza	0,041	Desv Mudança de liderança	0,088
Nota	Sup	Inf	Sup	Inf	Sup	Inf
1	0,037	0,037	0,203	0,203	0,23	0,23
0,9	0,046	0,028	0,21325	0,19275	0,252	0,208
0,8	0,055	0,019	0,2235	0,1825	0,274	0,186
0,7	0,064	0,01	0,23375	0,17225	0,296	0,164
0,6	0,073	0,001	0,244	0,162	0,318	0,142
0,5	0,082	0	0,25425	0,15175	0,34	0,12
0,4	0,091	0	0,2645	0,1415	0,362	0,098
0,3	0,1	0	0,27475	0,13125	0,384	0,076
0,2	0,109	0	0,285	0,121	0,406	0,054
0,1	0,118	0	0,29525	0,11075	0,428	0,032
0	0,127	0	0,3055	0,1005	0,45	0,01

	Desv Jogadas exepcionais	0,048	Desv Permanência	0,036	Desv Perda	0,079
Nota	Sup	Inf	Sup	Inf	Sup	Inf
1	0,065	0,065	0,118	0,118	0,256	0,256
0,9	0,077	0,053	0,127	0,109	0,27575	0,23625
0,8	0,089	0,041	0,136	0,1	0,2955	0,2165
0,7	0,101	0,029	0,145	0,091	0,31525	0,19675
0,6	0,113	0,017	0,154	0,082	0,335	0,177
0,5	0,125	0,005	0,163	0,073	0,35475	0,15725
0,4	0,137	0	0,172	0,064	0,3745	0,1375
0,3	0,149	0	0,181	0,055	0,39425	0,11775
0,2	0,161	0	0,19	0,046	0,414	0,098
0,1	0,173	0	0,199	0,037	0,43375	0,07825
0	0,185	0	0,208	0,028	0,4535	0,0585

Fonte: O Autor