Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Procural Content Generation - Algoritmo Genético (PCG-AG) com paralelização.

Raphael Luís Souza de Queiroz

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. Marcelo Ladeira

Coorientador
Prof. Dr. Claus de Castro Aranha

Brasília
2021

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Procural Content Generation - Algoritmo Genético (PCG-AG) com paralelização.

Raphael Luís Souza de Queiroz

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. Marcelo Ladeira (Orientador)
CIC/UnB

Prof. João Gondim
Coordenador do Curso de Engenharia da Computação

Brasília, 11 de outubro de 2021

Resumo

Esse projeto aborda o estudo de desenvolvimento de novos conteúdos especificamente para jogos utilizando Procedural Content Generation (PCG). Existem estudos de vários métodos para o PCG que são utilizado por várias empresas para criação de seu jogos. Dentre esses métodos, para esse projeto foi escolhido o método de Algoritmo Genético (AG) com divisão do problema em partes menores e paralelização na execução. O objetivo principal do projeto é otimizar a geração solução de problemas mais simples com um menor tempo de execução sem aumentar significativamente o custo e o esforço da máquina e extender esse estudo para problemas mais complexos. Com a disponibilização do *framework* Mario AI, foi possível prototipar todo o conteúdo gerado a partir desse estudo.

Palavras-chave: *Procedural Content Generation*, Algoritmo Genético, paralelização, *framework* Mario AI

Abstract

This project deals with the study of new contents developed specially for games using Procedural Content Generation (PCG). There are many studies for PCG that are used by many companies to develop their games. For this project the chosen method was Generic Algorithm (GA) spliting the main problem in small pieces and parallelizing the execution. The main objective to this paper is optimize the solution generation on simple problems lowering the execution time and without increasing significantly the costs and machine effort and extend it to more complex problems. Mario AI Framework made possible to prototype all the generated content from this study.

Keywords: *Procedural Content Generation*, Genetic Algorithm, parallelization, Mario AI *Framework*

Sumário

Lista de Figuras

Lista de Abreviaturas e Siglas

AG Algoritmo Genético.

NPC Non-playable Character.

PCG Procedural Content Generation.

PG Programação Genética.

UnB Universidade de Brasília.

Capítulo 1

Introdução

1.1 Contextualização

A quantidade de jogos, sejam eles para plataformas (PlayStation, Nintendo, Wii, entre outros) ou para computadores, tem crescido grandemente nos dias atuais. O número de jogadores que consomem essa grande quantidade de jogos tem aumentado mais ainda e não se limitam mais em idades e nem gêneros, havendo jogadores de 7 anos ou menos e até mesmo de 50 anos ou mais sendo homens e mulheres.

Com essa crescente enorme de usuários para seus jogos, várias empresas arriscam na criação de novos níveis para um jogo já existente ou até mesmo de um novo jogo desenvolvido inteiramente a partir do zero e com diversas mecânicas e novas informações. "Criar jogos de computadores envolve mais do que programação. Primeiro, precisa de um design. Depois, dependendo do gênero do jogo, precisa de personagens, mapas, níveis, tabuleiro, pista de corrida, árvores, pedras, armas, textura, efeitos sonoros, missões, nuvens e mais. Em muitas produções contemporâneas, criar todo esse conteúdo requer significativamente muito mais esforço e custo do que a atual programação do jogo."[1] Para otimizar todo o processo de criação, empresas buscam novas formas de gerar esses novos conteúdos continuamente e exaustivamente e para isso muitas utilizam do método de Procedural Content Generation (PCG).

O termo PCG refere-se a criação automática de conteúdo para jogos através de métodos algorítmicos [2]. O PCG possui uma curva de evolução bastante crescente, onde atualmente o algorítmo não apenas cria automaticamente um terreno do jogo, mas a criação de todo o conteúdo do mesmo, como por exemplo o comportamento dos chamados Non-playable Character (NPC) e até mesmo a engine de um jogo.

Essa crescente curva geraram inúmeros estudos acerca do assunto PCG e diversas formas de aplicar esse conceito junto com outros já existentes na programação, entre eles o *Machine Learn*, *Search Based*, etc. A Computação Evolutiva não fica de fora dessa lista.

Atualmente, existem diversos estudos sobre o Procedural Content Generation (PCG) com a metodologia existente em Algoritmo Genético (AG) ou Programação Genética (PG). Algoritmo Genético (AG) são algoritmos que possuem como base a otimização de soluções de problemas. Uma das vantagens de um algoritmo genético é a simplificação que eles permitem na formulação e solução de problemas de otimização [3].

1.2 Objetivos

Esse projeto visa o estudo do comportamento do algoritmo quanto a criação de vários níveis de jogos do Super Mario Bros usando Algoritmo Genético (AG) com uma paralelização na criação do nível. O projeto tem como principal objetivo a otimização na solução do problema de criação dos níveis em um curto intervalo de tempo e para cumprir esse objetivo a estrutura do problema é dividido em problemas menores e trablhados separadamente.

1.3 Justificativa

Existem muitos estudos sobre o PCG usando vários algoritmos. Dentre esses estudos, encontra-se a utilização de algoritmos genéticos com a construção de seus cromossomos e sua função *fitness* de várias formas distintas. Essa junção entre PCG e AG tem como objetivo encontrar as melhores soluções do problema proposto após diversas iterações. Porém, para problemas mais complexos e que demandam um maior estudo do problema como um todo, a utilização do AG pode ser custoso e até mesmo não encontrar a melhor solução se não tiver iterações suficientes. Para esses casos, a divisão do problema em problemas menores e independentes, diminui a complexidade existente no todo e para que essa divisão não aumente o tempo necessário para solução do problema, os problemas menores serão resolvidos em paralelo, sendo necessário uma boa separação entre esses "mini problemas".

Capítulo 2

Fundamentos Teóricos

2.1 Procedural Content Generation

"O termo *Procedural Content Generation* refere-se a criação automática de conteúdo para jogos através de métodos algorítmicos"[2]. É um ramo da síntese de mídia, que é um termo geral para produção artificial, manipulação e modificação de dados por meios automatizados, especialmente através do uso de algoritmos de inteligência artificial (IA). O termo automática refere-se a utilização de uma inteligência de máquina para a criação do conteúdo, podendo ou não ter interferência humana durante ou no final do processo.

Alguns métodos PCG podem ser classificados como [2]:

- Online ou Offline: refere-se a criação de conteúdo em tempo real (online) ou não (offline).
- Genérica ou adaptativa: o conteúdo pode ser criada uma vez só e utilizada sem alterações (genérica) ou pode sofrer alterações a medida que o conteúdo é consumido para se adaptar ao usuário (adaptativa).
- Geração automática ou autoria mista: o conteúdo gerado pode ser totalmente criado pelo algoritmo (automática) ou pode haver interferência humana no processo (autoria mista).
- Necessário ou opcional: o conteúdo gerado pode nem sempre ser crucial para o desenvolvimento do jogo, tornando-se um conteúdo opcional, enquanto existem conteúdos que são cruciais para o jogo, tornando-se necessário.

Para que um sistema seja considerado de fato um sitema de PCG, alguns critérios precisam ser avaliados. Primeiro de tudo, o termo "conteúdo"deve ser definido como tudo que existe em um jogo: níveis, mapas, missões, texturas, regras, dinâmicas e estruturas [4]. Porém, nem tudo é considerado um conteúdo para o método PCG. Por exemplo, um

jogador artificial para um jogo de tabuleiro. Sendo assim, a metodologia PCG, mesmo utilizando IA, se diferencia das outras abordagens, pois enquanto estes se preocupam em como jogar o nível, o método PCG se preocupa em como criar o nível.

A abordagem PCG, quando refere-se a jogos, propõe alguns critérios de avaliação para verificar se o conteúdo final é válido. Dentre elas, temos: velocidade, confiabilidade, diversidade, entre outras [5]. O mais importante e o que será trabalhado nesse projeto é a jogabilidade, ou seja, se é possível chegar ao fim do mapa ou nível gerado.

2.2 Algoritmos Genéticos

"Algoritmos Genéticos (AG) são algoritmos de otimização inspirados na seleção natural e na reprodução genética"[6] que buscam obter a melhor solução para um determinado problema. Fazem parte dos algoritmos de busca existentes na área da computação, mas se difere das demais técnicas nas seguintes características [7]:

- Trabalham com uma codificação do conjunto de parâmetros e não com um parâmetro apenas. Esses parâmetros são chamados de genótipos e seu conjunto forma um cromossomo.
- Trabalham com uma população e não com um único ponto.
- Trabalham com regras de transição probabilísticas e não determinísticas. As mais conhecidas são as taxas de cruzamento e de mutação.
- Trabalhar com informações de custo ou recompensa.

2.2.1 Seleção natural e reprodução genética

AG faz uso da evolução de indivíduos para atingir seus objetivos. Cada indivíduo possui suas próprias características e elas são representadas por um cromossomo, que é um agrupamento de genes. Com o passar do tempo, a população evolui através da reprodução e geram-se novos indivíduos e mutações dos mesmos. Naturalmente, os novos indivídos gerados herdam características de sua geração anterior. Usando o conceito de seleção natural de Darwin, os indivíduos com as melhores características são mantidas na população e esse conceito se repete para gerações futuras.

Em computação, as características de cada indivíduo são codificadas dentro de um genótipo

No contexto de programação, o indivíduo representa uma possível solução para um problema apresentado. Logo, um conjunto de soluções para o problema é o que chamamos

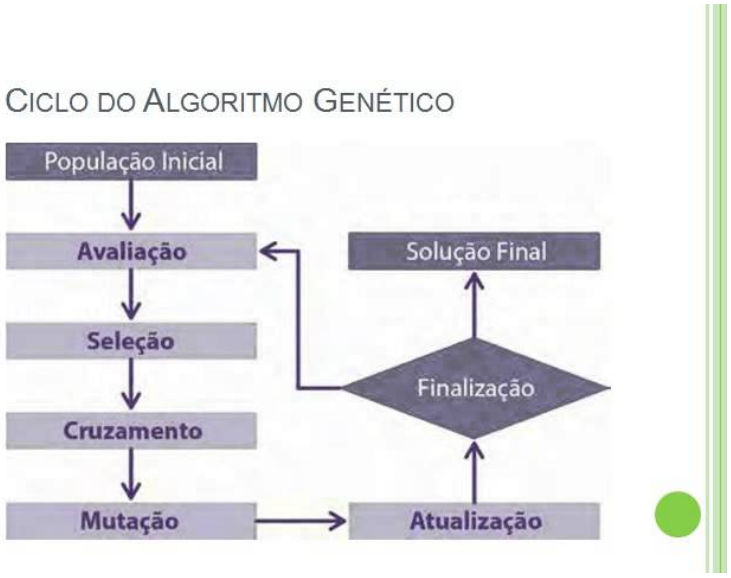

Figura 2.1: Estrutura do Algoritmo Genético

de população e os mesmo conceitos de seleção natural ocorrem nesse contexto, onde as soluções mais recentes tendem a serem melhores ou iguais às anteriores.

Para se atingir esse objetivo, a modelagem de um AG é muito importante, ou seja, deve-se encontar maneiras de avaliar a qualidade da nova solução gerada. Para isso, existem a função *fitness*.

2.2.2 Cromossomo

O cromossomo é responsável pelas informações da característica do indivíduo. Na computação, o cromossomo é representado por uma *string* (estrutura homogênea unidimensional). Esta pode ser binária, inteira ou real. Qualquer informação sobre o indivíduo pode facilmente ser representado por essa estrutura e interpretada através da mesma.

2.2.3 Função *Fitness* ou Função de Avaliação

A função *fitness* é a estrutura mais importante na modelagem de um AG. É através dela que podemos diferencias as inúmeras aplicações existentes, pois é através dela que um indivíduo é classificado dependendo da contextualização em que se encontra. O objetivo dessa função é mostrar o quão bom esse indivíduo é dentro do problema.

2.2.4 Operadores Genéticos

O AG trabalha de uma forma que permite que a novas características sejam introduzidas a cada geração, para isso são utilizado operadores genéticos.

Cruzamento ou *Crossover*

"O cruzamento é o operador responsável pela recombinação de características dos pais durante a reprodução, permitindo que as próximas gerações herdem essas características. É considerado o operador genético predominante"[7]. Sendo este o operador predominante, a probabilidade de ocorrência é alta. Esse operador genético pode ser utilizado de várias maneiras, entre eles:

- One pointer crossover: Para esse cruzamento, é escolhido apenas um ponto de cruzamento entre os indivídios para serem trocados. Informações de um indivíduo posteriores a esse ponto são trocados com as informações do outro indivíduo anteriores a esse ponto escolhido.
- Multi pointer crossover: Para esse cruzamento, dois ou mais pontos dentro do cromossomo são escolhidos para realizar a troca de informações. O processo de troca é equivalente ao one poiter crossover, porém o cromossomo é dividido de acordo com a quantidade de pontos escolhidos e é alternado a troca das informaçõe. Ou seja, informações anteriores ao primeiro ponto de um indivíduo é mantido, depois é trocada informações entre o primeiro e o segundo pontos escolhidos e assim sucessivamente).
- Uniforme: Nessa metodologia de crossover, não há um ponto escolhido, mas é usado uma taxa de probabilidade em cada elemento do cromossomo para verificar se ocorrerá a troca de informações entre os indivíduos.

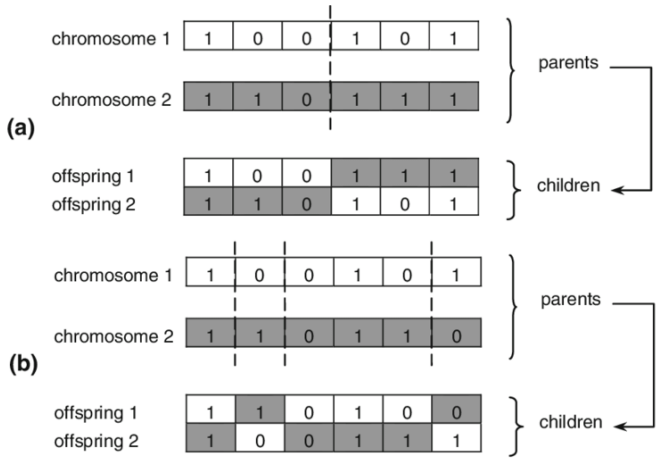

Figura 2.2: Representação dos métodos de cruzamento

Mutação

"A mutação é o operador responsável pela manutenção da diversidade genética da população, alterando arbitrariamente um ou mais componentes do cromossomo fornecendo assim,

meios para introdução de novas características na população"[7]. O operador de mutação assegura que a probabilidade de se chegar a um espaço de busca nunca seja zero, ou seja, impede que se atinga o mínimo local, pois ao alterar apenas 1 (uma) única informação, várias novas possibilidades de soluções são introduzidas ao problema. É considerado o operador secundário e, assim, sua probabilidade de ocorrência é baixa.

Figura 2.3: Representação do operador de mutação

Capítulo 3

Ferramentas Utilizadas

3.1 Linguagem Python

Python é uma linguagem de programação de alto nível, ou seja, com sintaxe mais simplificada e próxima da linguagem humana; orientada a objeto, baseado principalmente nos conceitos de classes e objetos; de tipagem dinâmica, onde o tipo da variável é dada durante o tempo de execução de acordo com o valor do dado. Essa linguagem foi criada no final dos anos 1980 por Guido Van Rossum e possui uma sintaxe semelhante ao ABC. Hoje em dia é uma das linguagens mais usadas por programadores e a mais indicada para ser ensinada a novos desenvolvedores.

Os algoritmos utilizados neste projeto foram escritos na linguagem Python (v3.9.5) pela facilidade de aplicação, pelo conhecimento do autor e por possuir uma diversidade de bibliotecas que facilitaram a implementação do código e a geração de resultados. Mesmo a linguagem *Python* possuindo diversas bibliotecas para auxílio para o desenvolvedor, o algoritmo implementado nesse programa não utilizou nenhuma biblioteca específica para o algoritmo genético e teve o uso de bibliotecas como o *numpy* e o *matplotlib* para que sejam realizados os cálculos e apresentados os resultados em gráfico para fácil compreensão.

3.2 Mario AI *framework*

O Mario AI *framework* foi introduzido em 2009 para uma competição de Mario AI que buscava a criação e desenvolvimento de uma inteligência artificial para geração de vários níveis do jogo Super Mario Bros. Foi desenvolvido em cima do Infinite Mario Bros (IMB), um clone do Super Mario Bros com código aberto [8]. O *framework* foi desenvolvido na linguagem Java e os mapas são gerados proceduralmente usando o método do criador do IMB, Markus Perrson [8]. Esses mapas são escritos em um arquivo tipo texto seguindo uma regra de símbolos para realizar a simulação do nível.

Figura 3.1: Mario AI *framework* referência de símbolos

Capítulo 4

Metodologia

A proposta deste projeto é de gerar níveis para o jogo Super Mario Bros utilizando algoritmo genético com geração paralela. Para isso, o jogo é dividido em quatro elementos considerados principais e independentes, ou seja, elementos que são necessários para ser gerado um mapa do jogo e que um elemento não afeta a criação do outro. São eles: *ground*, *block*, *enemy* e *coin* [9]. Para cada elemento, um cromossomo com suas características é gerado e o AG trabalhará em cima deles separadamente.

A população de níveis gerados ao final da última iteração não sofrerá futuras mudanças por parte do autor ou por parte de um outro agente. Sendo assim, a geração dos níveis atenderá aos métodos PCG *offline*, genérica, automática e necessário.

4.1 Definição dos Cromossomos

4.1.1 *Ground*

O cromossomo do elemento *ground* é um vetor de inteiros e contém as seguintes características:

- Altura máxima: a altura máxima é a característica mais importante do cromossomo. O valor desta característica é um inteiro entre 0 e 8, onde 0 significa que há um buraco e 8 é determinado por duas vezesa a altura máxima que o personagem Mario consegue alcançar durante o pulo.
- Tamanho máximo do cromossomo: essa característica informa o tamanho máximo do vetor de inteiros do cromossomo. O valor aqui determinado é igual ao valor máximo suportado para a renderização no Mario AI *framework* (132).
- Sequência: essa característica determina a quantidade máxima de sequência que um valor inteiro da altura pode ter. O valor é um inteiro entre 1 e 9, onde o valor

máximo 9 é determinado pela distância máxima que o personagem Mário consegue alcançar em um pulo, diminuindo assim a chance dele perder o jogo.

4.1.2 *Block*

O cromossomo do elemento *block* é um vetor de inteiros e contém as seguintes características:

- Tipo de bloco: é a característica mais importante do cromossomo. O valor é um inteiro entre 0 e 10, onde 0 significa que naquela posição do vetor não há um bloco e os valores de 1 a 10 equivalem ao tipo de bloco referenciado nos símbolos dos Mario AI *framework*.
- Tamanho máximo do cromossomo: essa característica informa o tamanho máximo do cromossomo. O valor aqui segue a mesma definição da característica no cromossomo explicado no elemento *ground*.
- Altura máxima: é a característica que informa a altura máxima que um bloco pode ficar. Possui valores entre 1 e 4, onde a altura máxima 4 é definida pela altura que o personagem Mario consegue pular. O valor desta característica é fixa, ou seja, todos os tipos de blocos irão permanecer na altura aqui definida.

4.1.3 *Enemy*

O cromossomo do elemento *enemy* é um vetor de inteiros e contém as seguintes características:

- Tipo de inimigo: é a característica mais importante do cromossomo. O valor é um inteiro entre 0 e 5, onde 0 significa que naquela posição do vetor não há um inimigo e os valores de 1 a 5 equivalem ao tipo de inimigo referenciado nos símbolos do Mario AI *framework*
- Tamanho máximo do cromossomo: essa característica informa o tamanho máximo do cromossomo. O valor aqui segue a mesma definição da característica no cromossomo explicado no elemento *ground*.
- Sequência máxima: é a característica que informa a quantidade máxima de inimigos do mesmo tipo durante o nível. Possui o valor entre 1 e 26, sendo assim, cada um dos inimigos podem se repetir até 26 vezes, totalizando um máximo de 130 inimigos. O valor máximo é menor em duas unidades que o tamanho máximo, pois essas duas posições estão ocupadas com as informações de início, representado pela letra M, e pelo final do mapa, representado pela letra F.

4.1.4 *Coin*

O cromossomo do elemento *coin* é um vetor de inteiros e contém as seguintes características:

- Tipo de moeda: é a característica mais importante do cromossomo. O valor é um inteiro entre 0 e 1, onde 0 significa que naquela posição do vetor não há uma moeda e 1 significa que há. Existem duas formas de se obter moeda segundo a referência de símbolos do Mario AI *framework*, na forma de uma moeda e na forma de um bloco onde o personagem Mario deverá acertar o bloco para se conseguir a moeda. Tendo em vista que a segunda forma exija uma atividade a mais do personagem, ela foi transferida para o elemento *block*.
- Tamanho máximo do cromossomo: essa característica informa o tamanho máximo do cromossomo. O valor aqui segue a mesma definição do cromossomo explicado no elemento *ground*.
- Altura máxima: é a característica que informa a altura máxima que uma moeda pode ficar no mapa. Esse valor segue a mesma definição da característica explicada no cromossomo *block*.
- Quantidade máxima: é a característica que informa a quantidade máxima de moedas no mapa. O valor é um inteiro entre 1 e 130. A diferença entre o valor máximo dessa característica em duas unidades segue a mesma definição da característica Sequência máxima do cromossomo *enemy*.

4.2 Execução do Algoritmo Genético

Após a divisão e criação dos cromossomos, um algoritmo genético é executado em cada um separadamente. Para esse projeto não foi utilizado uma biblioteca específica da linguagem *Python*, ou seja, cada processo dos operadores genéticos e da função *fitness* foi escrito pelo próprio autor.

Algumas regras precisaram ser definidas para que o código conseguisse rodar sem problemas. São elas:

- Tamanho da população: O tamanho da população deverá ser um valor par. Isso é explicado para que todos os indivíduos da geração consiga se reproduzir.
- Número de gerações: O número de gerações representa a quantidade de iterações que o código vai executar. Ela não tem um valor máximo definido, mas deve ter um valor mínimo de 2 (dois).

4.2.1 Operadores Genéticos

Uma das propriedades do algoritmo genético é trabalhar com dois tipos principais de operadores genéticos: o *crossover* e a mutação. Cada operador possui uma probabilidade de acontecer e deve ser definido no início do projeto. Para esse trabalho, foi definido uma ordem de execução dos operadores. O primeiro a ser executado é o *crossover* entre o casal de indivíduos e depois executa-se a mutação para cada novo indivíduo gerado.

Crossover

Para o operador de *crossover* ou cruzamento, uma taxa e um método de cruzamento foram determinados seguindo a literatura sobre algoritmos genéticos. Os valores padrões da taxa variam entre 60% e 65% dependendo do tamanho da população inicial. Para este projeto, a taxa possui valor de 75% e o método definido foi o *Multi pointer crossover*, para que haja uma maior diversidade entre as características trocadas entre os pais, e foi definido somente 2 (dois) pontos de cruzamento entre os indivíduos, por ser um valor mais simples de trabalhar em um *crossover*.

Um casal de indivíduos que passar pelo processo de cruzamento poderão ter seus pontos de cortes diferentes. Esse valor é escolhido randomicamente entre o início e a metade do cromossomo para o primeiro ponto e entre a metade e o final do cromossomo para o segundo ponto. Após gerado esses pontos de corte, as informações dos dois indivíduos são trocados e copiados em seus respectivos novos indivíduos.

Mutação

Para o operador de mutação, uma taxa foi determinada seguindo a literatura sobre algoritmos genéticos. Os valores padrões da taxa variam entre 0,1% e 5% dependendo do tamanho da população inicial. Para este projeto, a taxa possui valor de 1%.

Após a troca de informação ocorrida no *crossover*, cada novo indivíduo gerado passa pelo processo de mutação. Neste projeto, a taxa de mutação é aplicada em cada posição do vetor cromossomo em cada elemento do jogo e caso ocorra a mutação, a informação dominante do elemento é trocado por um novo valor (i. e., no caso do elemento *ground*, a informação dominante é a altura; vide informações sobre os cromossomos).

4.2.2 Função *Fitness*

O principal elemento de um algoritmo genético é a função *fitness* (ou função de avaliação) e deve ser bem determinado para que o objetivo do projeto seja alcançado. Para este projeto, a função de avaliação é verificado pela taxa de aleatoriedade de cada elemento.

Essa taxa é calculada verificando quantas mudanças da sequência anterior há em cada cromossomo dividido pelo tamanho máximo do cromossomo (132).

$$TA = QM/TMC \tag{4.1}$$

Onde,

- TA: Taxa de Aleatoriedade
- QM: Quantidade de Mudanças
- TMC: Tamanho Máxio do Cromossomo

Essa taxa possui valores entre 0% e 100%, onde 0% significa que o nível não possui variação e 100% significa que o nível varia em todas as posições. Ambos os valores de mínimo e de máximo da taxa são considerados valores de jogabilidade 0 (zero), pois queremos que o personagem Mario realize as ações propostas pelo jogo (andar pra direita, andar pra esquerda e pular) e não queremos que o jogo seja visualmente poluído com subidas e descidas a todo instante.

O objetivo da função de avaliação é o decréscimo dessa taxa, ou seja, é encontrar o mínimo aceitável para considerar um jogo que não tenha jogabilidade zero (taxa igual a 0%). Para isso, um critério de parada é definido, através de testes realizados no código, e possui valor de máximo de 50% e mínimo de 35%. Qualquer valor acima do máximo indica que a solução otimizada não foi alcançada e se uma geração atingir o valor mínimo, qualquer geração seguinte deverá manter esse mesmo valor de taxa.

4.3 Método Sequencial x Método Paralelo

O objetivo deste projeto é verificar a eficiência gerada quando a estrutura do algoritmo genético dividido em problemas menores é executado de forma paralela.

O método sequencial permite que cada elemento do jogo Super Mario Bros seja gerado um de cada vez e ao final do último elemento as populações são unificadas em uma matriz que será convertido no mapa em um arquivo do tipo texto utilizando os símbolos da 4.2.

Neste método, enquanto um elemento está passando pelo processo de algoritmo genético, os outros ficam em modo de espera até o final da execução atual. Cada população final é avaliada e ordenada para que os melhores de um elemento se juntem com os melhores dos outros elementos e os piores de um elemento se juntem com os piores dos outros elementos.

No entanto, este método tende a possui um tempo de execução diretamente proporcional ao tamanho da população e ao número de gerações definido no algoritmo genético, ou

seja, quanto maior o tamanho da população ou maior o número de gerações, maior será o tempo de execução e mais demorado será para chegar no resultado final do PCG-AG.

A ideia do método paralelo é diminuir essa proporcionalidade existente entre essas 3 (três) variáveis. Para que esse objetivo seja atingido, os elementos do problema devem ser divididos de forma que se tornem independentes e toda a estrutura do algoritmo genético (função *fitness*, operaedores genéticos) deve ser realizado com bastante cuidado, ou seja, um elemento não dependa de outro para ser gerado. A forma que os elementos foram divididos neste projeto garante essa primeira condição, sendo assim, a proposta de paralelização pode ser estudada.

Diferentemente do método sequencial, enquanto um elemento do problema é executado, os outros também são executados paralelamente. Para isso, algumas mudanças foram feitas no código. Por exemplo, cada elemento possui um espaço de memória alocado contendo suas respectivas informações, ou seja, há uma variável para cada elemento. O método sequencial utilizava apenas uma variável para os 4 (quatro) elementos, pois ao executar o próximo elemento, essa variável era esvaziada. Em termos computacionais, essa mudança não possui impacto significativo para os computadores existentes nos dias de hoje e a mudança garante que não haja troca de informações entre elementos diferentes durante a operação de *crossover* e de mutação.

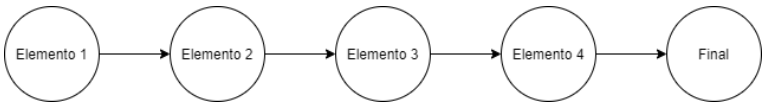

Figura 4.1: Gráfico método sequencial

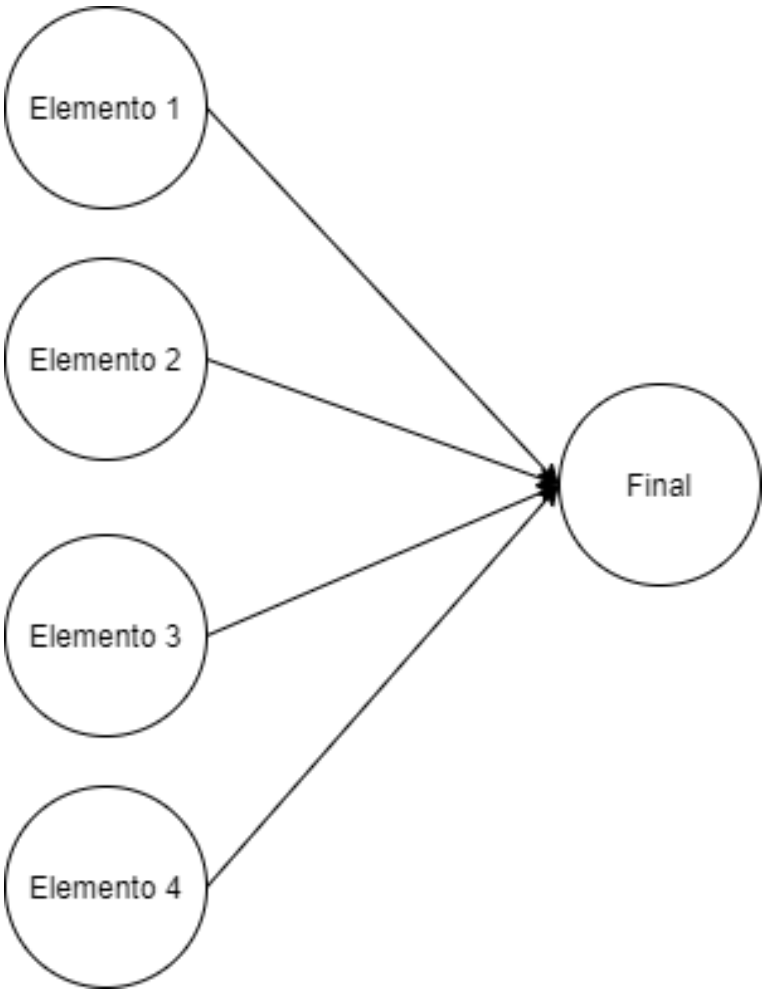

Figura 4.2: Gráfico método paralelo

Capítulo 5

Experimento

Nesta seção, será apresentado os resultados gerados pelo sistema proposto nas seções anteriores. Para isso, o sistema deverá cumprir com o objetivo proposto por este projeto, ou seja, deverá apresentar uma melhora no tempo de criação de novos mapas para o jogo Super Mario Bros sem que seja aumentado o custo e o esforço computacional. Também será avaliado se a população final gerado possui uma grande quantidade de níveis considerados jogáveis.

5.1 *Setup*

Os resultados apresentados a seguir tiveram as seguintes configurações de algoritmo genético:

- Tamanho da população: O tamanho da população necessariamente precisa ser um valor inteiro par, para que cada indivíduo tenha seu par para reproduzir. Sendo assim, o valor utilizado no estudo é de 50 indivíduos.
- Número de gerações: O número de gerações necessariamente precisa ser um valor inteiro maior que 2 e o valor máximo depende da máquina que vai executar o algoritmo. Neste projeto, o número máximo alcançado na fase de teste de estresse computacional foi de 2000 gerações, porém, para fins de comparação, o valor máximo é de 200 gerações.

As comparações feitas entre as metodologias devem exigir que o mesmo processamento do algoritmo genético ocorrido no método sequencial se repita no método paralelo. Para cumprir essa condição, é utilizado *seeds*. Estas irão salvar o passo-a-passo ocorrido na primeira geração do método sequencial e será utilizado para executar novamente no método sequencial e em seguida será executada no método paralelo. A segunda execução

no modelo sequencial se dá para fins de se retirar o atraso que possa existir quando os valores de ponto de corte e de mutação foram geradas randomicamentes, por exemplo.

5.2 Resultados

A primeira avaliação que foi verificada está relacionada à função *fitness*. Como descrição na seção 4 deste projeto, o objetivo do algoritmo genético é encontrar o mínimo aceitável para a função, ou seja, deve haver um decréscimo no valor da função de avaliação proposta.

Como cada elemento do jogo Super Mario Bros representa um cromossomo e irá passar pela execução de um AG, a função de avaliação é aplicada nos 4 (quatro) elementos separadamente. Em ambas as abordagens, podemos perceber que a função de avaliação possui um decréscimo leve com o aumento no número de gerações existente no algoritmo. O comportamento igual entre os dois métodos referindo-se ao decréscimo da taxa de aleatoriedade é esperado, tendo em vista que a execução salva nas *seeds* são as mesmas e os métodos se diferem somente no tempo de execução.

A segunda forma de avaliação está relacionada ao tempo de execução para a criação dos níveis obedecendo os mesmos critérios de execução em ambas as metodologias. Teoricamente, o método sequencial deve apresentar um maior tempo de execução considerando o critério de que um elemento só iniciará sua execução após o término do outro, sendo assim, o tempo de iteração de um elemento impacta no tempo de espera para que o próximo elemento seja executado e o tempo final se torna uma soma de tempos de iterações. O método paralelo deve dimiuir esse tempo significativamente, tendo em vista que os tempos de execução não são dependentes, ou seja, não existe um tempo de espera para execução. Sendo assim, o tempo final no método paralelo é verificado como o maior tempo de execução dentre os 4 (quatro) elementos do projeto. Para que esses valores de tempos sejam comparados, o sistema garantiu que toda a iteração que ocorre nos elementos do método sequencial ocorra no método paralelo, ou seja, o sistema garante que a *seed* utilizada e o passo-a-passo são as mesmas para os dois.

Por último, a terceira avaliação consiste na verificação se a população final gerada possui mais níveis considerados jogáveis. Utilizando a verificação apresentada na seção 4 deste projeto, a verificação da taxa de jogabilidade é verificado ao final de toda a iteração e na junção dos elementos em um mapa final.

Referências

[1] Julian Togelius, Jim Whitehead, Rafael Bidarra: *Guest editorial procedural content generation in games.* 1

[2] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley Cameron Browne: *Search-based procedural content generation: A taxonomy and survey.* 1, 3

[3] Miranda, Marcio Nunes de: *Algoritmos genéticos fundamentos e aplicações.* 2

[4] Julian Togelius, Alex J. Champandard, Pier Luca Lanzi Michael Mateas Ana Paiva Mike Preuss Kenneth O. Stanley: *Procedural content generation: Goals, challenges and actionable steps.* 3

[5] Noor Shaker, Mark J. Nelson, Julian Togelius: *Book Procedural Content Generation.* Springer International Publishing Switzerland, 2016. 4

[6] Cole, David A: *An Introduction to Genetic Algorithms for Scientists and Engineers.* World Scientific Publishing Co. Pte. Ltd, 1999. 4

[7] *Algoritmos genéticos.* 4, 6, 7

[8] Eduardo Hauck, Claus Aranha: *Automatic generation of super mario levels via graph grammars.* 8

[9] Lucas Ferreira, Leonardo Pereira, Claudio Toledo: *A multi-population genetic algorithm for procedural generation of levels for platform games.* 10