Criação automática de fases de jogos digitais via sintonização de parâmetros de um gerador procedural através de um modelo de otimização com muitos objetivos

Vitor G. S. L. Peixoto[1], **Elizabeth F. Wanner**[1], **André R. da Cruz**[1]

[1]Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Belo Horizonte, MG – Brasil

vitorgslpeixoto@gmail.com, {efwanner, dacruz}@cefetmg.br

Resumo. *Fases de jogos virtuais geralmente são criadas manualmente por equipes de* designers *e, com o aumento do porte dos jogos e da sua demanda, a geração procedural de conteúdo pode trazer vários benefícios. Este trabalho de conclusão de curso propõe um procedimento para a sintonia dos parâmetros de um gerador procedural de fases de jogos virtuais através da solução de um modelo de otimização com quatro objetivos que modelam critérios conflitantes e que descrevem qualidades específicas das fases geradas. Ao final do trabalho, o procedimento de sintonização de parâmetros é apresentado, bem como seus resultados. Além disso, a eficácia dos algoritmos evolutivos NSGA-III, CTAEA e RVEA em relação ao indicador de hipervolume da fronteira de Pareto foi comparada e, dos resultados, o melhor conjunto não dominado encontrado em uma execução foi apresentado e analisado, ilustrando os mapas gerados utilizando os parâmetros destas soluções.*

1. Introdução

O desenvolvimento de jogos virtuais tem como parte importante a criação de conteúdos, sejam quebra-cabeças, mapas, *assets*, adversários, entre outros. Esse processo em muitos jogos é feito de maneira manual por times de *designers*. O uso de métodos computacionais para gerar conteúdo é chamado de Geração Procedural de Conteúdo (do inglês, *Procedural Content Generation* – PCG). Os métodos de PCG tem sido usados na indústria de jogos com o objetivo de aumentar o interesse em re-jogar jogos, reduzir os custos de produção e consumir menos armazenamento com conteúdo [Summerville et al., 2018].

Dentro dos conteúdos de jogos, uma área de estudo com potencial é a de geração de fases procedurais. Existem estudos de aplicação de técnicas como Algoritmos Evolutivos, Gramáticas Generativas e Livres de Contexto, Autômatos Celulares e Redes Neurais Bayesianas na geração procedural de fases, mais especificamente fases de masmorras (do inglês, *dungeons*) e de plataformas [Compton e Mateas, 2021; van der Linden et al., 2014]. Além disso, existem estudos de meta-PCG, ou seja, o uso de técnicas para criar proceduralmente geradores de conteúdo, ou Geradores Procedurais de PCG [Kerssemakers et al., 2012].

O uso de PCG em jogos trás um conjunto de vantagens. Em jogos com fases pré-definidas, jogadores tendem a decorar caminhos, quebra-cabeças, personagens, locais de itens e outras características de fases após tentativas sucessivas. Uma vez que um conjunto

virtualmente infinito de conteúdos pode ser gerado com PCG, o valor de re-jogar jogos com conteúdo gerado proceduralmente aumenta.

Outra vantagem se dá na redução de custo e de esforço de produção de conteúdo. PCGs podem gerar diversos conteúdos baseados apenas em regras pré-definidas ou pequenos conjuntos de conteúdos de exemplo. Gerar altas quantidades de conteúdo manual pode ter um custo significativamente maior, podendo se tornar um processo inclusive não escalável [Iosup, 2009].

O objetivo deste trabalho é elaborar e solucionar um modelo de otimização *many-objective* para sintonizar os parâmetros de um gerador procedural de fases selecionado. O conjunto não dominado resultante possui soluções com parâmetros do gerador que os tornam capazes de elaborar automaticamente fases de jogos digitais otimizadas de acordo com os critérios conflitantes de avaliação, que descrevem níveis de dificuldade e de interatividade com os elementos do cenário. Para isso, foi elaborado um modelo de otimização com vários objetivos que visa sintonizar os parâmetros do gerador automático Notch Generator para o jogo Infinite Mario Bros. [Karakovskiy e Togelius, 2012]. Dessa forma, para selecionar um procedimento computacional que solucione tal modelo, foi planejado um experimento computacional em que se comparam a eficácia de três algoritmos da literatura em relação aos valores de hipervolume, ao se solucionar o modelo de otimização com orçamento computacional fixo (avaliações de função-objetivo). Os algoritmos evolucionários selecionados para análise neste trabalho são: *Non-dominated Sorting Genetic Algorithm III* (NSGA-III) [Deb e Jain, 2014], *Constrained Two-Archive Evolutionary Algorithm* (C-TAEA) [Li et al., 2019] e *Reference Vector Evolutionary Algorithm* (RVEA) [Cheng et al., 2016]. Ao final, exemplos de soluções são ilustradas através da descrição gráfica de fases geradas.

O restante deste texto é organizado da seguinte forma: a Seção 2 cita trabalhos relacionados que fizeram parte da revisão bibliográfica; a Seção 3 elucida a metodologia a ser aplicada; a Seção 4 define como os experimentos foram executados; a Seção 5 apresenta os resultados obtidos nos experimentos deste trabalho; e por fim, a Seção 6 anuncia as conclusões obtidas e os possíveis trabalhos futuros.

2. Trabalhos Relacionados

Nesta etapa foi feito um estudo de artigos e materiais relacionados com o tema deste trabalho, citando nas referências todos aqueles que foram pertinentes. Algumas das palavras chave em inglês utilizadas nas pesquisas foram: “Procedural Content Generation”, “Procedural Level Generation” e “Procedural Level Design” (Geração Procedural de Conteúdo, Geração Procedural de Fases e Design Procedural de Fases, tradução livre).

Em uma tentativa de se criar uma definição mais específica do que é a Geração Procedural de Conteúdo, Togelius et al., 2011 construíram dois geradores de conteúdo adaptativos para o jogo Infinite Mario Bros. (um clone em domínio público do jogo de plataforma Super Mario Bros.). O objetivo destes geradores era criar novas fases a partir das movimentações dos jogadores de maneira *offline*, em que os movimentos em uma fase impactam na geração das fases seguintes, e *online*, em que os movimentos influenciam em tempo real a geração das próximas seções da própria fase.

Como no trabalho anterior, pesquisas envolvendo jogos necessitam de um *benchmark*, geralmente composto por um conjunto de um ou mais jogos nos quais a pesquisa

pode ser aplicada. Karakovskiy e Togelius, 2012 analisam que, embora um *benchmark* de um jogo não consiga atender todos os tipos de pesquisa adequadamente, a existência de *benchmarks* padronizados pode trazer muito benefício para as áreas de pesquisa em que se adequam. Summerville et al., 2018 fazem uma análise de técnicas utilizadas para geração de conteúdo em diferentes gêneros de jogos: jogos de estratégia em tempo real - RTS (do inglês, Real-Time Strategy) como *StarCraft II*, jogos de Plataforma como *Super Mario Bros.* e *Legend of Zelda*, e jogos de cartas como *Magic: The Gathering*.

Karakovskiy e Togelius, 2012 também desenvolveram um *benchmark* baseado no jogo Infinite Mario Bros. chamado de Mario AI Framework. Este *framework* provê, entre outras funcionalidades, uma representação textual de fases (mapa e inimigos) e ferramentas que possibilitam a criação de geradores para gerá-las.

No quesito de comparação de geradores de conteúdo, Dahlskog et al., 2014 pontuam que este processo é muito manual, não havendo um bom método para comparar geradores de jogos diferentes, por exemplo. O trabalho propõe algumas métricas para auxiliar nesse processo, e faz um comparativo dos diferentes geradores do Mario AI Framework.

Pontes e Gomes, 2020 implementaram um gerador de fases para um jogo de plataforma em tempo real, utilizando como abordagem um Algoritmo Genético (AG) que gera diversas soluções candidatas para a próxima seção da fase na medida que o jogador progride. Estas soluções são avaliadas conforme uma função de *fitness*, que pontua seções jogáveis e penaliza seções mais difíceis ou não jogáveis. Desta forma o gerador objetiva gerar fases jogáveis e não muito difíceis.

O presente trabalho oferece uma vantagem em relação aos anteriores por oferecer um método automatizado baseado em técnicas de otimização com muitos objetivos para geração de fases. Ademais, este alia técnicas de computação evolutiva para sintonia dos parâmetros do gerador juntamente à técnica de busca A* para avaliar o conteúdo gerado conforme a performance do agente, simulando as ações de um jogador.

3. Metodologia

A metodologia utilizada neste trabalho é representada pelas seis etapas presentes na Figura 1. O problema que buscamos resolver é, escolhido um gerador de fases de jogos, sintonizar seus parâmetros de entrada de forma a gerar conteúdos que atendam um conjunto de objetivos pré-definidos. Desta forma, inicialmente fizemos a escolha do gerador, seguida por uma análise e definição dos parâmetros usados na geração e a definição dos indicadores de qualidade do conteúdo gerado por estes parâmetros. Com estas definições formulamos o modelo de otimização responsável pela sintonia, e definimos os algoritmos utilizados neste processo. Por fim, executamos experimentos e análises estatísticas para avaliar os resultados. O detalhamento destes passos se encontra nas seções a seguir.

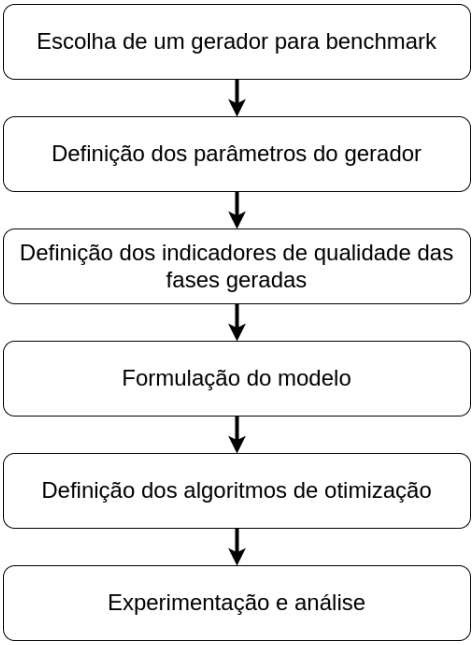

Figura 1. Metodologia utilizada no trabalho.

3.1. Escolha de um Gerador para *Benchmark*

Esta etapa do trabalho visou a escolha de um Gerador de Fases Procedural para utilizarmos como *benchmark*, de forma a otimizá-lo utilizando alguns algoritmos. Essa etapa ocorreu através de um estudo em trabalhos relacionados (vide Seção 2) considerando as diversas categorias de video-games e os desafios em gerar conteúdo procedural que variam entre elas.

Para o escopo deste trabalho optamos pela categoria de jogos de plataforma, sendo selecionado para este trabalho o jogo *Infinite Mario Bros.*, que consiste em um conjunto de fases bi-dimensionais e um personagem deve percorrê-las até o final para vencer. Cada fase é composta por um conjunto de blocos formando vales, montanhas e buracos que o personagem deve saltar. Há também inimigos que podem feri-lo durante o processo caso estes colidam com o personagem.

Deste jogo, o gerador selecionado foi o gerador padrão, implementado dentro do Mario AI Framework [Karakovskiy e Togelius, 2012]. Este gerador, chamado de Notch Generator, realiza a criação de fases combinando variações de pequenas pré-definições (canhões, montanhas, tubos, etc.), através de cálculos usando números pseudo-aleatórios, até alcançar um comprimento/altura de fase pré-definido. Uma análise mais detalhada das parametrizações do gerador é encontrada na Subseção 3.2.

Uma vez escolhido o gerador para *benchmark*, o problema que buscamos resolver é a sintonização de seus parâmetros de forma a atender um conjunto de objetivos.

3.2. Definição dos Parâmetros do Gerador

Uma vez escolhido o gerador para *benchmark*, a próxima etapa teve como objetivo definir quais parâmetros deste gerador seriam utilizados para geração de indivíduos (soluções que descrevem os parâmetros do gerador de mapas) durante a otimização. Após um estudo detalhado do gerador, da execução de testes e algumas modificações, os parâmetros

definidos encontram-se detalhados a seguir. Os seguintes parâmetros já existiam dentro do gerador e mantivemos no nosso modelo:

- Tipo de mapa ($tipo\_mapa$): variável discreta que define o tipo de nível a ser gerado. O valor zero habilita a geração de montanhas e o valor um habilita a geração de um teto feito de blocos.
- Dificuldade ($dificuldade\_fase$): variável discreta que define a dificuldade do mapa, e assume no nosso modelo valores de 0 a 15. Este parâmetro influencia na quantidade, tipo e variantes dos inimigos gerados (por exemplo, com ou sem asas), na chance de tubos conterem plantas carnívoras, e na chance de tubos, buracos e canhões serem gerados.
- Comprimento ($comprimento\_fase$): variável discreta que define o comprimento (em número de blocos horizontais) da fase. Este assume no nosso modelo valores de 149 a 746 blocos (menor comprimento das fases do Mario Bros. original até o dobro do maior comprimento).
- Altura ($altura\_fase$): variável discreta que define a altura (em número de blocos verticais) da fase. Esta assume no nosso modelo valores de 16 a 32 (menor altura das fases do Mario Bros. original até o dobro da maior altura).
- Tempo para completar ($tempo\_total$): variável discreta que define o tempo, em segundos, que o jogador tem para completar uma fase. Esta assume no nosso modelo valores de 30 a 300.

O gerador, por padrão, também trabalha com algumas variáveis não-parametrizáveis com valor fixo para calcular a probabilidade de gerar determinadas estruturas. Uma vez que estas constantes controlavam a probabilidade de gerar estruturas importantes para os mapas, optamos por adicionar fatores que as fizessem variar, trazendo mais diversidade às fases. Neste caso, os fatores adicionados foram:

- Geração de seção reta (p_1): possuía, por padrão, um valor fixo de 20. Adicionamos uma margem discreta variável de -5 a $+5$, portanto possui valor final entre 15 e 25.
- Geração de seção com montanhas (p_2): possuía, por padrão, um valor fixo de 10. Adicionamos uma margem discreta variável de -5 a $+5$, portanto possui valor final entre 5 e 15.
- Geração de seção com tubos (p_3): possuía, por padrão, um valor que variava de acordo com a dificuldade, no seguinte formato: $2 + dificuldade\_fase$. Adicionamos um multiplicador discreto no valor da dificuldade que varia de 1 a 5. Portanto, é definido por $2 + dificuldade\_fase \cdot multiplicador$ e possui valor final entre 2 e 77.
- Geração de seção com quedas (p_4): possuía, por padrão, um valor que variava de acordo com a dificuldade, no seguinte formato: $2 \cdot dificuldade\_fase$. Adicionamos um multiplicador discreto no valor da dificuldade que varia de 1 a 5. Portanto, é definido por $2 \cdot dificuldade\_fase \cdot multiplicador$ e possui valor final entre 0 e 150.
- Geração de seção com canhões (p_5): possuía, por padrão, um valor que variava de acordo com a dificuldade, no seguinte formato: $-10 + 5 \cdot dificuldade\_fase$. Adicionamos um multiplicador discreto no valor da dificuldade que varia de 0 a 10. Portanto, é definido por $-10 + 5 \cdot dificuldade\_fase \cdot multiplicador$ e possui valor final entre -10 e 740.

Com os parâmetros acima, o gerador faz uma primeira seleção de quais estruturas acima podem ser geradas. O cálculo de probabilidade para cada estrutura acima ser selecionada é dado pela Equação 1, em que t é a soma das variáveis p_i, tal que $t = \sum_{i=1}^{5} max(0, p_i)$. Ao final, caso mais de uma estrutura seja selecionada pelo gerador, a estrutura de maior índice (entre 1 e 5) é gerada.

$$prob_i = \frac{t + p_i - \sum_{j=1}^{i} p_j}{t}, i = 1, \ldots, 5. \quad (1)$$

Especificamente na função do código que gera seções com montanhas do gerador oficial foi encontrado um erro de implementação original. Se tratava de um erro de execução esporádico que ocorria apenas em casos específicos quando a montanha gerada estava no final do mapa e ultrapassava os limites dele. Neste caso um parâmetro de limite negativo era passado para o gerador de números aleatórios, resultando no erro de execução. Apesar de pouco frequente, esse erro impactaria nos nossos experimentos, que envolveriam a geração de diversos mapas em sequência. Analisamos o erro e implementamos uma correção, utilizando uma lógica já existente na própria função para pular a geração de estruturas inválidas. A correção segue o mesmo fluxo, pulando a geração da montanha caso não haja espaço suficiente para a geração.

3.3. Definição dos Indicadores de Qualidade das Fases Geradas

Uma vez selecionado o gerador e definidos os parâmetros utilizados para gerar os indivíduos, precisamos definir uma forma de avaliar a qualidade de cada solução gerada. Esta etapa, portanto, consistiu em definir os indicadores que foram usados para avaliar a qualidade dos mapas e, consequentemente, os objetivos do modelo de otimização.

Além dos geradores de conteúdo, o Mario AI Framework contém um pacote de agentes criados para tentar vencer os mapas, usando diferentes algoritmos e heurísticas. Ao instanciar um agente em um determinado mapa, este o percorrerá interagindo com o ambiente, tentando desviar ou atacar inimigos e pular quedas até chegar ao final do mapa no tempo definido. Dada esta execução, o framework gera um resultado com informações da tentativa do agente, como a quantidade de pulos efetuados, o número de inimigos mortos, a quantidade de dano sofrido, etc. Com isso, a execução de um agente em um mapa retorna informações que podem ser utilizadas para classificar a sua qualidade.

Para o escopo deste trabalho selecionamos um agente A* [Togelius et al., 2010] do framework para jogar os mapas e retornar os resultados, que foram utilizados na avaliação. Os indicadores de resultado que decidimos utilizar são:

- Número de golpes sofridos (num_golpes): quantos golpes o personagem sofreu durante aquela fase.
- Número de pulos (num_pulos): quantos pulos o personagem executou durante a fase.
- Número de monstros mortos (num_mortos): quantos monstros o personagem derrotou durante a fase.
- Maior pulo ($maior\_pulo$): pulo mais comprido no eixo x, em pixels (cada bloco tem 16×16 pixels).
- Maior tempo no ar ($tempo\_ar$): maior tempo, *em frames*, entre um pulo e a aterrissagem no chão.

- Tempo restante da fase ($tempo\_restante$): o tempo restante para completar a fase ao final da execução.

3.4. Formulação do Modelo

Considere x como vetor de decisão que possui as variáveis $tipo\_mapa$, $dificuldade\_fase$, $comprimento\_fase$, $altura\_fase$, $tempo\_total$, p_1, p_2, p_3, p_4 e p_5. Este vetor foi sintonizado no procedimento de otimização e utilizado posteriormente pelo gerador procedural. A região viável de busca X leva em consideração, para cada variável, os intervalos de valores mencionados na Subseção 3.2 e a restrição da Equação 2. Esta restrição visa garantir um tempo mínimo para que a fase seja completada baseado em seu tamanho, de forma que o comprimento da fase, dado um fator de escala, seja limitado superiormente pelo tempo máximo para passá-la. O fator escolhido foi arbitrário e maiores experimentações são necessárias para melhor análise.

$$0{,}2 \cdot comprimento\_fase \leq tempo\_fase \tag{2}$$

Após a definição dos indicadores de qualidade, restou a definição das funções-objetivo do modelo matemático a ser otimizado. Para o escopo deste trabalho optamos pela otimização com múltiplos objetivos, de forma a poder selecionar mais de um objetivo pertinente e obter conjuntos de soluções não-dominadas que atendam ambos em uma relação de compromisso em variadas distribuições. Utilizando os dados apresentados anteriormente, foram definidas as funções-objetivo apresentadas pela Equação 3.

$$\begin{aligned} f_1(x) &= num\_golpes(x) \\ f_2(x) &= num\_mortos(x) \\ f_3(x) &= num\_pulos(x) + maior\_pulo(x) + tempo\_ar(x) \\ f_4(x) &= tempo\_restante(x) \end{aligned} \tag{3}$$

O modelo matemático resultante é apresentado pela Equação 4.

$$\begin{aligned} \min_{x} \quad & f = [f_1(x), -f_2(x), -f_3(x), -f_4(x)] \\ \text{sujeito a:} \quad & x \in X \end{aligned} \tag{4}$$

A função $f_1(x)$ é uma representação de dificuldade baseada nos golpes que o agente sofre dos inimigos da fase durante uma execução. No jogo Infinite Mario Bros, um golpe é suficiente para matar o personagem, a não ser que o personagem tenha coletado algum *power-up* durante a fase, garantindo assim a resistência a mais um golpe. Minimizar $f_1(x)$ visa, portanto, obter fases mais fáceis, que o agente consiga completar.

A função $f_2(x)$ também é uma representação de dificuldade, porém baseada no número de inimigos derrotados pelo agente durante uma fase, seja pulando em cima deles ou através do uso de poderes de *power-ups* coletados. Maximizar $f_2(x)$ pretende, portanto, obter fases com muitos inimigos, mas que possam ser derrotados pelo agente, trazendo dificuldade às fases, mas também variabilidade.

A função $f_3(x)$ está relacionada aos pulos que o agente efetua durante uma fase, combinando distância e frequência. Maximizar $f_3(x)$ tende a gerar fases que tenham

combinações de muitos pulos, pulos muito longos e pulos muito altos, trazendo um nível de dificuldade que exigiria precisão de um jogador ao efetuá-los.

A função $f_4(x)$ é dada pelo tempo que restou de uma fase ao agente terminar sua execução. Maximizá-la pretende gerar fases que o agente consiga terminá-las com tempo de sobra, de certa forma facilitando-as.

Nossa hipótese é que minimizar $f_1(x)$, enquanto maximizamos os outros objetivos, garanta a geração de fases que respeitem uma relação de compromisso em termos de dificuldade, porém não imbatíveis.

O modelo *many-objective* [Ishibuchi et al., 2008] apresentado na Equação 4 é computacionalmente custoso, uma vez que é necessário simular um agente A* para avaliar uma função-objetivo.

3.5. Definição dos Algoritmos

Após a definição do modelo definido, também definiu-se os algoritmos que foram utilizados para otimizá-lo. Para o escopo deste trabalho, optamos por escolher mais de um algoritmo para comparar os resultados obtidos no gerador utilizado. Foram escolhidos três algoritmos *many-objective* com capacidade de lidar com restrições, detalhados abaixo.

3.5.1. NSGA-III

O *Non-dominated Sorting Genetic Algorithm III* (NSGA-III) é um algoritmo evolucionário que, assim como seu antecessor (NSGA-II), utiliza um algoritmo de ordenação de indivíduos não-dominados para selecionar a nova população entre as gerações.

A ordenação de indivíduos não-dominados ocorre através da separação das soluções de uma geração com base no número de indivíduos que as dominam, ou seja, que sejam melhores em pelo menos um dos objetivos, e pelo menos iguais em todos os outros. As fronteiras geradas por essa separação são conhecidas como fronteiras de Pareto. Dessa forma, os indivíduos das primeiras fronteiras (menos dominados) são priorizados na seleção.

Por outro lado, diferentemente do NSGA-II, o NSGA-III suporta bem otimização com com mais de três objetivos e, para tanto, utiliza um conjunto de vetores de referência no seu algoritmo de ordenação. Estes vetores ajudam na seleção de soluções não dominadas com base em sua distância perpendicular a elas [Deb e Jain, 2014].

3.5.2. C-TAEA

O algoritmo *Constrained Two-Archive Evolutionary Algorithm* (C-TAEA) é um algoritmo evolucionário criado para resolver problemas que contenham múltiplos objetivos e restrições, visando balancear a convergência, a diversidade e a viabilidade das soluções através das gerações.

A abordagem do C-TAEA é o uso de dois arquivos (populações) colaborativos simultaneamente. Um arquivo é orientado a convergência (CA), visando aproximar os indivíduos da fronteira de Pareto através da seleção (elitismo). O outro arquivo é orientado

a diversidade (DA) e tem o objetivo de manter a diversidade da população, explorando áreas inexploradas pelo CA e ajudando a sair de ótimos locais.

Ao final do processo, as operações de cruzamento ocorrem entre indivíduos dos dois arquivos após um segundo processo de seleção [Li et al., 2019].

3.5.3. RVEA

O algoritmo *Reference Vector Evolutionary Algorithm* (RVEA) é um algoritmo evolucionário para múltiplos objetivos guiado por vetores de referência – vetores arbitrários usados como entrada para o algoritmo – que influenciam no processo de seleção e de adaptação do algoritmo. A geração de descendentes do RVEA é muito semelhante a de outros algoritmos genéticos, com operação de cruzamento e mutação.

Uma das maiores diferenças do RVEA é no processo de seleção de indivíduos. Este processo ocorre através do particionamento dos indivíduos por proximidade dos vetores de referência, seguido de um cálculo de distância penalizado por ângulo (do inglês, Angle-Penalized Distance – APD), que visa balancear a convergência e a diversidade. O APD portanto calcula duas métricas para cada indivíduo da partição: como critério de convergência, a distância entre a solução e o ponto ideal (composto do melhor resultado individual de cada objetivo dentro da população), e como critério de diversidade, o ângulo agudo entre o vetor formado do ponto ideal ao indivíduo e o vetor de referência. Ao final, uma seleção elitista é feita baseada nos resultados do APD para definir a próxima população [Cheng et al., 2016].

4. Planejamento Experimental

4.1. Definição das Ferramentas

Para a execução dos nossos experimentos foi necessária a integração entre o gerador do Mario AI Framework e os algoritmos de otimização. Para otimização do gerador optamos pelo uso do framework Pymoo, um framework *open-source* para otimização mono e multi-objetivo construído em Python [Blank e Deb, 2020]. O framework disponibiliza a implementação dos algoritmos citados acima, bem como ferramentas importantes para definição do modelo, parametrização do experimento, geração de gráficos e análise de resultados. Além disso, uma vez que o gerador foi construído em Java, para possibilitar a comunicação entre o gerador e o otimizador, utilizamos a biblioteca Pyjnius, uma biblioteca construída em Python para carregar classes do Java [Kivy Team, 2021].

4.2. Implementação do Modelo e dos Algoritmos

Para implementação do modelo, o Pymoo fornece uma interface *Problem* que representa o problema que vamos otimizar. A implementação consistiu em definir a restrição, os quatro objetivos e as dez variáveis, explicitando os limites inferiores e superiores.

Além disso, para cada indivíduo de cada geração, implementamos uma função de avaliação que consiste em:

- Instanciar um novo gerador de mapa, com os parâmetros de tipo e dificuldade do indivíduo.

- Gerar um mapa com o comprimento e altura, tempo limite e parâmetros de estruturas (canhões, montanhas, etc.) definidos no indivíduo, e armazená-lo.
- Executar uma tentativa do agente A* de jogar e finalizar o mapa gerado, obtendo os resultados.
- Calcular as restrições e os valores dos objetivos a partir dos resultados do agente.

4.2.1. Cálculo do Hipervolume

Por se tratar de um problema com múltiplos objetivos, indicadores de performance precisam ser usados para comparar pares de conjuntos de soluções, ao invés de uma única solução. Além disso, como não é conhecido o conjunto não dominado e a fronteira ótima para nosso problema, de modo que indicadores de performance como GD [Van Veldhuizen, 1999] e IGD [Coello Coello e Reyes Sierra, 2004] que precisam de uma fronteira-alvo para comparação não são ideais. Portanto, optamos pelo uso do indicador de Hipervolume [Fortin et al., 2012].

A implementação que utilizamos do cálculo de hipervolume necessita da definição de um ponto de referência. A partir desse ponto, o algoritmo calcula o volume dominado pela fronteira de Pareto em relação a este ponto de referência.

Para uma comparação justa entre os resultados, utilizamos como referência um único e fixo ponto Nadir válido, ou seja, aquele composto pelos maiores valores individuais de cada objetivo entre as soluções, acrescido de $0{,}1$ em cada coordenada de objetivo.

4.3. Configuração das Execuções e Saídas Geradas

Devido ao custo computacional do modelo, o experimento envolveu a execução de cada um dos algoritmos com uma população de 20 indivíduos, e com um limite de 1000 avaliações máximas de função-objetivo como critério de parada. Os três algoritmos iniciaram com uma população gerada aleatoriamente e utilizaram os mesmos operadores de cruzamento e mutação.

O operador de cruzamento utilizado foi o *Simulated Binary Crossover* (SBX) [Deb et al., 2007], que opera sobre as variáveis de decisão de dois indivíduos pais utilizando um parâmetro chamado *eta* que define a semelhança que os indivíduos filhos terão em relação aos seus pais. Maiores valores para o parâmetro *eta* geram indivíduos mais parecidos com os pais, e menores valores geram indivíduos mais diversos. Os valores padrão da implementação do Pymoo são utilizados neste operador, com a probabilidade de cruzamento sendo de 90% e o *eta* sendo de 30, valor que torna os filhos semelhantes aos pais.

Já o operador de mutação utilizado é o *Polynomial Mutation* (PM), e a probabilidade de mutação utilizada também é a padrão da implementação, que se dá pela fórmula $1{,}0/num\_variaveis$, resultando numa probabilidade para o nosso modelo de 10%.

Além disso, dado o caráter estocástico do gerador, que utiliza números pseudo-aleatórios em diversos passos da geração, cada um dos algoritmos foi executado 30 vezes e coletada a respectiva amostra de valores de hipervolumes. Para cada uma das 30 execuções de cada algoritmo, foram armazenados os seguintes dados de cada solução não-dominada presente na fronteira de Pareto:

- Valores das variáveis de decisão e de objetivo.
- Conteúdo do mapa gerado pelos vetores não dominados.
- Gráfico de pétala com os valores normalizados de objetivos da fronteira de Pareto.

4.3.1. Análise Estatística dos Resultados

Ao final do experimento, obtivemos 90 fronteiras de Pareto, 30 de cada algoritmo e, consequentemente, uma amostra de 30 valores de hipervolume representando a qualidade de cada. Quanto maior forem os valores de hipervolume de um evolutivo, melhor será a sua performance.

Para definir o algoritmo com melhor resultado precisamos então verificar se há diferença estatística significativa, testando a hipótese nula que afirma que os algoritmos evolutivos experimentados possuem a mesma eficácia em relação à mediana de hipervolume com um nível de significância $\alpha = 0{,}05$.

Para verificar a existência de diferença estatística significativa optamos por testes não-paramétricos, uma vez que não assumimos as características de normalidade e homoscedasticidade nos nossos resultados, requisitos para os testes paramétricos [Derrac et al., 2011]. Os testes não-paramétricos escolhidos foram os testes de Friedman [Friedman, 1937] e o Post-Hoc Pareado Nemenyi [Nemenyi, 1963].

5. Resultados

Após a execução do experimento conforme o planejamento da seção anterior, obtivemos os resultados a seguir.

5.1. Hipervolume e Algoritmos

A Figura 2 apresenta concomitantemente gráficos boxplot, cuja média é o ponto laranja, a metade de um gráfico de violino representando a densidade das soluções em relação aos valores de hipervolume, e um gráfico de dispersão apresentando o resultado de cada uma das 30 execuções dos algoritmos. É possível perceber que o algoritmo RVEA obteve valores menores de hipervolume do que os demais, tendo um desempenho abaixo dos outros dois algoritmos na otimização do modelo. Já sobre os algoritmos NSGA-III e CTAEA, é possível notar intercessão entre os valores de amostras, apesar do primeiro apresentar quartis maiores.

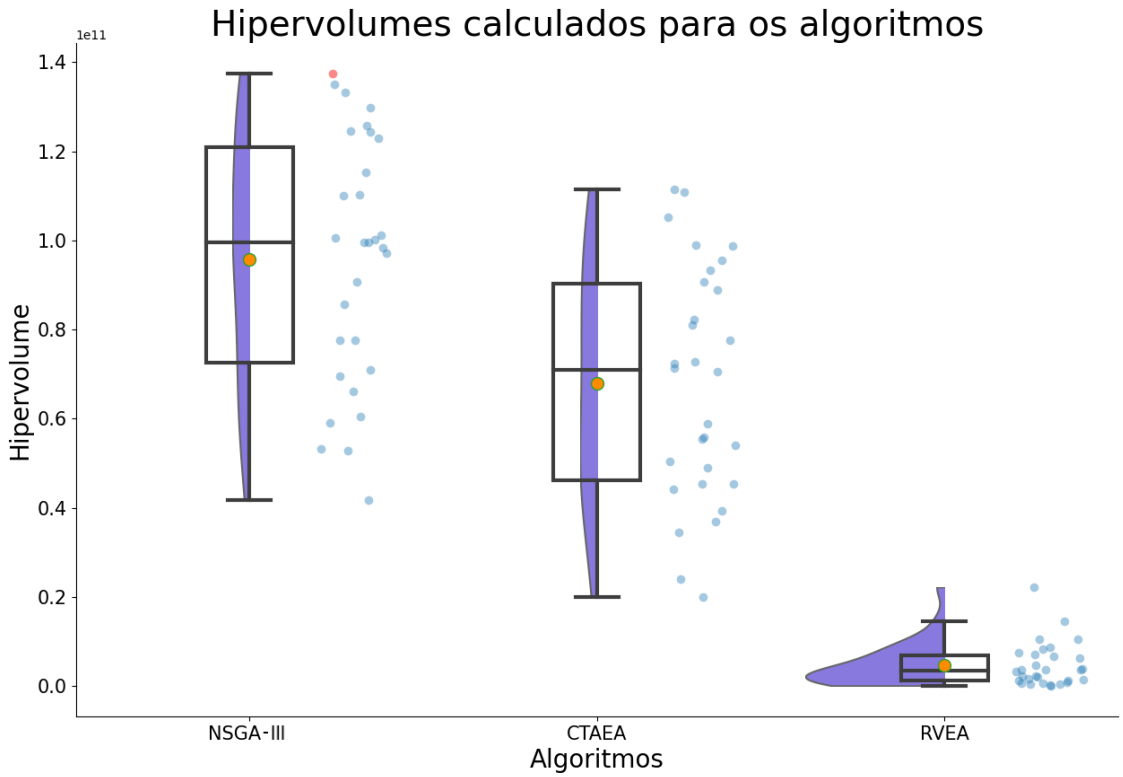

Figura 2. Gráfico combinado do hipervolume dos algoritmos.

Após aplicação do teste de Friedmann, este retornou o valor de Estatística χ^2 igual a $47{,}40$ e valor-P igual a $5{,}09 \cdot 10^{-11}$, que é muito menor que o nível de significância $\alpha = 0{,}05$ considerado. Isto indica que há diferença estatística significativa entre algum par de amostras de hipervolumes encontrados pelos três algoritmos.

Para identificar entre quais algoritmos há diferença, executamos o teste Post-Hoc Nemenyi, que obteve o resultado apresentado pela Tabela 1. Novamente, considerando $\alpha = 0{,}05$, pode-se confirmar na análise pareada o comportamento esperado pelo gráfico de que o algoritmo RVEA difere de maneira significativa dos outros dois. Porém, não verificou-se diferença estatística nos resultados dos algoritmos NSGA-III e CTAEA para este experimento.

Tabela 1. Valor-P das comparações pareadas pelo teste de Nemenyi.

	NSGA-III	CTAEA
CTAEA	0,268429	*
RVEA	0,001000	0,001000

5.2. Mapas Gerados

Para ilustrar o conteúdo gerado, selecionamos das 90 execuções aquela execução que retornou o conjunto não dominado de maior valor de hipervolume. Este conjunto de soluções foi aquele encontrado pelo algoritmo NSGA-III (ponto de maior valor na Figura 2, visível em vermelho) que contou com 8 soluções não-dominadas que são descritas pela

Tabela 2, em que as variáveis de decisão $x_1 \ldots x_{10}$ representam em ordem as variáveis definidas na Seção 3.2. A ilustração dos valores das quatro funções-objetivo para estas soluções está no gráfico de pétala da Figura 3, que apresenta valores normalizados.

Tabela 2. Variáveis de decisão e objetivos da solução de maior hipervolume.

	Variáveis de decisão										**Objetivos**			
Solução	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	f_1	f_2	f_3	f_4
1	0	13	728	32	300	0	2	1	1	0	1	−135,00	−872,98	−263820,00
2	0	12	187	29	300	0	−2	1	1	0	0	−47,00	−651,95	−290490,00
3	0	13	739	30	286	5	2	1	1	0	0	−198,00	−632,19	−249220,00
4	0	14	215	29	300	0	2	1	1	0	0	−64,00	−324,61	−289020,00
5	0	14	200	30	300	0	2	1	1	0	1	−64,00	−440,69	−289440,00
6	0	12	679	29	286	1	2	3	1	0	0	−140,00	−1036,38	−251560,00
7	0	12	594	32	298	3	3	1	1	0	0	−122,00	−580,49	−267670,00
8	0	13	734	29	300	0	2	1	1	0	2	−213,00	−468,51	−262260,00

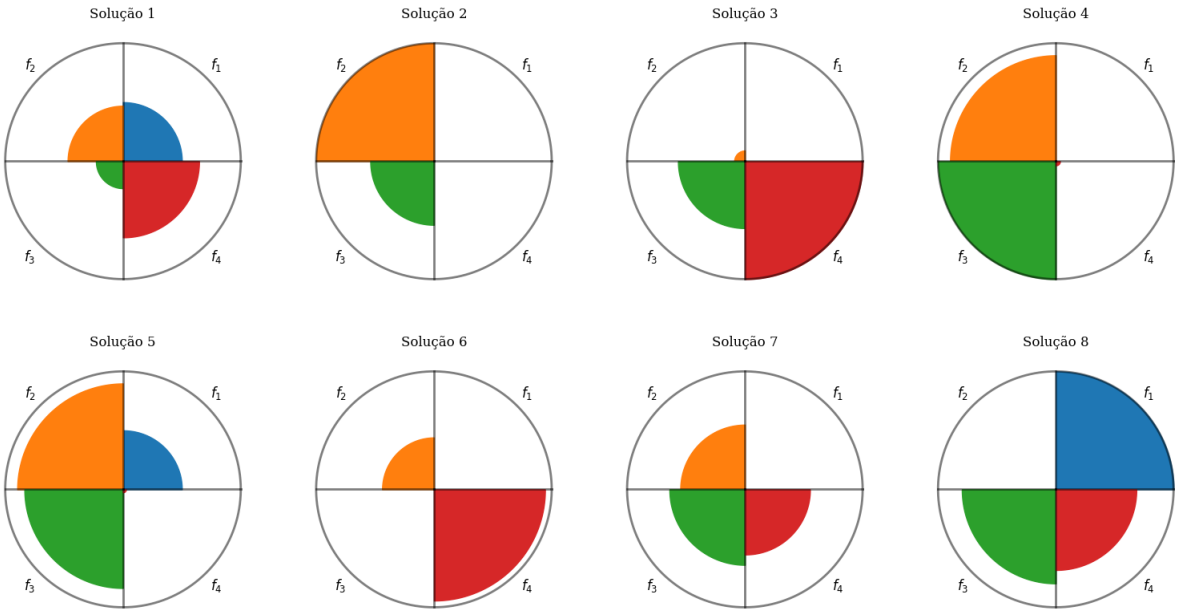

Figura 3. Gráfico de pétala dos objetivos dos indivíduos da execução de maior hipervolume.

Em uma análise qualitativa, é possível observar algumas características das soluções no gráfico de pétala. As soluções 2 e 4, por exemplo, tiveram um desempenho muito bom nos objetivos f_1 e f_4, ou seja, finalizaram as execuções recebendo poucos golpes e com bastante tempo sobrando. Entretanto, estas possuem desempenhos menores nos objetivos f_2 e f_3, ou seja, eliminando menos inimigos e efetuando menos pulos e/ou pulos menores. Isso indica fases possivelmente mais fáceis para o agente.

Já as soluções 1 e 7 tiveram desempenhos semelhantes entre os quatro objetivos, quando comparadas às outras soluções. Isso é um indicativo que estas fases trazem uma dificuldade mais balanceada.

Já na solução 8, apesar de um desempenho pior no objetivo f_3, que levaria a imaginar uma menor dificuldade, o agente sofreu dois golpes durante a execução. Este é um indicativo de dificuldade na fase.

Analisando também as variáveis de decisão, é possível verificar algumas semelhanças entre as soluções desta execução. As soluções 2, 4 e 5 que tem o menor comprimento de fase, tem também valores piores para o objetivo f_2, da mesma forma que as outras fases, mais compridas, tem valores melhores. Isso pode ser resultado do fato de fases maiores terem mais espaço para a geração de inimigos.

As Figuras 4, 5 e 6 representam os recortes dos mapas gerados pelos parâmetros das soluções 2, 7 e 8, respectivamente. É possível verificar que a seção do mapa 8 apresenta, de fato, vários inimigos, e que o mapa da solução 2, por ser menor, tende a apresentar uma quantidade total menor de inimigos.

6. Conclusões e Trabalhos Futuros

Este trabalho propôs a criação de um modelo de otimização *many-objective* para um gerador de fases procedurais de um jogo à escolha, bem como a otimização desse modelo comparando três algoritmos.

Os experimentos computacionais revelaram que os algoritmos NSGA-III e C-TAEA obtiveram bons resultados no cenário analisado e que são promissores para experimentos futuros no modelo proposto.

Para o conjunto não dominado selecionado do experimento, nossa análise da melhor solução encontrada mostrou tendências em alguns parâmetros, como $tipo\_mapa$, $dificuldade\_fase$, $altura\_fase$, $tempo\_total$, p_4 e p_5, que tiveram pouca variação entre seus valores nos indivíduos não dominados. Isso indica que os valores apresentados na Tabela 2 para estes parâmetros tem o potencial de obter bons resultados para o nosso modelo. Os outros parâmetros, como o $comprimento\_fase$, apresentaram nesta execução valores mais distribuídos dentro dos limites definidos, o que não demonstra uma similaridade clara entre estes parâmetros e o desempenho final destas soluções. De qualquer forma, uma análise estatística mais detalhada da distribuição destes parâmetros considerando outras execuções é necessária para maiores conclusões sobre a influência de cada parâmetro nos objetivos.

O uso de um modelo de otimização em geradores de conteúdo se mostrou promissor na geração de fases procedurais que atendam a múltiplos objetivos previamente definidos. Este trabalho, sendo um passo inicial, apresenta um modelo e os resultados para um gerador específico.

Ademais, ofereceu-se para a comunidade especializada, como avanços, um modelo *many-objective* que visa balancear dificuldade e diversidade e um método automático de sintonia de parâmetros de um gerador de fases procedurais para um jogo de plataforma escolhido, além da comparação de três algoritmos evolutivos para esta aplicação específica.

Em relação a limitações observadas e desafios a serem enfrentados, podemos citar

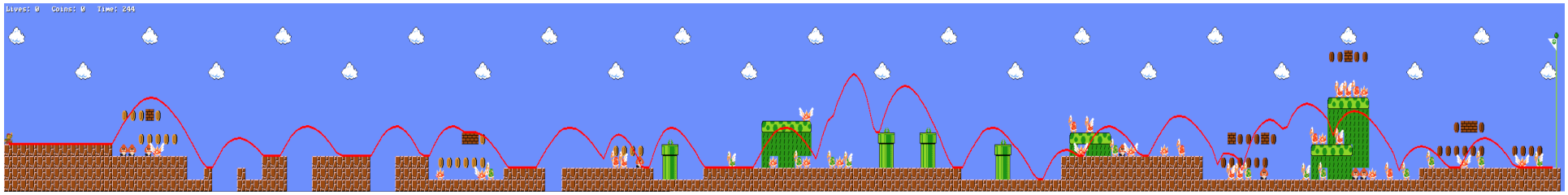

Figura 4. Mapa gerado pela solução 2 com o caminho percorrido pelo agente.

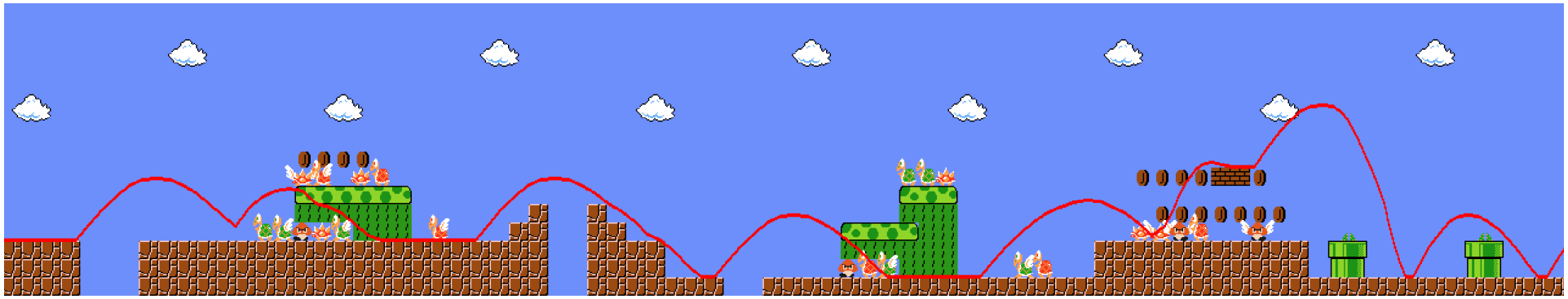

Figura 5. Seção do mapa gerado pela solução 7 com o caminho percorrido pelo agente.

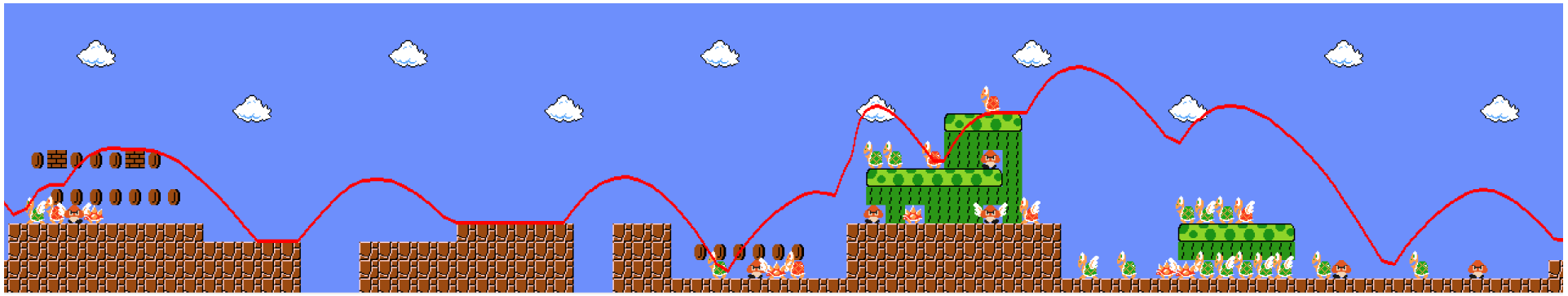

Figura 6. Seção do mapa gerado pela solução 8 com o caminho percorrido pelo agente.

o tamanho da população e o número de avaliações de objetivos utilizados nos experimentos, que p odem ser ampliados com o uso de maior poder computacional.

Além disso, não foi feita uma análise estatística individual dos parâmetros sintonizados, que pode trazer informações úteis para a obtenção de melhores resultados. Também não houve envolvimento humano na avaliação das fases geradas. A avaliação foi automática baseada apenas em parâmetros de uma Inteligência Artificial (IA), que pode ter mais facilidade em finalizar as fases do que humanos.

Por último, apenas uma IA foi utilizada na avaliação. O uso de mais de uma IA pode ajudar a mitigar possíveis vieses na avaliação do conteúdo.

Como trabalhos futuros, a execução de testes estatísticos nos resultados de todas as execuções, isolando cada variável de decisão, pode contribuir com a qualidade do modelo e fornecer com mais exatidão a informação de quais parâmetros devem ser alterados visando cada objetivo.

Como forma de avaliar os mapas gerados pelo gerador o modelo atual executa uma tentativa de finalização do mapa por um agente, e obtém dados pertinentes desta tentativa. Um único agente A* é utilizado para a avaliação, mas o Mario AI Framework possui outros agentes que podem ser utilizados também. Uma possível melhoria ao trabalho atual é a implementação de uma avaliação de conteúdo composta pela execução de mais de um agente, utilizando estratégias diferentes. Uma hipótese possível para essa abordagem é que a avaliação da qualidade dos conteúdos será mais confiável. É possível também que os conteúdos selecionados tenham mais diversidade.

Outra possibilidade é a de avaliar e otimizar os outros geradores do Mario AI Framework, que utilizam técnicas diferentes para geração de mapas. Dahlskog et al., 2014 propõe, por exemplo, um conjunto de métricas para comparar diferentes geradores de conteúdo procedural que foi aplicado nos geradores do Mario AI Framework. Entretanto, para otimizar os outros geradores seria necessário expandir o modelo para contemplá-los, uma vez que o modelo atual foi criado em cima de características específicas do gerador inicial do jogo Infinite Mario Bros.

Como complemento à avaliação de conteúdo, o Mario AI Framework ainda possui outros agentes que poderiam ser utilizados em complemento ao A* utilizado atualmente para avaliação dos conteúdos [Togelius et al., 2013]. O agente “andySloane” é uma implementação de um agente baseada também no algoritmo A*, mas com pequenas diferenças de design e performance em relação ao A* citado neste trabalho. Já os agentes “glenHartmann”, “spencerSchumann” e “trondEllingsen” utilizam heurísticas codificadas manualmente baseadas em regras específicas, e o agente “michal” foi desenvolvido com base em uma máquina de estados finita, Já os agentes “sergeyPolikarpov” e “sergeyKarakovskiy” são baseados em redes neurais, a primeira utilizando Aprendizado por Reforço, e a segunda usando Redes Neurais Recorrentes.

Além destas, a aplicação de pesquisas de opinião com humanos pode ajudar a avaliar os conteúdos gerados de maneira mais próxima aos jogadores, que são os alvos principais da geração de conteúdo.

Referências

Blank, J., & Deb, K. (2020). pymoo: Multi-Objective Optimization in Python. *IEEE Access*, *8*, 89497–89509.

Cheng, R., Jin, Y., Olhofer, M., & Sendhoff, B. (2016). A Reference Vector Guided Evolutionary Algorithm for Many-Objective Optimization. *IEEE Transactions on Evolutionary Computation*, *20*(5), 773–791. https://doi.org/10.1109/TEVC.2016.2519378

Coello Coello, C. A., & Reyes Sierra, M. (2004). A Study of the Parallelization of a Coevolutionary Multi-objective Evolutionary Algorithm. Em R. Monroy, G. Arroyo-Figueroa, L. E. Sucar & H. Sossa (Ed.), *MICAI 2004: Advances in Artificial Intelligence* (pp. 688–697). Springer Berlin Heidelberg.

Compton, K., & Mateas, M. (2021). Procedural Level Design for Platform Games. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, *2*(1), 109–111. https://ojs.aaai.org/index.php/AIIDE/article/view/18755

Dahlskog, S., Horn, B., Shaker, N., Smith, G., & Togelius, J. (2014). A Comparative Evaluation of Procedural Level Generators in the Mario AI Framework.

Deb, K., & Jain, H. (2014). An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. *IEEE Transactions on Evolutionary Computation*, *18*(4), 577–601. https://doi.org/10.1109/TEVC.2013.2281535

Deb, K., Sindhya, K., & Okabe, T. (2007). Self-Adaptive Simulated Binary Crossover for Real-Parameter Optimization. *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, 1187–1194. https://doi.org/10.1145/1276958.1277190

Derrac, J., García, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, *1*(1), 3–18. https://doi.org/https://doi.org/10.1016/j.swevo.2011.02.002

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., & Gagné, C. (2012). DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, *13*, 2171–2175.

Friedman, M. (1937). The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Journal of the American Statistical Association*, *32*(200), 675–701. https://doi.org/10.1080/01621459.1937.10503522

Iosup, A. (2009). POGGI: Puzzle-Based Online Games on Grid Infrastructures. Em H. Sips, D. Epema & H.-X. Lin (Ed.), *Euro-Par 2009 Parallel Processing* (pp. 390–403). Springer Berlin Heidelberg.

Ishibuchi, H., Tsukamoto, N., & Nojima, Y. (2008). Evolutionary many-objective optimization: A short review. *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 2419–2426. https://doi.org/10.1109/CEC.2008.4631121

Karakovskiy, S., & Togelius, J. (2012). The Mario AI Benchmark and Competitions. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(1), 55–67. https://doi.org/10.1109/TCIAIG.2012.2188528

Kerssemakers, M., Tuxen, J., Togelius, J., & Yannakakis, G. N. (2012). A procedural procedural level generator generator. *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, 335–341.

Kivy Team. (2021). Pyjnius (1.4.1).

Li, K., Chen, R., Fu, G., & Yao, X. (2019). Two-Archive Evolutionary Algorithm for Constrained Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation*, *23*(2), 303–315. https://doi.org/10.1109/TEVC.2018.2855411

Nemenyi, P. (1963). *Distribution-free Multiple Comparisons*. Princeton University. https://books.google.com.br/books?id=nhDMtgAACAAJ

Pontes, R. G. d., & Gomes, H. M. (2020). Evolutionary Procedural Content Generation for an Endless Platform Game. *2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 80–89. https://doi.org/10.1109/SBGames51465.2020.00021

Summerville, A., Snodgrass, S., Guzdial, M., Holmgård, C., Hoover, A. K., Isaksen, A., Nealen, A., & Togelius, J. (2018). Procedural Content Generation via Machine Learning (PCGML). *IEEE Transactions on Games*, *10*(3), 257–270. https://doi.org/10.1109/TG.2018.2846639

Togelius, J., Karakovskiy, S., & Baumgarten, R. (2010). The 2009 Mario AI Competition, 1–8. https://doi.org/10.1109/CEC.2010.5586133

Togelius, J., Kastbjerg, E., Schedl, D., & Yannakakis, G. N. (2011). What is Procedural Content Generation? Mario on the Borderline. *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. https://doi.org/10.1145/2000919.2000922

Togelius, J., Shaker, N., Karakovskiy, S., & Yannakakis, G. N. (2013). The Mario AI Championship 2009-2012. *AI Magazine*, *34*(3), 89–92. https://doi.org/10.1609/aimag.v34i3.2492

van der Linden, R., Lopes, R., & Bidarra, R. (2014). Procedural Generation of Dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*, *6*(1), 78–89. https://doi.org/10.1109/TCIAIG.2013.2290371

Van Veldhuizen, D. A. (1999). *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations* (tese de dout.) [AAI9928483]. USA, Air Force Institute of Technology.

Centro Federal de Educação Tecnológica de Minas Gerais
Curso de Engenharia de Computação
Avaliação do Trabalho de Conclusão de Curso

Aluno: Vitor Gomes Soares Lins Peixoto
Título do trabalho: Criação automática de fases de jogos digitais via sintonização de parâmetros de um gerador procedural através de um modelo de otimização com muitos objetivos
Data da defesa: 06/12/2022 Horário: 09:00
Local da defesa: Remoto, https://conferenciaweb.rnp.br/webconf/andre-rodrigues-da-cruz

O presente Trabalho de Conclusão de Curso foi avaliado pela seguinte banca:

Professor André Rodrigues da Cruz – Orientador
Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais

Professora Elizabeth Fialho Wanner – Co-orientadora
Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais

Flávio Roberto dos Santos Coutinho – Membro da banca de avaliação
Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais

Elisângela Martins de Sá – Membro da banca de avaliação
Departamento de Ciências Sociais Aplicadas
Centro Federal de Educação Tecnológica de Minas Gerais