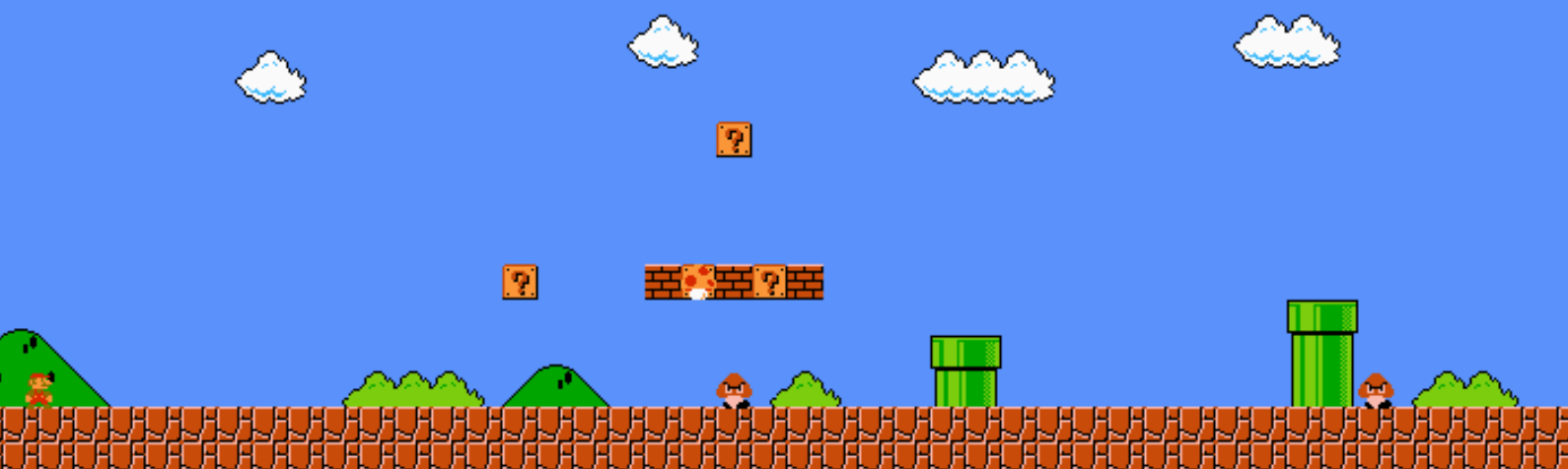
UNIVERSIDADE FEDERAL DE GOIÁS / INSTITUTO DE INFORMÁTICA

Procedural Content Generation Guiado por Prompt

Desenvolvimento de Datasets e Fine-Tuning de Modelos para Jogos

Edward Scott Carvalho Johnson

UFG
UNIVERSIDADE FEDERAL DE GOIÁS



UNIVERSIDADE FEDERAL DE GOIÁS (UFG)
INSTITUTO DE INFORMÁTICA (INF)

EDWARD SCOTT CARVALHO JOHNSON

Procedural Content Generation Guiado por Prompt

Desenvolvimento de Datasets e Fine-Tuning de Modelos para Jogos

Goiânia
2025

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC no 1240/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei no 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo dos Trabalhos de Conclusão dos Cursos de Graduação disponibilizado no RI/UFG é de responsabilidade exclusiva dos autores. Ao encaminhar(em) o produto final, o(s) autor(a)(es)(as) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG)

Nome(s) completo(s) do(a)(s) autor(a)(es)(as): EDWARD SCOTT CARVALHO JOHNSON

Título do trabalho: Procedural Content Generation Guiado por Prompt

Desenvolvimento de Datasets e Fine-Tuning de Modelos para Jogos

2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante: a) consulta ao(à)(s) autor(a)(es)(as) e ao(à) orientador(a); b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo do TCCG. O documento não será disponibilizado durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;

- Submissão de artigo em revista científica;

- Publicação como capítulo de livro.

Obs.: Este termo deve ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Edward Scott Carvalho Johnson**, **Discente**, em 04/02/2026, às 16:53, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Fernando Marques Federson**, **Professor do Magistério Superior**, em 13/03/2026, às 11:28, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

EDWARD SCOTT CARVALHO JOHNSON

Procedural Content Generation Guiado por Prompt

Desenvolvimento de Datasets e Fine-Tuning de Modelos para Jogos

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.

Orientador: Prof. Dr. Fernando Marques Federson

Goiânia

2025

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UFG.

JOHNSON, EDWARD SCOTT CARVALHO
Procedural Content Generation Guiado por Prompt [manuscrito]: Desenvolvimento de Datasets e Fine-Tuning de Modelos para Jogos / EDWARD SCOTT CARVALHO JOHNSON. - 2025.
113 f.: 2025

Orientador: Prof. Dr. Fernando Marques Federson
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Instituto de Informática (INF), Inteligência Artificial, Goiânia, 2025.

1. Inteligência Artificial. 2. Procedural Contents Generation. 3. Jogos.

I. Federson, Fernando Marques , orient. II. Título.

CDU 004

EDWARD SCOTT CARVALHO JOHNSON

Procedural Content Generation Guiado por Prompt
Desenvolvimento de Datasets e Fine-Tuning de Modelos para Jogos

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.

Data da Aprovação: 09 de dezembro de 2025.

Prof. Dr. Fernando Marques Federson
Orientador (INF-UFG)

Prof. Dr. Aldo André Díaz Salazar
Coordenador de TCC do BIA (INF-UFG)

Prof. Dr. Anderson da Silva Soares
Coordenador do BIA (INF-UFG)

Prof. Dr. Sávio Salvarino Teles de Oliveira
(INF-UFG)

EDWARD SCOTT CARVALHO JOHNSON

Procedural Content Generation Guiado por Prompt

Desenvolvimento de Datasets e Fine-Tuning de Modelos para Jogos

RESUMO

Este Relatório de Conclusão de Curso tem como objetivo reunir os resultados da minha jornada para me tornar um especialista em **Promotable Procedural Content Generation**. Uma ilustração e sua narrativa descrevem os períodos de trabalho. Os Apêndices contêm os Termos de Aceite de Entrega e os resultados obtidos durante cada período de trabalho.

Palavras-chave: Inteligência artificial; Procedural contents generation; Jogos.

ABSTRACT

This Course Completion Report aims to bring together the results of my journey to become an expert in **Promotable Procedural Content Generation**. An illustration and its narrative describe the work periods. The Appendices contain the Delivery Acceptance Terms and the results obtained during each work period.

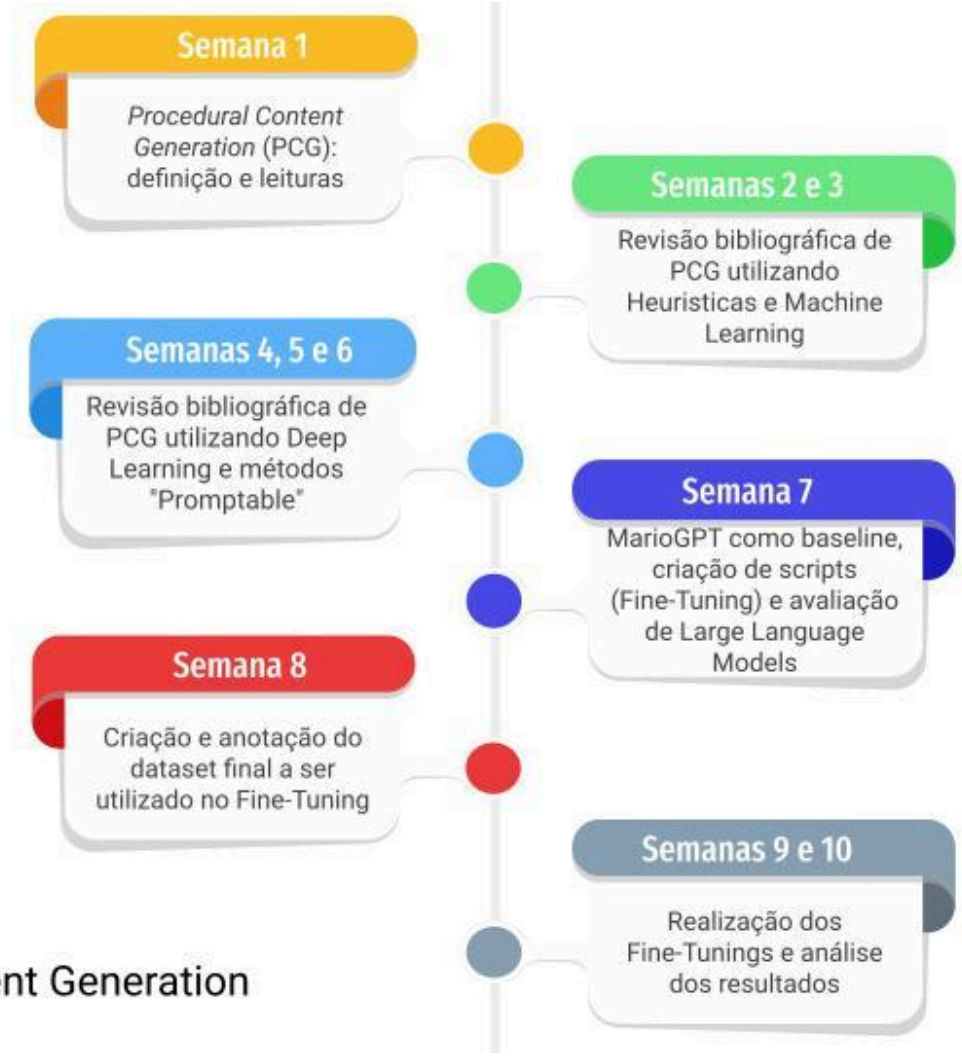
Keywords: Artificial intelligence; Procedural content generation; Games.

Goiânia

2025

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Minha Jornada



Edward Scott Carvalho Johnson
Especialista em: Promptable Procedural Content Generation

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

MINHA JORNADA

Nome: **Edward Scott Carvalho Johnson**

Especialidade: **Promptable Procedural Content Generation**

Objetivo deste documento

Durante o processo da disciplina Residência em IA[1], foram gerados diversos resultados na construção da minha especialização. A cada semana, um conjunto de resultados foi formalizado por um Termo de Aceite de Entrega e avaliado por uma banca, considerando o planejado e o realizado para o período. Este documento tem como objetivo descrever esses resultados obtidos, fazendo referência aos Termos de Aceite de Entrega e seus documentos associados.

Minha Jornada

Minha Jornada teve início na **Semana 1**, momento em que estabeleci as primeiras bases conceituais para compreender o vasto conjunto de possibilidades dentro do campo de Procedural Content Generation (PCG). Foi feito um estudo inicial e um levantamento estruturado sobre as origens do PCG e sobre como os métodos clássicos surgiram, se consolidaram e influenciaram a pesquisa atual. As leituras iniciais, reunidas no **Apêndice 1**, permitiram que eu tivesse uma visão histórica sólida antes de avançar para abordagens mais recentes. Elementos como algoritmos construtivos, métodos evolutivos tradicionais e estratégias simples de geração foram necessários para que eu reconhecesse como o campo evoluiu até chegar à aplicação de aprendizado de máquina. Esse entendimento inicial formou a base necessária para que as semanas seguintes ganhassem direção mais clara.

O avanço ocorrido na **Semana 2** foi a leitura de uma combinação de dois livros e três surveys. Os surveys, cuidadosamente resumidos como descrito no **Apêndice 1**, abordavam desde PCG orientado a busca, passando por PCG guiado pela experiência do jogador, até

[1] Dez Semanas, entre setembro de 2025 e dezembro de 2025.

PCG baseado em machine learning, incluindo métodos contemporâneos como redes recorrentes e abordagens inspiradas em deep learning. Paralelamente, selecionei partes relevantes dos livros que tratavam de representações, heurísticas, fundamentos da pesquisa e estratégias iniciais de modelagem. Ao reunir esse conjunto de leituras, percebi que muitas das discussões estruturais do campo não surgem de IA pura, mas de escolhas de design, representação e avaliação. Essa etapa me permitiu identificar lacunas e também enxergar possibilidades futuras dentro do PCG moderno, o que reforçou ainda mais a relevância do material reunido no **Apêndice 1**.

Com esse embasamento, a **Semana 3** foi dedicada a uma análise mais técnica das principais abordagens que unem IA e PCG, aprofundando leituras e anotações a partir dos capítulos 2, 7 e 8 do livro *Artificial Intelligence and Games*. Essa leitura, detalhada no **Apêndice 2**, foi essencial para delimitar tecnicamente o cenário de geração procedural moderna. Os capítulos apresentam de conceitos básicos de IA até um panorama detalhado dos métodos contemporâneos de PCG, passando por algoritmos construtivos, search based, quality diversity, solver based, machine learning, redes recorrentes, redes generativas adversariais e até métodos baseados em aprendizado por reforço. O estudo desses capítulos evidenciou a importância da escolha de representações, alguns *pipelines* de geração e os desafios na compatibilização entre coerência global e detalhes locais. Assim, defini que eu iria me aprofundar nas técnicas baseadas em machine learning.

Na **Semana 4**, me direcionei para a leitura de trabalhos envolvendo deep learning aplicado ao PCG, mais especificamente GANs e LSTMs, além de um estudo detalhado sobre o dataset VGLC. Durante esse período, descrito no **Apêndice 3**, percebi como a área tem lidado com a escassez de dados e como os modelos disponíveis tentam superar limitações estruturais. Trabalhos envolvendo GANs descreveram comportamentos interessantes em relação à coerência espacial, enquanto abordagens baseadas em LSTMs mostraram vantagens na geração de padrões sequenciais com dependências mais ricas. A leitura desses trabalhos permitiu uma visão mais integrada do problema, destacando como diferentes arquiteturas tentam equilibrar fidelidade, diversidade e preservação das características essenciais dos jogos originais. Ao observar como o dataset VGLC é utilizado, ficou evidente que a comunidade científica busca constantemente maneiras de adaptar ou

expandir o conjunto de dados para maximizar a qualidade das fases geradas. Após me aprofundar nessa área, decidi direcionar meus esforços para métodos modernos de PCG guiados por prompt, iniciando uma mudança de perspectiva que se tornaria central na **Semana** seguinte.

Durante a **Semana 5**, entrei definitivamente no território dos métodos “promptable” de PCG, ampliando o olhar para abordagens baseadas em LLMs e modelos de difusão. Trabalhos recentes, discutidos no **Apêndice 4**, mostraram ambientes nos quais o texto do usuário se torna o condutor do processo de criação, permitindo que modelos gerativos produzam fases completas, layouts, regras e até estruturas inteiras de jogos. Ao analisar essas abordagens, foi possível concluir que a maior dificuldade não está apenas em gerar conteúdo coerente, mas sim em garantir alinhamento com a intenção expressa no prompt. Essa é uma das principais diferenças entre métodos puramente numéricos e aqueles que buscam transformar linguagem em estrutura jogável. As leituras dessa etapa demonstram como modelos grandes lidam melhor com composição, coerência e organização espacial do que modelos de difusão, embora estes últimos entreguem maior refinamento visual quando aplicados corretamente.

A **Semana 6** representou um período de transição entre teoria e os experimentos. Além de aprofundar a leitura de métricas e avaliar métodos de diferentes trabalhos, também consolidei diversos conhecimentos adquiridos nos trabalhos anteriores em um conjunto organizado de sínteses, registradas no **Apêndice 5**. O estudo sistemático das métricas de PCG foi relevante, pois caracterizou o que se espera de um conteúdo gerado: jogabilidade, diversidade estrutural, aderência ao prompt e elementos de novidade. Essas métricas mostraram que, além de gerar conteúdo, o desafio está em garantir que esse conteúdo respeite critérios que tornam a experiência do jogador interessante e funcional. Ao analisar como autores de diferentes artigos aplicam suas métricas, percebi que a avaliação acaba moldando o próprio processo de geração. Paralelamente, a reflexão sobre como meu trabalho inicial na disciplina de Processamento de Linguagem Natural (PLN) do curso abriu espaço para reconhecer que muito do que eu já havia desenvolvido anteriormente poderia servir como ponto de partida para as próximas **Semanas** da Residência.

A partir desse ponto, a **Semana 7** foi dedicada à operacionalização de tudo aquilo que havia sido planejado. Coloquei em execução os códigos originais do modelo "MarioGPT", revisitei e rodei integralmente o código produzido durante meu trabalho de PLN e, em seguida, iniciei a organização sistemática do que já havia produzido. Esse processo, descrito com detalhes no **Apêndice 6**, envolveu a adaptação do dataset, a criação de novas representações, experimentos de métricas adicionais e a implementação de mecanismos de detecção de similaridade via análise *n gram* e *Jaccard*. Muitas das ideias presentes nos artigos recentes já estavam, em alguma medida, sendo trabalhadas de forma inicial. Além disso, pude testar representações mais detalhadas do dataset e ampliar a diversidade de exemplos já disponíveis para futuros treinamentos.

Assim, a **Semana 8** concentrou-se na tarefa demorada de anotar manualmente as fases faltantes do dataset. Essa etapa foi importante para gerar um conjunto de dados mais completo e permitir análises e experimentos mais robustos posteriormente. A **Semana 9** deu início à fase mais experimental do projeto. Nesse período, iniciei a criação de múltiplas variações de datasets para testar diferentes formas de representação dos dados, totalizando aproximadamente 60 variações. Também corrigi scripts utilizados nos processos de fine tuning e implementei os dois métodos de tokenização que seriam usados: tokens feitos via Byte Pair Encoding, e tokens feitos com cada caractere que ocorre no dataset. Os primeiros resultados, embora limitados, já apontavam possíveis caminhos de melhoria e algumas limitações. Os experimentos e registros dessas **Semanas** estão apresentados no **Apêndice 7**, contendo os resultados preliminares que me ajudaram a definir o que deveria ser lapidado na etapa final.

Ao chegar na **Semana 10**, concentrei meus esforços em analisar os melhores resultados obtidos dentro das limitações de tempo e hardware. Os ciclos de fine tuning, interrupções para correções e reinicializações contínuas acabaram consumindo grande parte considerável da **Semana**, reduzindo a quantidade de experimentos que consegui finalizar. Ainda assim, os resultados intermediários permitiram identificar diferenças importantes entre representações, comportamentos inesperados de alguns modelos e padrões que reforçam a necessidade de datasets mais robustos e processamentos mais eficientes. Mesmo sem resultados definitivos, foi possível observar melhorias em relação ao

baseline em algumas configurações específicas, além de identificar limitações estruturais que poderiam orientar estudos futuros. Os experimentos e registros desta Semana estão apresentados no **Apêndice 8.**

Ao fim de toda essa Jornada, reconheço que o estudo sobre Procedural Content Generation exige tanto rigor técnico quanto criatividade metodológica. A cada **Semana**, percebi que PCG não é apenas uma questão de gerar conteúdo, mas de compreender profundamente como sistemas interagem com suas representações, como jogadores experimentam o conteúdo e como modelos de IA interpretam instruções que não são explicitamente estruturadas. Por fim, eu gostaria de agradecer a todos os professores do bacharelado em Inteligência Artificial por formarem o profissional que virei, e em especial os professores Sávio Teles, Fernando Federson e Aldo Diaz. Também gostaria de agradecer os professores da banca examinadora Cedric Luiz e Leonardo Alves. Também gostaria de agradecer a terceira turma do bacharelado em Inteligência Artificial por serem grandes motivadores, colegas e principalmente amigos. Sem a ajuda deles eu não estaria terminando esse trabalho hoje. Em especial eu gostaria de agradecer a Letícia Lima Mendes, Daniel Fazzioni, Carlos Henrique, André Castro, Dayane Rodrigues, Pedro Rabelo e Michael Vinicius.

APÊNDICE 1

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 4 de set. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

EDWARD SCOTT CARVALHO JOHNSON

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Para a primeira Semana do processo da disciplina Residência em IA, foi feito:

- Estudo de interesse das áreas do CSCE 2025:
  - Data and knowledge representation
  - Neural networks and applications
- Definição do tema a ser abordado durante a residência
  - Procedural Content Generation (PCG)

- Pesquisa sobre as origens do PCG e métodos comumente aplicados
  - Estudo da história do PCG
  - Algoritmos de PCG "clássico"
  - Estudo sobre os autores mais importantes de PCG relacionado a inteligência artificial e as técnicas de IA mais usadas na área atualmente
    - Gate 01 - Pesquisa de PCG Tradicional e olhando PCG via ML

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Fazer um resumo sobre PCG com IA/ML partindo de um estudo profundo dos 5 papers mais citados de PCG, todos do autor Julian Togelius com o objetivo de entender os trabalhos mais marcantes da área.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

Documento de Apoio - Semana 1

Gate 01 - Pesquisa de PCG Tradicional e olhando PCG via ML

O que é PCG?

PCG (Procedural Content Generation), em sua definição mais abrangente, significa a geração de qualquer tipo de conteúdo por meio de um algoritmo em um computador, geralmente em jogos. O termo é usado quase exclusivamente em jogos pois mesmo com seu significado amplo, quando PCG é aplicado em outras áreas o nome muda.

- **Em Cinema e Efeitos Visuais (VFX):** O termo usado é **"proceduralismo"** ou **"fluxo de trabalho procedural"**. É usado para criar ambientes complexos, como florestas ou cidades, de forma algorítmica.
- **Em Arquitetura e Planejamento Urbano:** É conhecido como **"design paramétrico"** ou **"design generativo"**. Ajuda a criar e testar diferentes layouts de edifícios e cidades com base em um conjunto de regras e parâmetros.
- **Aumento de Dados em Machine Learning:** A técnica é chamada de **"aumento de dados" (data augmentation)**. Consiste em gerar dados sintéticos para treinar modelos de IA com mais eficiência, criando variações de dados existentes.
- **Inteligência Artificial:** Dado a definição, IA poderia ser considerada PCG.

Os exemplos acima servem para demonstrar que PCG é um termo dominado pela área de jogos, então vou me referenciar a PCG relacionado a jogos apenas como PCG, e se eu falar sobre qualquer outra área serei específico.

Para entender o PCG de forma conceitual, imagine dar ao computador uma receita para criar uma masmorra. Você define as regras (tamanho das salas, tipos de inimigos) e o algoritmo as aplica com um alguma aleatoriedade para gerar um resultado único, mas coerente. O processo pode começar com uma sala inicial, expandir com corredores e adicionar novas salas até que um layout funcional seja criado.

Um conceito central para o PCG que é necessário ter em mente é o uso de seeds, um valor inicial que direciona os algoritmos para gerar um resultado específico e replicável. A partir de um seed, o algoritmo gera um mundo de forma determinística. De forma análoga, seeds são usadas na inferência de IA para garantir que um modelo generativo produza uma saída consistente.

PCG dos anos 70 até hoje

O PCG não foi criado inicialmente como uma escolha de design, mas como uma necessidade técnica. Nos primórdios dos videogames (final dos anos 70 e início dos 80), a mídia de armazenamento, como disquetes, tinha uma capacidade extremamente limitada. Era impossível criar e armazenar grandes mundos de jogo manualmente.

A solução foi o PCG. Os primeiros desenvolvedores usaram algoritmos para gerar conteúdo de jogo dinamicamente, com o objetivo principal de compressão: usar um pequeno conjunto de regras e um seed para gerar enormes quantidades de conteúdo. Como discutido antês, o seed gera um mundo determinístico, então não é necessário carregar o mundo inteiro sendo gerado e salvar ou deixar carregado na RAM

Exemplos pioneiros desta época incluem ***Rogue* (1980)**, que deu origem ao gênero "roguelike" ao usar PCG para gerar uma nova masmorra a cada partida, e ***Elite* (1984)**, que gerou um vasto universo a partir de algumas regras simples, fazendo com que tudo coubesse em um disquete minúsculo.

Com o avanço da tecnologia, especialmente dos CD-ROMs, a necessidade de compressão diminuiu. No entanto, o PCG encontrou um novo propósito: a rejogabilidade. Jogos como ***Diablo* (1996)** popularizaram muitos elementos do gênero rogue-like para um público mais amplo, usando a geração procedural não para economizar espaço, mas para garantir que cada nova partida fosse diferente, com layouts de masmorras, monstros e itens aleatórios. Hoje, o gênero roguelike é extremamente popular, com muitos jogos de sucesso como *Hades*, *Slay the Spire* e *Dead Cells* baseando sua experiência principal na rejogabilidade fornecida pelo PCG.

A era moderna viu um ressurgimento do PCG, em grande parte devido ao sucesso de ***Minecraft* (2009)**, que demonstrou como a geração procedural poderia ser o pilar central de um jogo para criar mundos virtualmente infinitos e exploráveis.

Como PCG funciona (Tradicional)

A geração com PCG pode ser de terreno, objetos, padrões, etc. O objetivo de usar PCG é criar experiências de jogo completamente novas e empolgantes cada vez que o usuário joga.

Também não há muita literatura sobre essas técnicas de PCG em si. A literatura que existe geralmente é sobre as técnicas que foram adotadas pelos desenvolvedores, por exemplo, o paper "An Image Synthesizer" de Ken Perlin o qual introduz o “Ruído Perlin”, e “A note on two problems in connexion with graphs” de E. W. Dijkstra sobre o algoritmo de roteamento

do próprio. Alguns trabalhos são tão antigos que não é fácil encontrar as se tem um trabalho específico de origem, como:

- Os trabalhos de Andrey Markov's em 1906 sobre cadeias de Markov
- Stanislaw Ulam em 1940s criou o conceito de cellular automata

Os trabalhos mais antigos focados em PCG em jogos começam por volta de 2010, principalmente no Workshop Internacional de Procedural Content Generation, onde um breve histórico pode ser visto no paper de Antonios Liapis**.

O PCG em jogos não é feito com a mesma técnica, e nem utilizam de uma só técnica para tudo. Essas são as técnicas mais comumente utilizadas, uma conclusão na qual foi obtida pesquisando técnicas usadas nos jogos mais famosos que contém PCG, e no livro "Procedural Content Generation in Games".

Ruído Perlin / Simplex / OpenSimplex

- Explicação: São funções de ruído que produzem variação suave e natural em vez de aleatoriedade irregular. Elas funcionam atribuindo gradientes em pontos de uma grade e interpolando suavemente entre eles, o que cria características onduladas em vez de picos. A combinação de múltiplas "oitavas" (movimento Browniano fractal) adiciona detalhes finos sobre uma estrutura de grande escala.
- Caso de uso: Terrenos, nuvens, texturas.
- Exemplo: Mapa de altura (heightmap) 2D de terreno onde pontos próximos têm valores semelhantes (colinas onduladas). Um exemplo mais recente do seu uso é em "No Man's Sky" onde é utilizado "Uber Noise", que é a combinação de vários tipos de noise junto para criar um mundo 3D interessante.

Sistemas de Lindenmayer (Sistemas-L)

- Explicação: Um Sistema-L começa com uma string base (axioma) e a reescreve passo a passo usando regras. A string final é interpretada como comandos (como 'desenhar para frente' ou 'virar') para criar estruturas geométricas, tornando-o ideal para modelar padrões de crescimento e ramificação.
- Caso de uso: Plantas, cidades, rios.
- Exemplo: Axioma A, regras A→AB, B→A → interpretado como comandos de avançar/virar para criar caminhos ramificados.

FIGURE 26.5 Weeds generated by drawing instructions generated by L-systems.

Cadeias de Markov

- Explicação: Uma cadeia de Markov é um sistema onde o próximo estado depende apenas do atual, com probabilidades atribuídas a cada transição possível. Uma sequência é gerada "rolando" repetidamente essas probabilidades, o que produz resultados com padrões locais realistas sem a necessidade de uma simulação completa.
- Caso de uso: Clima, texto, nomes, música.
- Exemplo: Dois estados (Ensolarado, Chuvoso) com probabilidades; gere o clima para uma semana.

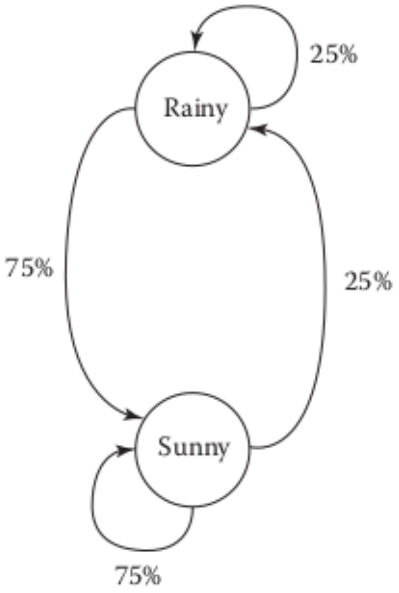


FIGURE 26.6 Simple Markov chain sequence generating a sequence of rainy and sunny days.

Passeios Aleatórios (1D/2D; com remoção de loop/Wilson)

- Explicação: Um passeio aleatório é um caminho formado por passos aleatórios a cada movimento. Em uma dimensão, cria linhas irregulares; em duas dimensões, caminhos sinuosos. A versão com remoção de loops (algoritmo de Wilson) apaga os loops à medida que se formam, garantindo que o resultado seja uma árvore que pode ser usada como um labirinto.
- Caso de uso: Rios, estradas, cavernas, labirintos.
- Exemplo: Caminho sinuoso 2D formando um rio; algoritmo de Wilson apagando loops para geração de labirintos.

FIGURE 26.10 A 2D random walk after hundreds of steps.

Autômatos Celulares

- Explicação: Um autômato celular opera em uma grade de células, onde cada célula possui um estado (por exemplo, 'parede' ou 'aberto'). Para cada célula, define-se uma vizinhança (as células adjacentes que importam, como as 4 ou 8 mais próximas). Em seguida, cria-se uma tabela de regras que determina o próximo estado de uma célula com base em seu estado atual e na contagem dos estados de seus vizinhos. Todas as células na grade são atualizadas simultaneamente, fazendo com que padrões em grande escala surjam a partir de regras locais simples.
- Caso de uso: Cavernas, propagação de fogo, crescimento de florestas.

- Exemplo: 55% de paredes aleatórias, regras que alteram as células → câmaras de caverna suaves após algumas iterações.

Partição Binária do Espaço (BSP)

- Explicação: A BSP divide um espaço grande em sub-retângulos menores por meio de divisão recursiva. Ao adicionar portais em cada divisão, a conectividade é preservada, facilitando a transformação posterior dessas sub-regiões em salas e corredores. Isso garante que os layouts sejam variados e navegáveis.
- Caso de uso: Geração de masmorras, layouts de salas.
- Exemplo: Divida um retângulo recursivamente, adicione portais para conectividade, posicione as salas.

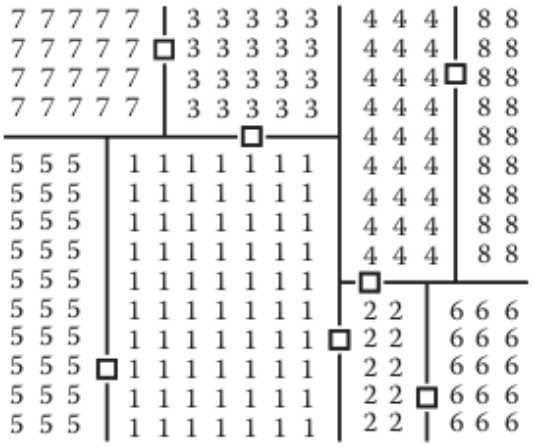


FIGURE 26.17 Step 4: After a third round of partition, the area starts to resemble a set of connected rooms.

Diagramas de Voronoi

- Explicação: Um diagrama de Voronoi atribui cada ponto no espaço à seed mais próxima, produzindo regiões ao redor dessas sementes. O resultado é uma tesselação natural do espaço em células que podem representar posse ou influência. Diferentes métricas e pesos alteram a forma das regiões.
- Caso de uso: Biomas, territórios, campos.

- Exemplo: 5 aldeias em um mapa → cada uma obtém fronteiras naturais através da partição pelo ponto mais próximo.

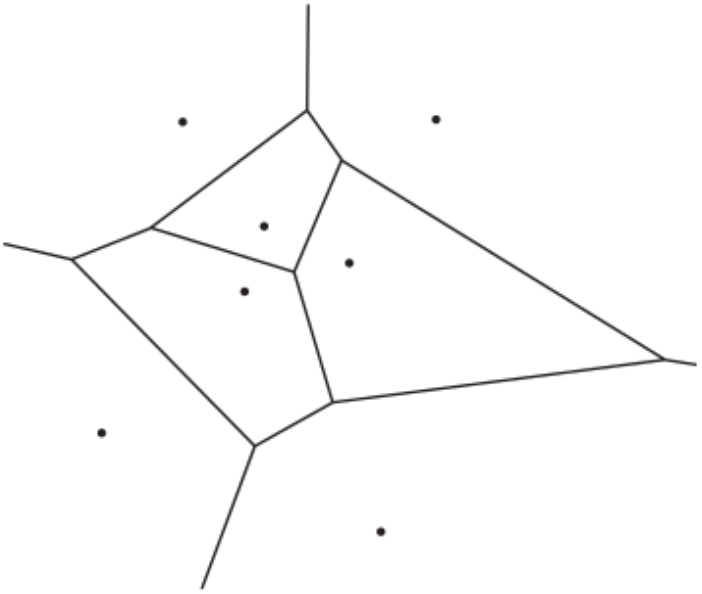


FIGURE 26.19 Simple Voronoi diagram.

Mapas de Dijkstra

- Explicação: Um mapa de Dijkstra 'inunda' uma grade ou grafo a partir de um ou mais pontos de partida, rotulando cada célula com sua distância até a fonte mais próxima. Isso fornece um equivalente discreto das regiões de Voronoi, ao mesmo tempo que codifica gradientes de distância. O resultado pode guiar o posicionamento, balanceamento ou navegação.
- Caso de uso: Busca de caminhos (pathfinding), posicionamento de encontros, análise de balanceamento.
- Exemplo: Preenchimento de distâncias a partir do início do jogador; chefe na célula mais distante, tesouros espalhados uniformemente.

FIGURE 26.20 A 20-seed Dijkstra map regionalization of a cave system in *Caves of Qud* used for terrain analysis and to create an evenly distributed population.

É importante ressaltar que quase nenhum jogo faz a geração completa de uma vez, ela faz em vários passes. Por exemplo, o minecraft primeiro define os biomas, depois gera o terreno, para depois colocar árvores e água, depois villas, e por assim vai.

Estudo inicial de PCG com IA

Não foi feita uma leitura de papers específicos, porém foi feito um estudo raso sobre os pesquisadores mais importantes das áreas e seus papers principais. Os pesquisadores mais influentes da área são:

- Julian Togelius (26612 citações)
- Ahmed Khalifa (3048 citações)
- Matthew Guzdial (2010 citações)
- Adam James Summerville (1983 citações)
- Sam Snodgrass (1863 citações)

É importante frisar que os 5 papers mais famosos e citados de PCG todos possuem autoria do Julian Togelius.

Foi possível notar que a maioria das técnicas utilizadas foram:

- LSTM's

- Variational Autoencoders
- Generative Adversarial Network
- Diffusion
- Reinforcement Learning

E agora está começando a ter um crescimento no uso de transformers/LLMs.

Adendos e referências:

Minha pesquisa foi feita principalmente sobre o livro “Procedural Content Generation in Game Design” (https://doi.org/10.1201/9781315156378)

Também utilizei palestras da Game Developers Conference (GDC), a maior conferência do mundo onde os maiores desenvolvedores e empresas de jogos se reúnem para compartilhar conhecimento técnico sobre jogos eletrônicos:
https://www.youtube.com/watch?v=C9RyEiEzMiU
https://www.youtube.com/watch?v=WumyfLEa6bU
https://www.youtube.com/watch?v=jV-DZqdKlnE

**: Liapis, A. (2020). 10 Years of the PCG workshop: Past and Future Trends. *Proceedings of the 15th International Conference on the Foundations of Digital Games*. https://doi.org/10.1145/3402942.3409598.

Também deixo um vídeo do jogo “Terraria” criando o mundo de jogo utilizando várias técnicas de PCG em vários “passes”.
What Actually Happens when Terraria creates Worlds!

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 2 de set. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

EDWARD SCOTT CARVALHO JOHNSON

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Para a primeira Semana do processo da disciplina Residência em IA, foi feito:

- A realização de que os 5 papers mais citados de PCG, todos de autoria do pesquisador Julian Togelius na verdade são 2 livros e 3 surveys, onde:
  - Um livro é sobre PCG com algoritmos no geral (Procedural Content Generation in Games), e o outro é sobre Jogos e Inteligência Artificial com um grande foco em PCG (Artificial Intelligence and Games)
  - Os três papers na verdade são surveys, onde os surveys tratam sobre:
    - PCG feito com algoritmos de busca/evolutivos
    - PCG Experience Driven (PCG que se adapta a feedback do jogador)
    - PCG com machine learning (entra em deep learning também)

- Foi feito um resumo para cada um dos três surveys com uma abordagem que visa me ajudar quando eu preciso lembrar de um certo trabalho específico sobre um determinado tema ou sobre alguma terminologia.
  - Search-based Procedural Content Generation (Survey tem 14 páginas)
  - Experience-Driven Procedural Content Generation (Survey tem 14 páginas)
  - PCGML (Survey tem 13 páginas)

- Também foi feito um resumo de alguns capítulos específicos do livro Procedural Content Generation in Games. Ainda que o livro inteiro consiga agregar conhecimento, dado a falta de tempo, não foi possível fazer uma leitura e estudo profundo do livro em uma semana. O livro no total tem 12 capítulos e foi feito o resumo de 5
  - Procedural Content Generation in Games - Textbook

- É importante notar que o único desses 4 conteúdos que tratam sobre IA diretamente é o survey PCGML. Porém, os outros conteúdos continuam sendo importantes dado alguns fundamentos que são necessários para conseguir fazer a modelagem dos modelos de ML/IA, como representações do jogo, métricas de avaliação e fundamentos de métodos de RL com PCG

- A leitura do livro Artificial Intelligence and Games é importante devido a riqueza de conteúdo, os pesquisadores nos quais contribuíram para as porções de PCG e sua recência (Novembro de 2024). O livro tem 550 páginas, porém muitas partes que não envolvem PCG diretamente (por exemplo, usar RL para treinar um modelo para jogar um jogo) podem ser retiradas diminuindo o livro para ~300 páginas.

Planejamento: **[descrever o que pretende fazer para realizar a próxima ENTREGA]**

- Ler a seção 3 do livro (capítulos 7, 8 e 9) que tem 75 páginas e anotar informações importantes. Decidir qual conteúdo quero aprofundar mais (não como tema final, mas como uma progressão).

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Documento de Apoio 1 - Semana 2

Search-based procedural content generation (SBPCG) applies evolutionary, stochastic, and metaheuristic search to generate game content. Motivations are

- **Efficiency:** PCG compresses memory use (e.g., *Elite*'s star systems).
- **Cost reduction:** Automates expensive manual design (e.g., *SpeedTree* in AAA games).
- **Novelty:** Enables endless, adaptive, and personalized content tuned to player style and experience.
- **Creativity support:** Algorithms can inspire human designers with fresh rulesets, levels, and narratives.

Categorizing PCG

PCG can be categorized along several axes:

- **Online vs. Offline:** Online generates content during play (fast, predictable runtime), while offline supports design (slower, editable output).
- **Necessary vs. Optional Content:** Necessary content (rules, dungeons, critical mechanics) must always be correct; optional content (weapons, decorations) can tolerate failure.
- **Random Seeds vs. Parameter Vectors:** Generators may use a seed for stochastic variety or explicit parameter vectors for designer control.
- **Stochastic vs. Deterministic:** Stochastic algorithms vary outputs; deterministic ones (e.g., *.kkrieger*, *Elite*) function like data compression.
- **Constructive vs. Generate-and-Test:** Constructive methods guarantee correctness during generation (e.g., fractals, Markov chains). Generate-and-test creates candidates, then filters them through evaluation (e.g., evolutionary algorithms).

SBPCG is a generate-and-test approach where a generator proposes content and an evaluation function assigns a numeric score (a single value or vector). New candidates are created guided by these scores to climb toward higher value. "Search-based" is broader than "evolutionary": it includes evolutionary algorithms but also simulated annealing, particle swarm optimization, and local search.

Core loop

Maintain a set of candidate artifacts → evaluate each → keep/improve the best

(mutation/recombination or other variation) → repeat. Regardless of the specific optimizer, two design choices dominate: how content is represented (which defines the search space and operators) and how it is evaluated (which defines “good”).

Representation and search space
Think genotype (what search manipulates) vs. phenotype (the realized content the evaluator sees).

- **Direct encodings:** genome parts map 1-to-1 to content parts (high locality; easy to reach any phenotype; can explode in dimensionality).
- **Indirect encodings:** a compact genome expands to content via a procedure (potentially small, expressive, but with mapping bias and variable locality).

Key properties: right dimensionality (avoid under/over-parameterization), **locality** (small genotypic edits → small phenotypic/fitness changes), and **reachability** (does the encoding cover interesting regions of phenotype space?).

Example: five ways to encode a maze
(1) grid of cells (direct; huge dimensionality, great locality)
(2) list of wall segments (shorter genome; decent locality if designed well)
(3) reusable motifs + placements (more compact; designer bias)
(4) target properties (rooms, doors, path lengths) with a decoder (very compact; locality depends on decoder)
(5) pure RNG seed (tiny genome; zero locality → search degenerates into randomness)

Evaluation functions
Three families, often mixed in practice:

- **Direct:** extract features from content and map to a score (fast; theory-driven handcrafting vs. data-driven learned mappings; supports personalization via player models).
- **Simulation-based:** let an agent play/solve; score from outcomes/telemetry. **Static** agents assume fixed behavior; **dynamic** agents learn during evaluation (captures learnability/fatigue but is costly).
- **Interactive:** measure with/through players. **Explicit** (surveys, pairwise preference) or **implicit** (usage, quit points, input intensity, physiology, gaze, expression). Powerful but noisy/interruptive; often practical in lab studies to train models used later.

Where SBPCG sits among PCG axes
SBPCG is generate-and-test and inherently **stochastic** (mutations, randomized steps). It

may accept **parameter vectors** (e.g., desired difficulty) that modulate evaluation/targets. There are no convergence or runtime guarantees; cost is dominated by evaluation (especially simulations), so SBPCG is often better offline though online use works for optional content or when evaluations are very fast.

Examples

The survey separates work on necessary content (must work for player progress) from optional content (can fail harmlessly or be skipped).

A. Necessary content

1) Rules and mechanics

Rules are the backbone of a game. Changing them mid-play is rare, so generators here must respect playability and balance. Search is attractive because rule spaces are combinatorial and hard to hand-tune; simulation-based evaluation (letting agents play) is common because “good rules” are hard to score directly.

- Hom & Marks [1] : Balanced two-player board-game rules in Zillions of Games (ZOG). Rules encoded as expression trees within a constrained family (Tic-Tac-Toe/Reversi/Checkers-like). Fitness: have the engine play both sides and minimize score imbalance (closer to draws = better). Static simulation, theory-driven notion of “fairness.”

- Togelius & Schmidhuber [2] : Evolved rules for grid-world arcade-style games. Genome is a fixed vector specifying collision effects and simple behaviors for colored entities. Fitness: an RL agent’s **learnability** of the rules (trivial or impossible rules score low; rules that become playable after learning score high). Dynamic simulation emphasizes “learnable, non-trivial” mechanics.

- Salge & Mahlmann [3] : Proposed an information-theoretic evaluator for mechanics: **relevant information** = minimum state information needed for optimal play. Approximate by evolving/learning a player and measuring mutual information between sensed state and actions. Low values correlate with degenerate designs (e.g., noisy, uninformative mechanics).

2) Puzzles

Puzzles must be solvable and target a challenge band (neither trivial nor impossible). Evaluators often wrap a solver and optimize for solutions with desirable properties (few clues, many deductions, set move counts).

- Oranchak [4] : GA for Shinro (8×8 logic puzzle). Direct matrix genome (holes/arrows). Fitness via a bespoke solver: prefer puzzles with a target number of moves and “nice” clue distributions; penalize over/under-constrained layouts.

- Ashlock [5] : Evolved two puzzle families: chess mazes (move like a chess piece through a maze) and chromatic grids. Direct encodings (piece lists or color grids). Fitness via dynamic programming solver; target specific solution lengths to hit desired difficulty.

3) Tracks and levels

Levels/tracks define spatial progression and difficulty pacing. Representations range from parametric (few numbers that expand to geometry) to part-based (design elements tiled under constraints). Evaluation is often simulation (bots playing) or direct features (gap sizes, linearity).

- Togelius et al. [6], [7] : Evolved racing tracks. Genome: spline parameters defining the midline; deterministic decoder. Fitness by driving a learned NN controller and scoring progress/speed smoothness; personalization comes from training the controller on the target player’s driving style, so evolved tracks suit that style.

- Pedersen et al. [8] : Personalized Super Mario Bros levels. Very indirect genome (few parameters for gap count/size/placement); stochastic decoder builds full levels. Fitness is a **data-driven** player-experience model (NNs predicting fun/challenge/frustration/etc.) from level features + player style features learned via web studies; optimize any combination (e.g., maximize fun, cap frustration).

- Sorenson & Pasquier [9] : Part-based, cross-genre level encoding using “design elements” (platforms/enemies/etc.) laid out in the genome for crossover locality. Two-stage evolution with FI-2Pop: first satisfy structural constraints (connectivity, quotas), then score difficulty with direct/weak simulation metrics to reward intermediate challenge.

- Jennings-Teats et al. [10] : Runtime difficulty adaptation for platformers (generate-and-test without full optimization). Pre-rate short segments by difficulty from player data; during play, assemble forward from a library guided by a set of “critics”

that accept/reject segments to meet the current difficulty target; rejected segments are re-rolled.

4) Terrains and maps

Maps affect pathing, balance, and strategy. Generators juggle aesthetics, accessibility, and symmetric fairness. Multiobjective search is common because objectives conflict (e.g., chokepoints vs. openness).

- Frade et al. [11] : Genetic programming terrain akin to CPPNs: expression trees queried at (x,y) produce height. Direct evaluator favors “accessibility”: maximize the largest connected nearly-flat region but bound it to avoid flat planes. Useful but sometimes visually unappealing → human screening required.

- Togelius et al. [12], [13] : RTS maps via multiobjective evolution. Two semi-direct encodings: (a) summed Gaussians for heightmaps; (b) StarCraft maps with turtle-graphics mountains + explicit base/resource coordinates. Objectives are gameplay-driven (A* path distances, resource fairness, chokepoint structure). SMS-EMOA yields a Pareto set; designers pick tradeoffs.

- Ashlock et al. [14] : L-system landscapes evolved to fit target silhouettes/shapes (indirect, fractal-style growth). Not evaluated for playability; demonstrates expressive shape control rather than game suitability.

- Mixed-initiative notes [15] : Integrations that let designers steer across multiple terrain methods (fractals, erosion, cellular automata), suggesting future hybrids with search.

B. Optional content

1) Weapons

Because players can ignore bad weapons, search can explore boldly. Implicit, in-the-wild feedback (usage) provides a scalable fitness signal.

- Hastings et al. (Galactic Arms Race) [16], [17], [18] : Online, distributed, interactive evolution of weapons. Genome encodes a neural particle-system controller; phenotype is the weapon’s visual/behavioral pattern. Fitness is **implicit usage** across players on a live server (fired often vs. left unused). Produces a moving ecology of player-preferred effects without interrupting play.

- Hastings et al. (Weapons Lab) [19] : Added direct gene editing so players can purposefully tweak weapon genomes they like. A hybrid of evolutionary search and user “genetic engineering,” earned as a gameplay privilege.

2) Buildings

Architectural content is rich and regular; interactive evolution leverages human aesthetic judgment and compact procedural encodings.

- Martin et al. [20] : Interactive evolution of buildings for Subversion. Indirect markup describes buildings as stacks of extruded 2D shapes with transform operators. Player picks two parents; system spawns 16 children via structural recombination + numeric mutation (an evolutionary-art style loop). Fast, designer-steerable exploration of a large space.

- Related CG foundations [21], [22], [23] : Shape grammars and component extraction form strong representations that future search-based systems can exploit for cities/buildings.

3) Trees

A test for exploring continuous design spaces with user guidance.

- Talton et al. [24] : Interactive exploration of a parametric 3D tree space. Models are fixed-length real vectors; the UI shows a 2D projection (via dimensionality reduction/density estimation) that users zoom/pan to browse and select interesting points. Conceptually akin to an EDA with interactive fitness, but framed as direct space navigation.

C. Cross-cutting

- **Direct vs. simulation vs. interactive evaluation**: most rules/puzzles use simulation (hard to score directly), levels/maps skew direct metrics, only a few works leverage data-driven player models or implicit online feedback.
- **Representations**: real-valued vectors and expression trees dominate; direct spatial matrices appear where locality and isomorphism help (grids/boards).
- **Objectives:** most optimize a single scalar or fixed linear mixes; multiobjective (e.g., RTS maps) is powerful when design goals conflict.

- **Gaps/opportunities:** more data-driven evaluators (player models, implicit telemetry), richer indirect encodings that preserve locality, and mixed-initiative loops that blend designer intent with search.

Referências

[1] V. Hom and J. Marks, “Automatic design of balanced board games,” in Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain., 2007, pp.25–30.
[2] J. Togelius and J. Schmidhuber, “An experiment in automatic game design,” in Proc. IEEE Symp. Comput. Intell. Games, 2008, pp. 111–118.
[3] C. Salge and T. Mahlmann, “Relevant information as a formalised approach to evaluate game mechanics,” in Proc. IEEE Conf. Comput. Intell. Games, 2010, pp. 281–288.
[4] D. Oranchak, “Evolutionary algorithm for generation of entertaining shinro logic puzzles,” in Proc. EvoAppl., 2010.
[5] D. Ashlock, “Automatic generation of game elements via evolution,” in Proc. IEEE Conf. Comput. Intell. Games, 2010, pp. 289–296.
[6] J. Togelius, R. De Nardi, and S. M. Lucas, “Making racing fun through player modeling and track evolution,” in Proc. SAB Workshop Adapt. Approach. Optimizing Player Satisfaction, 2006.
[7] J. Togelius, R. De Nardi, and S. M. Lucas, “Towards automatic person-alised content creation in racing games,” in Proc. IEEE Symp. Comput. Intell. Games, 2007, pp. 252–259.
[8] C. Pedersen, J. Togelius, and G. N. Yannakakis, “Modeling player experience in Super Mario Bros,” in Proc. IEEE Symp. Comput. Intell. Games., 2009, pp. 132–139.
[9] N. Sorenson and P. Pasquier, “Towards a generic framework for automated video game level creation,” in Proc. Eur. Conf. Appl. Evol. Comput., 2010, vol. 6024, pp. 130–139.
[10] M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin, “Polymorph: A model for dynamic level generation,” in Proc. Artif. Intell. Interact. Digital Entertain., 2010.
[11] M. Frade, F. F. de Vega, and C. Cotta, “Evolution of artificial terrains for video games based on accessibility,” in Proc. Eur. Conf. Appl. Evol. Comput., 2010, vol. 6024, pp. 90–99.
[12] J. Togelius, M. Preuss, and G. N. Yannakakis, “Towards multiobjective procedural map generation,” in Proc. FDG Workshop on Procedural Content Generation, 2010.
[13] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, and G. N. Yannakakis, “Multiobjective exploration of the starcraft map space,” in Proc. IEEE Conf. Comput. Intell. Games, 2010, pp. 265–272.
[14] D. A. Ashlock, S. P. Gent, and K. M. Bryden, “Evolution of l-systems for compact virtual landscape generation,” in Proc. IEEE Congr. Evol. Comput., 2005, pp. 2760–2767.
[15] R. M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra, “Integrating procedural

generation and manual editing of virtual worlds," in Proc. ACM Foundations Digital Games. New York: ACM Press, 2010.
[16] E. Hastings, R. Guha, and K. O. Stanley, "Evolving content in the galactic arms race video game," in Proc. IEEE Symp. Comput. Intell. Games, 2009, pp. 241–248.
[17] E. J. Hastings, R. K. Guha, and K. O. Stanley, "Automatic content generation in the galactic arms race video game," IEEE Trans. Comput. Intell. AI Games, vol. 1, no. 4, pp. 245–263, Dec. 2010.
[18] E. Hastings, R. Guha, and K. O. Stanley, "Neat particles: Design, representation, and animation of particle system effects," in Proc. IEEE Symp. Comput. Intell. Games, 2007, pp. 154–160
[19] E. J. Hastings and K. O. Stanley, "Interactive genetic engineering of evolved video game content," in Proc. FDG Workshop Procedural Content Generat., 2010.
[20] A. Martin, A. Lim, S. Colton, and C. Browne, "Evolving 3d buildings for the prototype video game subversion," in Proc. EvoApplications, 2010.
[21] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. V. Gool, "Procedural modeling of buildings," ACM Trans. Graph., vol. 25, pp. 614–623, 2006.
[22] J. Golding, "Building blocks: Artist driven procedural buildings," in Proc. Present. Game Develop. Conf., 2010.
[23] M. Bokeloh, M. Wand, and H.-P. Seidel, "A connection between partial symmetry and inverse procedural modeling," in Proc. SIGGRAPH, 2010.
[24] J. O. Talton, D. Gibson, L. Yang, P. Hanrahan, and V. Koltun, "Exploratory modeling with collaborative design spaces," ACM Trans. Graph., vol. 28, 2009.

Documento de Apoio 2 - Semana 2

Experience-Driven Procedural Content Generation (EDPCG) extends PCG by making player experience, their emotions, preferences, and play style the optimization target. Games are uniquely powerful for eliciting complex affective patterns, making them ideal testbeds for affective computing and adaptive content. The growing diversity of the player base makes one-size-fits-all design less viable; EDPCG aims to tailor content to individual profiles.

Affective games research frames this within the affective loop:

- **Elicitation**: game interaction provokes emotion.
- **Detection & modeling**: the system infers affective state.
- **Adaptation**: content is modified to optimize experience.

Not all content types are generatable, controllability is low, and quality isn't guaranteed. EDPCG addresses this by grounding PCG in explicit models of player experience.

What is EDPCG?

EDPCG treats content as building blocks of player experience. A game is essentially a synthesis of content that elicits affective responses; the generator's role is to find the right blocks for each player.

Core components:

- **Player experience modeling**: predicts how a given player reacts to content (via playing style, affect, cognition).
- **Content quality assessment**: evaluates content in terms of player experience outcomes.
- **Representation**: defines how content is encoded for efficient search.
- **Content generator**: searches the space of content, guided by the player model.

Example: Personalized Mario Levels

Pedersen et al. (2010) demonstrated EDPCG with Super Mario Bros:

- Levels encoded as short parameter vectors (number/size/placement of gaps, optional mechanics).
- Hundreds of players provided self-reports on affect (fun, challenge, frustration, etc.), while gameplay metrics (jumps, deaths, movement patterns) were logged.
- Neural networks trained via evolutionary preference learning mapped content + play style → predicted affect.
- These models then served as evaluation functions for search, allowing automatic generation of personalized levels (e.g., one player's "fun" = unpredictable gaps, while an AI agent's "fun" = large, challenging jumps).

Player Experience Modeling (PEM)

- Three inputs to model experience: **subjective** (self-reports), **objective** (physiology/behavioral signals), **gameplay-based** (interaction logs).
- These can be combined into hybrid approaches for richer, more reliable models.

Subjective PEM

- **Free-response** (rich but hard to analyze, requires annotation assumptions).
- **Forced self-reports** via questionnaires: **ratings/scales** and **pairwise preferences** (compare A vs. B).
- **Strengths:** direct access to perceived experience; can yield accurate models.
- **Limitations:** response noise (learning, self-deception), intrusiveness if in-game, memory bias if post-session; unclear optimal timing window.
- **Evidence:** self-reports have successfully guided ML models across arcade, platform, racing, and prey/predator games.

Objective PEM

- **Modalities:** ECG/heart rate, photoplethysmography, GSR, respiration, EEG, EMG, pupil size, gaze, facial expression, posture, speech.
- **Model-based → model-free continuum:** from theory-driven (e.g., arousal–valence mappings) to learned mappings from annotated data; many hybrids in practice.
- **Strengths:** holistic, multimodal view can improve accuracy.
- **Limitations:** intrusiveness, practicality (sensor placement), feasibility (real-time, noise, habituation), environment sensitivity (lighting/distance for gaze/pupil), player stillness/silence.

- **Example (model-free):** affective camera control in a 3D prey/predator game links physiological signals and camera profiles to reported challenge, frustration, fun; linear and nonlinear models map signals → affect.

Gameplay-Based PEM

- **Assumption:** actions and preferences reflect cognition and (indirectly) emotion; infer affect from interaction patterns + context.
- **Model-based inspirations:** usability, BDI, OCC, Skinner, Scherer, plus game-specific theories (Malone, Koster, Flow).
- **Features:** performance/time, death counts/unpredictability, weapon choices, spatial–temporal stats; many challenge/difficulty metrics.
- **Player modeling:** predict actions/intentions and cluster playstyles (model-free).
- **Strengths:** least intrusive, real-time efficient.
- **Limitations:** lower affect resolution; relies on strong assumptions linking behavior to emotion.
- **Example (implicit, online):** Galactic Arms Race—weapon “fitness” is how much players actually use it; popular weapons are recombined/evolved live.

Hybrid PEM

- **Subjective + Objective:** psychophysiology/affective gaming; correlate physiology with reports.
- **Subjective + Gameplay:** learn predictors of reported affect from in-game stats and preferences.
- **Objective + Gameplay:** correlate physiology with gameplay choices; explore mappings between annotated affect (video/speech) and gameplay features.

General Modeling Principles

- **If labeled/continuous effect:** use regression/classification (NNs, SVMs, trees, Bayesian nets, linear models).
- **If pairwise preferences:** use preference learning (e.g., neuro-evolutionary preference learning; also LDA-style linear methods) to predict orderings rather than absolute labels.

Evaluating Game Content

- **Direct functions:** map content features, quality, **theory-driven** or **data-driven**, can be personalized via PEM.
    - **Example:** Personalized Super Mario—NNs map (level params + playstyle) → predicted affect; used as a direct, data-driven evaluator.
- **Simulation-based functions:** artificial agents play the content, extract features → quality, **static** (fixed agent) vs **dynamic** (learning agents).
    - **Example (static):** Racing tracks evolved; human-like NN drivers assess progress variability and speed profile.
    - **Example (dynamic):** Grid prey/predator rulesets scored by how learnable they are (trivial/impossible penalized; moderately learnable rewarded).
- **Interactive functions:** compute quality during play from **explicit** (in-game self-reports) or **implicit** (usage, quitting, button intensity, physiology, gaze, speech, posture) signals.
    - **Example (implicit):** Galactic Arms Race—weapon use drives fitness.
    - **Example (explicit):** Interactive building evolution—players pick favorites; offspring generated accordingly; personalized quests mixing explicit/implicit cues.

Why search-based over purely constructive: controllability and goal-directed optimization (constructive methods are efficient but often uncontrollable); constructive components can still serve as genotype→phenotype decoders or be made more controllable via declarative modeling.

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

- **Procedural Content Generation via Machine Learning (PCGML)** generates game content using models trained on existing content, complementing constructive, search-based, and solver-based PCG.
- A model is trained on examples of game content and then **directly outputs content** (full or partial). This differs from search-based PCG where learned models may evaluate content but a separate optimizer generates it.
- Distinct from **Experience-Driven PCG**: PCGML models **content**, not player experience; though both can be combined.

Motivation

- Lets designers “speak the same language” as the generator by providing **example artifacts** instead of hand-coding rules/objectives.
- Captures implicit **design style and constraints** present in real content, aiding playability and thematic consistency.
- Learned latent spaces enable **style, variation, and interpolation** across content families.

Use Cases

Autonomous Generation:

- Train on representative artifacts and produce new, stylistically consistent content online or offline without hand-written evaluators.
- Useful for roguelikes and endless games where fresh, valid content is needed on demand.

Cocreative and Mixed-Initiative Design:

- Designer sketches or partially specifies content; the model **autocompletes/inpaints** missing parts.
- Lowers barrier to entry (no code/constraints authoring), reduces iteration friction, and can suggest alternatives in the designer’s style.

Repair

- Models trained on valid content detect **illegal or unplayable** patterns and propose fixes (e.g., autoencoders replace unseen/invalid tiles with nearby valid windows;

learned “move-intent” tiles bias toward passable terrain).

Recognition, Critique, and Analysis:

- Use learned embeddings to **classify, compare, cluster**, or flag anomalies in content; assess similarity across games; estimate novelty or typicality.
- Enables unsupervised **quality checks** or supports other generators as a content-aware critic.

Data Compression:

- Learn compact latent codes (e.g., autoencoders) to **store and transmit** large corpora of content efficiently; decode on demand (akin to classic PCG compression goals).

Methods and Taxonomy

- PCGML methods can be organized by **data representation** (sequences, grids, graphs) and **training methods** (backpropagation, evolution, frequency counting/Markov, expectation–maximization, matrix factorization). The same content type can be cast into different representations, and the same model family can be trained with different optimization schemes.

Sequences — Frequency Counting (n-grams/Markov)

- Treat content as an ordered token stream; learn **conditional probabilities** of the next token from the previous n tokens; generate by sampling.

- **Dahlskog et al.**: Convert Super Mario Bros. (SMB) levels into vertical “slice” tokens and train n-grams. n=0 → noise, n=1 → barely playable, n=2–3 → coherent, often playable segments; the slice trick exploits repetition in SMB levels to make a 2D level 1D-sequence-friendly.
- **Summerville et al. (n-grams + MCTS)**: Use the learned next-slice probabilities during Monte Carlo rollouts and score complete levels with designer objectives (e.g., difficulty, path length). Keeps generation within seen configurations but adds

playability guarantees and **designer control** via tree search.

Sequences — Evolution

- Evolve **generators** that emit token columns over time; fitness measures how well the generator **matches a corpus**.

- **Hoover et al. (FSMC for SMB)**: Adapt a music composition framework—evolve ANNs (via NEAT-style neuroevolution) that predict each tile-type per column given prior columns. Train on two-thirds of human levels, then iteratively add outputs back as inputs to capture **intra-level relationships**; yields levels that retain key design motifs while introducing novel combinations.

Sequences — Backpropagation (LSTMs/Seq2Seq)

- Learn to **next-token** or **conditional** generate with differentiable sequence models.

- **Summerville & Mateas (LSTM for SMB)**: Linearize levels into tile strings; add an **exemplar player path** channel and a **progress/position** signal so the network learns where levels should end and where paths should go. Outputs both geometry and a plausible path, improving **implicit playability**.
- **Summerville et al. (YouTube paths)**: Train with **real player trajectories** extracted from videos; conditioning on a player's style biases outputs (e.g., coin-collector yields coin-rich layouts), demonstrating **personalized style transfer** to levels.
- **@RoboRosewater (LSTM for Magic: The Gathering)**: Train on full card corpus; treat a card as a **sequence of textual fields** to generate novel cards. Limitation: early fields can't condition on later fields naturally (ordering dependency).
- **Mystical Tutor (Seq2Seq inpainting for cards)**: Encoder–decoder LSTM trained to reconstruct cards from **corrupted inputs** (MISSING fields). At generation, supply any subset of fields (e.g., name, color identity) and the model **fills in the rest**, a **co-creative autocomplete** that remedies the conditioning limits of pure LSTMs.

Grids — Frequency Counting (MdMC/MRF/WaveFunctionCollapse)

- Model 2D neighborhoods rather than 1D histories; sample tiles conditioned on **surrounding tiles**.

- **Snodgrass & Ontañón (MdMC)**: Learn 2D Markov neighborhoods over tile types; extend with **hierarchical** models for larger patterns and **constraints** to enforce

usability (e.g., reachable platforms). Also propose an **MRF** variant that outperforms MdMC on Kid Icarus by better handling **platform placement dependencies**.
- **Gumin (WaveFunctionCollapse)**: Learn all N×N tile **patch frequencies** from an example; generate by "collapsing" superpositions with **constraint propagation**, choosing tiles that keep local adjacency valid. Produces image/level samples (2D/3D) that strongly preserve **local style** without explicit rules.

Grids — Backpropagation (Autoencoders/CNNs)

- Learn **latent tile windows** or **spatial filters** to reconstruct or predict grid content.

- **Jain et al. (Autoencoders for SMB)**: Train on vertical windows to compress/reconstruct valid level patterns. Use the network to **repair** unplayable inputs: slide a window, replace illegal patterns with the autoencoder's **nearest valid reconstruction**; also discriminate generated vs. original and sample via noise transforms.
- **Lee et al. (CNNs for StarCraft II resources)**: Convert maps to **heightmaps** and train CNNs to predict plausible mineral/gas placements tied to terrain topology. With **post-processing controls**, designers can dial resource density up/down; notes **small-data overfitting** and the need for heuristics to stabilize outputs.

Grids — Matrix Factorization (NNMF/PCA)

- Factor large level sets into **parts × weights**; recombine/interpolate in latent space.

- **Shaker & Abou-Zleikha (NNMF over multiple SMB generators)**: Encode thousands of levels (from five different non-ML generators) as per-column vectors for platforms/hills/gaps/items/enemies; NNMF yields **part matrices** (global patterns) and **coefficients** (pattern usage). By mixing coefficients, generate levels **outside the expressive range** of any single source generator.
- **Summerville et al. (PCA for Zelda rooms)**: Treat each room as a tile grid; compute PCA eigenrooms to get a compact basis; **interpolate** between room codes to synthesize new rooms that smoothly blend structures. Used in a **hierarchical pipeline** (graph topology first, then per-room tiles).

Graphs — Expectation–Maximization (Clustering/Bayes Nets)

- Learn higher-level **structures and relations** not tied to fixed grids.

- **Guzdial & Riedl (Hierarchical K-means from gameplay video)**: Parse SMB video with CV + sprite sheets to extract frames → **chunks** → **shapes**; hierarchical clustering learns a **shape grammar** (styles and how they combine). Generate by sequencing learned chunks per video-derived order, producing levels that **mirror human play pacing** and composition.
- **Summerville et al. (Bayes Net for dungeon topology)**: Learn a probabilistic model over **room-graph properties** (size, path length, door types). Designers can **condition** on targets (e.g., 12 rooms, optimal path length 6), then sample a consistent room-to-room graph before filling rooms (via the PCA step above).

Graphs — Frequency Counting (Interactive Fiction)

- Learn **event graphs** and ordering from text corpora to generate playable narratives.

- **Guzdial et al. (Scheherazade-IF)**: Crowdsource simple stories in a domain (e.g., bank robbery). Cluster sentences into **events** (OPTICS for arbitrary-shape clusters), derive **plot graphs** (events as nodes, edges as precedence). At runtime, let players assume roles (robber/teller) and make choices constrained by the **learned event graph**, yielding coherent branching stories.

Conclusion

- **Platformers** are the most explored domain across all representations; **graphs** capture abstract structure (style/relations) but need more machinery at render time; **sequences/grids** encode geometry explicitly (often fixing width/height).

- **Training/runtime cost**: LSTMs/seq2seq train longer and sample more expensively (token-by-token) than MdMC/MRF; MdMC trains in a **single pass** and is fast at sample time. Matrix factorization holds all data in memory (fine for small corpora, problematic at scale). Graph grammars scale their generation cost with **training set complexity**.

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

Documento de Apoio 4 - Semana 2

Procedural Content Generation in Games

Capitulo 2 - Search Based Approaches (Content Representations)

Content Representations

Representation defines how game content is described as data

The choice of representation affects:

- **Expressive range**: how much variety the generator can produce
- **Locality**: whether small changes in the data cause small changes in the content
- **Dimensionality**: how many independent parameters are needed to describe content

Direct vs Indirect Representations

Direct (low-level)

- Each data element maps directly to content
- Example: array where each entry = a Mario tile
- Pros: fine-grained control
- Cons: very high dimensionality → huge search space

Structured / Indirect (mid-level)

- Data encodes entities, patterns, or rules rather than raw tiles
- Example: list of enemies, gaps, platforms, or reusable elements like coin stairs
- Pros: compact, easier to enforce structure
- Cons: less precise control

Abstract / Very Indirect (high-level)

- Data describes general properties or just a seed value
- Example: 10 enemies, 5 gaps, average gap width = 3
- Pros: very compact, simple to generate
- Cons: low locality → small data changes may drastically alter content

Capítulo 05 - Grammers

A **grammar** defines how strings in a language can be formed using rules.

- **Non-terminals**: placeholders that can be expanded further.
- **Terminals**: final symbols (cannot be expanded).
- **Production rules**: specify how non-terminals expand.

Example grammar for sentences:

```
<sentence> ::= <noun> <verb> <noun>
<noun> ::= cat | dog
<verb> ::= eats | chases
```

This can generate strings like:

- `cat eats dog`
- `dog chases cat`

The book gives an example using the Infinite Super Mario grammar that defines how levels are structured in terms of chunks (pieces of terrain) and enemies.

Example:

- Top level: A level is made of a sequence of chunks plus some enemies.

```
<level> ::= <chunks> <enemy>
```

- Chunks: A level can contain one chunk or multiple chunks

```
<chunks> ::= <chunk> | <chunk> <chunks>
```

```
<chunk> ::= gap(<x>,<y>, <wg>,<wbefore>,<wafter>)
          | platform(<x>,<y>,<w>)
          | hill(<x>,<y>,<w>)
          | blaster_hill(<x>,<y>,<h>,<wbefore>,<wafter>)
          | tube_hill(<x>,<y>,<h>,<wbefore>,<wafter>)
          | coin(<x>,<y>,<wc>)
          | blaster(<x>,<y>,<h>,<wbefore>,<wafter>)
          | tube(<x>,<y>,<h>,<wbefore>,<wafter>)
          | <boxes>
```

- `gap(...)` → a gap of certain width
- `platform(...)` → a flat platform
- `hill(...)` → a hill
- `tube(...)` or `blaster(...)` → pipes and cannons
- `coin(...)` → coins
- `<boxes>` → blocks

- Boxes: Boxes can appear 2 to 6 times at coordinates `(x,y)`. They may contain a coin, a powerup, or be empty.

```
<boxes> ::= <box_type>(<x>,<y>)2 | ... | <box_type>(<x>,<y>)6
<box_type> ::= blockcoin | blockpowerup | brickcoin | brickempty
```

- Enemies: Enemies are Koopas or Goombas, appearing 2 to 10 times at different positions.

```
<enemy> ::= (koopa | goomba)(<pos>)2 | ... | (koopa | goomba)(<pos>)10
```

- Parameters: Constraints are chosen so levels are playable (e.g., gaps can be jumped across, pipes can be cleared).

```
<x> ::= [5..95] (horizontal position in level)
<y> ::= [3..5] (vertical placement, jump height range)
<wg> ::= [2..5] (gap width)
<wbefore> ::= [2..5] (space before obstacle)
<wafter> ::= [2..5] (space after obstacle)
<w> ::= [2..6] (width of platforms/hills)
<wc> ::= [2..6] (number of coins)
<h> ::= [3..4] (pipe or cannon height)
<pos> ::= [0..100000] (enemy placement encoding)
```

Grammatical Evolution was used for this example, with the fitness function being:

- Rich design: number of chunks in the level (not too empty, not too crowded).

- Fewer conflicts: avoid overlapping chunks (like a pipe sitting inside a coin cluster).

Capítulo 09 - Representations (Focused on Search Based Methods)

Representation example: genotype to phenotype map

- **Genotype**: search-space variables (bit grid, room list, network weights/topology, agent params).
- **Phenotype**: realized game artefact (level layout, texture/field, generator's behaviour).
- **Mapping**: f:G→Pf:\mathcal{G}\rightarrow\mathcal{P}f:G→P (often many-to-one, sometimes stochastic).

Locality

- **Definition**: small Δ in genotype ⇒ small Δ in phenotype.
- **Importance**: smoother fitness landscape → gradient-free search (EA, ES) finds improvements reliably.

Expressivity

- **Definition**: diversity/coverage of reachable artefacts.
- **Proxies**: phenotype entropy; coverage of a feature space (e.g., MAP-Elites bins); number of distinct phenotypes per fixed-length genome.
- **Trade**: more expressive encodings often ↓ locality (indirect = compact but "jumpy").

Dimensionality

- **Definition**: number of free parameters (effective DoF).
- **Effects**: ↑ dims → ↑ search effort; but too few → underfit design space.
- **Obs**: Curse of dimensionality, dimensions change given the representation

Capítulo 10 - Experience Driven Perspective

Games are interactive systems where players and games adapt to each other. As players, we learn mechanics, enemy behaviours, and level structures. At the same time, games can observe every input and outcome, potentially learning just as much about us.

The simplest form of adaptation is **Dynamic Difficulty Adjustment (DDA):**

- If a player does well, increase challenge; if poorly, reduce it.
- Examples: rubber-banding in racing games, stronger items for losing players in *Mario Kart*, or skipping failed missions in *GTA V*.

This connects to **Flow theory** (Csikszentmihalyi): keeping players in a “flow channel” where difficulty is balanced with skill. However, challenge is only one dimension of experience; players also differ in preferences for pacing, atmosphere, and style.

Experience-driven PCG generalises adaptation: content generation adapts along many axes (difficulty, pacing, aesthetics, narrative) to match or shape player experience, not just skill.

Experience-driven PCG treats **game content as the building block of experience.** Levels, mechanics, rewards, enemies, and audiovisual elements all contribute to immersion, affect, and challenge.

Content can elicit experience in different ways:

- **Spatial involvement:** layout, navigation, and exploration.
- **Affective involvement:** narrative tone, atmosphere, aesthetics.
- **Agents/NPCs:** behaviours and emotional cues can strongly influence engagement.

Data about player experience can be collected through:

- **Short-term interactions:** immediate reactions to obstacles, rewards, or enemies.
- **Long-term interactions:** learning curves, player strategies, gradual adaptation.
- **Social interactions:** potentially powerful but hard to model; usually excluded from current PEM focus.

These elicited responses form the raw material for modelling (gameplay logs, physiological signals, annotated reactions). By linking them back to specific content features, games can begin to predict and adapt to individual experience patterns.

Modelling player experience

Player Experience Modelling (PEM) aims to predict how players feel or respond to game content. It is a special case of affective computing but more complex because of the richness of game–player interaction. PEM combines behaviour, cognition, and affect, linking gameplay, physiology, and game context to annotated measures of experience.

Model-based vs. Model-free

- **Model-based:** grounded in theory (e.g., Flow), interpretable but rigid.
- **Model-free:** learns unknown functions from data, flexible but opaque.
- Most systems mix both, using theory to guide features and ML to fit parameters.

Model input

Three main sources:

- **Gameplay data:** logs of actions (jumps, kills, timings, sequences). Can be analysed statistically or as frequent temporal patterns.
- **Objective data:** physiology (ECG, EEG, GSR, respiration), body posture, facial expression, even speech. Reflect arousal and affect but ambiguous without context.
- **Game context:** level geometry, difficulty, pacing, audiovisual intensity. Essential for disambiguating signals (e.g., high arousal = excitement or frustration?).

Feature extraction can use handcrafted statistics, sequence mining, or deep learning. Feature selection (PCA, forward search, genetic algorithms) reduces dimensionality.

Model output (experience annotation)

Ground truth is obtained through:

- **Self-reports:** questionnaires or probes during play (e.g., GEQ, Geneva Wheel).
- **Third-person annotation:** experts or observers.
- **Annotation formats:** ratings (scalar), classes (categorical), or preferences (comparisons).
  Preference data often yields higher reliability than ratings.

Modelling approaches

- **Regression/classification:** for ratings or categorical states (SVMs, decision trees, neural nets).
- **Ranking methods:** for preference data (RankSVM, neuroevolutionary preference learning).
- **Hybrid models:** combine theory (e.g., “enemy density → challenge”) with ML flexibility.

Integration with PCG

PEM provides evaluation functions for search-based generation. Instead of optimising only playability or difficulty, generators can optimise for predicted *experience* (engagement, frustration balance, flow), enabling personalised and adaptive content.

Exemplo: Super Mario Bros.

Shaker et al. used **Infinite Mario Bros.** as a test for experience-driven PCG.

- **Data collection:** combined gameplay logs (jumps, deaths, coins), objective video features (head movement, posture), and self-reports.
- **Feature selection:** used sequential forward selection with simple neural nets to identify which features best predict engagement, frustration, or challenge.
- **Model building:** expanded to larger neural networks via neuroevolution for higher accuracy.
- **Application:** models became evaluation functions in a **grammar-based level generator**, evolving Mario levels tailored to predicted player experience.

Capítulo 12 - Experience Driven Perspective

Evaluators are hard to judge due to:
- Players are subjective and diverse.
- Designers impose different constraints.
- Algorithms are stochastic, and players behave unpredictably.

Aside from correctness, there is the question of creativity: does the generator recombine known structures (exploratory creativity) or create new ones (transformational creativity)? Evaluation should capture not just feasibility, but also novelty, aesthetics, and whether the generator enhances designer creativity in mixed-initiative use.

Top-down evaluation: Expressivity measures

Instead of inspecting individual outputs, we evaluate the expressive range: the space of all content a generator can produce.

- Defined by metrics (e.g., linearity, leniency, density in platformers).
- Large samples are generated, evaluated along these axes, and visualised in heatmaps.
- Reveals biases (where the generator produces more/less content) and controllability (how input changes affect output space).

Metrics should capture *emergent qualities* rather than repeat input parameters. For example, if difficulty is already an input, it should not also be the main metric.

Bottom-up evaluation: Players

Complementing statistics, user studies provide insight into how players perceive generated content.

- **Questionnaires**: can measure content quality as ratings, classes, or preferences.
    - Preference-based schemes (ranking levels against each other) tend to be more reliable than ratings.
- **Alternatives to self-report**: physiological data (GSR, heart rate, EEG), behavioural traces (time stuck, retries), or implicit metrics (frequency of use).
- **Bias reduction**: rank-based surveys, video replays for annotation, or hybrid methods that combine designer constraints with player input.

Crowdsourcing can scale evaluation, but content should usually be filtered or pruned before presenting to players.

APÊNDICE 2

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 18 de set. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

EDWARD SCOTT CARVALHO JOHNSON

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Para a terceira Semana do processo da disciplina Residência em IA, foi feito:

Uma leitura dos capítulos 2, 7 e 8 do livro "Artificial Intelligence and Games".

- O livro funciona explicando certos conceitos e técnicas dentro da área, e traz exemplos da literatura para ajudar com o entendimento e demonstrar como as técnicas ensinadas são aplicadas.

- O capítulo 2 fala sobre fundamentos da IA e explicações básicas dos modelos.
- O capítulo 7 é um overview de PCG na geração de fases e ambientes no geral, passando uma visão do estado atual da pesquisa e do mercado.
- O capítulo 8 explica de forma aprofundado as técnicas e como são utilizadas. Eles são separados em:

    - Constructive Algorithms: Conteúdo é gerado de uma vez (grammers, celular automata, fractais, ruído, WFC).
    - Search-Based Algorithms: Trata a geração como uma busca (métodos evolutivos).
    - Quality Diversity: Algoritmos focados em trazer diversidade e qualidade, apropriados para o PCG.
    - Solver-Based: Trata PCG como um problema de satisfação de condições (Answer Set Programming).
    - Machine Learning: PCG onde é necessário uma base de dados para o treino de novos samples (n-gram, GANs, LSTMs, LLMs, Diffusion Based).
    - RL: Modelos de reforço onde geração de certos padrões retorna uma recompensa positiva.

- Foi feito anotações especificamente do capítulo 8: Gate 3 - Artificial Intelligence and Games

- Dado a leitura do livro, e também de leituras passadas, cheguei a conclusão que quero me aprofundar mais na parte de Machine Learning / IA do PCG, e me focar menos em procurar papers

sobre algoritmos construtivos, search-based, etc. Também anotei vários papers ao longo da minha leitura de surveys e o livro, e quero começar a escolher alguns para fazer a leitura.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Ler os seguintes papers:
- Deep Learning for Procedural Content Generation
- Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network (GANs)
- Level Generation Through Large Language Models (LLMs)
- Super Mario as a String: Platformer Level Generation Via LSTMs (LSTMs)
- The VGLC: The Video Game Level Corpus (Dataset)

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Documento de Apoio - Semana 3

Capítulo 8

Os métodos são separados em:

- Constructive Algorithms: Um método de PCG onde o algoritmo gera o conteúdo uma vez e o processo de geração acaba ali. Não necessariamente você não pode testar o conteúdo para saber se está bom, mas o algoritmo de geração não participa dessa verificação. Isso anda contrariamente a todos os outros algoritmos nessa lista, as quais são “generate-and-test”.

- Search-Based Algorithms: Algoritmos search based assumem que design é um processo de busca, onde dentro de um espaço de busca, existe ao menos uma solução boa de design. Para chegar nela, é necessário ter uma engine de busca, representação do conteúdo e uma função de avaliação, onde a função de avaliação vai ver as buscas feitas com a representação do conteúdo e vai avaliar sua qualidade para depois fazer leves modificações.

- Quality Diversity: Algoritmos de PCG que focam mais em trazer soluções de alta qualidade e diversidade.

- Solver-Based: Algoritmos solver-based tratam a geração de conteúdo como um problema de satisfação de restrições, onde o conteúdo é produzido a partir de condições pré-definidas que precisam ser atendidas. Em vez de buscar ou testar várias soluções, eles usam solvers lógicas ou matemáticas para eliminar opções inviáveis e encontrar diretamente configurações que satisfaçam todas as restrições.

- Machine Learning: Geradores que criam geradores dado o seu treinamento usando samples passados.

- Reinforcement Learning:

Vou preenchendo essas explicações ao longo da minha leitura, pois quero escrever não por escrever, mas por registrar minha própria explicação com minhas próprias palavras acima do assunto

Constructive Algorithms

Grammar Based Methods: Geralmente, grammers são usados para geração de coisas como árvores e caminhos. Um grammer é formalmente definido como uma série de regras para escrever uma string ou outra estrutura de dado similar. Um exemplo de um grammer é um L-System, mas grammers também podem ser adaptados para funcionar com outros métodos, como as search-based onde a expansão do grammer é mapeado como genótipo para phenotype

Cellular Automata: É uma forma de computacionalmente modelar fenômenos biológicos e físicos. O conceito mais básico é um grid com células que mudam seu estado depois de um tempo discreto dado algumas regras, as quais agem dependendo do estados da própria célula e das células adjacentes.

A representação pode ser em N dimensões, mas geralmente é em 1D ou 2D (vetor ou matriz) e cada célula tem um número fixo de estados. As suas células adjacentes definem a sua vizinhança (seu **neighborhood**), a qual define o estado da célula em timesteps futuros.

Cellular Automata é usado bastante para simulações de fogo, inundações, chuva, explosões. Um study case é o uso de CA por Johnson Et Al [1] para gerar masmorras infinitas em forma de cavernas, com aparência orgânica.

- Cada sala é uma grade de 50×50 com células de rocha ou vazias.
- O processo começa com a distribuição aleatória de rochas, seguido pela aplicação repetida de uma regra de CA:
    - Uma célula se torna rocha se pelo menos *T* de seus vizinhos forem rocha.

- Rochas na borda são transformadas em paredes por motivos estéticos.
- Para conectar as salas, gera-se também as salas vizinhas, criando túneis quando necessário, e em seguida suaviza-se o resultado com mais iterações de CA.
- Isso torna a geração rápida (menos de um milissegundo para 9 salas) e escalável.

- Vantagens: Simples, rápido, poucos parâmetros, produz cavernas naturais.
- Desvantagens: Parâmetros afetam várias características de forma imprevisível, não há garantias de jogabilidade ou requisitos específicos de design.

Algoritmos de Ruído: São bastante usados para heightmaps e texturas. Quando a resolução é maior que a matriz, usa interpolação.

Fractais: Também usados para geração de terreno 2D, também para plantas e elementos que requerem padrões.

Wave Function Collapse: Dado uma entrada, ele cria conteúdo localmente semelhante, assim garante que apenas coisas na entrada apareçam na saída. Com uma entrada, o algoritmo identifica todos os tiles dentro dele e depois define algumas regras de cada tile (tile x não pode conectar com tile y). Aí começa o wave, onde uma célula é escolhida para colapsar, e com isso as células em sua volta também vão colapsar.

Search Based Algorithms

Primeiro precisa considerar o custo e localidade na escolha da representação: representações mais detalhadas oferecem maior localidade, ou seja, soluções vizinhas no espaço de busca tendem a ter valores de qualidade semelhantes, mas exigem maior custo computacional, já representações mais simples reduzem o espaço de busca e o custo, mas sacrificam localidade.

Sobre a avaliação, três tipos de funções de avaliação são descritos. As diretas avaliam o conteúdo ou seus atributos sem necessidade de simulação, baseando-se em regras fixas ou modelos treinados a partir de dados; são rápidas e fáceis de aplicar, mas têm limitações em capturar aspectos complexos. As funções baseadas em simulação utilizam agentes de inteligência artificial que jogam o conteúdo para estimar sua qualidade, permitindo avaliar fatores como jogabilidade ou imitar experiências humanas; exemplos incluem medir se há um caminho jogável em níveis de plataforma ou calcular o tempo médio de combate entre bots em jogos de tiro. Já as funções interativas envolvem o jogador diretamente, podendo ser implícitas, quando se observa o comportamento do jogador (como frequência de uso de armas), ou explícitas, quando se coleta feedback direto. Essa última abordagem fornece informações mais confiáveis, mas pode interromper a experiência de jogo, razão pela qual muitas vezes são combinadas em métodos híbridos que associam comportamento, olhar e até dados fisiológicos.

Quality Diversity

Os métodos solver-based são adequados quando o problema pode ser totalmente descrito em termos de restrições lógicas ou matemáticas. É difícil integrar eles diretamente com simulações ou chamadas ao motor do jogo, algo que se torna mais natural com algoritmos evolutivos. Métodos evolutivos podem complementar os solver-based, combinando valores de fitness e restrições para guiar o processo de geração.

Machine Learning

O desafio principal de PCG com ML é que jogos possuem um número massivo de limitações com o jeito que você pode interagir com eles, então a saída do modelo não pode ser simplesmente uma aproximação do que ele pensa que seria uma fase, como uma imagem, porém não é o suficiente para um jogo. Onde em uma imagem que um modelo de difusão ou GAN gerou de um cavalo com pequenas imperfeições, o usuário comum não se importa com essas imperfeições, ou ao menos não afeta ele muito. No caso dos jogos, uma mini imperfeição pode quebrar o jogo inteiro.

Exemplos a seguir serão a respeito do Mario

N-Gram: Começa transformando os níveis bidimensionais em sequências unidimensionais. Para isso, cada nível foi dividido em fatias verticais (colunas de tiles), e cada fatia foi tratada como um token.
Como muitas dessas fatias se repetem ao longo dos níveis, isso gera uma sequência com bastante redundância, ideal para o treinamento do modelo de n-gramas. A partir dessa sequência, o algoritmo construiu tabelas de probabilidades condicionais. Ou seja, para cada subsequência de tamanho *n* (o contexto), o modelo registrava quais fatias poderiam aparecer em seguida e com que frequência.
Por exemplo, com *n* = *2*, o modelo observa duas fatias consecutivas e calcula a distribuição de probabilidades da terceira.
Na geração de novos níveis, o modelo mostrava essas distribuições e ia concatenando fatias verticalmente, resultando em fases que seguiam os mesmos padrões estatísticos encontrados nos níveis originais. Quanto maior o valor de *n*, mais o resultado se parecia com os níveis reais.

Redes Neurais Básicas: A ideia foi adaptar uma representação originalmente feita para compor música (FSMC), tratando cada coluna do nível como uma unidade de tempo. As redes aprendem relações ocultas entre tipos de tiles nos níveis originais e conseguem gerar fases inteiras a partir de poucas informações iniciais.

GANs: Modelo simples onde o discriminador e o gerador são treinados. Um exemplo de uso é o MarioGAN, um GAN é treinado com níveis do VGLC. Em seguida, o algoritmo CMA-ES busca vetores latentes que geram fases com propriedades específicas (por exemplo, número de canos, buracos ou blocos). Esse processo pode ser estendido com algoritmos de qualidade-diversidade, como MAP-Elites, para produzir níveis variados segundo diferentes descritores de comportamento.

LLMs: Podem ser usados em PCG tratando níveis de jogos como sequências de símbolos (por exemplo, tiles, salas ou embeddings). Um modelo GPT ou similar é então fine-tuned em corpora de níveis já existentes

RL: Reinforcement Learning (RL) pode ser aplicado em PCG tratando a geração de conteúdo como um problema de decisão sequencial: o agente faz edições em um nível e recebe recompensas conforme melhora a qualidade do resultado. Esse campo é chamado de PCGRL. Um exemplo é o trabalho de Khalifa, que gera níveis 14×14 para jogos como Binary, Zelda e Sokoban. No modo narrow, o agente percorre o mapa tile por tile, decidindo se troca o tile e recebendo recompensas com base em métricas simples (ex.: distância entre chave e porta). Há também os modos turtle (o agente decide a direção de movimento além da edição) e wide (pode editar qualquer posição a cada passo).

APÊNDICE 3

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 25 de set. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

Edward Scott Carvalho Johnson

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nas últimas 3 Semanas do processo foi feito:

- Decisão do estudo do tema Procedural Content Generation (PCG)
- Estudo dos algoritmos clássicos e métodos de ML

Para a quarta Semana do processo da disciplina Residência em IA, foi feito:

Estudo de 5 papers de PCG, os quais envolvem deep learning, especificamente GANs e LSTMs (Gate 04 - GAN, LSTM, Dataset)

- Foi notado que GANs dominam a área atualmente ao ler o paper "Deep Learning for Procedural Content Generation" e colocar os papers no "https://www.connectedpapers.com"
- GANs são usados em conjunto com algum algoritmo evolutivo para maximizar alguma função de fitness
- LSTMs são usados para remediar o tradeoff de outros algoritmos não terem capacidades de coerência local e global
- O dataset público que é usado (VGLC), o qual é inclusive utilizado para o paper de GAN e de LSTM, possui quase 400 fases de 10+ jogos. Ele tem uma grande variedade de jogos, porém a quantidade de fase por jogo é baixíssima. Isso faz com que a comunidade científica tente criar arquiteturas e pipelines que favorecem poucos dados. O paper do GAN que eu li faz treinamento zero shot (com apenas uma amostra).
- RL é usado pelo mesmo motivo que os algoritmos evolutivos, também pode ser usado para gerar a fase toda.

Tenho um grande interesse em métodos de PCG que permitem uma interação do usuário para a geração de uma fase de forma fácil, de forma parecida ao paper do GAN que maximiza um fitness definido pelo usuário. Fiz a decisão de ler alguns papers que usam Diffusão e LLMs para me aprofundar.

- Quero funilar meus estudos da Residência em "Promptable Procedural Content Generation"

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Ler os seguintes papers:
- https://arxiv.org/pdf/2404.08706 (Game Generation via Large Language Models)
- https://arxiv.org/pdf/2507.00184 (Text-to-Level Diffusion Models With Various Text Encoders for Super Mario Bros)
- https://arxiv.org/pdf/2305.18243 (Practical PCG Through Large Language Models)
- https://arxiv.org/pdf/2302.05981 (MarioGPT: Open-Ended Text2Level Generation through Large Language Models)

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Documento de Apoio - Semana 4

VGLC: Video Game Level Corpus

O VGLC tem 428 fases de 12 jogos diferentes, cada um adaptado a um de três formatos de anotação: Tile, Graph ou Vector. A maioria dos jogos 2D de plataforma, como Mario ou Kid Icarus, são descritos no formato Tile. Jogos com estrutura de salas conectadas, como Zelda, também aparecem no formato Graph. Doom 1 e 2, que têm mapas que podem ser representados em 2D mas com jogabilidade de FPS, podem ser encontrados tanto em Tile quanto em Vector.

- **Tile Format**: Esse formato representa o nível como uma matriz 2D de largura × altura. Cada posição na grade é um caractere que simboliza um tipo de tile, e existe um arquivo JSON que funciona como legenda. Em Super Mario Bros “X” representa chão sólido, “?” um bloco de interrogação cheio, “E” qualquer inimigo. Isso torna fácil raciocinar sobre o espaço em termos de tiles. Vários sprites diferentes podem ser mapeados para o mesmo caractere, o que traz perda de informação.
- **Graph Format**: Usado para jogos baseados em salas interconectadas. Cada nó do grafo é uma sala, e as arestas representam conexões (portas, passagens, bloqueios). A representação é feita em DOT (a linguagem do Graphviz), escolhida por ser fácil de parsear, fácil de visualizar e amplamente suportada. Como no Tile format, cada grafo tem uma legenda JSON que define o que significa cada vértice (ex.: “e” = sala com inimigo, “b” = boss, “k” = chave, “t” = triforce) e cada aresta (ex.: “b” = passagem bombável, “k” = passagem trancada por chave). Isso permite estudar a topologia do jogo sem entrar nos detalhes de cada tile.
- **Vector Format**: Pensado para jogos como Doom, que são tecnicamente 3D mas funcionam em um plano 2D. Os mapas são descritos em SVG, o que facilita tanto a leitura quanto a visualização. Linhas representam paredes ou portas, círculos e quadrados representam objetos como inimigos, armas, decorações ou teletransportes. A legenda JSON associa as cores do traço a categorias (vermelho para inimigo, azul para teleporte, verde para munição). Essa abordagem preserva melhor a estrutura geométrica do jogo e permite trabalhar com análises mais espaciais.

Limitações: No Tile format a simplificação causa perda de informação porque não há distinção entre diferentes tipos de inimigos ou blocos especiais. O paper surgere expandir para representações mais expressivas e completas no futuro, talvez com ontologias de tiles. Por isso, o corpus também inclui as imagens originais dos mapas, permitindo que novos pesquisadores criem suas próprias anotações mais detalhadas.

Tools: O VGLC inclui dois utilitários principais. Um solver de plataforma baseado em A*, que usa a legenda JSON e as dinâmicas do jogo (ex: arco de pulo) para calcular caminhos viáveis dentro de fases.

Deep Learning for Procedural Content Generation

Aprendizado Adversarial (GANs e variantes)

GANs são adequados para conteúdos representados como imagens ou matrizes de tiles. Um gerador cria novos conteúdos e um discriminador avalia se parecem reais. Em PCG, isso serve para criar fases, mapas, sprites e até terrenos 3D. Diversos trabalhos adaptaram GANs para melhorar qualidade, diversidade ou lidar com datasets pequenos.

- Kuang e Luo [1]: Sistema interativo de design de mapas 2D, com possibilidade de expansão para 3D, baseado em modelos generativos.
- Torrado et al. [2]: Criaram o CESAGAN, com atenção auto condicional, para aumentar qualidade e diversidade de níveis 2D em Zelda, usando bootstrapping para mais dados.
- Wulff-Jensen et al. [3]: Treinaram um DCGAN em mapas de elevação dos Alpes para gerar heightmaps usados em Unity como terrenos 3D.
- Giacomello et al. [4]: Transformaram fases de Doom em múltiplas imagens e treinaram dois GANs para gerar novos mapas.
- Park et al. [5]: Desenvolveram um DCGAN em múltiplas etapas para gerar fases do jogo educacional ENGAGE.
- Volz et al. [6]: Usaram GANs para níveis de jogos do tipo “match-3”, modelando padrões locais e globais.
- Awiszus et al. [7]: Criaram o TOAD-GAN, baseado no SinGAN, capaz de gerar variações a partir de um único exemplo.
- Di Liello et al. [8]: Introduziram CANs, redes adversariais com restrições para penalizar estruturas inválidas.
- Fontaine et al. [9]: Propuseram Latent Space Illumination para explorar o espaço latente de GANs com métodos de diversidade.

- Kumaran et al. [10]: Criaram uma arquitetura de GAN multijogos, com gerador ramificado e múltiplos discriminadores.
- Hong et al. [11]: Geraram sprites 2D com um GAN multidiscriminador, separando forma e cor.
- Wang e Kurabayashi [12]: Desenvolveram o Sketch2Map, um cGAN que converte esboços em mapas de elevação e terrenos 3D.
- Bontrager e Togelius [13]: Propuseram um ciclo Gerador Agente para coevolução entre níveis e desempenho do agente.

Avaliação de Geradores

Pode ser feito pela análise estatística do conteúdo, por simulação de jogabilidade (agentes) ou pela modelagem da experiência humana. Deep learning vem sendo usado para criar medidas mais semânticas e preditivas.

- Lucas e Volz [14]: Compararam distribuições estatísticas entre fases reais e geradas.
- Isaksen et al. [15]: Usaram autoencoders para hashing semântico de segmentos de níveis.
- Guzdial et al. [16]: Usaram RL profundo para playtesting humanoide em Mario e CNNs para prever dificuldade e diversão.
- Min et al. [17]: Criaram reconhecimento de objetivos em jogos abertos via autoencoders empilhados.
- Karavolos et al. [18]: Previram resultados de partidas FPS 3v3 usando CNNs.
- Karavolos et al. [19]: Estenderam o trabalho anterior como modelo substituto para sugerir combinações de mapas e classes.
- Gudmundsson et al. [20]: Treinaram CNNs para prever jogadas humanas em Candy Crush.
- Larsson e Petri [21]: Usaram NEAT para prever avaliações humanas de mapas de StarCraft.
- Shaker et al. [22–24]: Geraram níveis personalizados de Mario baseados em emoção e experiência do jogador.
- Camilleri et al. [25]: Previram quão críveis eram combinações entre níveis e comportamento de jogadores.
- Summerville et al. [26]: Mapearam métricas estatísticas de níveis às avaliações humanas.
- Pfau et al. [27]: Propuseram DPBM para prever ações prováveis de jogadores via MLPs.
- Martinez et al. [28]: Usaram CNNs para prever experiência em um jogo de labirinto 3D.

- Makantasis et al. [29]: CNNs para prever excitação em jogos de tiro diretamente de pixels.
- Melhart et al. [30]: Previram engajamento em transmissões de PUBG.
- Camilleri et al. [31]: Propuseram modelos de experiência generalizados entre jogos diferentes.

Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network

O paper junta GAN com algoritmos evolutivos. A ideia central é dividir o processo em duas fases.

Na primeira fase, uma DCGAN (Deep Convolutional GAN) é treinada para aprender a estrutura de níveis a partir de um único nível do corpus VGLC. O nível é representado como uma grade de tiles, em que cada tipo de bloco é codificado por um número inteiro e depois convertido em vetores *one-hot*. A GAN recebe como entrada um vetor latente de 32 números reais (inicialmente amostrados de uma distribuição Gaussiana) e gera como saída uma matriz de dimensões 10 × 28 × 14, que corresponde ao mapa de blocos da fase. O discriminador tenta distinguir entre fases reais e geradas, e o gerador aprende a enganar o discriminador. No fim do treinamento, o discriminador é descartado e o gerador se torna um mapeamento de genótipo para fenótipo: vetores no espaço latente são transformados em níveis jogáveis.

Na segunda fase, utiliza-se CMA-ES (Covariance Matrix Adaptation Evolution Strategy) para explorar esse espaço de forma orientada por funções de aptidão. O CMA-ES gera candidatos de vetores latentes, passa cada um ao gerador da GAN para obter uma fase, e depois avalia essa fase de acordo com o critério desejado. Dois tipos principais de funções de aptidão foram testados:

- **Baseadas em representação**, que olham diretamente para a distribuição de blocos (por exemplo, porcentagem de chão ou número de inimigos).
- **Baseadas em agente**, que simulam um agente A* jogando a fase para medir se ela é jogável e qual o nível de dificuldade (estimado pelo número de pulos).

Esse ciclo de evolução permite encontrar vetores latentes que geram fases com propriedades específicas, ao invés de depender apenas da amostragem aleatória.

Resultados

Os autores testaram a expressividade e localidade do espaço latente. Vetores aleatórios produzem fases diversas e reconhecíveis como Mario. Pequenas mutações em torno de um vetor latente geram fases parecidas, mas com variações significativas. Isso demonstra que o espaço é estável e adequado para busca evolutiva.

Nos testes baseados em representação, o CMA-ES foi capaz de ajustar a porcentagem de blocos de chão para valores-alvo específicos (20%, 50%, 70%), quase sempre chegando muito próximo do esperado. Também conseguiu produzir fases compostas por seções de dificuldade crescente: no início 100% chão, depois 70% chão com inimigos adicionados. Isso mostra que é possível guiar o processo para objetivos globais de design, como progressão de dificuldade.

Nos testes baseados em agente, as funções de aptidão focaram em dois objetivos:

- **F1**: maximizar pulos (fases mais difíceis).
- **F2**: minimizar pulos (fases mais fáceis).

O CMA-ES encontrou fases jogáveis onde o agente precisou pular muitas vezes (F1) e também fases simples com apenas um pulo (F2). Em alguns casos, surgiram fases injogáveis ou com erros estruturais, como canos quebrados. Isso ocorre porque regiões próximas no espaço latente podem gerar tanto níveis jogáveis quanto injogáveis.

Durante a evolução, a média da aptidão dos indivíduos melhorou consistentemente, apesar de haver ruído nas avaliações (um mesmo nível pode gerar trajetórias ligeiramente diferentes). O fato de indivíduos ruins aparecerem mesmo em fases avançadas sugere que funções de aptidão mais robustas, ou versões do CMA-ES voltadas para otimização com ruído, poderiam melhorar os resultados.

No geral, o método produziu uma grande variedade de fases, correspondendo a diferentes estilos de jogabilidade. A principal dificuldade é definir funções de aptidão que capturem bem as qualidades desejadas de uma fase (jogabilidade, dificuldade, estética). O estudo sugere que trabalhos futuros podem explorar:

- Modelos que preservem melhor a consistência local (ex: usar LSTMs para evitar canos quebrados).
- Dados mais ricos que incluam não só a geometria da fase, mas também comportamento de jogadores.
- Otimização multiobjetivo, para equilibrar critérios diferentes ao mesmo tempo.

Super Mario as a String: Platformer Level Generation Via LSTMs

O trabalho explora o uso de LSTMs para gerar fases de Super Mario Bros a partir de exemplos, comparando diferentes formas de representar os níveis como sequências. A motivação principal é superar limitações de métodos anteriores baseados em Markov chains, que produzem boa coerência local (um tile faz sentido em relação ao próximo) mas falham em coerência global.

A rede utilizada é composta por 3 camadas de 512 LSTMs cada, com codificação one-hot para os tipos de blocos e saída Softmax que prevê a distribuição sobre o próximo tile. Durante o treinamento, a função de perda é a negative log-likelihood, que avalia o quão provável o modelo considera o tile correto em cada passo. Foram usadas 39 fases originais (15 do Super Mario Bros e 24 do Mario japonês SMB2), com divisão 70%-30% treino/validação.

O ponto central do estudo é comparar oito especificações de dados, variando três dimensões:

- Ordenação: bottom-to-top vs. snaking (alternando direção de leitura das colunas).
- Path information: inclusão ou não do caminho percorrido por um agente A* como caracteres especiais no treinamento.
- Depth information: inclusão de marcadores de progressão de coluna (a cada 5 colunas um símbolo especial).

Essas variações buscavam melhorar tanto a coerência quanto a jogabilidade dos níveis gerados.

Resultados

Os experimentos mostram que:

- O fator mais decisivo foi a path information. Modelos que incluíam caminhos de jogador atingiram aproximadamente o dobro de desempenho em relação aos que não incluíam. Isso também levou a um salto enorme na taxa de níveis jogáveis: 97% dos níveis gerados eram completáveis por um agente A*, superando não só outros métodos de aprendizado de máquina (melhor anterior = 66%), mas também o melhor sistema humano na competição Mario AI 2011 (94%).

- O depth information, isolado, piorava a performance, mas combinado com path e snaking resultou no melhor modelo (Snaking + Path + Depth), que atingiu o menor erro (NLL = 0.0177). Esse modelo também foi o que mais se aproximou das propriedades estatísticas das fases originais.

- O snaking teve efeito moderado, ajudando principalmente em combinação com as outras técnicas, já que reduz a distância sequencial entre tiles relacionados (ex.: lados de um cano).

Além da métrica de log-likelihood, os autores avaliaram as fases em termos de características clássicas de design: linearidade, leniência, porcentagem de espaço vazio, quantidade de saltos, entre outras. Os níveis gerados pelo melhor modelo se alinharam muito bem ao espaço expressivo das fases originais, tanto estatisticamente quanto visualmente (incluindo padrões típicos como séries de canos, plataformas suspensas e estruturas piramidais).

Conclusão

O estudo demonstra que LSTMs podem aprender a estrutura de fases completas de Mario a partir de exemplos relativamente poucos (39 fases) e gerar conteúdo novo que mantém a jogabilidade e a variedade estilística. A introdução de informação de caminho se mostrou a chave para garantir a jogabilidade, enquanto o depth ajudou a capturar progressão global do nível quando combinado com outras técnicas.

Level Generation Through Large Language Models

Este trabalho olha se Large Language Models (LLMs), como GPT-2 e GPT-3, conseguem gerar níveis jogáveis de videogames (Sokoban). A motivação é que, embora esses modelos tenham mostrado grande capacidade em linguagem natural, código e outras tarefas, sua aplicação a dados espaciais estruturados, como fases de jogos, não é direta.

Os autores destacam dois desafios principais. Primeiro, o representacional: LLMs trabalham linearmente com tokens, enquanto níveis são estruturas bidimensionais. Isso exige transformar níveis em sequências de caracteres, preservando coerência espacial. Segundo, o desafio de dados: LLMs precisam de grandes quantidades de informação, mas datasets de fases são pequenos e heterogêneos. Apesar disso, os LLMs oferecem vantagens de

controlabilidade via prompts em linguagem natural e generalização para múltiplos domínios de jogo.

O paper treina modelos em Sokoban, usando dois datasets: Microban (282 fases humanas resolvíveis) e Boxoban (438.000 fases procedurais 10x10 com 4 caixas e 4 metas). As fases são convertidas em strings lineares e tokenizadas. O modelo base é GPT-2, com experimentos também envolvendo GPT-3.

A avaliação é feita por quatro métricas:

- Playability: se o nível é válido e solucionável por um agente A*.
- Novelty: se difere de todos os níveis do dataset (via edit distance).
- Diversity: se os níveis gerados diferem entre si (via cliques em grafo de edit distance).
- Accuracy: em experimentos de controle, se o nível gerado corresponde ao prompt (ex.: solução próxima ao valor desejado).

Os experimentos avaliaram quatro pontos: pré-treinamento, tamanho do dataset, controlabilidade por prompts e GPT-3.

- Pré-treinamento: GPT-2 padrão, em Java e sem pré-treinamento tiveram desempenho similar (score ≈0.55). O modelo sem pré-treinamento foi até ligeiramente melhor, indicando que experiência prévia em linguagem natural ou código não influencia na geração de Sokoban.
- Dataset: quanto mais dados, melhor o desempenho. Com Boxoban completo o modelo gerou níveis novos e jogáveis com alta taxa; já com 0.1% ou Microban, produziu níveis novel ou jogáveis, mas raramente ambos. Isso mostra que os transformers escalam bem com dados, mas sofrem em cenários pequenos.
- Controlabilidade: prompts sobre espaço vazio funcionaram (alta precisão), mas prompts sobre comprimento da solução falharam (desempenho próximo ao acaso). Importante: os prompts nunca reduziram a taxa de jogabilidade.
- GPT-3: mesmo com datasets pequenos (Microban + aumentações), gerou níveis comparáveis ao GPT-2 treinado no Boxoban completo, sugerindo melhor generalização e benefício do aumento de dados.

Referências

[1] Kuang P, Luo D (2020) Conditional convolutional generative adversarial networks based interactive procedural game map generation. In: Future of Information and Communication

Conference, Springer, pp 400–419

[2] Torrado RR, Khalifa A, Green MC, Justesen N, Risi S, Togelius J (2019) Bootstrapping conditional gans for video game level generation. arXiv preprint arXiv:191001603

[3] Wulff-Jensen A, Rant NN, Møller TN, Billeskov JA (2017) Deep convolutional generative adversarial network for procedural 3d landscape generation based on dem. In: Interactivity, Game Creation, Design, Learning, and Innovation, Springer, pp 85–94

[4] Giacomello E, Lanzi PL, Loiacono D (2018) Doom level generation using generative adversarial networks. In: 2018 IEEE Games, Entertainment, Media Conference (GEM), IEEE, pp 316–323

[5] Park K, Mott BW, Min W, Boyer KE, Wiebe EN, Lester JC (2019) Generating educational game levels with multistep deep convolutional generative adversarial networks. In: 2019 IEEE Conference on Games (CoG), IEEE, pp 1–8

[6] Volz V, Justesen N, Snodgrass S, Asadi S, Purmonen S, Holmgård C, Togelius J, Risi S (2020) Capturing local and global patterns in procedural content generation via machine learning. In: Proceedings of The 2020 IEEE Conference on Games (CoG)

[7] Awiszus M, Schubert F, Rosenhahn B (2020) Toad-gan: Coherent style level generation from a single example. In: Proceedings of the Sixteenth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2020)

[8] Di Liello L, Ardino P, Gobbi J, Morettin P, Teso S, Passerini A (2020) Efficient generation of structured objects with constrained adversarial networks. arXiv preprint arXiv:200713197

[9] Fontaine MC, Liu R, Togelius J, Hoover AK, Nikolaidis S (2020) Illuminating Mario scenes in the latent space of a generative adversarial network. arXiv preprint arXiv:200705674

[10] Kumaran V, Mott BW, Lester JC (2020) Generating game levels for multiple distinct games with a common latent space. In: Proceedings of the Sixteenth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2020)

[11] Hong S, Kim S, Kang S (2019) Game sprite generator using a multi discriminator gan. KSII Transactions on Internet & Information Systems 13(8)

[12] Wang T, Kurabayashi S (2020) Sketch2map: A game map design support system allowing quick hand sketch prototyping. In: Proceedings of The 2020 IEEE Conference on Games (CoG)

[13] Bontrager P, Togelius J (2020) Fully differentiable procedural content generation through

generative playing networks. arXiv preprint arXiv:200205259

[14] Lucas SM, Volz V (2019) Tile pattern kl divergence for analysing and evolving game levels. In: Proceedings of the Genetic and Evolutionary Computation Conference, Association for Computing Machinery, New York, NY, USA, GECCO '19, p 170–178

[15] Isaksen A, Holmgård C, Togelius J (2017) Semantic hashing for video game levels. Game & Puzzle Design 3(1):10–16

[16] Guzdial MJ, Sturtevant N, Li B (2016) Deep static and dynamic level analysis: A study on infinite mario. In: Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference

[17] Min W, Ha EY, Rowe J, Mott B, Lester J (2014) Deep learning-based goal recognition in open-ended digital games. In: Tenth Artificial Intelligence and Interactive Digital Entertainment Conference

[18] Karavolos D, Liapis A, Yannakakis G (2017) Learning the patterns of balance in a multi-player shooter game. In: Proceedings of the 12th International Conference on the Foundations of Digital Games, pp 1–10

[19] Karavolos D, Liapis A, Yannakakis GN (2018) Pairing character classes in a deathmatch shooter game via a deep-learning surrogate model. In: Proceedings of the 13th International Conference on the Foundations of Digital Games, pp 1–10

[20] Gudmundsson SF, Eisen P, Poromaa E, Nodet A, Purmonen S, Kozakowski B, Meurling R, Cao L (2018) Human-like playtesting with deep learning. In: 2018 IEEE Conference on Computational Intelligence and Games (CIG), IEEE, pp 1–8

[21] Larsson S, Petri O (2016) Content evaluation of starcraft maps using neuroevolution. URL http://urn.kb.se/resolve?urn=urn:nbn:se:bth-11684

[22] Shaker N, Yannakakis G, Togelius J (2010) Towards automatic personalized content generation for platform games. In: Sixth Artificial Intelligence and Interactive Digital Entertainment Conference

[23] Shaker N, Togelius J, Yannakakis GN, Weber B, Shimizu T, Hashiyama T, Sorenson N, Pasquier P, Mawhorter P, Takahashi G, et al. (2011) The 2010 Mario AI championship: Level generation track. IEEE Transactions on Computational Intelligence and AI in Games 3(4):332–347

[24] Shaker N, Nicolau M, Yannakakis GN, Togelius J, O'neill M (2012) Evolving levels for

Super Mario Bros using grammatical evolution. In: Computational Intelligence and Games, IEEE, pp 304–311

[25] Camilleri E, Yannakakis GN, Dingli A (2016) Platformer level design for player believability. In: 2016 IEEE Conference on Computational Intelligence and Games (CIG), IEEE, pp 1–8

[26] Summerville A, Mariño JR, Snodgrass S, Ontañón S, Lelis LH (2017) Understanding Mario: An evaluation of design metrics for platformers. In: Proceedings of the 12th International Conference on the Foundations of Digital Games, pp 1–10

[27] Pfau J, Liapis A, Volkmar G, Yannakakis GN, Malaka R (2020) Dungeons & replicants: Automated game balancing via deep player behavior modeling. In: Proceedings of the 2020 IEEE Conference on Games (CoG)

[28] Martinez HP, Bengio Y, Yannakakis GN (2013) Learning deep physiological models of affect. IEEE Computational intelligence magazine 8(2):20–33

[29] Makantasis K, Liapis A, Yannakakis GN (2019) From pixels to affect: A study on games and player experience. In: 2019 8th International Conference on Affective Computing and Intelligent Interaction (ACII), IEEE, pp 1–7

[30] Melhart D, Gravina D, Yannakakis GN (2020) Moment-to-moment Engagement Prediction through the Eyes of the Observer: PUBG Streaming on Twitch. In: Foundations of Digital Games

[31] Camilleri E, Yannakakis GN, Liapis A (2017) Towards general models of player affect. In: 2017 Seventh International Conference on Affective Computing and Intelligent Interaction (ACII), IEEE, pp 333–339

APÊNDICE 4

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 1 de out. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

Edward Scott Carvalho Johnson

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nas última Semana do processo, foi feito a leitura dos seguintes papers:

- Deep Learning for Procedural Content Generation
- Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network (GANs)
- Level Generation Through Large Language Models (LLMs)
- Super Mario as a String: Platformer Level Generation Via LSTMs (LSTMs)
- The VGLC: The Video Game Level Corpus (Dataset)

Nessa semana, foi feito a leitura dos seguintes papers:
- https://arxiv.org/pdf/2404.08706 (Game Generation via Large Language Models)
- https://arxiv.org/pdf/2507.00184 (Text-to-Level Diffusion Models With Various Text Encoders for Super Mario Bros)
- https://arxiv.org/pdf/2305.18243 (Practical PCG Through Large Language Models)
- https://arxiv.org/pdf/2302.05981 (MarioGPT: Open-Ended Text2Level Generation through Large Language Models)

Eles abordam PCG de forma "promptable", onde os papers são diversos, tendo:

- Um paper que usa modelos de difusão para geração de fases
- Um paper que faz fine-tunning no DistilGPT2 para geração de fases
- Um paper que usa o GPT4 e outros modelos grandes para não só gerar fases, mas também gerar as regras do jogo também, fazendo algo do zero
- Geração de fases via fine-tunning, porém feito via bootstrapping e com uma quantidade de fases extremamente limitada

Todos os papers trouxeram informações muito úteis, e ao ler todos foi perceptível que:

- O VGLC é o único dataset público que realmente é possível fazer experimentações sobre, e os papers que usam ele geralmente adaptam ele para aumentar a qualidade dos dados.
- A quantidade de dados existentes é extremamente baixo, e os papers sempre possuem um foco grande em como superar esse problema
- A geração geralmente é feito em pequenos pedaços de fases, é difícil gerar uma fase inteira de

maneira que tenha qualidade. Com LLMs é mais fácil, com difusão é mais difícil.

Planejamento: **[descrever o que pretende fazer para realizar a próxima ENTREGA]**

Para a próxima Semana da residência, planejo:
- Pesquisar mais sobre métodos de avaliação de modelos de PCG
- Aproveitar bastante os papers e apresentações sobre PCG aqui na SBGames, que está ocorrendo de forma paralela ao SVR

Observação: **[caso precise fazer alguma observação, de qualquer "natureza"]**

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

APÊNDICE 5

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 8 de out. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

EDWARD SCOTT CARVALHO JOHNSON

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nas primeiras 4 Semanas do processo foi feito:

- Decisão do estudo do tema Procedural Content Generation (PCG)
- Estudo dos algoritmos clássicos e métodos de ML
- Estudo de algoritmos de deep learning (GANs, LSTMs, etc)

Na quinta Semana do processo foi feito:

- Leitura de modelos "promptable" de PCG (LLMs, Diffusão)
- Realização de algumas coisas:
  - PCG tem poucos dados disponíveis então papers que não usam fases sintéticas têm uma tendência de "overfitar", e as que usam tem geram fases estruturalmente corretas, porém perdem semelhança com o jogo original
  - Existem "linguagens" que pessoas tentam usar para uma LLM gerar um jogo completo, junto com suas regras de jogo, código, etc
  - Geração quase nunca é de uma fase inteira, mas de poucos recortes de fases

Na sexta Semana do processo foi feito:

- Leitura dos seguintes papers:
  - SBGAMES: Customizable Procedural Content Generation with LLMs
  - SBGAMES: Machine Learning for Playable Room Generation: A Modular Approach with VAE and PCG

- Análise de quais métricas são usadas e como são implementadas nos seguintes papers:
  - Understanding Mario: An Evaluation of Design Metrics For Platformers
  - Text-to-Level Diffusion Models With Various Text Encoders for Super Mario Bros
  - Practical PCG Through Large Language Models
  - MarioGPT: Open-Ended Text2Level Generation through Large Language Models
  - Super Mario as a String: Platformer Level Generation Via LSTMs
  - Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network
  - Level Generation Through Large Language Models

- Resumo e anotações dos papers em relação a métricas: Gate 06 - Métricas, SBGames
- Métricas em comum em todos os papers e parecem ser o “padrão” são jogabilidade, novelty, diversidade e prompt adherence (para os “promptable” especificamente)
- Planejo utilizar do que já fiz no meu trabalho final da matéria de NLP para avançar de agora em diante na residência. Com a leitura de papers, pensei em inúmeras coisas para testar e fazer que eu nem sequer conseguiria pensar em fazer antes do processo começar.

Planejamento: **[descrever o que pretende fazer para realizar a próxima ENTREGA]**

Abrir o código do trabalho de NLP e verificar que tudo está rodando normalmente:
- Rodar o repositório do paper do MarioGPT (baseline)
- Juntar e documentar conclusões, conhecimentos e tudo que foi feito no trabalho que está “espalhado” em um único docs.

Observação: [caso precise fazer alguma observação, de qualquer “natureza”]

Agradecimento aos meus integrantes do grupo de trabalho final de NLP

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Documento de Apoio - Semana 6

SBGAMES: Customizable Procedural Content Generation with LLMs

O paper propõe o uso de um modelo de linguagem ajustado DeepSeek-R1-Distill-Llama-8B para gerar fases 2D no estilo The Legend of Zelda a partir de prompts. O modelo foi fine-tunado com 1000 níveis do jogo Zelda do GVGAI.

Os resultados mostram que o modelo gera fases altamente jogáveis (80-89%), muito originais (90% novas em relação ao conjunto de treino) e diversas entre si. A principal limitação está na precisão de aderência às instruções, que ficou em torno de 62,8%. Essa queda de acurácia ocorre principalmente em casos que envolvem equilíbrio entre múltiplos elementos, por exemplo, quando o prompt pede “muitos inimigos e alguns blocos”, o modelo tende a errar na proporção frequentemente omitindo blocos ou gerando menos do que o esperado. Isso indica que o modelo entende as relações gerais entre os elementos, mas tem dificuldade em controlar quantidades relativas quando há mais de uma restrição ativa.

SBGAMES: Machine Learning for Playable Room Generation: A Modular Approach with VAE and PCG

O paper cria um sistema que combina geração procedural clássica com aprendizado de máquina para criar salas jogáveis e equilibradas em jogos roguelike. A abordagem utiliza Autômatos Celulares para gerar salas iniciais e um Variational Autoencoder para refinar e otimizar os layouts com base em critérios definidos pelo usuário. O conjunto de dados inicial contém 500 salas, das quais 152 válidas foram usadas para treinar o modelo.

Os resultados mostram que o VAE gera mais de 80% de salas válidas, com melhor conectividade, acessibilidade e diversidade estrutural que o método puramente procedural, além de reduzir o tempo de geração em 58% (0,05 s por sala o qual acho negligenciável). Em testes com jogadores experientes, 85% preferiram ou consideraram equivalentes as salas geradas pelo VAE. Apesar disso, o sistema apresenta limitações principalmente o risco de overfitting devido ao pequeno número de exemplos válidos e a simplificação dos elementos (apenas paredes, inimigos, itens e jogador).

Text-to-Level Diffusion Models With Various Text Encoders for Super Mario Bros

Prompt Adherence: Discordo com a parte onde ele dá um score negativo quando o prompt não menciona uma classe mas ele acaba gerando a classe. Sinto que nesse caso é só dar um prompt negativo se você não quiser ele. Forçar o modelo a sempre gerar exatamente as classes que você quer é prejudicial especialmente quando tem muitas classes. No caso deles tem menos que eu.

Formula:

$$c(p, c) = \frac{1}{|T|} \sum_{t \in T} \text{match}(\text{phrase}(t, p), \text{phrase}(t, c))$$

match(pt, ct) rules

Case	Description	Score
Exact match	Same phrase	**1.0**
Countable, different quantities	(1 - \dfrac{	qu(pt)-qu(ct)
Full/uncountable vs counted	e.g., “full floor” vs “two gaps”	**0.1**
Presence ↔ absence conflict	one says “no X”, other doesn’t	**−1.0**
Both absent	both ∅ or “no X”	**1.0**

Countable-quantity difference scores

	one (0)	two (1)	a few (2)	several (3)	many (4)
one (0)	1.00	0.75	0.50	0.25	0.00
two (1)	0.75	1.00	0.75	0.50	0.25
a few (2)	0.50	0.75	1.00	0.75	0.50
several (3)	0.25	0.50	0.75	1.00	0.75
many (4)	0.00	0.25	0.50	0.75	1.00

Average Minimum Edit Distance: Eles fazem por tile distance (hamming distance por ser setado em 16x16). Eles fazem comparando com fases do dataset e fases geradas dentro de um “set” de gerações.
Solvability: Se a fase é possível zerar, usam o modelo A* de Robin Baumgarten.
Integridade: Verifica se estruturas que possuem múltiplas partes (canos, canhões) mantém integridade estrutural boa.

Practical PCG Through Large Language Models

Playability: Considerado jogável se é possível zerar a fase (mantém as regras impostas).
Novelty: Se a fase gerada é ao menos 10% diferente em nível de tile da sua fase mais similar no dataset de treino.
Accuracy: Acerto quando se trata do prompt, o quão fiel o output foi comparado a ele.

MarioGPT: Open-Ended Text2Level Generation through Large Language Models

Tile Reconstruction: Verifica se o modelo consegue reconstruir tiles da fase (non-air) e caminhos do jogador (o modelo também é treinado sobre os caminhos dos jogadores.
Playability: Se a fase é possível zerar, usam o modelo A* de Robin Baumgarten.
Model vs Agent path: Faz o MAE no caminho gerado pelo modelo e pelo agente A* e compara a diferença entre o caminho deles com fases jogáveis, fases não jogáveis e todos.
Prompt Adherence: Aderência ao prompt direto sem pesos.
t-sne: Usado para comparar diversidade dos samples pelo path que o agente usa (nada a ver com a fase gerada em si)
KNN: Ver se o modelo memoriza. O engraçado é que eles nunca reportam nada específico, só falam que ele não memoriza em temperaturas mais altas mas a qualidade cai.

Super Mario as a String: Platformer Level Generation Via LSTMs

C — Completabiliy: % of levels solvable by a simulated A* agent.
e — Empty space: fraction of tiles that are empty.
n — Negative space: fraction of tiles that are empty **and reachable** by the player.
d — Decoration density: fraction of tiles that are “interesting” (not solid and not empty).
p — Path coverage: fraction of tiles that lie on the (near-optimal) player path.
l — Leniency: l = enemies + gaps - rewards (medida de dificuldade)
R^2 — Linearity: goodness-of-fit (coefficient of determination) of a linear model to the level’s elevation profile.
j — Jumps: number of jumps along the optimal path.
ji — Meaningful jumps: subset of jumps forced by an enemy or a gap.

Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network

Playability: Se a fase é possível de completar.
“Prompt Adherence”: Dado a função de fitness, o quão perto ele estava de seguir ele.

Level Generation Through Large Language Models

Playability: Se a fase é possível de completar
Prompt Adherence: O quanto o prompt condiz com a fase
Novelty: Novelty é alcançado com k = 5 de diferença no dataset
Diversity: O quão diverso as fases são entre eles mesmos (mesmo threshold de 5)

Understanding Mario: An Evaluation of Design Metrics For Platformers

O artigo investiga como métricas simples e interpretáveis podem prever como humanos percebem dificuldade, estética visual e diversão em fases do jogo *Infinite Mario Bros*. Para isso, os autores coletaram mais de 2.700 avaliações humanas de 1.437 fases e calcularam 89 métricas de design, incluindo medidas estruturais (linearidade, leniência e densidade), distribuição e quantidade de inimigos e recompensas, simetria, equilíbrio visual e espaços vazios. Usando regressão LASSO multinomial, o modelo selecionou apenas as métricas mais relevantes para cada atributo: por exemplo, o número de inimigos foi o principal fator para dificuldade e diversão, enquanto posição de power-ups e simetria foram mais importantes para estética.

O estudo mostra que a dificuldade é determinada quase totalmente pela quantidade e distribuição dos inimigos. Mais inimigos tornam a fase mais difícil, enquanto uma maior separação horizontal facilita e uma variação vertical aumenta o desafio. Menos pulos também indicam menor dificuldade. Já a estética depende da coerência e acessibilidade do design. Power-ups bem posicionados e alcançáveis, espaços vazios moderados e simetria tornam a fase visualmente mais agradável e equilibrada.

A diversão surge do equilíbrio entre desafio (inimigos) e recompensa (power-ups). Inimigos bem espaçados, recompensas acessíveis e ritmo entre obstáculos e áreas abertas tornam o jogo mais envolvente. No geral, poucas métricas simples, densidade de inimigos, posicionamento de power-ups, alcançabilidade e simetria explicam a maior parte da percepção humana de dificuldade, estética e diversão.

APÊNDICE 6

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 15 de out. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

EDWARD SCOTT CARVALHO JOHNSON

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nas primeiras 6 Semanas do processo foi feito:

- Decisão do estudo do tema Procedural Content Generation (PCG)
- Estudo dos algoritmos clássicos e métodos de ML
- Estudo de algoritmos de deep learning (GANs, LSTMs, etc)
- Estudo de modelos do estilo "promptable" (LLMs, Diffusion)

Na sétima Semana do processo foi feito:

- Execução do código disponibilizado dos autores do paper "MarioGPT"
- Execução do código que eu estava pronto dado o meu trabalho na matéria de NLP. As seguintes coisas foram feitas na matéria:
  - Elaboração do dataset de vários jogos do Mario (37 fases do VGLC para 112) com um aumento de classes para representar os jogos do Mario da melhor forma possível (13 classes para 26). Dado um "janelamento" que é feito pro treino, a quantidade de fases pula para 17k samples de treino
  - Código de Fine Tuning via Unsloth
  - Código de inferência para os modelos gerados
  - Fork de um repositório que permite jogar as fases criadas
  - Código com algumas métricas para verificar qualidade das fases gerados
  - Código de evolução de prompts (não foi rodado sobre o dataset de fases todo)

- Elaboração do documento com tudo que já tinha feito: Gate 07 - Checkpoint Progresso

- Realização de algumas outras tasks de coisas que eu queria fazer:
  - Adaptar dataset para usar outra forma de representar os dados (ideia do paper de LSTM do Adam Summerville)
  - Criar métrica de integridade para ver se os objetos gerados que ocupam mais de um caracter estão corretamente formados
  - Após estudar métodos "bons de pegar" plágio de fases no output (overfitting), implementei uma métrica de n-gram que usa similaridade de Jaccard. Essa métrica é usada bastante para plágio por ser robusto contra pequenas modificações locais.
  - Outras "coisas" menores

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Planejo continuar a fazer as mudanças inspiradas nos papers que eu li, como:

- Deixar a métrica de aderência ao prompt mais robusto
- Testar tokens de fase “single tile” ao invés os merges do byte pair encoding normal
- Anotar as fases que o agente A* não conseguiu terminar para ter o caminho do jogador de todas as fases do dataset novo

Observação: [caso precise fazer alguma observação, de qualquer “natureza”]

ACEITE DA ENTREGA:

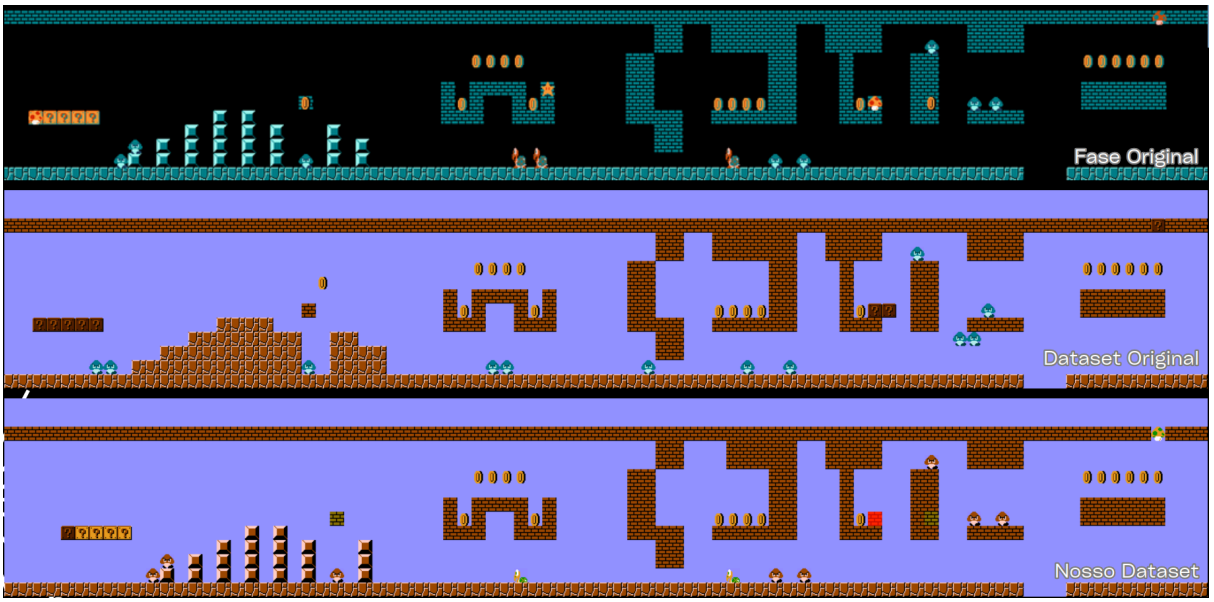
CEDRIC LUIZ DE CARVALHO: Go!

Documento de Apoio - Semana 7

Material Desenvolvido Anteriormente

A maior contribuição para a área que tenho no momento é o dataset. As maiores diferenças são:

- O dataset é mais fiel ao ground truth, completamente revisado contra erros, ao contrário do VGLC



- Relacionado ao ponto anterior, o dataset do VGLC tem 13 classes, enquanto o dataset que montamos tem 26

- O número de fases do VGLC vai de 37 para 112, então a quantidade de dados escala com a quantidade de classes, evitando pouca representatividade de cada classe. Quando eu separo as fases para o treino, a quantidade de fases vai de 7245 para 17071. Eu não faço o append de todas as fases para manter integridade, senão eu teria 22k samples.

- Um código de fine tuning via Unsloth e de algumas métricas comumente usadas (aderência a prompt, hamming distance, LCS)

Problemas do baseline (MarioGPT)

- O paper original tem um problema com plágio, e o paper não usa nenhuma métrica que consegue pegar essa memorização.

- Eles fazem append de todas as fases para criar uma fase gigante, o qual cria splits de fases que não existem nos jogos originais, trazendo ruído mesmo pequeno.

- Dataset que é anotado incorretamente, tem poucas classes

- Prompt é muito engessado, pares limitados de palavras que pode usar com 4 variações de adjetivos

Material a ser produzido

Imediato:

- [X] Tentar usar snaking para os dados
- [X] Usar **n-gram** como métrica para evitar que haja memorização e ver se isso mostra que o MarioGPT é overfitado
- [X] Usar integridade para ver qualidade dos canos/canhões, e também integridade da fase sendo gerada em si
- [X] Tentar criar uma métrica de jogabilidade onde todo objeto com física desce para a primeira posição de chão logo abaixo deles. Se não houver, o objeto “desaparece” do mapa
- [ ] Aderir a estratégia do paper de difusão quanto a prompt adherence, mas não penalizar a situação de não especificar uma classe no prompt porém ter ele aparecer.

Importante:

- [ ] Testar tokenizador com BPE e com single tile para ver diferenças
- [ ] Anotar o resto das fases que o agente A* não conseguiu terminar
- [ ] Também usar a métrica do Model vs Agent path (a diferença entre o path gerado e de um agente)

Opcional:

- Ver se o KNN realmente mostra algo sobre os dados
- Ver sobre diversidade entre as fases geradas, talvez não t-sne só para paths

APÊNDICE 7

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 22 de out. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

EDWARD SCOTT CARVALHO JOHNSON

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nas primeiras 7 Semanas do processo, foi feito:

- Decisão do estudo do tema Procedural Content Generation (PCG)
- Estudo dos algoritmos clássicos e métodos de ML
- Estudo de algoritmos de Deep Learning (GANs, LSTMs, etc)
- Estudo de modelos do estilo "promptable" (LLMs, Diffusion)
- Levantamento do que já tenho e do que quero fazer

Na oitava Semana do processo, foi feito:

- Anotação de todas as fases faltantes do dataset, um processo que demorou muito mais tempo do que eu achei que iria demorar. anotacaomario.mp4

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Planejo terminar as mudanças inspiradas nos papers que eu li:

- Deixar a métrica de aderência ao prompt mais robusto
- Testar tokens de fase "single tile" ao invés os merges do "byte pair encoding" normal

Dado o tempo mais longo da nossa Semana, planejo já fazer o Fine Tuning de alguns modelos e gerar resultados preliminares, rodar os códigos de avaliação e ver como ficam os modelos com as variações de modos de representar os dados.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 6 de nov. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

Edward Scott Carvalho Johnson

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nas primeiras 8 Semanas do processo, foi feito:

- Decisão do estudo do tema Procedural Content Generation (PCG)
- Estudo dos algoritmos clássicos e métodos de ML
- Estudo de algoritmos de Deep Learning (GANs, LSTMs, etc)
- Estudo de modelos do estilo "promptable" (LLMs, Diffusion)
- Levantamento do que já tenho e do que quero fazer
- Execução do código do repositório do paper do "MarioGPT"
- Anotação dos datasets para ter "pathing"
- Adição de algumas métricas adicionais

Na nona Semana do processo, foi feito:

- Criação de datasets para fine tunning de várias formas diferentes com o intuito de testar elas e chegar na "melhor" solução (60 variações de representação dos dados)
- Correção de alguns scripts que estavam problemáticos quando eu fazia fine tuning
- Elaboração de um código para treinar o modelo com cada tile sendo um token (resultados ficaram ruins, mas creio que a culpa é minha: Loss)
- Início dos fine tunings (minha casa tá muito quente e para "fine tunar" um modelo tá demorando uns 30% a mais)
    - Nunca vai dar tempo de rodar absolutamente todas as possibilidade para todos os modelos que quero testar, tem 60 datasets diferentes com 12 modelos diferentes com X parâmetros, mas já comecei a rodar
- ResultadosPreliminares

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a última Semana, planejo trazer os melhores resultados que eu conseguir e comparar com o MarioGPT base. Também quero ver se consigo resultados "legais" com o single tile.

Observação: **[caso precise fazer alguma observação, de qualquer "natureza"]**

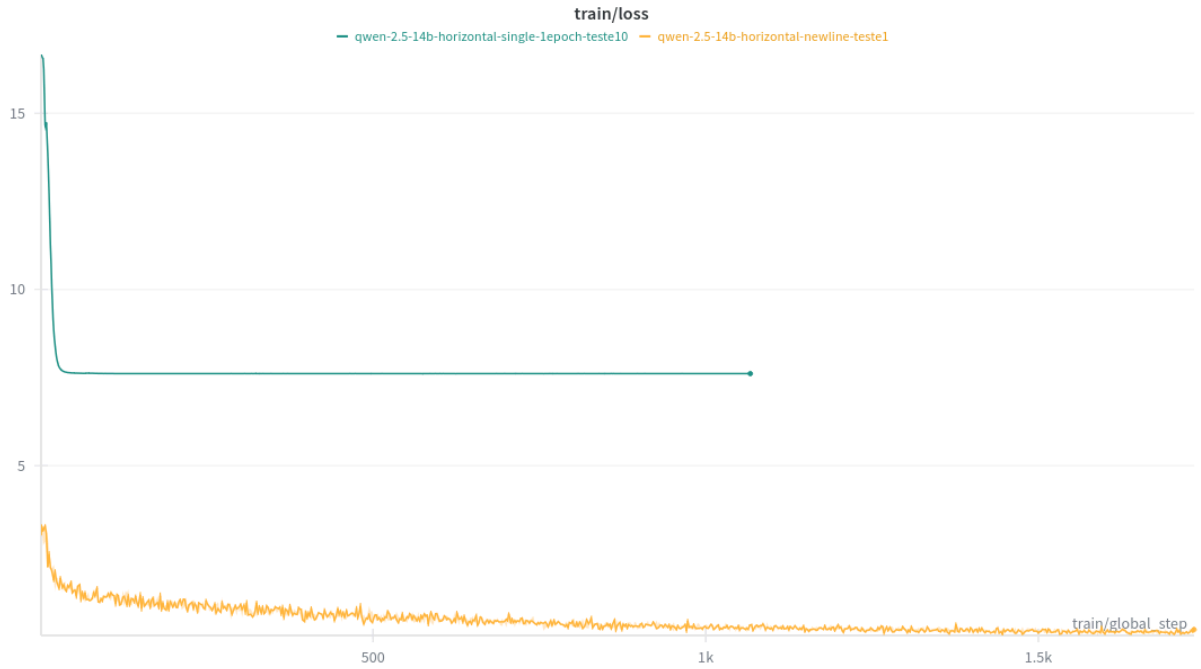
ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Documento de Apoio 1 - Semana 9

Loss

- Verde = Single token
- Amarelo = Tokenizador original



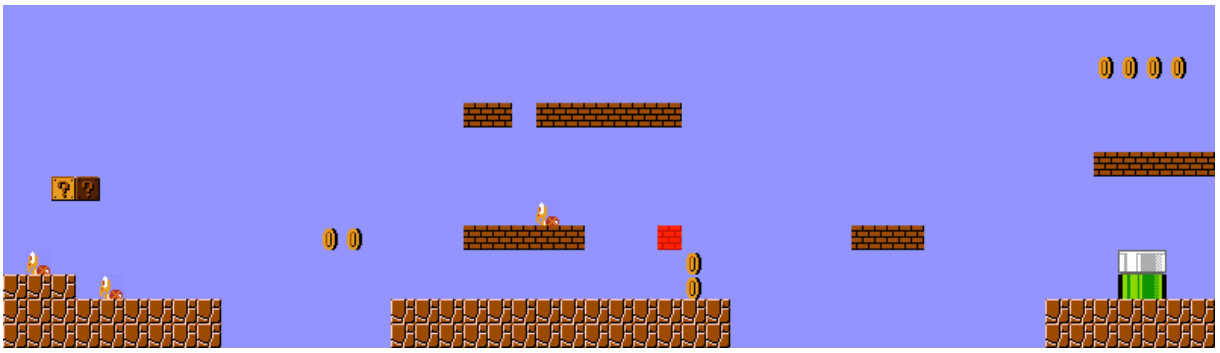
Documento de Apoio 2 - Semana 9

Prompt com Dataset Expandido:

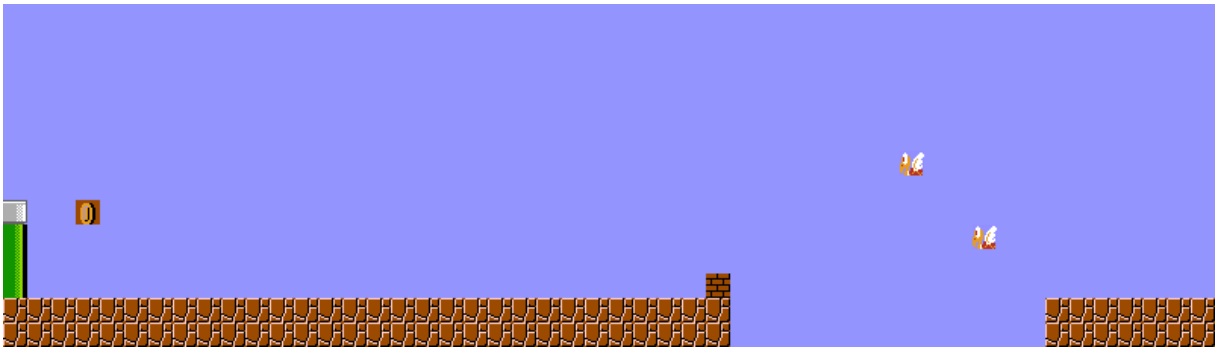
Prompt: Make me a level with some coins, some powerups, some koopas

Qwen 2.5 14B
Temperatura = 1.0

Fase gerada:



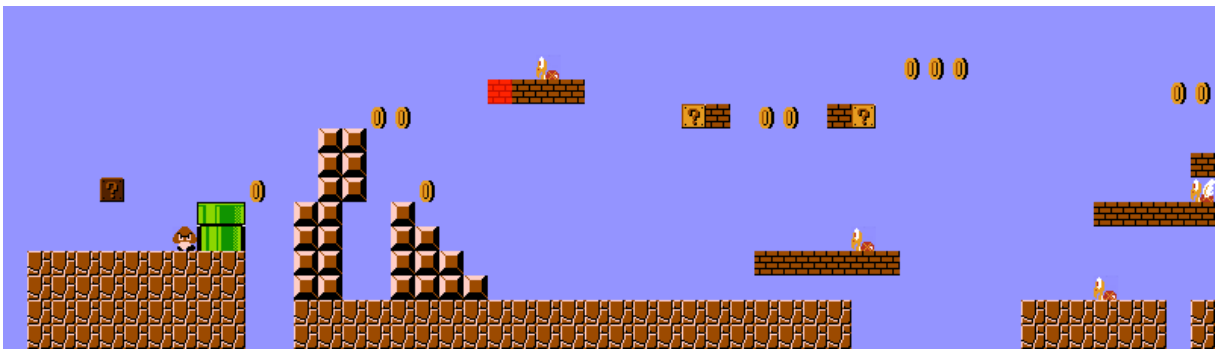
Fase mais parecida:



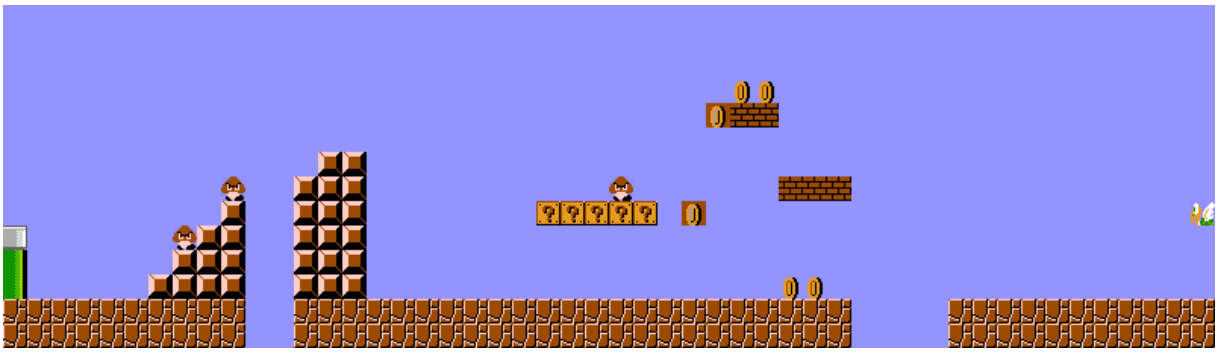
Distância de Hamming: 90%
Similaridade de Jaccard via Ngram: 7%
Prompt adherence: 89%

Temperatura = 2.0

Fase gerada:



Fase mais parecida:

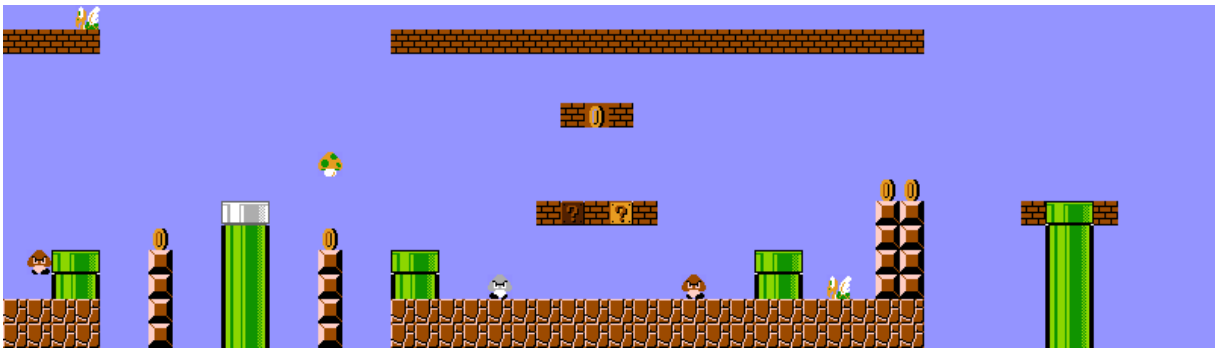


Distância de Hamming: 85%
Similaridade de Jaccard via Ngram: 4%
Prompt adherence: 89%

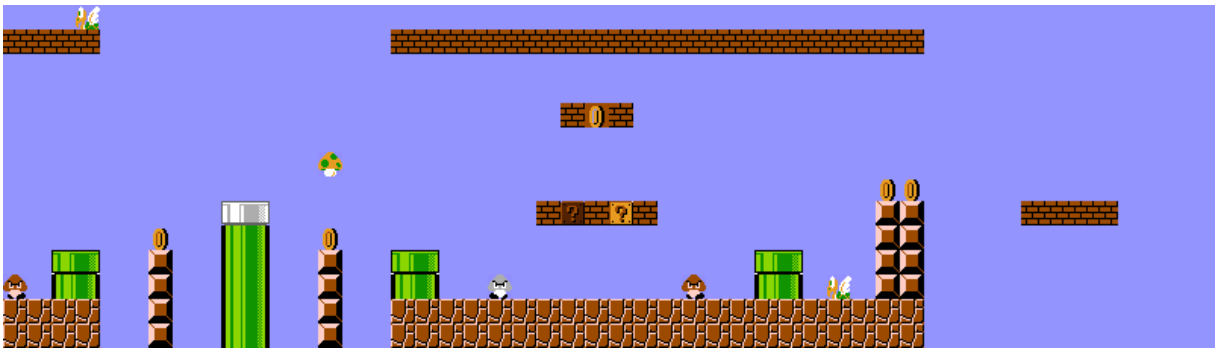
MarioGPT (dataset próprio)

Temperatura = 1.0

Fase gerada:

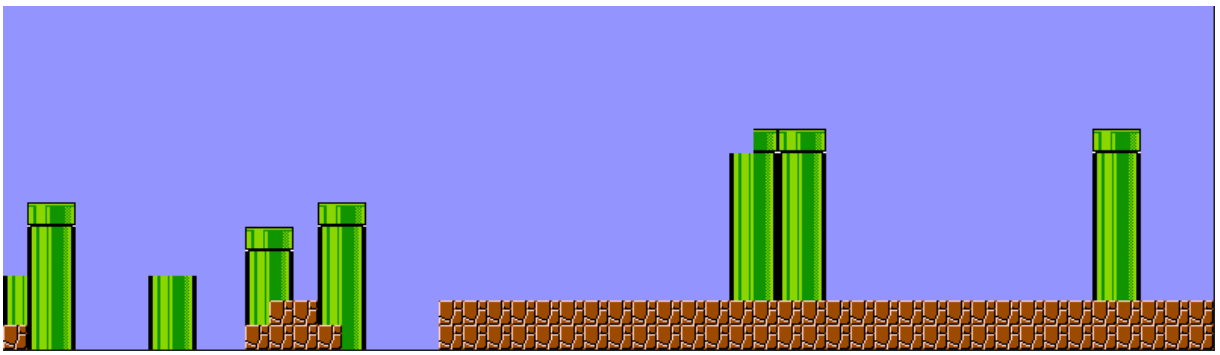


Fase mais parecida:



Distância de Hamming: 99%
Similaridade de Jaccard via Ngram: 84%
Prompt adherence: 100%

Temperatura = 2.0



Métricas e fase mais parecida não serão reportadas para essa fase por ser uma fase quebrada.

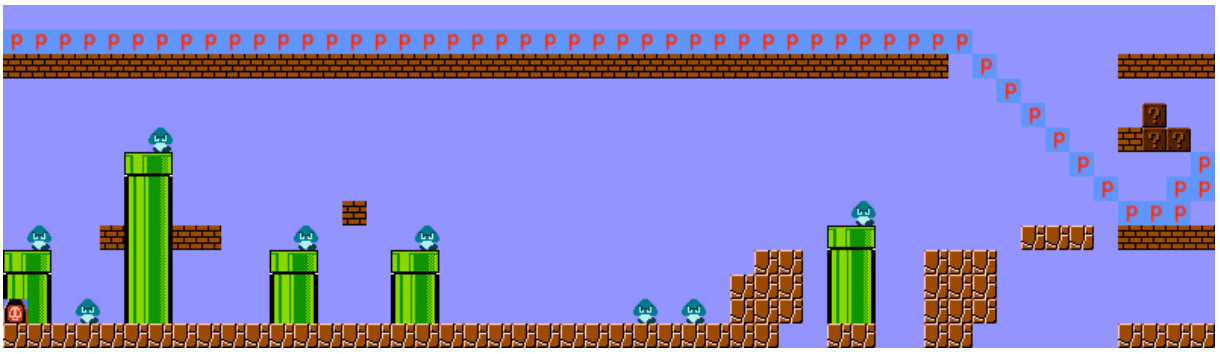
Prompt com Dataset simplificado do “MarioGPT”

Prompt: `["some pipes", "some enemies", "some blocks"]`

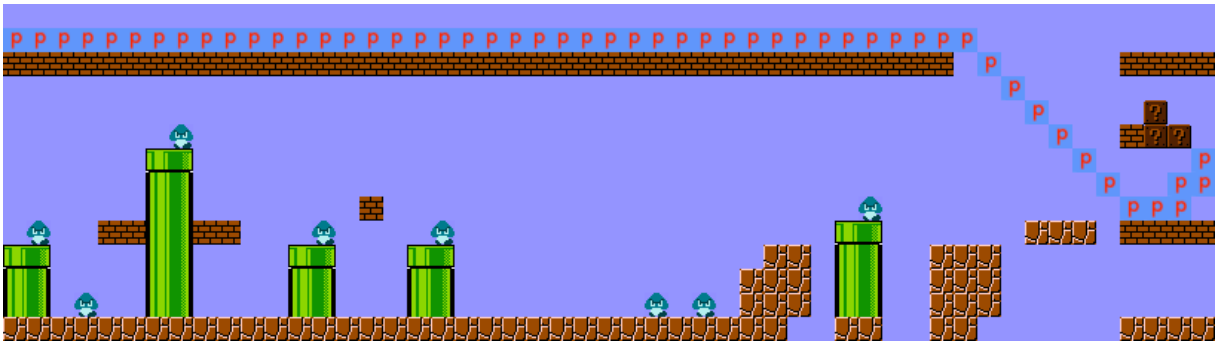
MarioGPT (versão do paper)

Temperatura = 2.0

Fase gerada:



Fase mais parecida:



Distância de Hamming: 89%
Similaridade de Jaccard via Ngram: 97%
Prompt adherence: 100%

APÊNDICE 8

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 13 de nov. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

Edward Scott Carvalho Johnson

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nas primeiras 9 Semanas do processo, foi feito:

- Decisão do estudo do tema Procedural Content Generation (PCG)
- Estudo dos algoritmos clássicos e métodos de ML
- Estudo de algoritmos de Deep Learning (GANs, LSTMs, etc)
- Estudo de modelos do estilo "promptable" (LLMs, Diffusion)
- Levantamento do que já tenho e do que quero fazer
- Execução do código do repositório do paper do "MarioGPT"
- Anotação dos datasets para ter "pathing"
- Adição de algumas métricas adicionais
- Criação dos datasets definitivos para os treinos

Na décima Semana do processo, foi feito:

- Execução dos fine tuning
  - Análise de resultados intermediários
  - Perceber de que alguma coisa estava errada
  - Correção do que estava errado
  - Reinicia o ciclo

- Não consegui fazer fine tunings o suficiente para tirar conclusões boas, mas foi possível ter alguns resultados:
  - ResultadosFinaisResidência

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

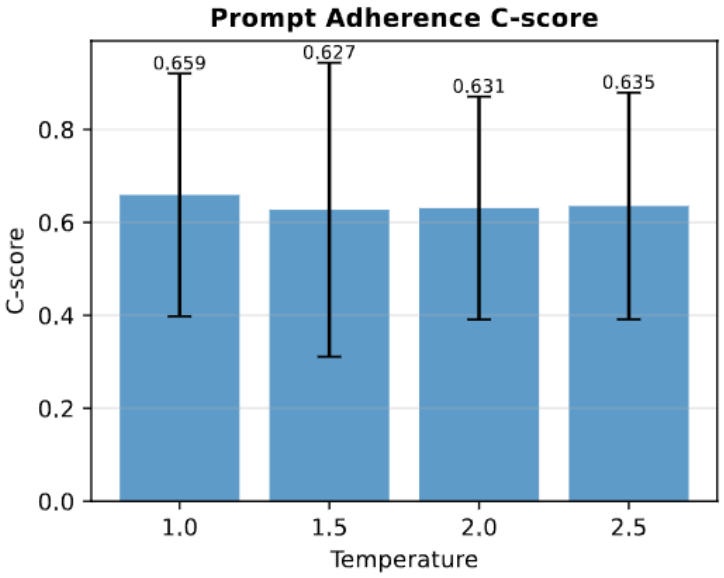
ACEITE DA ENTREGA:

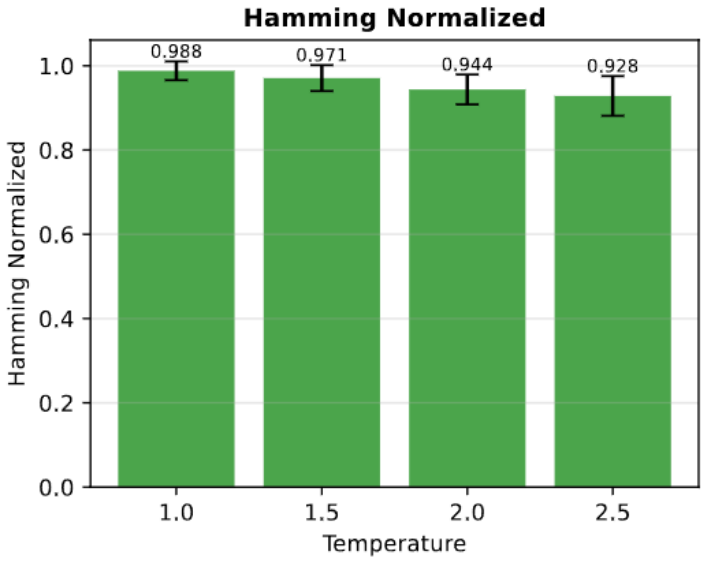
CEDRIC LUIZ DE CARVALHO: Go!

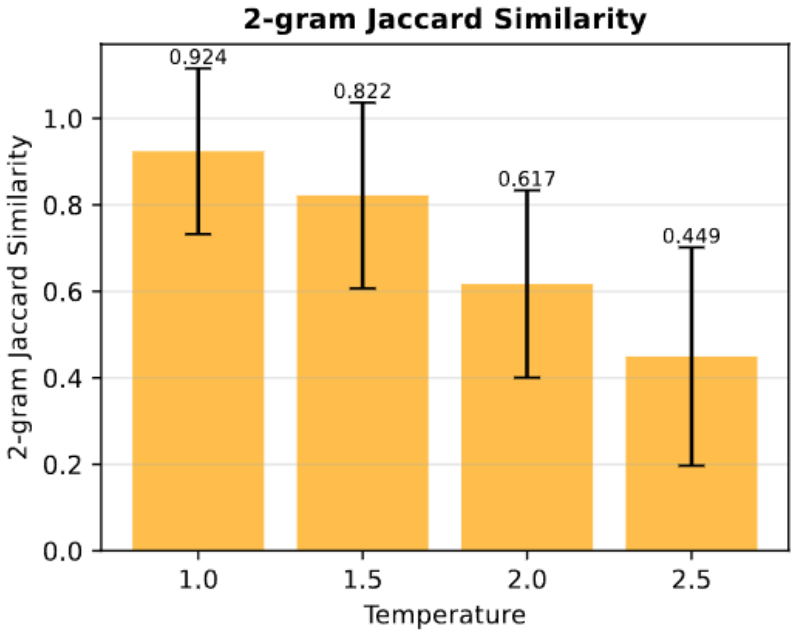
Documento de Apoio - Semana 10

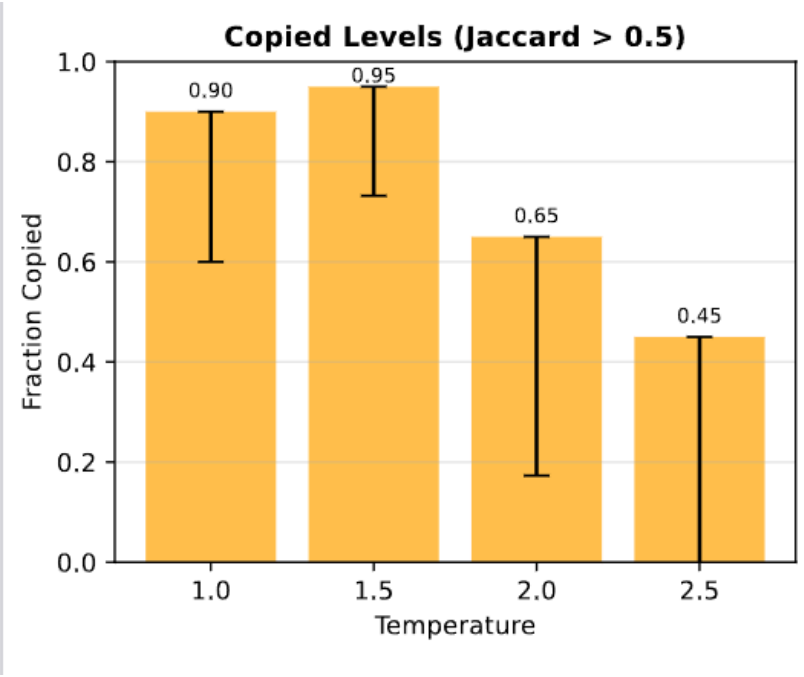
Resultados da Residência

MarioGPT Original:



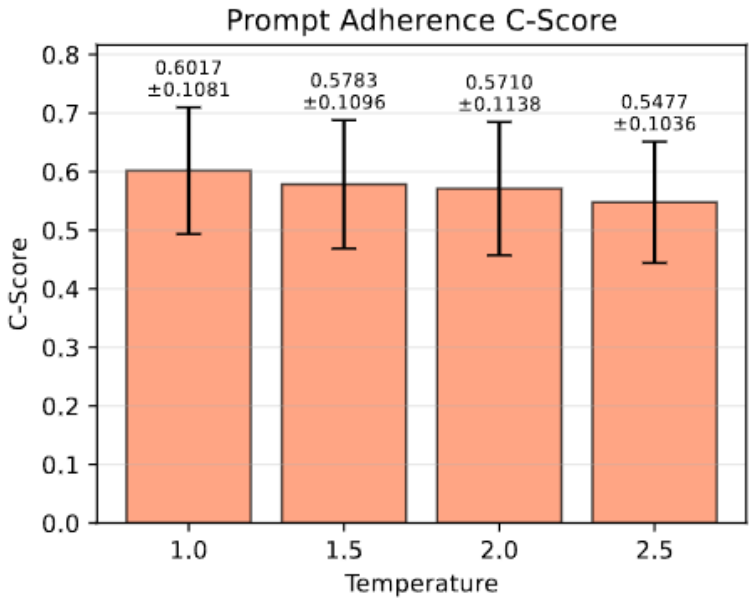


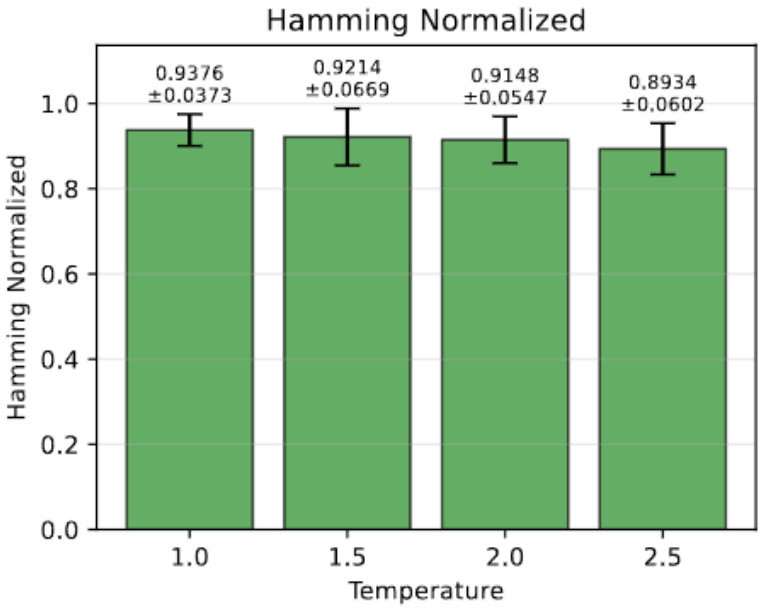

2-gram Jaccard Similarity
0.924
0.822
0.617
0.449
1.0
0.8
0.6
0.4
0.2
0.0
2-gram Jaccard Similarity
1.0
1.5
2.0
2.5
Temperature

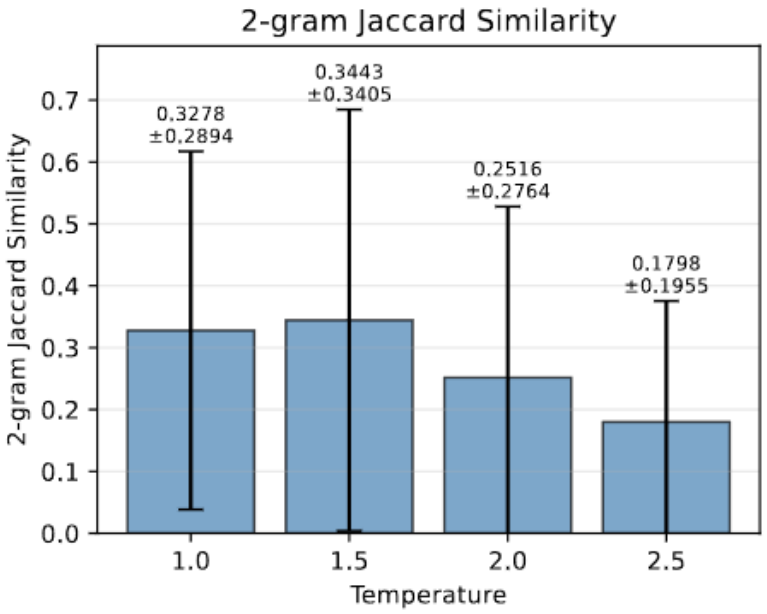

Copied Levels (Jaccard > 0.5)
0.90
0.95
0.65
0.45
1.0
0.8
0.6
0.4
0.2
0.0
Fraction Copied
1.0
1.5
2.0
2.5
Temperature

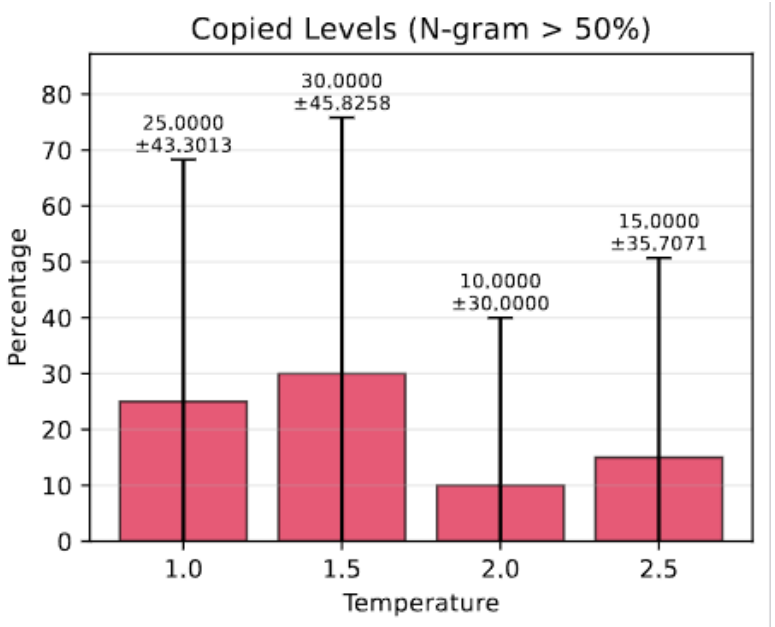
Meus Modelos:

Modelo: Qwen 2.5 14B (quantizado em INT4)
Fine-tuning com LoRA, rank 128, duas épocas
Dataset sem usar caminho, snaking na vertical









Conclusões:

Mesmo sem testar outros modos de fazer o fine-tuning como usar datasets que não colocam todas as classes no prompts, prompts expandidos, path do jogador, etc, resultados iniciais demonstram um potencial de melhora futura muito grande. Métricas de jogabilidade não estão representadas aqui, porém de forma qualitativa as fases do “MarioGPT” quando são originais geralmente não são jogáveis, algo que acontece menos com o Qwen.

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182