**UNIVERSIDADE FEDERAL DE JUIZ DE FORA**
**INSTITUTO DE CIÊNCIAS EXATAS**
**PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

**Paulo Vinícius Moreira Dutra**

# Geração Procedural de Conteúdo utilizando Aprendizado por Reforço com *Design* de Iniciativa Mista e Entropia como *Feedback* de Recompensa

Juiz de Fora

2023

**Paulo Vinícius Moreira Dutra**

# Geração Procedural de Conteúdo utilizando Aprendizado por Reforço com *Design* de Iniciativa Mista e Entropia como *Feedback* de Recompensa

Dissertação apresentada ao Programa Pós-graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Dr. Raul Fonseca Neto

Coorientador: Dr. Saulo Moraes Villela

Juiz de Fora

2023

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Dutra, Paulo Vinícius Moreira Dutra.
Geração Procedural de Conteúdo utilizando Aprendizado por Reforço com *Design* de Iniciativa Mista e Entropia como *Feedback* de Recompensa / Paulo Vinícius Moreira Dutra. – 2023.
88 f. : il.

Orientador: Raul Fonseca Neto
Coorientador: Saulo Moraes Villela
Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Pós-graduação em Ciência da Computação, 2023.

1. Aprendizado por reforço. 2. Geração procedural de conteúdo. 3. Aprendizado de máquina. 4. Expressive Range. 5. Iniciativa mista. 6. Entropia. I. Neto, Raul Fonseca, orient. II. Villela Saulo Moraes, coorient. Título.

**Paulo Vinícius Moreira Dutra**

**Geração Procedural de Conteúdo utilizando Aprendizado por Reforço com Design de Iniciativa Mista e Entropia como Feedback de Recompensa**

Dissertação apresentada Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação.

Aprovada em 16 de fevereiro de 2023

BANCA EXAMINADORA

**Prof. Dr. Raul Fonseca Neto** - Orientador
Universidade Federal de Juiz de Fora

**Prof. Dr. Saulo Moraes Villela** - Coorientador
Universidade Federal de Juiz de Fora

**Prof. Dr. Heder Soares Bernardino**
Universidade Federal de Juiz de Fora

**Prof. Dr. Matheus Ribeiro Furtado de Mendonça**
Laboratório Nacional de Computação Científica

Juiz de Fora, 19/01/2023.

Documento assinado eletronicamente por **Raul Fonseca Neto**, **Professor(a)**, em 28/02/2023, às 11:17, 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 **Matheus Ribeiro Furtado de Mendonça**, **Usuário Externo**, em 02/03/2023, às 17:52, 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 **Heder Soares Bernardino**, **Professor(a)**, em 11/05/2023, às 20:30, 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 **Saulo Moraes Villela**, **Professor(a)**, em 12/05/2023, às 11:21, 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 **Paulo Vinicius Moreira Dutra**, **Usuário Externo**, em 15/05/2023, às 09:36, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no Portal do SEI-Ufjf (www2.ufjf.br/SEI) através do ícone Conferência de Documentos, informando o código verificador **1117710** e o código CRC **4607B494**.

## AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer minha esposa Samara e meus filhos que durante esse período me apoiaram e contribuíram para eu concluir esse trabalho. Sem o grande apoio deles não conseguiria atingir os meus objetivos. A todos da minha família que me ajudaram diretamente ou indireta e principalmente a minha mãe Celeste pelo apoio que obtive ao longo dos meus estudos. Em terceiro lugar aos meus orientados, Raul e Saulo, pelo apoio durante o período de ensinamentos e pelas dificuldades enfrentadas. Não posso deixar de agradecer aos meus amigos de trabalho, Diego, Gustavo, Jean e Marcus, que me deram apoio para continuar os meus estudos. Por fim, a todos os professores do Departamento de Ciência da Computação que ao longo do período das aulas derem o seu melhor durante o período pandêmico.

“Em algum lugar, algo incrível está esperando para ser descoberto”.
(CARL SAGAN).

**RESUMO**

Os jogos ao longo dos anos têm se tornado uma das principais formas de entretenimento no ambiente digital e são frequentemente utilizados em pesquisas que envolvem a área do aprendizado de máquina. Dentre as ramificações do aprendizado de máquina, temos o aprendizado por reforço que é comumente utilizado para treinar agentes a jogar jogos. Atualmente existem muitos jogos que utilizam métodos de geração procedural para gerar algum conteúdo com o objetivo de aumentar experiência do jogador. Recentemente, artigos acadêmicos buscam aproximar o aprendizado por reforço com a geração procedural de conteúdo em jogos. Este trabalho investiga como podemos aplicar a geração procedural de conteúdo com aprendizado por reforço e o design de iniciativa mista. Uma segunda questão discutida aqui é como podemos utilizar métricas para avaliar a diversidade dos cenários gerados. A proposta deste trabalho possui como ideia principal utilizar modelos de cenários fornecidos por um especialista humano em *level design*, para que os agentes de aprendizado por reforço o utilizem para criar cenários. Os níveis fornecidos pelo especialista são separados em segmentos ou blocos que são utilizados para compor novas estruturas de cenários. Também é proposto o uso de uma função de recompensa baseado na entropia como métrica para avaliar a diversidade dos cenários gerado pelos agentes de aprendizado por reforço. Inicialmente, treinamos o modelo proposto para três diferentes ambientes de jogos no estilo 2D *Dungeon crawlers*. Analisamos os resultados obtidos através do valor de entropia e demonstramos que o modelo proposto pode gerar uma ampla gama de novos níveis com uma diversidade de segmentos. Um segunda análise dos resultados é através de *expressive range*, para avaliar a expressividade dos níveis utilizando as métricas linearidade e leniência.

Palavras-chave: Aprendizado por reforço. Geração procedural de conteúdo. Aprendizado de máquina. Expressive Range. Iniciativa mista. Entropia.

**ABSTRACT**

Over the years, video games have become one of the main ways of entertainment in the digital environment and are often used in the machine learning research. In machine learning, we can find different branches, among them, reinforcement learning, which is commonly used to train agents to play games. Currently, there are many games that use procedural generation methods to generate some content in order to increase the player's experience. Recently, academic articles have approach the problem of procedural generation of content in games though reinforcement learning.

In this work, we investigate how we can approach procedural content generation with reinforcement learning and mixed-initiative design. A second question discussed here is how we can use metrics to evaluate the diversity of the generated level. Our proposal has as its main hypothesis to use scenario models, provided by an expert human in level design, for the reinforcement learning agents in order to generate new scenarios. The levels provided by the specialist are separated into segments or blocks that are used to compose the new scenario structures. Also, a new reward function based on the use of entropy was proposed to measure the diversity of the generated scenarios. Initially, we trained our model for three different 2D Dungeon crawlers game environments. We analyzed our results through the value of the entropy, and it shows that our approach can generate levels with a wide diversity of segments. A second analysis of the results is through expressive range, to evaluate the expressivity of the levels using linearity and leniency metrics.

Keywords: Reinforcement learning. Procedural content generation. Machine learning. Expressive Range. Mixed-initiative. Entropy.

## LISTA DE ILUSTRAÇÕES

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| IA | Inteligência Artificial |
| DL | Deep Learning |
| DQN | Deep Q-Network |
| DRL | Deep Reinforcement Learning |
| GAN | Generative Adversarial Network |
| MDP | Markov Decision Process |
| MLP | Multilayer Perceptron |
| GPC | Geração Procedural de Conteúdo |
| PG | Policy Gradient |
| PPO | Proximal Policy Optimization |
| RL | Reinforcement Learning |
| SB3 | Stable Baselines3 |
| SGA | Stochastic Gradient Ascent |
| TD | Temporal Difference |
| TRPO | Trust Region Policy Optimization |

## LISTA DE SÍMBOLOS

| | |
|---|---|
| $\mathbb{R}$ | Conjunto de números reais |
| $\epsilon$ | Probabilidade de executar uma ação randômica na política $\epsilon - greedy$ |
| $\gamma$ | Parâmetro referente a taxa de desconto |
| $\alpha$ | Parâmetro referente a taxa de aprendizagem |
| $s, s'$ | Estados do agente no ambiente |
| $a$ | Uma ação |
| $r$ | Recompensa |
| $A_t$ | Ação no tempo $t$ |
| $S_t$ | Estado no tempo $t$ |
| $R_t$ | Recompensa no tempo $t$ |
| $s \in S$ | Um estado que pertence a um conjunto de Estado |
| $a \in A$ | Uma ação que pertence a um conjunto de Ações |
| $\pi$ | Regra para tomada das decisões do agente (política) |
| $\pi(a\|s)$ | Probabilidade de executar uma ação $a$ no estado $s$ sobre uma política estocástica $\pi$ |
| $\mathbf{w}$ | Pesos de uma rede neural |
| $\boldsymbol{\theta}, \boldsymbol{\theta}_t$ | Vetor de parâmetros de uma política |
| $d$ | Dimensão, o número de componentes de $\mathbf{w}$ |
| $d'$ | Número de componentes de $\boldsymbol{\theta}$ |
| $\hat{v}(s, \mathbf{w})$ | Valor aproximado do estado dado vetor de peso $\mathbf{w}$ |
| $\hat{v}_{\mathbf{w}}(s)$ | Notação alternativa para $\hat{v}(s, \mathbf{w})$ |

## SUMÁRIO

# 1 INTRODUÇÃO

A popularização dos jogos ao redor do mundo tem se tornado parte em nossas vidas ao longo dos anos. O mercado de jogos possui pouco mais de 3 bilhões de pessoas que jogam videogames. Consequentemente, gerando uma receita no mercado global no total de $ 196,00 bilhões de dólares (NEWZOO, 2022).

Os jogos são desenvolvidos por diversos profissionais, dentre eles o *level designer*. O *level designer* é o profissional responsável pela criação dos mapas ou fases de um jogo. A modelagem dos níveis em jogos é um processo que consome tempo de projeto, prototipação e testes. Uma das formas de reduzir os custos e tempo de desenvolvimento é através das técnicas de Geração Procedural de conteúdo (GPC). A GPC é um método utilizado para gerar conteúdos para os jogos utilizando algoritmos. Alguns exemplos de GPC em jogos incluem, geração de níveis, músicas, sons, personagens e narrativas.

Geração procedural de conteúdo em videogames possui diversos benefícios e tem se tornado uma ótima alternativa para os *game designers* projetar e desenvolver jogos de forma rápida e com um custo baixo. Há diversos gêneros de jogos que se beneficiam de métodos de geração procedural, e cada um deles com suas características e desafios.

A geração procedural em jogos possui um aspecto interessante e traz diversidade e novas experiências para os jogadores. A GPC pode ser utilizada em conjunto com diversas outras técnicas visando criar um conteúdo coeso para os jogos de computadores.

Atualmente, técnicas de aprendizado de máquina são aplicadas em diversas pesquisas aplicada a jogos de computadores visando treinar agentes de inteligência artificial para poderem se comportar de forma autônoma em um jogo. Dentre as técnicas, o aprendizado por reforço (do inglês Reinforcement Learning, RL), é comumente utilizado para treinar agentes a jogar jogos. Diversos trabalhados acadêmicos têm obtido sucesso em treinar agentes a jogar jogos de videogame, tais como o Sonic the Hedgehog (SOUZA; MIRANDA; BERNARDINO, 2022) e (Mnih, V., Kavukcuoglu, K., Silver, D. et al., 2015) que implementou um novo algoritmo de RL, chamado de *Deep Q-network* (DQN) para jogar jogos do videogame Atari 2600. DQN une algoritmos de RL com *Deep Learning* (DL), surgindo assim, o que chamados de *Deep Reinforcement Learning* (DRL).

Neste trabalho, introduzimos um framework de design de iniciativa mista (*mixed-initiative*) para tratar o problema de GPC com RL. Um sistema de iniciativa mista, segundo (SHAKER; NELSON, 2016), em criação de jogos refere-se a combinação de algoritmos de GPC e designer humanos. Hoje em dia é muito comum uma aproximação de humanos e máquinas com o objetivo de realizar uma colaboração para tornar mais efetiva a criação de um processo de designer. Nosso trabalho utiliza agentes de RL para gerar níveis para três ambientes de jogos no estilo 2D *Dungeon crawlers* (Dungeon, MazeCoin e Zelda) via modelos fornecidos por um especialista humano em *level design*. Intitulamos

o framework de PCGRLPuzzle, sendo a união dos termos em inglês *Procedural Content Generation (PCG)*, RL e a palavra *Puzzle*.

Finalmente, em aplicações de RL, o sucesso do aprendizado dos agentes depende de diversos aspectos que podem impactar diretamente na eficiência dos dados treinados, por exemplo, conjunto de ações, estados e função de recompensa.

Neste trabalho, nós propomos um estado, conjunto de ações e diversas funções de recompensa em que são retornadas como feedback durante o treinamento, dentre elas a entropia. A entropia é utilizada aqui para mensurar a diversidade dos níveis gerados pelos agentes de RL. Através dela conseguimos medir a quantidade de informação o nível gerado possui, assim, podemos calcular uma recompensa apropriada de acordo com o valor da entropia.

## 1.1 MOTIVAÇÃO E OBJETIVOS

A união de jogos de computadores com aprendizado de máquina oferece diversos benefícios, uma vez que os jogos fornecem ambientes que podem ser manipulados e controlados pelos agentes de inteligência artificial. Diversos gêneros de jogos são utilizados em pesquisas acadêmicas para ensinar a agentes a jogar jogos, bem como aplicados a geração procedural de conteúdo. Entre esses gêneros, destacam-se os conhecidos como *Dungeon crawlers*, em que possui um personagem que deve percorrer por um labirinto, enfrentar vários monstros, resolver quebra-cabeças e coletar tesouros.

A abordagem proposta nesta dissertação é investigar se algoritmos de RL são eficazes na geração níveis para jogos no estilo 2D *Dungeon crawlers*, assim como são eficazes no treinamento de agentes a jogar jogos. Nosso segundo objetivo é sugerir métricas de avaliação para os níveis gerados, bem como o uso de modelos de cenários fornecidos por um especialista humano em *level design* como fonte de dados para treinamento dos agentes de RL.

Durante o aprendizado, diferentemente de outras aplicações atuais por aprendizado por reforço, nossa abordagem consiste em avaliar a diversidade do nível gerado por meio de uma função objetivo que retorna um valor de recompensa com base no desempenho do agente durante a construção do cenário. A função objetivo possui como finalidade verificar o número mínimo de objetos a serem gerados e a distância entre os elementos do jogo, como paredes, portas, inimigos e outros.

Embora haja diversos trabalhos relacionados ao uso de RL e GPC, como em (KHALIFA et al., 2020), (GISSLEN et al., 2021) e (SHU; LIU; YANNAKAKIS, 2021), ainda são recentes e cada pesquisa possui suas características e utiliza recursos diversificados. Entretanto, o campo ainda é relativamente novo, essa dissertação tentará abordar uma nova visão para a pesquisa.

## 1.2 PROBLEMA

Esta dissertação apresenta diversos agentes de RL que modela os níveis a partir de modelos fornecidos por um especialista humano em *level design*. Para cada jogo, o agente realiza a leitura dos modelos que será utilizado para compor os níveis. Nossa abordagem com aprendizado por reforço e geração procedural de conteúdo possui um grande conjunto de ações que devem interagir com o ambiente, chamados de segmentos. Os níveis são construídos a partir desses segmentos (blocos) que são similares a peças de quebra-cabeças, no qual o agente deve aprender a encaixar as peças corretamente, a fim de construir um nível completo e diversificado. A cada bloco encaixado, o agente recebe uma recompensa positiva ou negativa como *feedback*. Os seguintes problemas foram formulados para essa dissertação:

1. É possível treinar o agente de RL para gerar níveis para jogos no estilo 2D *Dungeon crawlers* a partir de modelos fornecidos por um especialista humano em *level design*?
2. Como mensurar a diversidade e qualidade do nível gerado pelos agentes de *level design*?

## 1.3 CONTRIBUIÇÕES

Esse trabalho apresenta as seguintes principais contribuições: a primeira contribuição é a formulação de um método em que agentes de RL são capazes de gerar novos cenários a partir de modelos fornecidos por um especialista humano em *level design*. Uma segunda contribuição visa o uso de uma métrica para avaliar a diversidade dos cenários gerados pelos agentes de RL baseado na entropia, bem como analisar a qualidade dos cenários via *expressive range*. Por fim, é apresentado um estudo de caso demonstrando a formulação de RL aproximando GPC e a métrica avaliando os cenários gerados.

## 1.4 ORGANIZAÇÃO

Essa dissertação está organizada em 6 capítulos. Após o capítulo de introdução, o referencial teórico e os conceitos principais são apresentados no Capítulo 2, onde se discutem conceitos sobre geração procedural de conteúdo e entropia. Os conceitos fundamentais relacionados ao aprendizado por reforço, tais como: Processo de Decisão Markov, algoritmo Q-*Learning* e *Deep Reinforcement Learning* são apresentados. O Capítulo 3 apresenta os trabalhos relacionados ao aprendizado por reforço, geração procedural de conteúdo e iniciativa-mista. No Capítulo 4 é apresentado o método proposto para gerar cenários utilizando agentes de aprendizado por reforço. Os resultados dos experimentos são descritos no Capítulo 5. Por fim, o Capítulo 6 apresenta as conclusões e trabalhos futuros.

# 2 REFERENCIAL TEÓRICO

Neste capitulo será explanado algumas tecnologias e técnicas relacionadas com RL e GPC, bem como termos essenciais para o entendimento da proposta desta dissertação, relacionados a nossa abordagem com RL e GPC.

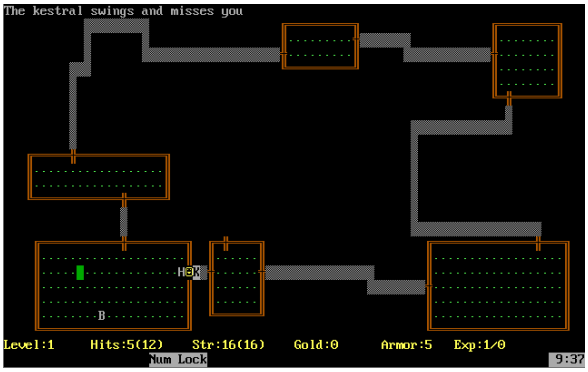
## 2.1 GERAÇÃO PROCEDURAL DE CONTEÚDO

Ao longo dos anos, diversos métodos de geração procedural de conteúdo tem sido desenvolvidos, cada um com diferentes formas para atingir um determinado objetivo (LAZARIDIS et al., 2022). A GPC é um método utilizado para gerar conteúdos de jogos utilizando algoritmos com alguma entrada direta ou indireta por um usuário. Os conteúdos são gerados usando um processo randômico, em outras palavras, o próprio algoritmo cria o conteúdo dos jogos sem a intervenção de designers humanos. O termo "conteúdo" em sua definição refere-se aos elementos de um jogo: níveis, mapas, regras do jogo, texturas, histórias, itens e músicas (SHAKER; NELSON, 2016).

A GPC em jogos teve como grande inspiração para o seu uso na década de 1980 devido à limitação de memória dos computadores da época, forçando os designers de games a criarem diferentes métodos de geração procedural.

Há vários jogos que fazem o uso de alguma técnica de GPC, como o clássico jogo de computador Rogue (figura 1) inspirado no jogo de tabuleiro Dungeons & Dragons desenvolvido na década de 1980, um jogo de masmorras (*dungeon-crawling*) cujos níveis são gerados randomicamente a cada vez que é iniciado (SHAKER; NELSON, 2016). Dentre

Figura 1 – Jogo Rogue da década de 1980




Fonte: Elaborada pelo autor (2022).

os jogos atuais têm se observado muitos gêneros específicos que utilizam técnicas de GPC, por exemplo, plataforma, aventura e puzzles. Assim, jogos como No Man's Sky (figura 2b), Spelunky (figura 2a), Diablo III (figura 2c) e Minecraft (figura 2d) tentam

usufruir dos benefícios dessas técnicas para torná-los mais diversos, evitando a repetição ou a linearidade.

Diversas pesquisas desenvolvidas na área de GPC tem sido de grande importância na área de jogos de entretenimento e, muitas empresas desenvolvedoras de jogos tem optado por utilizar alguma técnica de geração procedural ou criar a sua própria. A geração procedural de conteúdo tem se tornado uma ótima alternativa para desenvolver jogos de forma rápida, com um custo baixo e tem principalmente sido uma escolha por muitas empresas de desenvolvimento de jogos Indie[1].

O uso de alguma técnica de GPC representa diversos benefícios, tais como, agilidade no desenvolvimento e variação no *gameplay* proporcionando novas experiências aos jogadores.

Figura 2 – Jogos que fazem o uso de alguma técnica de geração procedural

(a) Spelunky

(b) No Man's Sky

(c) Diablo III

(d) Minecraft

Fonte: Elaborada pelo autor (2022).

Técnicas de GPC têm sido amplamente aplicada em diferentes contextos. O tipo da técnica aplicada dependerá do tipo de problema que se deseja resolver, por exemplo, gerar uma sequência simples de números pseudo-randômicos sem repetições. Há dois tipos

[1] O termo Indie é uma abreviação para jogo independente (do inglês, *Independent video game*). São jogos produzidos de modo autônomo e sem o apoio financeiro de uma grande empresa.

de técnicas de GPC que são frequentemente utilizadas em jogos: *online* e *offline*. Neste trabalho o foco e utilizar GPC com a técnica *offline*.

- ***Online***: Segundo (KORN; LEE, 2017) Técnica utilizada para gerar conteúdo enquanto o jogador está jogando. Para melhor contextualizar essa técnica, é quando o jogador acessa um ambiente, por exemplo, uma sala, e a estrutura desse ambiente é gerada instantaneamente. A cada acesso à sala, uma nova estrutura é criada. Essa técnica permite a geração de cenários infinitos. O jogo *Minecraft* é um exemplo que permite a geração de cenários infinitos.
- ***Offline***: diferentemente da técnica online, segundo (SHAKER; NELSON, 2016) o básico da construção do jogo é gerado durante o desenvolvimento, ou antes, de iniciar uma sessão do jogo. O algoritmo trata de realizar pequenas alterações no cenário caso tenha modelos do nível para gerar conteúdos diversificados. O jogo *Spelunk* gera os cenários baseados em modelos: antes de iniciar a sessão, o algoritmo seleciona os modelos randomicamente, define o início e o final do nível e realiza pequenas modificações nos cenários, incluindo inimigos, itens e entre outros elementos.

## 2.2 ENTROPIA

Antes de abordarmos os fundamentos da entropia, é importante citar sobre a teoria da informação. A teoria matemática da informação possui dois principais objetivos segundo (GRAY, 2011): primeiro é o desenvolvimento dos limites teóricos sobre a fonte de informação dado sobre um canal de comunicação. O segundo objetivo é desenvolver esquemas de codificação que fornecem uma comparação razoavelmente boa dado a teoria. Segundo (STONE, 2015), a teoria da informação teve seu surgimento em 1948 com Claude Shannon que publicou um artigo chamado *A Mathematical Theory of Communication* (SHANNON, 1948).

A teoria da informação de Claude Shannon fornece uma definição matemática para a informação, descrevendo a quantidade de informação necessária para comunicar entre diferentes elementos de um sistema (GRAY, 2011). A entropia foi criada com o objetivo de quantificar a quantidade de informação contida em uma mensagem ou fonte de informação. As informações são medidas em bits, e esses bits nos permite saber a probabilidade de igualdade entre duas informações distintas.

A entropia é calculada a partir de uma distribuição de probabilidade de informações em uma mensagem. Quanto mais uniforme for a distribuição de probabilidade, maior será o valor da entropia, portanto, maior será a aleatoriedade presente na informação. Podemos definir a entropia como o número de bits necessários para transmitir um evento aleatoriamente a partir de uma distribuição de probabilidade, cujos eventos são representados por $X = (x_1, x_2, x_3, ..., x_n)$. Para entendermos melhor essa definição, tomemos como exemplo

informações sobre o tempo: chuvoso, ensolarado e nublado. São três eventos de tempos distintos, cuja probabilidade de ocorrer também são distintas. Consideremos essas três variáveis de tempo de ocorrerem em um território desértico ao longo do ano. Com base nesse exemplo hipotético foi verificado que, a distribuição de probabilidade desses eventos ocorrerem ao longo do ano são: chuvoso 10%, ensolarado 70% e nublado 20%. Utilizando a entropia, quantificamos a quantidade de informação de cada um desses eventos e para isso, utilizamos a seguinte equação 2.1.

$$H(x) = -\sum_{i}^{n} p(x_i) \log(p(x_i)) \tag{2.1}$$

onde $p(x_i)$ é a probabilidade de ocorrência de cada evento para a variável $x$. A quantidade de informação que cada evento possui é calculada por $log(p(x_i))$. Para descobrir o valor de entropia dos eventos, primeiro devemos calcular o log de cada probabilidade e multiplicar por cada probabilidade. Observe que, para as probabilidades estamos utilizando a frequência relativa (vide tabela 1). Com base nisso, a probabilidade de um evento ocorrer ou não, por exemplo, tempo ensolarado é adotado os valores entre 0 e 1. Lembrando que a soma das probabilidades desses eventos ocorrerem deve ser um ($p = 0,10+0,70+0,20 = 1$).

Tabela 1 – Distribuição de probabilidade dos eventos

| | **Chuvoso** | **Ensolarado** | **Nublado** |
|---|---|---|---|
| | $x_1$ | $x_3$ | $x_3$ |
| **Probabilidade** | 0,10(10/100) | 0,70(70/100) | 0,20(20/100) |
| **log** | $\log(0.10) \approx -3.32$ | $\log(0.70) \approx -0.51$ | $\log(0.20) \approx -2.32$ |

Seguindo a fórmula da equação:

$$H(x) = -((0.10 \times -3.32) + (0.70 \times -0.51) + (0.20 \times -2.32)) = 1.849 \tag{2.2}$$

Como valor de entropia temos $H = 1.849$ bits de informação. Observe que, o tempo ensolarado possui uma probabilidade maior de ocorrência, dessa forma, os dados não são uniformes implicando um valor de entropia baixo. Como a informação é medida em bits, ela é usada em diversas aplicações práticas, tais como, compressão de dados, criptografia e codificação de informações. Podemos observar a importância de se utilizar a entropia para quantificar as informações. Importante citar que, não estamos interessados em uma única informação, mas sim, com todos os valores associados que a variável aleatória pode ter. Neste trabalho, utilizados a entropia para quantificar a diversidade dos níveis gerados pelos agentes.

## 2.3 APRENDIZADO POR REFORÇO

Aprendizado por Reforço é uma das principais ramificações do aprendizado de máquina que se preocupa como o aprendizado controla as políticas de interação com a experiência vinda do ambiente (BRUNTON; KUTZ, 2019).

RL é sobre o agente aprender interagindo com o ambiente através da tentativa-erro. Esse tipo de aprendizado de máquina possui como inspiração a psicologia do aprendizado, imitando como os animais aprendem interagindo com o ambiente, no qual, a cada etapa, você recebe recompensas positivas ou negativas. A ideia de aprender interagindo, é o princípio fundamental em todas as teorias de aprendizado e inteligência. O aprendizado por reforço aproxima e explora o aprendizado computacional com a ideia do aprendizado por interação.

O campo de estudos de *machine learning* (ML) é extremamente vasto e há diferentes classes de algoritmos projetados para atender problemas específicos no campo da I.A. Dentro dessa área é muito importante salientar as diferenças entre RL e outras técnicas de ML. Em RL o agente é treinado a tomar um conjunto de ações no ambiente, recebendo um sinal de recompensa por cada ação. No aprendizado por reforço é sobre como podemos mapear situações para ações, para então maximizar as recompensas recebidas durante o aprendizado. O aprendizado aqui não é dizer quais ações devem ser tomadas, mas sim, tentar descobrir quais ações possuem maiores recompensas (SUTTON; BARTO, 2018). Ao iniciar o processo de aprendizado, o agente não possui nenhum conhecimento sobre o problema proposto para aprender. O agente aprende interagindo com o ambiente e ao longo de suas experiências de interação. Após o agente aprender como ele é capaz de tomar decisões ao longo do tempo.

O aprendizado por reforço é um paradigma diferente do aprendizado supervisionado e não supervisionado. Nesse tipo de aprendizado não há dados ou rótulos como entrada. No aprendizado supervisionado há um conjunto de rótulos associados aos dados, ou seja, há um supervisor para informar o que é cada dado. No aprendizado não supervisionado não há rótulos, neste caso, não há um supervisor para informar o que é cada dado, neste tipo de aprendizado, o objetivo é encontrar padrões dado uma estrutura.

### 2.3.1 Elementos do aprendizado por reforço

O aprendizado por reforço possui elementos importantes para o seu entendimento: política (*policy*, $\pi$), sinal de recompensa (*reward signal,* $r$), função de valor (*value function,* $v_\pi$) e o ambiente (SUTTON; BARTO, 2018).

A política ($\pi$) define a forma de aprendizado do agente ao longo de seu treinamento. Consiste em uma função que mapeia estados para ações, sendo usada pelo agente para decidir qual ação tomar em um determinado estado de forma que as recompensas recebidas sejam maximizadas. A escolha da política afeta diretamente o treinamento do agente e pode ser modificada durante o treinamento para melhorar o aprendizado. Podemos entender que, a política corresponde ao que a psicologia denomina como estímulo-resposta do ambiente.

No aprendizado por reforço, o sinal de recompensa define o objetivo do problema a

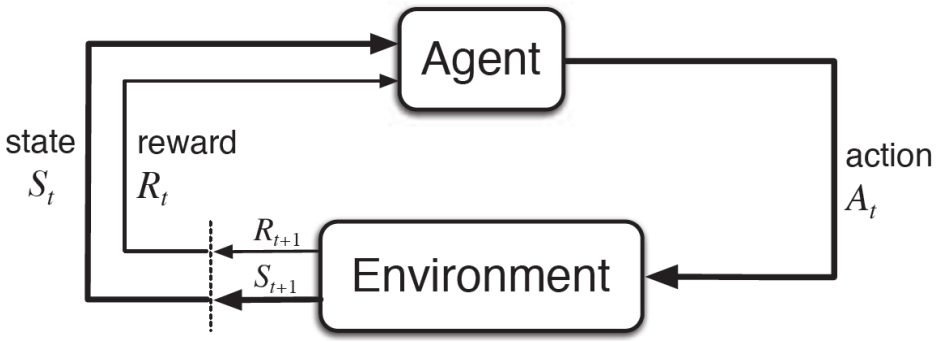
cada período de experiência adquirida. Esse sinal recebido pode ter um valor positivo ou negativo. O objetivo do agente em RL consiste em maximizar as recompensas recebidas ao longo do período de treinamento.

A função de valor $v_\pi(s)$ indica a expectativa das recompensas futuras dado um estado inicial $s$ e uma política $\pi$. Em outras palavras, ela acumula o valor total das recompensas que o agente pode esperar receber a partir do estado inicial seguindo a política $\pi$. Aplicações de RL consistem em um ciclo contínuo, em que o agente observa um estado do ambiente e escolhe uma ação para ser executada. Após executar uma ação, o agente recebe uma recompensa do ambiente. Em RL, a função de valor é usada para estimar o valor de um estado ou ação, e assim, guiar o agente na escolha de ações que levam a recompensas maiores.

### 2.3.2 Processo de Decisão de Markov

O Processo de Decisão de Markov (*Markov Decision Processes*, MDP), envolve a evolução do feedback, e está associado a escolha de diferentes ações em diferentes situações. O MPD é também uma tomada de decisão sequencial, no qual, suas ações influenciam imediatamente nas suas recompensas, mas também, em recompensas futuras. No MDP, temos um conjunto de estados, ações e recompensas representado por, $\mathcal{S}, \mathcal{A}, \mathcal{R}$ respectivamente. Neste modelo, especificamente, o agente interage com o ambiente em cada sequência de tempo $t = 0, 1, 2, 3, ...$(vide figura 3).

Figura 3 – O agente interagindo com o ambiente em um Processo de Decisão Markov.




Fonte: (SUTTON; BARTO, 2018)

A cada passo de tempo $t$, o agente recebe informações sobre o estado atual do ambiente, representado por $S_t \in \mathcal{S}$. Com base nesse estado, o agente seleciona uma ação do conjunto de ações disponíveis, representado por $A_t \in \mathcal{A}$. Em seguida, o agente recebe uma recompensa $R_{t+1} \in \mathcal{R}$ como feedback pela ação tomada e é levado a um novo estado $S_{t+1}$. É importante destacar que a transição de um estado para outro e a obtenção de uma recompensa dependem das probabilidades associadas a esses valores no tempo $t$ dentro do

MDP. Em sua definição mais formal temos:

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1}|S_t = s, A_t = a\} \tag{2.3}$$

Onde a probabilidade de transição de sair do estado atual, $s$, no tempo $t$, para ir para o próximo estado, definido como $s'$ no tempo $t$ dependem dado uma ação $a$ executada no estado atual $s$. Na equação 2.3 o símbolo $'|'$ significa condição de probabilidade e $p$ especifica a distribuição de probabilidade de escolha de cada estado, $s$, e ação, $a$.

Cada estado (*state*, $s$) e cada ação (*action*, $a$) estão associados a um valor estimado, $q$ (*Quality*). No MDP temos o que é chamado *quality value*, $q(s, a)$ que corresponde as recompensas de ação em um estado.

Ao observamos a figura 3 demonstrada anteriormente, temos importantes elementos relacionados ao RL - ***Agent*** e ***Environment*** e demais pontos de comunição - ***action***, ***reward*** e ***state***.

- ***Agent*:** O agente é a entidade que interage com o ambiente, realizando ações, observando os resultados e recebendo recompensas. Essa entidade pode ser representada por um cachorro ou gato em treinamento, por exemplo. O agente é a parte central do RL, uma vez que objetivo é que ele aprenda a tomar decisões inteligentes e maximizar suas recompensas. O ambiente pode variar em complexidade, podendo ser desde um simples jogo até um ambiente real, como um carro autônomo em tráfego urbano.

- ***Environment*:** O ambiente em aprendizado por reforço é o espaço onde o agente interage e executa ações em busca de uma recompensa. Um exemplo clássico de ambiente é o de um labirinto, onde o agente (como um rato) deve encontrar um pedaço de queijo. Nesse ambiente, o agente precisa tomar decisões inteligentes para encontrar o caminho certo e alcançar seu objetivo de chegar ao final do labirinto para receber sua recompensa.

- ***Reward*:** As recompensas no aprendizado por reforço são apenas um simples número, $R_t \in \mathbb{R}$, que é obtido como um feedback para o agente de acordo com a ação realizada no ambiente a cada passo de tempo. No MDP a função de recompensa, representa por $r(s, a, s')$, determina um sinal de recompensa que deve ser enviada para o agente ao mover do estado $S_t \in \mathcal{S}$ para um novo estado $S_{t+1} \in \mathcal{S}$ enquanto executa uma ação $A_t \in \mathcal{A}$. O principal objetivo da recompensa é informar para o agente o quão eficiente ele foi durante o aprendizado, conforme especificado no tópico **2.3.1**.

- ***Action*:** As ações são as decisões que o agente pode realizar no ambiente. Exemplos de ações podem ser: movimentar para esquerda ou direita, pular, atirar (caso esteja em um jogo de videogame). As ações podem ser dividas em dois tipos: Discretas ou Continuas. As ações discretas são aquelas que possuem um conjunto limitado de

opções, como escolher entre andar para esquerda ou direita, pular ou atirar em um jogo de videogame. Ações contínuas possuem um valor associado a elas, como uma força, ângulo ou velocidade.

- ***Observations/States*:** As observações são a forma pela qual o agente se comunica com o ambiente. Elas contêm informações sobre o ambiente, permitindo ao agente saber o que está acontecendo ao seu redor. Em uma corrida de carros, por exemplo, o ambiente é a pista de corrida e as observações podem incluir informações sobre a velocidade, ângulo e posição do carro na pista. Em outras palavras, as observações são informações que o agente recebe. De acordo com (SUTTON; BARTO, 2018), o estado é uma sequência alternada de ações $A_t \in \mathcal{A}$ e observações $O_t \in \mathcal{O}$, que pode ser representada pela palavra *history* e pela notação $H_t$. Podemos definir o histórico como:

$$H_t = A_0, O_1, A_1, O_2, A_2, O_3, A_3, O_4, ..., A_{t-1}, O_t \tag{2.4}$$

  Onde podemos sumarizar o estado como uma função $S_t = f(H_t)$. O estado contém informações mais completas relacionados as ações realizadas no ambiente pelo agente. É importante mencionar que as observações e os estados são frequentemente usados de maneira semelhante e sem distinção em diferentes literaturas.

Por fim, em RL, o principal objetivo do agente não é maximizar a recompensa total recebida imediatamente, mas sim, acumular a recompensa recebida ao longo do tempo, definida como:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{2.5}$$

Onde, $G_t$ é o retorno esperado (*expected return*) e $T$ é a etapa final. Entretanto, não é interessante para o agente, a cada passo de tempo, receber uma recompensa imediata após uma ação, $A_t$. Um conceito importante aplicado em algoritmos de RL é o fator de desconto, $\gamma$.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.6}$$

onde $\gamma$ é um parâmetro cujo valor é entre $0 \leq \gamma \leq 1$. O fator de desconto determina o valor presente de uma futura recompensa. O objetivo aqui é evitar a busca pela recompensa imediata para que as recompensas futuras também tenham valor quanto as recompensas imediatas. Segundo (SUTTON; BARTO, 2018), se o parâmetro $\gamma$ é mais próximo de 0 (zero), o agente é "míope", preocupando-se apenas em maximizar as recompensas imediatas, desta forma, reduzindo as possibilidades do retorno de futuras recompensas. A medida que $\gamma$ aproxima-se de 1, o agente prefere a dar mais importância as recompensas a longo prazo. Devemos optar pelo treinamento do agente para escolher as ações que atinjam a melhor solução global.

### 2.3.3 Classificação dos tipos de aprendizado e agente de aprendizado por reforço

Algoritmos de RL são classificados dependendo de diversos fatores. Neste tópico, será apresentado de forma geral os tipos de classificação existentes de acordo com o tipo aprendizado. Apresentar classificações faz se necessário para melhor compreensão dos termos apresentados sobre RL ao longo desta dissertação.

#### *2.3.3.1 Tabular Solutions x Approximate solutions*

- ***Tabular Solutions***: Neste método, as soluções do aprendizado podem ser encontradas facilmente, por fornecer exatamente a função valor e política ideal. A quantidade de possíveis estados e ações pode ser representada como tabela (uma matriz). No entanto, esse tipo de método não é eficiente para problemas muito grandes, por exemplo, um jogo de xadrez, onde é preciso mapear cada movimento das peças.

- ***Approximate solutions***: Diferentemente do método *Tabular Solutions*, podemos trabalhar com uma grande quantidade de estados e ações. Por exemplo, em um jogo de corrida que possui ações contínuas, o carro pode ir em diferentes direções, gerando uma diversidade de estados e ações e, consequentemente, uma maior variedade de combinações possíveis. O objetivo desse tipo de método é encontrar uma boa aproximação da solução. Um ponto importante desse tipo de método é sua utilização em conjunto com redes neurais, que auxiliam o agente na escolha das ações no ambiente.

#### *2.3.3.2 Estratégias de aprendizado Model-based x Model-free*

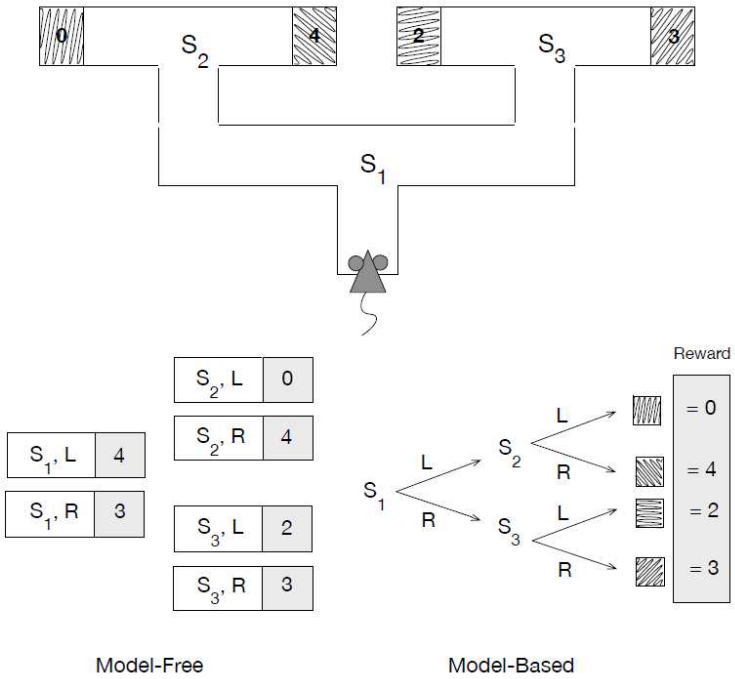
Nesta seção veremos de forma simplifica dois tipos de estratégias de aprendizado, com o olhar na perspectiva da psicologia do aprendizado e estudo de como os animais aprendem. As duas estratégias apresentadas estão ligadas com as soluções *Tabular* ou *Approximate* dependendo do tipo de algoritmo de RL.

- ***Model-based***: Segundo (SUTTON; BARTO, 2018) usa o modelo do ambiente que contém elementos que a psicologia chama de mapas cognitivos. O agente usa o modelo do ambiente para prever como ele responderá às ações, em termos de estados e recompensas. O modelo é utilizado para o agente simular o ambiente real para prever o próximo estado e recompensa. Em outras palavras, o agente possui uma descrição completa do ambiente, bem como, a probabilidade de transição entre os estados e a recompensa recebida.

- ***Model-free***: Refere-se quando o agente não conhece a dinâmica do ambiente, ou seja, não possui o modelo do ambiente como no modelo de aprendizado *model-based*. O agente tenta buscar a política ótima com a tentativa-erro aproximadamente.

A figura 4, ilustra as diferenças entre *model-based* e *model-free* que tenta resolver uma sequência de ações. Neste exemplo, o agente (o rato) tem como objetivo atingir objetivos distintos representados pelas caixas hachuradas, cada uma delas associada à recompensa. Observe que, utilizando a estratégia *model-based* o agente conhece as possíveis ações que leva a maior ou menor recompensa. A estratégia *model-free* possui associado um par de estado-ação(state-action, $(s_k, a_k)$). Os valores de ação são estimados de acordo com o maior valor esperado de cada ação tomada a partir de cada estado. Quando o valor de ação retorna um valor ótimo estimado, o agente apenas tenta selecionar no estado a ação com o maior valor.

Figura 4 – Diferenças entre *model-based* e *model-free*.




Fonte: (SUTTON; BARTO, 2018)

#### *2.3.3.3 Tipos de agentes RL*

Neste tópico será abordado os tipos de agentes classificados de acordo com seu tipo de política que podemos encontrar em implementações de algoritmos RL. A seguir são eles:

- ***Value-based***: O agente aprende a estimar as recompensas futuras para cada estado-ação através de uma função valor. Com base nessa estimativa, o agente é capaz de

tomar decisões, selecionando a ação que possui a maior recompensa futura esperada. O algoritmo Q-*learning* (veja a seção **2.3.4**) é um exemplo de método baseado em valor.

- ***Policy-based***: O agente utiliza a política atual para aprender a tomar decisões, sem depender de uma função valor que determine a melhor ação dado um estado. Esse método aprende diretamente a política ótima mapeando cada estado para melhor ação a ser tomada.

- ***Actor-critic***: Utiliza ambas as estratégias *value-based* e *policy-based* para auxiliar na tomada de decisões. Por meio desse método, é possível utilizar informações de uma função valor para avaliar ações em relação a um estado, além de atualizar diretamente a política utilizada para tomar decisões.

Os agentes citados, *value-based*, *policy-based* e *actor-critic* estão ligados também a estratégia de aprendizado *model-based* e *model-free*, além disso, na literatura iremos encontrar dois outros tipos de agentes associados aos modelos citados anteriormente, os métodos *off-policy* e *on-policy*.

Neste ponto, compreenderemos como as políticas são usadas durante o treinamento, ou seja, a coleta de experiências e interações. Entretanto, para um melhor entendimento sobre as políticas *off-policy* e *on-policy* dois importantes termos são utilizados: Comportamento da política (do inglês, *Behavior Policy*) e Política alvo (do inglês, *Target Policy*). *Behavior Policy* é a política que o agente utiliza durante o treinamento para determinar qual ação escolher em determinado estado. A política *behavior policy* pode ser determinística ou estocástica. *Target policy* é a política que se deseja aprender para que o agente possa maximizar as recompensas. A seguir são descritos as diferenças entre as políticas de aprendizado *off-policy* e *on-policy*.

- ***off-policy***: É um tipo de política que o agente aprende a partir de experiências coletadas definidas pela *behavior policy*. Em outras palavras, o agente aprende a executar ações no ambiente a partir de experiências coletadas por outra política. Este tipo de política durante o aprendizado tenta otimizar os dados gerados durante o treinamento. A política aprende sobre a *target policy* e a usa para gerar o comportamento da política (SUTTON; BARTO, 2018) ao longo da interação com o ambiente.

- ***on-policy***: Diferentemente do método *off-policy*, este método se baseia na política atual para tomar decisões. Isso significa que o agente sempre utiliza a mesma política para selecionar ações durante a interação com o ambiente. Ao contrário da abordagem *off-policy*, o método *on-policy* não tenta otimizar a política através da

utilização de dados gerados durante o treinamento. Em outras palavras, o agente utiliza a *behavior-policy* para selecionar as ações e estimar as recompensas futuras.

### 2.3.4 Q-Learning

Dentre os algoritmos e aproximação com aplicações RL, *Q-Learning* é o mais conhecido e aplicado em diversos ambientes de treinamento simples e sendo esse um dos principais algoritmos do método *off-policy Temporal Difference learning. Temporal Difference learning*, conhecida como *TD learning* é uma estratégia de aprendizado que aprende a partir diretamente da experiência interativa. No *Q-Learning*, a função estado-ação, $Q(s, a)$, é atualizada usando a regra apresentada na equação 2.7.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma maxQ_a(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.7}$$

Parâmetros importantes desta equação devem ser destacados: o parâmetro $\alpha$ representa a taxa de aprendizagem, o parâmetro $\gamma$ representa o fator de desconto.

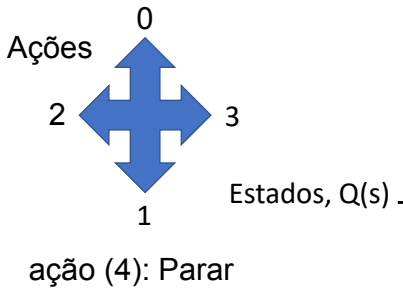
Por ser um método *off-policy* o *Q-Learning* aprende a tomar diferentes ações baseadas no comportamento da política, enquanto um método *on-policy* aprende a tomar as ações com base na política atual $\pi(a|s)$. O *Q-Learning* é considerado um método off-policy por utilizar duas políticas diferentes, uma que atualiza a função $Q(s, a)$ buscando uma política ótima e a segunda que seleciona diferentes ações ($a_{t+1}$) no ambiente. Por está razão, o *Q-learning* é considerado um método off-policy, pois atualiza a função $Q$ (*Q-function*) utilizando experiências coletadas por diferentes políticas. A *Q-function* captura a recompensa futura total esperada que um agente no estado $s$ pode receber ao executar uma determinada ação $a$, sendo representada por uma função ação-valor (*action-value*) $Q(s, a)$. A ação-valor é utilizada para representar a soma esperada das futuras recompensas que o agente recebe ao executar uma ação respeitando uma política $\pi$.

O algoritmo *Q-Learning* utiliza uma estratégia de aprendizado conhecida como *$\epsilon$-greedy*. Essa estratégia tem como objetivo permitir que o agente explore (*exploration*) o ambiente, selecionando ações randômicas com a probabilidade $\epsilon$ e, caso contrário, seleciona uma ação com a maior recompensa futura esperada com probabilidade $1-\epsilon$, conhecida como explotação (*explotation*), que é, $maxQ_a(S_{t+1}, a)$. A ação escolhida para a explotação é aquela com o maior valor de $Q_a(S_{t+1}, a)$, ou seja, a ação que maximiza a função estado-ação para o próximo estado $S_{t+1}$.

*Q-Learning* está diretamente associado com MDP, sendo que, aplica um espaço de ação discreta $a$ e um espaço de estado $s$ (BRUNTON; KUTZ, 2019). A *Q-function* é representada por uma tabela, chamada de *Q-table*, contendo *Q values* que são atualizados a cada iteração durante o treinamento do agente. Na tabela *Q-table* (vide figura 5), as linhas representam os estados do agente, os quais são associados a cada possível ação representada pelas colunas numeradas de 0 a 4. Um *Q-value* alto expressa uma melhor

ação, um *Q-value* baixo indica uma ação menos favorável de ser executado dado um estado. A política $\pi$ escolhe a melhor ação, definida como $max_a Q(s_{t+1}, a)$, a ser tomada dado um estado $s$, se o agente estiver utilizando a política $\epsilon$-*greedy*, caso contrário, escolhe uma ação randômica. Por exemplo, se após o treinamento o agente iniciar no estado $Q(s) = 0$, então, a política irá escolher a melhor ação ($max_a Q(s, a)$) referente ao estado $Q(s) = 0$, neste caso a ação $Q(a) = 3$.

Figura 5 – Q-table mapeando os estados para as ações




Ações, Q(a)

Estados, Q(s)

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | -1,88 | -1,86 | -1,92 | -1,63 | -972,78 |
| 1 | -1,71 | -1,22 | -1,87 | -1,70 | -997,19 |
| 2 | -1,54 | -1,22 | -1,35 | -0,44 | -953,69 |
| 3 | -1,12 | -1,37 | -1,35 | 1,14 | -971,79 |
| 4 | 0,92 | 4,14 | -0,54 | 4,28 | -970,15 |
| 5 | 4,17 | 10,56 | 0,80 | 4,01 | -994,63 |
| 6 | 23,12 | 9,67 | 4,25 | 22,88 | -982,51 |
| 7 | 48,25 | 22,47 | 10,23 | 22,72 | -948,57 |
| 8 | 23,12 | 9,74 | 9,97 | 4,14 | -986,47 |
| 9 | 10,56 | 3,38 | 9,65 | 0,16 | -978,10 |
| 10 | 4,28 | 0,93 | 3,46 | 1,07 | -920,58 |

Fonte: Elaborada pelo autor (2022).

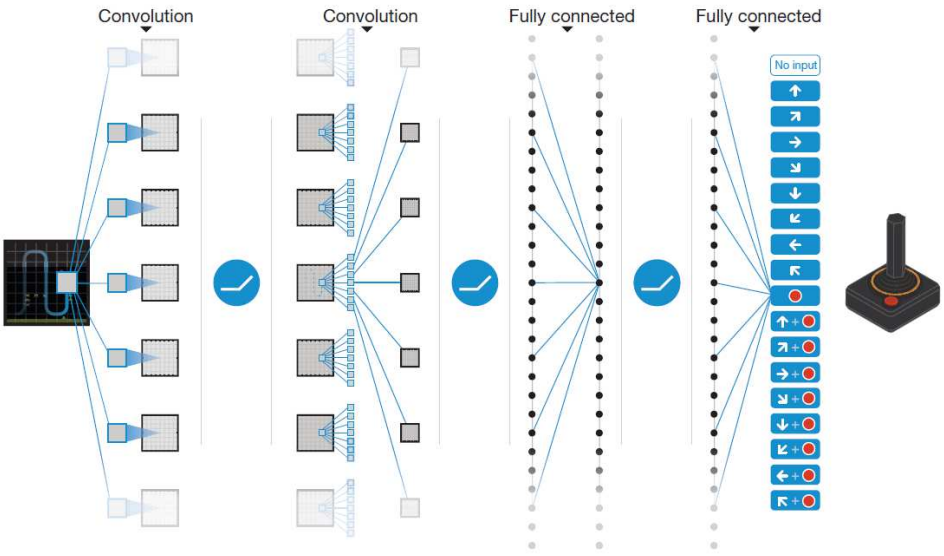
### 2.3.5 Deep Reinforcement Learning

Segundo (BRUNTON; KUTZ, 2019) *Deep Learning* (DL) tem revolucionado a nossa habilidade para representar funções complexas provenientes de dados e provendo um conjunto de arquiteturas para permitir alcançar uma performance ao nível humano, por exemplo, reconhecimento de imagens e processamento de linguagem natural. A palavra *Deep* em DL representa o uso de múltiplas camadas das redes neurais necessárias para transformar dados em dados que são importantes (LEI, 2021).

*Deep reinforcement learning* (DRL) é um subcampo do aprendizado de máquina que combina técnicas de RL e DL (FRANçOIS-LAVET et al., 2018). Nos últimos anos, houve um grande crescimento de pesquisas nesta área, por exemplo, o trabalho de (Mnih, V., Kavukcuoglu, K., Silver, D. et al., 2015), que propôs unir as redes neurais e *Q-Learning*, surgindo um agente de inteligência artificial chamado de *Deep Q-Network* (DQN). O objetivo do DQN era jogar jogos do videogame Atari, e para isso, utilizou uma *deep convolutional neural networks* (DCNN) para aproximar função ação-valor (*action-value*), *Q-function*. No DQN as entradas da rede neural consiste em imagens de jogo do videogame Atari (vide figura 6).

DRL permite suprir problemas no campo do RL que antes não era possível interagir, por exemplo, aprender jogos de vídeo games, como no trabalho proposto por (Mnih, V.,

Figura 6 – Estrutura da DQN usada para jogar jogos de Atari.




Fonte: (Mnih, V., Kavukcuoglu, K., Silver, D. et al., 2015)

Kavukcuoglu, K., Silver, D. et al., 2015). Algoritmos de DRL podem ser aplicados a diferentes propósitos, como na robótica, onde os robôs podem aprender através de entradas do mundo real fornecidas por imagens capturadas por câmeras. Para (BRUNTON; KUTZ, 2019) é possível usar DL de diferentes formas para aproximar as várias funções usada em RL. Devida a relação de RL com as redes neurais, a política antes representada por $\pi(a, s)$, agora é aproximada por:

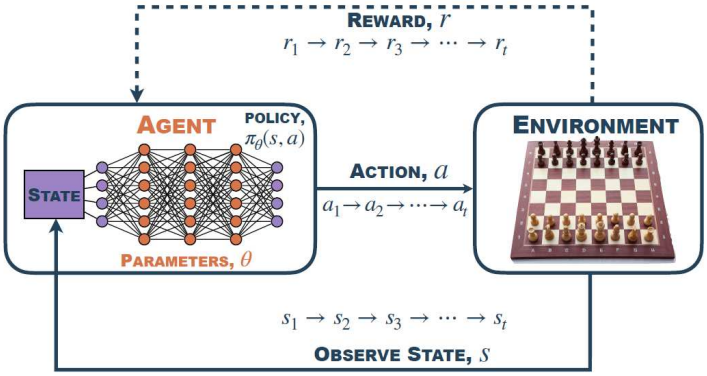
$$\pi(s, a) \approx \pi(s, a, \boldsymbol{\theta}), \tag{2.8}$$

onde $\theta$ representa os pesos de uma rede neural. Um segunda representação pode ser utilizada como $\pi_{\boldsymbol{\theta}}(s, a)$. A figura 7, ilustra um esquema semelhante ao demonstrado para ao processo de decisão Markov, a principal diferença está no agente que utiliza uma rede neural para definir a política. Segundo (BRUNTON; KUTZ, 2019) esse esquema é denominado como *deep policy network*.

## 2.4 MÉTODOS BASEADOS EM POLÍTICA DE GRADIENTE

Métodos relacionados à política de gradiente (*policy gradient*, PG) pertencem a uma família de algoritmos de aprendizado por reforço que têm como objetivo otimizar o parâmetro da política por meio do gradiente ascendente, maximizando a recompensa acumulada esperada. Esses métodos são considerados uma das técnicas mais comuns para otimizar a política em RL. Diversos algoritmos de RL tentam aprender um par de estado-ações (*state-action*) ou funções ações-valor (*action-value*), como o Q-learning (veja a seção **2.3.4**), que utiliza uma política $\epsilon$- *greedy*; sua política não existe sem estimar as funções ação-valor. Por outro lado, algoritmos baseados em política de gradiente

Figura 7 – Esquema de RL utilizando uma deep policy network




Fonte: (BRUNTON; KUTZ, 2019)

introduzem um parâmetro $\boldsymbol{\theta}$ que aprendem a selecionar as ações sem consultar a função valor. Neste caso, a função valor é utilizada para aprender o parâmetro da política e não é necessária para escolher uma ação (SUTTON; BARTO, 2018).

Segundo (SUTTON; BARTO, 2018) utiliza-se a notação $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ para definir um vetor de parâmetros da política, que formalmente temos (vide equação 2.9):

$$\pi(a|s, \boldsymbol{\theta}) = P_r\{A_t = a|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\} \tag{2.9}$$

para a probabilidade de tomar ação $a$ no tempo $t$ dado um ambiente em um estado $s$ no tempo $t$ com o parâmetro $\boldsymbol{\theta}$, em outras palavras, seleciona uma ação no estado $s$ de acordo com o parâmetro $\boldsymbol{\theta}$. Importante citar que, as parametrizações devem ser realizadas por uma rede neural, neste caso, uma *deep policy network*. PG possui um função de valor escalar definida como $J(\boldsymbol{\theta}_t)$ que mede a sua performance respeitando os parâmetros $\boldsymbol{\theta}$. Esse método visa maximizar o desempenho utilizando o gradiente ascendente para atualizar os parâmetros $\boldsymbol{\theta}$ da função escalar $J(\boldsymbol{\theta}_t)$:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\widehat{\Delta J(\boldsymbol{\theta}_t)} \tag{2.10}$$

Onde $\alpha$ é a taxa de aprendizagem e $\widehat{\Delta J(\boldsymbol{\theta}_t)} \in \mathbb{R}^{d'}$ simboliza o aumento do gradiente. Segundo (SUTTON; BARTO, 2018) todos os métodos que seguem essa estrutura são chamados de *policy gradient methods*. Esse processo é o oposto do gradiente descendente que minimiza a função de perda (*loss function*) de uma rede neural durante o treinamento.

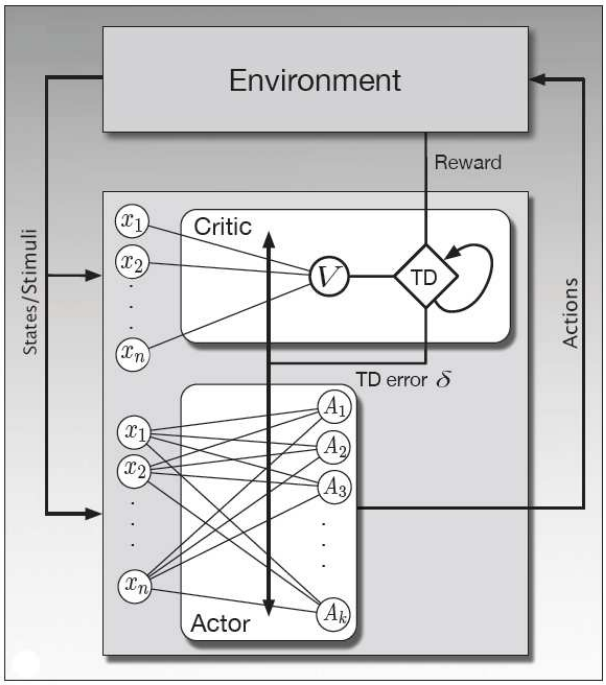
### 2.4.1 Actor-Critic

O método *actor-critic* segundo (RAVICHANDIRAN, 2020) é atualmente um dos métodos mais populares dentro os algoritmos de DRL. Trata-se de um tipo de algoritmo de RL que une dois tipos de métodos de aprendizado, *value-based* e *policy-based* visando aprender ambas as funções.

Neste método, existem essencialmente dois tipos de redes neurais conhecidas como *actor* e *critic* que trabalham em conjunto. A rede *actor*, representado por $\pi(a|s, \boldsymbol{\theta})$, controla a probabilidade do agente selecionar cada ação. A rede *critic*, representada $\hat{v}(s, \mathbf{w})$ ou $\hat{v}_{\mathbf{w}}(s)$ onde, $\hat{v}$ aproxima o valor do estado $s$ dado um vetor de pesos $\mathbf{w}$, avalia o quanto a ação selecionada é boa de acordo com a recompensa retornada para o agente.

Segundo (SUTTON; BARTO, 2018) a rede *critic* usa o algoritmo de aprendizado *Temporal Difference learning* (TD) para aprender a função estado-valor para avaliar as ações do *actor*. A rede *actor* seleciona suas ações a serem executadas no ambiente de acordo com o feedback enviado pela rede *critic* via TD *errors*, $\delta$, como valores positivos ou negativo. O valor positivo de $\delta$ significa o quão boa foi a ação para estado; um valor de $\delta$ negativo indicada uma ação ruim para o estado. Com base nesse feedback enviado pela rede *critic*, o *actor* atualiza a sua política. A figura 8, ilustra graficamente a rede neural *actor-critic*. Ambas as redes *actor* e *critic* recebem como entrada várias características

Figura 8 – Representação da rede neural *actor-critic*




Fonte: (SUTTON; BARTO, 2018)

referentes ao estado do agente no ambiente. No entanto, o *actor* não recebe diretamente o sinal de recompensa e o *critic* não possui acesso direto as ações.

### 2.4.2 Proximal Policy Otimization

O Policy Proximal Optimization (PPO) é um algoritmo de RL que utiliza um método baseado em política de gradiente, sendo *model-free* e *on-policy*. Esse algoritmo alterna entre a interação do agente com o ambiente para coletar amostras de dados e a

otimização de uma função objetivo, usando um gradiente ascendente estocástico (do inglês *Stochastic Gradient Ascent*, SGA) segundo (SCHULMAN et al., 2017).

O PPO é um método diferente do Q-*leaning*, apesar de ambos compartilharem teoricamente o MDP, entretanto, internamente possuem implementações diferentes. No Q-*learning* temos como objetivo aprender uma única ação determinística que faz parte de um conjunto de ações discretas, $a \in A$, ou seja, tenta aprender um par de estado-ação, escolhendo a melhor ação a partir de um estado. O Q-*learning* falha em muitos problemas de aplicações RL mais complexas quando precisamos mapear vários conjuntos de ações para um estado enquanto que métodos que utilizam política do gradiente como o PPO tendem a ter diversas vantagens utilizado um grande conjunto de dados. Outra vantagem em utilizar o PPO é que pode ser utilizado em ambientes com ações discretas e contínuas. Como o espaço de ação é amplo, métodos baseados em gradiente de política consegue resolver problemas em ambientes cuja política ideal para o aprendizado seja estocástica. Geralmente, esse método trabalha associado com método chamado função de vantagem ($\hat{A}$). A função de vantagem é usada para determinar o quão melhor uma ação $a$ é em relação às outras ações em um determinado estado $s$. O estimador de vantagem é importante para selecionar as ações que retornam a maior recompensa em relação às outras. Para realizar essa tarefa, é utilizado uma rede neural que é treinada para prever a vantagem de cada ação tomada em um determinado estado. Em outras palavras, se a função vantagem for positiva, aumenta a probabilidade de seleção da ação $a$, caso contrário, a probabilidade é diminuída. A função de vantagem é definida como:

$$\hat{\mathrm{A}}_t(s_t, a_t) = \mathrm{Q}(s_t, a_t) - \mathrm{V}(s_t) \tag{2.11}$$

O PPO tem como base de implementação o algoritmo *Trust Region Policy Optimization* (TRPO), (SCHULMAN et al., 2015), porém, utiliza um método de restrição diferente para otimizar a substituição da função objetivo penalizando as mudanças da política de acordo com a equação 2.12. O PPO faz um corte na função objetivo quando se move muito distante da política atual. Com isso, evitando um excessiva atualização da política.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{\mathrm{A}}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{\mathrm{A}}_t)] \tag{2.12}$$

Aqui, $r_t(\theta)$ define a razão da probabilidade entre a nova e a antiga política, vide equação 2.13:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{\mathrm{A}}_t(s_t, a_t) \tag{2.13}$$

O primeiro termo da equação 2.12 dentro de *min* é a equação 2.13. O segundo termo, $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)$, modifica a substituição da função objetivo realizando um corte na razão de probabilidade para tentar remover o incentivo que a nova política fique

muito distante da política antiga e não ficando fora do intervalo $[1-\epsilon, 1+\epsilon]$, desta forma, estabilizar o treinamento e restringir grandes mudanças durante as interações. O hiperparâmetro $\epsilon$ é um valor positivo que controla a largura do intervalo de corte. Por padrão, é definido como $\epsilon = 0.2$. Em resumo, o algoritmo PPO funciona coletando um conjunto de dados de experiência, usando esses dados para atualizar a política do agente em vários passos usando função e a regularização, e repetindo esse processo até que a política do agente se torne satisfatória.

Uma das formas mais comum de implementação do PPO e com o método *actor-critic*, ou seja, utiliza rede neural, onde *actor* é utilizado para aprender uma política e *critic* refere-se ao aprendizado da função de valor, usualmente função de estado-valor.

O algoritmo PPO adota uma abordagem de otimização de política em lote, isto é, ele coleta um conjunto de dados (estados, ações e recompensas) antes de atualizar a política do agente. Utilizando uma rede neural, o algoritmo atualiza a política do agente em várias etapas após a coleta de dados. Um ponto crucial do PPO é a técnica de amostragem de trajetórias, em que vários episódios são executados em paralelo para aumentar a eficiência do treinamento. Para isso, o algoritmo utiliza um tamanho de trajetória fixo, em que cada interação é executada em paralelo com $N$ atores (*actor*) coletando dados no tempo $T$ no ambiente (ver algoritmo 1) em paralelo. Nas linhas 2 a 5, os dados são coletados no ambiente para o passo de tempo $T$ (linha 3) e, em seguida, o estimador de vantagem é calculado (linha 4) para ser utilizado na linha 6. Nesse caso, podemos executar a otimização através do algoritmo Adam (KINGMA; BA, 2017) por K épocas na mesma amostra de dados coletados na trajetória, utilizando minilotes a cada passo de tempo $NT$ para atualizar a política.

**Algoritmo 1:** PPO, implementação com Actor-Critic (SCHULMAN et al., 2017).

1 **para** $iteration = 1, 2...$ **faça**
2  **para** $actor = 1, 2..., N$ **faça**
3  Run policy $\pi_{\theta_{old}}$ in environment for $T$ timesteps;
4  Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$;
5  **fim para**
6  Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size M $\leq$ NT;
7  $\theta_{old} = \theta$;
8 **fim para**

# 3 TRABALHOS RELACIONADOS

O campo de *machine learning* atualmente tem se mostrado bastante eficiente em suas aplicações para treinar agentes a jogar jogos, como no trabalho de (Mnih, V., Kavukcuoglu, K., Silver, D. et al., 2015). A principal vantagem em utilizar jogos são os amplos ambientes e desafios que essa área proporciona. ML além de ser aplicado com o foco em jogos temos uma aproximação com GPC. Segundo (SUMMERVILLE et al., 2018), define *Procedural Content Generation via Machine Learning* (PCGML) como a geração de um conteúdo para um game por modelos que foram treinados com um conteúdo de um jogo existente. Na literatura podemos encontramos diversos trabalhos relacionados com GPC como em (SHAKER; NELSON, 2016), (KORN; LEE, 2017), (LUCAS; VOLZ, 2019), (YANNAKAKIS; TOGELIUS, 2018), (RISI; TOGELIUS, 2020) e (LIU et al., 2021). Esse trabalho estuda, explora e expande a união de RL, GPC e design de iniciativa-mista.

Este trabalho, diferentemente ao trabalho proposto por (KHALIFA et al., 2020) é um framework de iniciativa mista, em que um humano e computador trabalham juntos para criar os níveis. Os agentes de aprendizado por reforço iniciam o treinamento a partir de modelos de níveis modelados por um especialista humano em *level design*. Os níveis aqui são separados em segmentos (blocos), similar a peças de quebra-cabeça, e o agente deve aprender a encaixar as peças corretamento para construir um nível completo e diverso. Comparado ao trabalho de (SHU; LIU; YANNAKAKIS, 2021) que precisa reparar os níveis gerados com defeitos, em nossa proposta, não há necessidade de reparar os níveis, porque nos usamos um conjunto de dados de níveis modelos pelo especialista humano em *level design*.

O estudo proposto nesta dissertação também foi influenciado pelo trabalho proposto por (LUCAS; VOLZ, 2019) que investiga como utilizar Kullback-Leibler (KL) Divergence como uma forma de comparar e analisar a diversidade em níveis gerados proceduralmente. Aqui, nós utilizamos a entropia como uma forma de mensurar a diversidade dos cenários gerados.

No trabalho proposto por (SMITH; WHITEHEAD; MATEAS, 2011) é implementado uma ferramenta de iniciativa mista chamada Tanagra, com o foco em *level design*, permitindo humano e computador trabalharem juntos para produzir nívels para um jogo de plataforma 2D de rolamento lateral. Tanagra garante que todos os níveis criados sejam jogáveis e permite que um designer humano visualize e procure por diferentes níveis que atendam aos seus objetivos. O designer humano pode ajustar o nível iterativamente refinando e colocando ou movendo os elementos do jogo no nível criado. Segundo (SMITH; WHITEHEAD; MATEAS, 2011) Tanagra é capaz de gerar diferentes níveis mais rapidamente do que designers humanos. Para garantir que o nível seja jogável, a ferramenta Tanagra utiliza uma série de restrições para isso. Neste trabalho (SMITH; WHITEHEAD;

MATEAS, 2011) propõe a ideia de *expressive range* como análise para mensurar a qualidade dos níveis gerados, definindo como métricas a linearidade para mensurar o quão linear é o chão dos níveis gerados e a leniência para medir o nível de dificuldade. Segundo, (SMITH; WHITEHEAD, 2010) Expressive range é uma forma de análise para mensurar a qualidade e variedade dos níveis gerados pelos métodos de geração procedural.

(JUSTESEN et al., 2018) introduz o conceito de Progressive PCG (PPCG) para gerar níveis utilizando agentes de RL. PPCG tem como ideia principal, gerar níveis onde o algoritmo aprende a controlar a dificuldade de acordo com a performance do agente de RL. Para esse artigo quatro jogos distintos foram utilizados: Boulderdash, Frogs, Solarfox e Zelda. Os níveis desses jogos foram construídos utilizando o framework GVG-AI e cada um, possui uma estratégia para a geração dos níveis. Importante citar que, o agente de RL não é responsável por gerar os níveis, e sim, aprender a jogar o nível de acordo com o nível de dificuldade. A dificuldade dos níveis gerados são controladas utilizando um parâmetro de dificuldade variando entre $[0; 1]$ que é informando durante o processo de geração. A figura 43 no anexo demonstra o uso do parâmetro de dificuldade em cada um dos jogos, bem como níveis modelados por um especialista humano em *level design*.

O processo de treinamento utilizando o método PPCG inicia com o parâmetro de dificuldade $\alpha$ com o valor 0. A dificuldade é aumentada ($\alpha = \alpha + 0.01$) ou diminuída ($\alpha = \alpha - 0.01$) de acordo com a performance do agente de RL. O experimento foi comparado com outros dois procedimentos de geração de níveis *Lv X* e *PCG X*. *Lv X* refere-se a níveis gerados por especialista humano em *level design*, onde X identifica o número do nível. *PCG X* são níveis gerados com uma dificuldade constante, onde X, refere-se ao nível de dificuldade. Os autores chegam a conclusão que os agentes tiverem performance variada para cada uma das estratégias. A estratégia de aumentar dificuldade dos níveis a medida que o agente atinge o objetivo demonstra que os agentes são capazes de resolver diferentes problemas complexos. PPCG aumenta a dificuldade dos níveis gerados quando o agente consegue vencer o nívei durante o treinamento é diminui a dificuldade quando perde. Entretanto, utilizando o método PPCG para os jogos *Solarfox* e *Boulderdash* não conseguiu atingir o nível de dificuldade máxima.

No artigo (KHALIFA et al., 2020), busca investigar como os algorítimos de RL podem ser utilizados para treinar agentes de *level-designing* aplicado a GPC em jogos. Segundo (KHALIFA et al., 2020), essa é a primeira vez que técnicas de RL é proposto para lidar com este tipo de problema.

O trabalho focou em gerar níveis de duas dimensões para três ambientes de jogos diferentes (Binary, Zelda e Sokoban). A geração procedural inicia com um nível populado randomicamente por tiles. A cada passo, o gerador é permitido realizar uma pequena alteração no nível, e após cada alteração, é verificado se o objetivo do nível foi atingido e o agente recebe uma recompensa. Cada ambiente possui uma condição para informar ao

agente quando o nível é considerado concluído. O ambiente Binary, um jogo de labirinto, deve possuir um único caminho mais longo entre dois pontos. O ambiente Zelda deve possuir um único jogador, uma porta, uma chave e o jogador deve ser capaz de coletar a chave para abrir a porta. Sokoban deve possuir um único jogador e gerar a mesma quantidade de caixas e alvos.

Para realizar os experimentos, (KHALIFA et al., 2020) desenvolveram um framework intitulado de PCGRL framework[1], que pode ser implementada para qualquer jogo facilmente. O framework foi implementado utilizando a linguagem Python e a toolkit OpenAI Gym[2]. O treinamento dos agentes de RL foi realizado utilizando o algoritmo PPO (veja a seção **2.4.2**) que está presente na biblioteca Stable Baselines2 (SB2). A biblioteca SB2 contém um conjunto de implementações de algoritmos de deep reinforcement learning que facilitam a implementação de aplicações de RL.

De modo a facilitar a implementação, o framework foi quebrado em três partes. *Problem module*, *Representation module*, e *Change Percentage*. A arquitetura do framework pode ser vista no anexo figura 44.

O *Problem module* armazena todas as informações sobre nível gerado, por exemplo, funções de recompensa e objetos do nível. Um exemplo de informação fornecida é o tamanho do nível e tipos de objetos que aparecerão em cada fase do jogo. O modulo também define a qualidade da geração do conteúdo e determina quando o objetivo é alcançado.

*Representation module* é definido o espaço de observação, espaço das ações e as funções de transição. O modulo é responsável pela construção dos níveis.

*Change Percentage* define a quantidade de tiles que o gerador permite alterar. A quantidade de mudanças permitidas pelo o agente afeta o seu treinamento.

Com resultados dos experimentos (KHALIFA et al., 2020) verificou que, o agente obteve uma ótima performance ao gerar os níveis para o jogo *Binary*, entretanto para os jogos *Zelda* e *Sokoban* teve dificuldades em realizar o design dos níveis, mas foi capaz gerar uma grande quantidade de níveis jogáveis.

No trabalho proposto por (GISSLEN et al., 2021), possui como ideia principal treinar agentes de RL em ambientes em constante mudanças geradas de forma procedural. A proposta do trabalho é utilizar um modelo adversarial com dois tipos de agentes de RL chamados de *Solver* e *Generator* que gera o ambiente e o testa ao mesmo tempo com o uso de *auxiliary input* como uma variável de controle. O *Generator* é um tipo de agente responsável por gerar os níveis desafiadores de acordo com a performance do *Solver*. *Auxiliary input* tem como objetivo de controlar indiretamente a saída do modelo treinado e o grau de dificuldade do ambiente. O valor desse parâmetro é passado como entrada

[1] https://github.com/amidos2006/gym-pcgrl
[2] https://gym.openai.com/

para a função de recompensa.

O modelo proposto, *Adversarial RL for PCG* (ARLPCG), segundo (GISSLEN et al., 2021), traz diversos benefícios, tais como: Testar ambientes em constantes mudanças que testam as habilidades do jogador em jogos onde o agente necessita resolver mapas gerados pelo jogador e realizar automação de teste em jogos.

Para comparar a performance dos agentes, dois tipos de ambientes em 3D foram desenvolvidos, representando dois gêneros populares de jogos: Plataforma em terceira pessoa e corrida de carros. O objetivo do agente *Generator* em ambos os jogos é criar os ambientes, segmento por segmento, e do *Solver* é tentar chegar ao segmento gerado no ambiente o mais rápido possível sem sair da pista ou cair das plataformas. A arquitetura do modelo proposto por (GISSLEN et al., 2021) pode ser visto na figura 45 em anexo. Segundo (GISSLEN et al., 2021) o modelo proposto em seu trabalho para gerar conteúdo para jogos com RL pode treina o agente *Generator* para criar ambientes. Entretanto, é necessário possuir o agente *Solver*, sem ele o *Generator* gera ambientes sem solução.

O trabalho de (SHU; LIU; YANNAKAKIS, 2021), possui como proposta aproximar *Experience-driven PCG* e PCG via RL, intitulado de ED(PCG)RL, ou EDRL abreviado. EDRL possui como objetivo de ensinar agentes designers de RL para gerar níveis para jogos *endless* de uma maneira *online*, ou seja, os cenários são gerados em tempo real enquanto o jogo está em execução conduzido pela experiência provenientes de funções de recompensa. Segundo (SHU; LIU; YANNAKAKIS, 2021), EDRL possui diversos benefícios através da experiência baseada em funções de recompensa, pois agentes RL são capazes de modelar níveis em tempo real baseado nessas funções. O framework foi testado inicialmente utilizando um modelo do jogo Super Mario Bros(SMB) e intitulado de *MarioPuzzle*. Diversas funções de recompensa foram modeladas para framework, são elas: *Moderating Diversity Makes Fun!*, *Historical Deviation* e *Playablility*. Cada função de recompensa possui sua característica e podendo ser utilizada em conjunto durante o treinamento dos agentes e com isso obtendo resultados diferentes durante a geração dos níveis. De forma geral, a geração dos níveis ocorre da seguinte forma: Os agentes de RL geram e concatenam sucessivamente os segmentos dos níveis como blocos similares a peças de quebra-cabeças utilizando uma *Generative Adversarial Networks* (GANs) treinada. A cada novo segmento adicionado é calculado a recompensa levando em consideração a diversidade desses segmentos. A diversidade dos segmentos gerados é quantificada utilizando *Kullback-Leibler divergence* (KL-divergence) comparando os padrões dos *tiles* nos cenários. A figura 46 em anexo ilustra a estrutura geral do framework proposto por (SHU; LIU; YANNAKAKIS, 2021).

# 4 MÉTODOS

Aplicações desenvolvidas com RL necessitam de atenções ao serem implementadas, pois dependem totalmente do objetivo que se deseja treinar. Os projetistas de aplicações de RL devem considerar diversos aspectos, já que, métodos de RL são totalmente dependentes do tipo de ambiente, estados, agentes e recompensas.

Para este trabalho, formulamos um estado, um conjunto de ações e funções de recompensas que são retornadas como feedback durante o treinamento dos agentes. O foco aqui é iniciar o treinamento dos agentes de RL a partir de níveis modelados por um especialista humano em *level design* e gerar níveis a partir desses modelos. Um segundo ponto a ser verificado é a utilização de uma métrica para avaliar a diversidade dos níveis gerados pelos agentes de RL. A métrica utilizada para avaliar a diversidade dos níveis é a entropia que é aplicada sobre as ações escolhidas pelos agentes.

Neste capítulo, serão apresentados os métodos utilizados para desenvolver a solução do problema de RL proposto nesta dissertação. Em particular, discutiremos os ambientes utilizados para tratar o problema de utilizar a GPC com RL, as funções de recompensa desenvolvidas e os agentes criados para solucionar o problema.

## 4.1 AMBIENTES

A formulação dos ambientes é um ponto fundamental em aplicações RL, pois é partir deles que o agente inicia o treinamento para atingir o objetivo proposto. Para essa dissertação, três modelos de ambientes foram projetados (Dungeon, MazeCoin e Zelda) para treinar os agentes responsáveis por construir os cenários dos jogos. A escolha por desenvolver três ambientes possui como objetivo avaliar como o agente de RL se comporta e como este constrói cada cenário, e com isso construir métricas melhores para avaliação. A formulação da modelagem dos níveis para cada ambiente é descrita na seção a seguir.

### 4.1.1 Modelagem dos ambientes

Os modelos dos ambientes é um ponto importante para implementação da proposta desta dissertação, pois é a partir desses modelos que os agentes iniciam o seu treinamento, a fim de construir um cenário completo. Ao todo, foram criados três diferentes tipos de cenários e cada um deles possui um objetivo e nível de dificuldade de construção. As características dos cenários são descritos a seguir:

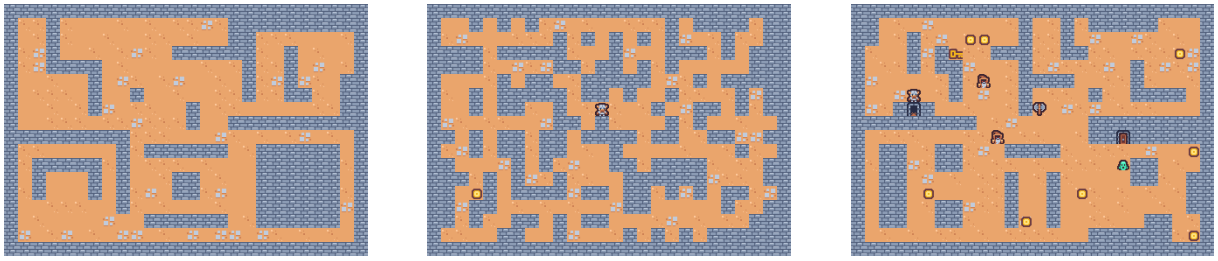
- Dungeon: Esse é um cenário mais simples (vide figura 9a), possuindo apenas regiões com paredes espalhadas estilo labirinto. Este cenário é utilizado para mensurar e verificar como o agente se comporta na criação de cenários vazios, que não possuam

objetos, tais como inimigos, chaves, portas, entre outros elementos existente em jogos.

- MazeCoin: Este cenário é baseado em um jogo de labirinto que possui um personagem controlado pelo jogador e moedas. Neste jogo (vide figura 9b), o objetivo do jogador é percorrer o labirinto para coletar as moedas no menor tempo possível.

- Zelda: Dentre os cenários, Zelda (vide figura 9c) é o mais complexo. Esse é um tipo de cenário inspirado no jogo *The Legend Of Zelda* desenvolvido e publicado pela empresa Nintendo em 1986. O cenário segue um design similar ao ambiente Dungeon, entretanto, com diferentes elementos adicionais. Neste jogo, o objetivo é enfrentar e derrotar inimigos, coletar objetos e localizar a saída das masmorras. Os elementos existentes nesse cenário são: personagem controlado pelo jogador, inimigos, espada, moeda e uma chave que precisa ser coletada para abrir a porta.

Figura 9 – Exemplos dos ambientes.



(a) Dungeon (b) MazeCoin (c) Zelda

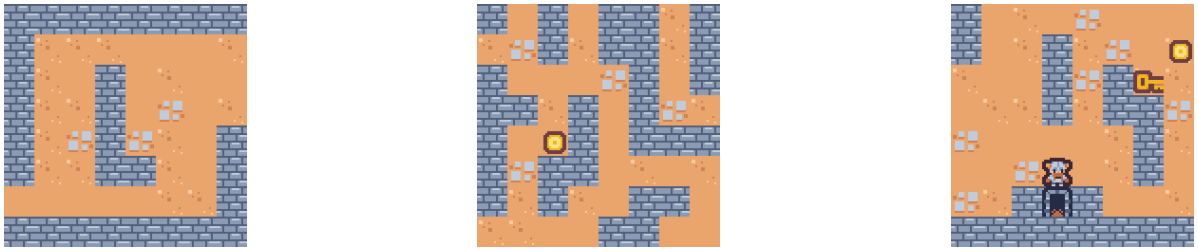
Fonte: Elaborada pelo autor (2022).

Similar a (KHALIFA et al., 2020) (PCGRL), nosso foco é criar um ambiente iterativamente. Diferentemente do PCGRL, que inicia o treinamento do agente a partir de um cenário gerado randomicamente, aqui, iniciamos o treinamento a partir de um cenário vazio como no trabalho de (GISSLEN et al., 2021) e de modelos de cenários implementados por um especialista humano em *level design*.

O especialista humano em *level design* possui o papel de projetar os modelos dos níveis que são separados em segmentos (blocos), similar a peças de quebra-cabeças (vide figura 10). Esse modelo de quebra-cabeça é baseado no trabalho de (SHU; LIU; YANNAKAKIS, 2021) que utiliza Generative Adversarial Network (GAN) para gerar os blocos dos níveis para o jogo Super Mario Bros e em (GISSLEN et al., 2021) que utiliza segmentos para construção dos ambientes.

Cada modelo de nível implementado pelo especialista é representado por uma matriz (vide figura 11) e cada valor representa um elemento do nível, tais como, paredes, inimigos e entre outros. Esse modelo de representação se assemelha com as ferramentas de

Figura 10 – Exemplos de segmentos utilizados para gerar os níveis.

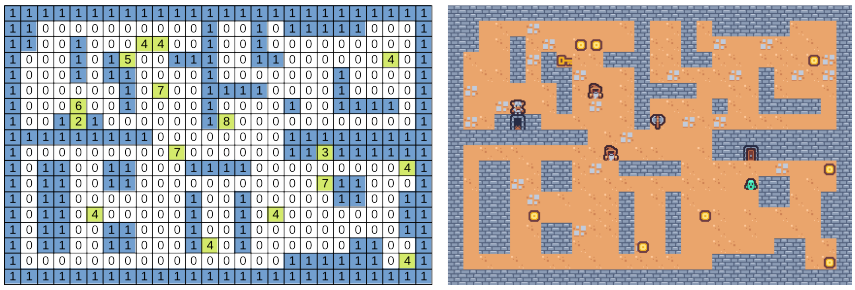


(a) Dungeon (b) MazeCoin (c) Zelda

Fonte: Elaborada pelo autor (2022).

level designer para jogos em 2D que associam cada objeto do cenário com um valor inteiro. Em jogos eletrônicos, principalmente do tipo 2D, os cenários são organizados em pequenas imagens retangulares, chamada de *tile* que corresponde a um objeto do jogo (WOLF, 2012), por exemplo, uma caixa, árvores ou casas. O cenário completo, que é composto pelos *tiles* é chamado de *tilemap*. A matriz demonstrada na figura 11 ilustra um modelo de cenário utilizando a técnica de *tiles*. Os modelos dos cenários e segmentos podem ser visualizados por completo no seguinte link: <https://github.com/dutrapaulovm/pcgrl-puzzle/tree/main/levels>

Figura 11 – Nível Zelda representado por uma matriz de números inteiros.



Fonte: Elaborada pelo autor (2022).

### 4.1.2 Definição das ações

Com os ambientes definidos, é muito importante projetar quais são as possíveis ações que o agente poderá executar em cada estado. As possibilidades de ações que o agente executará depende da dimensão e da quantidade de modelos de cenários fornecidos pelo especialista. É importante notar aqui que, cada ambiente possui uma quantidade diferente de cenários modelados pelo especialista humano em *level design* devido às características existentes em cada cenário. A tabela 2 demonstra as dimensões dos modelos dos cenários, quantidade de modelos e a quantidade de segmentos (Sn). Em nosso trabalho, os segmentos

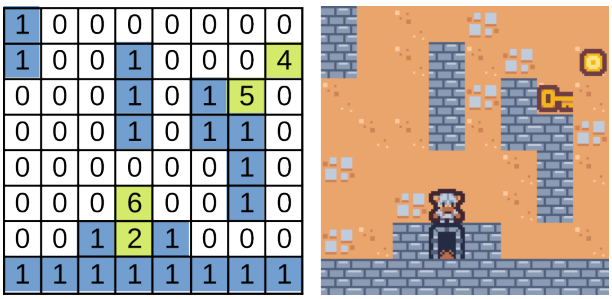
representam as ações que o agente poderá executar no ambiente para construir os cenários. Neste caso, o agente seleciona uma ação a partir de um conjunto de segmentos disponíveis em cada ambiente.

Tabela 2 – Informações dos ambientes modelados pelo especialista humano em *level design*.

| **Ambientes** | **Dimensão(mxn)** | **Modelos** | **Segmentos**(Sn) |
|---|---|---|---|
| **Dungeon** | 32 x 16 | 5 | 40 |
| **MazeCoin** | 32 x 16 | 10 | 80 |
| **Zelda** | 24 x 16 | 50 | 300 |

Para um melhor entendimento, os níveis modelados são separados em várias matrizes (segmentos) com dimensões $S_{mn} = 8 \times 8$ antes de iniciar o treinamento do agente (vide figura 12). Essa estratégia permite ao agente criar várias combinações possíveis de cenários a partir de cada ação selecionada.

Figura 12 – Segmento do Nível Zelda representado por um matriz de números inteiros.




Fonte: Elaborada pelo autor (2022).

### 4.1.3 Construção dos cenários

Como citado na seção **4.1.2**, as ações são segmentos que representam uma parte do modelo do cenário. A cada etapa da construção do nível, os segmentos são concatenados até que o nível esteja completo. O nível somente é considerado completo, quando a quantidade total de segmentos, Sn, e a condição específica de cada cenário é atingida.

Para todos os cenários é utilizado o algoritmo de preenchimento de região chamado ***flood-fill*** (vide pseudocódigo 2), desenvolvido para pintura de imagens na computação gráfica. Neste trabalho utilizamos para verificar se há uma passagem disponível para todas as áreas do cenário, pintando uma área especifica, desta forma, podemos garantir que o jogador consiga atingir o objetivo final de cada cenário. O algoritmo é aplicado da seguinte forma: dada uma matriz de números inteiros, $\boldsymbol{M}$, que representa o cenário, é escolhido um elemento da matriz para ser colorido com uma cor, por exemplo, um valor da matriz que representa um chão. Utilizando a busca em largura, o algoritmo inicia a partir de uma posição inicial $\boldsymbol{M(i, j)}$, onde $i$ e $j$ representam as linhas e colunas da matriz, colore seus vizinhos que representam o chão, finalizando, quando todas as regiões definidas estiverem coloridas. Nos cenários Dungeon, MazeCoin e Zelda, caso haja duas ou mais regiões coloridas com cores distintas, então o cenário não é considerado ideal. Somente é

considerado um cenário ideal caso haja somente uma única região colorida pelo algoritmo.

**Algoritmo 2:** Pseudocódigo algoritmo flood-fill

```
Entrada: M, Matriz que representa os elementos do cenário.
Entrada: el, Valor do elemento que se deseja colorir.
Saída: Retorna a quantidade de regiões coloridas.
1  início
2      nregioes = 0;//Armazena a quantidade de regiões coloridas;
3      para cada elemento n ∈ M faça
4          se n = el então
5              Defina uma fila Q;
6              Adicione n para o final de Q;
7              nelementos = 0;//Armazena a quantidade de elementos coloridos;
8              enquanto Q não está vazia faça
9                  Remova o primeiro elemento de Q e armazene em n;
10                 se n está em M então
11                     visitaNo(n) //Define o nó como visitado;
12                     nelementos = nelementos + 1;
13                     Adicione um nó igual el a oeste de n no final de Q;
14                     Adicione um nó igual el a leste de n no final de Q;
15                     Adicione um nó igual el ao norte de n no final de Q;
16                     Adicione um nó igual el ao sul de n no final de Q;
17                 fim se
18             fim enquanto
19         fim se
20         se nelementos > 0 então
21             nregioes = nregioes + 1 //Contabiliza a quantidade de regiões coloridas;
22         fim se
23     fim para
24     retorna nregioes;
25 fim
```

Diferentemente do trabalho de (KHALIFA et al., 2020), (GISSLEN et al., 2021) e (SHU; LIU; YANNAKAKIS, 2021) não utilizamos um agente para testar se o nível é jogável, já que, o agente de *level design* utiliza modelos fornecidos por um especialista humano em *level design*. As condições especificas de cada cenário são descritas a seguir.

- Dungeon: É considerado como cenário satisfeito quando o algorítimo verifica se há um caminho passando por todas as áreas do cenário.
- MazeCoin: O cenário é considerado satisfeito quando o algoritmo verifica se há um caminho que passa por todas as áreas do cenário. No entanto, outros elementos são levados em consideração, tais como, a existência da quantidade mínima ou máxima de moedas (1 a 2) e um único personagem controlado pelo jogador.
- Zelda: Neste cenário, é verificado se há um caminho passando por todas as áreas do ambiente, assim como no ambiente Dungeon. Isso é necessário para garantir que

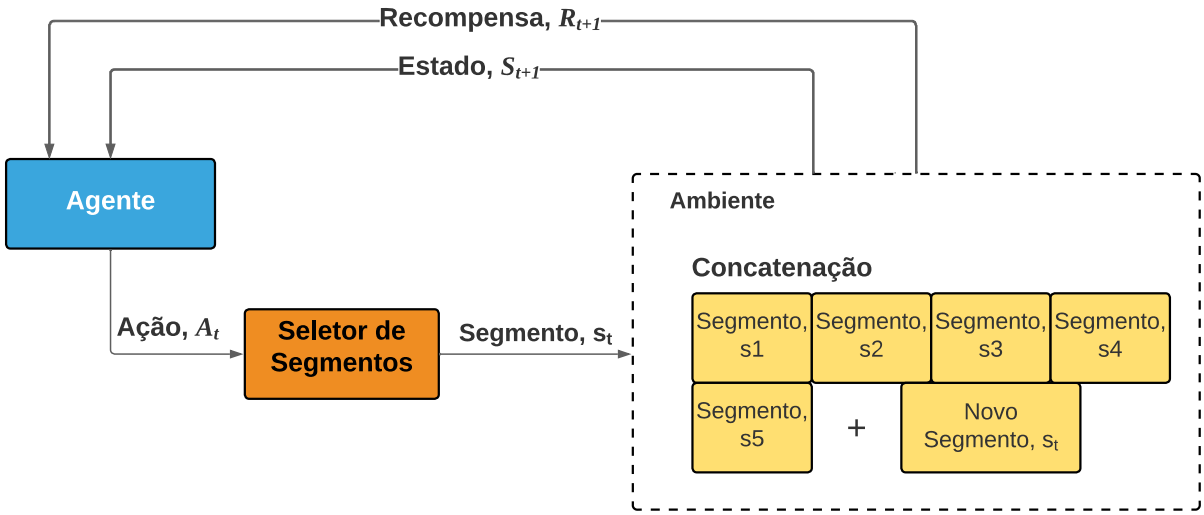
o jogador seja capaz de chegar em todas as áreas e que não haja locais totalmente bloqueados. No entanto, o ambiente Zelda possui diversos elementos que são características de um jogo, e isso deve ser considerado ao construir o cenário. Além disso, devemos considerar a existência de uma quantidade mínima ou máxima de inimigos (2 a 6), uma quantidade específica de moedas (4 a 10), uma única espada, um único personagem controlado pelo jogador, uma chave, uma entrada e uma saída.

A figura 13 ilustra o processo de concatenação dos níveis durante o treinamento do agente. O agente inicia a construção do cenário a partir de um ambiente vazio, observa o estado atual e a partir disso, envia uma ação ($A_t$) para o componente **Seletor de Segmentos** que realiza uma comunicação com o ambiente. O componente **Seletor de Segmentos** é responsável por selecionar um segmento de acordo com a ação do agente ($A_t$), que é concatenada no ambiente. Após a concatenação de cada segmento, é enviado para o agente uma recompensa ($R_{t+1}$) e um novo estado ($S_{t+1}$). O loop continua até que o agente consiga construir o cenário ou termine a quantidade de alterações máxima realizada no ambiente.

O agente de RL escolhe uma ação que correspondente ao número de segmentos implementados pelo especialista. Por exemplo, se o ambiente possui 40 segmentos, então, temos a possibilidade do agente escolher entre 40 possíveis ações a serem passadas para o componente **Seletor de Segmentos**, nesse caso, cada ação, representa um segmento. Importante citar que, em nossos experimentos, implementamos cenários que possuem um total de 6 e 8 segmentos. A concatenação dos níveis pode ser realizada utilizando uma estratégia chamada Narrow Puzzle (detalhada na seção 4.2), similar a implementação do trabalho de (KHALIFA et al., 2020). O espaço de observação é único para todos os cenários, representada por uma matriz contendo todos os segmentos adicionados.

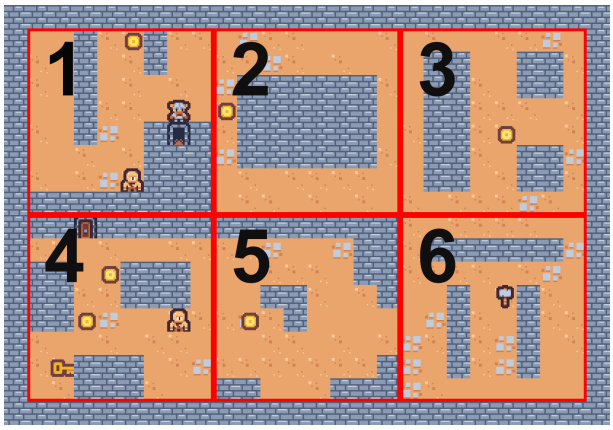
A figura 14, exibe em vermelho as áreas com segmentos utilizados para compor o nível dos jogos. Cada número representa um segmento ou uma ação realizada pelo agente durante a construção do cenário.

Figura 13 – Visão geral ilustrando como os níveis são concatenados.




Fonte: Elaborada pelo autor (2022).

Figura 14 – Ilustração demonstrando as áreas dos segmentos no jogo Zelda para um cenário com 6 segmentos.




Fonte: Elaborada pelo autor (2022).

## 4.2 ESTRATÉGIA NARROW PUZZLE

Baseado no trabalho de (KHALIFA et al., 2020), implementamos uma estratégia para alterações dos cenários para ser aplicada pelos agentes durante o treinamento, chamada de Narrow, que aqui, foi adicionado a palavra Puzzle, já que, a construção dos níveis propostas para esse trabalho se assemelha a jogos de quebra-cabeças. Para cada alteração no cenário é contabilizado a quantidade de alterações que o agente realizou ao longo do treinamento até finalizar a construção do cenário.

Essa é um tipo de estratégia que altera o ambiente dada a localização atual do agente na matriz que representa o cenário e a cada etapa de construção. Como os cenários

são construídos a partir de segmentos que fazem parte de um conjunto de ações ($a \in A$), o agente altera uma parte do cenário respeitando a dimensão dos segmentos. Diferentemente do trabalho de (KHALIFA et al., 2020) que realiza as alterações no cenário tile por tile, ou seja, objeto por objeto, neste trabalho, a construção do cenário ocorre em blocos de matrizes de dimensão $S_{mn} = 8 \times 8$ respeitando os modelos implementados pelo especialista humano em *level design*.

Caso o cenário possua uma dimensão total $M_{mn} = 24 \times 16$, a alteração ocorrerá a cada 8 linhas e 8 colunas. Em outras palavras, os segmentos são adicionados em sequência respeitando a quantidade total máxima de segmentos que o cenário suporta. Em nosso trabalho, são um total de 6 e 8 peças, sendo representada por Sn, formando um tabuleiro de quebra-cabeças $2 \times 3$ (2 linhas e 3 colunas) e $2 \times 4$ (2 linhas e 4 colunas).

O agente realiza um looping, colocando segmento por segmento até chegar a última posição. Caso a construção do cenário não atinja os objetivos, o agente reinicia a sua posição a partir da primeira peça do tabuleiro de quebra-cabeças substituindo cada segmento. A cada segmento substituído é verificado se as condições do cenário foram satisfeitas. Ao final da construção do cenário temos uma matriz (vide a equação 4.1) com os valores dos segmentos selecionados representando o espaço de observação para o agente de aprendizado por reforço.

$$S_{m \times n} = \begin{bmatrix} S_1 & S_2 & \cdots & S_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ S_{m1} & S_{m2} & \cdots & S_{mn} \end{bmatrix} \tag{4.1}$$

## 4.3 FUNÇÕES DE RECOMPENSAS

As recompensas são um elemento fundamental em aplicações de RL. As funções de recompensa são responsáveis por informar se o agente obteve sucesso ou falha ao executar um determinada ação no ambiente. Ao se projetar as funções de recompensa deve-se determinar as regras para o agente receber as recompensas positivas ou negativas. Recompensas mal formuladas podem fazer com que o agente não consiga obter um treinamento com resultados satisfatórios.

Para o projeto desta dissertação foram implementadas diversas funções de recompensa que o agente recebe como feedback enquanto está construindo o cenário dos jogos. Os tipos de funções de recompensas são descritas nos tópicos a seguir.

### 4.3.1 Entropy

Entropy é a recompensa base recebida pelo agente quando o cenário é finalizado. Essa medida permite avaliar a quantidade de informação o cenário possui, considerando a

presença ou não de segmentos repetidos. Quanto maior o número de segmentos repetidos, menor o valor da entropia e, quanto menor o número de repetições, maior será o seu valor. Desta forma, é possível avaliar e mensurar a qualidade do cenário a partir da variação dos segmentos, diferentemente do trabalho de (SHU; LIU; YANNAKAKIS, 2021) que utiliza KL-divergence (LUCAS; VOLZ, 2019) para quantificar a similaridade entre os segmentos.

Ao verificar se o cenário foi concluído, são informados quais segmentos foram gerados, $S_n = S_0, S_1, S_2, ...S_n$. Para calcular a entropia com base nos segmentos dos cenários, é necessário seguir os seguintes passos:

1. Calcule a distribuição de probabilidade dos segmentos $X = x_0, x_1, x_2, ...x_n \in S_n$.
2. Aplique a equação da entropia $H = -\sum_i^n p(x_i) \log(p(x_i))$

onde $p(x_i)$ é a probabilidade de ocorrência de cada evento para a variável $x$, neste caso, representado por cada segmento. A quantidade de informação que cada evento ou segmento possui é calcula por $log(p(x_i))$.

Para entendermos melhor essa recompensa, considere que o agente ao final da iteração conseguiu construir um cenário e selecionou os seguintes segmentos:

$$S = \{16, 32, 52, 16, 1, 0\}. \tag{4.2}$$

Ao aplicar a equação da entropia, será calculado a distribuição de probabilidade dos segmentos conforme a tabela 3.

Tabela 3 – Distribuição de probabilidade dos segmentos com o valor de entropia $H \approx 2.25$ bits de informação

| **Segmentos** | **16(2)** | **32(1)** | **52(1)** | **1(1)** | **0(1)** |
|---|---|---|---|---|---|
| | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ |
| **Probabilidade** | 0,33 | 0,16 | 0,16 | 0,16 | 0,16 |
| $\mathbf{log(x_i)}$ | $\approx -1,58$ | $\approx -2,58$ | $\approx -2,58$ | $\approx -2,58$ | $\approx -2,58$ |

Seguindo a formula da equação, temos:

$$H(x) = -((0.33 \times -1.58) + (0.16 \times -2.58) + (0.16 \times -2.58) \tag{4.3}$$

$$+(0.16 \times -2.58) + (0.16 \times -2.58)) \approx 2.25 \tag{4.4}$$

Precisamente temos um valor de entropia $H = 2,251629167$ que o agente receberá como recompensa referente a construção do cenário gerado. É importante notar que o valor da recompensa varia dependendo da quantidade máxima de segmentos do ambiente, sendo de 0 a $2,58$ para ambientes com 6 segmentos e de 0 a $3,00$ para ambientes com 8 segmentos.

### 4.3.2 Entropy Quality

Entropy Quality é um tipo de recompensa utilizada mensurar quais cenários possuem um valor de entropia ideal com base na auditoria de um especialista humano em *level design*. Com base no valor de entropia, visto na seção **4.3.1**, o especialista humano em *level design* pode definir um valor ideal de qualidade para o cenário gerado em relação à diversidade dos segmentos. Para o cálculo dessa recompensa utilizamos a seguinte equação 4.5 e 4.6:

$$\mathrm{R} = \mathrm{H}^{x} - \mathrm{HQ}^{x} \tag{4.5}$$

$$\mathrm{F}\left(\mathrm{R}\right) = \begin{cases} \mathrm{R} = \mathrm{R} + (-1) & \mathrm{R} < 0 \\ \mathrm{R} = \mathrm{R} + 1 & \mathrm{R} \geq 0 \end{cases} \tag{4.6}$$

onde H representa o valor da entropia do cenário gerado e HQ (Entropy Quality) refere-se ao valor da qualidade da entropia ideal definida pelo especialista. O parâmetro $x$ é uma constante, em nosso trabalho fixamos com o valor de $pi = 3,14159265$. A equação calcula a diferença entre o valor de entropia definido pelo especialista e o valor de entropia calculada para o cenário gerado. Após calcular a recompensa da equação 4.5 realizamos a seguinte verificação na equação 4.6: se o valor da recompensa ($R$) for menor que zero adicionamos um valor negativo $-1$ para aplicar uma penalidade maior se a construção do cenário não estiver próximo do ideal, caso contrario, adicionamos $+1$. Adicionar o valor $+1$ na recompensa é necessário, pois, pode ocorrer do resultado da equação 4.5 retornar 0 caso o valor de entropia do cenário gerado ($H$) seja igual ao parâmetro $HQ$.

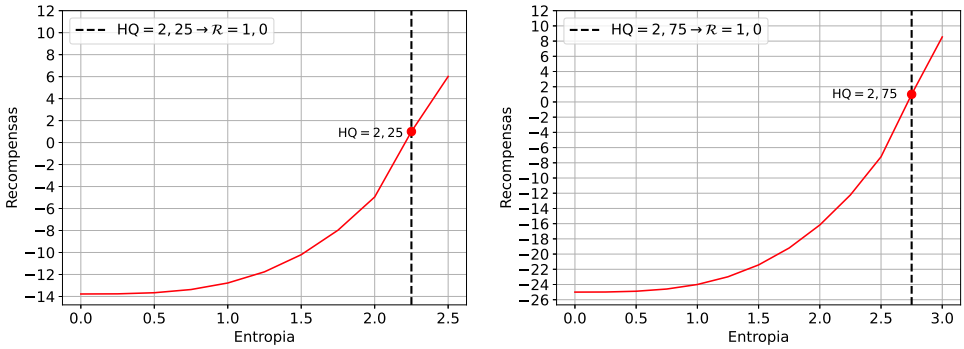
A figura 15 ilustra graficamente como é aplicado a recompensa. Observe que, quanto menor é o valor da entropia, maior é o valor da penalidade. No gráfico da figura 15 é traçado um valor mínimo de $\mathrm{HQ} = 2,25$ (vide figura 15a) e $\mathrm{HQ} = 2,75$ (vide figura 15b) que indica o ponto ideal de qualidade do cenário. Caso o cenário possua um valor de entropia maior ou igual ao ideal, então é considerado um bom nível, dessa forma, o agente recebe um feedback positivo. Importante citar que, esse valor considerado ideal é com base em cenários cujo tamanho do "tabuleiro" onde são adicionados os valores dos segmentos é igual a uma matriz $2 \times 3$ (6 segmentos, vide figura 15a). Para um "tabuleiro" $2 \times 4$ (8 segmentos, vide figura 15b) o valor de entropia ideal é $\mathrm{HQ} = 2,75$.

### 4.3.3 Change Penalty

É o tipo de penalidade recebida enquanto o agente está construindo os cenários. A recompensa negativa (vide equação 4.7), recebida pelo agente, é proporcional a quantidade de alterações no ambiente, incentivando-o a criar um cenário com a menor quantidade de alterações possíveis, respeitando a quantidade miníma de segmentos do ambiente.

$$R = -0.1 \tag{4.7}$$

Figura 15 – Gráfico demonstrando a evolução da recompensa




(a) Ambiente com 6 segmentos (b) Ambiente com 8 segmentos

O valor total da penalidade recebida dependerá da quantidade máxima de alterações permitidas, representada por C, que o agente pode realizar. Em nossos experimentos fixamos o valor máximo das alterações em $21, 61$ e $180$. A penalidade é recebida a cada etapa de construção do cenário, neste caso, o valor da recompensa é acumulada ao longo do tempo, conforme a equação 4.8:

$$C_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{4.8}$$

A quantidade máxima de alterações permite que o agente explore o ambiente de forma mais completa, aumentando suas chances de encontrar uma solução ótima para o problema. Sem essa limitação, o agente pode permanecer alterando o ambiente infinitamente impedindo de encontrar uma solução ideal.

## 4.4 AGENTES

Para esta proposta de dissertação, foram desenvolvidos três agentes ($\mathbf{A_S}$, $\mathbf{A_H}$ e $\mathbf{A_{HQ}}$), cada um projetado com uma estratégia de recompensa distinta, com o objetivo de avaliar sua eficácia na geração de cenários. A tabela 4 apresenta os tipos de agentes implementados e suas respectivas funções de recompensa:

Tabela 4 – Tipos de agentes e suas respectivas recompensas utilizadas durante o treinamento

| **Agentes** | **Tipos de Recompensa** | | |
|---|---|---|---|
| | **Entropy** | **Entropy Quality** | **Change Penalty** |
| $\mathbf{A_S}$ | | | X |
| $\mathbf{A_H}$ | X | | X |
| $\mathbf{A_{HQ}}$ | X | X | X |

As notações $\mathbf{A_S}$, $\mathbf{A_H}$ e $\mathbf{A_{HQ}}$ refere-se ao agente treinado usando uma ou mais funções de recompensa especifica.

Considerações pontuais devem ser destacadas em relação ao agente $\mathbf{A_S}$. Ao contrário dos demais agentes, ele não utiliza a recompensa base *Entropy*. Em vez disso, é atribuído ao agente um valor fixo como recompensa positiva, considerando o valor máximo de entropia para um determinado cenário. Para nossos experimentos com cenários de 6 e 8 segmentos, H$max \approx 2,58$ e H$max = 3,0$, respectivamente.

A implementação do algoritmo para calcular as recompensas dos agentes é simples. Para cada agente, o algoritmo é ligeiramente modificado para usar uma função de recompensa específica durante o treinamento. Os algoritmos 3 e 4 apresentam a implementação completa das funções de recompensa para os agentes $\mathbf{A_H}$ e $\mathbf{A_{HQ}}$ respectivamente.

**Algoritmo 3:** Formulação do algoritmo para o cálculo da recompensa Entropy e ChangePenalty.

**Entrada**: $\mathcal{S}$, Vetor contendo os segmentos selecionados.
**Saída**: O valor da recompensa

```
1  início
2      r = 0;
3      se Cenário não está finalizado então
4          r = ChangePenalty();
5      senão
6          H = Entropy(S);
7          r = H;
8      fim se
9      retorna r
10 fim
```

**Algoritmo 4:** Formulação do algoritmo para o cálculo da recompensa Entropy Quality.

**Entrada**: HQ, Valor de entropia para a qualidade dos cenários.
**Entrada**: $\mathcal{S}$, Vetor contendo os segmentos selecionados.
**Entrada**: $x = 3.14159265$, Constante para elevar a potência o valor da entropia. Valor padrão definido como pi.

```
início
    r = 0;
    se Cenário não está finalizado então
        r = ChangePenalty();
    senão
        H = Entropy(S);
        r = H^x − HQ^x;
        se r < 0 então
            r = r + (−1);
        senão
            se r ≥ 0 então
                r = r + 1;
            fim se
        fim se
    fim se
    retorna r
fim
```

## 4.5 LINEARIDADE E LENIÊNCIA

Como o trabalho proposto é um design iniciativa-mista que aproxima geração procedural de conteúdo e aprendizado por reforço, é muito importante examinar a qualidade dos níveis gerados pelos agentes. Nesta seção é apresentado como os resultados para os ambientes que possuem objetivo de gameplay (MazeCoin e Zelda) são analisados via *expressive range* com base nos trabalhos de (SMITH; WHITEHEAD; MATEAS, 2011) e (VIANA et al., 2022). Aqui, verificamos a qualidade dos cenários gerados em relação à linearidade e ao grau de dificuldade dos níveis gerados. A primeira métrica utilizada é a linearidade que aqui utilizamos para determinar o quão linear é o nível com relação à estética. Como segunda avaliação utilizamos a leniência para medir o grau de dificuldade do nível gerado. A dificuldade de um nível é altamente subjetiva, por depender da experiência individual de cada jogador. A leniência aqui é uma medida mais objetiva utilizada para analisar a dificuldade do nível de acordo com elementos existentes em cada jogo.

Cada ambiente possui características diferentes, desta forma, a avaliação para mensurar a linearidade e leniência dos ambientes MazeCoin e Zelda devem ser calculadas individualmente. Para os ambientes MazeCoin e Zelda a linearidade dos níveis gerados (vide equação 4.9), é calculada multiplicando a quantidade de segmentos (Sn) utilizados

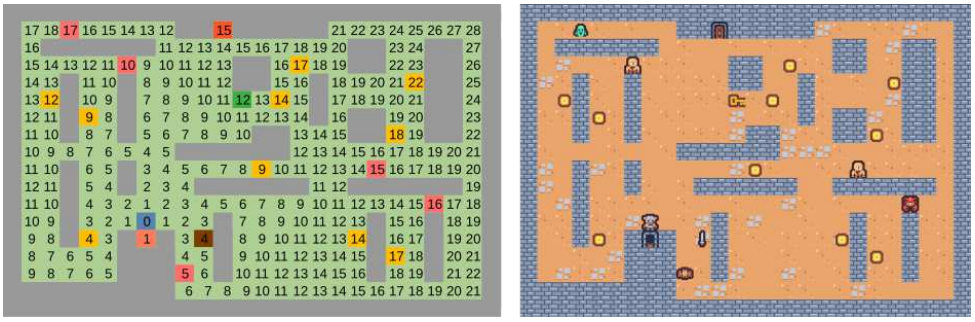
pelo resultado da diferença entre a entropia máxima ($Hmax$) que um nível pode possuir e o valor de entropia do nível gerado ($H$). O cenário de 6 e 8 segmentos temos o valor máximo de entropia H$max \approx 2,58$ e H$max = 3,0$ respectivamente.

$$linearidade = \text{Sn} \times (Hmax - H) \tag{4.9}$$

Com relação à leniência, para cada cenário gerado discretizamos o ambiente (vide figura 16). Discretizamos o ambiente para podermos calcular a distância entre o personagem jogável, representado na matriz pelo valor 0 (zero), até os demais elementos do jogo. Para discretizar o ambiente utilizamos um algoritmo de propagação chamado *wavefront* utilizado para planejamento de caminhos. Esse algoritmo realiza uma propagação em ondas que representam a distância entre o personagem a qualquer ponto no ambiente e é um eficiente método para planejar o menor caminho em um mapa representado por uma matriz (ZIDANE; IBRAHIM, 2018).

A partir de um mapa representado por uma matriz, em nosso caso, um cenário, marcamos na matriz o personagem e os obstáculos (paredes). Para o personagem na matriz definimos o valor 0 (zero). Após marcar a posição do personagem utilizamos o algoritmo *wavefront* para definir todos os trajetos possíveis na matriz. Ao final, marcamos a localização de cada objeto no cenário. Para o cenário MazeCoin definimos a localização apenas das moedas. Para o ambiente Zelda definimos as localizações das moedas, inimigos, chave, armas e a porta de saída. A figura 16 ilustra o cenário Zelda. A área hachurada são os obstáculos que o personagem não pode atingir.

Figura 16 – Exemplo do nível Zelda com o cenário discretizado.



Fonte: Elaborada pelo autor (2022).

Com o ambiente discretizado aplicamos a seguinte equação 4.10:

$$leniencia = \frac{\sum_{i=1}^{n} Di}{MAXDIST} \times (\sum_{i=1}^{m} Ei) \times H \tag{4.10}$$

onde n é a quantidade de objetos que calculamos a distância, $Di$ é a distância calculada entre o personagem e os objetos do cenário.

O parâmetro $MAXDIST$ é a distância máxima que o personagem pode atingir no ambiente. Na segunda parte da equação, m é a quantidade de tipos de objetos (moedas, inimigos, chave e arma) existentes no cenário gerado. O parâmetro $Ei$ é a quantidade total para cada objeto no cenário e $H$ é um peso que representa o valor da entropia do cenário gerado.

Para o ambiente MazeCoin é calculado a distância entre o personagem e as moedas que devem ser coletadas pelo jogador. Em relação à quantidade de elementos existentes, verificamos a quantidade de moedas, blocos que representam o chão e as paredes. Os objetos utilizados para ambiente Zelda no cálculo da distância são os inimigos, moedas, chave, espada e saída. Com relação à quantidade de elementos existentes, são considerados as moedas, inimigos, blocos que representam o chão e as paredes.

## 4.6 TREINAMENTO

Diversos experimentos e testes foram realizados para verificar a eficiência dos agentes de RL conforme as funções de recompensas projetadas. Cada agente foi treinado utilizando o algoritmo de RL PPO[1](veja a seção **2.4.2**) que esta presente na biblioteca Stable Baselines3[2] (SB3). A escolha pelo algoritmo PPO possui as seguintes motivações:

- Modela o algoritmo *actor-critic*, desta forma, utilizando as vantagens de métodos *policy-based* e *value-based*. Métodos *policy-based* lidam facilmente com uma abundância de estados e ações. Os métodos *value-based* aprendem o valor do estado-ação. Desta forma, o agente aprende a selecionar a melhor ação no estado.
- Possibilidade de utilizar um número maior de ações discretas e continuas. Neste trabalho são utilizadas ações discretas fornecidas por cada ambiente.

Na literatura, diversos autores optam por implementar suas próprias versões do algoritmo PPO ou utilizar soluções prontas, como no trabalho de (KHALIFA et al., 2020) e (HENDERSON et al., 2018), para realizar comparações nos experimentos. Neste trabalho, optou-se também por utilizar soluções prontas, a fim de facilitar a realização dos experimentos e comparações.

A biblioteca SB3 contém um conjunto de implementações de algoritmos de *deep reinforcement learning* desenvolvidas em Python que facilitam o desenvolvimento de aplicações de RL (RAFFIN et al., 2021). O PPO, por utilizar uma rede neural para as redes *actor-critic*, optou-se como politica das escolhas das ações utilizar a topologia Multilayer Perceptron (MLP), sendo uma rede neural padrão do framework SB3. O motivo pela escolha da rede neural MLP foi baseada no artigo que originou o algoritmo PPO

[1] https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html
[2] https://stable-baselines3.readthedocs.io

(SCHULMAN et al., 2017) e no trabalho de (HENDERSON et al., 2018) que realiza diversos experimentos com algoritmos de RL utilizando essa rede.

Em relação aos hiper-parâmetros, na literatura não há consenso qual é o melhor parâmetro de configuração a ser definido para treinar os agentes PPO. Além disso, a maioria dos trabalhos não descreve claramente quais foram utilizados. Para este trabalho, optou-se inicialmente utilizar os hiper-parâmetros definidos na tabela 5 que foram baseados no artigo de (SCHULMAN et al., 2017).

Tabela 5 – hiper-parâmetros para os agentes do PPO

| **Ambientes** | **Zelda** | **MazeCoin** | **Dungeon** |
|---|---|---|---|
| **Learning rate** | 3e-4 | | |
| **Time steps** | $10^6$ | | |
| **Trajetória(*actor-critic*)** | 2048 | | |
| **Fator de Desconto ($\gamma$)** | 0,99 | | |
| **Layer x units** | 2 x 64 | | |
| **Épocas(Rede neural)** | 10 | | |
| **Minibatch(Rede neural)** | 64 | | |
| **Função de Ativação** | Sigmoid | | |

Devemos destacar três importantes parâmetros para os agentes PPO especificados na tabela 5 em relação à quantidade de camadas e unidades (*Layer x units*) da rede neural e a Trajetória(*actor-critic*). Em relação ao parâmetro *Layer x units*, estamos definindo que as redes neurais *actor* e *critic* possuem a mesma arquitetura, ou seja, duas camadas ocultas com 64 unidades compartilhadas. O segundo parâmetro Trajetória(*actor-critic*), é o tamanho do espaço de tempo para o *actor* coletar os dados para treinamento da rede neural e auxiliar nas escolhas das ações. Para mais detalhes veja a seção **2.4.2**. Por fim, as redes *actor* e *critic* utilizam a mesma função de ativação Sigmoid.

A tabela 6 apresenta os parâmetros relacionados ao espaço de observação, ao número de ações, ao número máximo de segmentos necessários para construir um cenário completo (denotado por Sn), e à quantidade máxima de alterações permitidas que os agentes podem realizar no cenário (denotado por C).

Um importante parâmetro para o agente $\mathbf{A_{HQ}}$ deve ser fornecido, a qualidade da entropia, definida como, HQ. Para os experimentos, fixamos HQ $= 2,25$ para cenários com 6 segmentos e HQ $= 2,75$ para cenários com 8 segmentos, valores analisados por um especialista humano em *level design* como adequados para a geração dos cenários. É importante destacar que esses valores de entropia consideram a presença de pelo menos um segmento repetido no cenário. O algoritmo 5 demonstra o procedimento para inferência após treinamento dos agentes.

Tabela 6 – Parâmetros referentes ao espaço de observação, ações, quantidade de segmentos e a quantidade máxima de alterações permitidas nos cenários.

| **Ambientes** | **Dungeon** | **MazeCoin** | **Zelda** |
|---|---|---|---|
| **Espaço de Observação** | Matriz de números inteiros $S_{mn}$ armazenando os valores dos segmentos selecionados. | | |
| **Ações** | 40 | 80 | 300 |
| **Sn** | 6/8 | 6/8 | 6/8 |
| **C** | 21/61/180 | 21/61/180 | 21/61/180 |

**Algoritmo 5:** Procedimento para inferência após o treinamento dos agentes em qualquer cenário

**Entrada**: $\mathcal{S}$, Vetor de segmentos modelados por um especialista.
**Entrada**: *Agente*, Tipo de Agente($\mathbf{A_S}$, $\mathbf{A_H}$ ou $\mathbf{A_{HQ}}$).
**Entrada**: $\mathcal{M}$, Matriz que representa os elementos do cenário.
**Entrada**: $T$, Quantidade máxima de tempo para inferência.
**Entrada**: C, Quantidade máxima de alterações permitidas no cenário.
**Entrada**: Sn, Quantidade máxima de segmentos permitidos para criar os cenários.

**início**
  $t = 0$;
  **enquanto** $t < T$ **faça**
    $alteracoes = 0$;
    $finalizado = Falso$;
    $\mathcal{M} = []$;
    **enquanto** $alteracoes \leq$ C ***ou*** *cenário não estiver finalizado* **faça**
      $a_n = selecionaAcao()$ //Agente PPO seleciona uma ação;
      $s = \mathcal{S}(a_n)$; //Seleciona um segmento de acordo com a ação;
      $finalizado = constroiCenario(s, \mathcal{M}, \text{Sn})$ //Adiciona $s$ em $\mathcal{M}$;
      $alteracoes = alteracoes + 1$; //Calcula a quantidade de alterações;
      $r = Recompensa(Agente)$; //Calculada de acordo com o agente selecionado;
      Retorna o valor da recompensa, $r$ para o agente PPO
    **fim enquanto**
    $t = t + 1$;
  **fim enquanto**
**fim**

# 5 RESULTADOS EXPERIMENTAIS

Esse capítulo visa discutir e apresentar os resultados dos experimentos realizados para cada agente formulados na seção 4.4. As condições nas quais os experimentos foram realizados são apresentados na seção 4.6.

Para desenvolver a solução do problema proposto, utilizamos a linguagem Python, a *engine* Pygame[1] para desenvolvimento dos jogos em 2D e a *toolkit* OpenAI Gym[2]. Gym é uma *toolkit* para desenvolvimento e comparação de algoritmos de aprendizado por reforço. Em nosso trabalho, a utilizamos para desenvolver todos os ambientes utilizados para treinamento descritos nas seções a seguir. A nossa solução intitulamos o framework de PCGRLPuzzle[3]. A execução do treinamento e inferência dos dados foram realizados na plataforma Google Colaboratory[4](Colab).

Como primeiro passo, será discutido os resultados dos experimentos obtidos e avaliado o desempenho de cada agente durante a criação dos cenários utilizando uma quantidade de segmentos diferentes. Em ambos os ambientes, cada agente aprende a construir os cenários de acordo com suas funções de recompensa. Aqui, analisaremos como as funções de recompensa podem influenciar nos resultados, bem como a quantidade de segmentos que cada cenário possui. Para facilitar a compreensão, os resultados serão apresentados separadamente em relação à quantidade de segmentos utilizados nos cenários. Devemos destacar que não estamos comparando os resultados obtidos com os trabalhos de (KHALIFA et al., 2020), (SHU; LIU; YANNAKAKIS, 2021) e (GISSLEN et al., 2021), pois cada uma das propostas utiliza métodos e ambientes diferentes.

O segundo objetivo é avaliar os questionamentos definidos na seção 1.2. A primeira questão definida é sobre a possibilidade de treinar agentes de RL a partir de modelos fornecidos por um especialista humano em *level design*. Os resultados dos experimentos descritos nesse capítulo demonstram que isso é possível. A segunda questão definida é como mensurar a diversidade dos níveis gerados pelos agentes de level designer. Os experimentos realizados nesse trabalho indicam ser possível mensurar a diversidade dos níveis utilizando a entropia.

Antes de executar qualquer experimento, foi necessário definir como verificar a desempenho e qualidade dos resultados gerados pelos agentes treinados. Na seção 4.6 foram definidos os parâmetros de treinamento para os agentes, bem como, a quantidade de cenários que cada agente deve gerar. Como critério para inferência dos treinamentos, cada agente foi definido para gerar 1000 cenários em cada ambiente. Isso é necessário para avaliar como as funções de recompensa projetadas afetam os resultados do treinamento em

[1] https://www.pygame.org/
[2] https://gym.openai.com/
[3] https://github.com/dutrapaulovm/pcgrl-puzzle
[4] https://colab.research.google.com/

relação à quantidade de cenários gerados, aos segmentos utilizados e ao valor de entropia de cada um.

Na primeira análise dos resultados, utilizamos várias medidas de avaliação: quantidade e porcentagem de segmentos utilizados, quantidade e porcentagem de cenários gerados, média, mediana e desvio padrão da entropia. Para calcular a porcentagem em relação à quantidade de cenários gerados, consideramos o valor relativo a uma tentativa dos agentes gerarem 1000 cenários. Em relação à diversidade dos cenários, definimos como valor de entropia mínimo ideal para os cenários com 6 e 8 segmentos $H \geq 2,25$ e $H \geq 2,75$ respectivamente. Isso implica que, haverá 1 segmento repetido. Como segunda análise, a qualidade dos resultados para os ambientes MazeCoin e Zelda é verificada via *expressive range*, visando avaliar a linearidade e a leniência dos níveis gerados como nos trabalhos de (VIANA et al., 2022) e (SMITH; WHITEHEAD; MATEAS, 2011).

Por fim, nas seções 5.1 e 5.2 os resultados analisados são referentes ao parâmetro de treinamento C, com os valores 21, 61 e 180. Os resultados completos do treinamento podem ser acessados no seguinte link: <https://github.com/dutrapaulovm/pcgrlpuzzle-results>

## 5.1 AMBIENTES COM 6 SEGMENTOS

Nesta seção, nós apresentamos os resultados obtidos por cada agente e para cada ambiente em relação a quantidade e diversidade dos cenários gerados. Para um melhor entendimento, os dados apresentados na tabela 7 são respectivamente a quantidade de cenários gerados, porcentagem de segmentos utilizados e porcentagem de cenários com entropia abaixo de $H < 2,25$ e maior ou igual a $H \geq 2,25$. A porcentagem calculada é relativa a quantidade de cenários gerados por cada agente em seus respectivos ambientes. A tabela 8, apresenta informações sobre a media, mediana e desvio padrão da entropia dos cenários gerados. Os valores em destaque apresentam os melhores resultados para cada agente. Estamos considerando como bons resultados, agentes que conseguiram gerar pelo menos 80% (800) dos cenários com valores de entropia $H \geq 2,25$ e utilizar uma porcentagem de segmentos maior ou igual 50%, e cuja média e mediana da entropia seja $H \geq 2,25$. As imagens referentes aos cenários gerados por cada agente utilizando o valor do parâmetro $C = 61$ podem ser vistas no apêndice A.

Em relação aos dados apresentados na tabela 7, podemos observar que ambos os agentes obtiveram sucessos em gerar cenários com uma quantidade total ou aproximado de 1000 para cada um dos parâmetros. No entanto, ao aumentar o valor do parâmetro C para 180 no ambiente Zelda, foi gerada uma quantidade de cenários próxima a 1000, enquanto os demais parâmetros apresentaram resultados inferiores. Suspeitamos que isso ocorra devido ao fato dos agentes possuírem uma quantidade maior de tentativas para conseguir completar o cenário. Como segunda hipótese, devido ao fato da função objetivo

possuir mais componentes de restrição comparado a outros ambientes. Neste caso, os parâmetros C = 21 e C = 61 no ambiente Zelda precisam de mais modelos ou um tempo maior de treinamento para gerar uma quantidade maior de cenários, como ocorre com o parâmetro C utilizando o valor 180.

Além disso, é importante destacar a porcentagem dos segmentos utilizados por cada ambiente e agente em nossa análise. No ambiente Dungeon, apenas os agentes $\mathbf{A_H}$ e $\mathbf{A_{HQ}}$ foram capazes de utilizar uma porcentagem próxima a 100% dos segmentos, variando entre $92,50\%$ e $97,50\%$. Para o ambiente MazeCoin, a porcentagem de segmentos utilizados variou entre $65,00\%$ e $86,25\%$, enquanto para o ambiente Zelda, variou entre $74,00\%$ e $79,67\%$.

Ao analisar os dados referentes à porcentagem de cenários gerados com valor de entropia maior ou igual a H = 2,25, apenas os agentes $\mathbf{A_H}$ e $\mathbf{A_{HQ}}$ obtiveram sucesso em gerar cenários com uma porcentagem variando entre $84,70\%$ e 100%. Esses resultados sugerem que esses agentes são mais eficazes na geração de cenários com alta complexidade, enquanto o agente $\mathbf{A_S}$ pode apresentar limitações por não haver uma recompensa aplicada adequadamente.

Tabela 7 – Resultados obtidos por cada agente nos ambientes Dungeon, MazeCoin e Zelda. A tabela exibe a quantidade de cenários (L), a porcentagem de segmentos utilizados (S), porcentagem de sucesso de cenários gerados com entropia menor e maior ou igual a H = 2,25.

| **Parâmetro** | **Agentes** | **Dungeon** | | | | **MazeCoin** | | | | **Zelda** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *L* | *S* | H < 2,25 | H ≥ 2,25 | *L* | *S* | H < 2,25 | H ≥ 2,25 | *L* | *S* | H < 2,25 | H ≥ 2,25 |
| **C = 21** | $\mathbf{A_S}$ | 1000 | 57,50 | 92,40 | 7,60 | **939** | **71,25** | **17,68** | **82,32** | 396 | 42,00 | 0,76 | 99,24 |
| | $\mathbf{A_H}$ | **1000** | **92,50** | **15,30** | **84,70** | **939** | **70,00** | **6,71** | **93,29** | 270 | 45,33 | 0,74 | 99,26 |
| | $\mathbf{A_{HQ}}$ | **959** | **97,50** | **7,92** | **92,08** | **870** | **78,75** | **2,53** | **97,47** | 302 | 37,33 | 2,32 | 97,68 |
| | | | | | | | | | | | | | |
| **C = 61** | $\mathbf{A_S}$ | 1000 | 55,00 | 91,30 | 8,70 | 1000 | 67,50 | 58,60 | 41,40 | **627** | **71,00** | **2,07** | **97,93** |
| | $\mathbf{A_H}$ | **1000** | **95,00** | **9,90** | **90,10** | **1000** | **65,00** | **5,30** | **94,70** | **520** | **79,67** | **0,19** | **99,81** |
| | $\mathbf{A_{HQ}}$ | **1000** | **97,50** | **8,10** | **91,90** | **999** | **86,25** | **1,70** | **98,30** | **602** | **63,00** | **0,33** | **99,67** |
| | | | | | | | | | | | | | |
| **C = 180** | $\mathbf{A_S}$ | 1000 | 50,00 | 100,00 | 0,00 | 1000 | 70,00 | 39,10 | 60,90 | **912** | **75,67** | **0,55** | **99,45** |
| | $\mathbf{A_H}$ | **1000** | **92,50** | **12,80** | **87,20** | **1000** | **71,25** | **7,50** | **92,50** | **949** | **74,00** | **0,00** | **100,00** |
| | $\mathbf{A_{HQ}}$ | **1000** | **97,50** | **7,50** | **92,50** | **1000** | **80,00** | **2,50** | **97,50** | **983** | **68,33** | **0,92** | **99,08** |

Analisando os dados da tabela 8, é possível observar que a aplicação de uma função de recompensa adequada, como utilizada pelos agentes $\mathbf{A_H}$ e $\mathbf{A_{HQ}}$, permitiu a geração de cenários com média e mediana da entropia variando entre H = 2,25 e H = 2,58

Particularmente, ao analisarmos os dados obtidos referente ao agente $\mathbf{A_S}$ (vide tabelas 7 e 8) para os ambiente Dungeon e MazeCoin verificamos que a medida que aumentamos o valor do parâmetro C o agente tende a gerar uma porcentagem maior de cenários abaixo do valor de entropia H = 2,25. O valor abaixo do esperado indica para cada cenário uma quantidade maior de repetições nos segmentos. O agente $\mathbf{A_S}$ não aplica um valor de recompensa adequada conforme a diversidade dos cenários como ocorre para os agentes $\mathbf{A_H}$ e $\mathbf{A_{HQ}}$. Aplicando o mesmo valor de recompensa positiva para cenários

com baixa e alta diversidade. A quantidade de tentativas para o agente construir cada cenário também influência na recompensa final quando o cenário é considerado concluído. Desta forma, quanto maior a quantidade de tentativas maior será a penalidade, com isso, o agente busca selecionar sempre que possível, segmentos iguais para gerar os cenários e evitar que a penalidade seja maior.

Tabela 8 – Resultados obtidos por cada agente nos ambientes Dungeon, MazeCoin e Zelda. A tabela exibe a média (ME), mediana (MD) e desvio padrão (DP) da entropia dos níveis.

| **Parâmetro** | **Agentes** | **Dungeon** | | | **MazeCoin** | | | **Zelda** | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | *ME* | *MD* | *DP* | *ME* | *MD* | *DP* | *ME* | *MD* | *DP* |
| **C = 21** | $\mathbf{A_S}$ | 1,53 | 1,46 | 0,41 | **2,27** | **2,25** | **0,28** | 2,52 | 2,58 | 0,14 |
| | $\mathbf{A_H}$ | **2,33** | **2,25** | **0,26** | **2,40** | **2,58** | **0,22** | 2,47 | 2,58 | 0,16 |
| | $\mathbf{A_{HQ}}$ | **2,39** | **2,58** | **0,23** | **2,46** | **2,58** | **0,18** | 2,50 | 2,58 | 0,17 |
| | | | | | | | | | | |
| **C = 61** | $\mathbf{A_S}$ | 1,59 | 1,46 | 0,39 | 1,94 | 1,79 | 0,38 | **2,49** | **2,58** | **0,17** |
| | $\mathbf{A_H}$ | **2,37** | **2,25** | **0,23** | **2,42** | **2,58** | **0,21** | **2,54** | **2,58** | **0,11** |
| | $\mathbf{A_{HQ}}$ | **2,39** | **2,58** | **0,23** | **2,48** | **2,58** | **0,17** | **2,55** | **2,58** | **0,11** |
| | | | | | | | | | | |
| **C = 180** | $\mathbf{A_S}$ | 0,58 | 0,65 | 0,49 | 2,10 | 2,25 | 0,37 | **2,54** | **2,58** | **0,12** |
| | $\mathbf{A_H}$ | **2,35** | **2,25** | **0,25** | **2,41** | **2,58** | **0,23** | **2,56** | **2,58** | **0,08** |
| | $\mathbf{A_{HQ}}$ | **2,41** | **2,58** | **0,24** | **2,47** | **2,58** | **0,18** | **2,52** | **2,58** | **0,14** |

Uma segunda análise é em relação a entropia mínima e máxima gerada em ambos os ambientes, conforme apresentado na tabela 9. A entropia é uma medida de diversidade que indica a quantidade de informação contida em um conjunto de dados. No caso deste estudo, quanto maior a entropia, mais diversificados são os cenários gerados pelos agentes. As informações entre parenteses apresentadas na tabela 9 representam a porcentagem de entropia miníma e máxima gerada pelos agentes.

Para um melhor entendimento dos dados, é importante considerar que o valor máximo de entropia que um cenário pode possuir é H $= 2,58$, indicando que foram gerados cenários com segmentos totalmente diferentes uns dos outros. O valor mínimo de entropia H $= 0,00$ indica que os cenários possuem segmentos totalmente iguais. Dessa forma, quanto mais próximo do valor H $= 2,58$, mais diversificado é o cenário, e quanto mais próximo de H $= 0,00$, menor é a diversidade. As figuras 17, 18 e 19 apresentam a distribuição da entropia por cada um dos agentes. Observa-se que o agente $\mathbf{A_S}$ tende a gerar cenários com valores de entropia mais distribuídos entre $H = [0; 2,58]$, o que indica uma menor diversidade nos cenários gerados por esse agente.

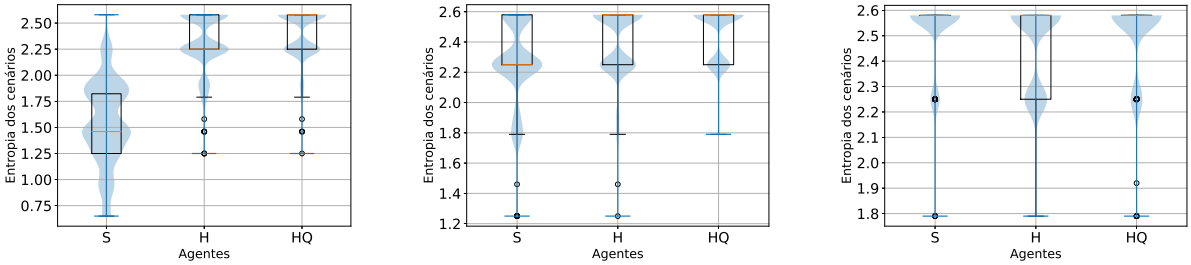
A partir dos dados obtidos, observou-se que os agentes que utilizam a entropia como recompensa (vide tabela 9 e figuras 17, 18 e 19), apresentam uma maior quantidade de ambientes e cenários cujo valor de entropia sejam iguais ou superiores a H $= 2,25$. Isso indica que os agentes conseguiram gerar cenários com uma maior diversidade de segmentos. Em relação ao valor mínimo, os agentes que utilizam a entropia como recompensa apresentam cenários com valores maiores que o agente $\mathbf{A_S}$, com exceção do ambiente Zelda,

que gerou resultados semelhantes para todos os agentes. Os resultados obtidos para o ambiente Zelda(vide 7, 8, 9 e figuras 17, 18 e 19) foram semelhantes devido à sua maior quantidade de segmentos em comparação aos demais ambientes.

Tabela 9 – A tabela apresenta entre parênteses a porcentagem relativa a entropia mínima e máxima dos cenários.

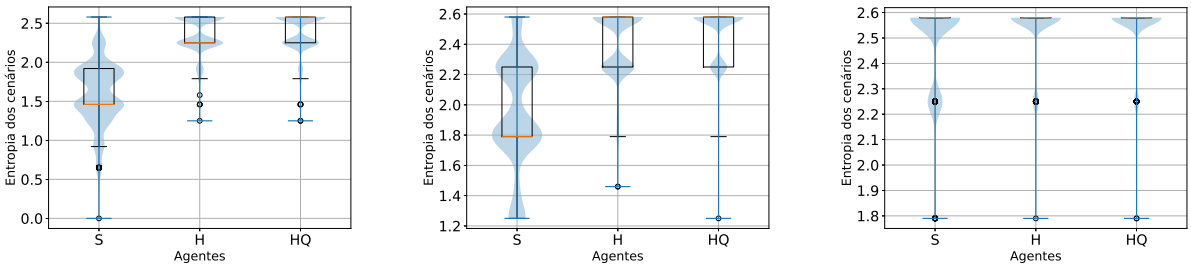
| **Parâmetro** | **Agentes** | **Dungeon** | | **MazeCoin** | | **Zelda** | |
|---|---|---|---|---|---|---|---|
| | | *Mínima* | *Máxima* | *Mínima* | *Máxima* | *Mínima* | *Máxima* |
| **C = 21** | $\mathbf{A_S}$ | 0,65(5,20) | 2,58(0,30) | **1,25(1,38)** | **2,58(31,20)** | 1,79(0,76) | 2,58(83,84) |
| | $\mathbf{A_H}$ | **1,25(0,30)** | **2,58(41,90)** | **1,25(0,11)** | **2,58(54,21)** | 1,79(0,74) | 2,58(68,52) |
| | $\mathbf{A_{HQ}}$ | **1,25(0,10)** | **2,58(52,55)** | **1,79(1,38)** | **2,58(66,78)** | 1,79(1,99) | 2,58(79,14) |
| | | | | | | | |
| **C = 61** | $\mathbf{A_S}$ | 0,00(0,10) | 2,58(0,50) | 1,25(12,80) | 2,58(8,50) | **1,79(2,07)** | **2,58(75,60)** |
| | $\mathbf{A_H}$ | **1,25(0,10)** | **2,58(46,80)** | **1,46(0,20)** | **2,58(57,00)** | **1,79(0,19)** | **2,58(88,85)** |
| | $\mathbf{A_{HQ}}$ | **1,25(0,20)** | **2,58(53,00)** | **1,25(0,10)** | **2,58(73,17)** | **1,79(0,33)** | **2,58(90,20)** |
| | | | | | | | |
| **C = 180** | $\mathbf{A_S}$ | 0,00(34,60) | 1,92(0,40) | 1,25(6,90) | 2,58(19,90) | **1,79(0,55)** | **2,58(87,50)** |
| | $\mathbf{A_H}$ | **0,65(0,10)** | **2,58(45,60)** | **1,25(0,30)** | **2,58(56,70)** | **2,25(6,53)** | **2,58(93,47)** |
| | $\mathbf{A_{HQ}}$ | **1,25(0,30)** | **2,58(58,70)** | **1,25(0,10)** | **2,58(68,80)** | **1,79(0,92)** | **2,58(81,99)** |

Figura 17 – Gráfico Boxplot ilustrando a distribuição da entropia para cada um dos agentes em cada ambiente com 6 segmentos e parâmetro C = 21.




(a) Dungeon (b) MazeCoin (c) Zelda

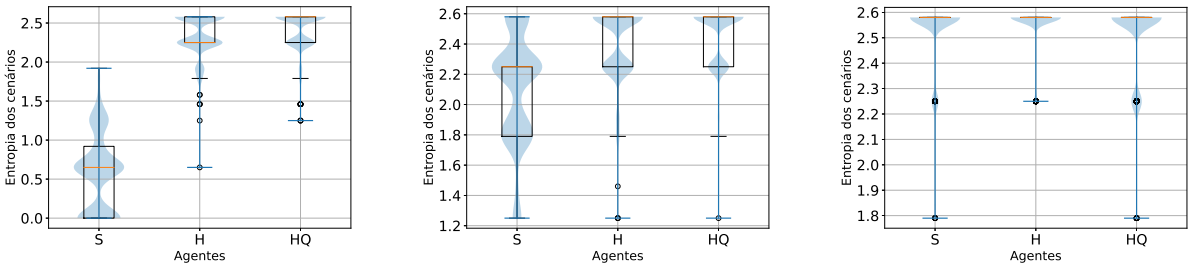
Figura 18 – Gráfico Boxplot ilustrando a distribuição da entropia para cada um dos agentes em cada ambiente com 6 segmentos e parâmetro C = 61.




(a) Dungeon (b) MazeCoin (c) Zelda

A última parte da análise dos resultados obtidos se refere à porcentagem de cenários gerados repetidos por cada agente (vide tabela 10), bem como à quantidade de alterações

Figura 19 – Gráfico Boxplot ilustrando a distribuição da entropia para cada um dos agentes em cada ambiente com 6 segmentos e parâmetro C = 180.




(a) Dungeon (b) MazeCoin (c) Zelda

realizadas por cada agente nos ambientes até que o cenário seja considerado concluído (vide tabela 11). É importante ressaltar que apenas o agente $\mathbf{A_S}$ gerou cenários repetidos nos ambientes MazeCoin e Dungeon. Os resultados sugerem que, quanto mais simples o ambiente, maior é a probabilidade do agente gerar uma quantidade maior de cenários repetidos. Em relação à quantidade de alterações que os agentes realizaram nos ambientes até que o cenário fosse concluído, verificou-se que o agente $\mathbf{A_S}$ tende a gerar cenários com menor quantidade de alterações nos ambientes mais simples (Dungeon e MazeCoin). No caso do ambiente Zelda, os agentes apresentaram resultados semelhantes.

O agente $\mathbf{A_S}$ não possui um bom desempenho para gerar cenários mais diversificados. Nos suspeitamos do fato do agente não aplicar uma recompensa positiva adequada de acordo com a diversidade dos segmentos selecionados. Aplicando o mesmo valor de recompensa positiva para cenários com baixa e alta entropia, desta forma, o agente não é incentivado a selecionar as melhores combinações de segmentos.

Outro ponto a ser considerado é que os cenários modelados por um especialista humano em design de níveis podem influenciar nos resultados. Portanto, não podemos garantir que agentes que não utilizam a entropia como métrica de avaliação gerarão bons cenários para cada treinamento e inferência dos agentes, como ocorre com o agente $\mathbf{A_S}$ no ambiente Zelda.

Além disso, observamos em nossos experimentos que o agente $\mathbf{A_S}$ precisa de mais segmentos do que os demais agentes que utilizam a entropia como recompensa para ser capaz de gerar um número maior de cenários com diversidade. Acreditamos que o agente $\mathbf{A_S}$ teve um desempenho melhor no ambiente Zelda porque este ambiente apresenta uma quantidade maior de objetos no cenário (inimigos, itens, etc.) e um número maior de segmentos em relação aos ambientes Dungeon e MazeCoin. Além disso, a quantidade de segmentos também é um fator importante, pois quanto maior a quantidade de segmentos, maior é a possibilidade de combinações para criar os cenários. Portanto, suspeitamos que o desempenho do agente $\mathbf{A_S}$ seja melhor em ambientes que possuam uma quantidade maior de segmentos.

Tabela 10 – Porcentagem de cenários gerados repetidos por cada agente em cada ambiente.

| **Parâmetro** | **Agentes** | **Dungeon** | **MazeCoin** | **Zelda** |
|---|---|---|---|---|
| **C = 21** | $\mathbf{A_S}$ | 10,10 | 0,11 | **0,00** |
| | $\mathbf{A_H}$ | **0,00** | **0,00** | **0,00** |
| | $\mathbf{A_{HQ}}$ | **0,00** | **0,00** | **0,00** |
| | | | | |
| **C = 61** | $\mathbf{A_S}$ | 7,50 | 3,00 | **0,00** |
| | $\mathbf{A_H}$ | **0,00** | **0,00** | **0,00** |
| | $\mathbf{A_{HQ}}$ | **0,00** | **0,00** | **0,00** |
| | | | | |
| **C = 180** | $\mathbf{A_S}$ | 7,80 | 1,60 | **0,00** |
| | $\mathbf{A_H}$ | **0,00** | **0,00** | **0,00** |
| | $\mathbf{A_{HQ}}$ | **0,00** | **0,00** | **0,00** |

Tabela 11 – A tabela exibe os resultados obtidos por cada agente nos ambientes Dungeon, MazeCoin e Zelda em relação a média (ME), mediana (MD) e desvio padrão (DP) da quantidade de alterações.

| **Parâmetro** | **Agentes** | **Dungeon** | | | **MazeCoin** | | | **Zelda** | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | *ME* | *MD* | *DP* | *ME* | *MD* | *DP* | *ME* | *MD* | *DP* |
| **C = 21** | $\mathbf{A_S}$ | 6,00 | 6,00 | 0,10 | 9,48 | 8,00 | 3,94 | 12,82 | 12,00 | 4,56 |
| | $\mathbf{A_H}$ | 6,29 | 6,00 | 0,96 | 9,56 | 8,00 | 3,83 | 12,56 | 12,00 | 4,46 |
| | $\mathbf{A_{HQ}}$ | 9,53 | 8,00 | 4,14 | 10,69 | 10,00 | 4,20 | 13,01 | 13,00 | 4,62 |
| | | | | | | | | | | |
| **C = 61** | $\mathbf{A_S}$ | 6,02 | 6,00 | 0,29 | 9,75 | 8,00 | 4,59 | 29,25 | 28,00 | 16,02 |
| | $\mathbf{A_H}$ | 6,29 | 6,00 | 0,95 | 11,27 | 9,00 | 6,38 | 31,01 | 30,00 | 16,35 |
| | $\mathbf{A_{HQ}}$ | 11,67 | 9,00 | 6,98 | 14,75 | 12,00 | 9,48 | 31,16 | 30,00 | 15,52 |
| | | | | | | | | | | |
| **C = 180** | $\mathbf{A_S}$ | 6,00 | 6,00 | 0,10 | 10,25 | 8,00 | 5,36 | 62,35 | 49,00 | 44,84 |
| | $\mathbf{A_H}$ | 6,33 | 6,00 | 1,10 | 11,43 | 9,00 | 6,67 | 56,71 | 46,00 | 42,51 |
| | $\mathbf{A_{HQ}}$ | 10,97 | 9,00 | 6,73 | 13,89 | 11,00 | 9,01 | 42,92 | 32,00 | 35,16 |

## 5.2 AMBIENTES COM 8 SEGMENTOS

Igualmente a seção 5.1, utilizaremos dos mesmos pontos de análise dos resultados obtidos para cenários gerados com 6 segmentos. Como critério para classificar os resultados dos ambientes com 8 segmentos como bons resultados, estamos levando em consideração os agentes que conseguiram gerar pelo menos 80% dos cenários com valores de entropia $H \geq 2,75$ e utilizar uma porcentagem de segmentos maior ou igual a 50% e cuja média e mediana da entropia seja $H \geq 2,75$. As imagens referentes aos cenários gerados por cada agente utilizando o valor do parâmetro $C = 61$ podem ser vistas no apêndice B.

Diferentemente dos ambientes gerados com 6 segmentos, os dados apresentados na tabela 12 são respectivamente a quantidade de cenários gerados e porcentagem de segmentos utilizados, porcentagem de cenários com entropia abaixo de $H < 2,75$ e maior ou igual a $H \geq 2,75$. A tabela 13 apresenta informações sobre a media, mediana desvio padrão da entropia dos cenários gerados. Os valores em destaque apresentam os melhores resultados para cada agente. Observa-se que, ao aumentar a quantidade de segmentos para construir os cenários o agente $\mathbf{A_S}$ tende a gerar resultados poucos satisfatório em relação aos agentes que utilizam a entropia como recompensa base. De forma particular, o agente $\mathbf{A_H}$ não atingiu todos os critérios mínimos de bons resultados em relação a média da entropia. Entretanto, os valores de mediana e porcentagem de segmentos utilizados atingiu o objetivo de critérios mínimos, sendo melhores que o agente $\mathbf{A_S}$. Nesta análise, observa-se o quanto as funções de recompensa aplicadas de forma adequada influenciam nos resultados finais. Principalmente quando aplicamos uma métrica de avaliação para os cenários.

Um segundo ponto é em a relação ao ambiente Zelda: como aumentamos a dimensão dos cenários, os agentes apresentaram uma dificuldade maior em gerar uma quantidade significativa de cenários, em nossos experimentos, abaixo dos 50% para os parâmetros $C = 21$ e $C = 61$. Igualmente ao ambiente com 6 segmentos, o ambiente Zelda precisa de mais modelos ou um tempo maior de treinamento para gerar uma quantidade maior de cenários.

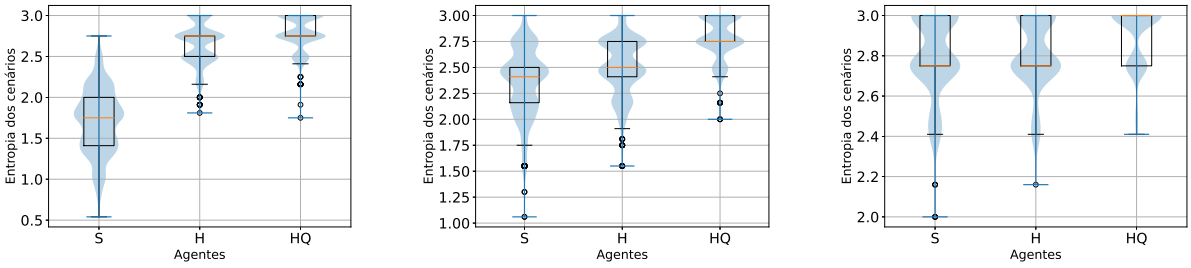
Um terceiro ponto é relação aos valores de entropia miníma e máxima. Igualmente ao ambiente com 6 segmentos, observou-se que agentes que utilizam como recompensa a entropia, vide tabela 14 e figuras 20, 21 e 22, possuem uma quantidade maior de ambientes e cenários cujo valor de entropia é maior ou igual a $H \geq 2,75$, indicando que os agentes geraram cenários com uma boa quantidade de diversidade de segmentos. Em relação ao valor mínimo, os agentes que utilizam a entropia como recompensa possuem cenários com valores maiores ou uma porcentagem menor de cenários com entropia baixa que o agente $\mathbf{A_S}$. Com exceção do ambiente Zelda que possui resultados similares para cada um dos agentes.

A última parte da análise dos resultados obtidos é em relação a porcentagem de cenários gerados repetidos por cada agente(vide tabela 15), bem como à quantidade de alterações realizadas por cada agente nos ambientes até que o cenário seja considerado concluído (vide tabela 16). Similar ao cenários com 6 segmentos, somente o agente $\mathbf{A_S}$ gerou cenários repetidos para os ambientes Dungeon e MazeCoin. Em relação à quantidade de alterações que os agentes realizaram nos ambientes até que o cenário fosse concluído, verificou-se que o agente $\mathbf{A_S}$ tende a gerar cenários com menor quantidade de alterações nos ambientes mais simples (Dungeon e MazeCoin). No caso do ambiente Zelda, os agentes apresentaram resultados semelhantes. De forma particular, ao observar o desempenho do agente $\mathbf{A_{HQ}}$, os dados sugerem uma recompensa que defino um valor de penalidade para cenários abaixo do mínimo do ideal tende a gerar cenários com uma diversidade maior em ambientes com 8 segmentos. Com base nos dados obtidos para os ambientes com 8 segmentos podemos concluir que, a medida que aumentamos a dimensão dos cenários é necessário um tempo maior treinamento para que os agentes consigam obter resultados mais satisfatórios.

Tabela 12 – Resultados obtidos por cada agente nos ambientes Dungeon, MazeCoin e Zelda. A tabela exibe a quantidade de cenários (L), a porcentagem de segmentos utilizados (S), porcentagem de sucesso de cenários gerados com entropia menor e maior ou igual a $H = 2,75$.

| **Parâmetro** | **Agentes** | **Dungeon** | | | | **MazeCoin** | | | | **Zelda** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $L$ | $S$ | $H < 2,75$ | $H \geq 2,75$ | $L$ | $S$ | $H < 2,75$ | $H \geq 2,75$ | $L$ | $S$ | $H < 2,75$ | $H \geq 2,75$ |
| **C = 21** | $\mathbf{A_S}$ | 1000 | 92,50 | 99,50 | 0,50 | 834 | 78,75 | 77,94 | 22,06 | 264 | 47,33 | 16,67 | 83,33 |
| | $\mathbf{A_H}$ | 999 | 95,00 | 43,54 | 56,46 | 808 | 78,75 | 61,88 | 38,12 | **143** | **56,67** | **11,89** | **88,11** |
| | $\mathbf{A_{HQ}}$ | **761** | **97,50** | **19,84** | **80,16** | **416** | **416** | **19,23** | **80,77** | 203 | 42,00 | 4,43 | 95,57 |
| | | | | | | | | | | | | | |
| **C = 61** | $\mathbf{A_S}$ | 1000 | 80,00 | 99,60 | 0,40 | 999 | 82,50 | 78,58 | 21,42 | **375** | **78,00** | **5,87** | **94,13** |
| | $\mathbf{A_H}$ | 1000 | 97,50 | 41,70 | 58,30 | 999 | 81,25 | 41,14 | 58,86 | **292** | **73,00** | **10,62** | **89,38** |
| | $\mathbf{A_{HQ}}$ | 974 | 100,00 | 20,33 | 79,67 | **941** | **81,25** | **17,43** | **82,57** | **394** | **74,00** | **0,76** | **99,24** |
| | | | | | | | | | | | | | |
| **C = 180** | $\mathbf{A_S}$ | 1000 | 67,50 | 100,00 | 0,00 | 1000 | 66,25 | 97,20 | 2,80 | **846** | **78,33** | **7,09** | **92,91** |
| | $\mathbf{A_H}$ | 1000 | 92,50 | 39,10 | 60,90 | 1000 | 77,50 | 44,50 | 44,50 | **807** | **77,33** | **6,44** | **93,56** |
| | $\mathbf{A_{HQ}}$ | 1000 | 97,50 | 20,90 | 79,10 | **999** | **81,25** | **15,22** | **84,78** | **890** | **70,00** | **1,35** | **98,65** |

Figura 20 – Gráfico Boxplot ilustrando a distribuição da entropia para cada um dos agentes em cada ambiente com 8 segmentos e parâmetro C = 21.




(a) Dungeon (b) MazeCoin (c) Zelda

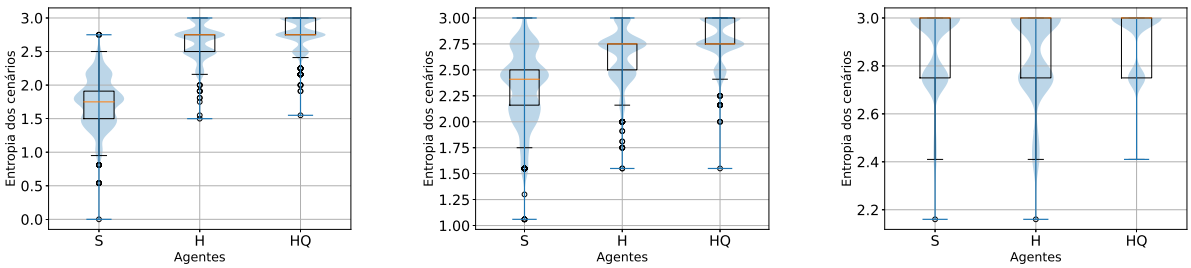
Tabela 13 – Resultados obtidos por cada agente nos ambientes Dungeon, MazeCoin e Zelda. A tabela exibe a média (ME), mediana (MD) e desvio padrão (DP) da entropia dos níveis.

| **Parâmetro** | **Agentes** | **Dungeon** | | | **MazeCoin** | | | **Zelda** | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | *ME* | *MD* | *DP* | *ME* | *MD* | *DP* | *ME* | *MD* | *DP* |
| **C = 21** | $\mathbf{A_S}$ | 1,70 | 1,75 | 0,40 | 2,35 | 2,41 | 0,33 | 2,79 | 2,75 | 0,22 |
| | $\mathbf{A_H}$ | 2,64 | 2,75 | 0,25 | 2,50 | 2,50 | 0,29 | **2,81** | **2,75** | **0,18** |
| | $\mathbf{A_{HQ}}$ | **2,78** | **2,75** | **0,21** | **2,78** | **2,75** | **0,21** | 2,91 | 3,00 | 0,15 |
| | | | | | | | | | | |
| **C = 61** | $\mathbf{A_S}$ | 1,74 | 1,75 | 0,37 | 2,32 | 2,41 | 0,36 | **2,88** | **3,00** | **0,16** |
| | $\mathbf{A_H}$ | 2,65 | 2,75 | 0,25 | 2,63 | 2,75 | 0,26 | **2,85** | **3,00** | **0,18** |
| | $\mathbf{A_{HQ}}$ | 2,77 | 2,75 | 0,22 | **2,79** | **2,75** | **0,21** | **2,93** | **3,00** | **0,12** |
| | | | | | | | | | | |
| **C = 180** | $\mathbf{A_S}$ | 1,20 | 1,20 | 0,41 | 1,94 | 2,00 | 0,40 | **2,86** | **3,00** | **0,16** |
| | $\mathbf{A_H}$ | 2,66 | 2,66 | 0,25 | 2,62 | 2,75 | 0,26 | **2,88** | **3,00** | **0,17** |
| | $\mathbf{A_{HQ}}$ | 2,77 | 2,75 | 0,22 | **2,80** | **2,75** | **0,20** | **2,94** | **3,00** | **0,12** |

Tabela 14 – A tabela exibe a porcentagem relativa a entropia mínima e máxima dos cenários.

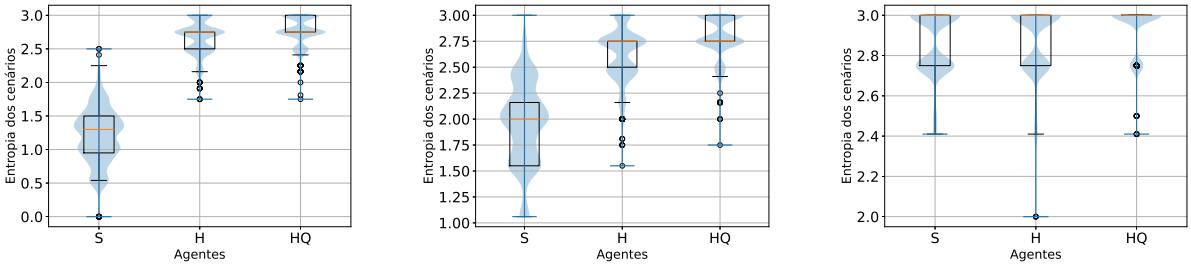
| **Parâmetro** | **Agentes** | **Dungeon** | | **MazeCoin** | | **Zelda** | |
|---|---|---|---|---|---|---|---|
| | | *Mínima* | *Máxima* | *Mínima* | *Máxima* | *Mínima* | *Máxima* |
| **C = 21** | $\mathbf{A_S}$ | 0,54(0,70) | 2,75(0,50) | 1,06(0,24) | 3,00(2,64) | 2,00(1,14) | 3,00(40,53) |
| | $\mathbf{A_H}$ | 1,81(0,10) | 3,00(17,52) | 1,55(0,99) | 3,00(5,32) | **2,16(0,70)** | **3,00(39,16)** |
| | $\mathbf{A_{HQ}}$ | **1,75(0,13)** | **3,00(38,37)** | **2,00(0,72)** | **3,00(38,22)** | 2,41(1,48) | 3,00(67,00) |
| | | | | | | | |
| **C = 61** | $\mathbf{A_S}$ | 0,00(0,10) | 2,75(0,40) | 1,06(0,60) | 3,00(2,30) | **2,16(0,27)** | **3,00(61,33)** |
| | $\mathbf{A_H}$ | 1,50(0,10) | 3,00(17,30) | 1,55(0,20) | 3,00(15,52) | **2,16(0,34)** | **3,00(52,40)** |
| | $\mathbf{A_{HQ}}$ | 1,55(0,10) | 3,00(36,14) | **1,55(0,11)** | **3,00(37,83)** | **2,41(0,51)** | **3,00(73,86)** |
| | | | | | | | |
| **C = 180** | $\mathbf{A_S}$ | 0,00(1,60) | 2,50(0,20) | 1,06(6,10) | 3,00(0,20) | **2,41(3,43)** | **3,00(52,60)** |
| | $\mathbf{A_H}$ | 1,75(0,40) | 3,00(19,90) | 1,55(0,10) | 3,00(14,30) | **2,00(0,37)** | **3,00(59,36)** |
| | $\mathbf{A_{HQ}}$ | 1,75(0,10) | 3,00(34,40) | **1,75(0,10)** | **3,00(41,54)** | **2,41(0,56)** | **3,00(77,53)** |

Figura 21 – Gráfico Boxplot ilustrando a distribuição da entropia para cada um dos agentes em cada ambiente com 8 segmentos e parâmetro C = 61.




(a) Dungeon (b) MazeCoin (c) Zelda

Figura 22 – Gráfico Boxplot ilustrando a distribuição da entropia para cada um dos agentes em cada ambiente com 8 segmentos e parâmetro C = 180.




(a) Dungeon (b) MazeCoin (c) Zelda

Tabela 15 – Porcentagem de cenários gerados repetidos por cada agente em cada ambiente.

| Parâmetro | Agentes | Dungeon | MazeCoin | Zelda |
|---|---|---|---|---|
| **C = 21** | $\mathbf{A_S}$ | 2,00 | **0,00** | **0,00** |
| | $\mathbf{A_H}$ | **0,00** | **0,00** | **0,00** |
| | $\mathbf{A_{HQ}}$ | **0,00** | **0,00** | **0,00** |
| | | | | |
| **C = 61** | $\mathbf{A_S}$ | 1,20 | **0,00** | **0,00** |
| | $\mathbf{A_H}$ | **0,00** | **0,00** | **0,00** |
| | $\mathbf{A_{HQ}}$ | **0,00** | **0,00** | **0,00** |
| | | | | |
| **C = 180** | $\mathbf{A_S}$ | 11,40 | 1,10 | **0,00** |
| | $\mathbf{A_H}$ | **0,00** | **0,00** | **0,00** |
| | $\mathbf{A_{HQ}}$ | **0,00** | **0,00** | **0,00** |

Tabela 16 – A tabela exibe os resultados obtidos por cada agente nos ambientes Dungeon, MazeCoin e Zelda em relação média (ME), mediana (MD) e desvio padrão (DP) da quantidade de alterações.

| Parâmetro | Agentes | Dungeon | | | MazeCoin | | | Zelda | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | *ME* | *MD* | *DP* | *ME* | *MD* | *DP* | *ME* | *MD* | *DP* |
| **C = 21** | $\mathbf{A_S}$ | 8,01 | 8,00 | 0,14 | 11,86 | 11,00 | 3,84 | 13,59 | 14,00 | 4,03 |
| | $\mathbf{A_H}$ | 8,64 | 8,00 | 1,59 | 12,06 | 11,00 | 3,86 | 14,46 | 14,00 | 3,96 |
| | $\mathbf{A_{HQ}}$ | 12,28 | 12,00 | 4,23 | 13,31 | 13,00 | 4,13 | 14,22 | 14,00 | 4,40 |
| | | | | | | | | | | |
| **C = 61** | $\mathbf{A_S}$ | 8,00 | 8,00 | 0,04 | 14,59 | 12,00 | 7,66 | 33,02 | 32,00 | 15,50 |
| | $\mathbf{A_H}$ | 8,39 | 8,00 | 1,32 | 15,18 | 12,00 | 8,13 | 33,04 | 32,00 | 15,37 |
| | $\mathbf{A_{HQ}}$ | 19,38 | 16,00 | 11,93 | 23,03 | 19,00 | 13,55 | 31,30 | 31,00 | 15,76 |
| | | | | | | | | | | |
| **C = 180** | $\mathbf{A_S}$ | 8,01 | 8,00 | 0,17 | 13,68 | 12,00 | 6,55 | 67,79 | 57,00 | 46,17 |
| | $\mathbf{A_H}$ | 8,35 | 8,00 | 1,26 | 14,64 | 12,00 | 7,64 | 70,54 | 60,00 | 46,14 |
| | $\mathbf{A_{HQ}}$ | 19,01 | 14,00 | 13,31 | 24,52 | 19,00 | 17,07 | 63,53 | 55,00 | 44,29 |

## 5.3 LENIÊNCIA E LINEARIDADE

Uma forma inicial de analisar a expressividade dos níveis gerados pelos agentes é por meio da utilização de um grande número de cenários, que são ranqueados de acordo com sua linearidade (vide equação 5.1) e leniência (vide equação 5.2). No entanto, para nossa análise, optamos por considerar a quantidade de níveis gerados por cada agente, assim como os valores atribuídos ao parâmetro C, que controla a quantidade máxima de tentativas que os agentes podem fazer para construir um cenário.

Nesta seção analisamos os resultados obtidos(vide figuras 23 à 28) somente os ambientes que possuem objetivo de *gameplay* (MazeCoin e Zelda). Para essa análise, cada hexágono é colorido para indicar a quantidade de níveis gerados que correspondem aos valores aplicados a linearidade e leniência. Os resultados obtidos foram normalizados em uma escala entre 0 e 1. Hexágonos de cor mais clara indicam que há mais níveis que se relacionam com a linearidade ou leniência.

$$linearidade = \mathrm{Sn} \times (Hmax - H) \tag{5.1}$$

$$leniencia = \frac{\sum_{i=1}^{n} Di}{MAXDIST} \times (\sum_{i=1}^{m} Ei) \times H \tag{5.2}$$

Para uma melhor compreensão dos dados, cenários gerados com um alto valor de leniência indicam níveis de dificuldade mais elevados. O valor da leniência varia de 0 a 1, onde um valor de 1 indica um nível de dificuldade mais elevado e um valor de 0 indica um nível de dificuldade mais baixo. No ambiente MazeCoin, um valor de leniência mais alto indica uma maior quantidade de moedas a serem coletadas e uma maior distância entre o personagem e as moedas. Para o ambiente Zelda, um valor de leniência mais alto indica uma quantidade maior de inimigos, mais moedas a serem coletadas e uma distância maior entre o personagem e a saída do nível, bem como, os demais elementos do jogo (espada, chave e moedas).

A linearidade é medida em relação à quantidade de segmentos utilizados em cada um dos ambientes. Um valor de linearidade mais alto indica níveis gerados com uma maior quantidade de segmentos repetidos, portanto, um valor de entropia mais baixo dos cenários.

Com base nos dados apresentados, pode-se inferir para o agente $\mathbf{A_S}$, que não utiliza a entropia como feedback de recompensa, tende a gerar cenários mais lineares e com baixa leniência no ambiente MazeCoin devido à repetição de segmentos nos níveis gerados. Podemos observar a quantidade de hexágonos de cor mais clara nos gráficos 23 a 25. Além disso, é importante notar que quanto maior for a quantidade de segmentos, maior será a linearidade do nível em relação ao ambiente MazeCoin. Desta forma, quando aumentamos

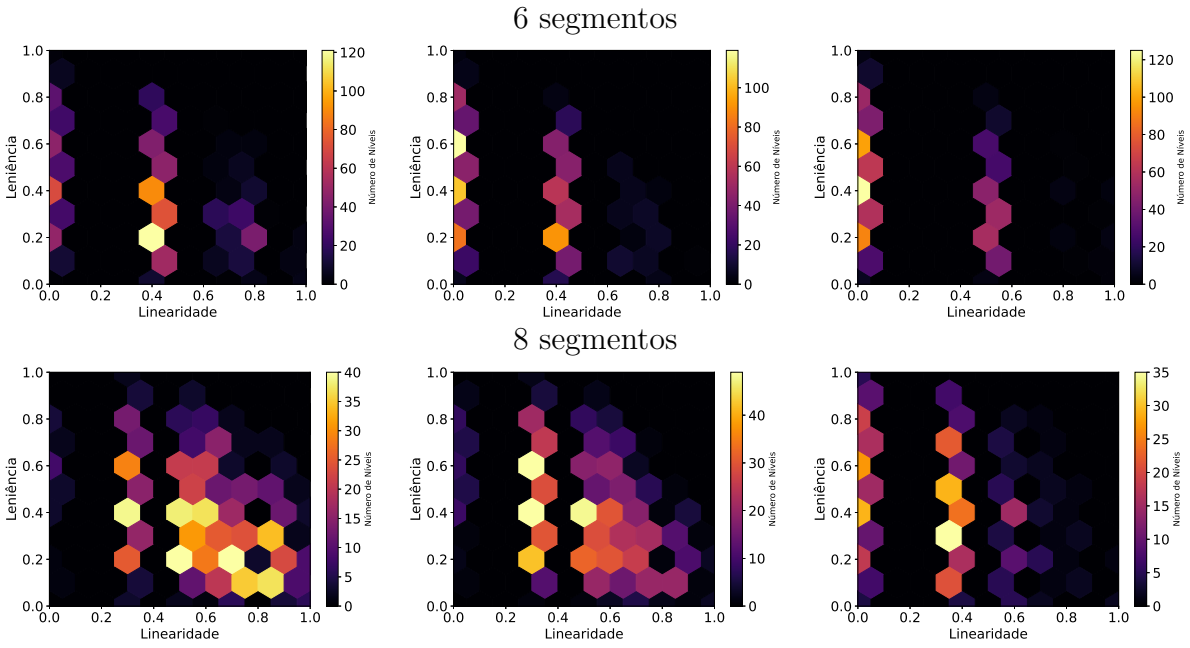
o valor do parâmetro C para 180 (vide figura 25) há uma maior quantidade de níveis lineares e com baixa leniência. Portanto, a medida que aumentamos o valor do parâmetro C há um agrupamento menor de níveis para a linearidade mais baixa. Isso indica que foram gerados níveis com uma grande quantidade de segmentos repetidos, implicando no valor da leniência.

Com relação aos níveis gerados para o ambiente Zelda (vide figuras 26, 27 e 28) possui uma grande similaridade entre os valores de linearidade e leniência para cada valor do parâmetro C. Suspeitamos que isso ocorra devido ao fato do ambiente Zelda possuir um número maior de segmentos para gerar os cenários. Desta forma, há uma possibilidade maior de combinações para gerar os cenários em relação aos demais ambientes (vide tabela 17). As figuras 29 e 30 demonstram diferentes cenários gerados com diversos valores de linearidade e leniência.

Tabela 17 – Quantidade de segmentos por cada ambiente.

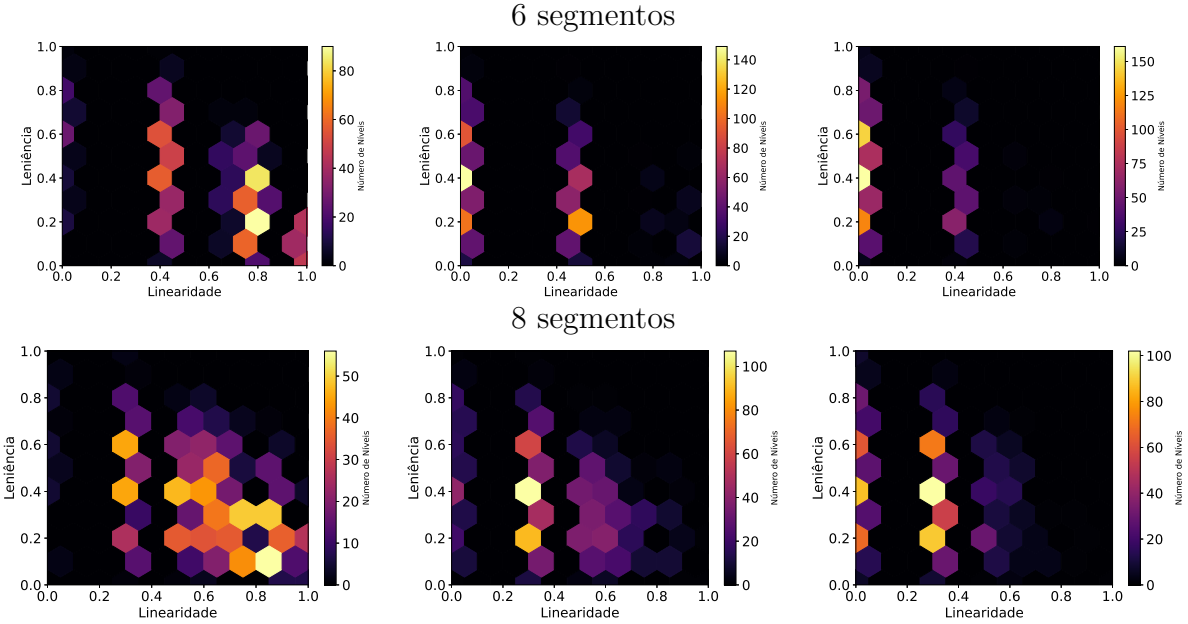
| **Ambientes** | **Segmentos** |
|---|---|
| **Dungeon** | 40 |
| **MazeCoin** | 80 |
| **Zelda** | 300 |

Figura 23 – Gráficos expressive range para as métricas linearidade e leniência para o ambiente MazeCoin com 6 e 8 segmentos utilizando o parâmetro C = 21.



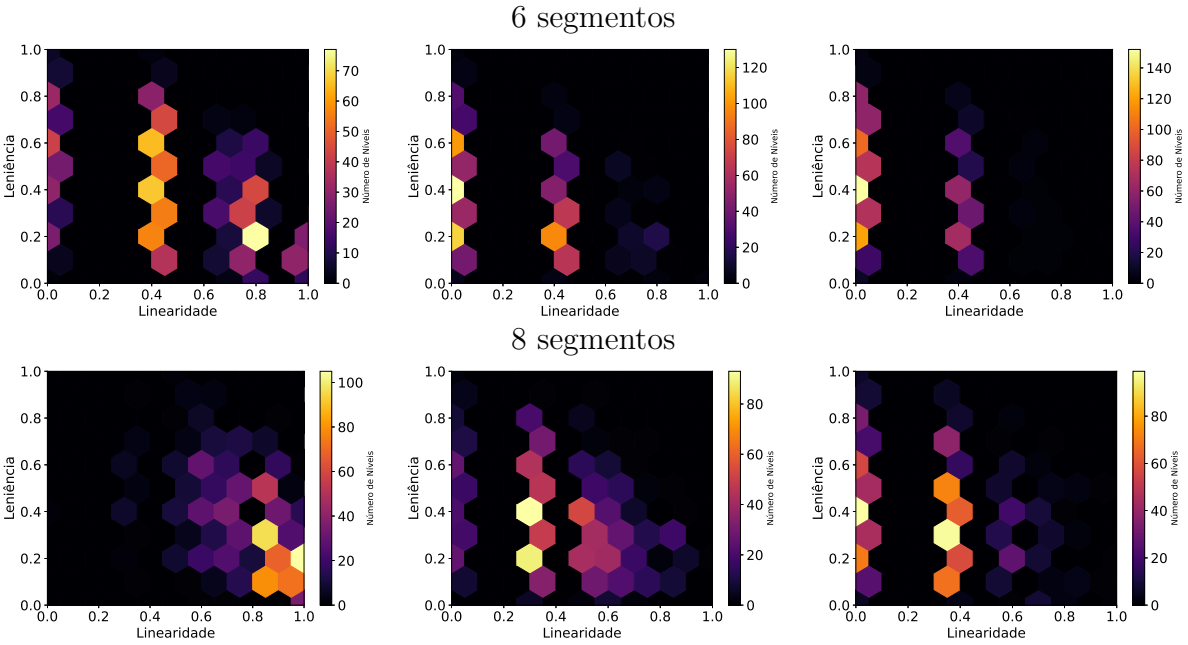

(a) Agente $\mathbf{A_S}$ (b) Agente $\mathbf{A_H}$ (c) Agente $\mathbf{A_{HQ}}$

Figura 24 – Gráficos expressive range para as métricas linearidade e leniência para o ambiente MazeCoin com 6 e 8 segmentos utilizando o parâmetro C = 61.



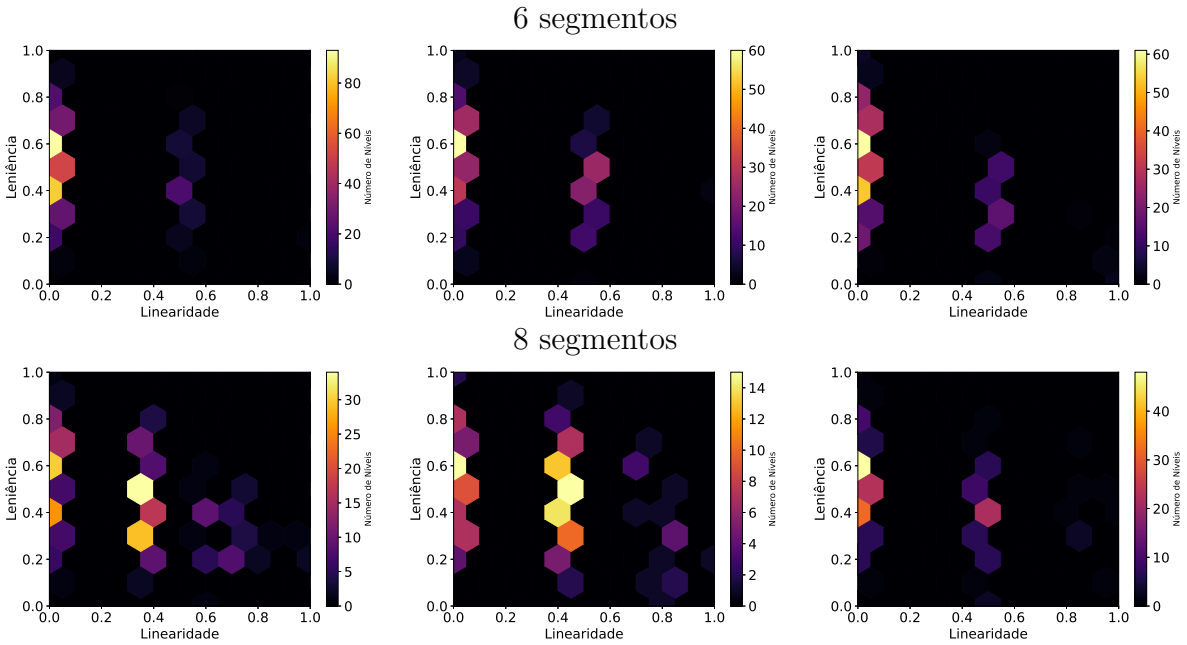

(a) Agente $\mathbf{A_S}$ (b) Agente $\mathbf{A_H}$ (c) Agente $\mathbf{A_{HQ}}$

Figura 25 – Gráficos expressive range para as métricas linearidade e leniência para o ambiente MazeCoin com 6 e 8 segmentos utilizando o parâmetro C = 180.



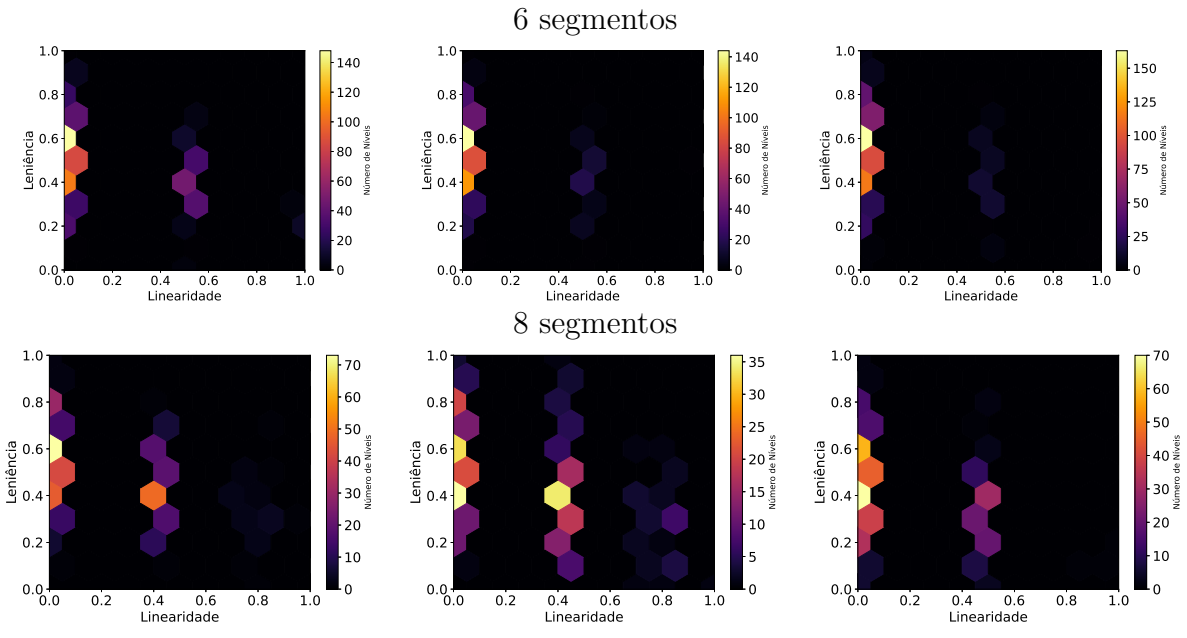

(a) Agente $\mathbf{A_S}$ (b) Agente $\mathbf{A_H}$ (c) Agente $\mathbf{A_{HQ}}$

Figura 26 – Gráficos expressive range para as métricas linearidade e leniência para o ambiente Zelda com 6 e 8 segmentos utilizando o parâmetro C = 21.



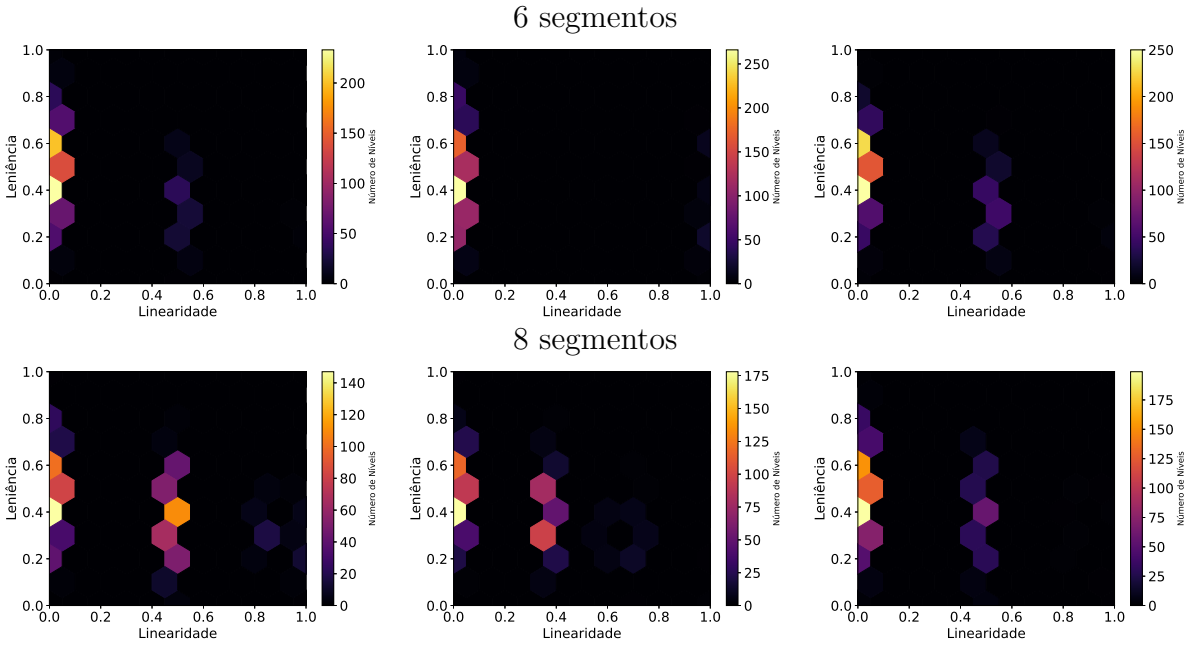

(a) Agente $\mathbf{A_S}$ (b) Agente $\mathbf{A_H}$ (c) Agente $\mathbf{A_{HQ}}$

Figura 27 – Gráficos expressive range para as métricas linearidade e leniência para o ambiente Zelda com 6 e 8 segmentos utilizando o parâmetro C = 61.



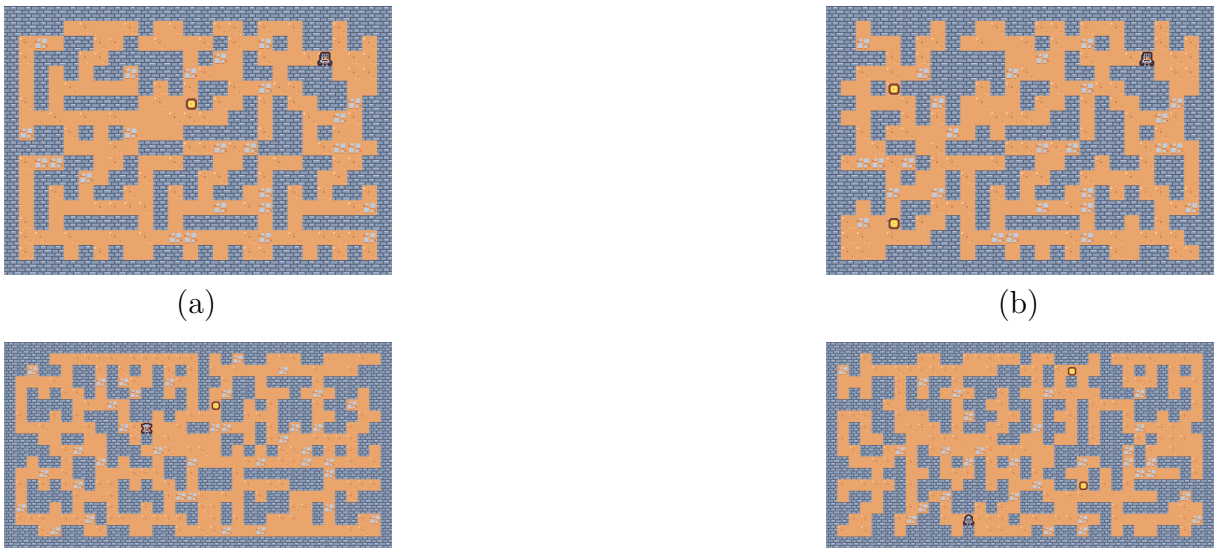

(a) Agente $\mathbf{A_S}$ (b) Agente $\mathbf{A_H}$ (c) Agente $\mathbf{A_{HQ}}$

Figura 28 – Gráficos expressive range para as métricas linearidade e leniência para o ambiente Zelda com 6 e 8 segmentos utilizando o parâmetro C = 180.




(a) Agente $\mathbf{A_S}$ (b) Agente $\mathbf{A_H}$ (c) Agente $\mathbf{A_{HQ}}$

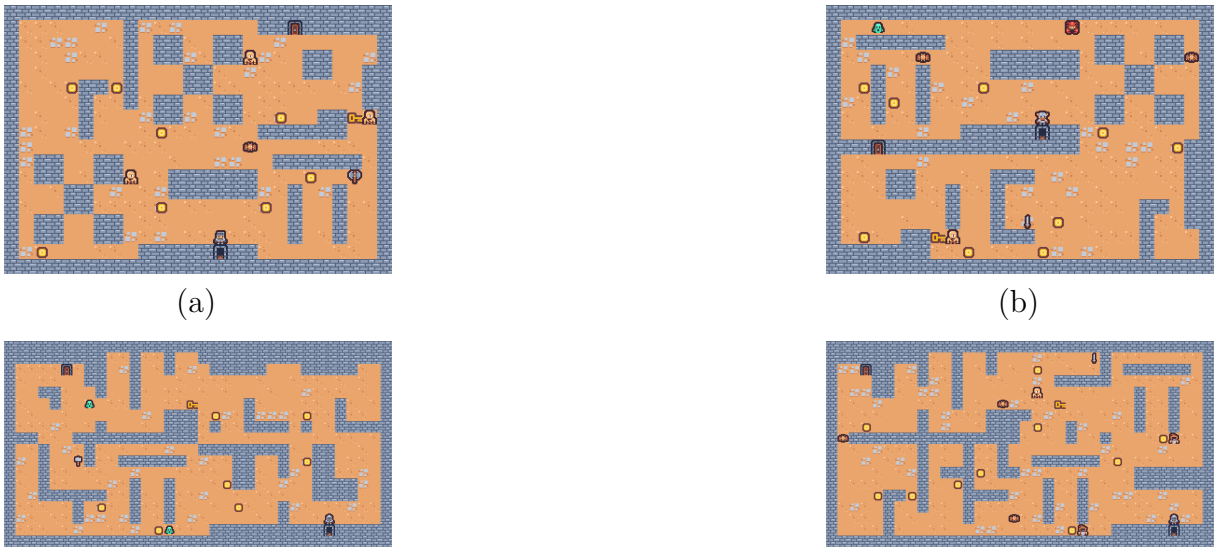
Figura 29 – Exemplos de cenários para o ambiente MazeCoin com 6 e 8 segmentos gerados com diferentes valores para linearidade e leniência. Cenário 29a com Linearidade(1,00) e Leniência (0,05), 29b com Linearidade(0,0) e Leniência (0,96), 29c com Linearidade(0,72) e Leniência (0,06) e 29d com Linearidade(0,00) e Leniência (0,46).



(a) (b) (c) (d)

Fonte: Elaborada pelo autor (2022).

Figura 30 – Exemplos de cenários para o ambiente Zelda com 6 e 8 segmentos gerados com diferentes valores para linearidade e leniência. Cenário 30a com Linearidade(0,52) e Leniência (0,29), 30b com Linearidade(0,0) e Leniência (0,95), 30c com Linearidade(0,49) e Leniência (0,19) e 30d com Linearidade(0,49) e Leniência (0,73).



(a) (b)

(c) (d)

Fonte: Elaborada pelo autor (2022).

# 6 CONCLUSÃO E TRABALHOS FUTUROS

A geração procedural de conteúdo tem se tornado um elemento cada vez mais popular nos jogos ao longo dos anos. Neste trabalho, apresentamos uma proposta que combina aprendizado por reforço, geração procedural de conteúdo e *design* de iniciativa mista. Para avaliar os cenários gerados proceduralmente, utilizamos uma métrica baseada no valor da entropia. Apresentamos uma forma efetiva de como gerar cenários diversificados utilizando agentes de RL.

A principal diferença entre este trabalho e trabalhos publicados que combinam GPC e RL é que utilizamos modelos de níveis fornecidos por um especialista humano em *level design* e uma métrica para quantificar a qualidade dos níveis com relação à diversidade dos segmentos. Para avaliar nossa proposta, testamos em três tipos de ambientes 2D no estilo de *Dungeon crawlers* para avaliar como os agentes se comportam ao construir diferentes cenários. Com base nos resultados obtidos, descobrimos que a entropia pode ser utilizada como uma métrica eficaz para medir a diversidade dos cenários criados em jogos, desde que sejam utilizados modelos fornecidos por um especialista humano em *level design*. Demonstramos que a aplicação adequada de valores de recompensa durante o treinamento permite ao agente buscar soluções melhores ao longo do tempo.

É importante destacar que os modelos de cenários implementados pelo especialista humano em *level design* podem ter um impacto significativo nos resultados obtidos em cada ambiente. Um dos principais problemas é a maneira como o especialista modela os cenários, pois se os modelos para cada ambiente forem muito semelhantes, isso pode afetar negativamente os resultados. Além disso, não podemos garantir que a construção de cenários sem o uso de métricas, como a entropia, resultará em cenários com segmentos diversos, como observado no agente $\mathbf{A_S}$ no ambiente Zelda. No entanto, os resultados mostram que a aplicação de recompensas adequadas durante o treinamento permite que os agentes busquem soluções ótimas para o problema, especialmente quando aumentamos a dimensão dos cenários.

O aprendizado por reforço demonstra ser uma técnica eficaz no objetivo do agente aprender por meio de experiências. Entretanto, é crucial que diversos aspectos sejam planejados cuidadosamente para cada tipo de problema. A formulação adequada das funções de recompensa, estados e ações é fundamental, pois se esses elementos não forem modelados adequadamente, podem ocorrer problemas durante o treinamento dos agentes.

Como trabalhos futuros nessa linha de pesquisa, sugerimos cinco possibilidades de investigação. A primeira sugestão é uma análise mais detalhada das métricas de avaliação para medir a qualidade de cenários gerados proceduralmente por algoritmos de RL. Uma abordagem sugerida pode ser a utilização de algoritmos genéticos para avaliar a diversidade e qualidade dos cenários gerados e, a partir disso, gerar uma recompensa apropriada, como

ocorre com a entropia. Um segunda contribuição é a utilização de agentes de aprendizado por reforço para jogar os cenários gerados, visando avaliar a dificuldade dos níveis e gerar novos níveis de acordo com o desempenho dos agentes. Essa abordagem é baseada no trabalho de (JUSTESEN et al., 2018), que introduziu o conceito de *Progressive PCG*. A terceira contribuição é o uso de jogadores humanos para avaliar a qualidade dos níveis gerados pelos agentes. Essa abordagem é inspirada no trabalho de (VIANA et al., 2022), que utilizou jogadores humanos para avaliar a experiência do usuário em relação aos níveis gerados. Essa abordagem permite obter feedback qualitativo que pode ser usado para melhorar a qualidade dos níveis gerados. Como quarto passo, implementar uma métrica para analisar a similaridade entre os níveis modelados pelo especialista humano em *level design*. De forma que, esses modelos não sejam utilizados durante o treinamento dos agentes ou avaliar e comparar os cenários gerados baseado em padrões como no trabalho de (LUCAS; VOLZ, 2019) utilizando *Kullback-Leibler (KL) Divergence*.

Por fim, implementar uma estrutura que permita que os próprios agentes de aprendizado por reforço definam as posições dos objetos que comporão os cenários, como inimigos e itens coletáveis, ou ajustem esses elementos definidos pelo especialista humano em *level design*. Essa abordagem permitirá que os agentes possam criar níveis mais variados e desafiadores e possam se adaptar às preferências dos jogadores.

# REFERÊNCIAS


BRUNTON, S. L.; KUTZ, J. N. *Data Driven Science and Engineeringl*: Machine learning, dynamical systems, and contro. 1. ed. [S.l.]: Cambridge University Press, 2019. ISBN 9781108422093.

FRANçOIS-LAVET, V. et al. *An Introduction to Deep Reinforcement Learning*. [s.n.], 2018. v. 11. 219–354 p. Disponível em: <http://dx.doi.org/10.1561/2200000071>.

GISSLEN, L. et al. Adversarial reinforcement learning for procedural content generation. In: *2021 IEEE Conference on Games (CoG)*. [s.n.], 2021. p. 1–8. Disponível em: <https://doi.org/10.1109/CoG52621.2021.9619053>.

GRAY, R. *Entropy and Information Theory*. [S.l.]: Springer US, 2011. ISBN 9781441979711.

HENDERSON, P. et al. Deep reinforcement learning that matters. *Proceedings of the AAAI Conference on Artificial Intelligence*, v. 32, n. 1, Apr. 2018. Disponível em: <https://ojs.aaai.org/index.php/AAAI/article/view/11694>.

JUSTESEN, N. et al. *Illuminating Generalization in Deep Reinforcement Learning through Procedural Level Generation*. arXiv, 2018. Disponível em: <https://arxiv.org/abs/1806.10729>.

KHALIFA, A. et al. Pcgrl: Procedural content generation via reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, v. 16, n. 1, p. 95–101, Oct. 2020. Disponível em: <https://ojs.aaai.org/index.php/AIIDE/article/view/7416>.

KINGMA, D. P.; BA, J. *Adam: A Method for Stochastic Optimization*. 2017.

KORN, O.; LEE, N. *Game Dynamics: Best Practices in Procedural and Dynamic Game Content Generation*. [S.l.]: Springer International Publishing, 2017. ISBN 9783319530888.

LAZARIDIS, L. et al. Auto generating maps in a 2d environment. In: FANG, X. (Ed.). *HCI in Games*. Cham: Springer International Publishing, 2022. p. 40–50. ISBN 978-3-031-05637-6.

LEI, C. Deep reinforcement learning. In: _____. *Deep Learning and Practice with MindSpore*. Singapore: Springer Singapore, 2021. p. 217–243. ISBN 978-981-16-2233-5. Disponível em: <https://doi.org/10.1007/978-981-16-2233-5_10>.

LIU, J. et al. Deep learning for procedural content generation. *Springer*, Neural Computing and Applications, v. 33, p. 19–37, 2021. Disponível em: <https://doi.org/10.1007/s00521-020-05383-8>.

LUCAS, S. M.; VOLZ, V. Tile pattern kl-divergence for analysing and evolving game levels. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA: Association for Computing Machinery, 2019. (GECCO '19), p. 170–178. ISBN 9781450361118. Disponível em: <https://doi.org/10.1145/3321707.3321781>.

Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. *Nature*, v. 518, n. 5, p. 529–533, 02 2015. ISSN 1476-4687. Disponível em: <https://doi.org/10.1038/nature14236>.

NEWZOO. *Newzoo Global Games Market Report 2022 - Free Version*. 2022. Disponível em: <https://newzoo.com/insights/trend-reports/newzoo-global-games-market-report-2022-free-version>.

RAFFIN, A. et al. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, v. 22, n. 268, p. 1–8, 2021. Disponível em: <http://jmlr.org/papers/v22/20-1364.html>.

RAVICHANDIRAN, S. *Deep Reinforcement Learning with Python: Master classic RL, deep RL, distributional RL, inverse RL, and more with OpenAI Gym and TensorFlow, 2nd Edition*. [S.l.]: Packt Publishing, 2020. ISBN 9781839215599.

RISI, S.; TOGELIUS, J. Increasing generality in machine learning through procedural content generation. *Nature Machine Intelligence*, v. 2, p. 428–436, 08 2020. ISSN 2522-5839. Disponível em: <https://doi.org/10.1038/s42256-020-0208-z>.

SCHULMAN, J. et al. Trust region policy optimization. In: BACH, F.; BLEI, D. (Ed.). *Proceedings of the 32nd International Conference on Machine Learning*. Lille, France: PMLR, 2015. (Proceedings of Machine Learning Research, v. 37), p. 1889–1897. Disponível em: <https://proceedings.mlr.press/v37/schulman15.html>.

SCHULMAN, J. et al. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. Disponível em: <http://arxiv.org/abs/1707.06347>.

SHAKER, J. T. N.; NELSON, M. J. *Procedural Content Generation in Games*. 1. ed. [S.l.]: Springer, 2016. ISBN 9783319427164.

SHANNON, C. A mathematical theory of communication. *Bell System Technical Journal*, v. 27, p. 379–423, 1948.

SHU, T.; LIU, J.; YANNAKAKIS, G. N. Experience-driven pcg via reinforcement learning: A super mario bros study. In: IEEE. *2021 IEEE Conference on Games (CoG)*. [S.l.], 2021. p. accepted.

SMITH, G.; WHITEHEAD, J. Analyzing the expressive range of a level generator. In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. New York, NY, USA: Association for Computing Machinery, 2010. (PCGames '10). ISBN 9781450300230. Disponível em: <https://doi.org/10.1145/1814256.1814260>.

SMITH, G.; WHITEHEAD, J.; MATEAS, M. Tanagra: Reactive planning and constraint solving for mixed-initiative level design. *IEEE Transactions on Computational Intelligence and AI in Games*, v. 3, n. 3, p. 201–215, 2011.

SOUZA, F. R. de; MIRANDA, T. S.; BERNARDINO, H. S. A reward function using image processing for a deep reinforcement learning approach applied to the sonic the hedgehog game. In: XAVIER-JUNIOR, J. C.; RIOS, R. A. (Ed.). *Intelligent Systems*. Cham: Springer International Publishing, 2022. p. 181–195. ISBN 978-3-031-21689-3.

STONE, J. *Information Theory: A Tutorial Introduction*. [S.l.]: Tutorial Introductions, 2015. (Tutorial Introduction Book). ISBN 9780956372857.

SUMMERVILLE, A. et al. Procedural content generation via machine learning (pcgml). *IEEE Transactions on Games*, v. 10, n. 3, p. 257–270, 2018.

SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning, second edition: An Introduction*. 2. ed. [S.l.]: The MIT Press, 2018. ISBN 9780262039246.

VIANA, B. M. et al. Feasible–infeasible two-population genetic algorithm to evolve dungeon levels with dependencies in barrier mechanics. *Applied Soft Computing*, v. 119, p. 108586, 2022. ISSN 1568-4946. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1568494622000989>.

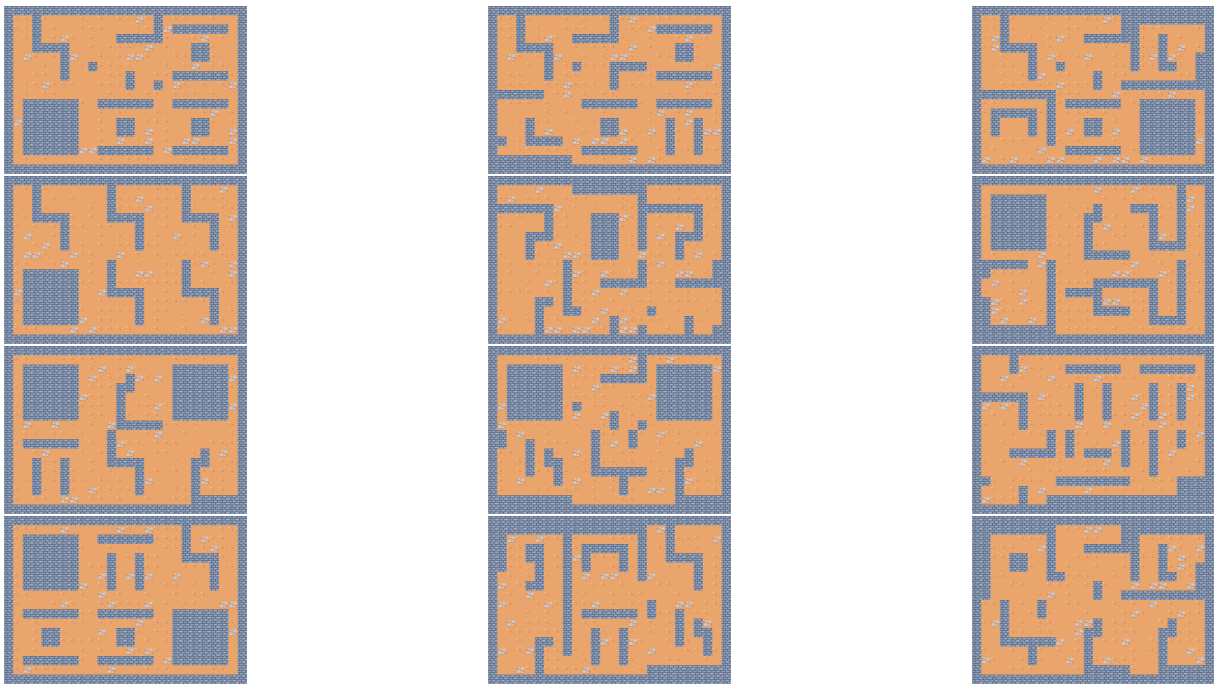
WOLF, M. *Before the Crash: Early Video Game History*. [S.l.]: Wayne State University Press, 2012. (Contemporary approaches to film and media series). ISBN 9780814337226.

YANNAKAKIS, G. N.; TOGELIUS, J. *Artificial Intelligence and Games*. 1. ed. [S.l.]: Springer, 2018. ISBN 978-3-319-63519-4.

ZIDANE, I. M.; IBRAHIM, K. Wavefront and a-star algorithms for mobile robot path planning. In: HASSANIEN, A. E. et al. (Ed.). *Proceedings of the International Conference on Advanced Intelligent Systems and Informatics 2017*. Cham: Springer International Publishing, 2018. p. 69–80. ISBN 978-3-319-64861-3.

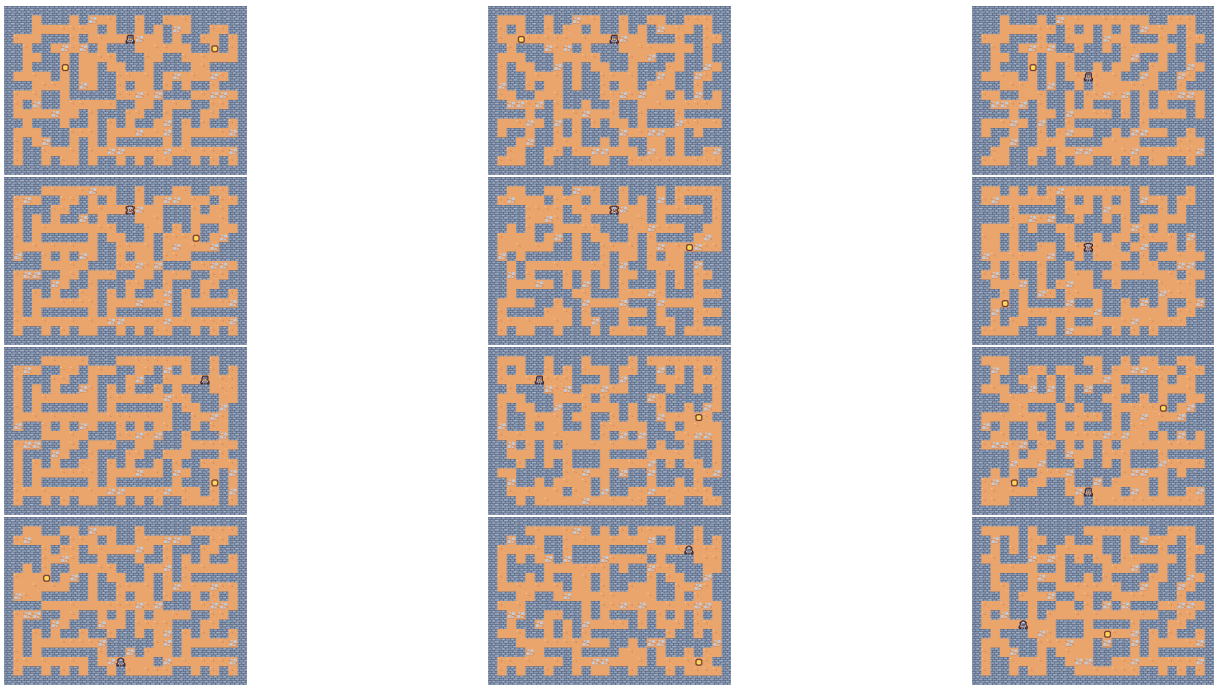
# APÊNDICE A – Cenários gerados com 6 segmentos

Figura 31 – Cenários com 6 segmentos gerados por cada um dos agentes para o ambiente Dungeon.



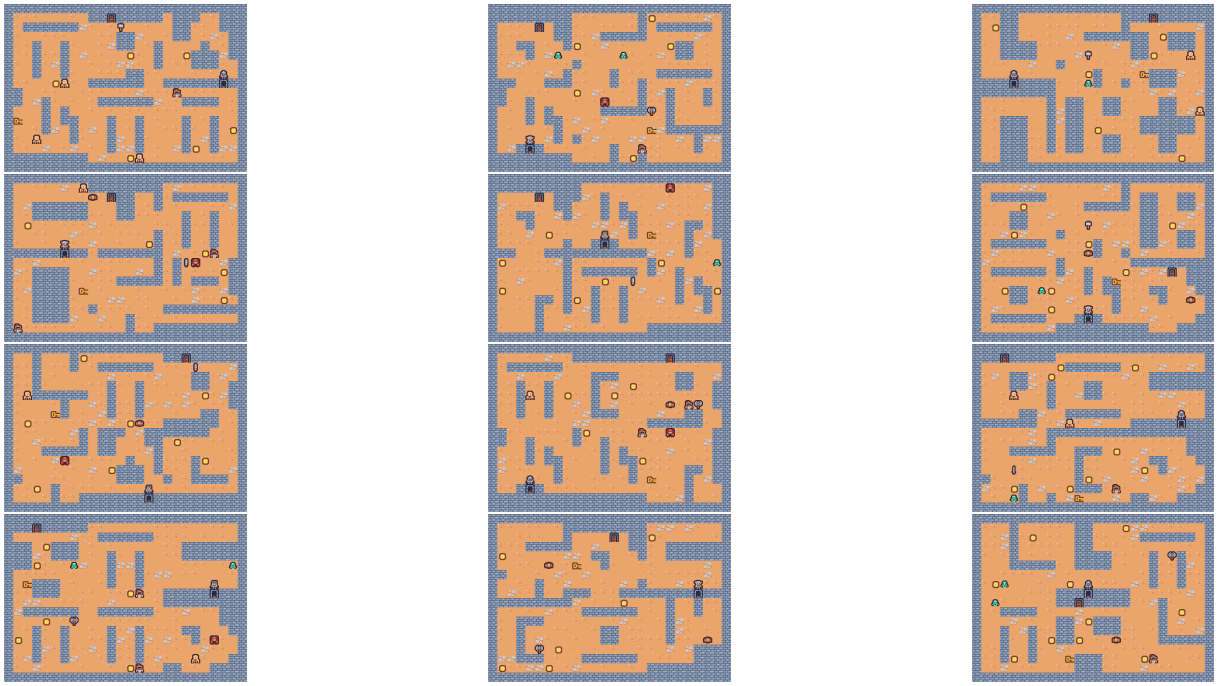
(a) Agente $\mathbf{A}_S$ (b) Agente $\mathbf{A}_H$ (c) Agente $\mathbf{A}_{HQ}$

Figura 32 – Cenários com 6 segmentos gerados por cada um dos agentes para o ambiente MazeCoin.



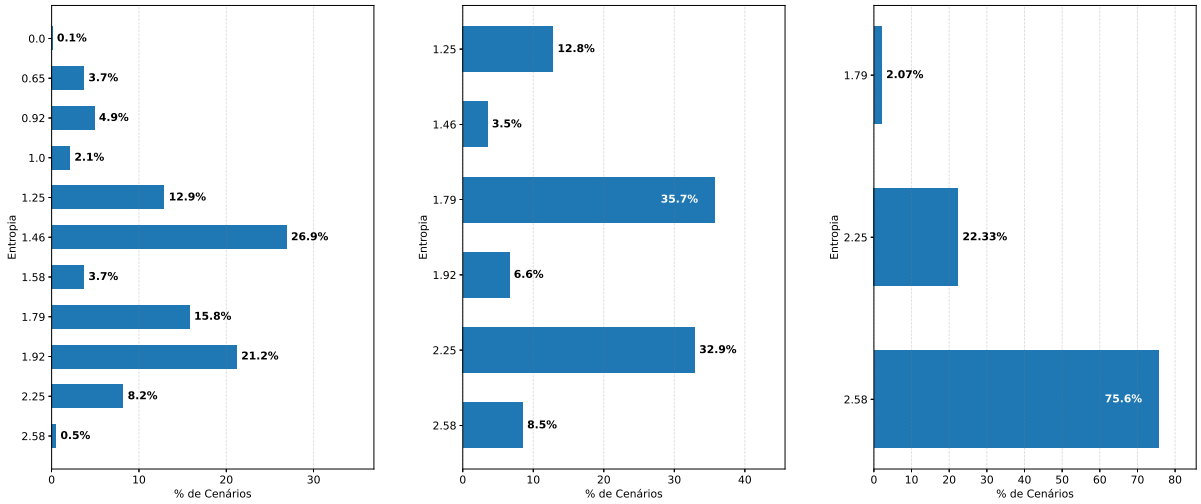
(a) Agente $\mathbf{A}_S$ (b) Agente $\mathbf{A}_H$ (c) Agente $\mathbf{A}_{HQ}$

Figura 33 – Cenários com 6 segmentos gerados por cada um dos agentes para o ambiente Zelda.



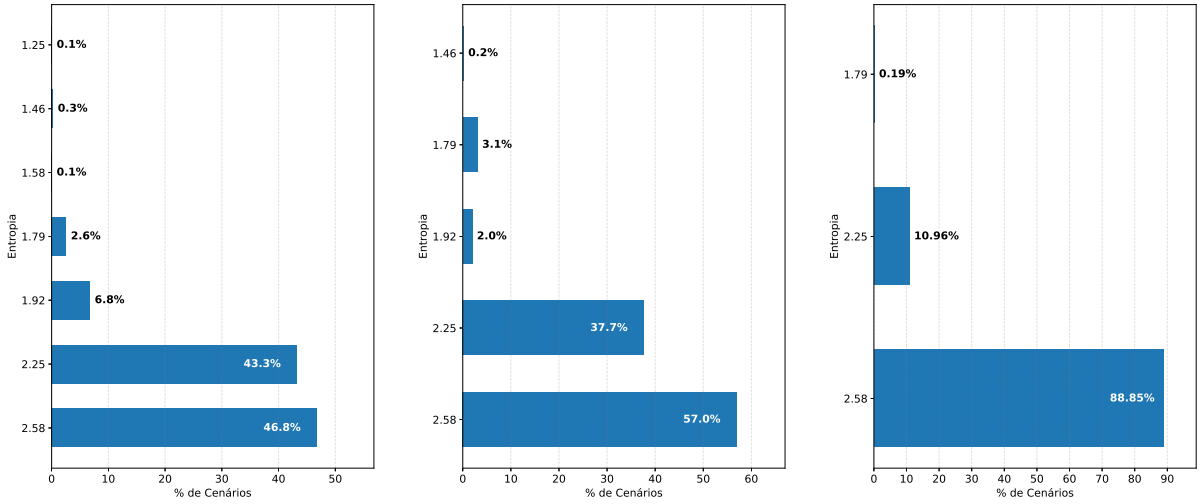
(a) Agente $\mathbf{A}_S$ (b) Agente $\mathbf{A}_H$ (c) Agente $\mathbf{A}_{HQ}$

Figura 34 – Gráfico de distribuição da entropia em cada ambiente utilizando o agente $\mathbf{A}_S$.



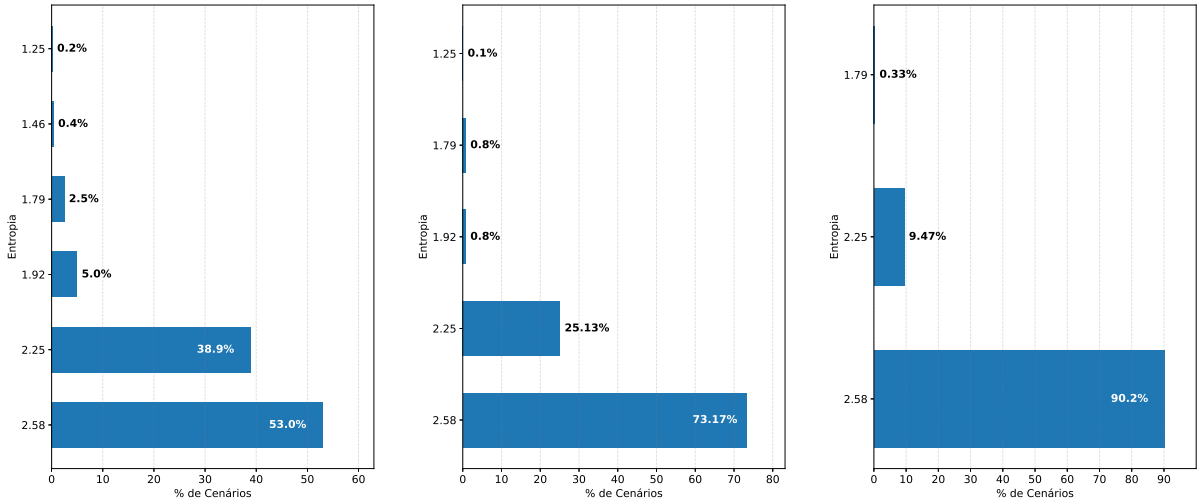

(a) Dungeon (b) MazeCoin (c) Zelda

Figura 35 – Gráfico de distribuição da entropia em cada ambiente utilizando o agente $\mathbf{A}_H$.




(a) Dungeon (b) MazeCoin (c) Zelda

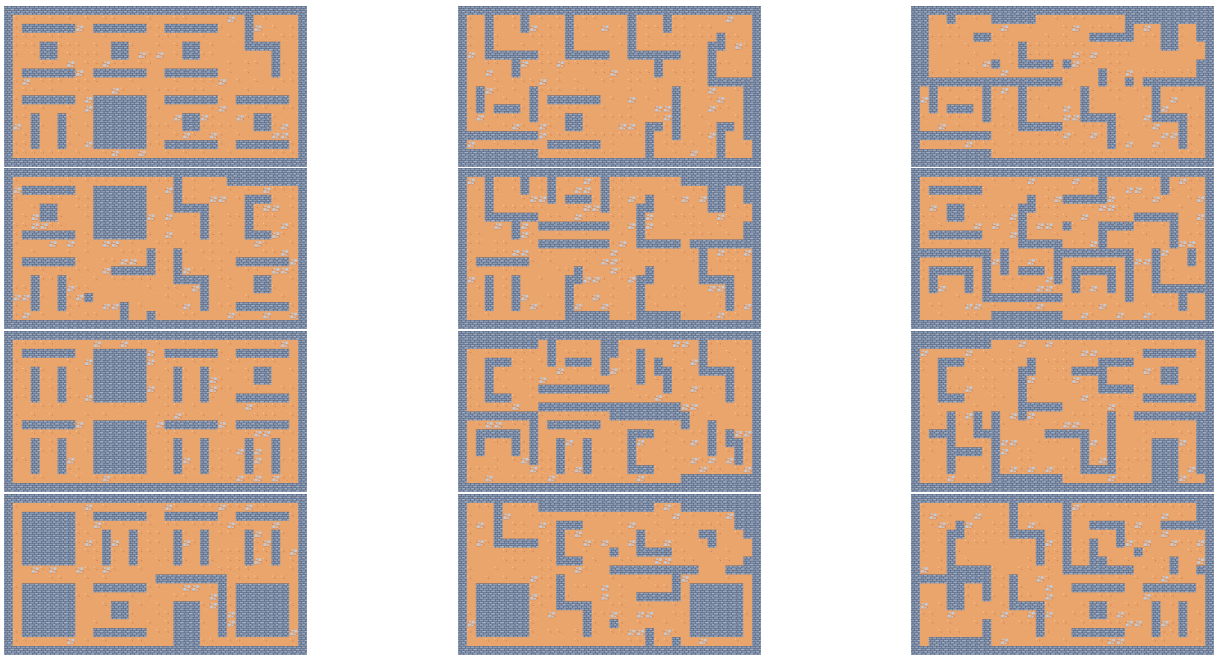
Figura 36 – Gráfico de distribuição da entropia em cada ambiente utilizando o agente $\mathbf{A}_{HQ}$.




(a) Dungeon (b) MazeCoin (c) Zelda

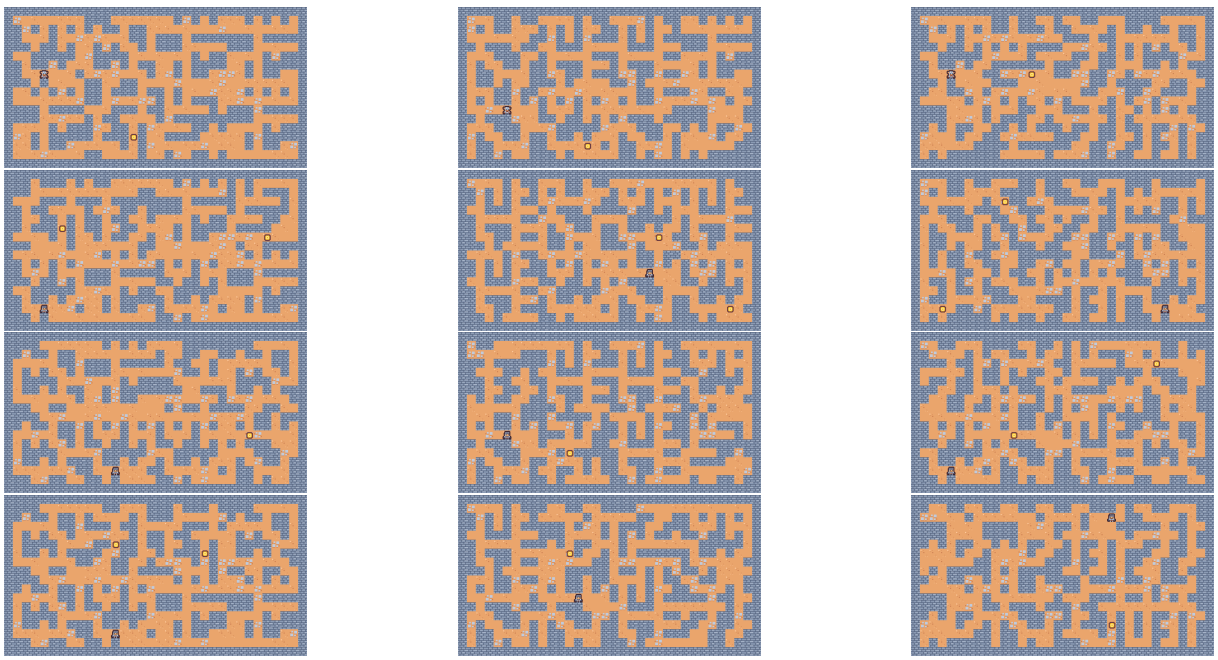
# APÊNDICE B – Cenários gerados com 8 segmentos

Figura 37 – Cenários com 8 segmentos gerados por cada um dos agentes para o ambiente Dungeon.



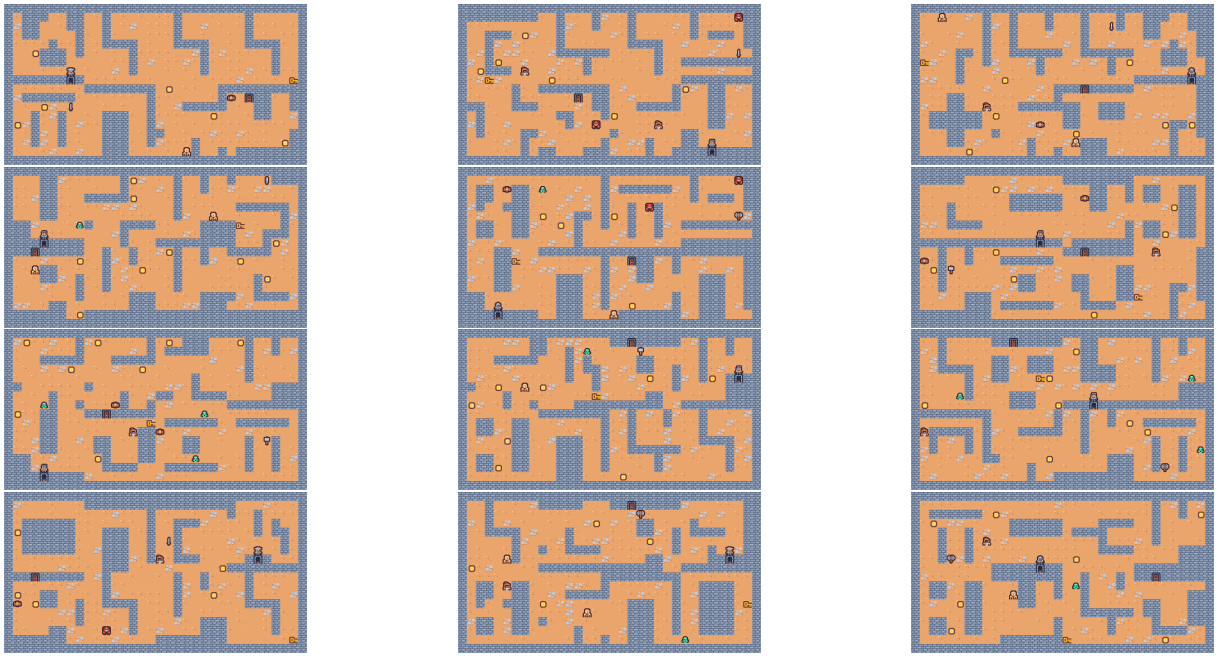
(a) Agente $\mathbf{A}_S$ (b) Agente $\mathbf{A}_H$ (c) Agente $\mathbf{A}_{HQ}$

Figura 38 – Cenários com 8 segmentos gerados por cada um dos agentes para o ambiente MazeCoin.



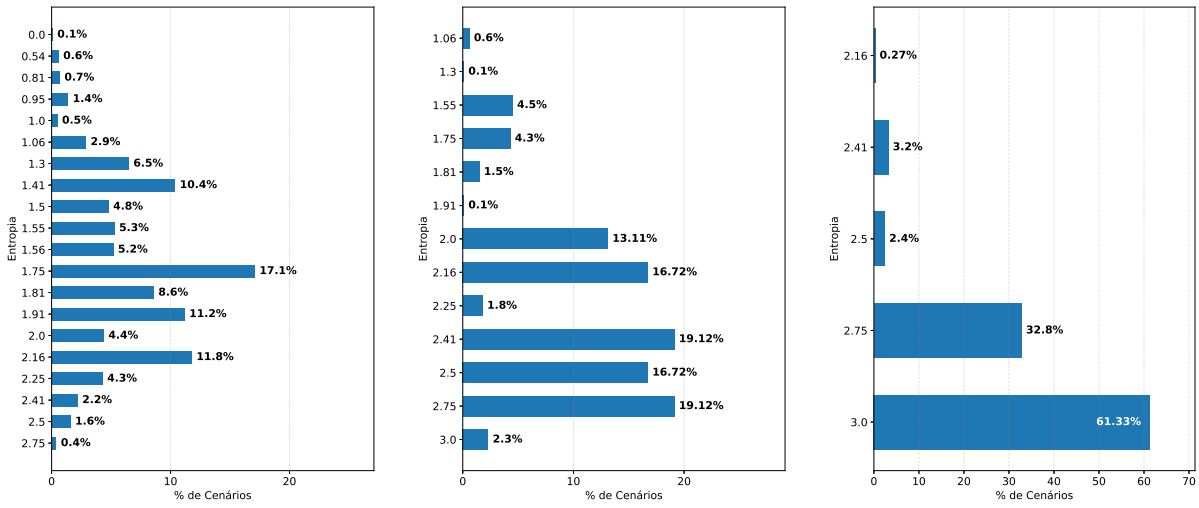
(a) Agente $\mathbf{A}_S$ (b) Agente $\mathbf{A}_H$ (c) Agente $\mathbf{A}_{HQ}$

Figura 39 – Cenários com 8 segmentos gerados por cada um dos agentes para o ambiente Zelda.



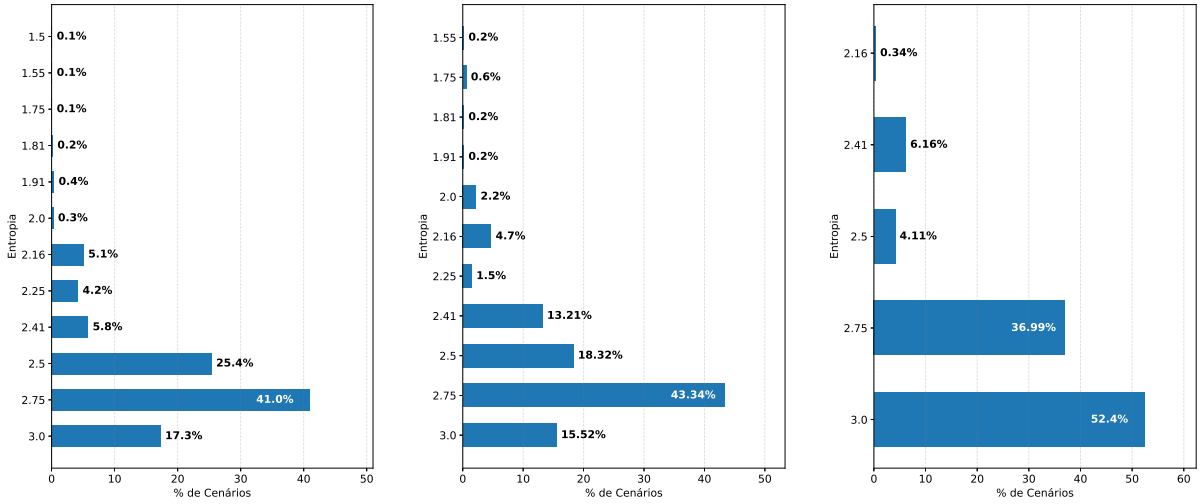
(a) Agente $\mathbf{A}_S$ (b) Agente $\mathbf{A}_H$ (c) Agente $\mathbf{A}_{HQ}$

Figura 40 – Gráfico de distribuição da entropia em cada ambiente utilizando o agente $\mathbf{A}_S$.



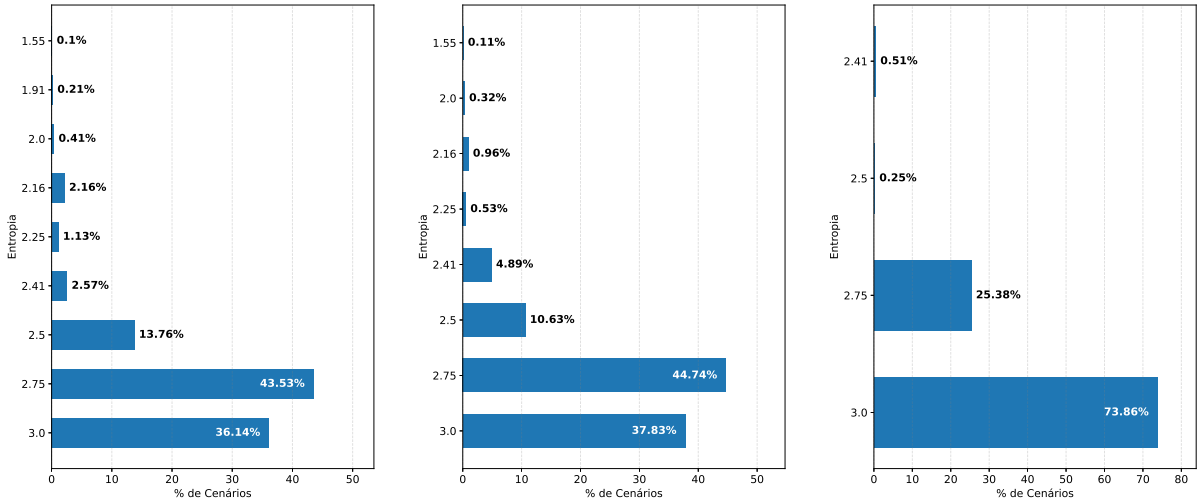

(a) Dungeon (b) MazeCoin (c) Zelda

Figura 41 – Gráfico de distribuição da entropia em cada ambiente utilizando o agente $\mathbf{A}_H$.




(a) Dungeon (b) MazeCoin (c) Zelda

Figura 42 – Gráfico de distribuição da entropia em cada ambiente utilizando o agente $\mathbf{A}_{HQ}$.




(a) Dungeon (b) MazeCoin (c) Zelda

# ANEXO A – Trabalhos relacionados

Figura 43 – Níveis gerados proceduralmente para os jogos Solarfox, Zelda, Frogs e Boulderdash com diferentes valores de dificuldade. (JUSTESEN et al., 2018)

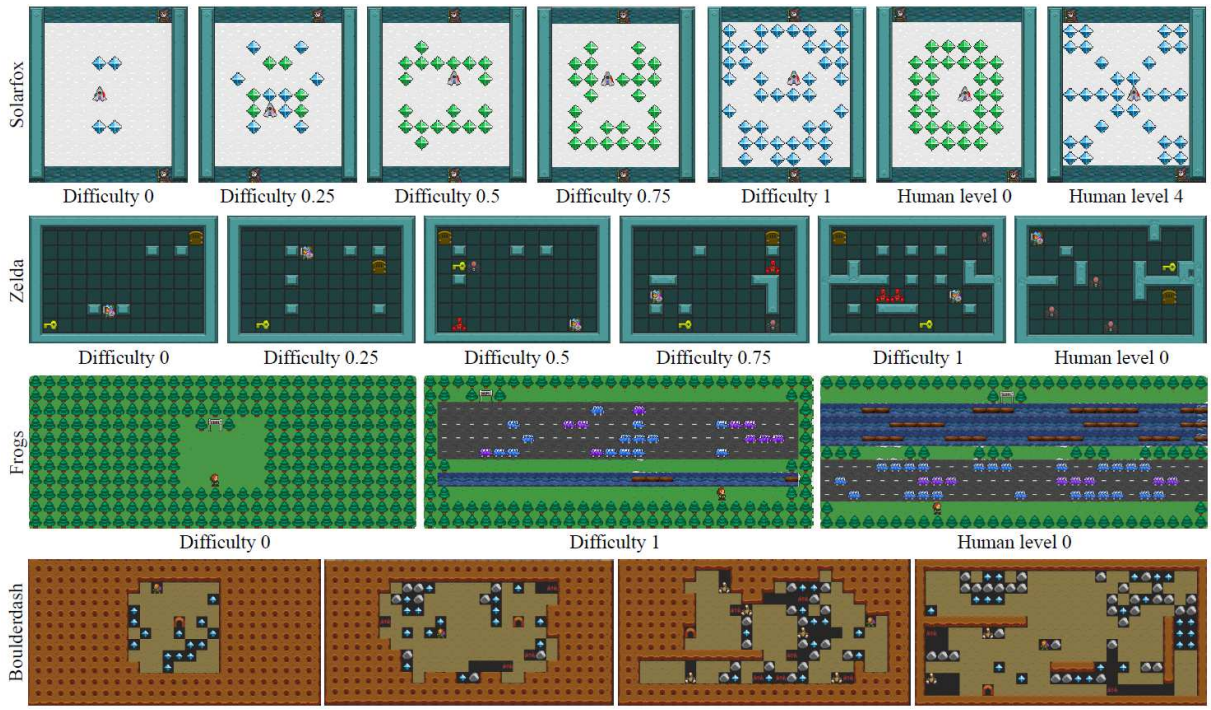



Figura 44 – Arquitetura do framework PCGRL (KHALIFA et al., 2020).

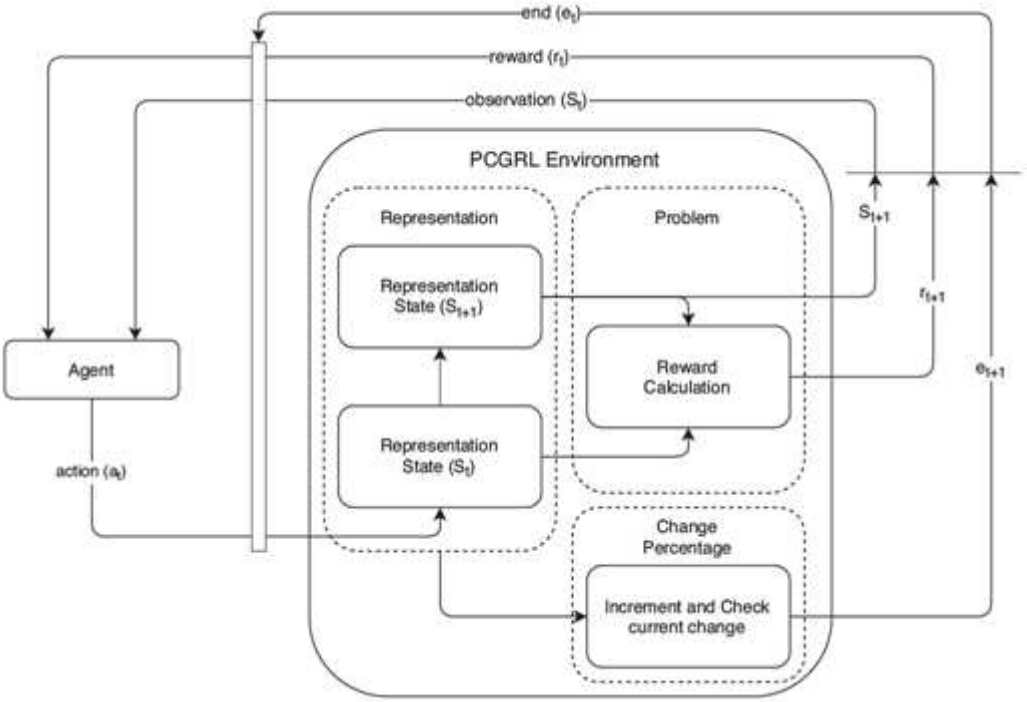


Figura 45 – Arquitetura do modelo proposto no trabalho Adversarial RL for PCG (GISSLEN et al., 2021).

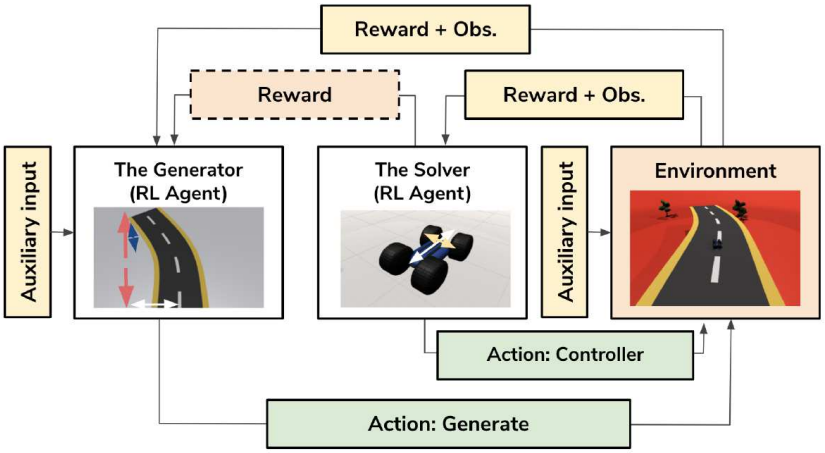



Figura 46 – Arquitetura para implementação EDRL (SHU; LIU; YANNAKAKIS, 2021)