Reinforcement Learning para treino do Pac-Man em speedrun

Emanuel Sales dos Santos Junior
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Rua Emídio dos Santos, Barbalho, Salvador, Bahia
emanuelsales@ifba.edu.br

Antônio Carlos dos Santos Souza
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Rua Emídio dos Santos, Barbalho, Salvador, Bahia
antoniocarlos@ifba.edu.br

RESUMO

Os algoritmos de *Reinforcement Learning* tem o objetivo de melhorar o comportamento do agente em um ambiente específico. Esses algoritmos se mostraram excelentes na melhoria da inteligência artificial em *games* de diversos gêneros. Nesse trabalho é usado o algoritmo *Q-Learning* do *Reinforcement Learning* aplicado no jogo Pac-Man, um dos jogos mais tradicionais que fomentam o desenvolvimento de novos algoritmos para a área de inteligência artificial. O objetivo principal é treinar o Pac-Man para escolher os melhores caminhos e assim terminar o jogo o mais rápido possível, de acordo com o modo de jogo *speedrun*. Os resultados obtidos por esse trabalho mostraram que foi possível fazer o Pac-Man ter bons resultados no modo *speedrun*.

Palavras-chave

Reinforcement Learning; Games; Q-learning; Speedrun; Machine Learning; Pac-Man.

1. INTRODUÇÃO

De acordo com Arthur Samuel [1], aprendizagem de máquina, ou *machine learning*, é um ramo da inteligência artificial (IA) que dá ao computador a habilidade de aprender uma tarefa sem ter sido explicitamente programado, tendo o aprendizado com base em dados que foram alimentados para o computador.

Um dos primeiros usos da IA para jogos foi feito em 1969, por Arthur Samuel [1], que conseguiu mostrar que é possível fazer a IA vencer pessoas reais em uma partida de um jogo de damas. Com isso, outras pesquisas relacionadas a *Machine Learning* foram desenvolvidas, porém em jogos mais complexos que o jogo de damas.

Em outubro de 2015, o CEO da empresa SpaceX Elon Musk e um grupo de investidores fundaram um grupo chamado OpenAI [2], que tinha como principal interesse desenvolver sistemas de *Reinforcement Learning*, uma sub-área de *Machine Learning* especializada no aprendizado de algoritmos via sistema de recompensa e punição. Em maio de 2017, a Google criou uma inteligência artificial chamada de ALPHA GO [2], que fez uso de uma técnica de *Machine Learning* para jogar um jogo popular na ásia, o GO, e vencer o Ke Jie, o melhor jogador de GO do mundo na época. Em agosto de 2017, Elon Musk e seu grupo mudaram o seu foco para o jogo Dota 2. Após 2 semanas de treino, a inteligência artificial do OpenAI conseguiu em um campeonato vencer vários jogadores que estavam no *ranking* mundial [2].

De acordo com Narula [3], a área de jogos é relevante para o estudo de *Machine Learning*, pois essa área possui tudo que é necessário para aplicação de seus algoritmos, permitindo a realização de comparações e testes de diferentes técnicas, uma vez que os jogos proporcionam um ambiente controlado excelente para o funcionamento desses algoritmos. Contudo, Ortega [4] alerta que o uso de *Machine Learning* em jogos requer cautela, pois ao se projetar um algoritmo de *Machine Learning*, é necessário levar em conta algumas considerações como, por exemplo, o nível de dificuldade desejado para o jogo, que permite que um algoritmo possa ser ajustado para tornar a vitória do jogador impossível ou a uma dificuldade que torna o jogo mais fácil, sem nenhum desafio.

Com o intuito de se desenvolver e avaliar um novo algoritmo de inteligência artificial, o jogo escolhido para a pesquisa foi o Pac-Man, que é um jogo leve e simples, possui apenas 4 comandos básicos (Cima, Baixo, Direita e Esquerda), que o diferenciam e popularizam o seu uso como um ambiente de testes para algoritmos de *Reinforcement Learning*.

No Pac-Man, o jogador tem como objetivo comer todas as pastilhas disponíveis no jogo. A pastilha normal garante 10 pontos e a super pastilha garante 50 pontos ao jogador, enquanto um fantasma garante 200 pontos. Na modalidade de *speedrun*, o objetivo é priorizar a aquisição das pastilhas e dos fantasmas da forma mais rápida possível.

Esse trabalho tem como problema checar a possibilidade da inteligência artificial em fazer *speedrun*, aplicado no Pac-Man utilizando o *reinforcement learning*, e tem como objetivo a validação do algoritmo de *Reinforcement Learning*, especificamente utilizando a tecnologia *Q-Learning*, aplicada no jogo Pac-Man. O objetivo principal do trabalho é treinar o Pac-Man a fim de fazer com que a IA consiga terminar o jogo o mais rápido possível no modo *speedrun*, cujo o objetivo, segundo Chase [5], é conseguir 10000 pontos de forma rápida e eficiente. O trabalho apresenta os resultados finais comparados com a de liderança de *speedrun* [5]. É esperado que o resultado final desse trabalho, mediante as alterações no algoritmo de inteligência artificial para o Pac-Man, seja superior ou igual a pelo menos os 3 primeiros colocados da liderança mundial do Pac-Man em *speedrun*.

O restante deste trabalho está organizado da seguinte forma: é apresentada na Seção 2 a fundamentação teórica que foi utilizada como base para o entendimento sobre *Reinforcement Learning*, *Q-Learning*, *speedrun*, o jogo Pac-Man e *Markov Decision Process*. Na Seção 3 estão os trabalhos relacionados, bem como uma análise comparativa entre as

diferentes abordagens utilizadas por esses trabalhos. A Seção 4 apresenta todas as partes importantes do algoritmo proposto para *speedrun*. Na Seção 5 são apresentados os experimentos e resultados, onde é feita uma análise dos resultados comparando com a de liderança de mundial de *speedrun*. Por último, na Seção 6, está a conclusão.

2. FUNDAMENTAÇÃO TEÓRICA

Essa Seção apresenta as principais referências utilizadas para a criação deste trabalho. A subseção 2.1 apresenta o jogo Pac-Man e sua história. Na subseção 2.2 é apresentado de forma geral o que é *speedrun*. A subseção 2.3 apresenta os paradigmas de *Machine Learning: Supervised Learning, Unsupervised Learning e Reinforcement Learning*. A subseção 2.4 apresenta de forma geral o que é *Q-Learning* e *Deep Q-Learning*. Por último, a subseção 2.5 apresenta *Markov Decision Process*.

2.1 O jogo Pac-Man

Pac-Man foi um jogo lançado em outubro de 1980, criado por Toru Iwatani, licenciado pela MIDWAY e desenvolvido pela NAMCO. A inspiração para a criação do Pac-Man veio enquanto Iwatani comia uma pizza. Por isso, o Pac-Man lembra o formato de uma pizza sem uma de suas fatias [6].

O objetivo geral do jogo Pac-Man é fazer com que o personagem principal, o Pac-Man coma todas as 240 pastilhas e 4 super pastilhas disponíveis em um labirinto, enquanto foge dos fantasmas, personagens que tem como objetivo impedir que o Pac-Man coma as pastilhas e vença o jogo. O próprio Pac-Man também pode comer os fantasmas temporariamente, desde que coma antes uma super pastilha disponível no jogo. Para o modo de jogo do *speedrun*, o objetivo é conseguir o máximo de pontuação em um menor tempo possível. A Tabela 1 ilustra, de forma resumida, o sistema de pontuação do Pac-Man. Além disso, se o Pac-Man comer mais de um fantasma logo depois de comer a super pastilha, a pontuação se torna ainda maior, pois o jogo dispõe de um multiplicador que dobra a pontuação por comer cada fantasma a mais. Cada fase do jogo é completada quando o Pac-Man come todas as pastilhas disponíveis no labirinto.

Item	Pontos
Pastilha Normal	10
Super Pastilha	50
Fantasma	200
Fruta	Entre 100 e 5000

Tabela 1: Sistema de pontuação do jogo Pac-Man.

De acordo com Birch [7] os fantasmas possuem algumas mecânicas únicas, por exemplo, existem dois modos o *Chase* e o *Scatter*, no *Scatter* os fantasmas fogem do Pac-Man e no *Chase* os fantasmas perseguem o Pac-Man, mas cada um possui uma personalidade própria, ou, em outros termos, movimentações diferentes como pode ser visto na Figura 1, No modo *Chase* os fantasmas se movimentam da seguinte forma:

- Fantasma vermelho ou Blinky: Sempre se movimenta para a posição atual do Pac-Man.
- Fantasma rosa ou Pinky: se movimenta para a posição atual do Pac-Man mais 4 posições a frente dele.
- Fantasma azul ou Inky: se movimenta para a posição do Pac-Man mais o resultado da distância do Blinky para o Pac-Man multiplicado por 2 é o número de posições a frente do Pac-Man que ele vai se movimentar.
- Fantasma laranja ou Clyde: se movimenta para a posição do Pac-Man, mas quando estiver menos de 8 posições próximo do Pac-Man ativa o modo *Scatter*.

Já no modo *Scatter* eles se movimentam da seguinte forma:

- Fantasma vermelho ou Blinky: Foge do Pac-Man e fica no canto superior direito da tela.
- Fantasma rosa ou Pinky: Foge do Pac-Man e fica no canto superior esquerdo da tela.
- Fantasma azul ou Inky: Foge do Pac-Man e fica no canto inferior direito da tela.
- Fantasma laranja ou Clyde: Foge do Pac-Man e fica no canto inferior esquerdo da tela.

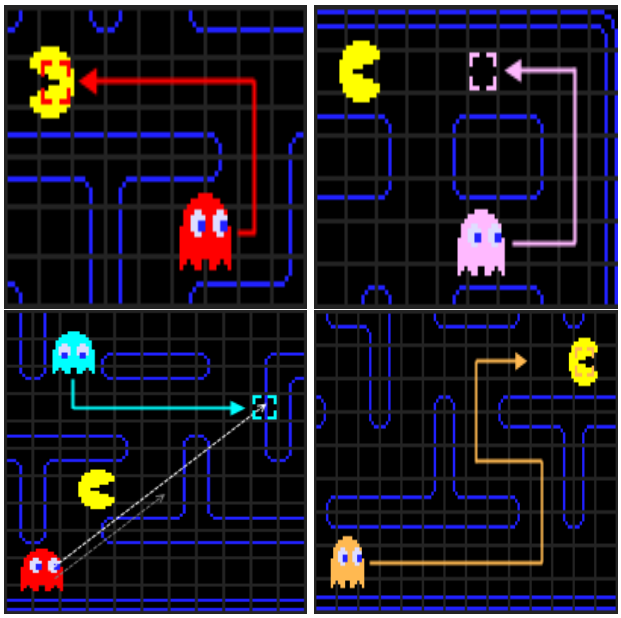

Figura 1: Movimentações dos fantasmas

Os dois modos ficam ativos por tempos diferentes, por exemplo:

- Primeiro turno: *Scatter* por 7 segundos depois *Chase* por 20 segundos;
- Segundo turno: *Scatter* por 7 segundos depois *Chase* por 20 segundos;
- Terceiro turno: *Scatter* por 5 segundos e *Chase* por 20 segundos;
- Último turno: *Scatter* por 5 segundos depois *Chase* permanentemente.

Ortega [4] afirma que o Pac-Man tem sido uma boa ferramenta para realizar pesquisas sobre *Machine Learning* nos jogos, além disso o Pac-Man também ajuda a encorajar os estudantes a aprender inteligência artificial, o que faz com

que o caminho do aprendizado se torne mais fácil. Com isso, existem muitos estudos sobre inteligência artificial abordando o jogo do Pac-Man, sendo que cada um desses estudos escolhe os recursos e o modo de jogo de acordo com seu trabalho. Neste trabalho o modo de jogo escolhido para o desenvolvimento do algoritmo de IA é o *speedrun*, que será apresentado na subseção a seguir.

2.2 O modo de jogo Speedrun

Speedrun pode ser definido como um modo de jogo em que o objetivo é completar o jogo o mais rápido possível sem usar nenhuma trapaça externa, fazendo isso não importa qual foi o meio usado para completar o jogo [8] [9] [10]. De acordo com Bertoli [11], cada jogo tem diversas categorias de *speedrun* que são previamente aceitas pela comunidade para então os competidores escolherem entre qual delas competir. Alguns exemplos de categorias existentes podem ser vistos na Tabela 2.

Categoria	Descrição
100%	Completar todos os objetivos do jogo, o que envolve coletar todos os *items*, derrotar chefes, terminar missões secundárias etc
Any%	Completar o jogo simplesmente alcançando o seu final sem importar de que forma terminou
Blind	Completar o jogo sem ter conhecimento sobre nenhuma das estratégias para esse jogo.
Glitchless	Completar o jogo sem o uso dos *glitchs*.
TAS	Completar o jogo através de um software ou um programa com um emulador que mostra a rota mais rápida possível.

Tabela 2: Categorias de *speedrun* segundo [12].

Adelikat [13] apresenta TAS (*Tool Assisted Speedrun*) como um *speedrun* feito a partir de ferramentas, alguns exemplos são:

- Avanço de quadro (*Frame Advance*): é um recurso de emulação para avançar no jogo quadro a quadro, parecido com recursos de câmera lenta. O avanço de quadro é utilizado para poder pressionar os botões no momento certo no jogo.
- Edição hexadecimal: É utilizado para editar a entrada que o jogo recebe. É feito para reordenar ou copiar e colar sequências de entrada. Não são editados os jogos, nem as imagens ou o som.
- *Re-recording*: É utilizado para substituir uma parte gravada por outra. Por exemplo, para carregar um *savestate* de um evento anterior depois de uma ação incorreta. Em um jogo esse recurso pode ser usado entre 50 a 200000 vezes dependendo da dificuldade do jogo.

Um ponto parecido entre utilizar *Reinforcement Learning* no *speedrun* e *Tool Assisted Speedrun* é que as duas formas não são praticadas por humanos, e uma diferença entre os dois como Adelikat [13] apresenta em relação ao *Tool Assisted Speedrun*, é que antes de começar o desenvolvimento para o *speedrun* em um jogo, é necessária uma análise feita pelos desenvolvedores, entendendo como o *layout* dos níveis funciona, aprendendo não somente as fraquezas e pontos fortes dos itens, inimigos, chefes, entre outros, bem como aprendendo também como o jogo funciona, a partir de uma análise de fatores como: movimentos, ataques, bônus etc. A diferença com *Reinforcement Learning* no *speedrun* é que o agente é responsável em aprender todos esses pontos apresentados pelo autor, o desenvolvedor só tem o trabalho em mostrar quais são as recompensas pelas ações do agente no jogo.

Nos últimos anos, o *speedrun* cresceu de forma rápida por todo o mundo, principalmente por causa do evento *Games Done Quick* [10] que tem o objetivo de fazer maratonas de *speedrun* ao vivo para todo o mundo, através de plataformas conhecidas como Twitch e Youtube. Além disso, todo dinheiro arrecadado é doado para instituições de combate ao câncer [10]. Existem sites que foram criados pela própria comunidade para organizar os recordes de cada competidor. Um desses sites, por exemplo [5], mostram a tabela de liderança mundial do jogo Pac-Man com os seus melhores *speed-runners*, nome dado aos jogadores participantes do *speedrun*. A hipótese apresentada por esse trabalho é fazer com que a inteligência artificial do Pac-Man treinada pelo algoritmo de *Reinforcement Learning*, consiga alcançar os resultados dos 3 primeiros colocados da tabela de liderança mundial de *speedrun* do jogo Ms. Pac-Man.

Através de uma análise dos melhores colocados da tabela de liderança mundial de *speedrun* na categoria de 10000 pontos [5], é possível chegar na conclusão de que todos que estão entre os 10 primeiros colocados usam estratégias muito parecidas, e o que diferencia um do outro é a sua percepção e também a velocidade em realizar essas estratégias. Por isso, se faz necessário estudar a área de aprendizagem de máquina, a fim de que se possa conhecer como aplicar essas estratégias que permitem que os competidores estejam nas melhores posições do *ranking* do *speedrun*.

2.3 Os Paradigmas de Machine Learning

Machine Learning é uma área de estudo que tem como objetivo principal fazer com que o computador complete uma tarefa sem ser explicitamente programado. De acordo com Sutton e Barto [14], a área de *Machine Learning* pode ser dividida em três paradigmas: *Supervised Learning* (ou aprendizagem supervisionada), *Unsupervised Learning* (ou aprendizagem não-supervisionada) e *Reinforcement Learning* (ou aprendizagem baseada em recompensa).

De acordo com Loon [15], no *Supervised Learning*, o usuário possui os dados que serão usados no algoritmo. Nesse caso, a máquina já conhece o que vai fazer antes de aprender ou começar a trabalhar em algo. Um exemplo básico desse conceito seria um aluno estudando em um curso onde ele já sabe o que será dado nesse curso.

Já no *Unsupervised Learning*, a máquina não possui os dados, e os resultados para a maioria dos problemas são amplamente desconhecidos. Esse paradigma tem a capacidade de interpretar e encontrar soluções através de mecanismos de lógica binária e entrada de dados [15].

Enquanto alguns autores afirmam que *Reinforcement Learning* é um tipo de *Unsupervised Learning*, pois ela também não se baseia em exemplos, diferentemente das técnicas tradicionais de *Unsupervised Learning*, *Reinforcement Learning* tenta maximizar o sinal de recompensa em vez de tentar encontrar estruturas escondidas. Descobrir estruturas em uma experiência do agente pode ser útil no *Reinforcement Learning*, mas isso por si só não aborda o problema que é de maximizar um sinal de recompensa. Por isso, *Reinforcement Learning* é considerado um paradigma direto de *Machine Learning*.

Uma vez que o foco deste trabalho está no uso de um algoritmo de *Reinforcement Learning* para uso no jogo do Pac-Man, será apresentada na próxima sub-seção a fundamentação teórica necessária para entendimento do algoritmo proposto.

2.4 Reinforcement Learning

A ideia apresentada por Sutton e Barto [14] sobre *Reinforcement Learning* é aprender o que fazer e como mapear as situações para ações, de modo a maximizar a sua recompensa. O agente não sabe quais ações tomar, mas precisa descobrir quais ações produzem mais recompensa. Em casos mais desafiadores, as ações além de influenciar as recompensas imediatas, também podem influenciar a próxima situação e com isso, todas as recompensas subsequentes. Sendo assim, um agente é treinado para interagir com um ambiente descobrindo os melhores caminhos possíveis para alcançar um objetivo. Neste trabalho, o agente do jogo será o Pac-Man, o ambiente será a fase do jogo, e o objetivo será terminar o jogo o mais rápido possível alcançando 10000 pontos no jogo.

Como foi apresentado por Dayan e Niv [16] os métodos do *Reinforcement Learning* são divididos em 2 grandes classes *model-based e model-free*. *Model-based Reinforcement Learning* usa a experiência para construir um modelo interno de transições e resultados imediatos no ambiente. *Model-free Reinforcement Learning*, por outro lado, usa a experiência para aprender diretamente uma ou duas das quantidades mais simples (estado/valores de ação ou políticas) que podem alcançar o mesmo comportamento ideal, mas sem estimativa ou uso de um modelo global [16]. Nesse trabalho será usada uma técnica *model-free*, uma vez que essa estratégia se encaixa melhor na proposta de jogo do Pac-Man.

Alguns elementos-chave precisam ser explicados para um melhor entendimento do *Reinforcement Learning*. Eles são: *Model*, *Policy*, *Reward*, *Value function* [17]:

- Recompensa (*Reward*): É um sinal de *feedback* que indica o quão bem o agente está indo em um intervalo de tempo. O trabalho do agente durante a fase de aprendizagem é maximizar as recompensas. De forma geral, uma recompensa é representada por um número. A recompensa com valor positivo indica que o agente fez alguma tarefa de forma apropriada. A recompensa com valor nulo indica que o agente não realizou nenhuma tarefa relevante. Por fim, a recompensa com valor negativo indica que o agente fez uma tarefa prejudicial ao seu objetivo.
- Política (*Policy*): É uma distribuição de probabilidade sobre ações dadas em um estado, ou seja, é o comportamento do agente ou como ele escolhe suas ações em algum estado específico.
- Função de Valor (*Value function*): É uma função que diz o quão bom é cada estado e/ou ação, ou seja, como é bom estar em determinado estado, e como é bom tomar uma certa ação. A função de valor informa ao agente quanto de recompensa o espera ao tomar uma ação em um determinado estado.
- Modelo (*Model*): O modelo prevê o que o ambiente fará em seguida, é a representação do ambiente para o agente, ou seja, como o agente acredita que o ambiente funciona.

Sutton e Barto [14] apresentam que no *Reinforcement Learning*, o proposito ou objetivo do agente é formalizado em termos de sinal especial, chamado de recompensa, passando do ambiente para o agente. A cada intervalo de tempo, a recompensa é um simples número. Informalmente, o objetivo do agente é maximizar a quantidade total de recompensas recebidas. Isso significa maximizar não apenas a recompensa imediata, mas a recompensa cumulativa a longo prazo. Por exemplo, para fazer um robô aprender a andar, é necessário atribuir recompensas a cada intervalo de tempo proporcional ao movimento para frente do robô. Para fazer um robô aprender a escapar de um labirinto, a recompensa para cada intervalo de tempo que ele passa tentando escapar é -1. Isso encoraja o agente a escapar o mais rápido possível. Para fazer um robô aprender a encontrar e coletar latas vazias para reciclar, é necessário dar uma recompensa de 0 na maior parte do tempo, e quando coletar uma lata, dar uma recompensa de +1. E um último exemplo em um jogo de damas, onde o agente precisa aprender a jogar, algumas das recompensas são: por ganhar +1, -1 por perder e 0 por empatar.

Os autores [14] mostram que o agente sempre tenta aprender como maximizar suas recompensas. Para que o agente faça algo de interesse da aplicação, é necessário fornecer recompensas, de forma que o próprio agente tente maximizá-la para que o objetivo da aplicação seja alcançado. Por isso, é necessário que as recompensas estabelecidas indiquem o que se deseja realizar. Em particular, o sinal de recompensa não é o lugar de conhecimento prévio sobre como conseguir o que se deseja fazer. Por exemplo, em um jogo de xadrez, o agente deve ser recompensado apenas por ganhar, e não por alcançar pequenos objetivos como ganhar controle do centro do tabuleiro. Se ele alcançar esses pequenos objetivos e for recompensado, então o agente pode encontrar uma maneira de alcançá-los sem atingir o objetivo real como, por exemplo, ele pode encontrar uma maneira de pegar as peças do oponente mesmo ao custo de perder o jogo. Por isso, o sinal de recompensa é a maneira de se comunicar com o agente sobre o que se deseja alcançar e não como se quer que seja alcançado.

Ouderra [18] explica que a política ótima para o agente é sempre selecionar a ação mais efetiva com base nas situações anteriores. Para melhorar ou aprender a política, o agente precisa encontrar situações que nunca foram observadas antes, portanto, surge um *trade-off* entre realizar a ação que leva a maior futura pontuação esperada (*exploiting*) ou tentar ações que levam para novas situações com o objetivo de adquirir novos conhecimentos (*exploring*). O autor mostra que existem diferentes formas para lidar com o dilema de *exploration vs exploitation*. O método mais simples é a seleção gananciosa (*greedy selection*) no qual o algoritmo sempre seleciona a ação encontrada mais efetiva, logo será sempre *exploit*. Quando aplicado ao *Q-learning*, a estratégia de se-

leção vai resultar na seleção da ação.

Sutton [14] apresenta algumas políticas de seleção de ação que tem o objetivo garantir o balanceamento do *trade-off* entre *exploitation/exploration*, Elas são:

- ϵ-greedy: Na maior parte do tempo a ação com a maior recompensa será escolhida, e tem uma probabilidade baixa ϵ, de uma ação ser escolhida aleatoriamente.
- ϵ-soft: Parecido com ϵ-greedy. A melhor ação é escolhida com a probabilidade ϵ - 1 e o resto do tempo são escolhidos ações aleatórias.
- softmax: Cada ação é classificada de acordo com a estimativa do valor da ação, ou seja, as piores ações não serão escolhidas, diferentemente de ϵ-greedy e ϵ-soft que ao escolher uma ação aleatória pode estar escolhendo a pior ação.

De acordo com [19], uma das áreas que torna o *Reinforcement Learning* muito útil são os jogos de computadores. De forma geral, o personagem estará em um conjunto de diferentes estados que podem ser enumerados, e que também podem ser descritos pelo estado atual do personagem e do jogador. O personagem tem um conjunto de diferentes ações que podem ser executadas e, além disso, precisa ser capaz de decidir qual é a melhor ação. O personagem pode receber recompensas positivas ou negativas dependendo do resultado de uma sequência de decisões. Essa recompensa vai depender do jogo e dos objetivos do personagem. Se por exemplo, o objetivo for derrotar o jogador isso fará com que tenha uma recompensa positiva e no caso oposto uma recompensa negativa.

2.5 Q-Learning

O motivo para ser usado o *Q-Learning* é por ser um *model-free* muito conhecido e usado em pesquisas, trabalhos e artigos por todo o mundo, principalmente aos que são aplicados em jogos. Na literatura, o *Q-Learning* já foi utilizado tanto no Pac-Man [20], quanto em jogos de primeira pessoa [21].

De acordo com Watkins [22] *Q-Learning* é um algoritmo de *Reinforcement Learning Model-Free* que foi desenvolvido baseado na diferença temporal tratada com programação dinâmica assíncrona. Ele fornece recompensas para os agentes com a capacidade de aprender a agir otimamente em domínios de *Markovian*, experimentando as consequências das ações, sem precisar que eles construam o mapa do domínio.

A principal ideia do *Q-Learning* é usar uma única estrutura de dados chamada função de utilidade Q(x,a). Essa utilidade de executar uma ação no estado x *Markovian*.

McCullock [23] explica que o que distingue *Q-Learning* de outras técnicas é a sua capacidade de escolher entre as recompensas imediatas e recompensas futuras. A cada intervalo de tempo t, um agente observa o vetor do estado X_t, então escolhe e aplica uma ação U_t. Conforme o processo se move para o estado X_{t+1}, o agente recebe um reforço $R(X_t, U_t)$. O objetivo do treinamento é encontrar a sequência de ações que maximiza a soma de reforços futuros, levando assim ao caminho mais curto do início ao fim.

McCullock [23] apresenta a fórmula da atualização do *Q-Value*

- Q(estado, ação) = R(estado, ação) + gama * Max[Q(próximo estado, todas ações)]
  - Q(estado, ação): é o valor atribuído a ação em um determinado estado.
  - gama (fator de regularização): O parâmetro gama tem um intervalo de 0 a 1 (0 <= gama > 1) e garante a convergência da soma. Se o gama estiver mais próximo de 0, o agente tenderá a considerar apenas recompensas imediatas. Se o gama estiver mais próximo de 1, o agente considerará recompensas futuras com maior peso, disposto a atrasar a recompensa.
  - R(estado, ação): Recompensa recebida em um estado por ter tomado uma determinada ação.
  - Max[Q(próximo estado, todas ações)]: A recompensa máxima que se pode receber no estado seguinte ao atual, ou seja, a recompensa por tomar a melhor ação consequentemente.
  - alpha (fator de aprendizagem): É um parâmetro que não foi apresentado pelo autor, mas tem sua importância, o parâmetro tem um intervalo de 0 a 1 onde se estiver mais próximo de 0, o agente priorizará informações passadas e se estiver mais próximo de 1 o agente priorizará informações mais recentes. Está diretamente ligado com *exploitation/exploration* apresentado na seção 2.4.

De acordo com McCullock [23], o algoritmo de *Q-Learning* funciona da seguinte forma:

1. Defina o parâmetro gama, e a recompensa do ambiente na matriz R;
2. Inicialize a matriz Q para 0;

for *Para cada episódio* **do**
  while *a meta do estado ainda não foi alcançado* **do**
    1. Selecione uma entre todas as possíveis ações para o estado atual;
    2. Usando essa possível ação, considere ir para o próximo estado;
    3. Obtenha o *Q-Value* máximo para este próximo estado com base em todas as ações possíveis.
       Calcule: Q(estado, ação) = R(estado, ação) + gama * Max[Q(próximo estado, todas ações)];
    4. Defina o próximo estado como o estado atual;
  end
end

De acordo com Matiisen [12] no *Q-Learning* se define a função Q(estado, ação) representando as recompensas futuras descontadas quando realiza uma ação em determinado estado e assim continuando a partir deste ponto. Então, a melhor forma para pensar sobre Q(estado, ação) é como "a melhor pontuação possível no final do jogo depois de realizar a ação A em um estado S", isso é chamado de *Q-function*, pois isso representa a 'qualidade' de certas ações em um determinado estado.

Matiisen [12] apresenta que é possível estimar a pontuação do final de um jogo apenas sabendo a situação atual do

estado e ação, e não das ações e recompensas que vem depois. Ele explica com o seguinte exemplo: suponha que o usuário está em um estado S e ponderando qual ação tomará ou A ou B O usuário deseja escolher a ação que resulta na maior pontuação no final do jogo. Quando o usuário possui *Q-function* a resposta se torna simples, escolher a ação com o maior *Q-value*.

De acordo com Ouderra [18] *Q-Learning* usa tabela de dados que contêm os *Q-values* de pares de estado e ação, porém, devido ao enorme tamanho do espaço de estado no Pac-Man, a quantidade necessária de memória e o número de iterações necessárias para a convergência torna praticamente impossível a configuração dessa tabela. A vantagem de usar o *Deep Q-learning* é que os *Q-values* de pares de estado e ação podem ser intercalado por rede neural convolucional em vez da tabela de dados. O autor explica que a rede neural convolucional é uma variação de rede neural *feed-forward* regular, sua arquitetura foi criada para explorar a conectividade local normalmente encontrada em dados altamente dimensionais como as imagens. Em contrates com as camadas totalmente conectadas em rede neurais regulares, uma camada em uma rede neural convolucional usa convoluções em sua entrada ao calcular sua saída, cada camada tem um intervalo de *kernels* convolucionais ou filtros, que juntos classificam os recursos. E os pesos dos *kernels* são atualizados por gradiente descendente estocástico.

2.6 Processo de Decisão de Markov

De acordo com Alzantot [24] os problemas de *Reinforcement Learning* são descritos utilizando o *framework* chamado de Processo de Decisão de Markov, do inglês *Markov Decision Process* (MDP). MDP é uma versão estendida da cadeia de Markov que acrescenta elementos de decisões e recompensas a ele. A palavra *Markov* refere-se a propriedade Markoviana que com os estado e ação atuais estabelecidos significa que o estado futuro é independente de qualquer histórico de estado anterior. Com isso, o estado atual encapsula tudo que é necessário para decidir o estado futuro quando uma ação é recebida. O autor mostra o seguinte exemplo: em um jogo de xadrez ao realizar um movimento no estado atual do tabuleiro, esse movimento foi pensado a partir de possíveis situações futuras e não a partir de movimentos e estados passados do tabuleiro. Alzatot [24] mostra que MDP é definido como uma tupla de 5 itens <S, A, P, R, Y> que são:

- S: Conjunto de Observações. O agente observa o estado do ambiente como um item desse conjunto.
- A: Conjunto de ações. O conjunto de ações que o agente pode escolher para interagir com o ambiente.
- P: (S' | S,A) Matriz de probabilidade de transição. Isso modela qual será o estado S' depois que o agente realizar uma ação A enquanto está no estado atual S.
- R: P(R | S,A) Modelo de recompensa que modela qual recompensa o agente receberá quando realizar uma ação A quando estiver no estado S.
- Y: Fator de regularização. Esse fator é um valor numérico entre 0 e 1 que representa a importância relativa entre recompensas imediatas e futuras. Ou seja, se o agente tiver que escolher entre 2 ações, uma delas dará uma alta recompensa imediata depois de executar uma ação, mas levará a um estado do qual os agentes esperam receber menos recompensas futuras.

Alzatot [24] conclui que o objetivo do agente do *Reinforcement Learning* é resolver o MDP encontrando política ótima, o que significa encontrar a sequência de ações que ele pode fazer para maximizar a recompensa total recebida.

Matiisen [12] apresenta como se formaliza o problema do *Reinforcement Learning* para que você possa raciocinar sobre isso, a forma mais comum é através do *Markov Decision Process*. Com isso, o autor contextualiza através de uma metáfora, onde suponha que você é um agente, situado em um ambiente, esse ambiente está em um estado, o agente pode realizar uma certa quantidade de ações nesse ambiente, essas ações transformam o ambiente e o guia para um novo estado, onde o agente pode realizar outras ações, e assim por diante. A regra com a qual você escolhe essas ações é chamada de política, além disso, o ambiente em geral é estocástico o que leva o próximo estado de certa forma ser aleatório.

De acordo com Matiisen [12] com base na Figura 2, o conjunto de estados e ações juntamente com as regras de transição de um estado para outro, e também por adquirir recompensas, forma o *Markov Decision Process*. Um episódio desse processo forma uma sequência finita de estados, ações e recompensas:

- S_0, A_0, R_1, S_1, A_1, R_2, S_2, ..., S_{n+1}, A_{n+1}, R_n, S_n.

S_n representa o estado, A_n representa a ação e R_{n+1} representa a recompensa depois de realizar uma ação. O episódio termina como, por exemplo, em um jogo através do *game over*. *Markov Decision Process* depende da suposição *Markov*, de que a probabilidade do próximo estado S_{n+1} dependa apenas do estado atual S_n e da ação realizada A_n, mas não em estados ou ações anteriores.

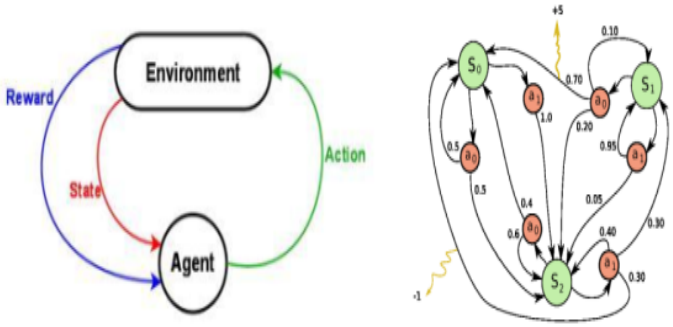

Figura 2: Esquerda: problema do reinforcement learning. Direita: Markov Decision Process.

Oregon State University [25] explica a formulação aplicada no Pac-Man como:

No espaço de ação representa qualquer situação de ação do Pac-Man, como escolher direções ou se movimentar, e no caso de 'a' significa de fato os seus movimentos esquerda, direita, cima e baixo.

No espaço de estado o principal objetivo é em um tempo representar todas as informações relevantes que ajude na tomada de decisão que resolva o MDP. O 's' é o Estado, que são os fantasmas, o Pac-Man e todos os conteúdos presentes no labirinto.

As recompensas são as pontuações que é acumulada do início até o final do jogo como, por exemplo, se o Pac-Man come uma pastilha, um *Ghost*, se o labirinto está limpo e até mesmo quando o Pac-Man é derrotado, o que leva a uma pontuação negativa.

2.7 Redes Neurais Multi-Camadas

O autor Svozil [26] começa explicando que as redes neurais são redes de elementos de processamento simples também chamados de neurônios, operando em seus dados locais e comunicando com outros elementos. Sua estrutura foi inspirada no cérebro real apesar que na prática não é bem assim, porém os princípios básicos são semelhantes. Cada neurônio na rede é capaz de receber sinais de entrada processá-los e enviar um sinal de saída. Cada neurônio está conectado com pelo menos um neurônio e cada conexão é avaliada por um número real, chamado de coeficiente de peso, que reflete o grau de importância da conexão dada na rede neural.

Svozil [26] explica como as Redes Neurais Multi-Camadas (*Multilayer Feed-Forward Neural Network* - MLF) funcionam, é apresentado que a rede neural é treinada com um algoritmo de aprendizagem *back-propagation*. Uma rede neural MLF possui neurônios que são ordenados em camadas, onde a primeira camada é chamada de camada de entrada, a última de camada de saída e as camadas do meio são chamadas de camadas escondidas, cada neurônio de uma camada particular está conectada com todos os neurônios da próxima camada como pode ser visto na Figura 3, e essa conexão é caracterizada pelo coeficiente de peso que indica a importância dessa conexão.

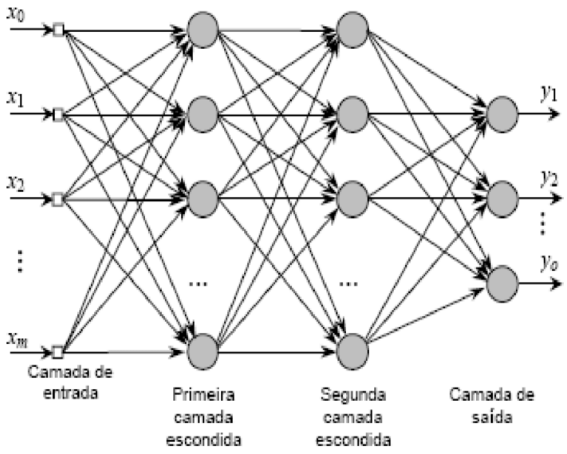

Figura 3: *Multilayer Feed-Forward Neural Network* [27]

De acordo com o autor [26] o algoritmo de *back-propagation* tem o objetivo de ajustar o coeficiente de peso nos neurônios usando a técnica de *Gradient Descent* para calcular o gradiente da função de erro. O nome é *back-propagation*, pois o erro da saída propaga da camada de saída pelas as camadas escondidas até a camada de entrada.

A Figura 4 é explicada por Saurabh [28] da seguinte forma:

- *Calculate the error*(Calcule o erro) – Quão distante é o seu modelo de saída da saída real.
- *Error minimum?*(Erro mínimo?) – Verifique se o erro está minimizado ou não.

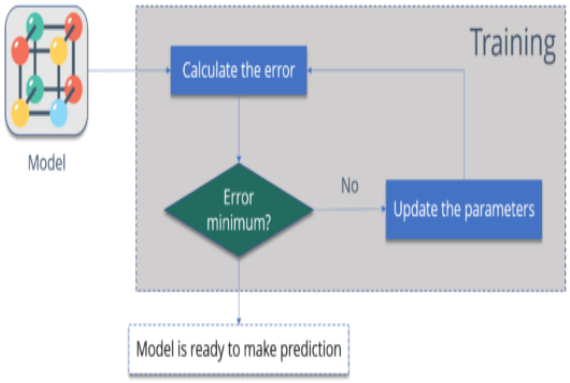

Figura 4: Funcionamento do algoritmo de *back-propagation* [28]

- *Update the parameters*(Atualize os parâmetros) – Se o erro for enorme, então, atualize os parâmetros (pesos e vieses). Depois disso, verifique novamente o erro. Repita o processo até que o erro se torne mínimo.
- *Model is ready to make a prediction*(O modelo está pronto para fazer uma previsão) – Uma vez que o erro se torna mínimo, você pode alimentar algumas entradas para o seu modelo e ele produzirá a saída.

Nesse trabalho a rede neural é treinada pelo *Q-Learning*, e aproxima Q(Estado, Ação) onde a ação é o movimento do Pac-Man e o estado é dado por diversos valores de entradas que são divididos entre os agentes do Pac-Man e Fantasmas. Todos estados serão apresentados e explicados na seção da proposta de *speedrun*.

3. TRABALHOS RELACIONADOS

O algoritmo do trabalho [29] tem como objetivo ensinar a inteligência artificial aplicada no Pac-Man a jogar. A simulação do jogo Pac-Man criada pelos autores tenta ser o mais parecido possível com o original. Na proposta do trabalho são apresentadas as seguintes informações sobre essa simulação: ela tem um Pac-Man, 4 fantasmas, pastilhas e super pastilhas que ao comer todas disponíveis o Pac-Man vence o jogo, e se colidir com qualquer fantasma o Pac-Man perde. Além disso, é apresentado que não tem as frutas como no original que dão como recompensa uma pontuação, possui apenas um *level*, uma vida e o comportamento dos fantasmas são estocásticos o que significa que seus movimentos são aleatórios. Os fantasmas possuem dois estados que é o de perseguir o Pac-Man e apenas se dispersar também conhecido como os modos *chase* e *scatter*. De acordo com Langbroek [29] foi utilizado a técnica do *Q-Learning*, e na parte do treino é levado em conta o desempenho do Pac-Man nas suas vitórias e derrotas. As recompensas foram aplicadas de acordo com a Tabela 3.

Esse algoritmo criado por [29] é usado como base para este trabalho, pois com ele a simulação do jogo Pac-Man está praticamente completa, porém o algoritmo não possui tempo de jogo, não tem pontuação, as diferentes personalidades dos fantasmas não estão implementadas, as recompensas serão necessárias a modificação de todos os seus valores, o labirinto precisa ser recriado, o estado do labirinto não possui alguns pontos importante para o objetivo de *speedrun*.

Evento	Recompensa	Descrição
Pastilha	+9	Pac-Man adquiriu uma pastilha
Super Pastilha	+1	Pac-Man adquiriu uma super pastilha
Eliminar um Ghost	+2	Pac-Man colidiu com um scared Ghost
Vitória	+100	Pac-Man adquiriu todas as pastilhas e super pastilhas
Derrota	-400	Pac-Man colidiu com um Ghost
Mudar de direção	-0.5	Pac-Man mudou de caminho
Passo	-5	Pac-Man se movimentou

Tabela 3: Recompensas aplicadas no Pac-Man segundo a referência [29]

O que foi definido como derrota e vitória precisa ser revisado, no *speedrun* a vitória será quando o Pac-Man alcançar 10000 pontos e a sua derrota não será apenas ao colidir com o fantasma, mas também ao comer todas as pastilhas e super pastilhas sem alcançar os 10000 pontos, por último, no algoritmo desse jogo não tem as frutas que também como as pastilhas e super pastilhas possuem uma certa pontuação.

No trabalho de Estevez et al. [30] é apresentado o treino do Pac-Man com *Reinforcement Learning* e *Case-based Reasoning* usando a tecnologia *Q-Learning* com o objetivo de alcançar o máximo possível de pontos e sobreviver o máximo possível. *Case-Based Reasoning* é uma técnica de aprendizagem e resolução de problemas baseado na intuição que problemas semelhantes geralmente tem soluções semelhantes. Em vez de tentar resolver cada problema a partir do zero, os problemas são resolvidos procurando experiências anteriores semelhantes e adaptando sua solução ao contexto atual. Foi feita uma análise sobre o problema e sua resolução e a partir disso, chegou-se na conclusão de que a utilização do *Case-based Reasoning* não foi muito interessante para o problema desse trabalho, pois os resultados deixaram o Pac-Man muito tempo vivo tentando conseguir o máximo de pontos possível, por isso, para o caso do *speedrun* não é uma solução boa, mesmo assim o trabalho de Estevez et al. serviu de grande ajuda nos processos do experimento.

Bom e Wiering [20] também apresentam o treino do Pac-Man com *Reinforcement Learning*, porém seu objetivo é tornar o treino eficiente e para isso eles criaram as entradas de *Higher-order Action-relative* para serem usados em diferentes labirintos no jogo Ms. Pac-Man, sem precisar fazer o treino novamente. *Higher-order Action-relative* são entradas de dados relativos as ações do Pac-Man. Elas são dadas a uma única rede neural que propaga sequencialmente as entradas relativas às ações para obter *Q-Values* de diferentes ações. Em comparação com esse artigo o treino é feito com a mesma técnica de *Reinforcement Learning*, porém esse trabalho não tem objetivo de criar um treino eficiente, mas os resultados e experimentos do artigo são usados como análise para o trabalho na etapa 2 dos processos do experimento para decidir as melhores formas de treino.

Niel et al. [31] apresentam *Reinforcement Learning* aplicado em jogos de estratégia em tempo real, com os algoritmos do *Q-Learning* e o método de Monte Carlo, a divisão das recompensas foi por unidade individual e/ou compartilhada entre todos no jogo, as unidades são os personagens que fazem parte do jogo como o arqueiro, lanceiro etc, e as recompensas são, por exemplo, inimigos mortos, vitória, derrota etc. Os resultados mostram que o desempenho do *Q-Learning* em relação a quantidade de derrotas e empates foram melhores que o do método de Monte Carlo, mas em relação as vitórias o método de Monte Carlo teve um melhor desempenho, além disso, com as recompensas aplicados individualmente entre as unidades, os autores chegaram na conclusão que o desempenho final foi melhor do que se as recompensas fossem aplicadas compartilhadamente.

4. PROPOSTA DE SPEEDRUN

Essa seção apresenta o funcionamento do algoritmo de *reinforcement learning* no treino do Pac-Man para realizar *speedrun*. Como foi apresentado anteriormente na Seção 2, os sistemas de *reinforcement learning* possuem alguns elementos-chaves, modelo, política, recompensa e *value function*. Nesse trabalho o modelo representa a simulação do game, a política representa como o Pac-Man a partir de valores providos pelo *value function* se comportará em um estado, agentes são representados pelo fantasmas e Pac-Man, estado representa o ambiente em um momento e recompensa representa quais são as recompensas do Pac-Man a partir de suas ações. Nas subseções abaixo será explicado mais detalhadamente sobre todas as partes do algoritmo.

4.1 Modificações no jogo do Pac-Man

Para o jogo do Pac-Man foi utilizado como inspiração o projeto de Lankbroek [29], codificado em C++, que usa o *Q-Learning* com rede neural artificial para treinar o personagem do Pac-Man para que ele consiga jogar o jogo de forma automática. A implementação original de Lankbroek foi adaptada neste trabalho para incluir algumas características próprias do jogo do Pac-Man, principalmente o sistema de pontuação, essencial para o modo de jogo do *speedrun*, a vitória acontece quando consegue coletar 10.000 pontos em menos de dois minutos, e para a derrota, o personagem principal tem que colidir com um dos fantasmas, ou também se não conseguir chegar à pontuação de 10.000 pontos no período específico.

No algoritmo foram feitas muitas mudanças no funcionamento do jogo para atender as necessidades do trabalho, lembrando que todas as mudanças no jogo foram feitas para ser o mais próximo possível do jogo original do Pac-Man como, por exemplo, Adicionado um novo labirinto o mais próximo possível do original. Modificado o tempo dos modos *Scatter* e *Chase* para apenas o primeiro turno e segundo turno explicados na seção 2.1, pois como o jogo tem uma duração de no máximo 2 minutos e o objetivo é diminuir mais ainda esse tempo, não foi necessário adicionar os outros turnos, foi modificado também o tempo que a super pastilha fica ativa. Adicionado a pontuação máxima e o seu número de rounds no treino quando ocorrer uma vitória, para uma melhor análise dos testes. Adicionado o sistema de pontuação onde a pastilha possui 10 pontos, a super pastilha 50 pontos e o fantasma eliminado 200 pontos, além

disso, foi adicionado o combo que ao fantasma for eliminado em sequência o Pac-Man ganha a pontuação do fantasma eliminado anteriormente multiplicado por 2. Foram adicionadas as personalidades dos fantasmas quando eles entram no modo *Chase*, foi implementado da seguinte forma:

- Blinky: Se movimenta para a posição atual do Pac-Man.
- Pinky: Se movimenta para a posição atual do Pac-Man mais 4 posições na direção onde o Pac-Man está apontando.
- Inky: Se movimenta parecido com o Pinky a diferença é o número de posições a frente do Pac-Man, O Inky usa a distância do Blinky para o Pac-Man multiplicado por 2 para ser a posição que vai se movimentar. Possui um limite de 8 posições a mais.
- Clyde: Se movimenta para posição atual do Pac-Man, mas quando estiver pelo menos 4 posições próximo ele entra no modo *scatter* e foge do Pac-Man.

4.2 Modelo e Estado

De acordo com [32] um espaço de estado ideal para representação do jogo Pac-Man poderia incorporar todas as informações incluídas em um *snapshot* do jogo, como: a posição do Pac-Man no labirinto, a situação das pastilhas e super pastilhas e as condições dos fantasmas ao redor do Pac-Man. No algoritmo na *model* foi necessário adicionar um labirinto que fosse mais parecido possível com o labirinto do jogo original do Pac-Man, com 240 pastilhas e 4 super pastilhas.

O estado é representado no algoritmo por diversos valores que definem todas as informações contidas no jogo, além dos estados que já existiam, novos foram adicionados, Todos os estados que foram utilizados para o objetivo desse trabalho são:

1. Pac-Man
   - A posição inicial e a posição atual do Pac-Man.
   - Notificações quando come uma pastilha, super pastilha, quando elimina um fantasma, vence e perde.
2. Fantasmas
   - A posição inicial e a posição atual dos fantasmas.
   - Quando os fantasmas estão no modo *Chase* ou *Scatter*.
   - Duração para a mudança de modo.
   - A duração para o fantasma ser revivido.
   - Tipo de fantasma, por exemplo, Blinky, Pinky, Inky e Clyde.
3. Labirinto
   - Todas as posições e representações visuais das paredes, caminhos livres, área de renascimento, pastilhas, super pastilhas, Pac-Man, Blinky, Pinky, Inky e Clyde.
   - Todas as pontuações, por exemplo, por adquirir pastilhas, super pastilhas, eliminar um fantasma e o total.
   - Duração da super pastilha.
   - Os combos por eliminar fantasmas em sequência.
   - Quantidade total de pastilhas e super pastilhas, também a quantidade total adquiridas pelo Pac-Man.
   - Distância para a pastilha ou super pastilha mais próxima.
   - Distância do fantasma mais próximo.
   - Posição válida para se movimentar.

4.3 Recompensa(Reward)

No algoritmo as recompensas foram modificadas para atingir o objetivo do trabalho, todas elas estão representadas na tabela 4.

Evento	Recompensa	Descrição
Pastilha	+30	Pac-Man adquiriu uma pastilha
Super Pastilha	+500	Pac-Man adquiriu uma super pastilha
Eliminar um Fantasma	+3000	Pac-Man colidiu com o primeiro fantasma
Combo x2	+6000	Pac-Man colidiu com o segundo fantasma
Combo x3	+9000	Pac-Man colidiu com o terceiro fantasma
Combo x4	+12000	Pac-Man colidiu com o quarto fantasma
Vitória	+3000	Pac-Man adquiriu todas as pastilhas e super pastilhas
Derrota	-9000	Pac-Man colidiu com um fantasma
Passo	-10	Pac-Man se movimentou
Direção reversa	-200	Pac-Man se movimentou para uma direção contrária

Tabela 4: Recompensas aplicadas no Pac-Man

As recompensas foram distribuídas com o foco em fazer com que o Pac-Man consiga a maior pontuação possível para dessa forma ter um melhor desempenho no *speedrun*, pois para conseguir chegar aos 10000 pontos uma das formas e a mais rápida é eliminar os fantasmas através de combos, o que significa eliminar um atrás do outro em sequência. Com 3 super pastilhas e 12 fantasmas eliminados em sequência são 9150 pontos sem contar com as pastilhas normais que já levam aos 10000 pontos independente do caminho escolhido para chegar nas super pastilhas.

5. EXPERIMENTOS E RESULTADOS

5.1 Processos do Experimento

Para a realização desse trabalho, foi necessário realizar as seguintes etapas:

1. Foi necessário o estudo e análise do jogo Pac-Man e da tecnologia *Q-Learning* para, com isso, ter as melhores opções de mudança de código para o objetivo de *speedrun* como, por exemplo, garantir um foco maior em estratégias para eliminar os 4 fantasmas em sequência logo após comer a super pastilha, pois é necessário conseguir 10000 pontos o mais rápido possível. Uma estratégia usada por competidores é esperar os fantasmas chegarem perto antes de comer a super pastilha, para então, aumentar as chances de fazer com que o Pac-Man consiga comer os 4 fantasmas em sequência, com isso aproveitar os multiplicadores da melhor forma possível.

2. Com as opções de mudanças de código em mãos as mesmas já puderam ser testadas no jogo, porém como *Reinforcement Learning* não existem dados para serem adicionados a aplicação, pois o próprio paradigma que decide quais são os melhores dados a serem armazenados, então também foi feita uma análise de quanto tempo em média será preciso para treinar o Pac-Man em cada uma dessas opções para uma melhor eficácia.

3. Por último, uma comparação dos resultados alcançados com os resultados que foram almejados, por exemplo, em quanto tempo a inteligência artificial do Pac-Man conseguiu alcançar os 10000 pontos e comparar esse resultados com a de liderança mundial de *speedrun*, e caso não tenha sucesso, servirá como base tudo que foi aprendido para não repetir os mesmos erros, e com isso recomeçar da primeira etapa.

5.2 Configuração Experimental

Para informação todos os treinos foram feitos em um notebook com a seguinte configuração:
Processador: Intel(R) Core(TM) I3-6100U
Memória RAM: 4GB
Sistema Operacional: Arch Linux

Houve 3 possíveis linguagens como opções para a criação do algoritmo, essas são, Java, Python e C++, mas por causa da máquina foi escolhido fazer com a linguagem C++, pois das 3 opções foi a que teve melhor desempenho na máquina citada. Cada treino da rede neural é feita em 5000 jogos que duram em torno de 40 minutos dependendo da quantidade de jogos e dos parâmetros modificados, logo depois do treino são feitas 3 rodadas de testes onde cada rodada possui 5000 jogos que tem uma duração mais rápida do que a do treino. Os experimentos que foram aplicados neste trabalho são:

- No primeiro experimento foram feitas mudanças nos parâmetros do fator de aprendizagem α e o fator de regularização γ no algoritmo *Q-Learning*.
- No segundo experimento foram mudanças na Rede Neural Multi-Camadas mais especificamente nas camadas escondidas e neurônios escondidos.
- No terceiro experimento foram mudanças nas recompensas por adquirir as pastilhas e super pastilhas.
- No quarto experimento foram mudanças nas recompensas por eliminar um fantasma.

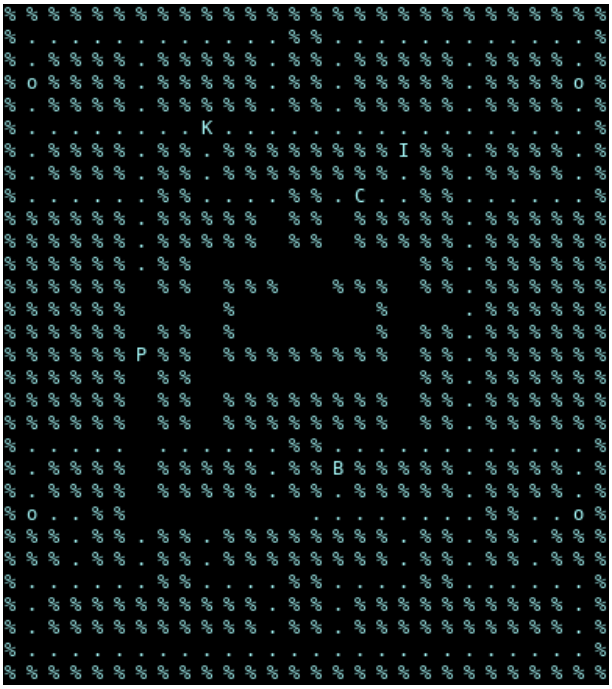

Figura 5: P: Pac-Man, B: fantasma Blinky, K: fantasma Pinky, I: fantasma Inky, C: fantasma Clyde, ' ': espaço livre, %: parede, .: pastilha, o: super pastilha.

O jogo do Pac-Man é uma simulação feita no console que tenta ser o mais próximo possível do jogo original, as representações visuais estão presentes no estado do labirinto e são representadas por caracteres, como pode ser visto na Figura 5. A implementação deste trabalho está disponível no github: https://github.com/emanuelssj/Lunafreya-Pacman-Speedrun.

5.3 Resultados e Discussão

No primeiro experimento foram testados e avaliados os dois parâmetros do *Q-Learning*: o fator de aprendizagem α e o fator de regularização γ. Esses foram avaliados com base no sistema de recompensas que pode ser visto na Tabela 4. Foram feitas diferentes rodadas de treinos e testes da rede neural, onde cada uma dessas rodadas possui 5000 jogos que duram entre 40 minutos. As recompensas foram fixas, mas os parâmetros citados foram testados com diferentes valores.

O fator de aprendizagem foi variado entre 0,1 a 0,4 e o fator de regularização de 0,0000001 a 0,0000009. Esses valores foram escolhidos em torno dessa faixa de variação porque essa faixa que garantiu os resultados mais consistentes para o modo de jogo *speedrun* do Pac-Man. Nos testes, foi avaliado com os parâmetros citados acima se o Pac-Man conseguiria uma pontuação de 10000 pontos no menor tempo possível.

Nas Figuras 6 e 7, é mostrada a influência dos fatores de aprendizagem e regularização na pontuação de 10000 pontos obtida pelo Pac-Man no período de no máximo 2 minutos. Assim como pode ser visto na Figura 6, quanto menor o fator de aprendizagem, maior a precisão do treinamento com a rede neural, bem como a pontuação máxima obtida pelo Pac-Man com destaque para o 0,0000001 que conseguiu com 834 rounds em torno de 1 minuto e 1 segundo. Os resultados obtidos na Figura 7 mostram que quanto maior o fator de

regularização, melhor o desempenho do Pac-Man com destaque para o 0.7 que conseguiu em 635 rounds em torno de 47 segundos.

No segundo experimento foram testados e avaliados os parâmetros da Rede Neural Multi-Camadas: Camadas e neurônios escondidos. Esses também como no primeiro experimento foram avaliados com base no sistema de recompensas que pode ser visto na Tabela 4. Foram feitas diferentes rodadas de treinos e testes da rede neural, onde cada uma dessas rodadas possuem 5000 jogos que duram entre 40 a 60 minutos. As recompensas foram fixas, mas os parâmetros citados foram testados com diferentes valores.

A camada escondida foi variada entre 1 a 4 e o neurônio escondido foi testado com os valores 50, 100, 150 e 200. Esses valores foram escolhidos em torno desses números porque com valores muito baixos as derrotas eram constantes apesar disso essa faixa garantiu alguns resultados favoráveis para o modo de jogo *speedrun* do Pac-Man. Nos testes, foi avaliado com os parâmetros citados acima se o Pac-Man conseguiria uma pontuação de 10000 pontos no menor tempo possível.

Nas Figuras 8 e 9, é mostrada a influência da camada escondida e neurônio escondido na pontuação de 10000 pontos obtida pelo Pac-Man no período de no máximo 2 minutos. Assim como pode ser visto na Figura 8, apenas com 2 e 3 camadas que foi possível chegar no objetivo, com destaque para 2 que foi em 640 *rounds* em torno de 48 segundos. Já na Figura 9 é possível ver que todos os valores testados conseguiram chegar no objetivo com destaque para o 200 que foi em 749 *rounds* em torno de 56 segundos.

No terceiro experimento foram testados e avaliados os parâmetros de recompensa por: pastilha e super pastilha. Esse diferente dos dois experimentos anteriores, a tabela 4 teve mudanças nas recompensas por adquirir uma super pastilha e uma pastilha, as outras recompensas ficaram fixas. Foram feitas diferentes rodadas de treinos e testes da rede neural, onde cada uma dessas rodadas possuem 5000 jogos que duram entre 40 minutos.

Na recompensa por pastilha foi testado os valores 0, 5, 10, 15, 20, 25, 30 e 40 e a recompensa por super pastilha foi testados os valores 150, 200, 250, 300, 350, 400, 450 e 500. Esses valores foram escolhidos em torno dessa faixa de variação porque tiveram resultados melhores para o *speedrun* e também são valores que se encaixam melhor no ambiente de recompensas que foi mostrado na tabela 4, pois com valores altos não teve bons resultados. Nos testes, foi avaliado com os parâmetros citados acima se o Pac-Man conseguiria uma pontuação de 10000 pontos no menor tempo possível.

Nas Figuras 10 e 11, é mostrada a influência das recompensas por pastilha e super pastilha na pontuação de 10000 pontos obtida pelo Pac-Man no período de no máximo 2 minutos. Assim como pode ser visto na Figura 10 e na Figura 11 praticamente todos os valores chegaram no objetivo com destaque no caso da recompensa por pastilha para o 20 que conseguiu em 584 rounds em torno de 43 segundos e no caso da recompensa por super pastilha para o 400 que conseguiu em 599 rounds em torno de 45 segundos.

No último e quarto experimento foram testados e avaliados os parâmetros de recompensa por: eliminar um fantasma. Esse igual o terceiro experimento, a tabela 4 só teve mudanças nas recompensas por eliminar um fantasma e combos, as outras recompensas ficaram fixas. Foram feitas diferentes rodadas de treinos e testes da rede neural, onde cada uma dessas rodadas possuem 5000 jogos que duram entre 40 minutos.

Na recompensa por eliminar um fantasma foi testado os valores 500, 1000, 1500, 2000, 2500, 3000, 3500 e 4000. Esses valores foram escolhidos em torno dessa faixa de variação porque tiveram resultados melhores para o *speedrun* e também são valores que se encaixam melhor no ambiente de recompensas que foi mostrado na tabela 4, pois como no *speedrun* do Pac-Man eliminar fantasmas é obrigatório para conseguir chegar aos 10000 pontos, valores baixos tiveram apenas péssimos resultados e com valores muito altos tiveram resultados medianos. Nos testes, foi avaliado com os parâmetros citados acima se o Pac-Man conseguiria uma pontuação de 10000 pontos no menor tempo possível.

Na Figura 12, é mostrada a influência das recompensas por eliminar um fantasma na pontuação de 10000 pontos obtida pelo Pac-Man no período de no máximo 2 minutos. Quase todos os valores chegaram no objetivo com destaque para o 3500 que conseguiu em 959 rounds em torno de 1 minuto e 11 segundos.

6. CONCLUSÃO

Neste trabalho, foi apresentada uma avaliação do tradicional algoritmo de *Q-Learning* no contexto do jogo Pac-Man, jogado no modo *speedrun*.

No primeiro experimento os resultados com valores altos no fator de aprendizagem não tiveram bons resultados e não eram consistentes, por isso, os valores testados foram baixos. O problema é que com valores próximo de 1 o Pac-Man tem o foco em explorar o ambiente, o que não é o foco do modo de jogo *speedrun*, e com valores próximo de 0 o foco é em completar os objetivos o que faz com que todo o tempo gasto seja útil em relação ao *speedrun*. O fator de regularização foi possível testar em diversos valores, mas os melhores resultados foram com valores altos, isso porque com valores próximo de 1 o foco é em recompensas futuras e no Pac-Man juntar os fantasmas antes de adquirir a super pastilha é necessário para garantir um combo X4, com isso, a pontuação será bem maior garantindo assim um melhor desempenho para o *speedrun* na categoria de 10000 pontos. Com valores próximo de 0 o foco é em recompensas imediatas o que no Pac-Man não vai garantir altas pontuações rapidamente, pois eliminar os fantasmas a partir de combos é a única forma de garantir 10000 pontos.

No segundo experimento só foi possível testar com valores baixos, pois ao testar com valores altos a máquina usada para os experimentos parava de funcionar ou os experimentos demoravam muito tempo para completar.

No terceiro experimento não se esperava grandes resultados, seria usado como uma análise para o próximo, neste experimento seria escolhido os melhores valores para testar junto com o quarto experimento mas, foi o que garantiu o melhor resultado do trabalho.

No quarto experimento foi onde se esperava os melhores resultados e foi possível chegar aos 10000 pontos mas, precisou de muito tempo para chegar nessa pontuação o que para o *speedrun* não é nada bom.

Foi verificado a partir dos experimentos realizados que com um algoritmo de *Reinforcement Learning* é possível o Pac-Man não só jogar, mas também realizar *speedrun*, pois os resultados conseguiram confirmar a hipótese criada por esse trabalho que foi chegar a resultados iguais ou melhores dos 3 primeiros colocados dos competidores de *speedrun* no jogo Pac-Man, no caso do primeiro colocado seu tempo foi

de 43 segundos e o resultado obtido neste trabalho foi de 43 segundos no terceiro experimento.

Como trabalhos futuros, objetiva-se garantir uma consistência nos resultados de *speedrun* da inteligência artificial no Pac-Man, ou seja, aumentar o número de vitórias para então garantir melhores resultados constantes, a avaliação de outras técnicas como o *deep Q-Learning* que possa garantir melhores resultados ou a consistência nas vitórias e além disso, almeja-se a realização de testes com outras versões do Pac-Man, como o Ms. Pac-Man ou uma versão sem as personalidades dos fantasmas para então checar a diferença nos resultados.

7. REFERÊNCIAS

[1] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, July 1959.

[2] R. Haridy. The year AI beat us at all our own games. https://newatlas.com/ai-2017-beating-humans-games/52741/. Acesso em: 2018-03-29.

[3] G. Narula. Machine Learning in Gaming – Building AIs to Conquer Virtual Worlds. https://www.techemergence.com/machine-learning-in-gaming-building-ais-to-conquer-virtual-worlds/. Acesso em: 2018-03-29.

[4] D. B. Ortega. Machine Learning Applied to Pac-Man Final Report. https://upcommons.upc.edu/bitstream/handle/2099.1/26448/108745.pdf. Acesso em: 2018-06-30.

[5] P. Chase. Pac-Man Leaderboard. https://www.speedrun.com/Pac-Man. Acesso em: 2018-03-29.

[6] M. do Computador. História do Ms. Pac-Man.. http://www.museudocomputador.org.br/historia-pacman. Acesso em: 2018-07-15.

[7] C. Birch. Understanding Pac-Man Ghost Behavior. http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior. Acesso em: 2018-08-20.

[8] R. Scully-Blaker. (2014, 01) A practiced practice: Speedrunning through space with de certeau and virilio. http://gamestudies.org/1401/articles/scullyblaker. Acesso em: 2018-04-25.

[9] M. Uyama. Speed Demos Archive. http://speeddemosarchive.com/. Acesso em: 2018-02-23.

[10] ——. Games Done Quick. https://gamesdonequick.com/. Acesso em: 2018-02-23.

[11] B. Bertoli. How To Start Speedrunning Video Games. https://kotaku.com/how-to-start-speedrunning-video-games-1796984207. Acesso em: 2018-07-21.

[12] T. Matiisen. Demystifying deep reinforcement learning. http://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/. Acesso em: 2018-07-17.

[13] Adelikat. TAS(Tool Assisted Speedrun). http://tasvideos.org. Acesso em: 2018-07-21.

[14] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[15] R. v. LOON. Machine learning explained: Understanding supervised, unsupervised, and reinforcement learning. http://bigdata-madesimple.com/machine-learning-explained-understanding-supervised-unsupervised-and-reinforcement-learning/. Acesso em: 2018-03-12.

[16] P. Dayan and Y. Niv, "Reinforcement learning: The Good, The Bad and The Ugly," vol. 18, pp. 185–96, 09 2008.

[17] M. Ashraf. Reinforcement Learning Demystified: A Gentle Introduction. https://towardsdatascience.com/reinforcement-learning-demystified-36c39c11ec14. Acesso em: 2018-07-25.

[18] T. v. d. Ouderaa. Deep Reinforcement Learning in Pac-man. https://esc.fnwi.uva.nl/thesis/centraal/files/f323981448.pdf. Acesso em: 2018-07-18.

[19] U. P. de Catalunya. PACMAN and reinforcement learning. http://www.lsi.upc.edu/~bejar/apren/docum/trans/pacman-eng.pdf. Acesso em: 2018-07-17.

[20] L. Bom, R. Henken, and M. Wiering, "Reinforcement learning to train ms. pac-man using higher-order action-relative inputs," in *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, April 2013, pp. 156–163.

[21] G. Lample and D. S. Chaplot, "Playing FPS Games with Deep Reinforcement Learning," in *AAAI Conference on Artificial Intelligence*, 2017.

[22] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[23] J. McCullock. Mnemosyne studio - q-learning. http://mnemstudio.org/path-finding-q-learning.htm. Acesso em: 2018-07-25.

[24] M. Alzantot. Deep Reinforcement Learning Demystified (Episode 0). https://medium.com/@m.alzantot/deep-reinforcement-learning-demystified-episode-0-2198c05a6124. Acesso em: 2018-07-25.

[25] O. S. University. PAC MAN as Markov Decision Process. . https://oregonstate.instructure.com/files/67193237/download?download_frd=1. Acesso em: 2018-07-25.

[26] K. V. P. J. Svozil, Daniel. Introduction to multi-layer feed-forward neural networks. https://pdfs.semanticscholar.org/09cf/727949c915ae8fd8b0a2ead55bcafedc0a12.pdf. Acesso em: 2018-08-15.

[27] A. Oliveira, A. Souza, W. Lacerda, and L. Resende Gonçalves, "Aplicação de redes neurais artificiais na previsão da produção de álcool," *Ciencia E Agrotecnologia - CIENC AGROTEC*, vol. 34, 04 2010.

[28] Saurabh. Backpropagation – Algorithm For Training A Neural Network. https://www.edureka.co/blog/backpropagation/. Acesso em: 2018-09-17.

[29] Langbroek, David, Aurucci, Davide, Roca, A. Albert Segarra, and Fthi. Learning to play Pac-Man with Reinforcement Learning. https://github.com/albertsgrc/q-mspacman. Acesso em: 2018-06-15.

[30] F. Estévez, A. Ruiz, and P. Martin. Training Pac-Man bots using Reinforcement Learning and Case-based Reasoning. http://ceur-ws.org/Vol-1957/CoSeCiVi17_paper_14.pdf. Acesso em: 2018-07-25.

[31] R. Niel, J. Krebbers, M. Drugan, and M. Wiering, "Hierarchical reinforcement learning for real-time strategy games," in *International Conference on Agents and Artificial Intelligence*, 01 2018.

[32] N. Tziortziotis, K. Tziortziotis, and K. Blekas, "Play ms. pac-man using an advanced reinforcement learning agent," in *Artificial Intelligence: Methods and Applications*, A. Likas, K. Blekas, and D. Kalles, Eds. Cham: Springer International Publishing, 2014, pp. 71–83.

8. ANEXOS

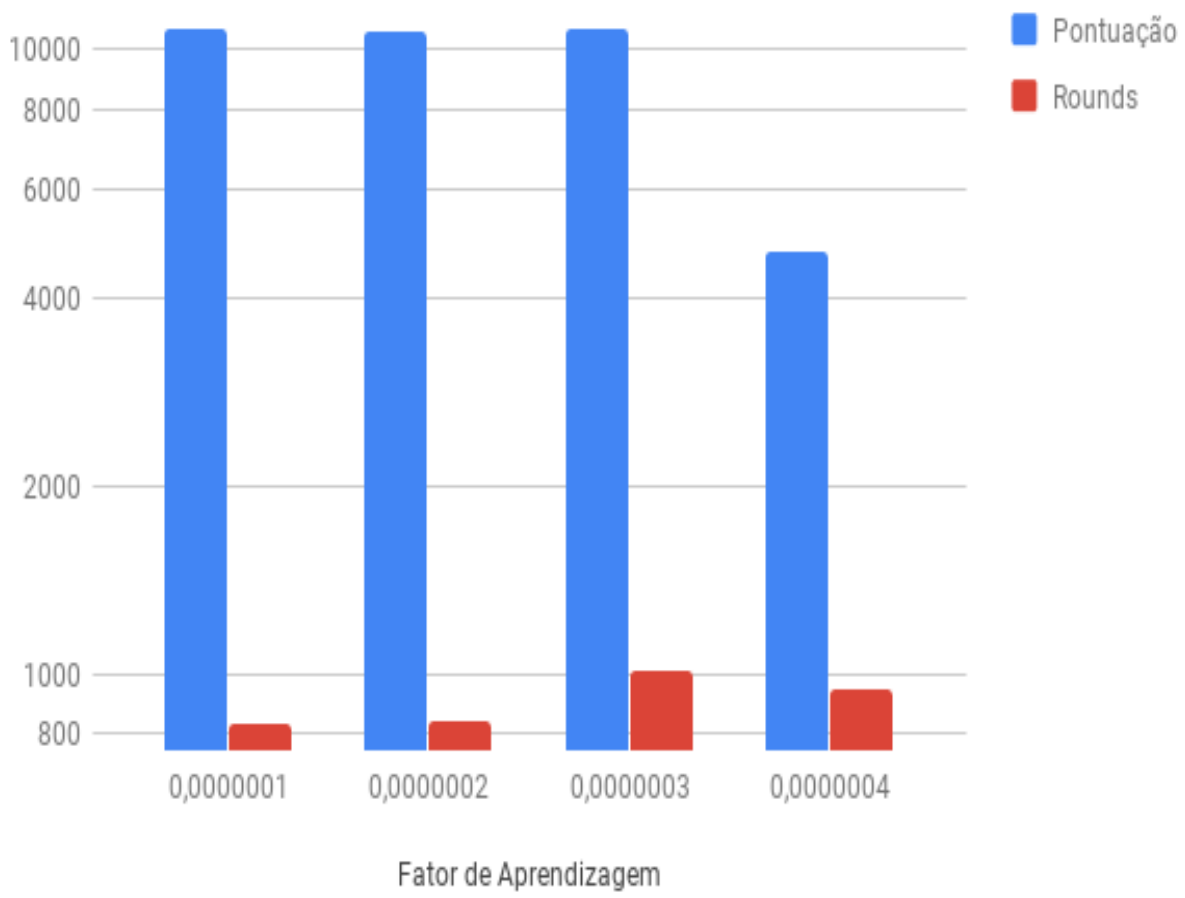

Figura 6: Experimento 1 variação do Fator de Aprendizagem
- Obs: 13,5 Rounds = 1 Segundo

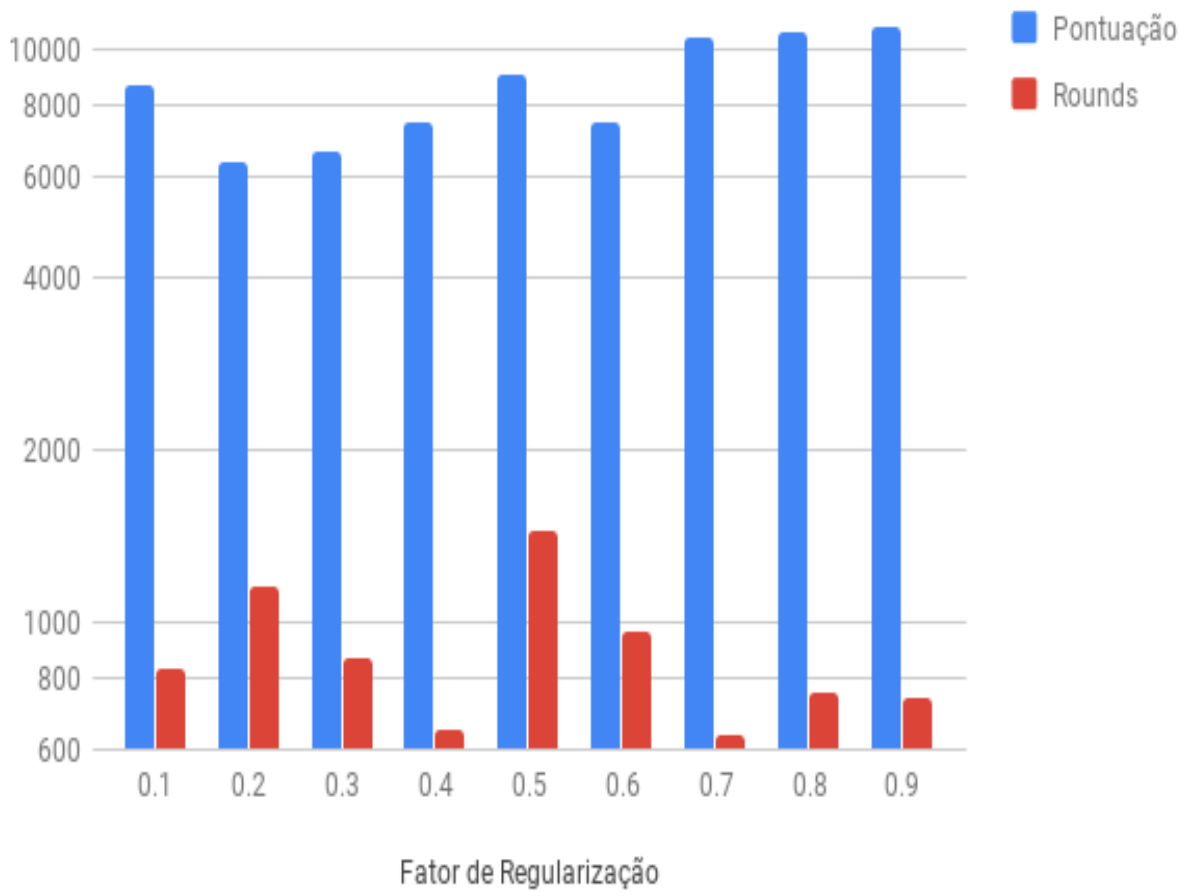

Figura 7: Experimento 1 variação do Fator de Regularização

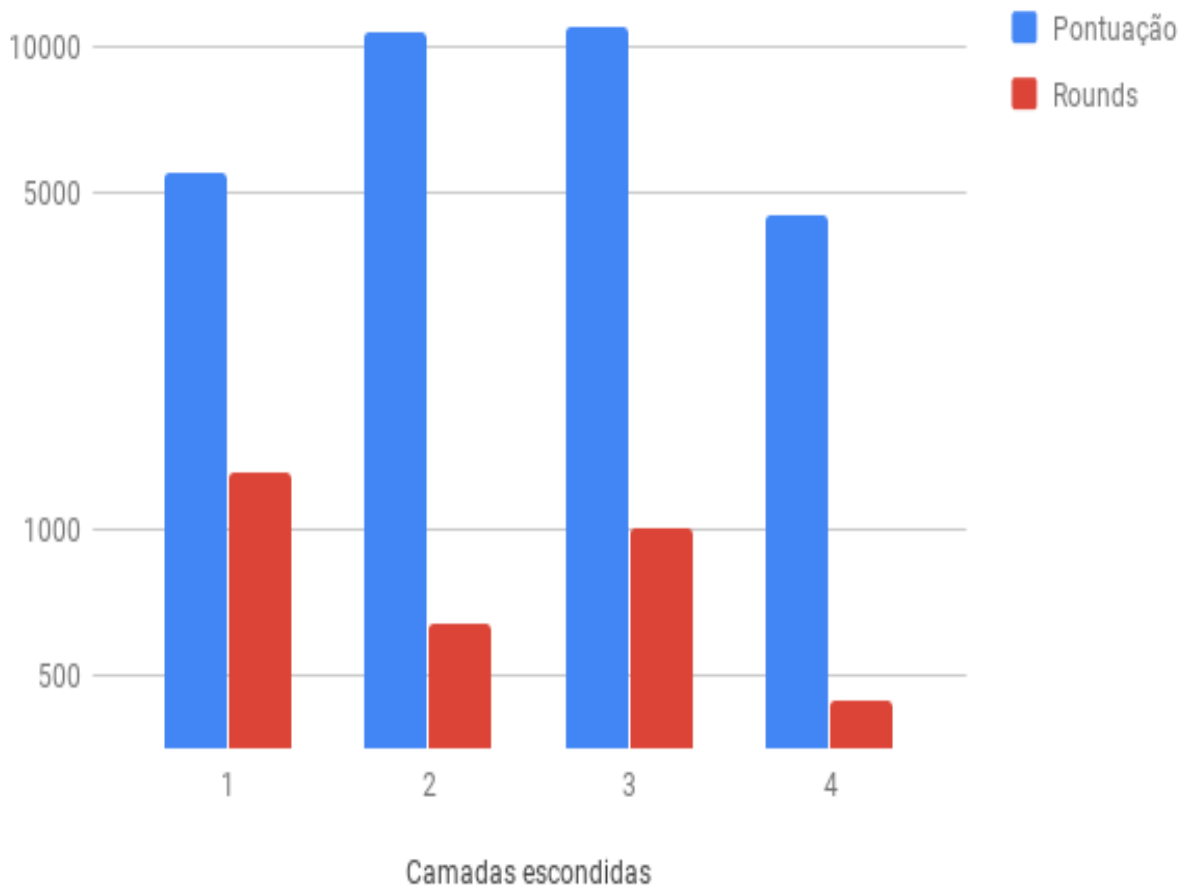

Figura 8: Experimento 2 variação da Camada Escondida

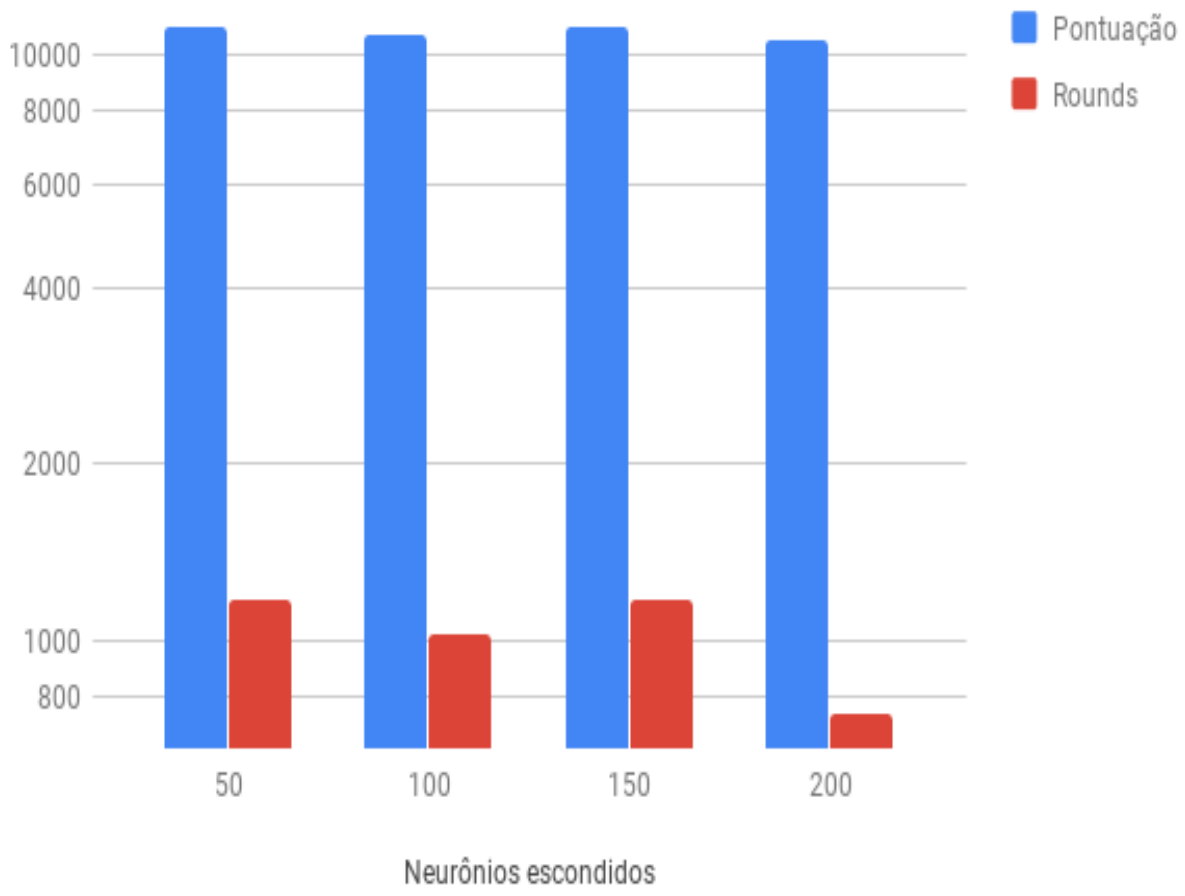

Figura 9: Experimento 2 variação do Neurônio Escondido

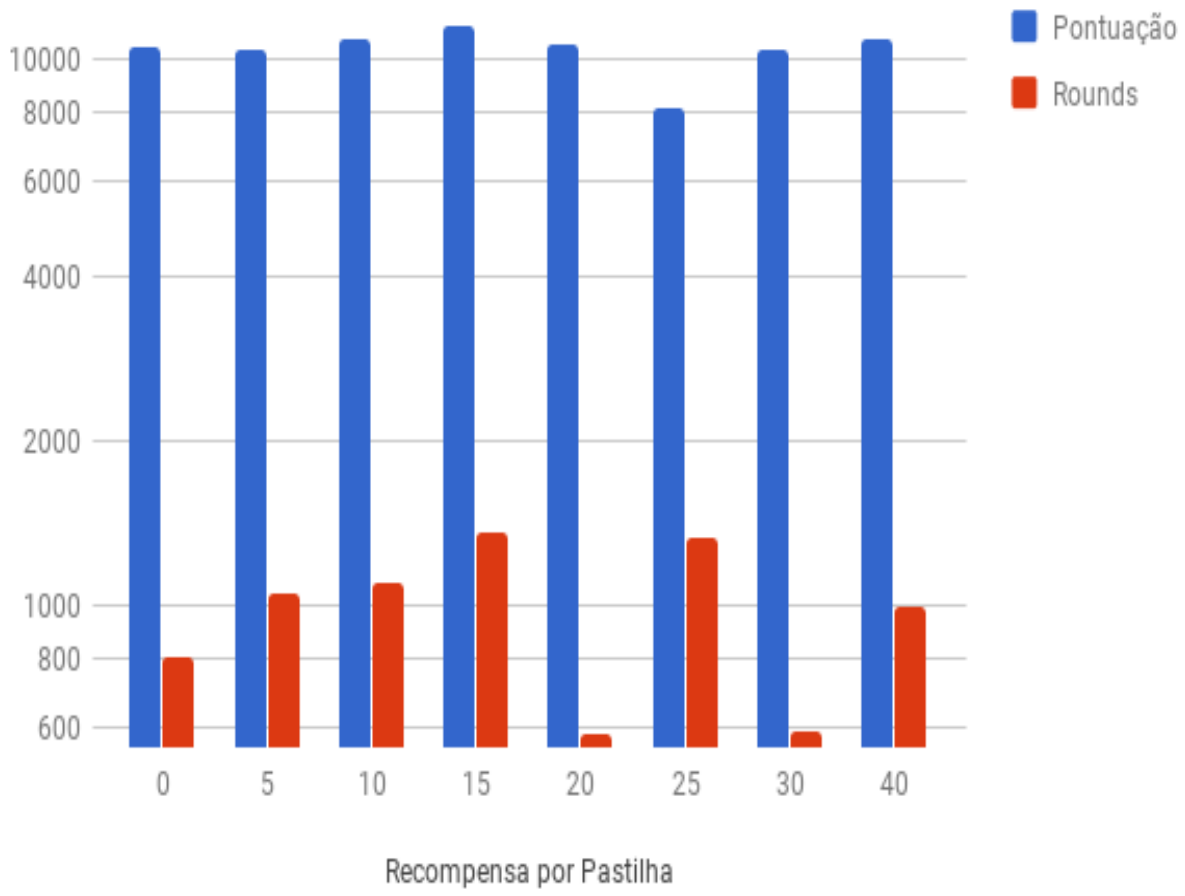

Figura 10: Experimento 3 variação da Recompensa por Pastilha

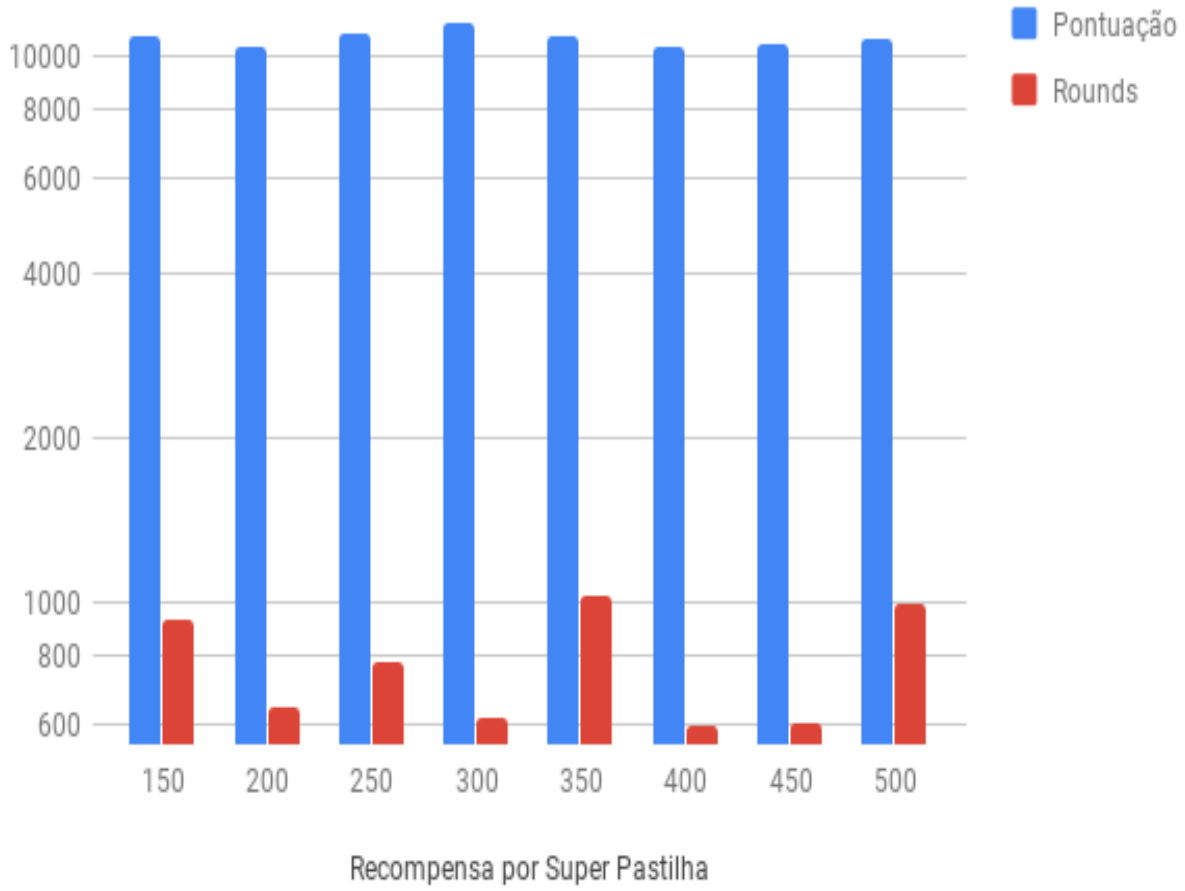

Figura 11: Experimento 3 variação da Recompensa por Super Pastilha

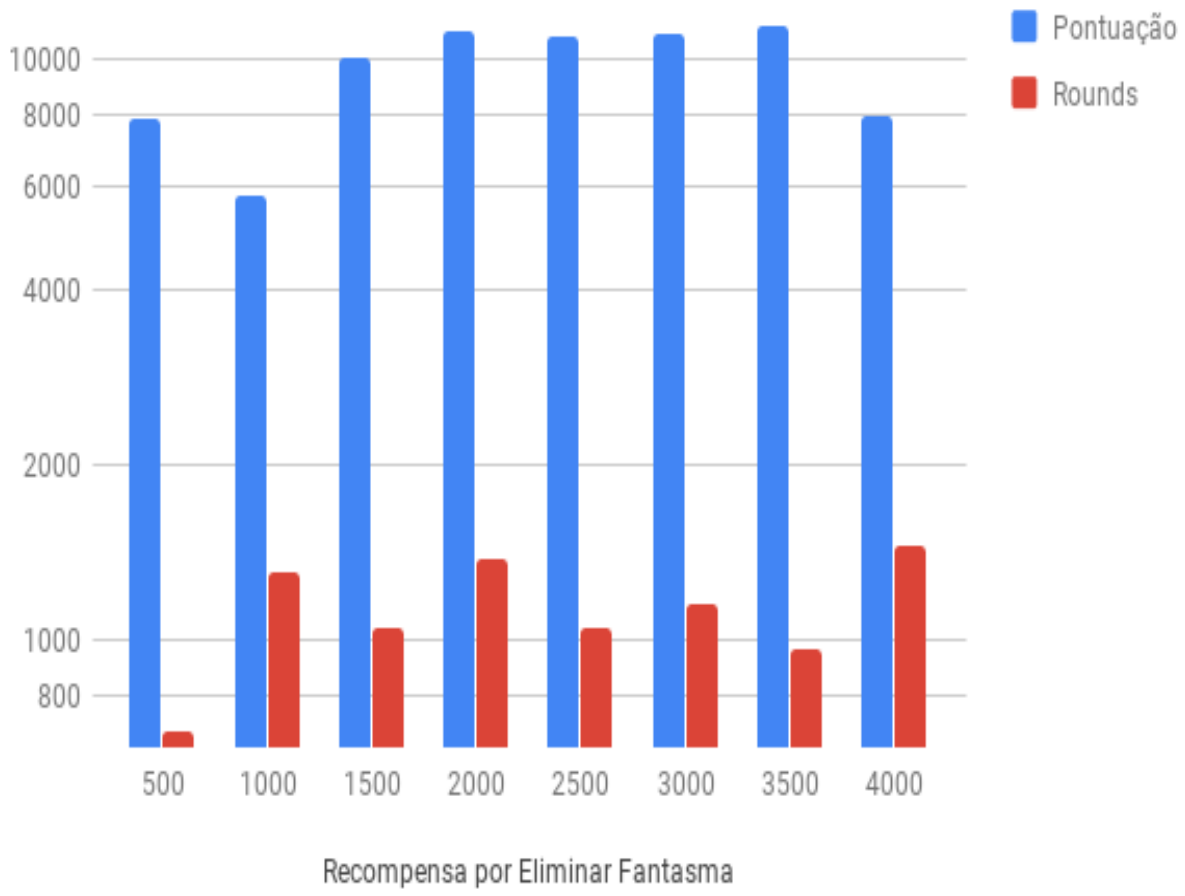

Figura 12: Experimento 4 variação da Recompensa por Eliminar Fantasma