Aplicação de Aprendizado por Reforço no Jogo Pommerman

Gabriel Farias Cação[1], **Anderson Viçoso de Araujo**[1]

[1] FACOM - Faculdade de Computação
Universidade Federal do Mato Grosso do Sul (UFMS) – CAMPO GRANDE, MS – Brasil

gabrielfc1995@gmail.com

Abstract. *In the context of machine learning, various types of learning are used to solve problems in a variety of areas. In this sense, this article presents the project of the implementation of a reinforcement learning agent for a strategy board game called Pommerman, with use of a advantageous model for the developer. In this case, it was noticed a low efficiency of the model adopted and the need for more trainings to improve the results.*

Resumo. *No contexto de aprendizado de maquina, vários tipos de aprendizados são utilizados para resolver problemas nas mais diversas áreas. Neste sentido, este artigo apresenta o projeto da implementação de um agente de aprendizado por reforço para o jogo de estratégia em tabuleiro chamado Pommerman, com a utilização de um modelo vantajoso para o desenvolvedor. Neste caso notou-se pouca eficiência do modelo adotado e a necessidade de mais treinamento para melhoria dos resultados.*

1. Introdução

Ao pensar na natureza da aprendizagem humana, depara-se com a capacidade de se adaptar ao meio ambiente e aprender com ele. Quando um bebê nasce, ele não sabe o que fazer para dizer "estou com fome" e ele não possui um professor explícito para ensiná-lo, mas tem uma conexão sensório-motora com o seu ambiente, e com o tempo descobre que chorar é uma boa alternativa. Ao longo de nossas vidas, tais interações com o ambiente são indubitavelmente uma fonte de conhecimento importante. Aprender da interação é uma ideia fundamental relevante em quase todas as teorias de aprendizagem [Sutton 2018].

Este artigo lida com a uma aplicação de aprendizagem por reforço, que consiste em projetar ações através de leituras constantes de interações com o ambiente através da maximização de uma recompensa dada para cada ação executada. O agente não é informado sobre as ações a serem tomadas, mas deve descobrir a que tem melhor retorno para si experimentando. Os casos mais interessantes e desafiadores envolvem ações que podem afetar não só a recompensa imediata, mas também a próxima situação, sua recompensa e assim subsequentemente.

Dentro do contexto de aprendizado natural, procura-se um modelo para simular tarefas com variação infinitamente significativa, o que é comum no mundo real, afim de trata-las utilizando a classe de métodos de solução de aprendizado por reforço. Neste contexto, utiliza-se como ambiente de aprendizado o jogo *Pommerman*, uma versão *open source* do aclamado jogo *Bomberman*[Bom 2018]. Este consiste em um jogo de estratégia simulado em tabuleiro, na qual jogadores ou equipes de jogadores movimentam-se pelo

tabuleiro com objetivo de eliminar seus adversários através de explosão de bombas que são posicionadas como uma das opções de ação. Este artigo propõe-se a descrever uma metodologia de aprendizado por reforço aplicado dentro deste jogo.

2. Objetivos

O objetivo deste trabalho consiste em elaborar um modelo de agente para atuar no jogo *Pommerman*, apenas de sua iteração com o ambiente, utilizando de uma estratégia de aprendizagem por reforço. Cada agente implementado compartilhara seu aprendizado individual com sua equipe, constituída de pelo menos um outro agente de mesma implementação. Além disso, o trabalho visa a implementação de um contexto de mapeamento de estados visível para o desenvolvedor, de modo a possibilitar a interpretação da Tabela estados-ações pelo especialista. Assim podemos sumarizar as seguintes etapas do projeto:

- Modelar uma estratégia de aprendizado por reforço no contexto do jogo *Pommerman* a ser utilizada em multiagentes;
- Implementar o modelo de agente de aprendizado para este ambiente;
- Coletar dados do aprendizado e realizar distribuição desses dados para reforço dentro da equipe;
- Analisar os dados e avaliar a eficiência do agente e da estratégia como um todo.

3. Justificativa

Nos últimos anos, o interesse por *machine learning* cresceu em todas as áreas do conhecimento pela versatilidade de aplicações nas quais ela pode ser inserida. Neste sentido, o aprendizado por reforço surge como uma das possibilidades de implementação que valhese da interação dos agentes com o ambiente e entre si para definir uma recompensa que irá orientar a próxima ação. No contexto do jogo *Pommerman*, a possibilidade de aprender sem a necessidade de um *dataset* de treinamento torna o aprendizado por reforço uma opção bastante interessante no contexto do jogo.

Destá forma, o trabalho justifica-se pela possibilidade de aplicação de um tipo de aprendizado que apesar de não ser o mais utilizado neste contexto, quando comparado a técnicas *single player* de aprendizado supervisionado, possui vantagens a serem exploradas, tais como melhor desempenho quando não se tem uma grande base de dados para treinamento, o que no caso do aprendizado por reforço, possibilita a criação de modelos mais eficientes. Podendo ser estendido para outras aplicações no qual as restrições quanto a base de dados são um dos fatores limitantes. Também possui a vantagem de explorar possibilidades não necessariamente pré-registradas o que proporciona dinamicidade ao algoritmo, possibilitando a descoberta de estrategias nunca registradas anteriormente.

Ainda pode-se observar que realizar tarefas com variação infinitamente significativa é comum no mundo real e difícil de simular. O aprendizado multiagente competitivo permite isso, e o *Pommerman* se trata de um jogo criado para fins de pesquisa nestá área [Cinjon Resnick 2017].

4. Pommerman

Pommerman é uma plataforma estilisticamente semelhante ao *Bomberman*. Cada batalha começa em uma arena 11x11 simétrica desenhada aleatoriamente, como pode ser observado na Figura 1. Existem quatro agentes, um em cada canto da tela. Além dos agentes, o tabuleiro contém paredes de madeira e paredes rígidas geradas aleatoriamente de forma a garantir que os agentes terão um caminho acessível ao outro. Paredes rígidas são indestrutíveis e intransitáveis. Paredes de madeira podem ser destruídas por bombas. Até que sejam destruídos, eles são intransitáveis. Depois que eles são destruídos, eles se tornam uma passagem ou um item de efeito, podendo ser um aumento de bomba, aumento na quantidade de munição, habilidade de chute e outros.

Figura 1. Renderização do Pommerman, nove passos Após o inicio do jogo.

As principais diferenças que podem ser notadas no *Pommerman* quando comparado ao *Bomberman* são: O tabuleiro com dimensões quatro unidades maiores para cada lado e a forma randômica como ele é gerado, no *Bomberman* o tabuleiro sempre possui as paredes rígidas na mesma posição, o que diminui significativamente o número de variações do tabuleiro.

A plataforma *Pommerman* disponibiliza a implementação inicial do jogo, onde todo *Pomme* (ator do jogo) é controlado por um agente que recebe uma observação do estado do jogo. Todo agente começa com a habilidade de instalar uma única bomba por vez. A bomba tem uma força explosiva que começa com poder de três unidades de espaco. A força explosiva determina o quão longe nas direções vertical e horizontal que a bomba irá. Uma bomba tem uma vida de 10 etapas de tempo. Depois que sua vida expira, ela explode e quaisquer paredes de madeira, agentes, itens de efeito ou outras bombas em seu alcance (dadas pela força da explosão) são destruídas. No caso de outras bombas serem atingidas, as mesmas explodem simultaneamente, mesmo que suas etapas de vida não tenham atingido o valor dez [Cinjon Resnick 2017].

Metade das paredes de madeira escondem um item de efeito que são revelados quando a parede é destruída, que ao serem coletados proveem efeitos:

- Bomba Extra: Aumenta a munição do agente em um;

- Pode chutar: Permite que um agente chute bombas. A bomba então viaja na direção em que o agente estava se movendo a uma velocidade de uma unidade por degrau de tempo até que eles sejam impedidos por um jogador, uma bomba ou uma parede;
- Aumentar Alcance: Aumenta a força de detonação do agente em um;
- Caveira: Traz um efeito que pode ser ruim ou bom, como inverter os controles de direção, ou tornar a força explosiva infinita, entre outros.

As observações recebidas pelos agentes estão dispostas em um dicionário que possui os seguintes valores fornecidos pela plataforma [Cinjon Resnick 2017]:

- *Board*: Uma matriz de inteiros com 121 posições que representa o tabuleiro achatado. Todos os quadrados fora do alcance do agente serão cobertos com o valor de neblina (5), que representa o que o agente não consegue visualizar, está representado do lado direito da Figura 1;
- *Position*: Uma tupla que representa a posição do agente (x, y) na grade, onde x, y varia de 0 a 10.;
- *Ammo*: Um inteiro que representa a munição do agente;
- *Blast Strength*: Um inteiro que representa a força de explosão da bomba do agente;
- *Can Kick*: Um inteiro que representa se o agente pode ou não chutar bombas;
- *Teammate*: Um inteiro que represente quem é o companheiro de equipe, podendo variar de -1 a 3, caso a execução não esteja em modo de equipe o valor assume -1;
- *Enemies*: Três inteiros que representam quais são os adversários deste agente. Se esta for uma competição de equipes o último valor será -1 para indicar que só há dois inimigos;
- *Bombs*: Uma lista de inteiros contendo as bombas no alcance do agente, especificadas por (x, y, *BlastStrength*), sendo x e y coordenadas da grade.

Os agentes possuem 6 ações possíveis: Podem se mover para qualquer direção, uma casa da arena por vez, ficar parado ou colocar bomba.

O jogo possui 3 modos: *FFA, Team, TeamRadio*. Sendo respectivamente um modo *single player*, um cooperativo de duas equipes com observações com neblina e por último outro modo cooperativo, porém diferenciado pela comunicação entre as equipes, que pode possuir até duas palavras em um dicionário de tamanho oito a ser transmitido para o parceiro de equipe.

O problema proposto pela plataforma consiste em treinar um agente para que vença os demais, utilizando o aprendizado por reforço, onde o agente iniciaria seu treino sabendo somente as ações que pode tomar, sem saber o resultado das mesmas. Após um determinado número de interações estará apto para teste, pois terá aprendido as mecânicas de jogo. A plataforma deseja impulsionar a pesquisa através de *benchmarks* fortes testando diferentes facetas da inteligência. O *Multi-Agent Learning* não tem um, e a mesma visa corrigir isso.

5. Aprendizado por Reforço

O aprendizado por reforço pode ser considerado um paradigma do aprendizado de máquina, assim como o aprendizado supervisionado e o não supervisionado. Em linhas gerais, este busca orientar as ações de um agente de aprendizado através da maximização de uma função de recompensa que é percebida pela interação com o ambiente e pela avaliação de multi-estados. Em casos mais interessantes e desafiadores, as ações podem afetar não apenas as recompensa, mas também a próxima situação e, através disso, todas as recompensas subsequentes. Para obter um ganho de aprendizado, o agente deve escolher opções que ele já experimentou no passado e que se revelaram vantajosas no ganho de recompensas, mas para descobrir tais ações ele deve experimentar caminhos que ele não selecionou antes. O dilema é que nem um nem outro caminho pode ser seguido exclusivamente sem falhar na tarefa. Isto torna este tipo de aprendizado uma tarefa estocástica, na qual cada ação deve ser tentada diversas vezes para obter uma estimativa confiável de sua recompensa esperada.

está modelagem de aprendizado possui duas características, muito importantes, que o distinguem de qualquer outra: Pesquisa por tentativa e erro e recompensa atrasada. O aprendizado por reforço assume uma tática inversa, começando com um busca completa, interativa e de buscas de objetivos do agente. Além disso, todo agente tem metas explícitas, mas devem agir mesmo com incertezas. Ele é simultaneamente um problema, uma classe de métodos de solução que funcionam bem no problema, e o campo que estuda este problema e seus métodos. Um agente de aprendizagem deve ser capaz de sentir o estado do seu ambiente até certo ponto e deve ser capaz de realizar ações que afetem o estado.

Um dos desafios que surgem no aprendizado por reforço, é o *trade-off* entre *exploration e exploitation*. Para obter uma recompensa, um agente de aprendizagem de reforço deve preferir ações que tentou no passado e descobriu-se ser eficaz na produção de recompensa. Mas para descobrir tais ações, ele tem que tentar ações que ele não selecionou antes. O agente tem que prospectar o que ele já possui, a fim de obter recompensas boas, mas também tem que explorar novas ações e estados a fim de fazer melhor seleções de ação no futuro, conhecendo todas as possibilidades de ação para o estado e suas recompensas. O paradigma neste caso consiste que ambas *exploration e exploitation* não podem ser alcançados exclusivamente, sem falhar na execução da tarefa. O agente deve tentar uma variedade de ações e progressivamente a favorecer aqueles que parecem ser os melhores. Em uma tarefa estocástica, cada ação deve ser tentada muitas vezes para obter uma estimativa confiável de sua recompensa esperada. O dilema *exploration e exploitation* tem sido intensamente estudada por matemáticos por muitas décadas, ainda permanece sem solução [Sutton 2018].

Para lidar com este *trade-off* a exploração $\epsilon - Greedy$ é uma das mais utilizadas e métodos mais simples que realiza a *exploration e exploitation*. Ele usa o parâmetro ϵ para determinar a porcentagem das ações é selecionada aleatoriamente. o parâmetro cai no intervalo $0 \leq \epsilon \leq 1$, onde 0 refere-se a não há exploração e 1 há total exploração. A ação com o maior valor Q é escolhida com probabilidade 1 - ϵ e uma ação aleatória é selecionada de outra forma. O $\epsilon - Greedy$ resolve este *trade-off* explorando os efeitos das diferenças entre as ações.

O Processo de Decisão de Markov (MDP) é um modelo para problemas de tomada

de decisão sequencial totalmente observáveis em ambientes estocásticos.

- S é um conjunto finito de estados, onde $s^t \in S$ é o estado na etapa temporal t;
- A é um conjunto finito de ações, onde $a^t \in A$ é a ação executada no passo t;
- A função de recompensa Recompensa(s, a, s_0) denota a recompensa esperada ao transitar do estado s para o estado s_0 após a execução da ação a;
- A recompensa no passo de tempo t é denotada com r^t;
- O fator de desconto $\gamma \in [0, 1]$ atribui uma importância menor às futuras recompensas para a tomada de decisão ideal.

O treinamento do agente consiste em determinar uma política ótima com o um mapeamento entre estados e ações. Para aprendizagem da política ideal de um agente é usado *Q-learning* ([Joseph Groot Kormelink 2018]). Para cada ação em um estado um Q-valor $Q(s, a)$ denota a soma esperada de recompensas obtidas após a execução da ação. *Q-learning* atualiza a função Q usando as informações obtidas depois de selecionar uma ação conforme a equação 1, que popula a *Q-table*, um dicionário que contém as chaves estado e ação e o valor associado a estas chaves.

$$Q(s, a) = r + \gamma(maxQ(s, a)) \tag{1}$$

A representação gráfica do modelo de aprendizagem adotado pode ser observada na Figura 2.

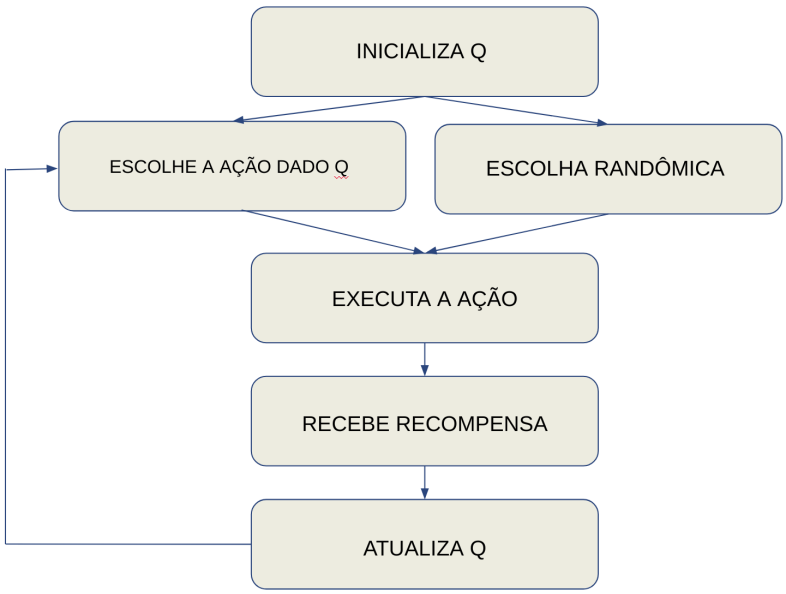

Figura 2. Representação gráfica do modelo de aprendizagem adotado para o agente.

6. Hipóteses

Para a implementação do agente, foram adotadas as seguintes hipóteses:

- O modo de jogo de equipes, onde jogam dois agente contra dois agentes;
- A arena gerada randomicamente para cada episódio do jogo;
- estados gerados a partir das informações disponibilizadas pela plataforma;
- Uma recompensa por cada ação que leva o agente a um novo estado.

A definição dos estados foi realizada a partir da observação disponibilizada pela plataforma, as informações utilizadas para caracterizar os estados podem ser observadas na Tabela 1, e as ações possíveis estão dispostas na Tabela 2:

Tabela 1. Características dos estados.

Características	Possíveis valores
Vivo	1, 0
Bloco de madeira	1, 0
Inimigo próximo (3 casas)	1, 0
número de inimigos vivos	1, 2, 0
Local seguro para por bomba	1, 0
Local seguro para permanecer	1, 0

Tabela 2. Ações e valores.

Ação	Valor
Ficar parado	0
Andar para cima	1
Andar para baixo	2
Andar para esquerda	3
Andar para direita	4
Colocar bomba	5

O número de estados possíveis considera apenas as situações onde há inimigos, por tanto o valor 0 para número de inimigos vivos é descartado já que este representa o fim do jogo. logo temos $2^6 = 64$ possibilidades de estados. Combinado ao número de ações observamos $64 * 6 = 384$ possíveis valores dentro da *Q-table*.

7. Desenvolvimento

Foi elaborado um agente de aprendizado para a plataforma, associado a uma função de recompensa para popular a *Qtable*.

7.1. Agente

O agente foi modelado utilizando da equação 1 da *sessão* 5 para criar a política ideal, com auxílio da utilização do $\epsilon - Greedy$. O jogo é implementado na linguagem de programação *Python* versão 3.6, por este motivo o agente também teve sua implementação na linguagem, o pseudo código deste método é apresentado no algoritmo *Listing* 1. A incorporação do método aos agentes de uma equipe acontece com a utilização simultânea de uma mesma *Qtable* pelos agentes da equipe.

```
#enquanto o jogo nao terminar, make
while not done:
    #renderiza o jogo
    env.render()
    #recebe a acao dos 3 agentes fornecidos pela plataforma
    actions = env.act(state)
    #recebe o mapeamento do estado atual, para o agente 0
    #a partir da observacao fornecida pela plataforma
    stateAtual = QState(state[0])
    actions[0] = actionQ(state[0], EPSILON, dl, stateAtual)
    #game Step
    state, reward, done, info = env.step(actions)
    #se o agente 0 est    vivo, make
    if not die:
        nextState = QState(state[0])
        Qvalue = 1*get_reward(stateString, actions[0], nextState) +
    0.9*MaxQ(nextState, dl)
        att_q_table(stateAtual, actions[0], Qvalue, dl)
```

Listing 1. Pseudo Algoritmo de treinamento do agente para o jogo Pommerman.

7.2. Função de Recompensa

A função de recompensa implementada utiliza a relação existente entre o estado da arena, a ação tomada e o novo estado gerado em consequência da ação, para retornar um valor inteiro que representa uma recompensa positiva ou negativa. Com estes valores o agente é capaz de popular *Qtable*, onde ele poderá verificar qual a melhor ação a ser tomada para cada estado. A recompensa recebida anteriormente será parâmetro de eficiência a ser avaliado como medida de decisão.

Para tornar mais genérica a função de recompensa, agrupamos as ações de movimentos direcionais em um conjunto, assim passamos a ter as possibilidades descritas na Tabela 3. Visto que os movimentos direcionais são escolhidos apenas quando possível ir na direção escolhida e este calculo é feito pela plataforma, para os parâmetros adotados na função de recompensa, basta saber que houve um movimento direcional.

Tabela 3. Ações e valores para recompensas.

Ação	Valor
Ficar parado	0
Movimento	1, 2, 3, 4
Colocar bomba	5

Para mensurar as recompensas foram utilizadas as seguintes políticas:

- Penalidade para morrer = -1000;
- Penalidade por ficar parado em um local não seguro = -100;
- Gratificação por sair de um local não seguro = 100;
- Gratificação por explorar o tabuleiro = 100;
- Gratificação por colocar bomba próximo a inimigos quando não há bomba do inimigo = 500;
- Gratificação por colocar bomba próximo a inimigos quando se está em perigo = 100;
- Gratificação pela redução do número de inimigos = 1000;
- Gratificação por colocar bomba ao lado de blocos de madeira = 50.

Onde identifica-se local não seguro como sendo aquele em que o agente está a ser atingido por uma explosão. Como a plataforma não permite saber quem efetivamente eliminou um adversário, foi adotada a a recompensa por redução no número de inimigos. Nota-se que ficar parado em uma posição não torna o jogo dinâmico e leva ao empate caso o mesmo atinga o número máximo de etapas, então a exploração do mapa também é recompensada.

7.3. *Qtable*

Após um treinamento de 40 mil episódios, algumas métricas foram aplicadas para verificar a efetividade das práticas adotadas para o meio em questão. Observa-se que dentro da hipótese de 64 estados, apenas 16 são visitados, o que condiz com a rotina do jogo, quando observado um número menor de características para definir os estados. Nota-se ainda que dois estado tem o número de acessos como maioria absoluta em relação aos outros, como pode ser observado na Tabela 4.

Tabela 4. estados mais acessados no treino com 40k episódios.

estado	Percentual
Vivo, com 2 inimigos e colocar bomba	42,8%
Vivo, com 2 inimigos, colocar bomba e bloco de madeira ao lado	25,4%
Vivo, em local não seguro, com 2 inimigos	10,9%

Considerando estes estados, o maior valor associado a uma ação dentro da *Qtable* será determinante para escolher a ação a ser tomada quando o agente se encontrar em um destes estados. Considere um especialista que monta sua *Qtable* baseado em sua experiência no jogo, assim ele determina com valores altos as ações que ele tomaria. Conforme a Figura 3 podemos notar que as predições do agente não estão sempre corretas.

Para o estado mais acessado a ação escolhida pelo agente e pelo especialista condiz em partes, pois foi considerado o conjunto de ações agrupado como descrito na Tabela 3, porém neste caso, nota-se que a ação de colocar bomba está com valores de escolha de sinal oposto e o agente não a escolherá mesmo que nenhum movimento direcional seja

NOME DO ESTADO	# state/ ACTION	0	1	2	3	4	5
		Stop	Up	Down	Left	Right	Bomb
['alive', 2, 'bomb']	33	3,98535852	237,149665	220,9158995	307,0139	397,214619	-10
['alive', 2, 'bomb']	33	-100	100	100	100	100	100
['alive', 'near_wood', 2, 'bomb']	37	-1,04	360	0,109999	174,76	111,2811	100
['alive', 'near_wood', 2, 'bomb']	37	-100	100	100	100	100	200
['alive', 'unsafe_directions', 2]	48	203	324,4418118	190	106,4536011	190	-710
['alive', 'unsafe_directions', 2]	48	-100	100	100	100	100	-1000

LEGENDA: ESPECIALISTA AGENTE

Figura 3. Tabela de comparações das escolhas do agente com o especialista para os estados mais acessados.

possível. O segundo estado mais acessado tem a ação escolhida pelo agente completamente diferente da escolhida pelo especialista. E o estado com 10% de acesso possui a única escolha totalmente condizente com a do especialista.

A Figura 4 mostra o gráfico com a oscilação da recompensa média de cada episódio. Pode-se notar que há picos negativos que representam as interferências do $\epsilon - Greedy$ na escolha das ações. Observa-se que não há uma crescente visível das recompensas com o passar do tempo, o que é um forte indicativo de que o agente não está alcançando seu objetivo com a atual implementação. Mesmo apresentando uma variância muito alta e com resultados negativos após um volume alto de treinos, a linha de tendência indica que, em média, o valor da recompensa sobe com o aumento dos treinos.

Figura 4. Gráfico de recompensa média em função do número de episódios treinados.

Utilizando o *Mean Squared Error (MSE)* como modelo estatístico para comparação de estimadores, podemos efetivamente exprimir o erro quadrático médio em função do viés do estimador em função da variância. O problema em falar sobre o erro médio de um determinado modelo estatístico é que é difícil determinar quanto do erro é devido ao modelo e quanto é devido à aleatoriedade. O *MSE* fornece a estatística necessária para fazer tais afirmações, este refere-se simplesmente à média da diferença quadrática entre o parâmetro previsto e o parâmetro observado [Dennis D. Wackerly 2008].

Utilizando das ferramentas disponibilizadas por [Pedregosa et al. 2011], a biblio-

teca *scikit-learn* da linguagem de programação *python*, foi gerado a Figura 5 que mostra o gráfico com o *MSE* da *Qtable* em função do número de episódios treinados. Novamente nota-se uma alta variância, neste caso o esperado seria que o *MSE* tendesse a zero com o passar dos treinos. A linha de tendência mostra que há afastamento do zero com o aumento dos treinos.

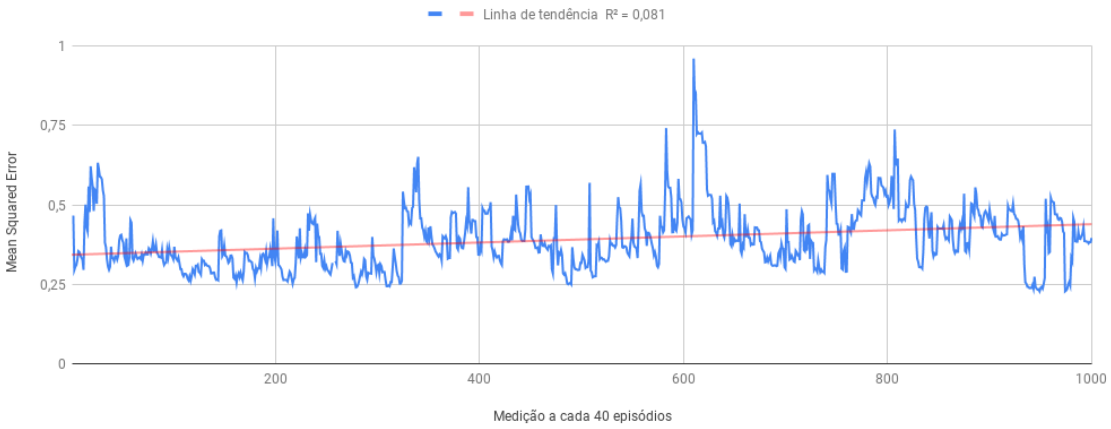

Figura 5. Gráfico de evolução *MSE* em função do número de episódios treinados.

8. Conclusão

O modelo implementado possui vantagens para o desenvolvedor, uma vez que traz transparência sobre como o aprendizado do agente está acontecendo de fato, mostrando claramente os estados utilizados. Este fato o difere de outras técnicas geralmente usadas quando se trata de aprendizado por reforço, nos trabalhos existentes costuma-se utilizar redes neurais para auxiliar na predição dos estados a serem gerados, o que apresenta ótimo resultados mas não torna possível ao desenvolvedor, do agente, a visualização real de como o agente aprendeu.

No entanto à aplicação do modelo não se mostrou muito eficaz. O contexto da plataforma interferiu nos resultados uma vez que este não possibilitou uma larga escala de treinos, na casa dos milhões, visto que essa normalmente é uma característica dos agentes de aprendizagem por reforço. Em geral, como não há a inserção de dados no aprendizado, os dados devem ser gerados e testados, o que se faz necessária a grande quantia de treinos.

A avaliação dos resultados obtidos mostra que com o aumento de execuções a recompensa média tende a aumentar, o que mostra uma certa efetividade do agente. Porém em uma execução teste posterior ao treino, em mil episódios da aplicação, há uma taxa de vitória de 21%. Como se trata de um jogo, é esperado que um agente treinado tenha como conquista a maioria das vitórias, o que não ocorre neste caso.

Observando o Gráfico de *MSE* fica claro que há uma taxa de erros muito alta, verifica-se isso ao observar o pico de valor 0,96 Após 24400 execuções, o que propõem um erro alto na *Qtable* e conferiu ao agente uma recompensa de -172,04. O *MSE* alto propõem que está não seja uma recompensa ruim gerada pela escolha randômica proporcionada pelo $\epsilon - Greedy$, logo verifica-se um erro grave do agente no decorrer dos episódios que levaram a uma *Qtable* incompatível. Essa incompatibilidade pode ainda ser verificada na Figura 3.

Conclui-se que o modelo propõem bons apontamento para a técnica de aprendizado por reforço aplicada ao *Pommerman*, verifica-se a necessidade de um número muito maior de treinamento e eventualmente mudanças no cenário das políticas escolhidas para obter resultados mais próximos dos resultados de um especialista.

Referências

(2018). Bomberman. https://pt.wikipedia.org/wiki/Bomberman. [Online; accessed 21-Novembro-2018].

Cinjon Resnick, Denny Britz, D. H. J. F. e. W. E. (2017). Pommerman - game description. https://www.pommerman.com/about. [Online; accessed 19-Novembro-2018].

Cinjon Resnick, Denny Britz, D. H. J. F. e. W. E. (2018). Playground: Ai research into multi-agent learning. https://github.com/MultiAgentLearning/playground. [Online; accessed 19-Novembro-2018].

Dennis D. Wackerly, William Mendenhall III, R. L. S. (2008). *Mathematical Statistics with Applications*. Thomson Brooks Cole TM, seventh edition.

Joseph Groot Kormelink, M. M. D. e. M. A. W. (2018). Exploration methods for connectionist q-learning in bomberman.

Lee, S. R. (2018). Pommerman - end to end ai. https://www.endtoend.ai/blog/pommerman-1/. [Online; accessed 19-Novembro-2018].

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Raval, S. (2017). Deep q learning for video games - the math of intelligence. https://www.youtube.com/watch?v=79pmNdyxEGo.

Romero, R. A. F. (2017). Aprendizado por reforço. http://wiki.icmc.usp.br/images/b/b3/AprendizadoReforco_Robotica_RAFR.pdf.

Sutton, Richard S., B. A. G. (2018). *Reinforcement learning: An introduction*. The MIT press, second edition.