Um Agente de Aprendizado por Reforço no jogo digital Tibia

Ricardo Avelar, Aluno, UFMG e Luiz Chaimowics, Professor Orientador, UFMG

Abstract

Utilizou-se a rede PPO e posteriormente a rede DQN no jogo digital Tibia, como pesquisa tecnológica. Após mais de uma centena de horas de treinamento total observou-se, ao menos para um cenário simplificado do Tibia, que ou: 1. Um agente de aprendizado por reforço não é capaz de aprender sem passar por dezenas de horas de treinamento, com um step de tempo mínimo de 0.3 segundo para atualização do ambiente; ou 2. Encontrar os parâmetros corretos para o aprendizado rápido, consistente e eficaz nos condições testadas é extremamente demorado, dado o tempo mínimo para um treinamento completo.

I. Introdução

O Presente trabalho tem como objetivo desenvolver um agente de inteligência artificial treinado por meio da técnica de Aprendizado por Reforço, utilizando como ambiente o jogo digital Tibia. Inspirado por projetos de grande impacto, como o OpenAI Five [4] e o AlphaStar [5] — que revolucionaram a aplicação de IA em jogos eletrônicos —, este estudo busca explorar estratégias de implementação, compreender os desafios envolvidos e construir um agente funcional e adaptável dentro do ambiente proposto.

No contexto da indústria de jogos, agentes inteligentes mais completos e competitivos possibilitam a criação de funcionalidades mais sofisticadas, como o treinamento de jogadores profissionais, a análise automatizada de replays e a melhoria geral da experiência dos usuários. Assim, este trabalho busca investigar o potencial e as limitações do Aprendizado por Reforço em um cenário realista e desafiador como o de Tibia.

O jogo Tibia é, de acordo com a própria empresa dona do jogo, "um jogo de RPG online multijogador massivo gratuito (MMORPG)". Nele, os jogadores controlam seus personagens por uma câmera top-down, e podem desenvolver as habilidades de seus personagens, buscar tesouros, resolver enigmas e explorar áreas como cidades, masmorras, florestas, desertos, ilhas, praias, minas, etc. Os personagens podem disputar lutas entre si ou com monstros, utilizando armas e magias. Dado essa visão geral, o que é importante tecnicamente para este trabalho são elementos específicos, como o personagem controlado pelo jogador tendo pontos de vida, habilidades especiais, itens e recursos para derrotar os monstros. Da mesma forma, os monstros têm características específicas, como pontos de vida e habilidades. Além disso, levar em consideração o ambiente e suas regras, como subir e descer escadas e ser bloqueado por paredes é de extrema importância.

II. Referencial Teórico

Nos últimos anos, várias abordagens têm sido propostas para criar agentes inteligentes em jogos digitais utilizando-se aprendizado por reforço. Por exemplo, o jogo Dota 2, desde sua criação, sempre foi considerado extremamente complexo, mesmo entre jogadores profissionais. Nesse cenário, a organização OpenAI desenvolveu o projeto OpenAI Five, um agente de Aprendizado por Reforço Profundo que, após anos de desenvolvimento e treinamento, foi capaz de derrotar os melhores jogadores do mundo na época.

Segundo a própria OpenAI, "Começamos o OpenAI Five para trabalhar em um problema que parecia fora do alcance dos algoritmos de aprendizado de reforço profundo existentes. Esperávamos que, ao trabalhar em um problema que era insolúvel pelos métodos atuais, precisaríamos fazer um grande aumento na capacidade de nossas ferramentas." Assim, a escolha de Dota 2 como ambiente de testes visava justamente desafiar os limites da tecnologia e impulsionar o desenvolvimento de novos métodos de aprendizado.

De forma semelhante, a DeepMind, empresa responsável pelo projeto AlphaStar, aplicou técnicas avançadas de Aprendizado por Reforço para desenvolver um agente voltado ao jogo StarCraft II. Em dezembro de 2019, o AlphaStar derrotou um dos melhores jogadores profissionais do mundo com um placar de 5 a 0. Após a partida, o jogador derrotado afirmou: "O agente demonstrou estratégias que eu nunca pensei, o que significa que ainda há novas formas de jogar que nós ainda não exploramos". Essa declaração ressalta não apenas o alto desempenho do agente, mas também o potencial da inteligência artificial em expandir as fronteiras do design estratégico em jogos.

Embora o objetivo deste trabalho não seja atingir o mesmo grau de sofisticação dos projetos mencionados, a proposta se inspira nessas abordagens de sucesso para, de forma simplificada, desenvolver um agente inteligente em um ambiente mais acessível: o jogo Tibia. A escolha de um ambiente menos complexo permite focar em aspectos fundamentais do Aprendizado por Reforço, facilitando a análise e a implementação de estratégias adaptativas de forma controlada e gradual.

Fig. 1. Spritesheet extraído pelo SpiderClientConverter

III. METODOLOGIA

A. Criação do ambiente

Para ter total controle do ambiente não seria possível utilizar dos servidores e do jogo oficial para o treinamento do agente. No entanto, para se manter fiel à ideia, foi feito toda a configuração do ambiente com um binário extremamente semelhante ao jogo oficial e um servidor local, onde havia muito menos limitação que no jogo oficial.

B. Configuração do ambiente

Para facilitar a implementação, foi utilizada a biblioteca Gymnasium para a configuração do ambiente e utilização da rede PPO, inicialmente. Ainda, decidi usar uma CNN para transformar os pixels da imagem que o jogo renderizava em dados para alimentar a rede de treinamento do agente.

Para treinar a CNN foi necessário usar o software SpiderClientConverter [1] para extrair spritesheets através do binário do jogo (Fig. 1), e foi necessário criar um pequeno software para separar esses sprites em tiles do mesmo tamanho do tile do jogo, já treinando a CNN com a resolução correta.

É muito comum em treinamento de Reinforcement Learning termos a velocidade do treinamento limitada pelo hardware, como o ambiente de exemplo LunarLander-v3 do Gymnasium, por exemplo, onde você consegue rodar dezenas ou centenas de steps por segundo em modos sem renderização. No caso deste trabalho, no entanto, pelo fato de não ter acesso ao código fonte do jogo, a estratégia utilizada forçava um step mínimo de 0.3 segundo, uma vez que após a ação do agente o ambiente precisava reagir. Por exemplo, se o agente decidisse andar, para ir para o próximo step era necessário esperar a animação de ”andar” ter tocada e o boneco precisava chegar no próximo tile, para que o ambiente fosse lido novamente.

C. Treinamento da CNN

Pela decisão de utilizar uma CNN para a detecção de tiles na imagem renderizada, fica evidente que essa CNN precisa ser treinada, uma vez que ainda não existia CNN treinada para tal tarefa. Para tanto, foi criado um segundo software para a separação dos spritesheets criados pelo SpiderClientConverter, onde esse software me permitiu manualmente selecionar cada tile desejado. Os tiles selecionados foram os monstros que eu gostaria que a CNN reconhecesse, o chão e as paredes, para que o agente pudesse entender onde é possível andar e o que ele pode atacar. A figura 2 mostra as imagens já separadas.

Treinar a CNN para o reconhecimento dos monstros por si só já foi um desafio. Inicialmente a rede não teve nenhuma capacidade para separar os monstros ou o background, o que me fez tentar utilizar outras estratégias, como filtragem por Autoencoder e utilização de YOLO, em vez de uma CNN comum. Após várias tentativas falhas, consegui ser bem sucedido ao utilizar Segmentation Net, uma CNN pré-treinada perfeita para essa tarefa, por funcionar muito bem para imagens que devem ser segmentadas, como é o caso aqui, onde a imagem é segmentada em tiles.

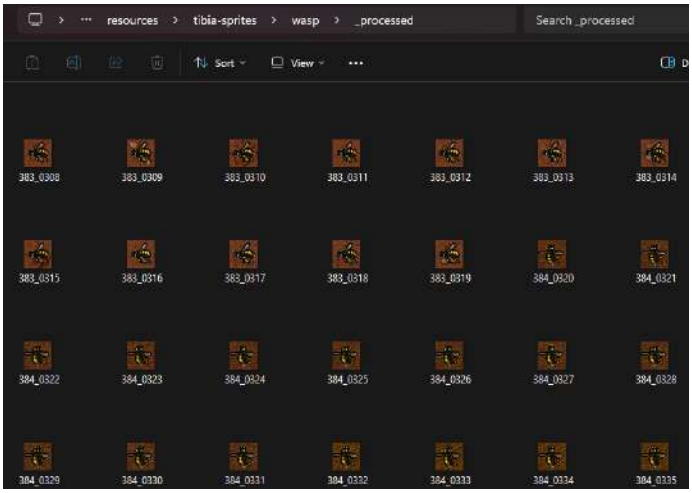

Fig. 2. Sprites após tratamento

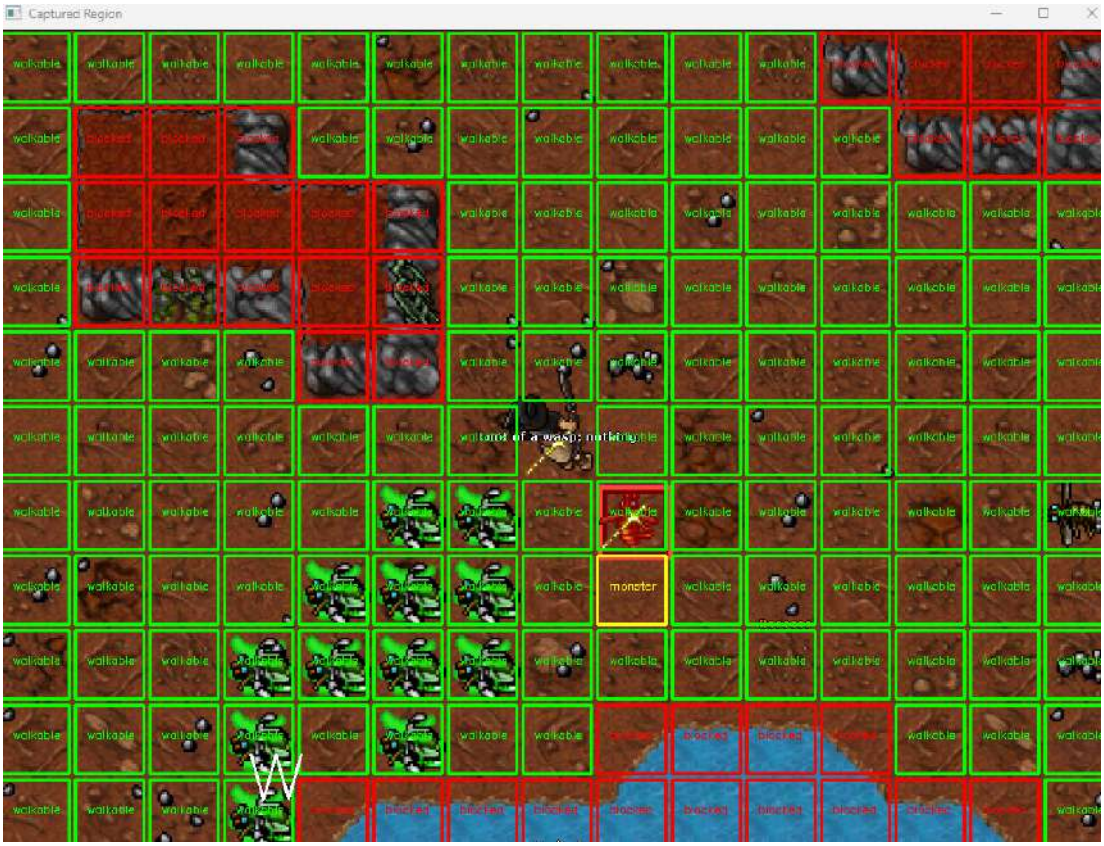

Fig. 3. Visualização da classificação da CNN

Por fim, a última modificação feita na CNN foi a otimização de usar GPU para o reconhecimento, em vez de CPU, aumentando a velocidade de reconhecimento de uma média de 1 segundo para uma média de 0.003 segundo. Na figura 3 podemos ver o resultado final. Note que para nossa visualização existe um pequeno delay, o que faz com que o tile classificado do monstro não seja preciso para a nossa visualização, se ele estiver em movimento, mas isso não afeta a visão real da rede.

Observe que temos três tipos de classificação de tiles, *Walkable*, *Blocked* e *Monster*. Essa classificação foi feita pois assim conseguimos passar como input para a rede do agente duas matrizes binárias concatenadas, cada uma representando o estado atual da tela em termos de classificação. Em outras palavras, podemos ter as matrizes W (walkable), e M (monster), onde a matriz W é uma matriz 11x15, que representa a quantidade de tiles na tela, e será 1 nos tiles que são ”andáveis” e 0 em tiles que não são ”andáveis” (estão bloqueados). Ainda, temos a matriz M 11x15 que é 1 onde há monstro e 0 onde não há monstro.

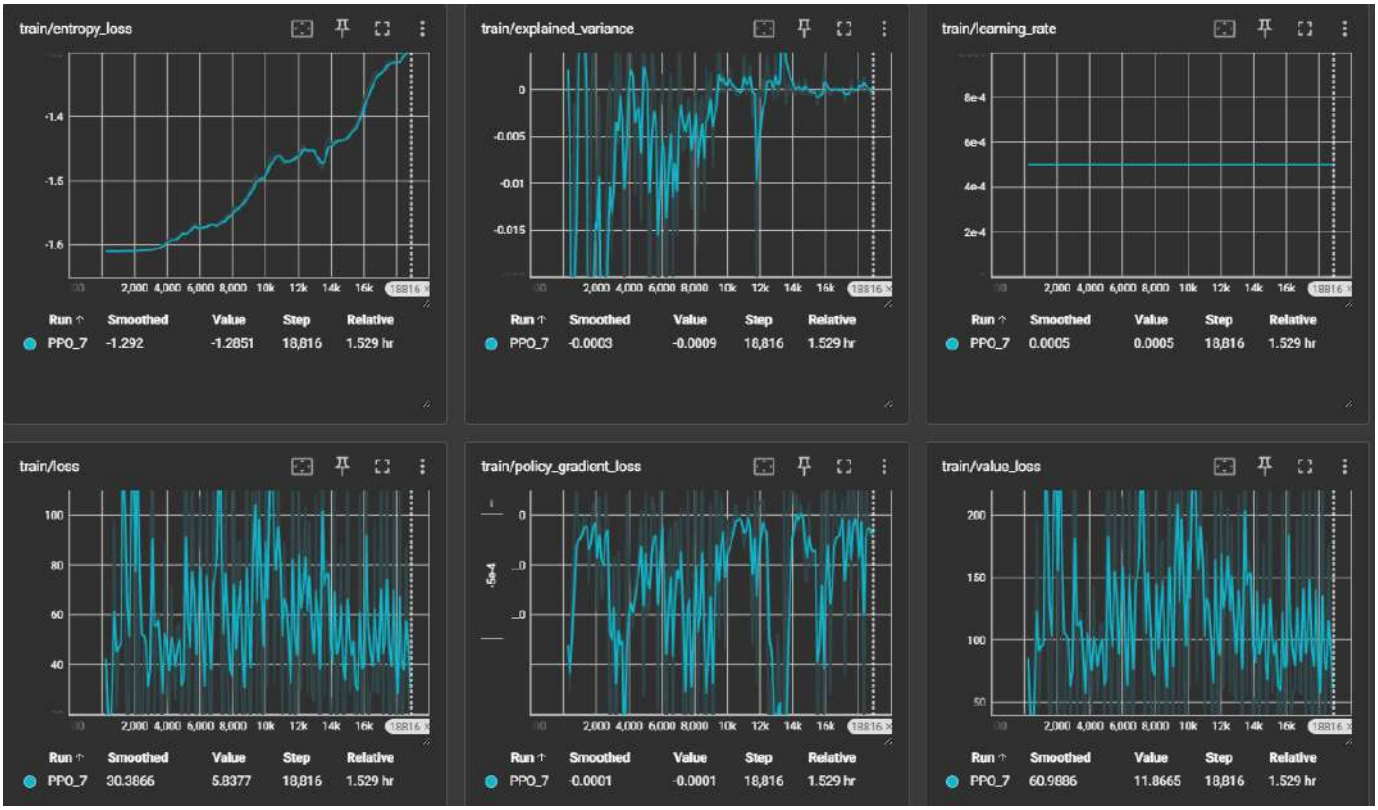

Fig. 4. Sétimo treinamento, PPO

D. Configuração da Rede

Por ser o estado da arte em Reinforcement Learning em jogos, a primeira rede utilizada para tentar alcançar o objetivo foi a PPO. Hiperparâmetros quase aleatórios foram definidos e a função de recompensa abaixo foi colocada no ambiente.

$$\mathbf{R = -1*B + W + E}$$

Na função de recompensa acima, B é o fator que diz se o agente foi bloqueado, W é o fator que diz se o agente andou com sucesso e E é a quantidade de experiência adquirida naquele step. Note que caso o agente seja bloqueado e o fator de bloqueio seja igual a 10, então o agente sofrerá uma penalidade de -10. Caso o agente ande com sucesso e o fator que diz se andou seja igual a 10, então o agente receberá uma recompensa de 10. Caso o agente derrote um monstro, recebera uma recompensa igual à experiência dada pelo monstro. Em resumo, a ideia é recompensar o agente caso ande corretamente ou derrote um monstro, e penalizar caso ande incorretamente.

E. Treinamento PPO

Durante todo o treinamento foram coletados logs, criados gráficos e avaliados os hyperparâmetros a cada treinamento. Como é possível ver na figura 4, os mais importantes dados coletados são a *entropy loss*, que diz o quão exploratória a política ainda é, a *explained variance*, que diz o tanto que a rede consegue explicar corretamente as ações que estão sendo feitas, e as métricas de *loss*, que juntas conseguem medir o progresso do agente.

Inicialmente, como é possível ver pela figura 4, treinamentos com poucos steps foram feitos. Comecei com algumas centenas, depois alguns milhares, até chegar no valor máximo de 80000 steps para um treinamento completo. Um treinamento com 80000 steps utilizando de um intervalo médio de 0.3s entre steps durava aproximadamente sete horas. Algumas vezes treinamentos de até 200 mil steps foram feitos, mas não houve uma diferença significativa.

Entre cada um dos steps foi modificado pelo menos um hyperparâmetro da rede com o objetivo ou de ter mais estabilidade no treinamento, ou conseguir algum indício de convergência. O último treinamento da rede PPO, com as métricas podendo ser verificadas na figura 5, ainda foi extremamente ineficáz, mesmo com 100 mil steps. Aqui, algumas estratégias diferentes foram adotadas, como diminuição na learning rate em uma curva de coseno e mudança na função de recompensa.

Esse foi o último treinamento da rede PPO, agora com a função de recompensa abaixo, onde a única mudança foi a adição do parâmetro C, que recompensava em algum valor caso o agente clicasse em um inimigo ao seu redor, e penalizava o agente quando o clique era longe de seu boneco. Isso se tornou importante e foi uma estratégia bem sucedida, pois assim conseguimos

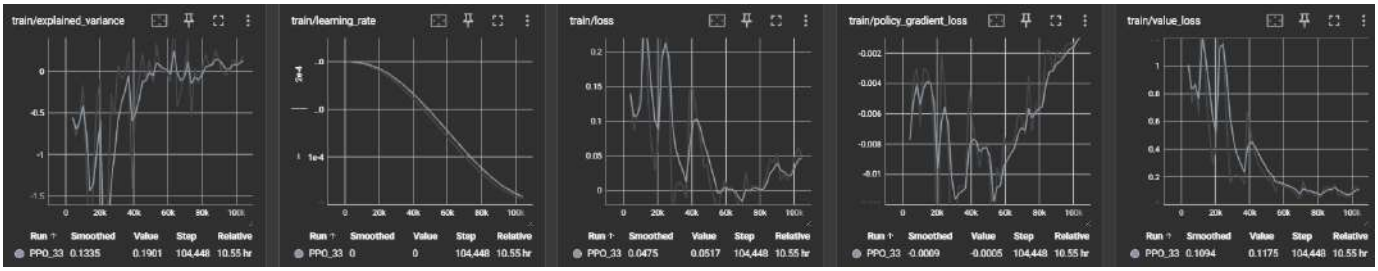

Fig. 5. Último treinamento, PPO

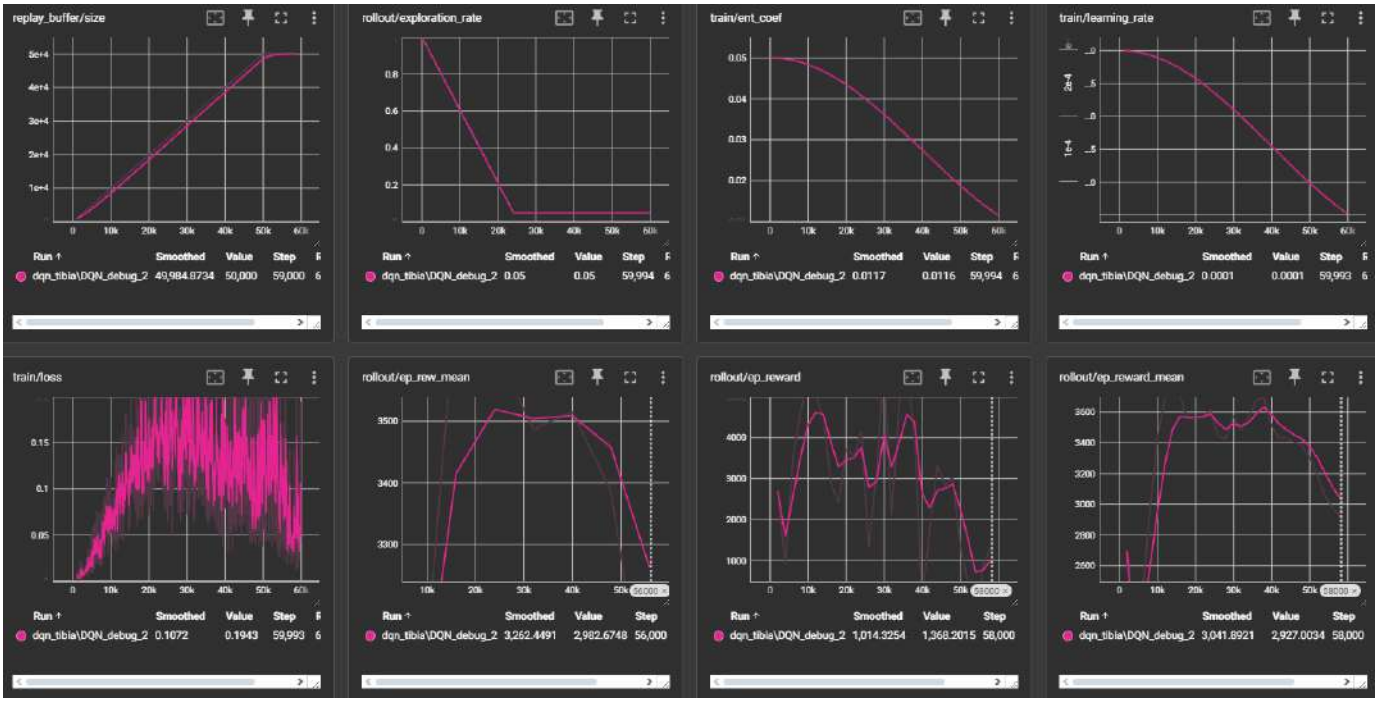

Fig. 6. Métricas da primeira tentativa, DQN

fazer o agente entender que é importante clicar no monstro que está perto, dado que o boneco só consegue derrotar monstros que estão ao seu lado.

$$\mathbf{R = -1*B + W + E + C}$$

Durante o treinamento da PPO, o comportamento mais próximo do objetivo principal foi o momento em que o agente aprendeu a andar de um lado para o outro, repetindo o tile imediatamente anterior, e aprendeu a clicar nos monstros que entravam na tela. Isso evidencia um comportamento de mínimo local, onde ele era recompensado por andar para um local ”andável” e era recompensado por clicar e derrotar um monstro quando ele eventualmente chegava perto.

F. Treinamento DQN

Após dezenas de treinamentos sem progresso claro utilizando a PPO, decidi mudar de estratégia e testar a rede DQN. A DQN se torna interessante por também ser uma rede bastante usada, mais simples e em alguns casos pode precisar de menos steps, fato esse que foi determinante para essa tentativa.

Além disso, como é possível ver pela figura 6, no primeiro treinamento usando DQN as méticas a serem rastreadas foram inevitavelmente diferentes, sendo as mais importantes a loss e o ganho de recompensa.

Assim como na PPO, ao utilizar a DQN diversos parâmetros foram mudados ao mesmo tempo, mas na DQN rapidamente os gráficos começaram a se tornar mais previsíveis. As mudanças eram mais consistentes e de fato causavam o resultado esperado.

Como é possível ver na figura 7 que são as métricas do treinamento 21, todas as métricas apresentam um padrão consistente de aprimoramento, mesmo que até o meio do treinamento a rede teve certa dificuldade para começar a aumentar as recompensas adquiridas.

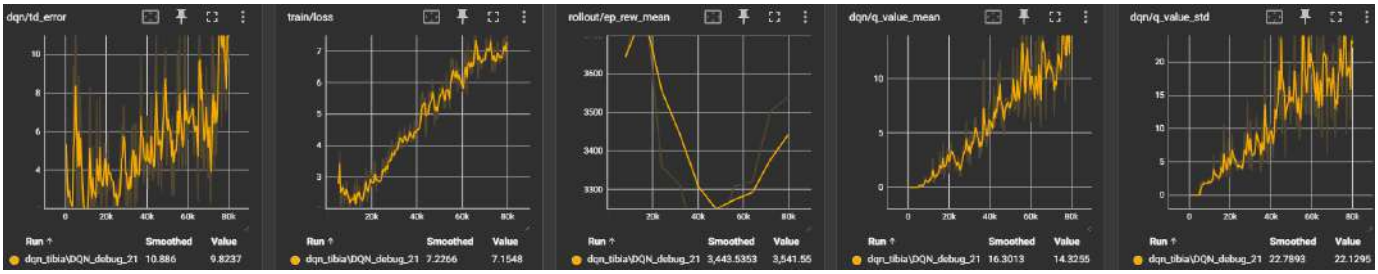

Fig. 7. Treinamento 21, DQN

$$f = \frac{3}{\sqrt{1 + (|50| - 1 - x)}}$$

$$R = a + (w \cdot f) + (b \cdot -10) + c$$

Fig. 8. Melhor função de recompensa encontrada

Após dezenas de treinamentos os melhores resultados eram alcançados em mínimos locais, onde o agente ficava repetindo o comportamento de andar apenas para o tile imediatamente anterior. Para tentar resolver esse problema, a função de recompensa foi fortemente modificada. A figura 8 mostra essa função, onde a é a diferença de experiência do step anterior para o atual, w é 1 quando o agente anda com sucesso e 0 caso contrário, b é 1 quando o agente tenta andar e é bloqueado, e 0 caso contrário, e c é 5 quando o agente clica num monstro ao seu lado, -5 quando clica num monstro longe do boneco e 0 quando não clica. Note que f está em função de x, que é o index de um vetor que contém o hash de todos os últimos tiles andados.

Esse vetor é tal que quando repete-se os tiles mais próximos, a recompensa é mais negativa. Na figura 9 pode-se analizar diversas curvas de recompensa em função do index de tiles, onde a curva W mostra o que acontece ao andar repetindo um tile.

IV. Resultados

Após dezenas de treinamentos e ajustes, o último treinamento demonstrou que o agente ainda estava aprendendo no final do treinamento. Como é possível ver na figura 10, as rewards ainda estavam aumentando, a loss e o TD-Error estavam estáveis (reforçando o aprendizado correto) e o Q-Value indicou que a rede estava ficando mais confiante.

O comportamento, no entanto, foi um agente andando de um tile para o outro, repetindo o tile imediatamente anterior, e atacando os monstros que chegavam perto. Percebeu-se, ainda, que o agente tinha preferência para atacar monstros em tiles mais pertos em vez de monstros que estavam mais distantes do agente.

V. Conclusão

Após dezenas de treinamentos utilizando a PPO com diversas combinações de hiperparâmetros, e após repetir mais treinamentos ainda com a DQN, é possível perceber com todos esses dados que o treinamento de um agente de aprendizado por reforço é inviável se o código fonte não estiver disponível. Pelo fato do step durar no mínimo 0.3 segundo, especialmente para

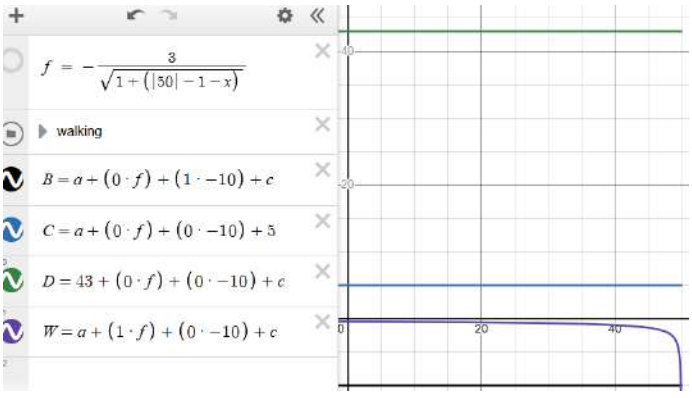

Fig. 9. Curva de recompensa com diferentes valores

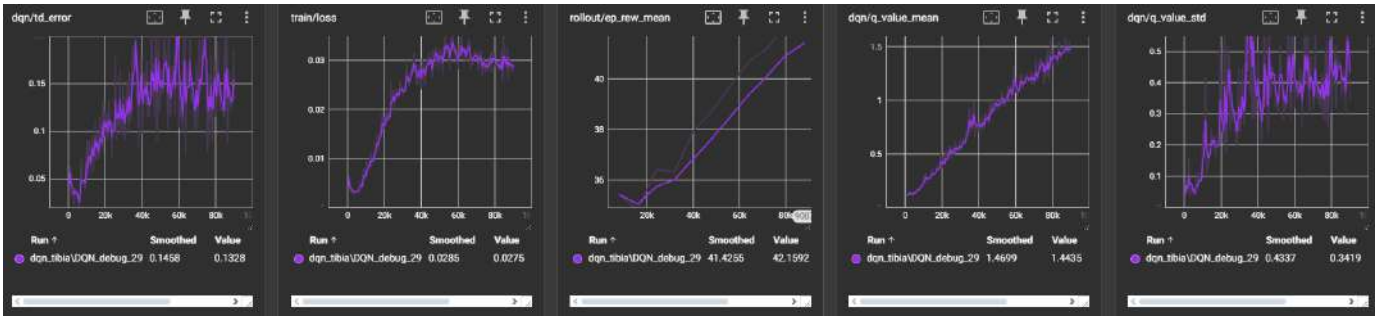

Fig. 10. Resultado do último treinamento da DQN

que o ambiente consiga atualizar devido à ação do agente, um treinamento pouco efetivo em termos de quantidade de steps já demora tempo suficiente para qualquer projeto se tornar inviável.

Portanto, no caso estudado, chega-se à conclusão que é necessário ou mais tempo de treinamento, aumentando a quantidade de steps para além de 200 mil (valor máximo testado), ou é necessário mais tentativas para adequar melhor os hiperparâmetros. Em ambos os casos fica evidente que Tibia não é um jogo fácil para o aprendizado por reforço, sem seu código fonte.

REFERENCES

[1] https://github.com/Arch-Mina/ClientConverter
[2] Richard S. Sutton and Andrew G. Barto. 2nd Edition. MIT Press, Cambridge, MA, (2019). Reinforcement Learning: An Introduction
[3] Li, Y. (2017). Deep Reinforcement Learning: An Overview.
[4] https://openai.com/index/openai-five-defeats-dota-2-world-champions
[5] https://deepmind.google/discover/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii/
[6] Vinyals, O., Babuschkin, I., Czarnecki, W.M. et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature 575, 350–354 (2019).