Instituto Tecnológico de Aeronáutica - ITA
Departamento de Computação
Engenharia da Computação
CMC-12

EXAME DE CONTROLE

Alunos: Thomas Alberto de Castro Neto, Vinícius de Pádua Dias Araújo e Henrique Fernandes Feitosa

Professor: Dr. Marcos Ricardo Omena de Albuquerque Máximo

São José dos Campos - SP

Flappy Bird utilizando Q-Learning

Thomas Alberto de Castro Neto, Vinícius de Pádua Dias Araújo, Henrique Fernandes Feitosa

2021/1

1 Motivação

Diante do enorme apelo que aprendizado de máquina, inteligência artificial têm no mundo atual, nosso grupo ficou tentado em tentar aplicar técnicas de controle que possuam interseção com essa área do conhecimento. Para tanto, pensamos em utilizar aprendizagem por reforço. O conceito de aprendizagem por reforço foi escolhido para aprofundarmos.

O conceito de aprendizado por reforço foi bastante estudado por ramos diversos como psicologia, estatística e neurociência, além da ciência da computação. É um método de fácil compreensão, já que intuitivamente, imaginamos um processo de aprendizado como escolhas que cometemos possuem consequências que podem ser quantificadas e utilizadas futuramente para novas decisões serem, no mínimo, melhores do que as tomadas previamente. Neste tipo de aprendizado ressaltamos que não há necessidade de especificar como uma tarefa deve ser realizada. O método "didático"já oferece a solução.

Assim, para utilizar os conceitos que desejavámos aprender, escolhemos como tema o jogo *Flappy Bird*. O jogo foi desenvolvido em 2013, e no ano seguinte alcançou o topo da categoria de jogos gratuitos da *Apple Store* chinesa e americana.

Figura 1: *Flappy Bird*.

O objetivo do jogo é ganhar o maior número possível de pontos, controlando um pássaro (tocando na tela) sem deixá-lo colidir nos objetos. O pássaro cair ou encostar em algum objeto, o jogo termina.

2 Mergulhando no Algoritmo

2.1 Cadeia de Markov e Aprendizado por Reforço

Para o algoritmo de reforço, utilizaremos o conceito de Cadeias de Markov. A ideia central da cadeia, é criar uma relação contínua entre um agente e seu ambiente.

No caso do nosso problema, o agente é a IA e o ambiente será um estado e sua função de recompensa. Baseado nestes *inputs*, o algoritmo que construiremos decidirá qual será a melhor ação

para executar. Assim, finalmente, ele nos dará um novo estado e uma nova recompensa como *outputs*.

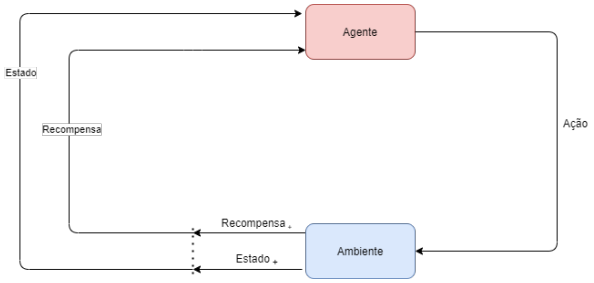

Figura 2: Cadeia de Markov.

Mais especificamente, o processo é desenhado para alimentar a si mesmo, utilizando a interação entre eles para refinar o aprendizado do algoritmo. Dessa maneira, a nossa missão é que o agente maximize a medida de desempenho, chamaremos de Q (**Q**ualidade) baseado em reforços - recompensas e punições - que recebe ao interagir com o ambiente. O agente receberá um valor de reforço cada vez que executar uma ação, de maneira que indicará qual o próximo estado. Conforme o algoritmo for se aprimorando, espera-se que as decisões tomadas pelo agente satisfaçam o objetivo proposto, i.e., não deixar o pássaro tocar no chão ou objetos.

2.2 Definindo os Estados

No ambiente do jogo, definimos duas observações por estado: a distância para o cano e a altura do pássaro. Dado este estado, nosso agente deve ser capaz de decidir entre duas possíveis ações: tocar na tela, ou não. Esta decisão será recompensada ou penalizada e como saída, obteremos um novo estado uma vez que retornaremos a ação ao ambiente. Resumindo o que foi dito, temos:

Tabela 1: Estados e Ações do modelo.

Estado	**Ações**	
Distância Horizontal	Flap	No Flap
Distância Vertical	Flap	No Flap
Velocidade do pássaro	Flap	No Flap

Abaixo, mostra-se como é medido tais valores.

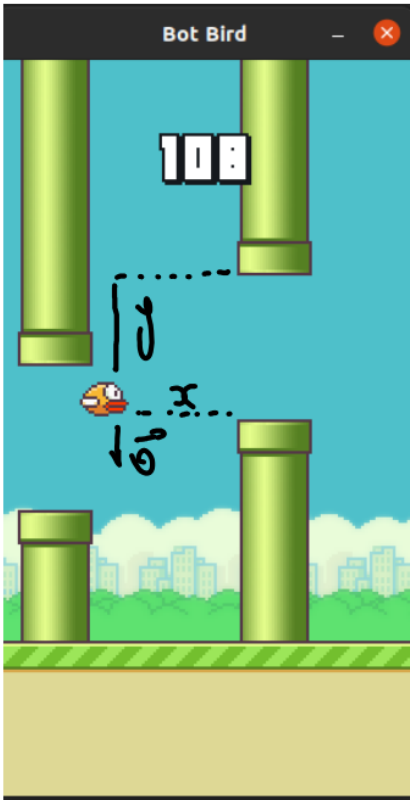

Figura 3: *Estados medidos segundo o passaro.*

2.3 Recompensas e Punições

Aqui está o cerne do agente fazer boas decisões. Devemos calibrar as recompensar para estimular o agente a deixar o pássaro mais tempo vivo conforme for possível. Dessa maneira, conseguiremos a maior pontuação no jogo.

Para o nosso algoritmo, iremos escolher uma recompensa de 0 pontos para cada escolha que mantenha o pássaro vivo e iremos penalizar com -1000 pontos cada decisão que faça nosso pássaro se chocar com o cano ou atingir o chão. Acredita-se que não dar pontos para a ação que mantenha o pássaro vivo irá lhe mostrar que não é nada mais que sua obrigação estar vivo e, portanto, mostrando que, nem mesmo se permanecer vivo para sempre ele poderá morrer. Para mostrarmos a eficiência, compararemos com uma rodada de simulação em que a recompensa por permanecer vivo será de 1 ponto.

Tabela 2: Valores de Recompensas.

	Recompensa
Morrer	-1000
Permancer vivo	0

3 Q-Learning

3.1 Equação de Qualidade

Como dito anteriormente, queremos maximizar o valor de uma função que quantifica a "qualidade"de uma ação. Definimos da seguinte maneira a função qualidade

$$Q^{new}(s_t, a_t) \longleftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot max_a Q(s_{t+1}, a)) \quad (1)$$

Onde α é a taxa de aprendizado (o quão rápido nós desejamos que o modelo se adapte ao problema, geralmente entre 0 e 1), r_t é a recompensa, γ é o fator de desconto (uma métrica de importância do futuro na decisão atual, geralmente entre 0 e 1) e a parcela $max_a Q(s_{t+1}, a)$ nos dá uma estimativa do valor ótimo futuro. Podemos notar na equação acima que o novo valor da função de qualidade é uma ponderação entre a taxa de aprendizagem. Aumentando, nós aumentamos o peso da segunda parcela da equação. Da mesma maneira, vemos que aumentando o valor de γ, daremos mais peso ao valor da estimativa futura.

Assim, o algoritmo pode achar um conjunto de ações ótimas começando de um estado inicial para maximizar o valor da recompensa final. Em termos mais simples, o algoritmo "olha no futuro"para maximizar a sua recompensa final como um todo e essa escolha baseia sua decisão.

3.2 Construção da Tabela-Q

Utilizando os seguintes valores de α, γ e recompensa r_t. Para testar a escolha de α, iremos comparar a situação com um teste em que $\alpha = 0.8$.

Tabela 3: Valores escolhidos dos parâmetros

	Valor escolhido
α	0.707
γ	0.97
r	-1000 e 0

4 Resultados e Conclusões

Feito o código, rodou-se 3 situações. Fez-se *iterations* = 15000 com $\alpha = 0.8$ e o reward para estar vivo valendo 0. Após isso, tentou-se rodar *iterations* = 15000 com $a = 0.707$ e mesmo reward, mas o pássaro aprendeu tão rápido que, após 2 horas, ele já havia chegado em um score de 14000 e, dessa forma, decidiu-se parar em 5294 iterações. Como ultimo teste, fez-se $\alpha = 0.8$ e *iterations* = 10000 com reward para estar vivo valendo 1. Com isso, compila-se nos gráficos abaixo os resultados obtidos.

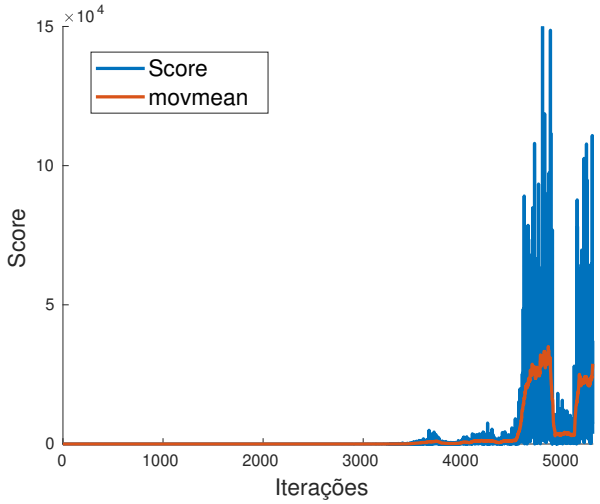

Figura 4: *Curva de treino para* $\alpha = 0.707$ *e* $r_{vivo} = 0$.

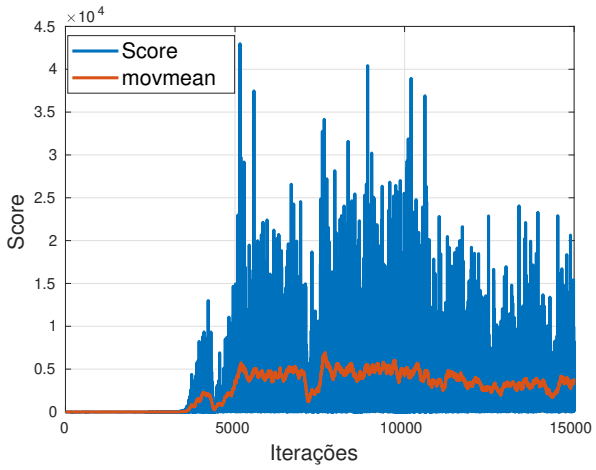

Figura 5: *Curva de treino para* $\alpha = 0.8$ *e* $r_{vivo} = 0$.

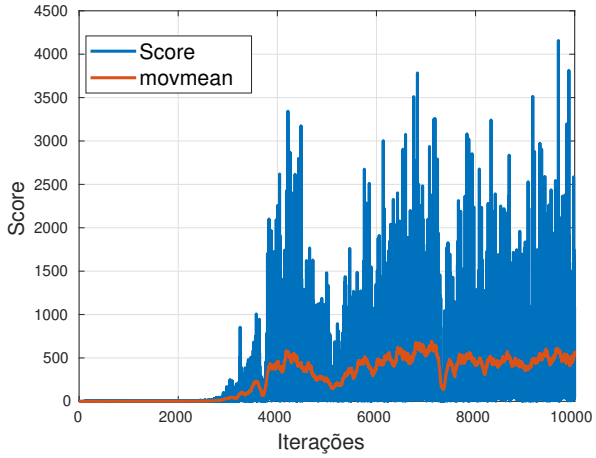

Figura 6: *Curva de treino para* $\alpha = 0.8$ *e* $r_{vivo} = 1$.

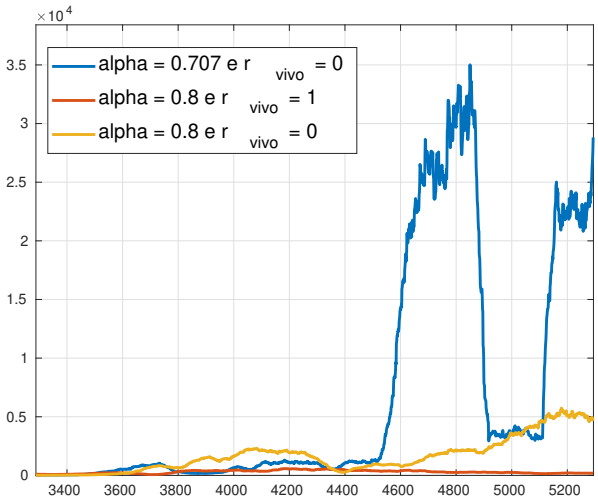

Figura 7: *Comparação entre uso dos parâmetros.*

Inicialmente, nota-se que os scores, para todas as baterias de treino, são muito oscilantes. Isso se deve à programação da visão dos estados, pois considera-se que o pássaro passará a ver o próximo

cano, e deixará de ver o atual, assim que ele entrar 30px dentro do cano (cano possui 52px). Dessa forma, ele acaba batendo no cano sem saber o porquê, como se o pássaro tivesse morrido batendo em um nada. Por isso, mostra-se a média móvel de aprendizado junto aos gráficos, para que se tenha ideia de uma curva de aprendizado "contínua".

Vê-se, também que, de fato os parâmetros utilizados e mostrado na tabela 3 são os que resultam em uma melhor curva de treino. Antes da iteração de número 5000 o pássaro já havia conseguido chegar no score de 150000!

Além disso, nota-se, claramente, como a curva de treino fica pior quando consideramos um score não nulo para a situação em que o pássaro vive. Mesmo que ele consiga chegar em níveis alto, o aprendizado é estagnado e fica na ordem de 10 - 100 vezes pior que os outros dois casos de teste.

Nota-se, curiosamente, que os bots conseguem aprender rapidamente até um pico, mas logo depois tem um decréscimo de rendimento, demorando a chegar no seu máximo novamente. Esse efeito é conhecido como inferência catastrófica. Ocorre, depois de aprender várias vezes um mesmo cenário de falha, uma espécie de overfit de falha, que gera um esquecimento da ação ótima para tomar em certa situação.

Dessa forma, acredita-se que o projeto foi um sucesso. Futuras melhorias envolvem adicionar mais estados a fim de mitigar o erro introduzido pela visão do cano e técnicas para diminuir o efeito do esquecimento.