Utilizando IA para jogar blackjack

Gustavo Lopes Rodrigues
Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica
de Minas Gerais (PUC-MG)
Belo Horizonte, Brasil
Email: gustavolr@gmail.com

Homenique Vieira Martins
Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica
de Minas Gerais (PUC-MG)
Belo Horizonte, Brasil
Email: homenique.t@gmail.com

***Abstract*—Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.**

***Index Terms*—Keywords, here, separated, by, comma**

I. Introdução

[1].

No ramo de inteligência artificial, existe um mercado amplo onde têcnicas dessa área são usadas para criarem oponentes inteligentes, que possam derrotar qualquer jogador. Alguns exemplos famosos a serem citados são: a Deep Blue da IBM que conseguiu derrotar o ex-campeão mundial de xadrez Garry Kasparov, ou, o AlphaGo da Google, que derrotou o campeão Fan Hui. Todos os jogos citados anteriormente são jogos determinísticos, ou seja, não são jogos com fatores probabilísticos envolvidos na hora de calcular a jogada. Um dos maiores desafios desta área da computação é como tratar os jogos de azar, onde a sorte é um elemento que precisa ser calculado para a realização das jogadas.

Este estudo tem como objetivo criar uma aplicação prática que possa ser usada para criar um jogador inteligente de Blackjack(no Brasil conhecido como 21). Além disso iremos avaliar diferentes estratégias/políticas e avaliar a utilidade esperada de cada estratégia para a inteligência artificial.

Este artigo está dividido da seguinte maneira: primeiros na secção 2 será explorado um pouco da literatura já existente do assunto. Na secção 3 as regras do Blackjack serão expostas e analisadas no contexto da aplicação que será feita. No estudo em questão, a Godot Engine será utilizada para criar a interface do programa. Na secção 4 será analisado as diferentes estratégias e políticas utilizadas no programa. Na secção 5 acontecerá uma análise dos resultados chegados, a precisão de cada modelo e o tempo de execução. Por fim, terá a conclusão, onde será dado uma opinião final sobre os dados chegados, além de possíveis futuros trabalhos que podem ser feitos a partir deste.

II. Trabalhos Correlatos

Um padrão muito comum encontrado na literatura do assunto que está sendo abordado, é o fato de que os algoritmos de aprendizado de máquina, que usam de técnicas avançadas de inteligência artificial, ainda estão em sua juventude neste campo. Como exemplo, tem o artigo de Raphaela(2015), que utiliza de tais técnicas para formar regras para maximizar as vantagens do jogador em relação ao cassino. Entretanto, os algoritmos em geral chegaram a um número pequeno de regras, o que fazia com que certas jogadas não contemplassem as configurações do jogo.Marvin e Fernand(2012) também tentou utilizar uma abordagem moderna utilizando uma arquitetura cognitiva chamada CHREST, entretanto, apesar que tal modelo tenha chegado a bons resultados, foi concluído que tal arquitetura seria melhor utilizada em jogos com um número maior de padrões a serem memorizados e reconhecidos, como o jogo Poker. Por fim temos Robert(2012) que além de usar uma abordagem moderna com aprendizado genético, também utilizou de estratégias usando inteligências artificiais baseadas em regras, chegando ao resultado que ainda existe espaço a ser otimizado nas estratégias atuais, enquanto que as abordagens mais tradicionais continuam produzindo melhores resultados de maneira geral.

Todos os trabalhos até então mencionados tentam de alguma forma ou outra criar um jogador inteligente de Blackjack para aplicações, Raphaela(2015) criou uma aplicação em C, enquanto que tanto Marvin e Fernand(2012) e Robert(2012) criam aplicações em Java. Nosso objetivo é também criar um jogo, porém que possa ser feito em uma linguagem de alto nível e que tenha uma utilização mais fácil, por isso a linguagem escolhida foi a GDScript da Godot Engine, que possui uma tipagem parecida com Python, mas que já foi construída para criar jogos. Além disso, tentaremos utilizar abordagens baseadas em regras (Expectiminimax) como também iremos tentar fazer a aplicação usando Aprendizado por máquina.

III. Metodologia ou Proposta de Arquiteturas

Para a elaboração de teste utilizamos o simulador-Amnésia para elaborar algumas arquiteturas (tabelas **??** e I) que a partir desses cenários serem executados executamos os testes (tabela II)

Ao modificar o valor do tamanho da memória e da política de substituição da cache, é possível criar 8 cenários distintos para se avaliar o impacto do tamanho da cache no desempenho dos métodos de substituição. Sendo assim, com os cenários

TABLE I
CARACTERÍSTICAS GERAIS DA ARQUITETURA II

Especificações da Arquitetura	Partes da Arquitetura	
	Memória Principal	*Cache*
Tamanho da linha / bloco	1	1
Ciclos por leitura	1	1
Ciclos por escrita	2	2
Tempo do ciclo	10	1
Tamanho da memória	16	*2
Associatividade	—	2
Politica de escrita	—	Write-Through
Politica de substituição	—	*FIFO

* Os valores com o asterisco foram os valores modificados.

TABLE II
DADOS MODIFICADOS

Especificações da Arquitetura	Valores	
	Tamanho da Memoria	Politica de Substituição
Cenário 1	2	FIFO
Cenário 2	2	LRU
Cenário 3	4	FIFO
Cenário 4	4	LRU
Cenário 5	8	FIFO
Cenário 6	8	LRU
Cenário 7	16	FIFO
Cenário 8	16	LRU

propostos foram realizados diversos testes com o intuito de analisar a localidade temporal e localidade espacial nas arquiteturas.

IV. AVALIAÇÃO DOS RESULTADOS

Nos gráficos subsequentes, há uma demonstração de como a memória cache se comporta ao ser usada com um grande conjunto de instruções de *load* e *store* (maior que sua capacidade) em cenários com caches de 2, 4, 8 e 16 linhas. Para fazer esses testes utilizamos um conjunto de 10, 50, 100, 250, 500, 750, 1.000 e 10.000 instruções.

A. Gráficos de hit rate por número de instruções

Os dois gráficos mostrados foram montados usando os valores que extraímos. Nota-se que os resultados foram muito próximos, aos resultados de James E. Smith e James R. Goodman [**?**]. Considerando que no início, pensávamos que haveria uma diferença significativa entre os algoritmos LRU e FIFO para substituição de valores na cache, achamos que nossos testes estavam errados e que precisaríamos refazê-los, ao ler o artigo vimos que os nossos resultados na verdade estavam bem dentro do esperado.

Nossos testes apenas foram feitos usando caches completamente associativos, pensamos em fazer com a associatividade por conjunto e direta, mas depois de resultados tão inesperados, pensamos em pesquisar outros artigos relacionados que já tivessem feito esses testes para comparar os resultados e ver se os resultados seriam diferentes. Ao ver os dados do artigo [**?**], descobrimos que teríamos valores bem semelhantes dentro dessas três arquiteturas.

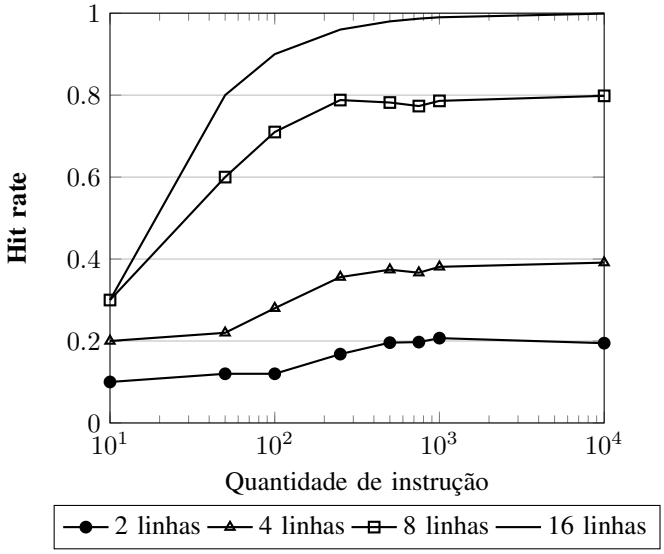

Fig. 1. ***Cache com algoritmo de substituição FIFO***

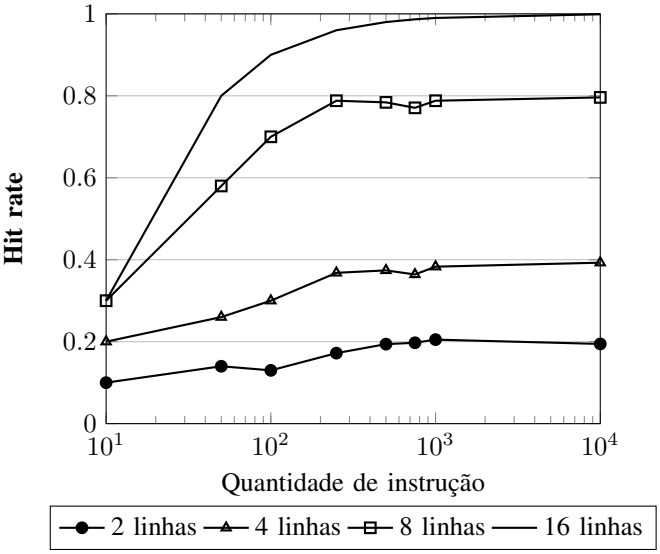

Fig. 2. ***Cache com algoritmo de substituição LRU***

Para colocar em perspectiva nossos resultados, os dados em [**?**] foram medidos usando benchmark mantido pelos grupos de pesquisa, considerando uma cache de 2KB com 32 bytes de linhas. Eles são expressos em termos da porcentagem de hits na cache detectada pela análise feita pelo grupo. A política de LRU se demonstrou mais eficiente do que a PLRU e a política aleatória de troca, em apenas alguns casos que não houve diferença entre as análises, porém na pesquisa d [**?**] foi utilizado também como parâmetro o tempo de vida mínimo de um elemento na cache(*Minimum life span)*)

B. Tempo total de acesso aos dados por quantidade de instruções

Esses dois últimos gráficos, seguindo a mesma linha do hit rate, mantêm proximidade dos valores no tempo de acesso

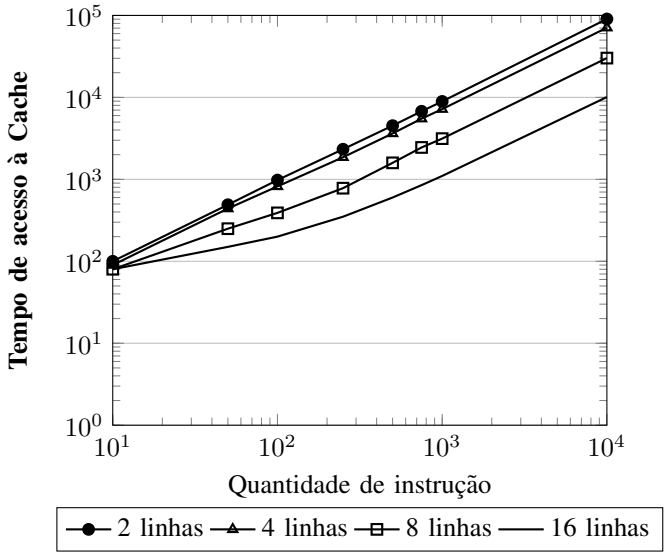

Fig. 3. ***Tempo total de acesso aos dados com algoritmo FIFO***

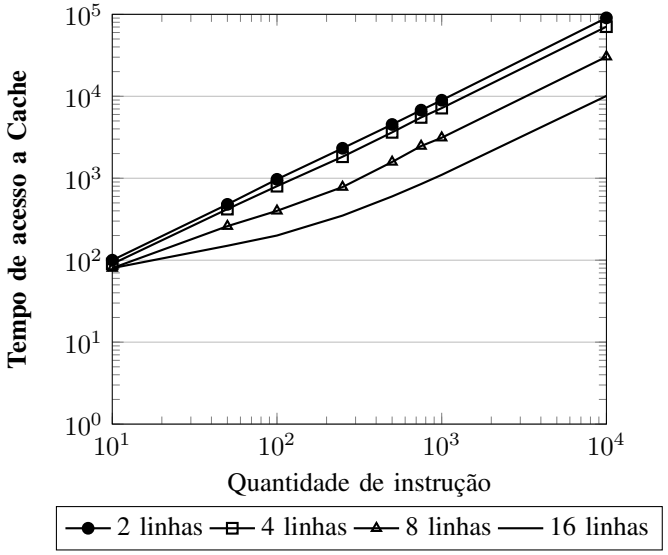

Fig. 4. ***Tempo total de acesso aos dados com algoritmo LRU***

dos dados, desde copiar da RAM para a cache até acessar esse valor no processador. Dentro dos valores que experimentamos, os valores se mantiveram bem próximos. O tempo de acesso aos dados se mostrou fora do padrão apenas no caso da cache com 16 linhas (mesmo tamanho que o processador) houve uma grande queda no tempo de acesso dos dados. Em todos os outros casos, o tempo se manteve muito próximo, mesmo que entre caches de 2 a 8 linhas de tamanho.

Esses dadas nos mostram que para ter algum ganho significativo de performance seria necessário ter uma cache de tamanho próximo ao da RAM. Seria algo completamente impraticável considerando a diferença de preço por unidade de memória entre as duas. O cache se apresenta com gasto de tempo bastante próximos quando em tamanhos usados em processadores comerciais. Demonstrando que independente desses algoritmos possuem um maior hit hate em caches maiores. Dessa forma, aumentar a cache importa mais do que o algoritmo escolhido.

V. Conclusões

Após a realização dos testes em todos os cenários propostos, e uma análise dos gráficos apresentados, houve um receio de que o ponto da pesquisa não havia sido alcançado, já que os resultados se mostraram pouco expressivos. Em nosso caso, tanto o LFU quanto o FIFO tiveram um desempenho com diferença menor que 5% entre eles, em alguns casos nem havendo diferença alguma. Porém, após uma análise e estudo dos artigos correlacionados, foi possível observar que a política de substituição LRU se mostra de forma pouco significativa mais eficiente do que a política FIFO.

Tudo que descobrimos esteve desde Smith e Goodman [**?**] que disseram que não há diferença de desempenho entre os algoritmos e que aleatoriedade é mais eficiente. Enquanto no artigo da Samira Mirbagher Ajorpaz, Elba Garza, Sangam Jindal, e Daniel A. Jiménez [**?**], mostrou que o desempenho do LRU é pouco superior ao FIFO, algo que consegue ser detectado, mas seu uso diário não teria melhora perceptível e que seria mais importante considerar outros tantos algoritmos que possuem um desempenho significativamente superior aos dois anteriores. Por fim, o artigo de Aurore Junier, Damien Hardy, Isabelle Puaut [**?**], mostrou algo inusitado, indo na direção oposta dos outros dois artigos, apresentou dados de que o LRU pode sim ter um desempenho superior à inserção aleatória de dados na cache.

Concluindo, nossos testes demonstraram que não há grande diferença desses algoritmos quando usados em caches pequenas, os dois primeiros artigos concordam com nosso posicionamento, enquanto o outro artigo discordou. Dessa forma, é importante manter essa discussão aberta para novos experimentos. Considerando que cenários de testes diferentes apresentam resultados completamente dispersos.

References

[1] *Proceedings of the 34th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. Gramado, RS, Brazil: IEEE, October 2021.