Alunos: Cleberton de Almeida Pereira **Matrículas: 201933840044**
Thiago Vinicius de Sousa Barroso **201833840030**

Etapa 1 do projeto: Gerador aleatório do ambiente de Wumpus

Conjuntos de regras que foram trabalhadas na inicial do projeto:

- Definir o tamanho do ambiente: o ambiente foi criado com tamanho do mapa maior ou igual a 3 (tamanho >= 3).
- Reservar a posição [0,0] para o agente: a posição (tamanho -1,0) foi reservada para a ser a posição inicial do agente. No caso aqui optamos por reservar o canto inferior esquerdo para o agente iniciar, quando a posição (0,0) seria o canto superior esquerdo.
- Definir de maneira aleatória a posição do ouro: nessa etapa o ouro não poderá ocupar a mesma posição do agente, então quando a posição do ouro é sorteada de maneira aleatória, existe a condição que se a posição do agente for sorteada, deve-se fazer um novo sorteio, até que uma posição diferente do agente seja encontrada.
- Definir de maneira aleatória as posições dos poços: nesta etapa, deve ser seguir algumas regras: não pode haver 2 poços nas casas abaixo(sul) e a direita(leste) do agente, e quando o ouro estiver em algum dos 3 cantos (superior direito e inferior direito e inferior esquerdo), não podem estar 2 poços, de modo que seja impossível o agente se deslocar para pegar o ouro. A quantidade de poços foi definida a partir do cálculo: Quantidade_de_poços = ((tamanho_do_mapa*tamanho_do_mapa)-1) * 0.2). Onde separamos 20% das caras disponíveis para colocar os poços;
- Definir de maneira aleatória a posição do wumpus: foi usado o mesmo tratamento do ouro e foi implementado que o wumpus não poderá ocupar a mesma casa do poço. Contudo, o wumpus poderá estar na mesma casa que o ouro. Nessa etapa a quantidade de wumpus está relacionado com a quantidade de poços seguindo a linha de cálculo a seguir; quantidade_wumpus = (Quantidade_de_poços * 0.4), separando 40% da quantidade de poços para o wumpus.

Desafio da primeira etapa: bom, um desafio encontrado, foi encontrar uma maneira de não gerar um ambiente impossível de se ganhar. Ex: o ouro não ficar preso entre 2 poços em um canto, ou o próprio agente não ficar preso na sua casa inicial.
Como foi resolvido: para ambientes em que a quantidade de poços é <=3, foi resolvido tratando especificamente o número das posições. Já para ambientes com números de poços maiores que 3, tivemos que limitar a quantidade de poços nas paredes. Porém, pode ser que em algum ambiente gerado, o ouro fique preso no meio.

Etapa 2 do projeto: Agente reativo (Versão 1).

Na etapa dois, foram desenvolvidos métodos baseados em regras para ações do agente, o ambiente foi dividido em 3 partes, sendo cantos, paredes e centro, os cantos para a versão quadrática foi divido em 4 cantos (superior e inferior direito, superior e inferior esquerdo), o ambiente foi dividido também em 4 paredes (parede lateral esquerda e direita e parede superior e inferior) e por fim um centro (todas as demais posições que não sejam cantos e paredes). Bom, seguindo uma hierarquia de regras como mostra na tabela 2, o agente pode se movimentar no ambiente, para essa etapa, seus movimentos são de forma aleatórias, porém suas ações devem seguir as regras como mostrado na tabela. No fim de cada movimento feito, o agente pode receber ou não uma consequência.

Tabela 1. Versão distribuída do ambiente wumpus, para qualquer tamanho de ambiente, incluindo possíveis ações, e seus movimentos.

VERSÃO DE AMBIENTE																					
Local		Percepçoes			Ações															Inventário	
		Brilho	Brisa	Fedor	Mover				Atirar				Pegar							Flecha	
CANTOS	SUPERIOR DIREITO	Sentir Brilho											x								
		Sentir fedor (com flecha)								S		O								1	
		Brisa || Fedor(sem flecha) || Não Sentir				S		O												0	
	SUPERIOR ESQUERDO	Sentir Brilho											x								
		Sentir fedor (com flecha)								S	L									1	
		Brisa || Fedor(sem flecha) || Não Sentir				S	L													0	
	INFERIOR ESQUERDO	(Fedor|| Brisa && Fedor) && com flecha								N	L									1	
		(Brisa && fedor(Sem flecha)) || Fedor(sem flecha) || Brisa|| Não Sentir			N		L													0	
	INFERIOR DIREITO	Sentir Brilho											X								
		Sentir fedor (com flecha)							N			O								1	
		Brisa || Fedor(sem flecha) || Não Sentir			N			O												0	
PAREDES	LATERAL DIREITA	Sentir Brilho											X								
		Sentir fedor (com flecha)							N	S		O								1	
		Brisa || Fedor(sem flecha) || Não Sentir			N	S		O												0	
	LATERAL ESQUERDA	Sentir Brilho											X								
		Sentir fedor (com flecha)							N	S	L									1	
		Brisa || Fedor(sem flecha) || Não Sentir			N	S	L													0	
	SUPERIOR	Sentir Brilho											X								
		Sentir fedor (com flecha)								S	L	O								1	
		Brisa || Fedor(sem flecha) || Não Sentir				S	L	O												0	
	INFERIOR	Sentir Brilho											X								
		Sentir fedor (com flecha)							N		L	O								1	
		Brisa || Fedor(sem flecha) || Não Sentir			N		L	O												0	
CENTRO		Sentir Brilho											X								
		Sentir fedor (com flecha)							N	S	L	O								1	
		Brisa || Fedor(sem flecha) || Não Sentir			N	S	L	O												0	

Tabela 1. Hierarquia de regra usado na etapa 2

Hierarquia de Regras	
1º	Brilho/Pegar
2º	Fedor/Atirar(flecha>=1)
3º	Brisa || Fedor(flecha=0)/Mover

Obs: o agente deve seguir para casa que atirou, para quando a percepção grito for sentida, isso se aplica apenas a regra número 2.

Tabela 2. Consequências que podem ser encontradas depois de cada movimento no ambiente.

Consequencia pós movimentos/ações	
1º	Poço || Wumpus / Morte
2º	Posição (0,0) com ouro/Jogo Ganho
3º	Nada/Continua

Etapa 3 do projeto: Agente reativo (Com memória).

Na etapa 3, foi criado a memória do agente fazendo com que ele tivesse 3 mapas diferentes: Um para as sensações, outro para os elementos que ele tem certeza ou deduziu que estão no mapa e o terceiro para a quantidade de vezes que ele andou em cada casa, o agente guarda na memória a sua última casa que ele andou, assim como sabe o tamanho do mapa.

Com base na memória que ele tem a partir dos mapas, é mais fácil mapear o poço/wumpus, quando o agente atirasse e estivesse na parede, na figura 1 é possível visualizar, se o agente atirasse para posição do poço, ele ouviria o eco, logo saberia que onde ele atirou tem um poço, sabendo que a posição abaixo é uma casa segura pois ele veio de lá, a casa a esquerda é fora do mapa e a direita foi onde ele atirou, deduzindo que ele também estaria sentido fedor, então a posição do monstro seria a única casa restante(cima).

O mesmo se aplica na figura 2, com a diferença que ele escutaria o grito e mataria o monstro, colocando então aquela casa como segura, e deduzindo que o poço estaria na casa à direita.

A lógica se aplica em todas as 4 paredes, mudando apenas as casas avaliadas.

Figura 1 Agente na parede esquerda, atirando no poço.

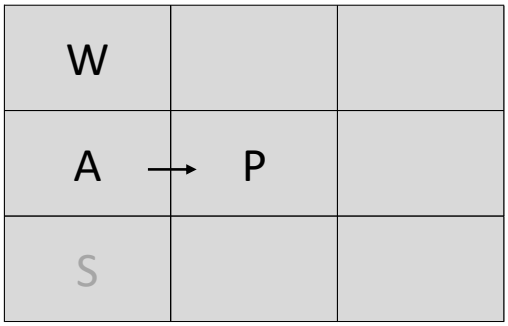
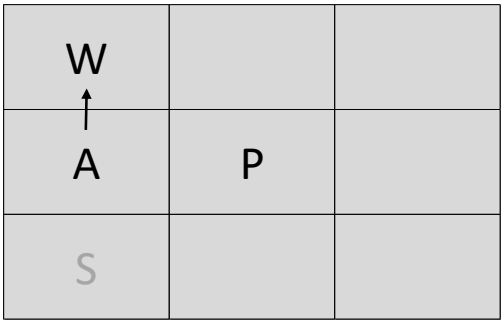


Figura 2 Agente na parede esquerda, atirando no wumpus.



Para os cantos, no exemplo da figura 3, a lógica é que se o agente estiver sentindo fedor, e deduzindo que a posição que ele veio já é segura(S), além de já saber que as posições norte e oeste estão fora do limite do mapa, é possível ter a certeza que o wumpus está na casa leste, podendo assim acertar a flecha.

Figura 3 Agente no canto superior esquerdo sentido fedor.

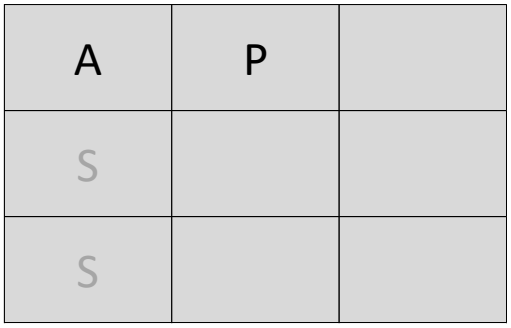
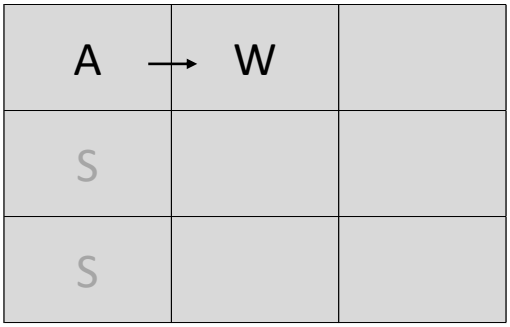


Figura 4 Agente no canto superior esquerdo sentido brisa.



.

No centro, quando o agente tiver flechas e sentir fedor, ele vai atirar, se a direção que atirou ele escutar grito, ele sabe que matou o monstro, como ilustrado na figura 5, então coloca na casa que ele atirou como seguro figura 6. Caso o agente erre o monstro, o próximo movimento dele será para a casa que está segura.

Figura 5 Agente no centro sentindo fedor.

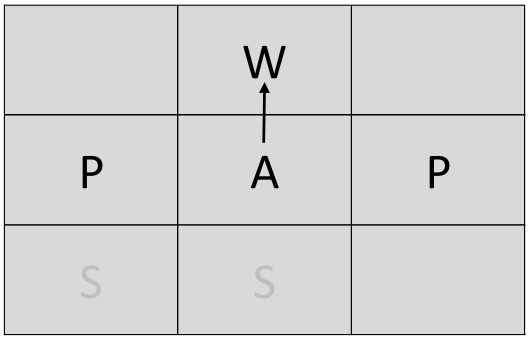
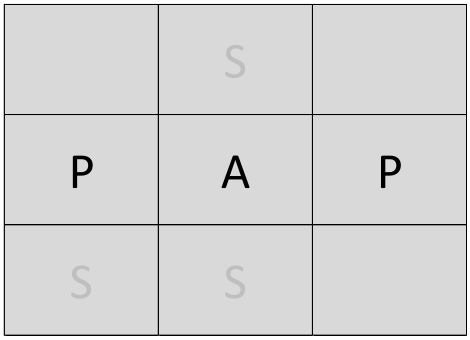
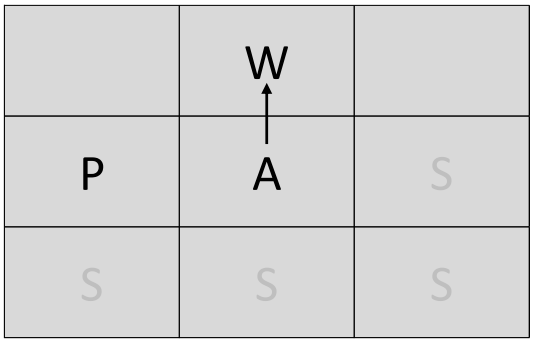


Figura 6 Mapa elementos depois do acerto da flecha.



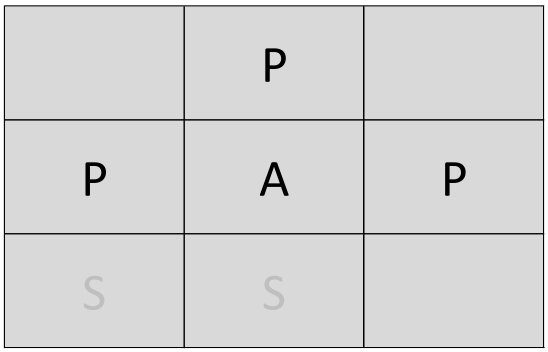
Para a situação de quando o agente estiver no centro e sentir fedor e brisa, ele já mapeia as posições seguras e não atira nelas, no caso da figura 7, se o agente acertar o wumpus ele vai colocar a casa que ele atirou como segura, e quando fizer o mapeamento ao redor o agente vai verificar que 3 casas estão seguras e se ele estiver sentido brisa, ele colocará o poço na única posição restante. E quando ele acerta um poço, o caso se assemelha com a figura 1.

Figura 7 Agente no centro sentindo fedor.



Para a situação em que o agente estiver no centro sentindo brisa, ele irá para a posição segura, ele só vai arriscar o passo diferente quando o contador de passos naquela casa for maior que 5, isso serve para a versão que ele está na parede e sentir brisa.

Figura 8 Agente no centro sentindo brisa.



Um caso específico é quando o agente está em uma casa e não sente nem brisa e nem fedor, ele deduz que nas casas ao redor estão seguras e as preenche como seguras, com exceção das casas fora do limite do mapa, como mostrado nas figuras 9, 10 e 11.

Figura 9 Agente no canto, sentindo nada.

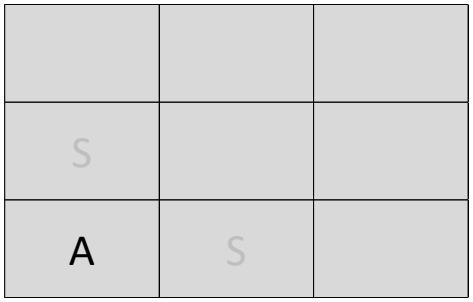


Figura 10 Agente na parede, sentindo nada.

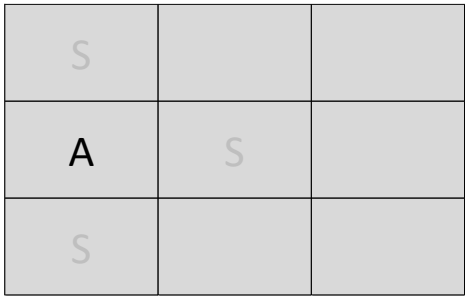
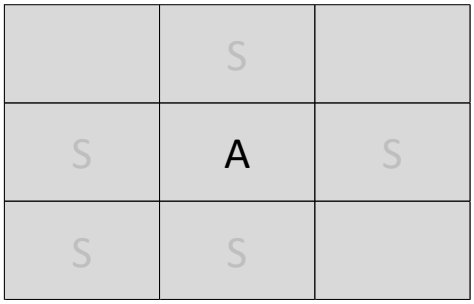


Figura 11 Agente no centro, sentindo nada.



Desafio da terceira etapa: bom, um desafio encontrado, foi que o agente não conseguiu ganhar nenhuma partida, pois ele tinha um comportamento medroso, e andava muito nas casas seguras, e dificilmente arriscava um movimento não seguro. E até então não foi encontrado uma solução.

Etapa 4 do projeto: Agente de aprendizagem.

Treinamento do agente de aprendizagem com uso de Redes Neurais Artificiais

A implementação do agente de aprendizagem foi feita usando redes neurais artificiais, um algoritmo que se assemelha ao funcionamento do cérebro humano. Cada rede é composta por camadas e cada camada é composta por neurônios artificiais, o modelo de rede neural criado é baseado no processo de feedforward chamado de Multilayer Perceptron. Para esse modelo, como entrada, foram usadas algumas variáveis relacionadas ao ambiente e ao próprio agente que a rede controla, sendo elas:

- Posições norte, sul, leste e oeste disponíveis para movimentação;
- Sensações na casa atual tais como fedor, brisa e brilho;
- Atributos do agente que a rede controla como a informação se ele pegou o ouro, matou o wumpus e se ainda tem flecha.

O algoritmo de treino se baseia em simular diversas redes neurais, cada uma com suas diferentes características, e deixar elas jogando até todos os agentes morrerem ou excederem um limite de ações pré estabelecido. Os passos mais detalhados são comentados a seguir:

- Criar vários agentes com redes neurais aleatórias;

- Após todos os agentes terminarem de jogar, calculamos as pontuações para cada agente;
- Selecionamos os melhores indivíduos daquela geração, baseado na sua pontuação (fitness);
- De acordo com o método evolutivo utilizado, criamos uma nova população para jogar a partida;
- Repetimos o processo até conseguir um agente que ganhe o jogo.

Detalhes do algoritmo genético usado

No problema proposto, utilizamos dois métodos de evolução para as gerações, mutações e crossover, em ambos os métodos foram atribuídas taxas de mutação e crossover. Mais detalhes dos algoritmos são:

- Na mutação, nós selecionamos o melhor indivíduo da geração, clonamos a rede neural dele para os seus filhos e cada filho sofrerá uma mutação em cada peso da rede neural dele;
- No método de crossover são selecionados os dois melhores agentes da geração, serializamos as redes em uma forma vetorial, com as redes em formato de vetores é selecionado aleatoriamente um ponto de corte, todos os genes anteriores a esse ponto serão dedicados ao melhor agente e todos os genes acima do corte serão dedicados ao segundo melhor agente, Com isso temos um novo DNA resultante de uma combinação, em seguida convertemos o vetor de dna no modelo de rede neural usado e aplicamos pequenas mutações nele;
- Cada método evolutivo possui seu próprio valor de taxa, essa taxa é responsável por dizer se o indivíduo que estamos escolhendo vai sofrer algum processo evolutivo, por exemplo, caso a taxa seja de 1 (correspondente a 100%), todos os indivíduos sofrerão o processo evolutivo. Para aqueles que não passarem por nenhum desses processos, apenas replicamos o DNA do melhor agente, aplicando o método de elitismo na evolução.

Cálculo da pontuação do agente

Para evitar comportamentos inesperados, o agente possui algumas variáveis na hora de calcular a sua pontuação(fitness).

- A pontuação do agente é aumentada caso ele explore uma casa que não tinha ido anteriormente, caso ele ande muitas vezes na mesma casa, esse comportamento resultará numa penalidade na pontuação dele. O mesmo vale para as vezes que ele tentar andar para fora dos limites do mapa, é calculado quantas vezes ele "bateu na parede" e também aplicamos uma penalização.
- Quando o agente atirar, iremos verificar quatro condições: se o agente acertou o Wumpus, se o agente tentou atirar sem ter flecha, se o agente atirou numa parede (fora dos limites do mapa) e se o agente atirou numa casa qualquer sem o monstro.
- Caso o agente decida pegar numa casa que contém o ouro, sua pontuação aumentará drasticamente, também aplicamos uma penalidade caso o agente decida pegar o ouro numa casa que não tem o ouro.
- Por último, quando o agente já possuir o ouro e estiver de volta na casa de origem, sua pontuação aumentará drasticamente.

A partir desses critérios, nós tentaremos moldar um comportamento inteligente para o agente, fazendo ele explorar mais o mapa, evitar bater nas paredes, andar menos em círculos, atirar somente quando necessário e pegar o ouro somente quando necessário. Com isso, exigimos que

o algoritmo de treino replique os genes dos agentes que tiveram as melhores decisões baseadas no objetivo proposto.

Mais informações sobre a rede

Alguns dados mais técnicos da melhor configuração que conseguimos até agora foram:

- Dez neurônios na camada de entrada para os dados do ambiente e do agente;
- Três camadas ocultas com 9 neurônios cada;
- Nove neurônios na camada de saída para cada ação que o agente pode executar;
- Função de ativação usada nas camadas ocultas foi a tangente hiperbólica;
- Função de ativação usada na camada de saída foi a Argmax;
- O valor dos pesos aleatórios da primeira geração de redes varia entre -100 e 100;
- Bias como um neurônio adicional em cada camada, com exceção da camada de saída;

Desafio da quarta etapa: bom, um desafio encontrado, é que a rede só consegue jogar no mapa que ela aprendeu, para novos mapas é necessário repetir o treino no mapa novo. solução.