UB Social

CS50AI with Python

4.Minesweeper

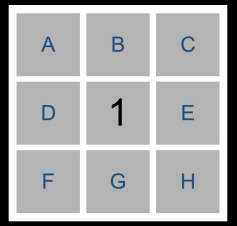
Mateus Schwede

Problemática

- Escrever IA para jogar Minesweeper (Campo Minado);
- Minesweeper é jogo de quebra-cabeça que consiste em grade de células, onde algumas das células contêm "minas" escondidas. Clicar sob célula que contém mina detona-a, e usuário perde o jogo. Clicar sob célula "segura" (célula que não contém mina) revela nº que indica quantas células vizinhas - onde vizinho é célula que está 1 quadrado à esquerda, direita, para cima, para baixo ou diagonal da célula fornecida - contém mina;
- Neste jogo 3x3, por exemplo, os 3 valores 1 indicam que cada célula tem célula vizinha que é mina. Os 4 valores 0 indicam que cada célula não tem nenhuma mina vizinha;
- Com isso, jogador lógico pode concluir que deve haver 1 mina na célula inferior direita e que não há nenhuma mina na célula superior esquerda, pois somente nesse caso os rótulos numéricos em cada uma das outras células são precisos;
- Objetivo do jogo é sinalizar (identificar) cada uma das minas. Em muitas implementações do jogo, incluindo neste projeto, jogador pode sinalizar mina clicando com botão direito em célula;

Problemática

- Objetivo neste projeto é construir IA que possa jogar Campo Minado. Agentes baseados em conhecimento tomam decisões considerando sua base de conhecimento e fazendo inferências com base nesse conhecimento;
- Uma maneira de representar conhecimento de IA sobre o jogo de é tornando cada célula uma variável proposicional, que é verdadeira se célula contém mina e falsa caso contrário;
- IA saberá toda vez que célula segura fosse clicada e verá nº dessa célula. Considere o tabuleiro, onde célula do meio foi revelada, e outras células foram rotuladas com letra de identificação para fins de discussão;
- 1 das 8 células vizinhas é mina, cuja expressão lógica indica que 1 das células vizinhas é mina:
    - Or(A, B, C, D, E, F, G, H)



Problemática

- Mas sabe-se mais do que a expressão. Sentença lógica expressou ideia de que pelo menos 1 das 8 variáveis é verdadeira. Mas pode-se afirmar que exatamente 1 das 8 variáveis é verdadeira, conforme sentença lógica:

```
Or(
   And(A, Not(B), Not(C), Not(D), Not(E), Not(F), Not(G), Not(H)),
   And(Not(A), B, Not(C), Not(D), Not(E), Not(F), Not(G), Not(H)),
   And(Not(A), Not(B), C, Not(D), Not(E), Not(F), Not(G), Not(H)),
   And(Not(A), Not(B), Not(C), D, Not(E), Not(F), Not(G), Not(H)),
   And(Not(A), Not(B), Not(C), Not(D), E, Not(F), Not(G), Not(H)),
   And(Not(A), Not(B), Not(C), Not(D), Not(E), F, Not(G), Not(H)),
   And(Not(A), Not(B), Not(C), Not(D), Not(E), Not(F), G, Not(H)),
   And(Not(A), Not(B), Not(C), Not(D), Not(E), Not(F), Not(G), H)
)
```

Problemática

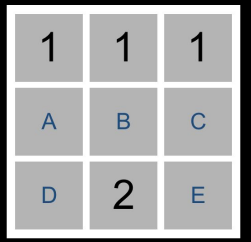
- Utilizar model-checking neste tipo de problema torna-se intratável;
- Então, representar-se cada sentença lógica da IA dessa forma:
    - {A, B, C, D, E, F, G, H} = 1
- Cada sentença lógica nesta representação tem 2 partes: conjunto de células no tabuleiro que estão envolvidas na sentença, e contagem numérica, representando contagem de quantas dessas células são minas. Sentença lógica citada informa que das células A, B, C, D, E, F, G e H, exatamente 1 delas é mina;
- Via conhecimento do nº inferior esquerdo, constrói-se sentença {D, E, G} = 0 para significar que das células D, E e G, exatamente 0 delas são minas. Intuitivamente, pode-se inferir dessa sentença que todas células devem estar seguras. Então, sempre que há sentença cuja contagem é 0, sabe-se que todas células da sentença estão seguras;
- IA construirá sentença {E, F, H} = 3. Pode-se inferir que todos E, F e H são minas. Sempre que nº de células for igual à contagem, sabe-se que todas células dessa sentença são minas;
- Uma vez que sabe-se se célula é mina ou não, pode-se atualizar sentenças para simplificá-las e potencialmente tirar novas conclusões;

A	B	C
D	E	F
0	G	H

A	B	C
D	E	F
G	H	3

Problemática

- Exemplo, se IA souber sentença {A, B, C} = 2, ainda não há informações suficientes para conclusões. Mas ao informar que C é seguro, pode-se remover C da sentença completamente, deixando sentença {A, B} = 2, possibilitando conclusões;
- Se IA souber sentença {A, B, C} = 2, informando que C é mina, pode-se remover C da sentença e diminuir valor de contagem (já que C era mina que contribuiu para contagem), concluindo sentença {A, B} = 1. Pois, se 2 de A, B e C são minas, e sabe-se que C é mina, então A ou B será mina;
- Em 2 sentenças onde IA entende com base na célula central superior e célula central inferior. Da célula central superior, tem-se {A, B, C} = 1. Da célula central inferior, tem-se {A, B, C, D, E} = 2. Logicamente, pode-se inferir novo pedaço de conhecimento, que {D, E} = 1. Afinal, se 2 de A, B, C, D e E são minas, e apenas 1 de A, B e C são minas, então é lógico que exatamente 1 de D e E deve ser outra mina;
- Sempre que há 2 sentenças set1 = count1 e set2 = count2 onde set1 é subconjunto de set2, então pode-se construir nova sentença set2 - set1 = count2 - count1;



Instruções

- Baixe código de https://cdn.cs50.net/ai/2023/x/projects/1/minesweeper.zip e descompacte-o.

Funcionamento

- 2 arquivos: runner.py e minesweeper.py. minesweeper.py contém toda lógica do jogo em si para IA jogar. runner.py contém todo código para executar interface gráfica do jogo. Após concluir todas funções necessárias em minesweeper.py, pode-se executar runner.py para jogar Minesweeper;
- minesweeper.py possui 3 classes, Minesweeper, que lida com jogabilidade; Sentence, que representa frase lógica que contém conjunto de células e contagem; e MinesweeperAI, que lida com inferência de quais movimentos fazer com base no conhecimento;
- Na classe Minesweeper, cada célula é par (i, j) onde i é linha (variando de 0 a altura - 1) e j é coluna (variando de 0 a largura - 1);
- Classe Sentence será usada para representar sentenças lógicas do formato descrito no Background. Cada sentença possui conjunto de células dentro dela e contagem de quantas dessas células são minas. classe também contém funções known_mines e known_safes para determinar se alguma das células na sentença é conhecida como mina ou segura. também contém funções mark_mine e mark_safe para atualizar sentença em resposta a novas informações sobre célula;

Funcionamento

- Classe MinesweeperAI implementará IA para jogar Minesweeper. Classe AI mantém controle de série de valores. self.moves_made contém conjunto de todas células já clicadas, então IA sabe que não deve usá-las novamente. self.mines contém conjunto de todas células conhecidas como minas. self.safes contém conjunto de todas células conhecidas como seguras. E self.knowledge contém lista de todas sentenças que IA sabe que são verdadeiras;
- Função mark_mine adiciona célula a self.mines, então IA sabe que é mina. Também percorre loop em todas frases no conhecimento da IA e informa cada frase que célula é mina, para que frase possa atualizar-se adequadamente caso contiver informações sobre tal mina. Função mark_safe faz mesma coisa, mas para células seguras;
- Funções restantes, add_knowledge, make_safe_move e make_random_move, devem ser implementadas.

Especificações

- Conclua implementações das classes Sentence e MinesweeperAI em minesweeper.py;
- Classe Sentença, conclua implementações de known_mines, known_safes, mark_mine e mark_safe:
  - Função known_mines deve retornar conjunto de todas células em self.cells que são conhecidas por serem minas;
  - Função known_safes deve retornar conjunto de todas células em self.cells que são conhecidas por serem seguras;
  - Função mark_mine deve 1º verificar se célula está incluídas na sentença:
    - Se célula estiver na sentença, função deve atualizar sentença para que célula não esteja mais nela, mas ainda represente sentença logicamente correta, dado que célula é conhecida por ser mina;
    - Se célula não estiver na sentença, nenhuma ação será necessária.

Especificações

- Função mark_safe deve 1º verificar se cell é célula incluída na sentença:
  - Se célula estiver na sentença, função deve atualizar sentença para que célula não esteja mais na sentença, mas ainda represente sentença logicamente correta, dado que célula é conhecida por ser segura;
  - Se célula não estiver na sentença, nenhuma ação será necessária.

- Classe MinesweeperAI, conclua implementações de add_knowledge, make_safe_move e make_random_move:
  - add_knowledge deve aceitar célula (representada como tupla (i, j)) e sua contagem correspondente, e atualizar self.mines, self.safes, self.moves_made e self.knowledge com qualquer nova informação que IA possa inferir, dado que célula é conhecida por ser segura com contagem de minas vizinhas;
    - Função deve marcar célula como um dos movimentos feitos no jogo;
    - Função deve marcar célula como segura, atualizando também todas sentenças que a contêm;

Especificações

- Função deve adicionar nova sentença à base de conhecimento da IA, com base no valor de cell e count, indicando que count dos vizinhos da célula são minas. Certifique-se de incluir apenas células cujo estado ainda seja indeterminado na sentença;
- Se, com base em qualquer uma das sentenças em self.knowledge, novas células podem ser marcadas como seguras ou minas, então função deve fazê-lo;
- Se, com base em qualquer uma das sentenças em self.knowledge, novas sentenças puderem ser inferidas (via método de subconjunto descrito no Background), então essas sentenças devem ser adicionadas à base de conhecimento também;
- Sempre que você fizer qualquer alteração no conhecimento da IA, pode ser possível tirar novas inferências que não eram possíveis antes. Certifique-se de que tais novas inferências sejam adicionadas à base de conhecimento, se for possível fazê-lo.

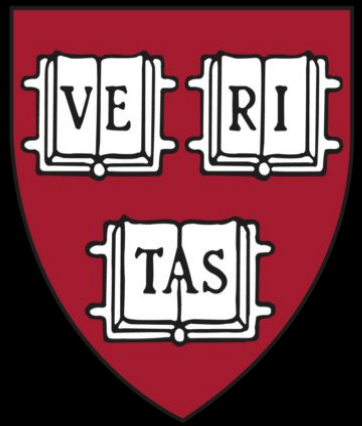
- make_safe_move deve retornar movimento (i, j) que seja conhecido por ser seguro;
  - Movimento retornado deve ser conhecido como seguro, e não movimento já feito;
  - Se nenhuma movimentação segura puder ser garantida, função deve retornar None;
  - Função não deve modificar self.moves_made, self.mines, self.safes ou self.knowledge.

Especificações

- make_random_move deve retornar movimento aleatório (i, j):
  - Função será chamada se movimento seguro não for possível: se IA não souber para onde se mover, ela escolherá mover-se aleatoriamente;
  - Movimento não deve ser movimento já sido feito;
  - Movimento não deve ser movimento que seja conhecido por ser mina;
  - Se nenhum movimento for possível, função deverá retornar None.

Submissão

- Visual Studio Code online: https://cs50.dev
- Testar precisão da lógica do algoritmo: check50 ai50/projects/2024/x/minesweeper
- Testar estilização do código: style50 minesweeper.py
- Para submissão:
    - Em https://submit.cs50.io/invites/d03c31aef1984c29b5e7b268c3a87b7b, entre com GitHub e autorize CS50;
    - Instale pacote Git, Python 3 (e pip), instalando pacotes: pip3 install style50 check50 submit50
    - Submeta o projeto: submit50 ai50/projects/2024/x/minesweeper
- Verificar avaliação: https://cs50.me/cs50ai.



UB Social

Mateus Schwede
HBS ID 202400167108 - DCE ID @00963203