Projeto Nº 1: Época Normal

Inteligência Artificial - Escola Superior de Tecnologia de Setúbal
2021/2022

Prof. Joaquim Filipe
Eng. Filipe Mariano

1. Descrição do Jogo

O Blokus é um jogo de estratégia, que na sua versão original se desenrola num tabuleiro quadrado de `20` linhas por `20` colunas, perfazendo um total de `400` quadrados. É jogado com um total de `89` peças, organizadas em `21` formas coloridas por jogador, ocupando entre um a cinco quadrados cada tipo de peça.

1.1. Regras

O jogo desenrola-se da seguinte forma:

- Os jogadores escolhem uma das suas peças e colocam-nas de modo a que um dos quadrados da peça cubra um dos quadrados de canto do tabuleiro de jogo (posição inicial).
- As jogadas são feitas à vez e, em cada turno, o jogador coloca uma peça de modo a que toque pelo menos numa das suas peças já existente, **mas apenas nos cantos**. Peças do mesmo jogador **nunca se podem tocar nas laterais**, mas podem tocar lateralmente em outras peças (Figura 1).
- Uma vez colocada, a posição da peça não poderá ser alterada até ao final do jogo.
- Quando um dos jogadores não consegue colocar uma peça no tabuleiro de jogo, deverá passar a vez.
- O jogo termina quando nenhum dos jogadores consegue colocar mais peças.
- É definido um objetivo para o jogador atingir, em termos do número mínimo de casas preenchidas.

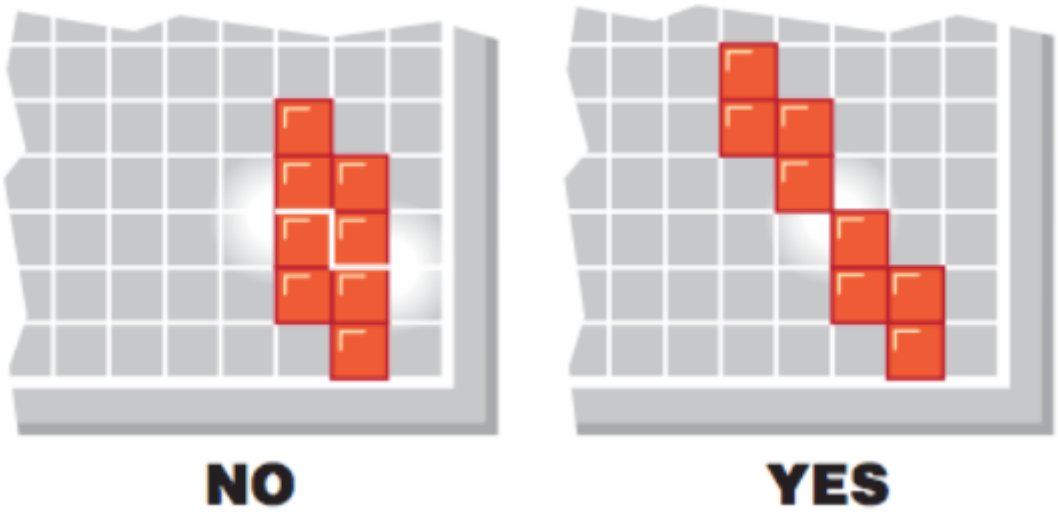

Figura 1: Regra de posicionamento das peças.

1.2. Determinar o vencedor

Quando o jogo termina, os jogadores contam o número de quadrados existentes nas peças que restaram, e o jogador que tiver **o menor** número de quadrados é o vencedor.

2. Objetivo do Projeto

No âmbito do projeto vamos considerar uma versão simplificada do jogo Blokus, denominada Blokus Uno, cujas diferenças a nível de regras são:

- Existem `35` peças de três tipos (Figura 2). Sendo que a peça `C` pode ser colocada em duas posições distintas.
- O tabuleiro tem apenas `14` linhas por `14` colunas (Figura 3).
- Existe apenas um jogador.
- O jogo termina quando todas as peças tiverem sido colocadas ou não for possível colocar mais peças respeitando as regras.

Pretende-se desenvolver um programa, em Lisp, que indique a sequência de passos que conduzem de uma posição inicial do tabuleiro (contendo posições previamente preenchidas), até uma posição final em que o objetivo é atingido.

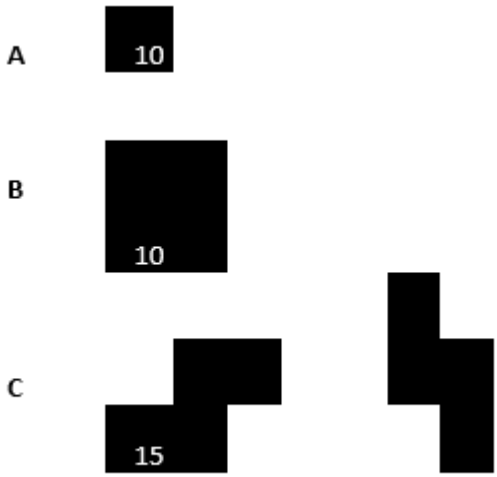

Figura 2: Conjunto e número de peças existentes no Blokus Uno. Sendo que a peça C pode ser colocada em duas posições distintas.

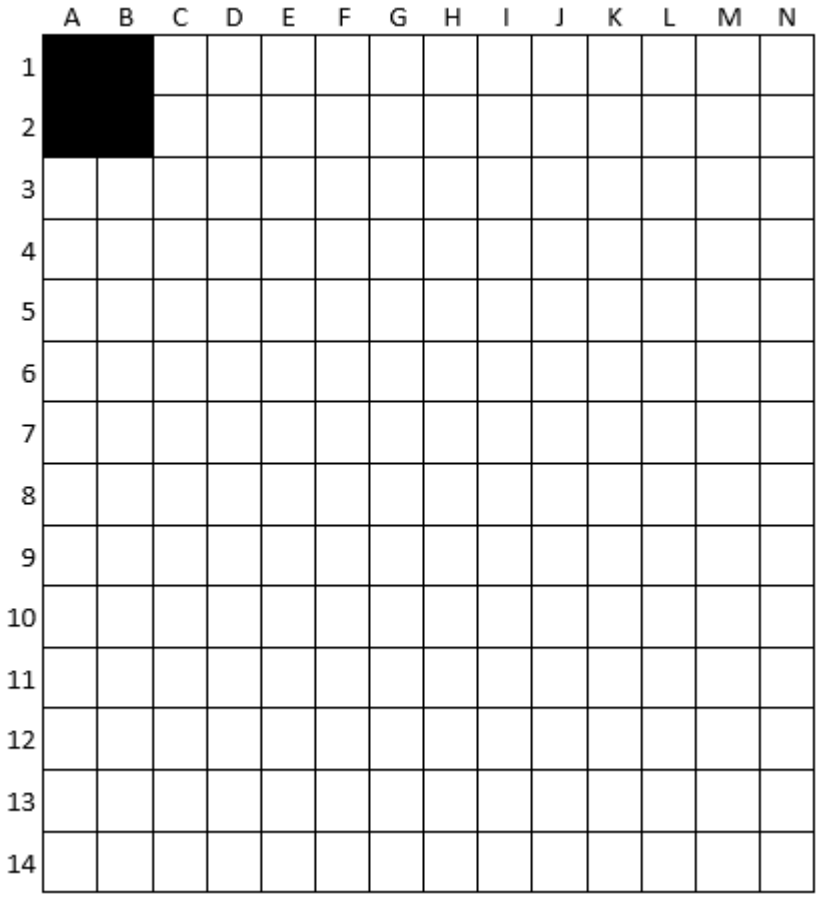

Figura 3: Exemplo de tabuleiro vazio no Blokus Uno com uma peça do jogador numa posição inicial.

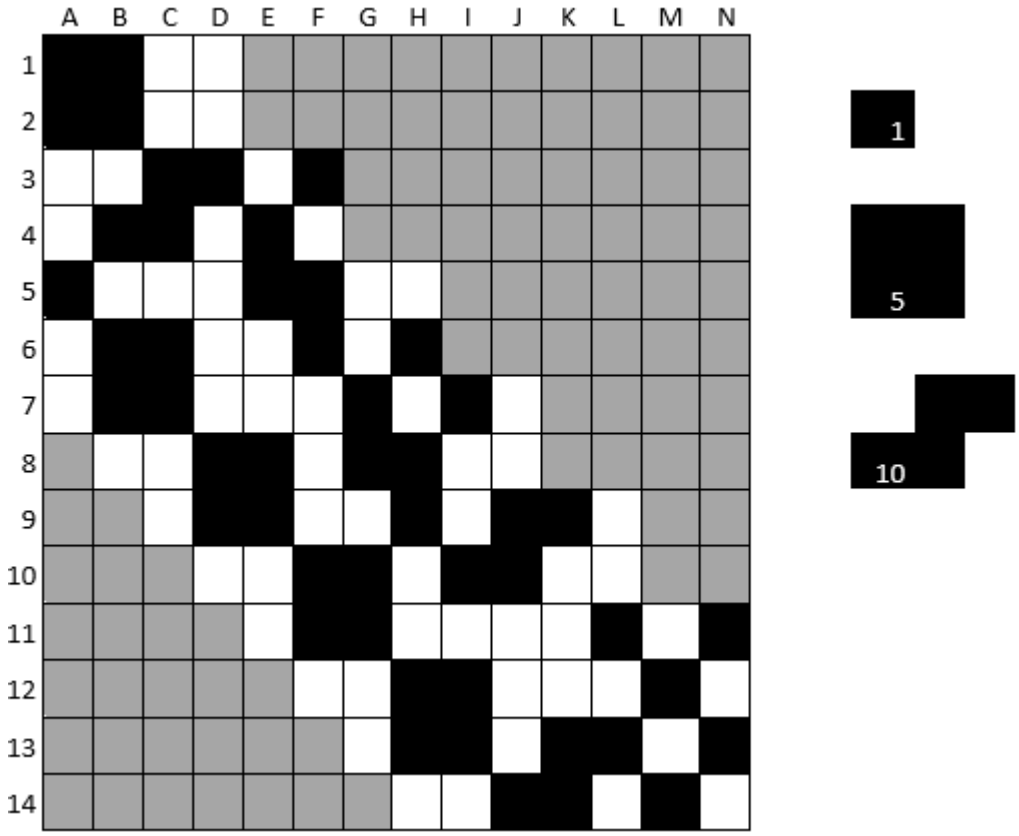

Figura 4: Exemplo de tabuleiro de jogo, finalizado no Blokus Uno, com a indicação de quantas peças sobraram de cada tipo.

Nota: As zonas cinzentas indicam manchas pré-preenchidas e onde não podem ser colocadas peças.

3. Formulação do Problema

3.1. Tabuleiro

O tabuleiro é representado sob a forma de uma lista composta por `14` outras listas, cada uma delas com `14` átomos. Cada uma das listas representa uma linha do tabuleiro, enquanto que cada um dos átomos representa uma coluna dessa linha.

Uma posição linha/coluna, contendo o valor `0` simboliza um quadrado vazio. Caso contenha o valor `1` significa que está ocupado por uma peça do jogador, e caso contenha o valor `2` significa que está ocupado por uma casa pré-preenchida.

Abaixo mostram-se respectivamente a representação do tabuleiro vazio (Figura 5) e do tabuleiro finalizado apresentado na Figura 4 (Figura 6).

```
(
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0)
```

```
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0)
)
```

Figura 5: Representação do tabuleiro vazio.

```
(
 (1 1 0 0 2 2 2 2 2 2 2 2 2 2)
 (1 1 0 0 2 2 2 2 2 2 2 2 2 2)
 (0 0 1 1 0 1 2 2 2 2 2 2 2 2)
 (0 1 1 0 1 0 2 2 2 2 2 2 2 2)
 (1 0 0 0 1 1 0 0 2 2 2 2 2 2)
 (0 1 1 0 0 1 0 1 2 2 2 2 2 2)
 (0 1 1 0 0 0 1 0 1 0 2 2 2 2)
 (2 0 0 1 1 0 1 1 0 0 2 2 2 2)
 (2 2 0 1 1 0 0 1 0 1 1 0 2 2)
 (2 2 2 0 0 1 1 0 1 1 0 0 2 2)
 (2 2 2 2 0 1 1 0 0 0 0 1 0 1)
 (2 2 2 2 2 0 0 1 1 0 0 0 1 0)
 (2 2 2 2 2 2 0 1 1 0 1 1 0 1)
 (2 2 2 2 2 2 2 0 0 1 1 0 1 0)
)
```

Figura 6: Representação do tabuleiro finalizado apresentado na Figura 4.

3.2. Peças

Existem 3 tipos de peças, com as formas e quantidade indicadas na Figura 2. As mesmas podem ser geridas pelo programa da forma que entender ser mais apropriada, contudo deverão respeitar as regras de posicionamento indicadas na Figura 1.

A terceira peça tem dois modos distintos de ser colocada, como foi apresentado anteriormente na Figura 2.

3.3. Operador inicial em tabuleiro vazio

Foi referido anteriormente a forma como se podem colocar as peças no tabuleiro mediante outras já colocadas, contudo há que ter também especial atenção na primeira peça que será colocada. Obrigatoriamente, a casa do canto superior esquerdo (casa 1A) tem de ficar preenchida, independentemente da peça que seja colocada, na jogada inicial.

3.4. Estrutura do programa

O programa deverá estar dividido em três partes, cada uma num ficheiro diferente:

1. Uma parte com a implementação dos métodos de procura, de forma independente do domínio de aplicação;
2. Outra para implementar a resolução do problema, incluindo a definição dos operadores e heurísticas, específicos do domínio de aplicação;
3. E a terceira parte para fazer a interação com o utilizador e para proceder à escrita e leitura de ficheiros.

Enquanto a primeira parte do programa deverá ser genérica para qualquer problema que possa ser resolvido com base no método de procura selecionado, a segunda parte é específico do domínio de aplicação, nomeadamente o problema que este projeto se propõe resolver.

O projeto deverá apresentar um estudo comparativo do comportamento dos três métodos seguintes, conforme explicado na secção 4: procura em largura (BFS), procura em profundidade (DFS) e A*.

Para além destas três formas de procura, cada grupo pode programar, aplicar e estudar os algoritmos Simplified Memory A* (SMA*), Interactive Deepening A* (IDA*) e Recursive BestFirst Search (RBFS), que representarão um bónus para quem os implementar (ver Secção 7).

No caso dos métodos informados, o programa deverá utilizar funções heurísticas modulares, ou seja, que possam ser colocadas ou retiradas do programa de procura como módulos.

As heurísticas não devem estar embutidas de forma rígida no programa de procura. Exige–se a utilização de duas heurísticas, uma fornecida no fim do presente documento e outra desenvolvida pelos alunos.

O projeto deverá incluir a implementação de cada um dos métodos, de forma modular, permitindo que o utilizador escolha qualquer um deles, conjuntamente com os seus parâmetros (heurística, profundidade, etc.) para a resolução de um dado problema.

4. Experiências e Estudos

Pretende-se que o projeto estude, para cada problema fornecido em anexo, o desempenho de cada algoritmo e, no caso dos algoritmos de procura informados, de cada uma das heurísticas propostas, apresentando, em relação a cada problema, a solução encontrada e dados estatísticos sobre a sua eficiência, nomeadamente o fator de ramificação média, o número de nós gerados, número de nós expandidos, a penetrância, o tempo de execução e o caminho até à solução.

Os projetos deverão apresentar os dados acima referidos num ficheiro produzido automaticamente pelo programa, sendo descontado `0,5` valor por cada problema não resolvido. No caso de ser apresentada a solução, mas não o estudo de desempenho das heurísticas o desconto é de apenas `0,2` valor por cada caso.

Estes problemas deverão estar num ficheiro `problemas.dat`, utilizando a notação atrás indicada. O último problema (`G`) será apresentado durante a avaliação oral e inserido no ficheiro `problemas.dat` para verificar o funcionamento do projeto.

5. Heurísticas

Sugere-se usar como heurística de base, uma heurística que privilegia os tabuleiros com o maior número de quadrados preenchidos. Para um determinado tabuleiro *x*:

$h(x) = o(x) - c(x)$

em que:

- *o(x)* é o objetivo para esse tabuleiro: o número de quadrados a preencher no tabuleiro *x*,
- *c(x)* é o número de quadrados já preenchidos no tabuleiro *x*.

Esta heurística pode ser melhorada para refletir de forma mais adequada o conhecimento acerca do puzzle e assim contribuir para uma maior eficiência dos algoritmos de procura informados.

Além da heurística acima sugerida, deve ser definida pelo menos uma segunda heurística que deverá melhorar o desempenho dos algoritmos de procura informados em relação à primeira fornecida.

6. Grupos

Os projetos deverão ser realizados em grupos de, no máximo, duas pessoas sendo contudo sempre sujeitos a avaliação oral individual para confirmação da capacidade de compreensão dos algoritmos e de desenvolvimento de código em LISP.

O grupo poderá ser constituido por alunos que frequentam turmas ou turnos diferentes.

7. Bónus

O projeto inclui uma parte opcional que permite atribuir um bónus aos alunos que a conseguirem implementar.

Para além dos três métodos de procura anteriormente mencionados, cada grupo tem a opção de programar, aplicar e estudar uma ou mais das seguintes estratégias *Simplified Memory–Bounded A** (SMA*), *Iterative Deepening A** (IDA*) ou *Recursive Best First Search* (RBFS).

A programação e estudo dos métodos opcionais vale 1 valor cada, a somar ao valor total do projeto, sendo a nota final do projeto limitada ao máximo de 20 valores.

8. Datas

Entrega do projeto: 17 de Dezembro de 2021, até as 23:00.
Discussão do projeto: Inicio de Fevereiro de 2022, após a entrega da 2ª fase do projeto.

9. Documentação a entregar

A entrega do projeto e da respetiva documentação deverá ser feita através do Moodle, na zona do evento "Entrega do 1º Projeto". Todos os ficheiros a entregar deverão ser devidamente arquivados num ficheiro comprimido (`ZIP` com um tamanho máximo de 5Mb), até à data acima indicada. O nome do arquivo deve seguir a estrutura `nomeAluno1_numeroAluno1_nomeAluno2_numeroAluno2_P1`.

9.1. Código fonte

Os ficheiros de código devem ser devidamente comentados e organizados da seguinte forma:

projeto.lisp Carrega os outros ficheiros de código, escreve e lê ficheiros, e trata da interação com o utilizador.

puzzle.lisp Código relacionado com o problema.

procura.lisp Deve conter a implementação de:

1. Algoritmo de Procura de Largura Primeiro (BFS)

2. Algoritmo de Procura do Profundidade Primeiro (DFS)
3. Algoritmo de Procura do Melhor Primeiro (A*)
4. Os algoritmos SMA*, IDA* e/ou RBFS (caso optem por implementar o bónus)

9.2. Exercícios

Deverá haver um ficheiro de exercícios, com a designação `problemas.dat`, contendo todos os exemplos de tabuleiro que se quiser fornecer ao utilizador, organizados de forma sequencial, e que este deverá escolher mediante um número inserido no interface com o utilizador.

Esse número representa o número de ordem na sequência de exemplos. O ficheiro deverá ter várias listas, separadas umas das outras por um separador legal, e não uma lista de listas. Essas listas serão tantas quantos os problemas fornecidos.

Na oral, os docentes irão solicitar que se adicione mais um exemplo, numa dada posição do ficheiro, que deverá imediatamente passar a ser selecionável através do interface com o utilizador e ser resolvido normalmente.

9.3. Manuais

No âmbito da Unidade Curricular de Inteligência Artificial pretende–se que os alunos pratiquem a escrita de documentos recorrendo à linguagem de marcação ***Markdown***, que é amplamente utilizada para os ficheiros ***ReadMe*** no ***GitHub***. Na Secção 4 do guia de Laboratório nº 2, encontrará toda a informação relativa à estrutura recomendada e sugestões de ferramentas de edição para ***Markdown***.

Para além de entregar os ficheiros de código e de problemas, é necessário elaborar e entregar 2 manuais (o manual de utilizador e o manual técnico), em formato PDF juntamente com os sources em MD, incluídos no arquivo acima referido.

Manual Técnico:

O Manual Técnico deverá conter o algoritmo geral, por partes e devidamente comentado; descrição dos objetos que compõem o projeto, incluindo dados e procedimentos; identificação das limitações e opções técnicas. Deverá ser apresentada uma análise critica dos resultados das execuções do programa, onde deverá transparecer a compreensão das limitações do projeto. Deverão usar uma análise comparativa do conjunto de execuções do programa para cada algoritmo e cada problema, permitindo verificar o desempenho de cada algoritmo e das heurísticas. Deverá, por fim, apresentar a lista dos requisitos do projeto (listados neste documento) que não foram implementados.

Manual de Utilizador:

O Manual do Utilizador deverá conter a identificação dos objetivos do programa, juntamente com descrição geral do seu funcionamento; explicação da forma como se usa o programa (acompanhada de exemplos); descrição da informação necessária e da informação produzida (ecrã/teclado e ficheiros); limitações do programa (do ponto de vista do utilizador, de natureza não técnica).

10. Avaliação

Funcionalidade	Valores

Funcionalidade	Valores
Representação do estado e operadores	2.5
Funções referentes ao puzzle	2.5
Procura em profundidade e largura	2.5
Procura com A* e heurística dada	2.5
Implementação de nova heurística	1.5
Resolução dos problemas a) a f)	3
Resolução do problema g) (dado na avalição oral)	0.5
Qualidade do código	2
Interação com o utilizador	1
Manuais (utilizador e técnico)	2
Total	20
Bónus	3

O valor máximo da nota são 20 valores, já com a integração do valor do bónus.

Os problemas a) a f) estão descritos na Secção Experiências, enquanto que o problema g) será facultado durante a avaliação oral. A avaliação do projeto levará em linha de conta os seguintes aspectos:

- Data de entrega final – Existe uma tolerância de 4 dias em relação ao prazo de entrega, com a penalização de 1 valor por cada dia de atraso. Findo este período a nota do projeto será 0.
- Correção processual da entrega do projeto – (Moodle; manuais no formato correto). Anomalias processuais darão origem a uma penalização que pode ir até 3 valores.
- Qualidade técnica – Objetivos atingidos; Código correto; Facilidade de leitura e manutenção do programa; Opções técnicas corretas.
- Qualidade da documentação – Estrutura e conteúdo dos manuais que acompanham o projeto.
- Avaliação oral – Eficácia e eficiência da exposição; Compreensão das limitações e possibilidades de desenvolvimento do programa. Nesta fase poderá haver lugar a uma revisão total da nota de projeto.

11. Recomendações finais

Com este projeto pretende–se motivar o paradigma de programação funcional. A utilização de variáveis globais, de instruções de atribuição do tipo set, setq, setf, de ciclos, de funções destrutivas ou de quaisquer funções com efeitos laterais é fortemente desincentivada dado que denota normalmente uma baixa qualidade técnica. A sequenciação só será permitida no contexto das funções de leitura/escrita.

As únicas excepções permitidas a estas regras poderão ser a utilização da instrução loop para implementar os ciclos principais dos algoritmos implementados (como alternativa a uma solução puramente recursiva), em conjugação com as variáveis globais Abertos e Fechados para manutenção das listas de nós.

ATENÇÃO: Suspeitas confirmadas de plágio serão penalizadas com a anulação de ambos os projetos envolvidos (fonte e destino), e os responsáveis ficam sujeitos à instauração de processo disciplinar.

Anexos - Problemas

Para resolver os problemas aqui anexados deverá ter em atenção aos seguintes aspetos:

- As zonas cinzentas indicam manchas pré-preenchidas e onde não podem ser colocadas peças.
- Cada tabuleiro possui um objetivo de número de casas mínimo que deve ser preenchido, para encontrar uma solução para o problema.

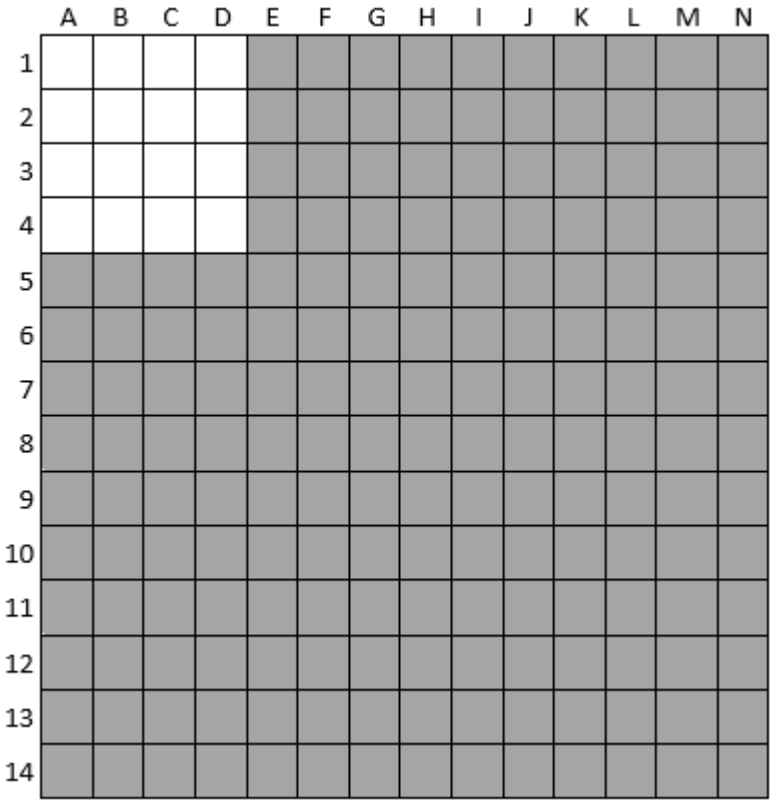

Figura 7: Problema a). Objetivo: (pelo menos) 8 casas preenchidas.

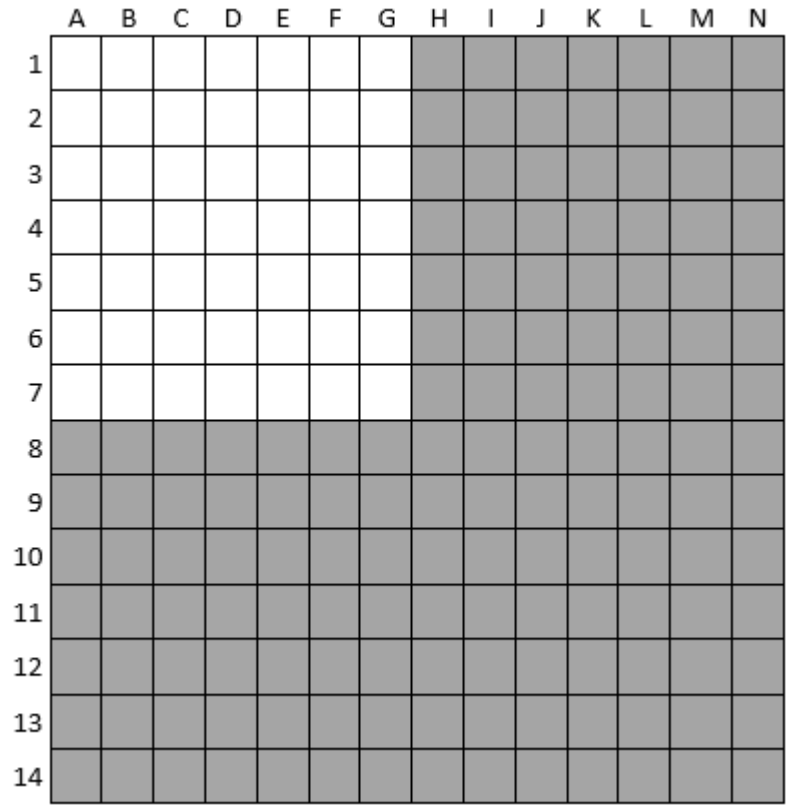

Figura 8: Problema b). Objetivo: (pelo menos) 20 casas preenchidas.

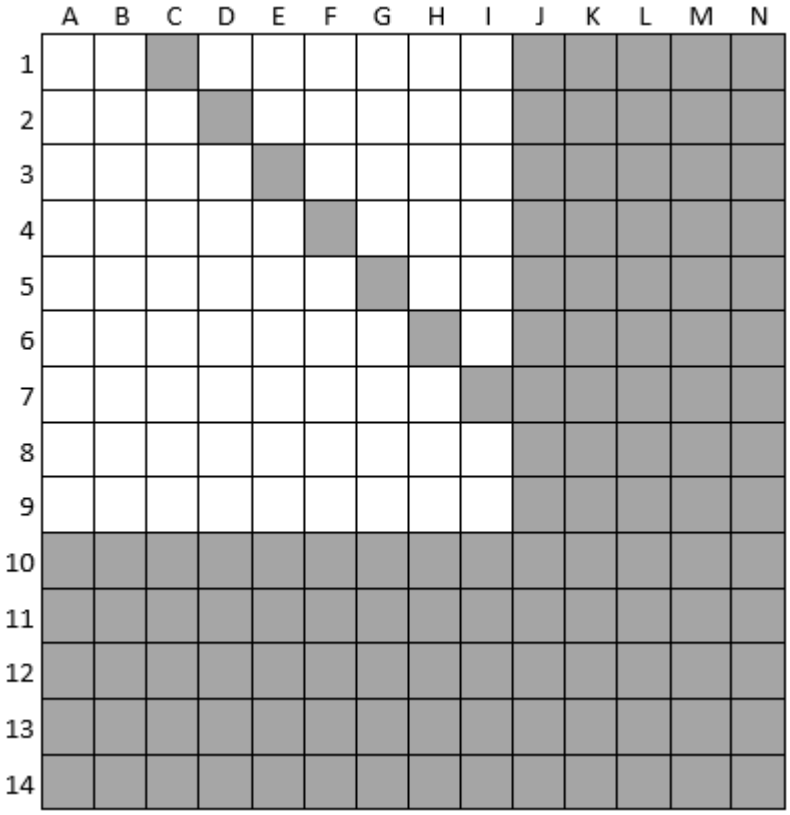

Figura 9: Problema c). Objetivo: (pelo menos) 28 casas preenchidas.

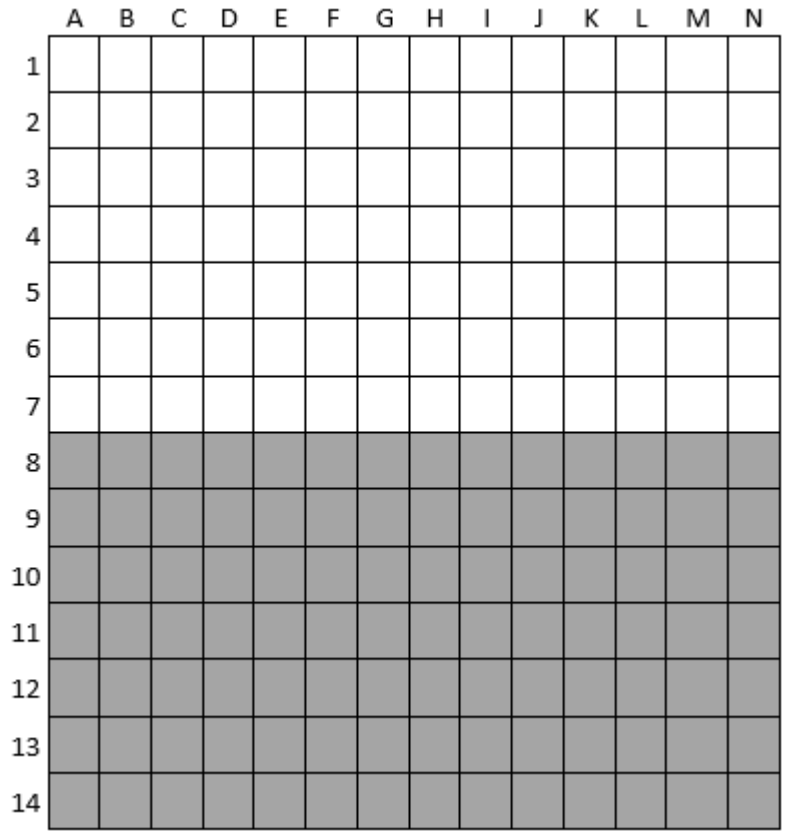

Figura 10: Problema d). Objetivo: (pelo menos) 36 casas preenchidas.

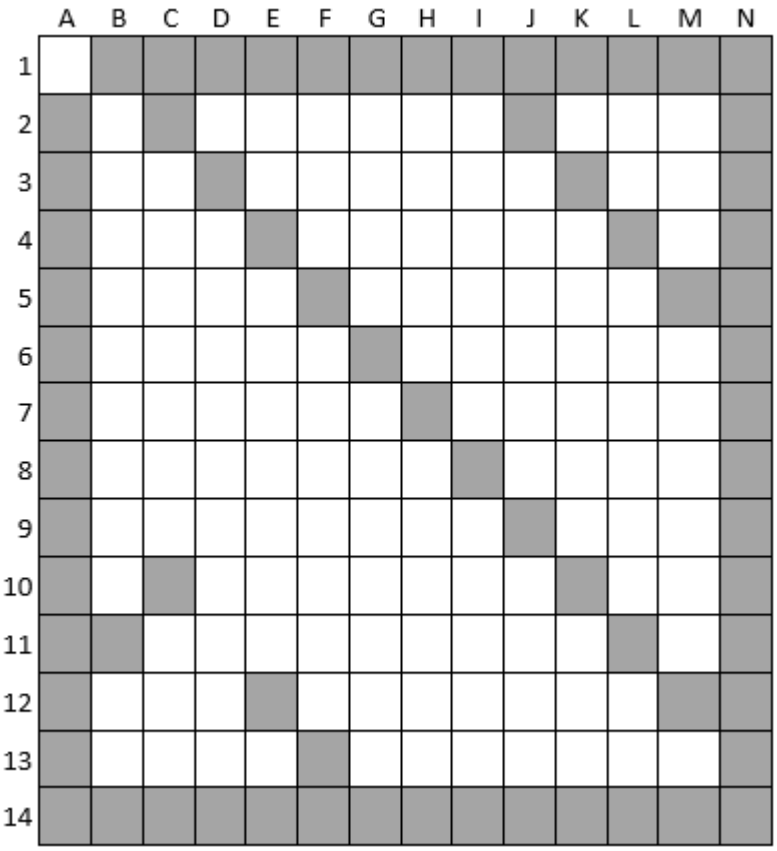

Figura 11: Problema e). Objetivo: (pelo menos) 44 casas preenchidas.

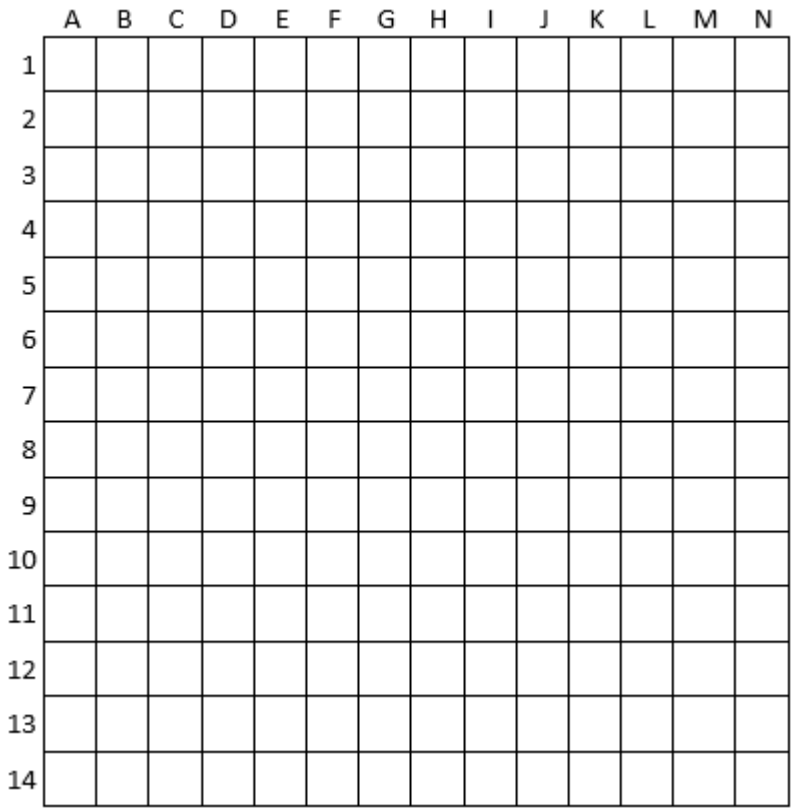

Figura 12: Problema f). Objetivo: (pelo menos) 72 casas preenchidas.