- **Defineix: Variables, Domini i Restriccions**

A la meva implementació, les variables són cada una de les 64 caselles que té el tauler d'escacs on, a cada una d'elles, insereixo l'índex de visita (en el mateix ordre en que el cavall les visita) que va des de 1 a 64.

En quant al domini, aquest canvia en cada "iteració" del *backtraking* (quelcom que no passa, per exemple, en el Sudoku) i és cada una de les caselles a les que es pot moure el cavall en cada moment.

Les restriccions que he aplicat són les del Moviment en forma de "L" del cavall, la de no visitar caselles fora del rang del taulell (no anar més enllà de 8x8) i la de no visitar cap casella que ja hagi sigut visitada anteriorment.

- **Dimensions de l'arbre que haurà de construir-se.**

L'arbre té 8 opcions a cada casella, donat el moviment del cavall, aproximadament (ja que hi ha opcions que cauen fora del taulell depenent d'on és el cavall). Per tant, l'arbre tindria una mida aproximada (sempre menor) de 8^(64-1) (8 opcions a cada una de les caselles menys l'última).

- **Quines tècniques podries aplicar per millorar l'eficiència i com ho faries?**

En el meu cas, vaig aplicar les restriccions, no incloure visitar caselles prèviament en el domini així com no incloure en el domini les caselles que queden fora del tauler. D'aquest manera l'algorisme resolia un tauler 3x4 ràpidament i un 7x7 en una llarga estona de còmput però no era capaç de donar una solució per a un 8x8 en un temps raonable.

Va ser llavors quan vaig afegir una heurística que fa que el domini prioritzi (de fet l'ordena) les caselles que derivaran en un número menor de moviments en el futur. Amb aquest implementació, resol el taulell 8x8 molt ràpidament. (Es pot observar al fitxer *ChessHorse.cpp* al métode *getVisitableCells()*).

Resultat a taulell 3x4	Resultat a taulell 7x7	Resultat a taulell 8x8
[1] [4] [7] [10] [12] [9] [2] [5] [3] [6] [11] [8]	[1] [14] [3] [22] [17] [12] [49] [4] [23] [16] [13] [48] [21] [18] [15] [2] [47] [24] [19] [44] [11] [32] [5] [36] [43] [46] [25] [20] [35] [42] [33] [28] [39] [10] [45] [6] [31] [40] [37] [8] [29] [26] [41] [34] [7] [30] [27] [38] [9]	[1] [34] [3] [18] [49] [32] [13] [16] [4] [19] [56] [33] [14] [17] [50] [31] [57] [2] [35] [48] [55] [52] [15] [12] [20] [5] [60] [53] [36] [47] [30] [51] [41] [58] [37] [46] [61] [54] [11] [26] [6] [21] [42] [59] [38] [27] [64] [29] [43] [40] [23] [8] [45] [62] [25] [10] [22] [7] [44] [39] [24] [9] [28] [63]