Specificaties

We gebruiken drie klasses voor deze opdracht: Punt, Table (van practicumopdracht 2) en Sudoku. De specificaties voor Table zijn letterlijk dezelfde als die in opdracht 2.

|  |
| --- |
| Table |
| -elements : int[][]  //a table consists of n rows, and m columns |
| +Table(int n, int m, int val)  pre: n > 0, m > 0  post: has created a Table object with n rows and m columns, the value of all elements = val  +set(int row, int col, int val)  pre: 0 <= row < n, 0 <= col < m  post: if pre elements[row][col] = val  +get(int row, int col) : int  pre: 0 <= row < n, 0 <= col < m  post: if pre returns elements[row][col], returns -1 otherwise  +numRows() : int  post: has returned the number of rows  +numCols() : int  post: has returned the number of columns  +toString()  post: has returned a String representation of the Table object |

|  |
| --- |
| **Punt**  //We gebruiken het punt-object om systematischer vakjes van de sudoku te kunnen doorlopen en checks uit te voeren; het kan in principe ook zonder dit object, maar wij dachten op deze manier meer verwarring te voorkomen. |
| **-X : int**  **-Y : int** |
| **+Punt(x1: int, y1: int)**  post: er is een Punt-object geconstrueerd met X-coördinaat x1 en Y-coördinaat y1.  **+GetX() : int**  post: retourneert X.  **+GetY() : int**  post: retourneert Y. |

|  |
| --- |
| **Sudoku** |
| **-tabel : int[][]** //Is 9x9 en stelt de sudoku voor.  **-blokken : boolean[][][]**  //blokken gebruiken we als een 3x3 tabel van bitsets; elk vakje van de tabel stelt een 3x3 blok van de sudoku voor, en met de bitset houden we bij welke getallen al in het respectievelijke blok voorkomen. Als bijv. blokken[1][2][3] = false, dan betekent dit dat in het middelste van de bovenste rij vierkanten (rij 1, kolom 2 van een 3x3 tabel) nog geen 3 voorkomt. |
| **+Sudoku()**  post: er is een leeg sudoku-object geconstrueerd met een tabel van  9 rijen en 9 kolommen, en een 3-dimensionale boolean array van 3 bij 3 bij 9.  **+checkKolom(p : Punt, w : int) : boolean**  post: controleert of het getal w voorkomt in de kolom van punt p. Retourneert true als dit wel het geval is, en false als dat niet zo is.  **+checkRij(p : Punt, w : int) : boolean**  post: controleert of het getal w voorkomt in de rij van punt p. Retourneert true als dit wel het geval is, en false als dat niet zo is.  **+nummerVierkant(p: Punt) : Punt**  post: we geven elk blok van 3 bij 3 van de tabel een coordinaat mee; deze methode geeft de coordinaat van het blok waar het punt p in ligt.  **+checkBlok(p : Punt, w : int): boolean**  post: controleert of het getal w voorkomt in het blok van punt p. Retourneert true als dit wel het geval is, en false als dat niet zo is.  (Deze methode maakt gebruik van de methode nummerVierkant())  +**vulIn(a : int[]): boolean**  post: construeert de sudoku gegeven door array a en kijkt bij iedere toevoeging van een getal aan de sudoku of de sudoku überhaupt op te lossen is. Zo niet retourneert false en anders retourneert deze methode true.  **+LosOp(p : Punt)**  pre:de sudoku is ingevuld met behulp van methode vulIn(int[] a)  post: vult de sudoku aan zodat er wel of niet een oplossing ontstaat  //Het idee van deze methode is dat een aanroep van LosOp alle getallen van 1 tot en met 9 uitprobeert in het punt p, totdat een werkende oplossing wordt gevonden. Elk getal wordt eerst door de checks gehaald voordat het wordt ingevuld. Nadat het getal is ingevuld wordt LosOp aangeroepen in het volgende punt; als in dat volgende punt alle getallen van 1 tot en met 9 zonder succes worden geprobeerd, gaat ‘deze’ LosOp verder met getallen proberen totdat er weer een wordt gevonden die door de checks heen komt, etc. Hier zit de backtracking in. |