Lista de chequeo - Plantilla

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ítem** | **Respuesta** | **Check** | **FECHA** | **Porcentaje** |
| Lista de participantes  (Restricciones: debe haber a lo sumo un estudiante MACC 1.0) | Miguel Gutiérrez Vidal y David Felipe Martinez Castiblanco | OK | 17 de agosto  (1%) | 1 |
| Repositorio GitHub | https://github.com/juanmigutierrez/Proyecto-Logica.git | OK | 24 de agosto  (1%) | 1 |
| Situación a representar  (Resumen) | Ubicar n número de reinas en un tablero de n x n de tal manera que no se puedan atacar, empezaremos con n=4 y se ira expandiendo. | OK | 31 de agosto  (2%) | 2 |
| Situación a representar  (archivo en repositorio) | Problema\_Reinas.pdf | OK |
| Representación  (Resumen) | 1. Cuadros del tablero => 16 átomos (V: hay una reina; F: no hay reina) 2. Exactamente cuatro reinas en el tablero => Cláusulas con 16 literales 3. Reglas de ataque de la reina => Fórmulas “Sí …, entonces …” con antecedente casilla donde se ubica la reina y consecuente casillas que no pueden tener otra reina. | OK | 14 de septiembre  (1%) | 1 |
| Representación  (archivo en repositorio) | Claves\_De\_Representacion.pdf | OK |
| Implementación gráfica de la representación  (uso en línea de comando) | -librería pygame necesaria 🡪instalar\_pygame.txt  -reina.png necesarios para correr el código correctamente | OK | (1%) | 1 |
| Implementación gráfica de la representación  (archivo en repositorio) | Codigo Definitivo/Tableaux/Tablero.py  Codigo Definitivo/Tableaux/reina.png  Codigo Definitivo/Tableaux/tablero.PNG | OK |
| Aplicación mediante tableaux  (uso en línea de comando) | * Modificamos los métodos StringtoTree (), y Obtiene literales () en tableaux.py para leer letras con más de un char. * NOTA: **PYTHON 2 NECESARIO(TABLEAUX)** |  | (3%) | 3 |
| Aplicación mediante tableaux  (archivo en repositorio) | Proyecto-Logica/Codigo Definitivo/Tableaux/Reinas(aplicado).py |  |
| Aplicación algoritmo DLL  (uso en línea de comando) | ENTREGA FINAL:   * Profesor corrimos el de las reinas y nos da un error de memoria de recursión (el computador estalla), ya que al utilizar los códigos de quitar negaciones el árbol es muy grande. * A Nosotros el tableaux nos botó las soluciones * Utilizamos de ultima recursión hacer el código para la solución de los caballos en un tablero 3x3, con el caso que usted ya tenía para mostrar que algoritmo DPLL funciona. * Es necesario correr en **versión 3 de Python(DPLL)** ya que si no le va correr "Set" como lista, cuando realmente es un conjunto. |  | 9 de noviembre  (3%) | 3 |
| Aplicación algoritmo DLL  (archivo en repositorio) | * Códigos:  --1) Proyecto-Logica/Codigo Definitivo/DPLL/Caballo(aplicado) - DPLL.py  (DPLL funciona) --2) Proyecto-Logica/Codigo/Reinas(aplicado) - DPLL.py (DPLL funciona pero la recursión estalla) |  |
| Sustentación |  |  | 21 de noviembre | 3 |