**2508ICT Principles of Intelligent Systems**

**Assignment 1: The Knight’s Attack**

Due Date: 5pm Tuesday 3 September (Week 7, Semester 2, 2013).

Campus: Nathan

Weighting: 14% of the total marks for the course

**Student Name:** Rory Hiscock  
**Student Number:** 2776863

**Instructions for use:**

Extract the files from Assignment1\_KnightAttack into their folders in the Aima project.

The files located in Assignment1\_KnightAttack.zip\aima-core\src\main\java\aima\core\environment\ will go into the environment folder under knightspath

The main GUI located in Assignment1\_KnightAttack.zip\aima-gui\src\main\java\aima\gui\applications\search\gameswill go into the games folder under KnightsPathApp.java

Run the KnightsPathApp.java in eclipse. Then select from the drop down boxes at the top of the app which board size, opponent piece, and search algorithm (only Manhattan Distance) you’d like to use. Then hit the Run/Step button on the right hand side to start it. Left clicking on any of the fields will alternate between the pieces so that you may place the pieces in any square you wish.

The files have comments through them to let you know the changes that were made from the original nQueens program.

**Report**

**How you represented the state space of the search:** The state space was represented as a chess board (4x4, 6x6, 8x8, 10x10, 12x12) with a knight piece and an opposing piece selected by the user (bishop, rook, or queen). Each state required the knight piece to move towards its goal state without landing in a square that was being attacked by the opposing piece (if it could not move or was already in its goal state it would return with no action). The goal state of the search was to have the knight piece attacking the opposing piece without being attacked.

**How you implemented the search procedure:** The search procedure looked at the current available moves that the knight could make without landing on an attacked square or out of the boards boundaries. Once the available moves (nodes) had been expanded the A\* heuristic would tell it which of the squares would reach the goal state faster, then move the knight to that square. Then if the knight is still not in the goal state; it would search again from the new position.

**What heuristic you used in the A\* procedure:** For the assignment I used the Manhattan Distance heuristic where you take the x-axis and y-axis of each piece then minus the opposing piece’s axis from the knights axis which gives you a X distance and Y distance difference from each other. Then you add the distance different together to get a whole number, then divide that number by 3 (knights can only move a total of 3 squares in an L shape) to get the A\* admissible heuristic.

**An explanation of why this heuristic is admissible:** An admissible heuristic is admissible if the heuristic distance is not larger than the actual distance. The heuristic used is admissible because its distance not larger than the actual distance between the pieces.

**Experimental results showing two actual attacks executed by your**

**knight, detailing the order in which each move was taken and the**

**heuristic cost of each move:** Starting state (6x6 board) (Knight(0,0)) (Bishop(5,5)) (Manhattan Distance):  
 Knight moves from 0,0 to 1,2 (cost 2) -> Knight moves from 1,2 to 2,4 (cost 1) -> Knight moves from 2,4 to 4,3 (cost 0) -> Knight can take Bishop (goal state).  
 Pathcost = 3.  
 Nodes expanded = 15.  
 Queue size = 12.  
 Max queue size = 17.

Starting state (10x10 board) (Knight 7,3)) (Bishop (1,4)) (Manhattan Distance):  
 Knight moves from 7,3 to 5,2 (cost 1) -> Knight moves from 5,2 to 3,3 (cost 0) -> Knight can take Bishop (goal state).  
 Pathcost = 2.  
 Nodes expanded = 9.  
 Queue size = 24.  
 Max queue size = 25.

**Any extensions you have implemented in your solution:** The left click to alternate between pieces to let you change their positions instead of selecting their starting positions via drop down menu was added in because it’s easier to use.

**Any requirements that were not implemented or that do not work correctly:** The program works.

**Parts of the program that work correctly:** If the knight starts on the goal state the program will not do anything (goal test works)  
 The program finds the shortest route to the opposing piece using the A\* heuristic  
 The program successfully checks if the piece is under attack or out of bounds (demonstrated by moving the pieces into the line of sight of the other piece and the piece turning red)  
 All of the board dimensions and opposing pieces have been added  
 The bishop, rook, and queen have all been added with their move sets and appropriate functions