THE UNIVERSITY OF WESTERN AUSTRALIA
FACULTY OF
Engineering, Computing and Mathematics

School of Computer Science and Software Engineering

Sample Questions 2009

CITS4211 AI

This paper contains: 11 pages (including the title page)
Time allowed: 2 hours 10 minutes

Rubric from the actual paper:
This paper contains 5 questions, each of which is worth 15 marks.
Candidates should attempt FOUR (4) questions.
Marks for this paper total 60.

PLEASE NOTE

Examination candidates may only bring authorised materials into the examination room. If a supervisor finds, during the examination, that you have unauthorised material, in whatever form, in the vicinity of your desk or on your person, whether in the examination room or the toilets or en route to/from the toilets, the matter will be reported to the head of school and disciplinary action will normally be taken against you. This action may result in your being deprived of any credit for this examination or even, in some cases, for the whole unit. This will apply regardless of whether the material has been used at the time it is found.

Therefore, any candidate who has brought any unauthorised material whatsoever into the examination room should declare it to the supervisor immediately. Candidates who are uncertain whether any material is authorised should ask the supervisor for clarification.

Instructions

The rubric for the actual exam paper is as shown on the previous page.

The paper will contain 5 questions.
Candidates should attempt 4.
Each question is worth 15 marks. Marks for the paper total 60.

If more than two questions have been attempted in any section, only the *first* two as submitted will be marked. If you change your mind after starting a question you should put a cross through the whole question and write "Not Attempted".

This paper contains some sample questions. The first (essay question) is a question from the actual exam that you can prepare for in advance. The remainder of the questions are taken from previous exams, or are examples of what could have been exam questions on the current course material. They are provided to give you an idea of the size and flavour of the questions — obviously the real exam questions will be different and will include material that is not covered in the sample questions.

While you should not post answers verbatim on help4211, you are welcome to discuss your understanding of the material.

The exam will cover the lectures, the lecture notes, the lab work and project. It will not cover parts of the text that have not been discussed in lectures. The relevant chapters of the text do however contain a wealth of problems that you can use to test your understanding of the material.

You should *not* consider doing this practice exam as the primary part of your revision for the exam. It is important that you understand and can explain all the material that was covered in the unit. This includes not just descriptive material, but also the technical material (formulas, complexity results, proofs, etc.)

Data Sheet

The following tables provide a reminder of the API for relevant classes used in the laboratories.

`Node` Method Summary

Method Summary	
java.lang.Object	**clone**()
double	**getCost**()
Actions	**getPath**()
State	**getState**()
double	**getUtility**()
void	**setCost**(double cost)
void	**setPath**(Actions path)
void	**setState**(State state)
void	**setUtility**(double utility)
void	**update**(Action action)

`NodeInfo` Method Summary

Method Summary	
double	**getDepthLimit**() Get the depth/cost limit for depth limited and iterative deepening searches.
boolean	**isGoal**(Node node)
boolean	**isTerminal**(Node node)
void	**setDepthLimit**(double limit) Set the depth/cost limit for depth limited and iterative deepending searches.
double	**utility**(Node node)

`State` Method Summary

Method Summary	
java.lang.Object	clone()
Actions	getActions()
void	update(Action action)

The following tables summarise some useful methods from the standard Java API.

`ArrayList` (and hence `Actions`) Method Summary

boolean	add(Object o) Appends the specified element to the end of this list.
void	add(int index, Object element) Inserts the specified element at the specified position in this list.
boolean	add(Object o) Appends the specified element to the end of this list.
Object	get(int index) Returns the element at the specified position in this list.
int	indexOf(Object elem) Searches for the first occurence of the given argument, testing for equality using the equals method. Returns -1 if not found.
boolean	isEmpty() Tests if this list has no elements.
Object	remove(int index) Removes the element at the specified position in this list.
int	size() Returns the number of elements in this list.
ListIterator	listIterator() Returns an iterator of the elements in this list (in proper sequence).

`ListIterator` Method Summary

boolean	hasNext() Returns true if this list iterator has more elements when traversing the list in the forward direction.
Object	next() Returns the next element in the list.

1.

(a) *This essay question will appear in the actual exam. You can prepare for it in advance.*

Some argue that scientific advancement comes primarily from a series of many incremental steps, while others believe that advances come primarily from fewer, but significant, landmark discoveries. During this unit you will have seen examples of each of these, both in the lecture material and the associated labs and the project.

Discuss these two views of technological development as applied to AI in the context of what you have learned while undertaking this unit. In addition to lecture material, your answer should include a discussion of your practical project work, in which you were given the opportunity to investigate successively more sophisticated techniques for a game playing agent. *A good answer will convey what you have learned from your own explorations in this unit.*

(10)

(b) *This question is not in the exam, but would have been a good question.*

In the laboratories we developed software for both WordChess and CheX. Describe the way in which the software was structured, and how this permits a distinction between *generic* and *domain specific* components. Why is this important?

(5)

2.

(a) Briefly compare *iterative deepening* search with *breadth-first* and *depth-first* search. Your answer should consider the four main criteria on which search algorithms are usually compared. You may assume unit step cost (and finite branching).

You may answer in point form. For each point, you should state how the search performs under that criterion, and briefly why. For example:

Breadth-first

- **Completeness**: yes — since the children of a visited node are added to the end of a finite queue, all nodes will eventually be visited
-

(6)

(b) A *general search* algorithm can be described as follows:

General Search

Given a *start state* $s_0 \in S$, a *goal function* $g(s) \rightarrow \{true, false\}$, and a *terminal condition* $t(s) \rightarrow \{true, false\}$:

Initialise a set $U = \{s_0\}$ of *unvisited* nodes containing just the start state and an empty set $V = \{\}$ of *visited* nodes.

(i) If U is empty halt and return `null` (no goal found).
(ii) Select, according to some (as yet undefined) *strategy*, a node s from U.
(iii) If $g(s) = true$ halt and return the goal node.
(iv) If $t(s) = true$ discard s and repeat from 1.
(v) Otherwise move s to the set V, and add to U all the nodes reachable from s. Repeat from (i).

Complete the method `search` in the following class to implement the general search. Recall that `Actions` is an `ArrayList`. Assume `Node` contains a method `update(Action a)` that applies an action to the node to obtain its successor.

```
package search;
import agent.*;
import java.util.*;

public abstract class GeneralSearch {

  Environment environment;
  UtilityFunction utilityFunction;
  ArrayList unvisited, visited;

  public GeneralSearch (Environment environment,
                        UtilityFunction utilityFunction) {
    this.environment = environment;
    this.utilityFunction = utilityFunction;
    unvisited = new ArrayList();
    unvisited.add(new Node(environment, new Actions()));
    visited = new ArrayList();
  }

  public Node search () {
    Actions arcs;
    Action arc;
    Node visit, successor;
    while ___________________ {
      visit = select();
      if (utilityFunction.goal(visit.environment)) return visit;
      else if (!_____________________________) {
        arcs = visit.environment.getActions();
        ---------------------------------
        ---------------------------------
        ---------------------------------
        ---------------------------------
        ---------------------------------
        ---------------------------------
      }
      visited.add(visit);
    }
    return null;
  }
```

```
  public abstract Node select ();
  // removes a node from the unvisited list

  public abstract void insert (Node node);
  // inserts a node in the unvisited list

  public abstract boolean terminalCondition (Node node);
  // returns true if node satisfies the terminal condition
}
```

(5)

(c) Describe how the above class can be extended or supplemented to provide a *depth-first* search, a *breadth-first* search, and an *iterative deepening* search.

(4)

3.

(a) A *knowledge-based agent* must be able to draw conclusions from the knowledge it is given. What are the two main components of a knowledge-based agent? Why is it useful to separate them? Give an example of how each component could be implemented.

(4)

(b) Assume you wish to mechanise (that is, write a program to implement) the process of finding conclusions from a knowledge base. Would you prefer a syntactic or semantic approach? Why?

(3)

(c) Describe the *resolution* inference rule.

(1)

(d) What is *conjunctive normal form (CNF)*? Why is it necessary to convert formulas into CNF in order to use resolution?

(2)

(e) Show that the resolution inference rule is *sound*.

(4)

(f) What is the difference between an *intractable* procedure and an *undecidable* procedure?

(1)

4.

(a) What do we mean by a heuristic? How is a heuristic used in the A* algorithm? What property of a heuristic ensures A* is optimal?

(3)

(b) What problem with A* is *iterative deepening A** (IDA*) designed to overcome? Briefly describe, using diagrams if necessary, how it achieves this while at the same time preserving optimality.

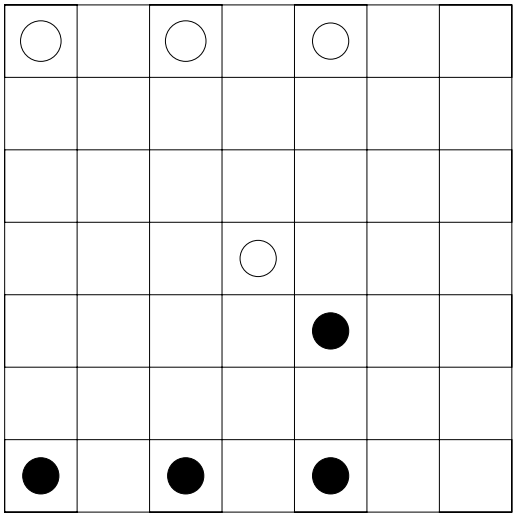
(3)

(c) Under what situation does IDA* reach a worst-case time performance and why? If N nodes are expanded by A*, give a "big O" expression for the number of nodes expanded by IDA* in the worst case, showing how this expression is derived.

Briefly explain how this performance can be improved by trading accuracy for time performance.

(5)

(d) The following picture shows a game of Finish First (a draughts-like game in which the draughts are moved one square diagonally, and the aim of each player is to get all of their pieces to the last row of the board). White is moving up the board, and it is white's turn to move.



Note that (although black *should* win) either player can win from this position. Applying a standard minimax algorithm, however, white may be equally likely to choose a move which prevents it from having any chance of winning. Why is this?

(2)

(e) Briefly describe the *horizon problem* brought about by depth-limited searches.

(2)
