Lincoln School of Computer Science	
Assessment Component Briefing Document	
Title: CMP2020M Artificial Intelligence – Component One	**Indicative Weighting: 40%**
Learning Outcomes: On successful completion of this component a student will have demonstrated competence in the following areas: - [LO2] Apply Artificial Intelligence techniques to solve practical problems - [LO3] Locate and reference relevant information.	

Requirements

This coursework assignment is a staged submission, comprising two parts (*criterions*), weighted at 70% and 30%, respectively (see CRG):

1. **Criterion 1 (70%)**: Ongoing engagement and accomplishment with a total of **4 small implementation tasks** to be completed within the workshop sessions. These will be made available following the corresponding lectures. Their successful implementation needs to be demonstrated to a delivery team member (academic or demonstrator) on duty **during the regular** workshop sessions. Successful accomplishment needs to be demonstrated in the **following week's workshop session the latest** to gain full marks. For instance, if a task is given out in week B2, it can be shown working to the delivery team either in week B2 directly or in week B3 the latest to gain full marks. Demonstrating a working version of the implementation later than a week after it has been made available in a workshop session will yield a penalty of 50% for that respective task, i.e., the marks will be halved. The table below indicates the respective last week each mini task needs to be shown working to the demonstrator.

Task	**Introduced in week commencing**	**Latest demonstration week to gain full marks, week commencing**		**Marks**
		Groups (A, B & D)	Groups (C & E)	Total is 70% from 40%
Mini Task 1: Logic Programming	27/1/2020	10/02/2020	03/02/2020	7
Mini Task 2: Artificial Neural Network	10/2/2020	24/02/2020	17/02/2020	7
Mini Task 3: Games & Search	02/03/2020	16/03/2020	09/03/2020	7

Mini Task 4: Probabilistic AI	30/03/2020	27/04/2020	20/04/2020	7

Overall, this criterion is contributing a total of 70% to the coursework mark; which itself is weighted as 40% in this module, so each mini task can contribute (70%*40% /4 =) **7 marks** to your final module mark. It is the students' responsibility to approach a member of the delivery team to have their work signed off. Work for this criterion is ***not*** **submitted** to the blackboard and it will not be accepted by email.

Criterion 2 (30%): Independent and individual implementation of solutions to **AI planning problems** using PDDL (the Planning Domain Definition Language). This criterion is contributing a total of (30% *40% =) **12 marks**. All the source code must be suitably documented, i.e. *every relevant* statement in your code needs to be explained in terms of the function it fulfils to receive full marks. Also, any resources (websites, books, papers, etc.) used to solve the tasks need to be correctly referenced in the source code, using appropriate comments (a semicolon in PDDL) and formatted per the Harvard referencing guide. The exact task to be implemented is included at the end of this brief (**Appendix A**). Students will be using the planning web service at http://lcas.lincoln.ac.uk/fast-downward/ to implement and test their solution.

The submission for this criterion is via the respective submission area on the blackboard. The implementations are to be downloaded from the planning web service at http://lcas.lincoln.ac.uk/fast- downward/, named "domain.pddl", and submitted via the blackboard. The submitted code will be tested using the very same web service for marking against the two exemplary problems given there. The domain files can be downloaded from the web service by clicking the **Download** button.

Note: The submission date indicated on the hand-in spreadsheet on blackboard only applies to the 2nd criterion (The 4 small implementation tasks) needs to be done during workshop sessions to a member of the delivery team only to obtain full marks (see table above).

Not complying with the above requirements will result in a presentation penalty.

Useful Information

This assessment is an **individually assessed** component (your source code submission will automatically and manually be assessed for plagiarism). Your work must be presented per the Lincoln School of Computer Science guidelines for the presentation of assessed written work. Please make sure you have a clear understanding of the grading principles for this component as detailed in the accompanying Criterion Reference Grid (CRG). If you are unsure about any aspect of this assessment component, please seek the advice of a member of the delivery team. It is your responsibility to have a full understanding of the task and ask for clarification as required.

Submission Instructions

Work for criterion 1 (small implementation tasks) is *not* submitted on the blackboard but demonstrated by the students during workshop sessions.

The deadline for submission of **program code for criterion 2** is given in the official documentation for hand-in dates on the blackboard. Students submit their work for that criterion 2 (The AI planning problem) as a file named “domain.pddl” via blackboard (you can compress them into a zip file and submit under **Assessment Documents – Assessment Item 1—CMP2020M Assessment Item 1 Supporting Documentation Upload**). Students need to make sure they read the instructions for submission carefully, as failing to do so may incur a presentation penalty.
DO NOT include this briefing document with your submission.

Appendix A: Assessment Item 1 – Criterion 2 – AI Planning [30%]

Your task is to write a PDDL domain that can solve a planning problem for floor-cleaning robots. A set of robots has the task to clean floor tiles. The robots can move around the floor tiles in four directions (up, down, left and right). Robots have a brush mounted at the front and at the back, so they can clean in front and behind them (up and down, respectively), but they cannot turn around. The robots, however, cannot drive on wet surfaces, so they must never drive onto tiles they have already cleaned (and are therefore wet). The task is to write a planning domain, which can solve this task for general environments given some further constraints detailed in the sub-tasks below. You should use the planning web service provided to you at http://lcas.lincoln.ac.uk/fast-downward/.

In the "Problem" section of that web service you find exemplary problem definitions, and in the "Domain" section you will find a skeleton PDDL domain for this problem (named **floor-domain-template**) and other examples.

Complete the domain skeleton so it solves the given problems **(floor-problem-01** and **floor-problem- 02**). These problem definitions already define the configurations representing an environment. As hinted in the template, you need to define the preconditions and effects for six actions, namely up, down, right, left, clean-up, clean-down. Make sure, you name the actions precisely as asked for, or you will not be able to receive full marks.

Make sure you comment your solution appropriately using comments starting with ";". Your comments must explain what your actions model and how this helps to solve the problem. You are advised to extend on the given domain template, but you may also develop your own from scratch. But your domain must be able to solve the given problem definition as they are. You are not allowed to provide your own problem definition; your code must be able to solve the given problems.

In any case, make good use of the demonstrators and the delivery team. Approach them, ask questions, and present partial solutions and your thinking.
Note 1: If you code is not commented well, you won’t be getting a very good mark! The comments need to explain your thinking and explain how and why you have implemented something.
Note 2: Double-check the file you submit. If your submitted domain file does not yield a solution when copied directly into the web service, you will not be able to achieve a good mark. Your code is automatically checked for correctness (does the submitted file solve the problems). If it fails the test, your implementation is considered not to be working.