# Cardiff School of Computer Science and Informatics

## Coursework Assessment Pro-forma

**Module Code**: CM3112
**Module Title**: Artificial Intelligence
**Lecturer**: Steven Schockaert
**Assessment Title**: Coursework
**Assessment Number**: 1
**Date Set**: 21 October 2019
**Submission Date and Time**: 11 November 2019 at 9:30am
**Return Date**: 2 December 2019

This assignment is worth 30% of the total marks available for this module. If coursework is submitted late (and where there are no extenuating circumstances):

1 If the assessment is submitted no later than 24 hours after the deadline, the mark for the assessment will be capped at the minimum pass mark;
2 If the assessment is submitted more than 24 hours after the deadline, a mark of 0 will be given for the assessment.

Your submission must include the official Coursework Submission Cover sheet, which can be found here:

https://docs.cs.cf.ac.uk/downloads/coursework/Coversheet.pdf

---

## Submission Instructions

You should submit a single zip file, containing the source code of your implementation and a PDF version of your report.

| Description | | Type | Name |
|---|---|---|---|
| Cover sheet | **Compulsory** | One PDF (.pdf) file | [student number].pdf |
| Solution | **Compulsory** | One zip file | [student number].zip |

Any code submitted will be compiled using Java JDK 8 and must be submitted as stipulated in the instructions above.

Any deviation from the submission instructions above (including the number and types of files submitted) will result in a reduction in marks for the assessment of 5%.

Staff reserve the right to invite students to a meeting to discuss coursework submissions

---

## Assignment

Your task is to implement an A* solver for the puzzle game Rush Hour, by extending the partial java implementation that is provided. In addition to the implementation, you should provide a short report (maximum 1 page) with a discussion about the choice of the heuristic evaluation function and a brief evaluation of your code.

Detailed instructions can be found in the appended description.

---

## Learning Outcomes Assessed

1. Choose and implement an appropriate search method for solving a given problem.
2. Define suitable state spaces and heuristics.
3. Understand the basic techniques for solving problems.

---

## Criteria for assessment

Credit will be awarded against the following criteria.

For the implementation:

- [Correctness] Does the code correctly implement the required algorithm?
- [Efficiency] Is the computational complexity of the implementation optimal?
- [Clarity] Is the code clearly structured and easy to understand?

For the report:

- [Correctness] Does the answer demonstrate a solid understanding of the problem?
- [Clarity] Is the answer well-structured and clearly presented?

Marks will be assigned based on the following benchmarks:

| | |
|---|---|
| 1st (70-100%) | The code generates a correct solution on all test cases, is efficient and clearly structured. The report demonstrates a clear understanding of the considered problem. |
| 2.1 (60-69%) | The code generates a correct solution on all test cases, but may be too inefficient to solve some of the more difficult levels within a reasonable amount of time. The report demonstrates a reasonable understanding of the considered problem. |
| 2.2 (50-59%) | The code is largely correct, but does not always give the correct answer. The report has minor errors. |
| 3rd (40-49) | A reasonable attempt towards a fully working implementation has been made, but the code is not finished. |

---

## Feedback and suggestion for future learning

Feedback on your coursework will address the above criteria. Feedback and marks will be returned at the latest on 2 December 2019 via learning central. General feedback on the coursework will be provided in the lecture.

## Getting Started

To get started with this coursework, first download `rushhour.zip` from Learning Central. In this archive, you will find a partial implementation of an A* solver for the game Rush Hour. Rush Hour is a puzzle game in which the player needs to move a number of cars to allow a designated target car to reach the exit of a car park. An example of a Rush Hour puzzle is shown in Figure 1. The target car is shown in red. The aim of the puzzle is always to move the target car to the right-most column of the grid. The target car can only be moved horizontally. Each of the other cars can either only be moved horizontally or only vertically. In the example from Figure 1 a possible solution would be to (i) move car 3 one step to the left, (ii) move car 2 three positions upwards, (iii) move car 1 four positions to the right.

The Java code in the `rushhour` archive contains several classes. First, the `search` package contains a generic implementation of the A* algorithm. The package `rushhour` contains the following classes:

**RushHour** This class contains the `main` method to run your code. In case a solution is found, it prints the following information to the screen: the amount of time it took to find the solution, the number of nodes that were expanded and the cost of the solution. The puzzle that needs to be solved is passed as a command line argument (see below). You don't need to make any changes to this class.

**GameState** This class contains methods for reading the puzzle from file, as well as several helper methods that may prove useful when solving the coursework. This class implements the `State` interface from the `search` package, but the implementation of four methods from this interface is still missing. Your main assignment is to implement these missing methods.

**Car** This class represents the properties of a car (i.e. its length and orientation), as well as its position on the grid. It is used by the `GameState` class.

**Position** This class represents a position on the grid.

The archive also contains a few sample levels. These levels are encoded as plain text files, so you can easily create your own (e.g. to test particular aspects of your computer player). An example

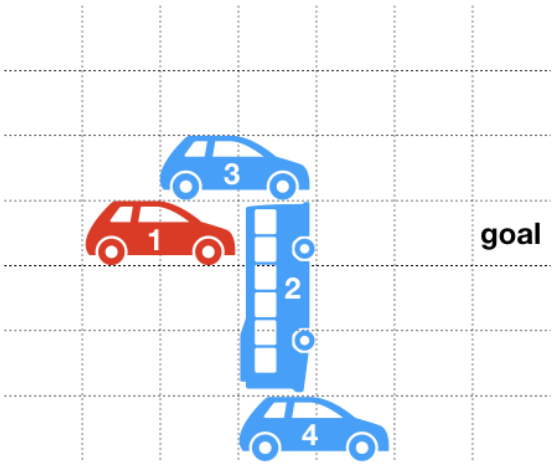


Figure 1: Illustration of the Rush Hour game.

of a level is:

```
6
7
2 3 1 H
0 4 2 H
3 5 3 V
```

The first line contains the number of rows of the grid, while the second line contains the number of columns. The next line always contains the initial position of the target car, with the next lines containing the initial positions of the blocking cars. Specifically, each of these lines mentions the row, column, length and orientation of the car. The orientation is specified as `H` (for horizontal) or `V` (for vertical). Note that the orientation of the target car should always be horizontal.
To compile the code, from outside the directory that contains the source files, you can use[1]:

javac rushhour/*.java

To run the code on a Java Virtual Machine with 1GB of memory, you can use:

java -Xmx1g rushhour/RushHour random1.txt

where `random1.txt` is a text file containing the specification of a level.

## Assignment

**Task 1: Implementing a computer player for Rush Hour (70 marks)**.
Your task is to finish the computer player by implementing the following methods from the `State` interface:

- `public List<Action> getLegalActions();`
- `public boolean isLegal(Action action);`
- `public State doAction(Action action);`
- `public int getEstimatedDistanceToGoal();`

An important part of this assignment is to choose and implement a suitable heuristic. In your implementation you should assume that sliding a car up, down, left or right always has a cost of 1, regardless of the number of steps by which the car is moved. You should take this into account when designing your heuristic function. To facilitate marking, your code should not output anything to the screen, apart from the standard output which is generated by the `RushHour` class.

**Task 2: Report (30 marks)**
You are asked to write a concise report discussing the following points. The total length of your report should not be longer than one page (5 marks will be deducted for each additional page).

1. Briefly the discuss the effectiveness of your chosen heuristic, compared to the trivial heuristic that always returns 0. Note that because all actions have uniform cost, using this trivial heuristic is equivalent to using breadth-first search. [10]

[1] Replace / by \ on Windows.

2. Explain why your chosen heuristic is admissible. [10]

3. Outline a strategy that could be used to further improve your chosen heuristic. [10]