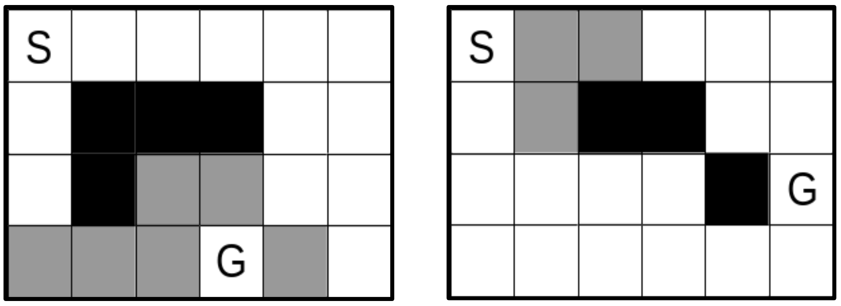
F29AI – Artificial Intelligence and Intelligent Agents

Coursework 1

A* Search, Knowledge Representation, and Automated Planning

You should complete this coursework **individually**. Coursework 1 has **three parts** (A* search, knowledge representation in Prolog, and automated planning using PDDL) and is worth **17.5%** of your overall F29AI mark. Details of what you should do and hand in, and how you will be assessed, are described below.

Part 1: A* Search problem description



The above grids represent two problem environments where an agent is trying to find a path from the start location (S) to the goal location (S). The agent can move to an adjacent square provided the square is white or grey. The agent cannot move to a black square which represents a wall. No diagonal movement is allowed. Moving into a white square has a cost of 1. Moving into a grey square has a cost of 3.

What to do: Undergraduate students, 1-year Edinburgh postgraduate students, and all Dubai postgraduate students (full time and part time)

Implement a solution to the grid problem in Java using A* search. Starter code has been provided for you and is available in the accompanying zip file. In particular, the code in the package **uk.ac.hw.macs.search** is a set of classes that can be used to represent a search problem. To implement a specific search problem, you will need to do the following:

1. Define a class that implements the `State` interface. This should include the following:
   a. A method for determining whether a state is a goal state (`isGoal()`)
   b. A method for computing the heuristic value of a state (`getHeuristic()`)
2. Define a class that implements the `SearchOrder` interface. This interface has one public method, `addToFrontier`, that is used to add a set of nodes to the frontier. You can use the costs and/or heuristic values to determine the order that nodes are added to the frontier

The classes in the **uk.ac.hw.macs.search.example** package show examples of these two interfaces being used to implement depth-first search and breadth-first search, as well as a simple integer-based state. The `Main` class in this package shows an example of how to use the classes.

To solve the problem, you will need to implement the following:

1. An encoding of the state in the grid by implementing the `State` interface appropriately, including methods for detecting a goal state and computing a heuristic value. You should use the Manhattan distance heuristic for generating heuristic values in your search.
2. A class implementing A* search by implementing the `SearchOrder` interface and implementing `addToFrontier` appropriately.

What to hand in

Test your code on the two grid problems provided by running A* search on them and capture the output. Submit the code for the implementation as well as the output. Make sure your code includes appropriate comments in the parts of the code you implemented. Do not change any of the classes in the **uk.ac.hw.macs.search**: we will test your implementation against the provided classes.

What to do: 2-year Edinburgh postgraduate students only

Describe a solution to the grid problem using A*search. In particular, you should do the following:

1. Draw a graph (with nodes and edges) to represent each of the grid problems as a search problem. Label each of the nodes in your graph and include appropriate costs on the edges.
2. Use the Manhattan distance heuristic and calculate appropriate heuristic values for each of the nodes in your graph.
3. Apply A* search to each grid and list the final set of states expanded and the goal path for each grid. You do not need to provide a full derivation for each search (unless you wish to do so), however, you should provide at least the first 5 steps of each derivation with calculations for the f values to demonstrate you understand the application of the A* algorithm in your graph.

What to hand in

Submit your answers in a report in PDF format. You do not need to write or submit any code.

Part 2: Knowledge Representation problem description

A popular fictional monster battling game features a type system that is used to determine the effectiveness of a monster's ability to attack or defend against another monster. In this coursework you will write a **Prolog program** to represent knowledge about this system and to answer queries using the resulting knowledge base.

The version of the game we will use includes five **monsters**: `bewear`, `fraxure`, `polteageist`, `rookidee`, and `sirfetchd`. Each monster can be one of five basic **types**: `dragon`, `fighting`, `flying`, `ghost`, and `normal`. Each monster has four moves that it can use. Each **move** is also assigned one of the basic types. The details of each monster, its type, its moves, and the move types are given in the following table:

Monster	**Monster type**	**Move**	**Move type**

`bewear`	`normal`	`strength`
		`takeDown`
		`superpower`
		`shadowClaw`
			`normal`
			`normal`
			`fighting`
			`ghost`

`fraxure`	`dragon`	`falseSwipe`
		`dragonClaw`
		`outrage`
		`shadowClaw`
			`normal`
			`dragon`
			`dragon`
			`ghost`

`polteageist`	`ghost`	`facade`
		`shadowBall`
		`astonish`
		`teaTime`
			`normal`
			`ghost`
			`ghost`
			`normal`

`rookidee`	`flying`	`revenge`
		`peck`
		`braveBird`
		`furyAttack`
			`fighting`
			`flying`
			`flying`
			`normal`

`sirfetchd`	`fighting`	`brickBreak`
		`braveBird`
		`bodySlam`
		`closeCombat`
			`fighting`
			`flying`
			`normal`
			`fighting`

E.g., `sirfetchd` is a `fighting` type monster with 4 moves: `brickBreak` (a `fighting` type move), `braveBird` (a `flying` type move), `bodySlam` (a `normal` type move), and `closeCombat` (a `fighting` type move).

The **effectiveness** of a monster's move when used on another monster depends on the move type (of the monster using the move) and the monster type (of the monster the move is being used on). Certain moves are `strong` against certain types of monsters while other moves are `weak` or `superweak` against other monster types. The effectiveness of a move type against a monster type is represented in the following table:

`move \ monster`	`dragon`	`fighting`	`flying`	`ghost`	`normal`
`dragon`	`strong`	`ordinary`	`ordinary`	`ordinary`	`ordinary`
`fighting`	`ordinary`	`ordinary`	`weak`	`superweak`	`strong`
`flying`	`ordinary`	`strong`	`ordinary`	`ordinary`	`ordinary`
`ghost`	`ordinary`	`ordinary`	`ordinary`	`strong`	`superweak`
`normal`	`ordinary`	`ordinary`	`ordinary`	`superweak`	`ordinary`

E.g., a `fighting` type move is `weak` against `flying` type monsters but a `flying` type move is `strong` against `fighting` type monsters. Combinations that aren't `strong`, `weak`, or `superweak` are `ordinary`.

What to do

Write a Prolog program to represent the knowledge in the monster game according to the following specification:

1. Encode information about the monsters and their abilities using Prolog **facts** of the following form:
   - `type(t)`: to represent the idea that `t` is a basic type.
   - `monster(mo,t)`: to represent the idea that `mo` is a monster of type `t.`
   - `move(mv,t)`: to represent the idea that `mv` is a move of type `t.`
   - `monsterMove(mo,mv)`: to represent the idea that monster `mo` has a move `mv.`
2. Encode effectiveness information using Prolog **facts** of the form `typeEffectiveness(t1,t2,e)`: a move of type `t1` used again monsters of type `t2` has effectiveness `e`.
3. Encode **basic** effectiveness information using Prolog **facts** of the form `moreEffective(e1,e2)`: `e1` is more effective than `e2`. You should only encode the strict ordering on effectiveness in this way, i.e., `strong` is more effective than `ordinary`, `ordinary` is more effective than `weak`, and `weak` is more effective than `superweak`.
4. Encode **transitive** effectiveness information using a Prolog **rule** of the form `moreEffective(E1,E2)`: `E1` is more effective than `E2`. The rule should cover information not represented by the facts in part 3, e.g., `strong` is more effective than `weak`, `ordinary` is more effective than `superweak`, etc. `E1` and `E2` should be variables in your rule definition.
5. Define a Prolog **rule** called `monsterMoveMatch(MO,MV)` which represents the idea that monster `MO` has a move `MV` and that `MO` and `MV` have the same type. `MO` and `MV` should be variables in your rule definition.
6. Define a Prolog **rule** called `moreEffectiveMove(MV1,MV2,T)` to represent the idea that move `MV1` is more effective than move `MV2` against monsters of type `T`. `MV1`, `MV2`, and `T` should be variables in your rule definition.
7. Define a Prolog **rule** called `moreEffectiveMonster`(MO1,MV1,MO2,MV2) to represent the idea that if monster `MO1` performs move `MV1` and monster `MO2` performs move `MV2` that `MV1` is a more effective against `MO2` than `MV2` is against `MV1`. `MO1`, `MV1`, `MO2`, and `MV2` should be variables in your rule definition.

NOTE: For parts 4-7 of the specification, ensure that you write Prolog rules. You should **not** implement Prolog facts as your solution for these parts and you will lose marks if you do this. However, it is okay if need to write multiple rules for each definition. If you are interested, the game mechanics in this coursework are based on information from https://pokemondb.net/.

What to hand in

- **Prolog program file:** Submit a single Prolog program file with all your Prolog facts and rules. Ensure that the file is a plain text file with the name `monster.pl`. Make sure there are comments in your program file to describe the different parts of your program

- **Output file:** Test your program by selecting a series of Prolog **queries** to demonstrate the various facts and rules you have implemented. Capture the queries and output to a text file. Include at least 10 queries in your output file. Ensure that the file is a plain text file with the name `output.txt`.

Part 3: Automated Planning problem description

Heriot-Watt Universal has decided to extend its planetary exploration activities by terraforming an uninhabited planet. The planet has been divided into a series of grid-based regions. A base of operations is located in one of the regions. Terraforming operations will be controlled by mission plans generated using an automated planner that will direct the terraforming personnel, equipment, and activities on the planet. Several types of personnel serve at the base, including the commander, engineers, science officers, and security personnel. The base also contains certain equipment that is useful to terraforming operations, including shield emitters and terraforming machines. All personnel and equipment initially begin at the base but may be directed to move to other regions of the planet as necessary. Some of the operations of the terraforming mission are described in the following list (which isn't exhaustive):

1. Each region of the planet can be either flat, hilly, or mountainous. The base of operations is located in a flat region of the planet.
2. All personnel can individually move between different regions of the planet, provided the regions are adjacent to each other. As a result, it may take multiple moves to reach distant regions. Personnel can only move through flat or hilly regions, not mountainous regions.
3. Engineers can carry and deploy shield emitters. An engineer can carry at most two shield emitters at a time. Shield emitters can only be deployed in flat regions.
4. A shield emitter can be turned on and off. Turning on a shield emitter causes a shield to become active which protects the region in which it is deployed. A region can only contain at most one shield emitter.
5. Science officers can carry and activate a terraforming machine. A science officer can carry at most one terraforming machine. A terraforming machine can be activated in either a flat or hilly region, but only if the region is protected by a shield and the terraforming machine is charged. Once the terraforming machine has been used it becomes discharged and must be recharged at the base before it can be used again. Successfully activating a terraforming machine causes the region in which it was activated to become terraformed.
6. Activating a terraforming machine in a region without a shield causes the machine to become damaged and unusable, due to the dangerous radiation on the planet.
7. A science officer can recharge a terraforming machine at the base.
8. An engineer can fix a damaged terraforming machine at the base.
9. If three individual shield emitters are set up on the planet, a planetary shield can be activated which networks the individual shields to protect the entire planet.
10. The commander can turn the planetary shield on and off at the base.
11. Space pirate camps have been reported in several regions of the planet. If base engineers or science officers enter a region of the planet with space pirates then they will be captured unless there are security personnel already in the region. Captured personnel cannot move or perform any activities.
12. Captured personnel can be released if the commander visits the pirate camp to negotiate their release. If the commander enters a region of the planet with space pirates they will also be captured unless there are security personnel already in the region.

At the start of operations, the base is given a series of missions to complete that involve terraforming several regions of the planet.

What to do: all students

1. **PDDL implementation:** You must model the planetary terraforming domain in PDDL by defining the properties, objects, and actions that are needed to describe the domain. Note that the planning domain is described at an abstract level and is somewhat incomplete, with certain pieces of information missing. You must make design decisions as to how you will represent the knowledge and actions necessary to encode this scenario as a planning problem. Some requirements are more difficult than others. It is strongly recommended that you try to implement the domain incrementally, ensuring that some parts of the domain work correctly before moving on to others. Use the example domains from the PDDL lectures as a starting point for your solutions. You may also find that the planning time increases as you add more complexity. You may have to consider whether an alternative knowledge representation leads to a better

solution. You may use the planning tools available at http://editor.planning.domains/, the Fast Forward (FF) planner, or the Fast Downward planner. You should ensure that your solution works with one of these planners. Note that the performance of certain planners may outperform that of the web-based planner. Do not use any features of PDDL that we have not covered in the course. Make sure you test your solution on a series of different problem scenarios. Include comments in your PDDL files to describe important sections of your code.

2. **PDDL report:** Write a short report (maximum 2 pages) showing the regions of your planet, how they are connected, the type of each region, and the location of the base. Also list in your report the particular problem instances you have included in your code and what planner you have used to test your domain/problems. The report will be used to help understand your code.

What to do: 1-year Edinburgh postgraduate students and all Dubai postgraduate students (full time and part time)

In addition to the above instructions, 1-year Edinburgh postgraduate students and all Dubai postgraduate students (full time and part time) should design an additional feature in PDDL to add to the domain (e.g., new personnel that can perform some task, a new activity, etc.) that isn't included in the above domain description. Add this feature to your domain and test it. Include a description of the additional feature in your PDDL report (maximum 1 extra page).

What to hand in: all students

- **PDDL source files:** Submit your PDDL source files (domain + problems) in a zip/tar file. Make sure your source files have comments describing the properties and actions you've defined. Your solution should consist of a single PDDL domain file and at least 4 different PDDL problem files illustrating different scenarios that are supported by your domain. You should aim for comprehensive scenarios that support multiple missions in each problem file. Your source files will be checked for plagiarism and tested to see if they are operational. For 1-year Edinburgh postgraduate students and all Dubai postgraduate students (full time and part time): one of the problem files must test your additional feature.

- **PDDL report:** Submit your report as a PDF file. Your report will be checked for plagiarism.

Deadlines

The deadline for submitting Coursework 1 (all parts) is **Thursday, 28 October 2021**. Submissions are due by **3:30pm (Edinburgh local time)** for the Edinburgh Campus and **11:59pm (Dubai local time)** for the Dubai Campus. Submit your coursework on the Canvas site for F29AI.

Feedback

Individual written feedback will be provided to students approximately three working weeks after the submission of Coursework 1.

Additional notes

This is an **individual coursework** assignment. All submitted files will be checked for plagiarism. You must confirm to the naming conventions described in this document. If files are unreadable or code does not run, you will receive **0 marks** on that part of the assessment.

Assessment

This coursework will count towards **17.5%** of your overall mark for F29AI and will be marked out of **40 marks**.

A* Search: Java code (10 marks)

Your Java code will primarily be marked on its correctness with respect to the two grid problems. We will also check the implementation of the heuristic in your solution.

0 **None**	**1-2** **Poor**	**3-4** **Fair**	**5-6** **Good**	**7-8** **Very good**	**9-10** **Excellent**
No code submitted.	Major problems with code. Solution is incorrect and/or code does not run.	Code partially works but there are major problems with code structure, solution, and/or heuristic.	Good code and solution. Code runs almost perfectly but there are problems with code structure, solution or heuristic.	Very good code and solution. Code runs perfectly. Small problems with code structure, solution, or heuristic.	Exceptional code and solution. Code runs perfectly. Heuristic is well defined.

A* Search: report (10 marks)

Your report will primarily be marked on its correctness with respect to the two grid problems. We will also check your graph and heuristic values you are using in your solution.

0 **None**	**1-2** **Poor**	**3-4** **Fair**	**5-6** **Good**	**7-8** **Very good**	**9-10** **Excellent**
No report submitted.	Major problems with report. Solution is incorrect and/or many parts of the solution are wrong or missing.	Major problems in the report. Some parts of the solution partially incorrect or missing. Description of the solution could be improved in many places.	Good report. Solution is mainly correct but there are some minor problems. Description of the solution could be improved.	Very good report. All parts of the solution are described and the solution is correct but there are some small problems with the description that could be improved.	Exceptional report. All parts of the solution are well described and solution is perfect.

Knowledge Representation (10 marks)

You will be assessed on the correctness of your Prolog program and the output it produces.

0 **None**	**1-2** **Poor**	**3-4** **Fair**	**5-6** **Good**	**7-8** **Very good**	**9-10** **Excellent**
No source code or minimal source code submitted.	Weak solution. Many important requirements of the specification or important parts of the output missing. Comments missing.	Adequate solution. Some requirements of the specification implemented but important parts missing. Output cases do not provide complete coverage and/or missing many comments.	Thorough solution. Majority of program meets specification. Some small problems with the program, missing cases in the output file, and/or missing comments.	Very thorough solution. Program almost works perfectly. There might be very small problems with the program, some missing cases in the output file, and/or missing comments.	Program works perfectly. Output illustrates all important cases. Comments in code are descriptive.

Automated Planning (20 marks)

The automated planning mark will primarily be based on the PDDL code you have supplied and how correctly it implements the coursework specification. You may submit a single PDDL domain file and multiple PDDL problem files in a single zip/tar file. The supplied files should provide coverage of the various scenario requirements and demonstrate correct behaviour. The marker will test a selection of the specified PDDL files for plan correctness. You should clearly indicate in your report what planner you have used to test your solution. PDDL files will only be tested on editor.planning.domains, FF, or Fast Downward. Correctness will be assessed not only against the coursework requirements but also with respect to the specific implemented solution. (I.e., non-obvious or incorrect/missing action preconditions or effects may lead to strange plan output and mark deductions.) Usual program quality criteria (e.g., use of whitespace, comments, naming conventions, etc.) will apply here to assess the readability of the code. The marker will be looking to see if you understand how to write PDDL domains and problems and have made good use of the language features that are available. Deductions for poor code quality (up to 5 marks, depending on severity) may be made even if your domain works perfectly.

1-year Edinburgh postgraduate students and all Dubai postgraduate students (full time and part time): Up to 2 marks will be allocated for the correctness of the additional feature. The additional feature should be similar in terms of complexity to the requirements in the main specification (and may be more complex if you'd like), and you are encouraged to be creative in your solutions. Trivial additional features will receive very few marks.

0 **None**	**1-4** **Poor**	**5-8** **Fair**	**9-12** **Good**	**13-16** **Very good**	**17-20** **Excellent**
No source code submitted.	Weak solution. Many important requirements or test cases missing and/or not working perfectly. Basic domain features. PDDL code is not understandable	Adequate solution. Some requirements implemented but important requirements missing. Test cases do not provide complete coverage. Some test cases may not work perfectly. Mostly basic domain features. PDDL code is very difficult to understand.	Thorough solution. Majority of requirements are met. Choice of test cases is convincing. Very few test cases do not work perfectly. A good quality solution overall with a mix of basic and advanced domain features. PDDL code is mainly well written, clear, and understandable.	Very thorough solution. All requirements met and choice of test cases provides maximum coverage of requirements. Solution plans are quite convincing. Everything works perfectly. A good mix of domain features, with advanced features enhancing the overall approach. PDDL code is very well written, completely clear and understandable.	Exceptional solution. All requirements are easily met and well demonstrated. Test cases are comprehensive and demonstrate robust behaviour. Everything works perfectly. Excellent design designs and use of advanced features which enhance the overall approach. PDDL code is excellent, with all aspects of the code easily understandable.

Learning Objectives

This coursework is meant to contribute to the following high-level aims for F29AI:

- To introduce the fundamental concepts and techniques of AI, including planning, search, and knowledge representation.
- To introduce the scope, subfields and applications of AI, including autonomous agents.
- To develop skills in AI programming in an appropriate language.

It is also meant to contribute to the following specific learning objectives for the course:

- Critical understanding of traditional AI problem solving and knowledge representation methods.
- Use of knowledge representation techniques (such as predicate logic).
- Critical understanding of different systematic and heuristic search techniques.
- Practice in expressing problems in terms of state-space search.
- Broad knowledge and understanding of the subfields and applications of AI.
- Detailed knowledge of one subfield of AI (e.g., planning) and ability to apply its formalisms and representations to small problems.
- Detailed understanding of different approaches to autonomous agent and robot architectures, and the ability to critically evaluate their advantages and disadvantages in different contexts.
- Practice in the implementation of simple AI systems using a suitable language.
- Identification, representation, and solution of problems.
- Research skills and report writing.
- Practice in the use of information and communication technology (ICT), numeracy, and presentation skills.

Late submission of coursework

Coursework deadlines are fixed and individual coursework extensions will not be granted. Penalties for the late submission of coursework follow the university's policy on late submissions:

- The mark for coursework submitted late, but within 5 working days of the coursework deadline, will be reduced by 30%.
- Coursework submitted more than 5 working days after the deadline will not be marked.
- In a case where a student submits coursework up to 5 working days late, and the student has valid mitigating circumstances, the Mitigating Circumstances policy will apply. Students should submit a Mitigating Circumstances application for consideration by the Mitigating Circumstances Committee.

The MACS School policy on coursework submission is that the **deadline for coursework submissions**, whether hard-copy or online, is **3:30pm (Edinburgh local time)** for the Edinburgh Campus and **5:00pm (Dubai local time)** for the Dubai Campus. The University Policy on the Submission of Coursework can be found here:
https://www.hw.ac.uk/services/docs/learning-teaching/policies/submissionofcoursework-policy.pdf

Mitigating Circumstances (MC)

There are circumstances which, through no fault of your own, may have affected your performance in an assessment (exams or other assessment), meaning that the assessment has not accurately measured your ability. These circumstances are described as **mitigating circumstances**. You can submit an application to have mitigating circumstances taken into account. Full details on the university's policies on mitigating circumstances and how to submit an application can be found here:
https://www.hw.ac.uk/students/studies/examinations/mitigating-circumstances.htm

Plagiarism

"**Plagiarism** is the act of taking the ideas, writings or inventions of another person and using these as if they were your own, whether intentionally or not. Plagiarism occurs where there is no acknowledgement that the writings, or ideas, belong to or have come from another source." (Heriot-Watt University Plagiarism Policy). This coursework must be completed independently:

- Coursework reports must be written in a student's own words and any submitted code (e.g., PDDL) in the coursework must be your own code. Short sections of text or code taken from approved sources like the lecture examples may be included in the coursework provided these sources are **properly referenced**.
- Failure to reference work that has been obtained from other sources or to copy the words and/or code of another student is plagiarism and, if detected, this will be reported to the School's Discipline Committee. If a student is found guilty of plagiarism, the penalty could involve voiding the course.
- Students must **never** give hard or soft copies of their coursework reports or code to another student. Students must **always refuse** any request from another student for a copy of their report and/or code.
- Sharing a coursework report and/or code with another student is **collusion**, and if detected, this will be reported to the School's Discipline Committee. If found guilty of collusion, the penalty could involve voiding the course.

Plagiarism will be treated extremely seriously as an act of academic misconduct which will result in appropriate student discipline. All students should familiarise themselves with the university policies around plagiarism which can be found here: https://www.hw.ac.uk/students/studies/examinations/plagiarism.htm