CM3038 Artificial Intelligence for Problem Solving

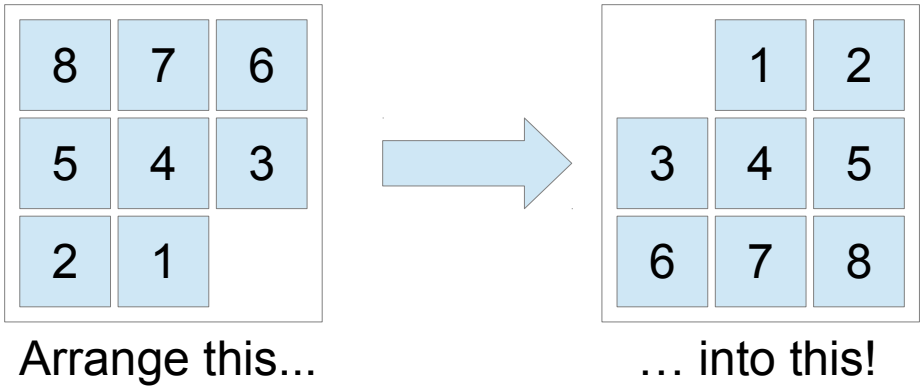
SolutionDiscussion to Laboratory #3: Solving the 8-Puzzle Problem

1. Aims

- To model and solve the 8-Puzzle problem using the `aips.search` library.

2. The 8-Puzzle Problem

In the 8-Puzzle, the game board has 8 tiles which can slide into a blank space. The task is to move the tiles into a given pattern. A typical problem is shown below:



2.1. Resources

For Java, download the `cm3038-java-lab03.zip` file from Campus Moodle. This ZIP file contains the followings:

`aips.search`:	This package contains the search library/framework.
`aips.lab03.sq8`:	This package contains classes for the 8-Puzzle problem. It uses `aips.search`.
`aips.lab03.sq8.test`:	This package contains testing classes for the 8-Puzzle problem.

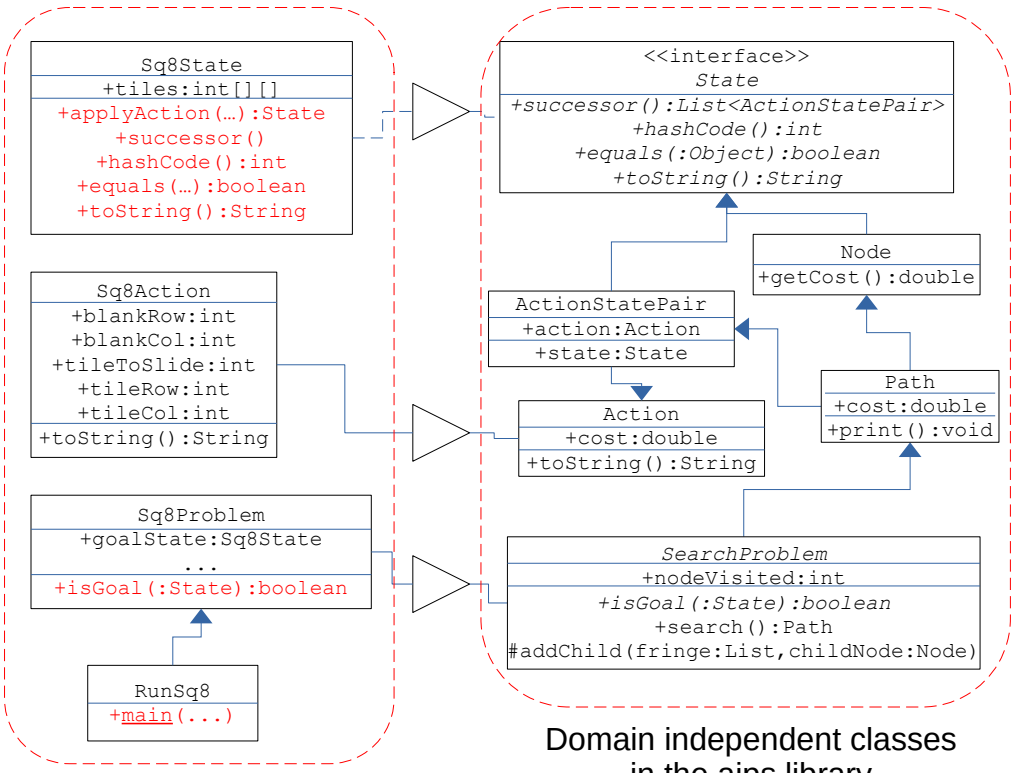
For Python, download the `cm3038-python-lab03.zip` file from Campus Moodle. This ZIP file contains the followings:

`aips/search.py`:	This module contains the search library/framework.
`aips/lab03/sq8.py`:	This file contains classes defined for the 8-Puzzle problem. This uses `aips/search`.
`aips/lab03/test`:	This folder contains testing files for the 8-Puzzle problem.

Depending on you are using Java/Python, copy the source code into the root of your project.

2.2. The Domain Specific Classes

The following UML class diagram shows the domain-specific classes that model the 8 Puzzle, and the domain-independent classes of the `aips` search library. You need to complete those methods in red:



You can see the main task is to model the domain using a data structure of your choice. In this case we use a 2D `int` array. If you use another data structure, then you need to change the methods in red so that they will work with the chosen representation.

This is not an AI task but a problem modelling task.

2.3. The `Sq8State` Class

The `Sq8State` class represents the state of the 8-Puzzle problem. Here are the things you need to know about this class:

- `Sq8State` implements the `aips.search.State` interface (or in Python, extends the `State` superclass).
  - According to the UML diagram, what methods are needed in the `Sq8State` class?
- `Sq8State` uses its `tiles` attribute, a 2D `int` array, to model the 9 spaces on the game board.

- To access a tile in the array, you must specify the row and column numbers. The first index is the row. The second index is the column. From the top-left corner, the tiles are stored in `tiles[0][0]`, `tiles[0][1]`, `tiles[0][2]`,..., `tiles[2][2]`, etc. A value of `0` means a blank.

- I chose this representation/data structure as it is natural for the game board which is 2D. Alternatively you can use a 1D array. You just need to map the row and column into the index.

2.3.1. The Java Constructors (or Python `__init__(...)` Function)

The `Sq8State` class has 2 constructors:

- The default constructor creates a game board with all blanks.
- *We do not use this constructor but it gives you some hints for the next one.*
- The 2nd constructor creates a `Sq8State` object from a given 2D `int` array.
  - Complete this method.
    - Note: **You need to make a copy of the given array** (i.e. clone it!). DO NOT simply make `tiles` point to the given array. See Lecture 03, slide 37 for a potential problem if you do so.
  - You cannot test this constructor until the `toString()` method (or Python `__str__(...)`) is completed, which is the next task.
  - This is a handy method/constructor to create any state that you want by giving a 2D array of `int`.
  - Some of you may simply assign the `int` array to the `tiles` attribute. This can work but ideally we want to make a copy of the `int` array instead of pointing to it directly, so that whatever you do to the `int` array later, it won't affect our created `Sq8State` object.

2.3.2. The Java `toString()` Method (or Python `__str__(...)` Function)

This method returns the current `Sq8State` object as a `String` (or `str` in Python). This is used when we print a `Sq8State` object. A possible returned result is:

```
8 7 6
5 4 3
2 1 -
```

- Complete this method.
- Run the `TestToString` class (or Python `testString.py`) to test your constructor and `toString()` method (or `__str__()` function). You should see an output similar to the above.

- Note that the `toString()` or `__str__(...)` method does not print but constructs a `String` (or a `str` in Python). You are freely print the result if you want.
- You are simply iterating through the `int` array but do the following extras:
  - If the content is "`0`", append a "`-`" to the result. Otherwise just append the tile number.
  - After every row, append a "`\n`" to give a line break.

2.3.3. The `equals(...)` Method (or Python `__eq__(...)` Function)

This method compares if the current object (i.e. "`this`" in Java, or "`self`" in Python) object equals to another object given in the parameter. It tests for equality of attribute values between 2 objects rather than identity.

- To know the difference between the `equals(...)` method and "==" operator in Java, see: https://www.java67.com/2012/11/difference-between-operator-and-equals-method-in.html
- In Python, the `__eq__(...)` function is invoked by the "==" operator.
  - See this for an explanation of `__eq__`: https://www.pythontutorial.net/python-oop/python-__eq__/
  - See this page for the difference between "==" and "`is`" in Python: http://net-informations.com/python/iq/is.htm
- The confusion between `equals(...)` and "==" in Java (in Python, "==" and "`is`") is a common mistake in students who don't fully master the object-oriented programming concept of equality.
  - `equals(...)` in Java is similar to "==" in Python. They check: "Do these 2 objects have the same attribute values?". i.e. Are they twin?
  - "==" in Java is similar to "`is`" in Python. they check: "Do these 2 objects have the same object identifier?". i.e. Are they the same person?
- Question:
  - Where in the search process do you think the Java `equals(...)` method (or Python `__eq__(...)` function) is used?
    - The equality test is used in 2 places:
      - To test if a state is a goal.
      - To test if a state is in the history.
  - Given the role of the `equals(...)` (or `__eq__(...)`) method, what are the problems (in the search) if:
    - `equals(...)` returns `false` when the 2 states are equal.
      - 1st problem: You won't know you hit a goal state. So the search may go on forever even after you hit a goal.

        - 2nd problem: When you check the history, you may not recognise that you are hitting a state you have seen before. Again, this lead to the search going in a loop.
    - `equals(...)` returns `true` when the 2 states are not equal.
        - 1st problem: The search may stop at a non-goal state.
        - 2nd problem: When looking at the history, you may think that you are re-visiting a historical state and thus do not explore from there. You will miss a whole subtree of states which may contain a goal. So you may miss the solution while there is one.
- Complete the `equals(...)` (or `__eq___(...)`) method:
    - The input parameter is a `java.lang.Object` object.
    - The method returns `true` only if the followings are true:
        - If the input is a `Sq8State` object.
        - And the `tiles` attributes in both `Sq8State` objects contain the same values.
        - The `equals(Object)` method (or its Python counterpart the `__eq__(...)` function) is defined on `java.lang.Object` and is inherited by all Java objects. We are overriding the default behaviour and tailor it to our `Sq8State` object.
        - Some of you may made the mistake of having a premature true/false. The correct logic should be:
            - Using the row and column, scan the 2 `int` arrays (from both `Sq8State` objects) from the beginning to the end.
            - If there is a mismatch, you don't need to continue as it is sure that the 2 arrays will not be the same.
            - But you cannot conclude they are equal until you scan to the end and do not find any mismatch.
    - In Java:
        - Can you use "==" between 2 arrays to check for their equality?
            - In Java, arrays are objects. If you use "==" then you are testing if both values refer to the same object. This will give you "false negative" if these 2 arrays contain the same values in their elements.
        - If not, what is the correct way to check for equality?
            - The correct way is to scan through the arrays and check if their corresponding elements have the same value.
    - In Python:
        - Can you use "`is`"?
            - Similarly, "is" in Python checks for the object identifiers/references. It will give you "false negative" when the 2 arrays are different objects but have the same values in their elements.
        - If not, what is the correct way to check for equality?
            - In Pyhon, the correct way is to use "==" on the 2 arrays as this checks "if they are twin".
- Run the `TestEquals` class (or `testEquals.py`) to test your implementation.

- It should return true if the 2 `Sq8State` objects contain the same values in the 2D int array, even though they are 2 different objects.
- It should return false if the 2 arrays do not match in their values.

2.3.4. The `hashCode()` Method (or Python `__hash__()` Function)

The `aips.search` library uses the Java `HashSet` object to store all visited states as a history log. Thus a `hashCode()` method is needed for the `Sq8State` object.

- In general, the `hashCode()` method (or Python `__hash__(…)` function) is optional. Even if you don't implement it, the program will work. Customising this method improves the efficiency of hash-based data structures. e.g. `HashSet` in Java, dictionary in Python.
- See this link for the requirements of the `hashCode()` method:
  https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#hashCode--
- This method has been implemented for you. See the Java/Python code to understand how it is design and implemented.
  - The `hashCode()` method (or `__hash__(…)` in Python) is used to compute a hash value when the object is to be stored into a hash-table (or similar structure). The only requirement is that 2 objects which are the same (attribute-wise) must have the same hash code. You are free to invent any formula to calculate a hash code from an object's attributes.
  - It controls how effective the hash table data structure is when it comes to distributing the data across the different "buckets". A good hash function should evenly distribute the data.
  - Again, this is not an AI issue. This is a data structure issue.
- Question:
  - Do you think you can conclude that 2 states are equal if their hash codes are equal? Does it mean you can/cannot use hash code to test for equality of 2 states in the `equals(…)` method or `__eq__(...)` function?
    - 2 objects with the same attribute values will have the same hash code. However, 2 hash codes being the same does not guarantee the 2 objects have the same attribute values.
      - e.g. If you use the 1st letter of a person's name as the hash code, both "Peter" and "Patrick" will have "P", but their names are different.
    - That means checking for hash code alone in the `equals(…)` or `__eq__(…)` method is WRONG. It may give you "false positive".

2.4. The Sq8Problem Class

The `Sq8Problem` class extends the `SearchProblem` superclass in the `aips` library.

2.4.1.The goalState Attribute

`Sq8Problem` has an attribute `goalState`.

- Why isn't this `goalState` attribute in the `SearchProblem` superclass but in the subclass?
  - Because not all problem domains have explicit goals. i.e. a goal state that you can write down its values. Some domains may have an infinite number of goals and depend on the use of a "goal test". That's why the domain-independent `SearchProblem` class does not define a goal attribute but `isGoal(…)` method.
  - In the 8-puzzle, we have an explicit goal state. That's why we need an attribute in the problem class.

2.4.2.Goal Formulation/Test

The `isGoal(…)` method in the `Sq8SProblem` class tests if a given `State` object is a goal state. It returns a `true` or `false`.

- Question:
  - What are the problems if:
    - `isGoal(...)` returns `false` when the given state is a goal. (i.e. false negative)
    - You will miss the goal when you encounter it. So you will definitely fail to find the solution when it exists. The search may also loop forever as you missed your stopping condition.
    - `isGoal(...)` returns `true` when the given state is not a goal. (i.e. false positive)
    - You may falsely announce a solution which may/may not exist.
- Complete the method.
  - Hint: In a `Sq8Problem`, how can you tell if a given `State` (via the method parameter) is a goal state?
  - There are 2 ways to do it:
    - You can use the `equals(...)` method (or "==" in Python) to test if the state equals to the goal state (which is stored in the `goalState` attribute of the `Sq8Problem` object).
    - Another approach is to scan the `int` arrays in the 2 states and check if they contain the same values. This is a repeat of effort as the `equals(...)` method is already doing this.
- Run the `TestIsGoal` class (or `testGoal.py`) to test your `isGoal(...)` method.

2.5. Action Formulation

An action moves a tile into the blank space. Our `Sq8Action` class extends the `aips.search.Action` class to model an action in the 8-Puzzle problem.

This class has been done for you. Read `Sq8Action.java` (or `Sq8Action` class in `sq8.py`) to understand its design and implementation.

- Questions:
  - What attributes are in the `Sq8Action` class to store information of an action?
    - In this representation I put the original tile position row and column, target position row and column, and the tile to move in an action.
  - Do we have a `cost` attribute in `Sq8Action`?
    - We inherit this attribute from the `Action` superclass. That's why we don't need to declare it in the subclass.
  - If you want to change how the action is printed, what do you need to do?
    - Customise the `toString()` or `__str__(...)` method in your action subclass.

2.6. The `applyAction(Sq8Action)` Method

The `applyAction(Sq8Action)` method in the `Sq8State` class applies a `Sq8Action` to the current `Sq8State` object to return the next state.

- Complete this method.
- Hints:
  - Do not modify the current `Sq8State` object. Instead, clone it and modify the clone to give the next state.
    - It should be a deep copy. See Lecture 03, slide 37 for the potential problem if you do it wrongly.
  - A `Sq8Action` object contains the following attributes:
    - `tileToSlide`: The tile to slide. Its value is `1-8`.
    - `tileRow, tileCol`: The row and column of the tile in the `tiles` array.
    - `blankRow, blankCol`: The row and column of the blank in the `tiles` array.
    - Use the above information to swap the tile and blank in the new state.
- Run the `TestApplyAction` class (or `testApplyAction.py`) to test your implementation.
  - The original state should not be changed after an action is applied to it.

2.7. The `successor()` Method

The `successor()` method in the `Sq8State` class explores all possible actions on the current state and returns all valid action-state pairs as a list (of `ActionStatePair` objects). In other words, it "expands" a node.

- Questions:
  - What are the problems if:
    - The `successor()` method misses some valid actions from the current state.
      - You will miss some branches of the search tree. If a goal is in these subtrees, you will fail to find a solution while one exists.
    - The `successor()` method includes invalid actions from the current state.
      - Your "solution" may include invalid actions. So it is not a solution according to the game rules.
- Complete this method.
  - Hint:
    - At each state of the 8-Puzzle problem, there are 4 possible actions. However, depending on where the blank is, not all actions are valid.
    - There must be some tests to check if an action is applicable before you create the corresponding `Sq8Action` and `Sq8State` objects.
    - For a legal action, you will have to create the next state as a `Sq8Action` object. This can be easily done by invoking the `applyAction(...)` method.
    - It is important that your successor function finds all children. If you encounter a "no solution" in your search, maybe your successor function is missing some children. Try testing with a state with the blank at the centre of the board. The successor function should return all 4 children.
- Run the `TestSuccessor` class (or `testSuccessor.py`) to test your implementation.

2.8. Doing the Search

The final bit is to create the initial and goal states of the game.

- Complete the `main(...)` method of `RunSq8.java`. (or `run()` in `sq8.py`)
  - Create an initial state as a `Sq8State` object and assign it to the `initialState` local variable.
    - Hints:
      - You are free to choose any initial/goal state.
      - You can use the `Sq8State(int[][])` constructor.
  - Similarly, create a goal state.

If everything is fine, you can now run the `RunSq8` class (or `sq8.py`).

The default strategy is breadth-first. In Java it runs fast, but in Python it runs much slower. **If you are using Python, you may want to use an easier initial state.** For example:

```
631
742
85-
```

If you want to do depth-first search, you can override the `addChild(...)` method in `Sq8Problem` class (also the same method in `search.py`). You can compare the cost and number of nodes explored in the two search strategies.

If you use breadth-first search, it should give you the optimal solution (of the minimum number of moves).

If you use deep-first search, the result is very dependent on the order of your actions when you add it into the list in the successor function.

The slowness of the Python version may lie in the set that stores the history. A more efficient data structure may speed it up.

2.9. Alternative Representation

You are open to any representation of the state and action. Here are some alternative designs if you want to try:

- For `Sq8State`, instead of using a 2D `int` array, you can use a 1D array of 9 elements for the 9 tiles, with each element being a `Position` object of each tile.
  - A `Position` can have a `row` and `column` attributes (or `x`,`y` if you prefer).
- For `Sq8Action`, instead of having attributes of the original and target positions, plus the tile to move, you can only store the tile number to move. This is enough to tell us how the state will change.
- No matter what representation you use, DO NOT include attributes which are not needed, as this will confuse you.
  - In the state class, only include attributes which are needed to tell you the world configuration. They should be enough for you to write the `toString()` or `__str__(…)` method. DO NOT include anything about an action here.
  - In the action class, only include attributes required to tell you how to change an old state into a new state. Again, they should be enough for you to write the `toString()` or `__str__(…)` method. DO NOT include include anything of the state here.
  - In the problem class, only include attributes which can tell you if you have reached a goal state (required by the `isGoal(…)` method).

    - It is a common mistake to include unnecessary attributes in a class (especially for students who don't understand Object-oriented Programming.)
    - Attributes should be defined in a class where they make sense.
        - e.g. Having the 2D int array in the action class makes not sense as the 2D describes the state, not action.
        - Similarly, the tile to move (and its target position, if needed) should not appear in the state but the action class.

If your representations are different, the program code in `successor()`, constructor, etc. will be different. This will not affect the search engine in the `aips` library as it interacts with your domain-dependent classes through the specified methods.

- This is the beauty of Object-oriented Programming: The internal of an object does not matter as far as the "interface" of the object stays the same. (i.e. methods defined on the object do what they promised)
- The `aips` search engine invokes domain-specific version of these methods. As far as these methods behave the way they should, the search engine will work properly.
- If the search doesn't work, you need to check your customised methods to see if they are correct. The testing classes provided give you some hints on how to test your methods.