Generic Framework For State Space Search

15.12.2018

Sanketh R	01FB15ECS267
Sachit Batra	01FB15ECS251
Parashara R	01FB15ECS202

Objective

We propose to apply design patterns and principles of inheritance and composition to build a generic framework for the problem domain of state space search.

Patterns Used

1. **Strategy**

   Pattern

   The strategy pattern is a behavioral software design pattern that enables selecting an algorithm at runtime. Instead of implementing a single algorithm directly, code receives run-time instructions as to which in a family of algorithms to use.

   Usage

   Used for encapsulating the goal verification mechanism and the generation of successor states.

2. **Bridge**

   Pattern

   The bridge pattern is a design pattern used in software engineering that is meant to "decouple an abstraction from its implementation so that the two can vary independently".

   Usage

   Used for separating the implementation of search strategies in a search controller class from the search class. The search class maintains an abstract coupling with the search controller, thereby accessing its functionalities via a well-defined interface.

3. **Decorator**

   Pattern

   The decorator pattern is a design pattern that allows behavior to be added to an individual object, dynamically, without affecting the behavior of other objects from the same class.

   Usage

   Used for allowing reusable variations of the search control strategies.

4. **Flyweight**

Pattern

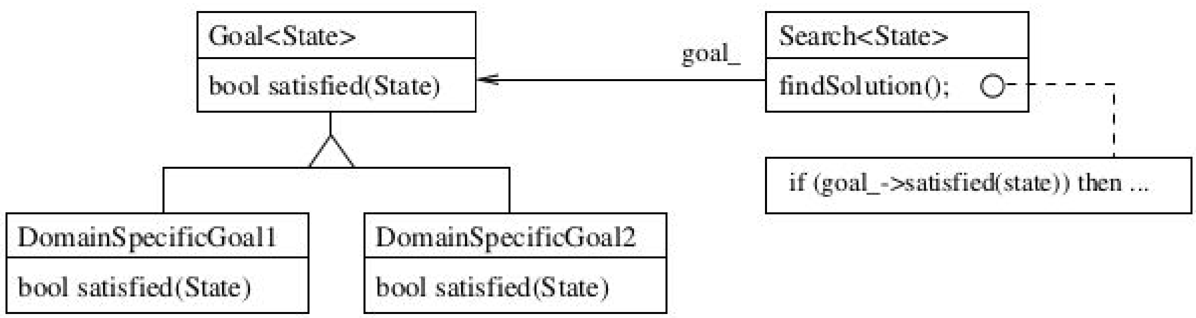
Flyweight is a software design pattern. A flyweight is an object that minimizes memory usage by sharing as much data as possible with other similar objects; it is a way to use objects in large numbers when a simple repeated representation would use an unacceptable amount of memory.

Usage

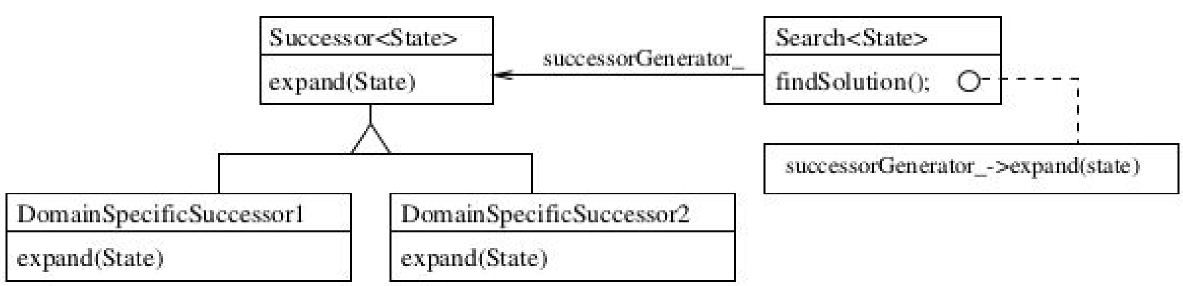
Used in creating the input graph for the distance search problem

The Big picture

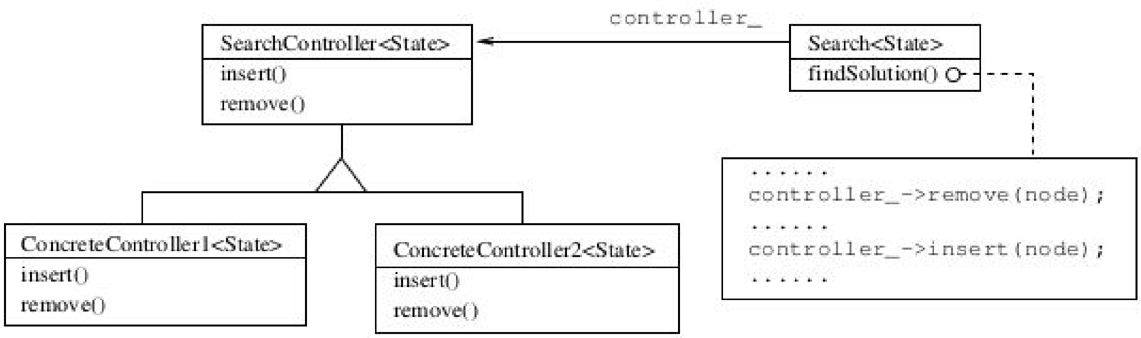
Goal verification



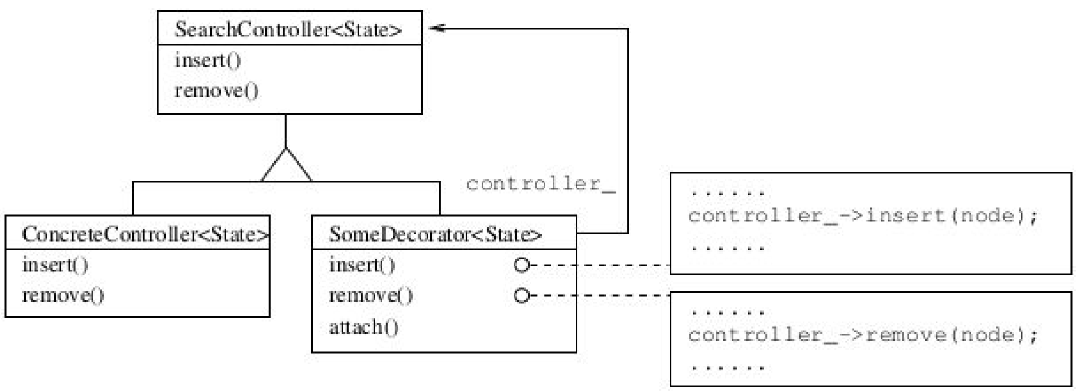
Successor generator



Search controller



Search Controller Decorator



Testing the framework

I. Jug water transfer problem

This problem deals with two jugs of different capacities (M,N) where the goal is to transfer,fill or empty water from the given jugs until we get a specific capacities in both the jugs (m,n) specified by the user.

```
The SearchPath is:
{ Prevstate: (null) ; Transformation : (start) ; CurrState : { Jug1 : 0; Jug2 : 0)}
{ Prevstate: ( Jug1 : 0; Jug2 : 0) ; Transformation : (fillJug2) ; CurrState : { Jug1 : 0; Jug2 : 5)}
{ Prevstate: ( Jug1 : 0; Jug2 : 5) ; Transformation : (transferJug2ToJug1) ; CurrState : { Jug1 : 3; Jug2 : 2)}
{ Prevstate: ( Jug1 : 3; Jug2 : 2) ; Transformation : (emptyJug1) ; CurrState : { Jug1 : 0; Jug2 : 2)}
{ Prevstate: ( Jug1 : 0; Jug2 : 2) ; Transformation : (transferJug2ToJug1) ; CurrState : { Jug1 : 2; Jug2 : 0)}
{ Prevstate: ( Jug1 : 2; Jug2 : 0) ; Transformation : (fillJug2) ; CurrState : { Jug1 : 2; Jug2 : 5)}
{ Prevstate: ( Jug1 : 2; Jug2 : 5) ; Transformation : (transferJug2ToJug1) ; CurrState : { Jug1 : 3; Jug2 : 4)}
{ Prevstate: ( Jug1 : 3; Jug2 : 4) ; Transformation : (emptyJug1) ; CurrState : { Jug1 : 0; Jug2 : 4)}
```

II. City distance problem

This problem deals with finding the minimum distance path from a given state to a goal/destination state. This is a commonly used search problem example. We applied Uniform Cost Path Search which greedily selects the minimum distance node at each step in the search. We applied both a DFS and BFS for this problem(using Priority Queue as the node store).

```
The SearchPath is:
{ Prevstate: (null) ; Transformation : (start) ; CurrState : ( City: A)}
{ Prevstate: ( City: A) ; Transformation : (1) ; CurrState : ( City: B)}
{ Prevstate: ( City: B) ; Transformation : (4) ; CurrState : ( City: D)}

Process finished with exit code 0
```

```
Added to the Controller: { Prevstate: (null) ; Transformation : (start) ; CurrState : ( City: A)}
The removed Search Node is: { Prevstate: (null) ; Transformation : (start) ; CurrState : ( City: A)}
The generated Search Node is{ Prevstate: ( City: A) ; Transformation : (1) ; CurrState : ( City: B)}
The generated Search Node is{ Prevstate: ( City: A) ; Transformation : (2) ; CurrState : ( City: C)}

Added to the Controller: { Prevstate: ( City: A) ; Transformation : (1) ; CurrState : ( City: B)}
Added to the Controller: { Prevstate: ( City: A) ; Transformation : (2) ; CurrState : ( City: C)}

The removed Search Node is: { Prevstate: ( City: A) ; Transformation : (1) ; CurrState : ( City: B)}
The generated Search Node is{ Prevstate: ( City: B) ; Transformation : (1) ; CurrState : ( City: A)}
The generated Search Node is{ Prevstate: ( City: B) ; Transformation : (3) ; CurrState : ( City: C)}
The generated Search Node is{ Prevstate: ( City: B) ; Transformation : (4) ; CurrState : ( City: D)}
The generated Search Node is{ Prevstate: ( City: B) ; Transformation : (1) ; CurrState : ( City: E)}

Added to the Controller: { Prevstate: ( City: B) ; Transformation : (1) ; CurrState : ( City: A)}
Added to the Controller: { Prevstate: ( City: B) ; Transformation : (3) ; CurrState : ( City: C)}
Added to the Controller: { Prevstate: ( City: B) ; Transformation : (4) ; CurrState : ( City: D)}
Added to the Controller: { Prevstate: ( City: B) ; Transformation : (1) ; CurrState : ( City: E)}

The removed Search Node is: { Prevstate: ( City: A) ; Transformation : (2) ; CurrState : ( City: C)}
The generated Search Node is{ Prevstate: ( City: C) ; Transformation : (2) ; CurrState : ( City: A)}
The generated Search Node is{ Prevstate: ( City: C) ; Transformation : (3) ; CurrState : ( City: B)}
The generated Search Node is{ Prevstate: ( City: C) ; Transformation : (5) ; CurrState : ( City: D)}
The generated Search Node is{ Prevstate: ( City: C) ; Transformation : (6) ; CurrState : ( City: E)}

Added to the Controller: { Prevstate: ( City: C) ; Transformation : (2) ; CurrState : ( City: A)}
Added to the Controller: { Prevstate: ( City: C) ; Transformation : (3) ; CurrState : ( City: B)}
Added to the Controller: { Prevstate: ( City: C) ; Transformation : (5) ; CurrState : ( City: D)}
Added to the Controller: { Prevstate: ( City: C) ; Transformation : (6) ; CurrState : ( City: E)}

The removed Search Node is: { Prevstate: ( City: B) ; Transformation : (1) ; CurrState : ( City: A)}
The generated Search Node is{ Prevstate: ( City: A) ; Transformation : (1) ; CurrState : ( City: B)}
The generated Search Node is{ Prevstate: ( City: A) ; Transformation : (2) ; CurrState : ( City: C)}

Not Added to the Controller: { Prevstate: ( City: A) ; Transformation : (1) ; CurrState : ( City: B)}
Not Added to the Controller: { Prevstate: ( City: A) ; Transformation : (2) ; CurrState : ( City: C)}

The removed Search Node is: { Prevstate: ( City: B) ; Transformation : (3) ; CurrState : ( City: C)}
The generated Search Node is{ Prevstate: ( City: C) ; Transformation : (2) ; CurrState : ( City: A)}
The generated Search Node is{ Prevstate: ( City: C) ; Transformation : (3) ; CurrState : ( City: B)}
The generated Search Node is{ Prevstate: ( City: C) ; Transformation : (5) ; CurrState : ( City: D)}
The generated Search Node is{ Prevstate: ( City: C) ; Transformation : (6) ; CurrState : ( City: E)}

Not Added to the Controller: { Prevstate: ( City: C) ; Transformation : (2) ; CurrState : ( City: A)}
Not Added to the Controller: { Prevstate: ( City: C) ; Transformation : (3) ; CurrState : ( City: B)}
Not Added to the Controller: { Prevstate: ( City: C) ; Transformation : (5) ; CurrState : ( City: D)}
Not Added to the Controller: { Prevstate: ( City: C) ; Transformation : (6) ; CurrState : ( City: E)}
```

```
The removed Search Node is: { Prevstate: ( City: B) ; Transformation : (4) ; CurrState : ( City: D)}
The SearchPath is:
{ Prevstate: (null) ; Transformation : (start) ; CurrState : ( City: A)}
{ Prevstate: ( City: A) ; Transformation : (1) ; CurrState : ( City: B)}
{ Prevstate: ( City: B) ; Transformation : (4) ; CurrState : ( City: D)}

Process finished with exit code 0
```