# CMPE 480 Project 2


## Şadi Uysal

## 2015400162


In this project,we expected to write some PDDL code to solve wolf-goat-cabbage problem. In this document, I will explain my strategy to solve the problem and also my code.

```
(:types place physobj - object
        package vehicle - physobj
        boat - vehicle
        location - place
 )
```

- For this problem, i started by defining types. I used package and vehicle as physical object and boat as a vehicle while location is a place.

```
(:predicates   (at ?obj - physobj ?loc - place)
         (in ?pkg - package ?veh - vehicle)
         (eat ?pkg - package ?pkg1 - package ))
```

- Then I defined predicates that I would need later. I used :

at(obj,loc) as returns True if pyhsical object “obj” is at that place “loc”

in(pkg,veh) as returns True if package “pkg” in that vehicle “veh”

eat(pkg,pkg1) as returns True if package “pkg” eats package “pkg1”

**(:action load-boat**

**:parameters (?pkg - package ?boat - boat ?loc - place)**

**:precondition (and (at ?boat ?loc) (at ?pkg ?loc) (not (exists ( ?pkg - package) (in ?pkg ?boat))))**

**:effect (and (not (at ?pkg ?loc)) (in ?pkg ?boat))**

**)**

- I defined three action as load-boat,drive-boat and unload-boat.
- Which load-boat action does is taking package,boat,place as parameter and if:

boat is at that location,

package is at that location,

there is not exist any package in the boat,

It takes effects as:

Changing value of at(pkg,loc) to False,

Changing value of in(pkg,boat) to True.

**(:action unload-boat**

**:parameters (?pkg - package ?boat - boat ?loc - place)**

**:precondition (and (at ?boat ?loc) (in ?pkg ?boat))**

**:effect (and (not (in ?pkg ?boat)) (at ?pkg ?loc))**

**)**

- Which unload-boat action does is taking package,boat,place as parameter and if:

boat is at that location,

package is in the boat,

It takes effects as:

Changing value of at(pkg,loc) to True,

Changing value of in(pkg,boat) to False.

**(:action drive-boat**

**:parameters (?boat - boat ?loc-from - place ?loc-to - place)**

**:precondition (and(at ?boat ?loc-from) (not (exists ( ?pkg1 ?pkg2 - package ) (and (at ?pkg1 ?loc-from) (at ?pkg2 ?loc-from) (eat ?pkg1 ?pkg2)))))**

**:effect (and (not (at ?boat ?loc-from)) (at ?boat ?loc-to))**

**)**

- Which drive-boat action does is taking boat and two place (loc-from,loc-to) as parameter and if:

boat is at that loc-from location,

there is not exist any package that would eat one another in the location loc-from

It takes effects as:

Changing value of at(boat,loc-from) to False,

Changing value of at(boat,loc-to) to True.

**(:objects**

**boat - boat**

**side1 side2 - location**

**wolf goat cabbage - package)**

**(:init**

**(at boat side1)**

**(at wolf side1)**

**(at goat side1)**

**(at cabbage side1)**

**(eat wolf goat)**

**(eat goat cabbage)**

**)**

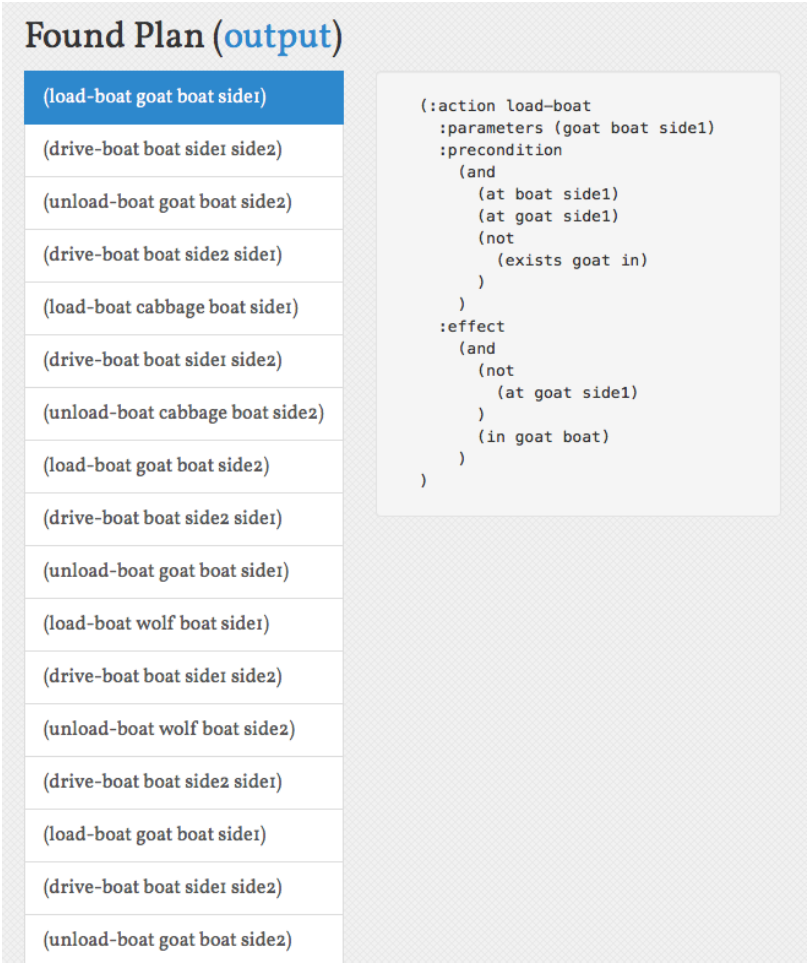
- In the problem file, I started by specifying boat as a boat, side1-side2 as locations and lastly packages as wolf-goat-cabbage.
- Then I specified initial conditions:

boat-wolf-cabbage at location side1,

Wolf eats goat,goat eats cabbage.

**(:goal (and (at wolf side2) (at goat side2) (at cabbage side2) ))**

- Lastly I defined goal, crossing the river as changing all of the package's location from side1 to side2




- My found plan is:

-Load goat to boat from side1, drive boat and unload boat to side2,drive back to side1.

-Load cabbage to boat from side1, drive boat and unload boat to side2.

- Load goat to boat from side2, drive boat and unload boat to side1.

- Load wolf to boat from side1, drive boat and unload boat to side2,drive back to side1.

- Load goat to boat from side1, drive boat and unload boat to side2.

- We achieved our goal conditions.