Assignment 1

The domain is a simple trucking domain with three types of objects: trucks, cities, and packages. Trucks are used to transport packages from one city to another. The domain has three operators: "drive ", "load", and "unload ". A truck can go directly from any city to any other city with a "drive" operator. Each truck can transport just one package from one city to another. To load a package in a truck, the truck must be empty, and the truck and package must be in the same city. When you unload a package from a truck, it becomes empty again.

a) Define this planning domain using PDDL.

b) Test your domain on a problem that requires moving packages from one location to another. Make sure that the package is in one city at a time. Having 2 trucks, one at each city.

Code:

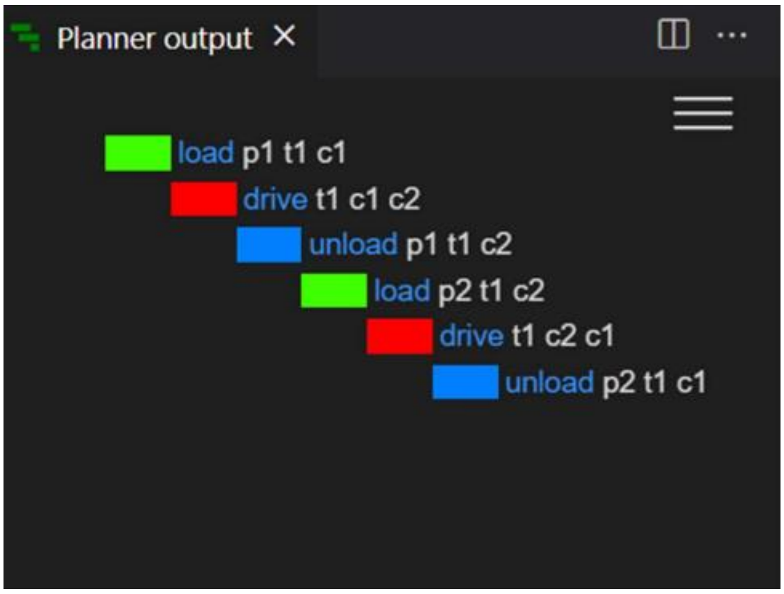
Problem:

```
(define (problem reincarnaθon) (:domain truck_kun)
(:objects
t1 ; trucks
c1 c2 ; ciθes
p1 p2 ;packages
)
(:init
 ;todo: put the iniθal state's facts and numeric values here
 (TRUCK t1)
 (CITY c1)
 (CITY c2)
 (PACKAGE p1)
 (PACKAGE p2)
 (truck-at t1 c1)
 (package-at p1 c1)
 (package-at p2 c2)
 (free t1)
)
(:goal
 (and
 ;todo: put the goal condiθon here
 (package-at p1 c2)
 (package-at p2 c1)
 )
)
;un-comment the following line if metric is needed
;(:metric minimize (???))
)
```

Domain:

```
;Header and descripƟon
(define (domain truck_kun)
(:requirements :strips :negaƟve-precondiƟons :equality)
(
 :predicates ;todo: define predicates here
 (TRUCK ?t) ; t is a truck
 (CITY ?c) ; c is a city
 (PACKAGE ?p) ; p is a package
 (truck-at ?t ?c) ; truck t is at c
 (loaded-on ?p ?t) ; package p is loaded on t
 (package-at ?p ?c) ; package p is at c
 (free ?t) ; truck t is free now
)
; (:funcƟons ;todo: define numeric funcƟons here
; )
;define acƟons here
(:acƟon drive ;drive t from ci to cj
 :parameters (?t ?ci ?cj)
 :precondiƟon (and (TRUCK ?t) (CITY ?ci) (CITY ?cj) (truck-at ?t ?ci) (not
(truck-at ?t ?cj)) )
 :effect (and (truck-at ?t ?cj) (not (truck-at ?t ?ci)))
)
(:acƟon load ; load p on t at c
 :parameters (?p ?t ?c)
 :precondiƟon (and (TRUCK ?t) (CITY ?c) (PACKAGE ?p) (free ?t) (package-at ?p
?c) (truck-at ?t ?c))
 :effect (and (loaded-on ?p ?t) (not (free ?t))
)
)
(:acƟon unload ; unload p from t on c
 :parameters (?p ?t ?c)
 :precondiƟon (and (TRUCK ?t) (CITY ?c) (PACKAGE ?p) (not (free ?t)) (loaded-
on ?p ?t) (truck-at ?t ?c))
 :effect (and (free ?t) (package-at ?p ?c) )
)
)
```

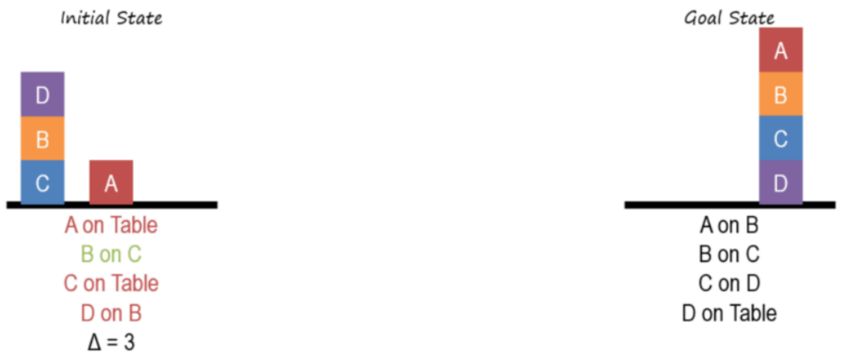
Output:



```
load p1 t1 c1
drive t1 c1 c2
unload p1 t1 c2
load p2 t1 c2
drive t1 c2 c1
unload p2 t1 c1
```

Assignment 2

In the "Block World“, the goal is to move blocks in a world with two locations to achieve a specific desired state or configuration.

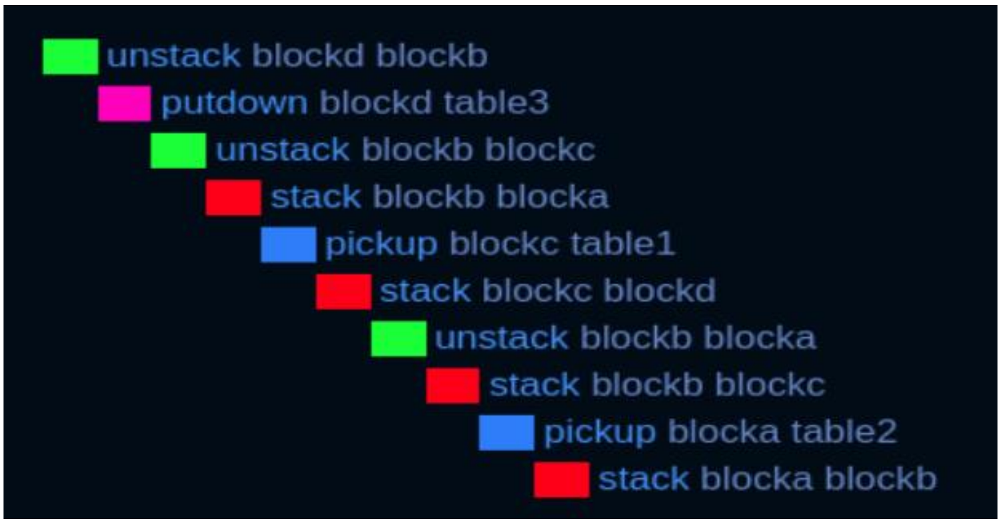


Code:
Problem

```
(define (problem problem2) (:domain domain2)
(:objects
 table1 table2 table3
 blocka blockb blockc blockd
)
(:init
 (TABLE table1) (TABLE table2) (TABLE table3)
 (BLOCK blocka) (BLOCK blockb) (BLOCK blockc) (BLOCK blockd)
 (arm_empty)
 (ontable blockc table1) (ontable blocka table2)
 (top_free blockd) (top_free blocka)
 (on blockd blockb)
 (on blockb blockc)
 (table_empty table3)
)
(:goal (and
(ontable blockd table3)
 (on blockc blockd)
 (on blockb blockc)
 (on blocka blockb)
 (top_free blocka)
 ;todo: put the goal condiθon here
))
;un-comment the following line if metric is needed
;(:metric minimize (???))
)
```

Domain:

```
(define (domain domain2)
;remove requirements that are not needed
(:requirements :strips :negaθve-precondiθons :equality)
(:predicates
(BLOCK ?x)
(TABLE ?x)
(on ?x ?y)
(ontable ?x ?y)
(top_free ?x)
(holding ?x)
(arm_empty)
(table_empty ?x)
)
(:funcθons ;todo: define numeric funcθons here
)
(:acθon stack
 :parameters (?x ?y)
 :precondiθon (and (BLOCK ?x) (BLOCK ?y) (top_free ?y) (holding ?x) (not
(arm_empty)))
 :effect (and (not (top_free ?y)) (not (holding ?x)) (arm_empty) (on ?x ?y)
(top_free ?x))
)
(:acθon unstack
 :parameters (?x ?y)
 :precondiθon (and (BLOCK ?x) (BLOCK ?y) (arm_empty) (on ?x ?y) (top_free ?x))
 :effect (and (not (arm_empty)) (not (on ?x ?y)) (holding ?x) (top_free ?y) )
)
(:acθon pickup
 :parameters (?x ?y)
 :precondiθon (and (BLOCK ?x) (TABLE ?y) (top_free ?x) (ontable ?x ?y)
(arm_empty) (not
(table_empty ?y)))
 :effect (and (not (ontable ?x ?y)) (not (arm_empty)) (holding ?x)
(table_empty ?y))
)
(:acθon putdown
 :parameters (?x ?y)
 :precondi0on (and (BLOCK ?x) (TABLE ?y) (holding ?x) (not (arm_empty))
(table_empty ?y))
 :effect (and (not (holding ?x)) (ontable ?x ?y) (arm_empty) (top_free ?x)
(not (table_empty ?y)))
)
)
```



```
unstack blockd blockb
 putdown blockd table3
  unstack blockb blockc
   stack blockb blocka
    pickup blockc table1
     stack blockc blockd
      unstack blockb blocka
       stack blockb blockc
        pickup blocka table2
         stack blocka blockb
```

Assignment 3

He who must not be named has regained power and is terrorizing the Wizarding World. In order to destroy him Harry Potter, who is at Hogwarts, must first destroy the last two Horcruxes. He also needs three magical objects (The Sword of Godric Gryffindor, Basilisk's fang, and The Elder Wand) in order to do so. For travelling from one location to another Harry uses his broom which lies at the room of requirements at Hogwarts. Salazar Slytherin's Locket is at Ministry of Magic which can be destroyed using The Sword of Godric Gryffindor (Hidden at Forest of Dean). The second Horcrux, Helga Hufflepuff's cup is at a Gringott's vault. The cup can be destroyed with Basilisk's fang in the Chamber of Secrets. Finally Harry must retrieve the Elder Wand from Olivander's to kill the person who need not to be named at the Forbidden Forest.

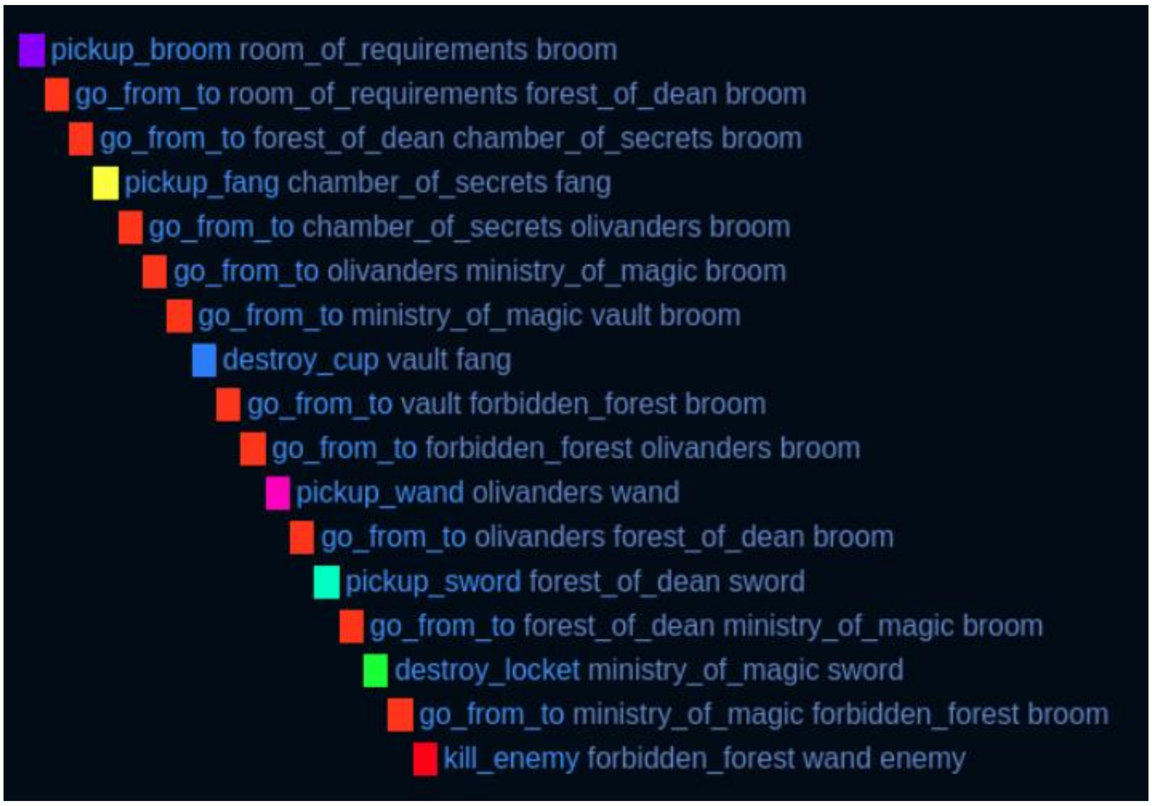
1. Model this problem in PDDL and check can Harry kill the person who need not to be named?

Problem:

```
(define (problem problem2) (:domain domain2)
(:objects
 table1 table2 table3
 blocka blockb blockc blockd
)
(:init
 (TABLE table1) (TABLE table2) (TABLE table3)
 (BLOCK blocka) (BLOCK blockb) (BLOCK blockc) (BLOCK blockd)
 (arm_empty)
 (ontable blockc table1) (ontable blocka table2)
 (top_free blockd) (top_free blocka)
 (on blockd blockb)
 (on blockb blockc)
 (table_empty table3)
)
(:goal (and
(ontable blockd table3)
 (on blockc blockd)
 (on blockb blockc)
 (on blocka blockb)
 (top_free blocka)
 ;todo: put the goal condiθon here
))
;un-comment the following line if metric is needed
;(:metric minimize (???))
)
```

Domain:

```
(define (domain harry)
; Remove unnecessary requirements
(:requirements :strips :negaθve-precondiθons :equality
:disjuncθve-precondiθons)
; Uncomment the following line if constants are needed
;(:constants )
(:predicates
; Predicates
(LOCATION ?x)
(MAGICALOBJECTS ?x)
(HORCRUXES ?x)
(ENEMY ?x)
(HARRY ?x)
(HAS ?x)
(BROOM ?x)
(WAND ?x)
(FANG ?x)
(SWORD ?x)
(LOCKET ?x)
(CUP ?x)
(ROOMOFREQUIREMENTS ?x)
(FORESTOFDEAN ?x)
(CHAMBEROFSECRETS ?x)
(OLIVANDERS ?x)
(MINISTRYOFMAGIC ?x)
(VAULT ?x)
(FORBIDDENFOREST ?x)
(LOCKETDESTROYED)
(CUPDESTROYED)
(ENEMYDEAD)
(AT ?x)
(DEAD ?x)
)
; Actions
(:action kill_enemy :parameters (?x ?y ?z)
:precondition (and (AT ?x) (FORBIDDENFOREST ?x) (not (DEAD ?z)) (ENEMY
?z)
(HAS ?y) (WAND ?y) (LOCKETDESTROYED) (CUPDESTROYED))
:effect (DEAD ?z))
(:action destroy_locket :parameters (?x ?y)
:precondition (and (AT ?x) (MINISTRYOFMAGIC ?x) (not
(LOCKETDESTROYED)) (HAS ?y) (SWORD ?y))
:effect (LOCKETDESTROYED))
(:action destroy_cup :parameters (?x ?y)
:precondition (and (AT ?x) (VAULT ?x) (not (CUPDESTROYED)) (HAS ?y)
(FANG ?y))
:effect (CUPDESTROYED))
(:action pickup_wand :parameters (?x ?y)
:precondition (and (AT ?x) (OLIVANDERS ?x) (not (HAS ?y)) (WAND ?y))
:effect (HAS ?y))
(:action pickup_fang :parameters (?x ?y)
:precondition (and (AT ?x) (CHAMBEROFSECRETS ?x) (not (HAS ?y)) (FANG
?y))
:effect (HAS ?y))
(:action pickup_sword :parameters (?x ?y)
:precondition (and (AT ?x) (FORESTOFDEAN ?x) (not (HAS ?y)) (SWORD
?y))
:effect (HAS ?y))
(:action pickup_broom :parameters (?x ?y)
:precondition (and (AT ?x) (ROOMOFREQUIREMENTS ?x) (not (HAS ?y))
(BROOM ?y))
:effect (HAS ?y))
(:action go_from_to :parameters (?x ?y ?z)
:precondition (and (AT ?x) (not (AT ?y)) (HAS ?z) (BROOM ?z))
:effect (and (AT ?y) (not (AT ?x)))
)
)
```



```
pickup_broom room_of_requirements broom
 go_from_to room_of_requirements forest_of_dean broom
  go_from_to forest_of_dean chamber_of_secrets broom
   pickup_fang chamber_of_secrets fang
    go_from_to chamber_of_secrets olivanders broom
     go_from_to olivanders ministry_of_magic broom
      go_from_to ministry_of_magic vault broom
       destroy_cup vault fang
        go_from_to vault forbidden_forest broom
         go_from_to forbidden_forest olivanders broom
          pickup_wand olivanders wand
           go_from_to olivanders forest_of_dean broom
            pickup_sword forest_of_dean sword
             go_from_to forest_of_dean ministry_of_magic broom
              destroy_locket ministry_of_magic sword
               go_from_to ministry_of_magic forbidden_forest broom
                kill_enemy forbidden_forest wand enemy
```