COMP90054 Assignment 2

Student Number: 1173913

Student Email: zhuoqingz@student.unimelb.edu.au

Student Name: Zhuoqing Zheng

Link to planning.domain:

http://editor.planning.domains/#read_session=c1cmLX5m59

PDDL Domain Files:

T1 Domain:

```
(define (domain BubbleTeaRobot)
    (:requirements :strips :typing :negative-preconditions :conditional-effects :equality :disjunctive-preconditions)

    (:types
        ice - obj
        tea - notBalls
        tapioca-balls - obj
        syrup - notBalls
        notBalls - obj
        obj flavor

    )
    (:constants
        ORIGINAL - flavor
        ICE - ice
        TEA - tea
        T-BALLS - tapioca-balls
        SYRUP - syrup
    )

    (:predicates
        (heated ?x - obj ?f - flavor)
        (inCup ?x - obj ?f - flavor)
        (hasTea)
        (hasBalls)
        (hasSyrup)
        (addedIce)
        (isHeated)

        (iced)
        (empty)
```

```
        (needBalls)
        (needSyrup)
        (need2Syrup)
        (needTea)
        (unheated ?x - obj ?f - flavor)
)

(:action addIce
        :parameters ()
        :precondition (and (not (iced)) (not (addedIce)) )
        :effect (and

                (inCup ICE ORIGINAL)
                (not (empty))
                (addedIce)
                (forall
                        (?x - notBalls ?f - flavor)
                        (when
                                (heated ?x ?f)
                                (and (unheated ?x ?f) (not (heated ?x ?f)) (not (iced)) )))
                (when (not (isHeated)) (iced))


        )

)
(:action addTea
        :parameters ()
        :precondition (and (not (hasTea)) (needTea))
        :effect (and
                (inCup TEA ORIGINAL)
                (unheated TEA ORIGINAL)
                (hasTea)
                (not (empty))
        )

)

(:action addBalls
        :parameters (?f - flavor)
        :precondition (and (not (hasBalls)) (or (needBalls) (need2Syrup)))
        :effect (and
                (inCup T-BALLS ?f)
                (unheated T-BALLS ?f)
```

```
            (not (empty))
            (hasBalls)
        )
    )
    (:action addSyrup
        :parameters (?f - flavor)
        :precondition (and (not (hasSyrup)) (or (needSyrup) (need2Syrup)))
        :effect (and
            (inCup SYRUP ?f)
            (unheated SYRUP ?f)
            (not (empty))
            (hasSyrup)
        )
    )


)
```

T2 domain:

```
(define (domain BubbleTeaRobot)
    (:requirements :strips :typing :negative-preconditions :conditional-effects :equality :disjunctive-preconditions)

    (:types
        ice - obj
        tea - notBalls
        tapioca-balls - obj
        syrup - notBalls
        notBalls - obj
        obj flavor

    )
    (:constants
        ORIGINAL - flavor
        ICE - ice
        TEA - tea
```

```
        T-BALLS - tapioca-balls
        SYRUP - syrup
)

(:predicates
        (heated ?x - obj ?f - flavor)
        (inCup ?x - obj ?f - flavor)
        (hasTea)
        (hasBalls)
        (hasSyrup)
        (addedIce)
        (isHeated)
        (mixed)
        (iced)
        (empty)

        (needBalls)
        (needSyrup)
        (need2Syrup)
        (needTea)
        (unheated ?x - obj ?f - flavor)
)

(:action addIce
        :parameters ()
        :precondition (and (not (iced)) (not (addedIce)) )
        :effect (and

                (inCup ICE ORIGINAL)
                (not (empty))
                (addedIce)
                (forall
                        (?x - notBalls ?f - flavor)
                        (when
                                (heated ?x ?f)
                                (and (unheated ?x ?f) (not (heated ?x ?f)) (not (iced)))))
                (when (not (isHeated)) (iced))



        )

)
(:action addTea
```

```
        :parameters ()
        :precondition (and (not (hasTea)) (needTea))
        :effect (and
                (inCup TEA ORIGINAL)
                (unheated TEA ORIGINAL)
                (hasTea)
                (not (empty))
        )

)

(:action addBalls
        :parameters (?f - flavor)
        :precondition (and (not (hasBalls)) (or (needBalls) (need2Syrup)))
        :effect (and
                (inCup T-BALLS ?f)
                (unheated T-BALLS ?f)
                (not (empty))
                (hasBalls)
        )
)
(:action addSyrup
        :parameters (?f - flavor)
        :precondition (and (not (hasSyrup)) (or (needSyrup) (need2Syrup)))
        :effect (and
                (inCup SYRUP ?f)
                (unheated SYRUP ?f)
                (not (empty))
                (hasSyrup)
        )
)
(:action heat
        :parameters ()
        :precondition (and (not (empty) ) (not (isHeated)))
        :effect (and (forall
                        (?x - ice)
                        (and (not (inCup ?x ORIGINAL)) (not (iced))))
                (forall
                        (?x - obj ?f - flavor)
                        (when
                                (inCup ?x ?f)
                                (and (heated ?x ?f) (not (unheated ?x ?f))))
                )
                (not (iced))
```

```
            (isHeated)
        )
    )

    (:action mix
        :parameters ()
        :precondition (and (not (empty)) (not (mixed)) (or (not (hasBalls)) (not (hasSyrup)) (need2Syrup)) (or (and (hasSyrup) (hasTea)) (and (hasSyrup) (hasBalls)) (and (hasTea) (hasBalls)))  )
        :effect (and
            (mixed)
            (forall
                (?b - tapioca-balls ?f - flavor)
                (when
                    (and (heated ?b ?f) (iced))
                    (and (not (inCup ?b ?f)) (not (heated ?b ?f)) (not (unheated ?b ?f)) (inCup SYRUP ?f) (unheated SYRUP ?f) )))
            (forall
                (?b - tapioca-balls ?f - flavor)
                (when
                    (and (heated ?b ?f) (not (iced)))
                    (and (not (inCup ?b ?f)) (not (heated ?b ?f)) (not (unheated ?b ?f)) (inCup SYRUP ?f) (heated SYRUP ?f) )))
            (forall
                (?b - tapioca-balls ?f - flavor)
                (when
                    (and (unheated ?b ?f))
                    (and (not (inCup ?b ?f)) (not (heated ?b ?f)) (not (unheated ?b ?f))  (inCup SYRUP ?f) (unheated SYRUP ?f) )))


        )
    )
)
```

T3 domain:

```
(define (domain BubbleTeaRobot)
    (:requirements :strips :typing :negative-preconditions :conditional-effects :equality :disjunctive-preconditions)

    (:types
        ice - obj
        tea - notBalls
        tapioca-balls - obj
```

```
        syrup - notBalls
        notBalls - obj
        obj flavor cup

)
(:constants
        ORIGINAL - flavor
        ICE - ice
        TEA - tea
        CUP1 - cup
        CUP2 - cup

        T-BALLS - tapioca-balls
        SYRUP - syrup
)

(:predicates
        (heated ?x - obj ?f - flavor)
        (inCupC1 ?x - obj ?f - flavor)
        (inCupC2 ?x - obj ?f - flavor)
        (inBoth ?x - obj ?f - flavor)
        (hasTeaC1)
        (hasBallsC1)
        (hasSyrupC1)
        (hasTea)
        (hasBalls)
        (hasSyrup)

        (addedIce)

        (icedC1)
        (emptyC1)
        (hasTeaC2)
        (hasBallsC2)
        (hasSyrupC2)
        (mixedC1)
        (mixedC2)
        (mixed)
        (icedC2)
        (emptyC2)
        (heatedCup ?x - cup)
        (needSyrup)
        (needBalls)
        (needTea)
```

```
        (need2Syrup)

        (unheated ?x - obj ?f - flavor)
)

(:action addIceC1
        :parameters ()
        :precondition (not (addedIce))
        :effect (and

                (inCupC1 ICE ORIGINAL)
                (inBoth ICE ORIGINAL)
                (not (emptyC1))
                (addedIce)
                (forall
                        (?x - notBalls ?f - flavor)
                        (when
                                (heated ?x ?f)
                                (and (unheated ?x ?f) (not (heated ?x ?f)) (not (icedC1)) )))
                (when (not (heatedCup CUP1)) (icedC1))

        )

)
(:action addIceC2
        :parameters ()
        :precondition (and (not (addedIce)))
        :effect (and

                (inBoth ICE ORIGINAL)
                (inCupC1 ICE ORIGINAL)
                (not (emptyC2))
                (addedIce)
                (forall
                        (?x - notBalls ?f - flavor)
                        (when
                                (heated ?x ?f)
                                (and (unheated ?x ?f) (not (heated ?x ?f)) (not (icedC2)))))
                (when (not (heatedCup CUP2)) (icedC2))
        )

)

(:action addTeaC1
```

```
        :parameters ()
        :precondition (and (not (hasTea)) (needTea))
        :effect (and
                (inCupC1 TEA ORIGINAL)
                (inBoth TEA ORIGINAL)
                (unheated TEA ORIGINAL)
                (hasTeaC1)
                (hasTea)
                (not (emptyC1))
        )

)
(:action addTeaC2
        :parameters ()
        :precondition (and (not (hasTea)) (needTea))
        :effect (and
                (inCupC2 TEA ORIGINAL)
                (inBoth TEA ORIGINAL)
                (unheated TEA ORIGINAL)
                (hasTeaC2)
                (hasTea)
                (not (emptyC2))
        )
)

(:action addBallsC1
        :parameters (?f - flavor)
        :precondition (and (not (hasBalls)) (or (needBalls) (need2Syrup)))
        :effect (and
                (inCupC1 T-BALLS ?f)
                (inBoth T-BALLS ?f)
                (unheated T-BALLS ?f)
                (not (emptyC1))
                (hasBallsC1)
                (hasBalls)
        )
)
(:action addBallsC2
        :parameters (?f - flavor)
        :precondition (and (not (hasBalls)) (or (needBalls) (need2Syrup)))
        :effect (and
                (inCupC2 T-BALLS ?f)
                (inBoth T-BALLS ?f)
                (unheated T-BALLS ?f)
```

```
            (not (emptyC2))
            (hasBallsC2)
            (hasBalls)
        )
)

(:action addSyrupC1
        :parameters (?f - flavor)
        :precondition (and (not (hasSyrup)) (or (needSyrup) (need2Syrup)))
        :effect (and
            (inCupC1 SYRUP ?f)
            (unheated SYRUP ?f)
            (not (emptyC1))
            (hasSyrup)
            (hasSyrupC1)
        )
)
(:action addSyrupC2
        :parameters (?f - flavor)
        :precondition (and (not (hasSyrup)) (or (needSyrup) (need2Syrup)))
        :effect (and
            (inCupC2 SYRUP ?f)
            (unheated SYRUP ?f)
            (not (emptyC2))
            (hasSyrup)
            (hasSyrupC2)
        )
)

(:action heatC1
        :parameters ()
        :precondition (and (not (emptyC1)) (not (heatedCup CUP1)) )
        :effect (and (forall
                (?x - ice)
                (and (not (inCupC1 ?x ORIGINAL)) (not (icedC1))))
            (forall
                (?x - obj ?f - flavor)
                (when
                    (inCupC1 ?x ?f)
                    (and (heated ?x ?f) (not (unheated ?x ?f))))
            )
            (not (icedC1))
            (heatedCup CUP1)
```

```
    )
)

(:action heatC2
    :parameters ()
    :precondition (and (not (emptyC2)) (not (heatedCup CUP2)))
    :effect (and (forall
            (?x - ice)
            (and (not (inCupC2 ?x ORIGINAL)) (not (icedC2))))
        (forall
            (?x - obj ?f - flavor)
            (when
                (inCupC2 ?x ?f)
                (and (heated ?x ?f) (not (unheated ?x ?f))))
        )
        (not (icedC2))
        (heatedCup CUP2)

    )
)

(:action tipToC1
    :parameters ()
    :precondition (not (emptyC2))
    :effect (and (forall
            (?x - obj ?f - flavor)
            (when
                (inCupC2 ?x ?f)
                (and (not (inCupC2 ?x ?f))
                    (inCupC1 ?x ?f))))
        (emptyC2)
        (not (hasSyrupC2))
        (not (hasBallsC2))
        (not (hasTeaC2))
        (not (icedC2))
        (not (heatedCup CUP2))
        (when
            (icedC2)
            (icedC1))
        (when
            (heatedCup CUP1)
            (not (icedC1)))

    )
```

```
    )
    (:action mixC1
        :parameters ()
        :precondition (and (not (emptyC1)) (not (mixedC1) ) (or (not (hasBalls)) (not (hasSyrup)) (need2Syrup))  (or (and (hasSyrupC1) (hasTeaC1)) (and (hasSyrupC1) (hasBallsC1)) (and (hasTeaC1) (hasBallsC1))) )
        :effect (and (when
                (and (hasSyrupC1) (hasTeaC1))
                (and (mixed) (mixedC1)) )
            (when
                (and (hasSyrupC1) (hasBallsC1))
                (and (mixed) (mixedC1))
            )
            (when
                (and (hasTeaC1) (hasBallsC1))
                (and (mixed) (mixedC1))
            )
            (forall
                (?b - tapioca-balls ?f - flavor)
                (when
                    (and (heated ?b ?f) (icedC1) (inCupC1 ?b ?f))
                    (and (not (inCupC1 ?b ?f)) (not (heated ?b ?f)) (not (unheated ?b ?f))  (inCupC1 SYRUP ?f) (unheated SYRUP ?f))))
            (forall
                (?b - tapioca-balls ?f - flavor)
                (when
                    (and (heated ?b ?f) (not (icedC1)) (inCupC1 ?b ?f))
                    (and (not (inCupC1 ?b ?f)) (not (heated ?b ?f)) (not (unheated ?b ?f))  (inCupC1 SYRUP ?f) (heated SYRUP ?f))))
            (forall
                (?b - tapioca-balls ?f - flavor)
                (when
                    (and (unheated ?b ?f)  (inCupC1 ?b ?f))
                    (and (not (inCupC1 ?b ?f)) (not (heated ?b ?f)) (not (unheated ?b ?f))  (inCupC1 SYRUP ?f) (unheated SYRUP ?f))))

        )
    )

(:action mixC2
        :parameters ()
        :precondition (and (not (emptyC2)) (not (mixedC2) ) (or (not (hasBalls)) (not (hasSyrup)) (need2Syrup))  (or (and (hasSyrupC2) (hasTeaC2)) (and (hasSyrupC2) (hasBallsC2)) (and (hasTeaC2) (hasBallsC2))) )
        :effect (and (when
                (and (hasSyrupC2) (hasTeaC2))
```

```
                (and (mixed) (mixedC2)))
            (when
                (and (hasSyrupC2) (hasBallsC2))
                (and (mixed) (mixedC2))
            )
            (when
                (and (hasTeaC2) (hasBallsC2))
                (and (mixed) (mixedC2))
            )
            (forall
                (?b - tapioca-balls ?f - flavor)
                (when
                    (and (heated ?b ?f) (icedC2) (inCupC2 ?b ?f))
                    (and (not (inCupC2 ?b ?f)) (not (heated ?b ?f)) (not (unheated ?b ?f)) (inCupC2 SYRUP ?f) (unheated SYRUP ?f) )))
            (forall
                (?b - tapioca-balls ?f - flavor)
                (when
                    (and (heated ?b ?f) (not (icedC2)) (inCupC2 ?b ?f))
                    (and (not (inCupC2 ?b ?f)) (not (heated ?b ?f)) (not (unheated ?b ?f))  (inCupC2 SYRUP ?f) (heated SYRUP ?f))))
            (forall
                (?b - tapioca-balls ?f - flavor)
                (when
                    (and (unheated ?b ?f)  (inCupC2 ?b ?f))
                    (and (not (inCupC2 ?b ?f)) (not (heated ?b ?f)) (not (unheated ?b ?f))  (inCupC2 SYRUP ?f) (unheated SYRUP ?f))))



        )
    )

)
```

Problem Files:

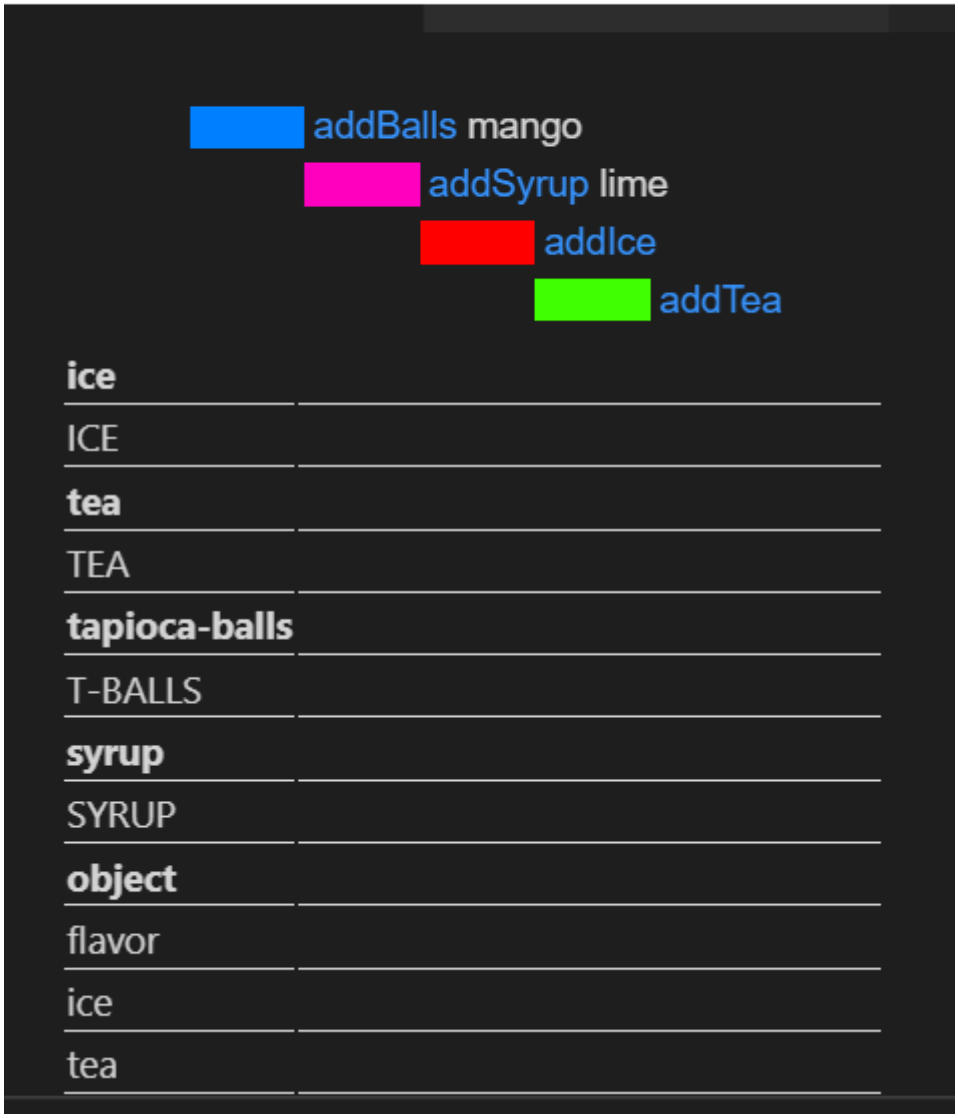
T1A:

```
(define (problem t1)
   (:domain BubbleTeaRobot)
```

```
    (:objects
        mango - flavor
        orange - flavor
        lime - flavor
    )
    (:init
        (empty) (needTea) (needBalls) (needSyrup))
    (:goal
        (AND
            (inCup tea ORIGINAL)
            (inCup SYRUP lime)
            (inCup T-BALLS mango)
            (iced)


            )
    )
)
```

Output:



T1B:

```
(define (problem t1)
   (:domain BubbleTeaRobot)

   (:objects
      mango - flavor
      orange - flavor
      lime - flavor
   )
   (:init
```

```
    (empty) (needTea) (need2Syrup))
  (:goal
    (AND
      (inCup tea ORIGINAL)
      (inCup SYRUP lime)
      (inCup SYRUP mango)
      (iced)


      )
  )
)
```

Output: no plan

```
--[4294967295 / 4]--
--[2 / 4]--
--[2 / 3]--
--[2 / 2]--
Total time: 3.8147e-09
Nodes generated during search: 77
Nodes expanded during search: 20
Plan found with cost: 6.0379e-39
BFS search completed
Planner found 0 plan(s) in 0.721secs.
```

T2A:

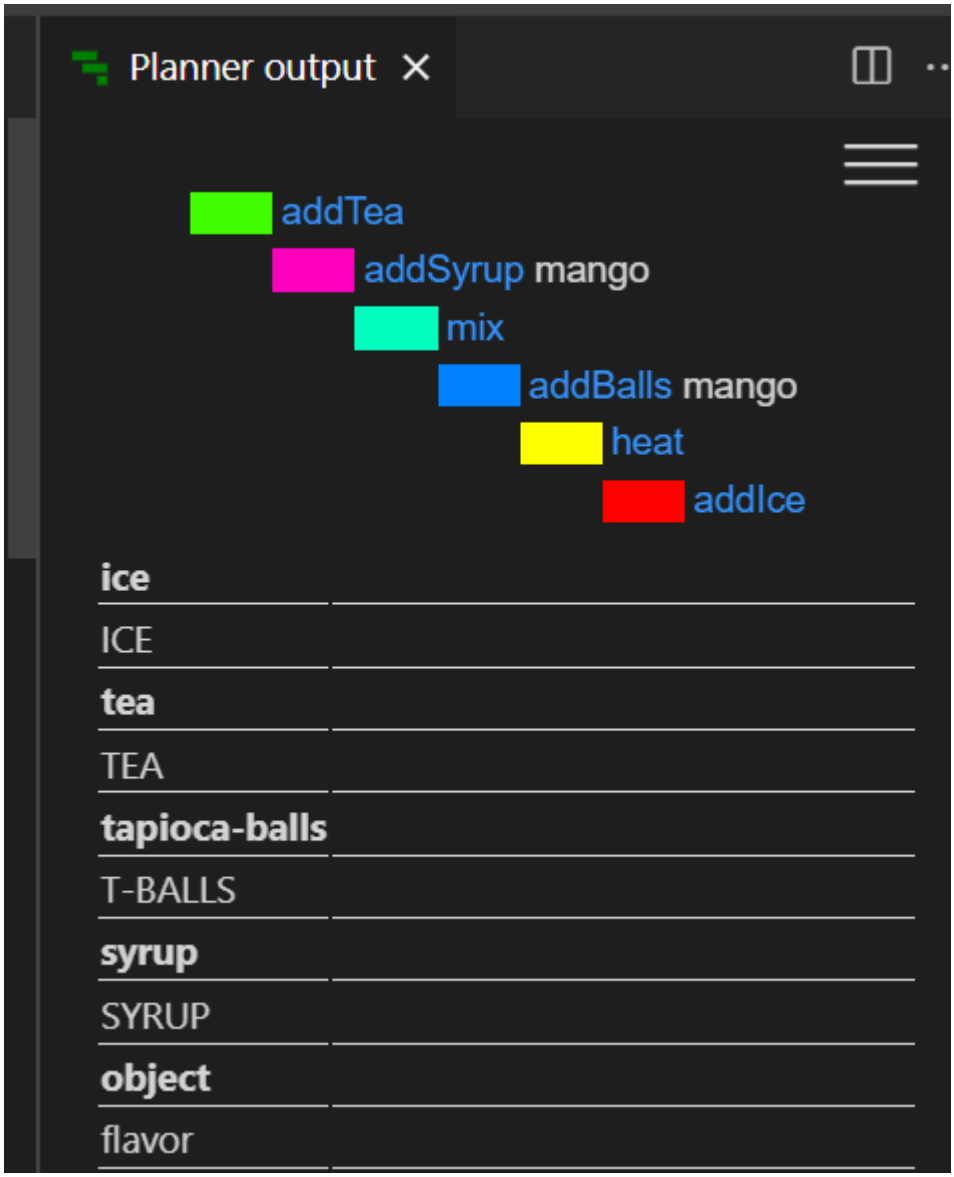
```
(define (problem t1)
  (:domain BubbleTeaRobot)

  (:objects
    mango - flavor
    orange - flavor
    lime - flavor
  )
  (:init
    (empty) (needTea) (needBalls) (needSyrup))
  (:goal
```

```
        (AND
            (inCup tea ORIGINAL)
            (unheated tea ORIGINAL)
            (inCup T-BALLS mango)
            (heated T-BALLS mango)
            (inCup SYRUP mango)
            (unheated SYRUP mango)

            (mixed)
            (not (iced))
            )
    )
)
```

Output:

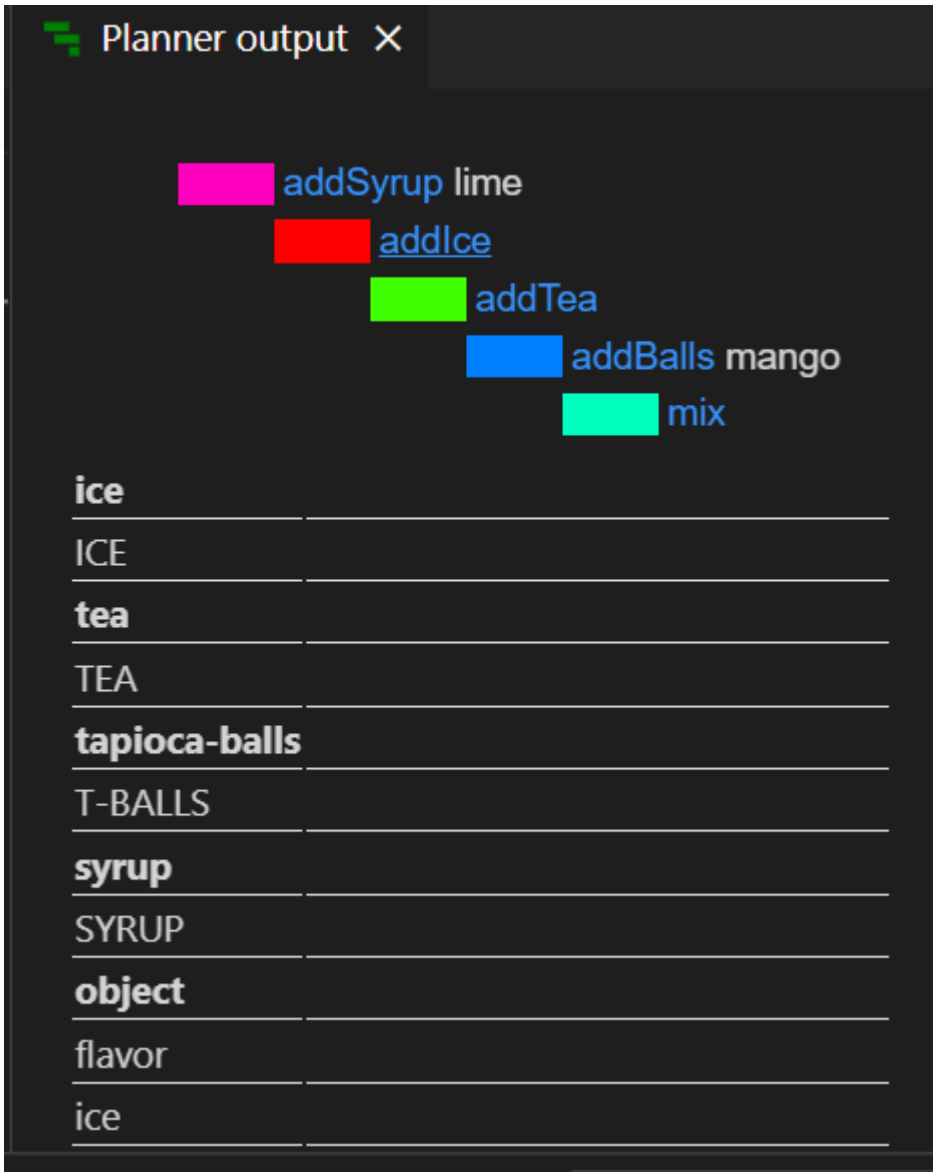


T2B:

```
(define (problem t1)
   (:domain BubbleTeaRobot)

   (:objects
      mango - flavor
      orange - flavor
      lime - flavor
   )
   (:init
      (empty) (needTea) (need2Syrup))
   (:goal
      (AND
         (inCup tea ORIGINAL)
         (unheated tea ORIGINAL)
         ;(inCup T-BALLS mango)
         ;(heated T-BALLS mango)
         (inCup SYRUP mango)
         (unheated SYRUP mango)
         (inCup SYRUP lime)
         (unheated SYRUP lime)
         (mixed)
         (iced)
         )
   )
)
```

Output:



T3A:

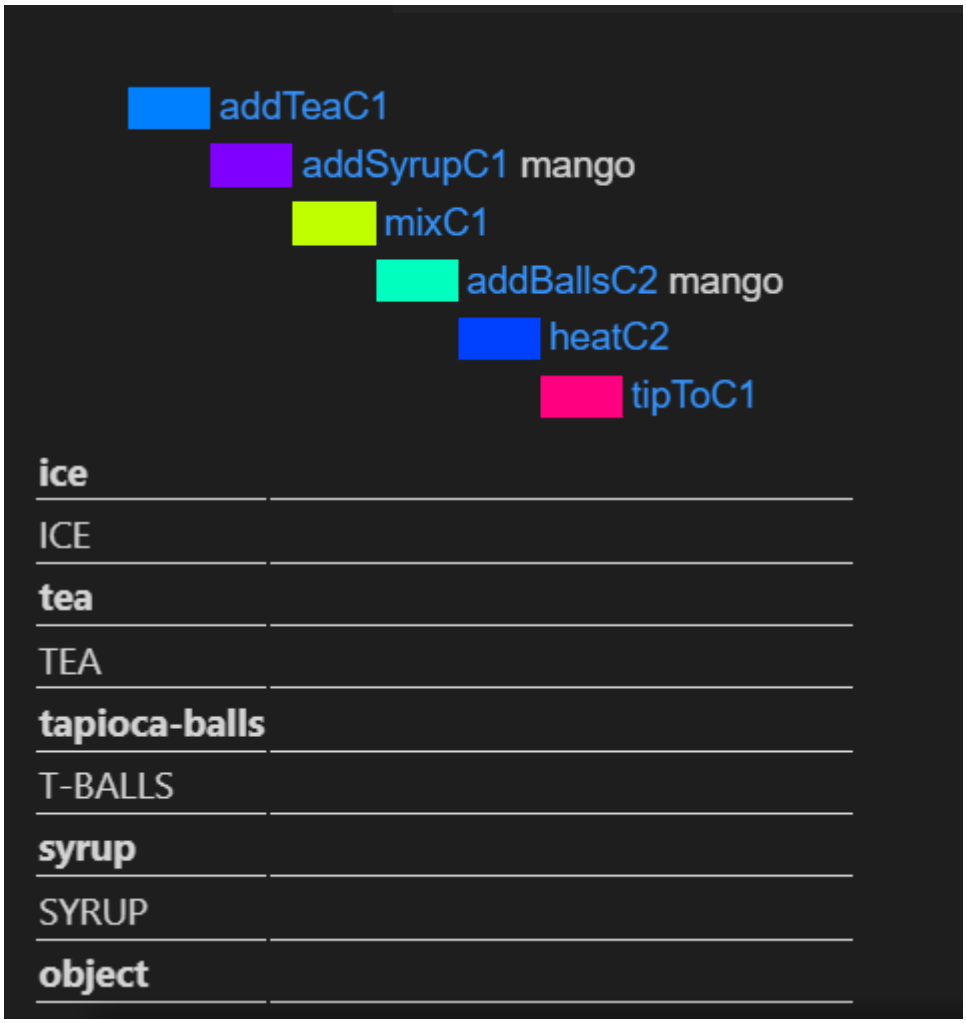
```
(define (problem t1)
   (:domain BubbleTeaRobot)

   (:objects
      mango - flavor
      orange - flavor
      lime - flavor
   )
   (:init
      (emptyC1) (emptyC2) (needSyrup) (needBalls) (needTea))
   (:goal
      (AND
```

```
        (inCupC1 TEA ORIGINAL)
        (unheated TEA ORIGINAL)
        (inCupC1 SYRUP mango)
        (unheated SYRUP mango)
        (inCupC1 T-BALLS mango)
        (heated T-BALLS mango)
        (not (icedC1))
        (not (heatedCup CUP1))
        (mixed)

        )
    )
)
```

Output:



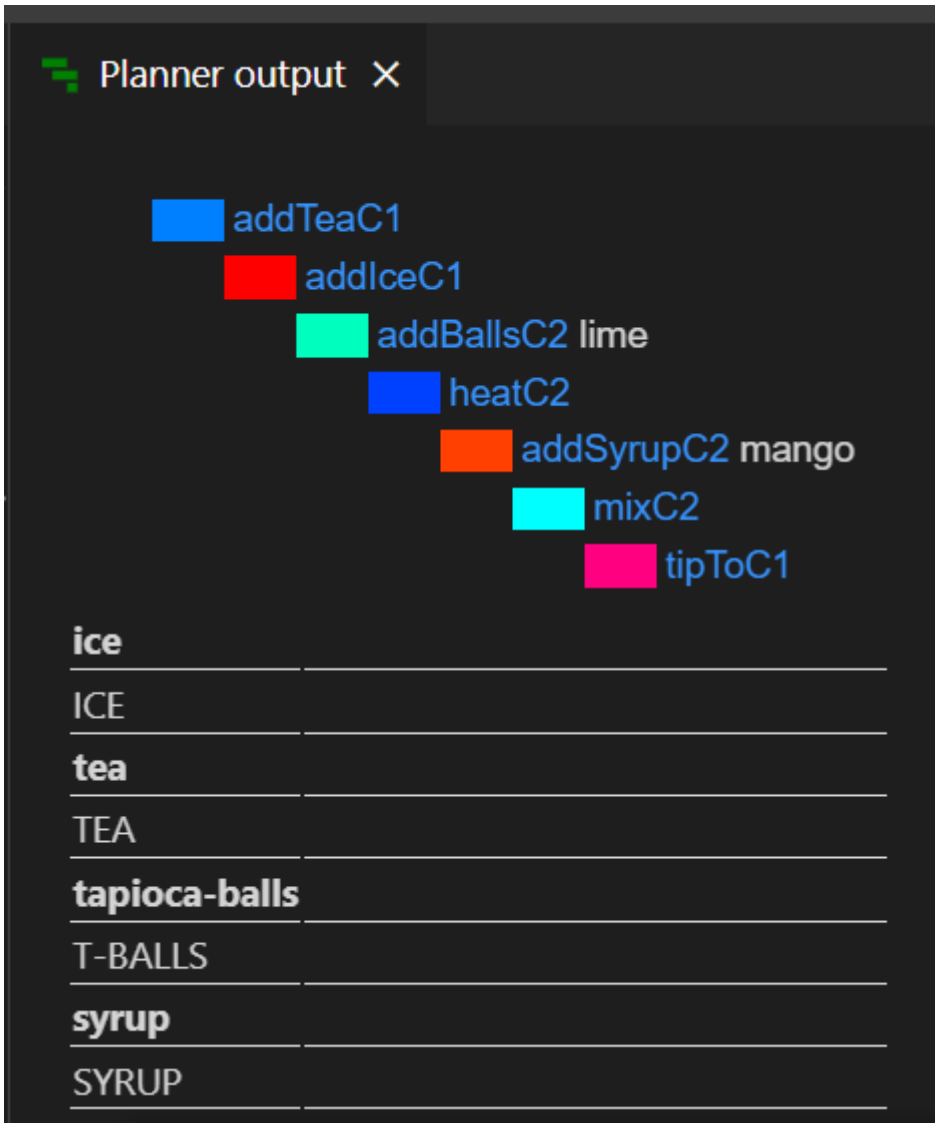
T3B:

```
(define (problem t3B)
   (:domain BubbleTeaRobot)

   (:objects
      mango - flavor
      orange - flavor
      lime - flavor
   )
   (:init
      (emptyC1) (emptyC2) (need2Syrup)    (needTea))
   (:goal
      (AND
         (inCupC1 TEA ORIGINAL)
         (unheated TEA ORIGINAL)
         (inCupC1 SYRUP mango)
         (unheated SYRUP mango)
         (inCupC1 SYRUP lime)
         (heated SYRUP lime)
         (icedC1)
         (not (heatedCup CUP1))
         (mixed)

         )
   )
)
```

Output:


Planner output
addTeaC1
addIceC1
addBallsC2 lime
heatC2
addSyrupC2 mango
mixC2
tipToC1
ice
ICE
tea
TEA
tapioca-balls
T-BALLS
syrup
SYRUP

END of Files

Self Evaluation:

Task1:

I could generate correct output plans for task 1. Task 1 was fairly straight-forward with simple actions to make. I check the plans by ensuring that all actions made are useful, all the goal states are achieved and that the actions are modeled correctly by my hand. The model is not robust as I thought clearly about the problem. I created 4 constants: ICE, SYRUP, TEA, T-Balls (stands for tapioca balls). I made flavors as object that can be customized at problem creating phase. That means when the domain remains, there are no limits on the number of flavors. Therefore, I can test my solution against tests other than the 2 tests provided. And it worked well.

The model was also concise, there are no useless predicates or actions in my model.

It was super fun learning modelling and abstraction, I think one very important thing is that typing is important. We can create different types to categorise different ingredients.

I think my solution for task1 should be 3/3.

Assumptions:1. for those ingredients that do not have flavors, they can be flavored as "ORIGINAL"

2. If the cup is heated, additional ice will melt

3. I created predicates needIce, needTea to make sure that all the ingredients added are needed, the robot will not add random ingredients that are not mentioned. They are specified in the :init of the problem.

Task2:

I could generate correct (realistic and logical) plans for task2. I checked the correctness of my task 2 solution by checking if the actions made in the plans are logical (e.g. There need to be at least 2 ingredients in the cup before mixing) and necessary. I also modeled the solution plan by hand to check if it meets all the goal states.

My model is not robust as it have succinct actions (no unnecessary duplicate actions in the solution plan). My model was thought very clearly to check if it is logical in real life. For example, the tapioca balls will remain heated after ice is added (they are still soft). So I can test my solution against tests other than the 2 tests provided. And it worked well.

My model was also clear and concise, there are no unnecessary duplicate actions or predicates in the domain file.

It was fun learning modeling and abstraction. I took a very deep look into predicate logic and condition checks. For example, for task2, there is no point to mix the cup twice as first: it is not logical (waste of power and electricity); second: unnecessary (one mix will make the drink "mixed")

I think my solution for task2 should be 5/5.

Assumptions:1. for those ingredients that do not have flavors, they can be flavored as "ORIGINAL"

2. If the cup is heated, additional ice will melt

3. I created predicates needIce, needTea to make sure that all the ingredients added

are needed, the robot will not add random ingredients that are not mentioned. They are specified in the :init of the problem.

4. There need to be at least 2 ingredients (except ice) for the robot to mix the drink.
5. if the drink is iced, heated balls will turn into unheated syrup
6. if the drink Is not iced, heated balls will turn into heated syrup
7. if the balls is unheated, it will turn into unheated syrup.

Task3:

I could generate correct (realistic and logical) plans for task2. I checked the correctness of my task 2 solution by checking if the actions made in the plans are logical and necessary (if the action is contributed to the goal state or it can be removed). I also modeled the solution plan by hand to check if it meets all the goal states. For example, I made sure that there can be only 1 type of ingredient added for each drink (instead of each cup). My model is not robust, it is created against a general variety of problems instead of just 2 tests provided. I created my own tests and iteratively improved my solution against the results.

My solution was also concise, there are no duplicate or unnecessary predicates or actions written in the domain file. My solution was exactly the amount of work needed to implement smart model for all different possible tests.

I learned that I am available to simplify and transform the requirement of the problem into easier model. For example, one cup is almost always needed to be unheated. Instead of thinking about a random cup, I can just make 1 cup that is always used to hold the drink for customers. And hence there can be less predicates and condition checks. Also, during debugging my code, I had learnt the pddl language and modelling/abstraction a lot deeper.

I think my solution for Task 3 is 2/2

Assumptions:1. for those ingredients that do not have flavors, they can be flavored as “ORIGINAL”

2. If the cup is heated, additional ice will melt
3. I created predicates needIce, needTea to make sure that all the ingredients added are needed, the robot will not add random ingredients that are not mentioned. They are specified in the :init of the problem.
4. There need to be at least 2 ingredients (except ice) for the robot to mix the drink.
5. if the drink is iced, heated balls will turn into unheated syrup
6. if the drink Is not iced, heated balls will turn into heated syrup
7. if the balls is unheated, it will turn into unheated syrup.
8. the drink will always be put in CUP1, to present to the customer
9. When the ice is tiped into another cup, if another cup is heated, it is still not iced