实验报告

（2023~2024 学年第二学期）

课程名称 人工智能课程设计

实验名称 课程设计作业 Project3

姓名 范潇 学号 2254298

1. 项目概述

1.1. 主要内容与目标

本次项目的主要内容为基于逻辑推理的智能体实现，要求我们用逻辑语句来对吃豆人游戏规则进行建模，并帮助吃豆人完成路径规划、定位、建立地图以及 SLAM 这四个任务。

本项目的八个问题可以分为三部分。

第一部分旨在熟悉该项目中所用到的 Expr 和 PropSymbolExpr 类的使用，同时考察了命题逻辑的基本内容。

第二部分的任务是利用逻辑语句吃豆人的游戏规则进行建模，并帮助吃豆人完成路径规划以及收集所有食物这两项目标。

第三部分的任务围绕 SLAM 问题展开，要求先分别帮助吃豆人完成定位和建立地图的任务，然后综合在一起，实现 SLAM 的功能。

1.2. 已有代码

下面对已有代码中与本项目的具体实现相关的代码进行分析。

1.2.1. logic.py

该文件中提供了以下有关逻辑的类：

1. Expr：是关于谓词逻辑表达式的类
2. PropSymbolExpr：用于构造谓词逻辑表达式中的符号
3. SpecialExpr：用于构造一些可能用到的常项

同时还提供了以下在本项目中需要调用的函数：

1. parseExpr：用于解析谓词逻辑表达式中的符号
2. pl_true：判断所给表达式是否满足给定模型，这里要求模型对所有命题赋值。
3. to_cnf：将给定表达式转换为合取式
4. conjoin：用合取将所给语句连接为合取式
5. disjoin：用析取将所给语句连接为析取式

1.2.2. logicPlan.py

该文件用于存储各种有关逻辑的功能的实现，已有代码中给出了以下函数

1. findModel：将给定的表达式转化为合取式，并通过 SAT 求解出一个满足的模型

2. SLAMSuccessorAxiomSingle：该继承公理是 SLAM 独有的，用于表示智能体在给定时刻位于给定位置的可能原因。返回的逻辑语句分为两部分：1）上一时刻成功移动，则上一时刻不在给定位置，且给定位置没有墙壁，同时移动行为使得智能体移动到该位置 2）上一时刻移动失败，则上一时刻吃豆人便在该位置，且移动方向上有墙壁阻挡

3. SLAMSuccessorAxioms：遍历所有的非墙壁位置，对给定时刻调用 SLAMSuccessorAxiomSingle

4. sensorAxioms：返回语句的语义是“被某个方向上的墙壁阻挡了等价于吃豆人在某个无墙壁的位置且在这个方向上紧邻着墙壁”

5. fourBitPerceptRules：将智能体的感知解析为逻辑语句，输入的感知是四个方向上墙壁存在与否

6. numAdjWallsPerceptRules：将智能体的感知解析为逻辑语句，这是 SLAM 特有的，输入的感知是三个方向上的墙壁数量

7. SLAMsensorAxioms：SLAM 所特有的公理，所增加的部分的语义是“感知到了四周有若干处墙壁等价于四周有相同数量的墙壁”，这里利用所有可能的组合来涵盖各种可能

8. allLegalSuccessorAxioms：遍历所有的非墙壁位置，对给定时刻调用 pacmanSuccessorAxiomSingle

1.2.3. searchAgents.py

该文件中存储的是关于智能体相关的代码。在项目中，需要使用智能体类的以下成员：

1. actions：这里存储的是智能体在各个时刻所规划的动作

2. moveToNextState：让智能体执行指定动作

2. 第一部分

2.1. 问题概述

该部分包含第 1，2 小题，要求完成以下任务：

1. 利用 Expr 类表示由以下语句形成的合取式

   (a)
   - $A \vee B$
   - $\neg A \leftrightarrow (\neg B \vee C)$
   - $\neg A \vee \neg B \vee C$

   (b)
   - $C \leftrightarrow (B \vee D)$
   - $A \rightarrow (\neg B \wedge \neg D)$
   - $\neg(B \wedge \neg C) \rightarrow A$
   - $\neg D \rightarrow C$

2. 用 PropSymbolExpr 类构造符号，并形成含有以下语义的逻辑语句，最终连接形成合取式

   (a) Pacman is alive at time 1 if and only if he was alive at time 0 and he was not killed at time 0 or he was not alive at time 0 and he was born at time 0.

   (b) At time 0, Pacman cannot both be alive and be born.

   (c) Pacman is born at time 0.

3. 补全 findModelCheck 函数，使得它的返回值和 findeModel(Expr('a')) 一致。该问题考察的是对于 findModel 函数功能的理解

4. 补全 entails 函数，其功能是判断给定前提是否蕴含指定结论

5. 补全 plTrueInverse 函数，其功能是判断给定赋值是否满足所给语句的否定。

6. 补全 atLeastOne 函数，其功能是接受一列文字，返回一个合取式，该合取式为真当且仅当给定文字中至少有一个文字为真

7. 补全 atMostOne 函数，其功能是接受一列文字，返回一个合取式，该合取式为真当且仅当给定文字中至多有一个文字为真

8. 补全 exactlyOne 函数，其功能是接受一列文字，返回一个合取式，该合取式为真当且仅当给定文字中恰好有一个文字为真

在补全 atLeastOne,atMostOne,exactlyOne 函数中，不允许调用 to_cnf 及其相关函数。

2.2. 算法设计

满足由单一文字构成的表达式的模型仅由一个 True 组成。

判断蕴含关系可以使用反证法，即只需要判断所给前提和所给结论的否定所形成的合取式是否为可满足式即可。如果时，则说明蕴含关系不成立，否则成立。

给定一列文字，其中至少有一个为真当且仅当他们的析取式为真，而由于文字之间的析取式作为整体时可以视为合取式，所以 atLeastOne 函数只需要返回给定文字的合取式即可。

给定一列文字，其中至多有一个为真当且仅当任意两个文字之间的合取式不为真，为了转换为合取式，需要利用德摩根律，转化为任意两个文字的否定之间的析取式为真。

给定一列文字，其中恰好有一个为真当且仅当其中至少有一个为真且其中至多有一个为真。

2.3. 算法实现

在完成该项目的全部过程中，为了便于调试并且与 autograder 的答案匹配，均调用 conjoin 和 disjoin 函数来分别构造合取式和析取式，而不是使用 & 和 | 运算符。

在完成任务 1 时，我从 logic.py 中引入了 SpecialExpr 类，来表示简化代码。

由于 findModel 以 {符号: 赋值} 的形式返回满足给定合取式的模型，所以，首先要构造'a' 对应的符号，而这可以用提供的 dummyClass 实现，然后将其对应的值设为 True 即可。

实现 plTrueInverse 时，只需调用提供的 plTrue 函数，将参数 inverse_statement 的否定作为参数输入即可。

在实现 atMostOne 时，借助 combinations 生成器来遍历给定文字的所有二组合。

```python
from logic import SpecialExpr

TRUE, FALSE = tuple(map(SpecialExpr, ['TRUE', 'FALSE']))
ZERO, ONE, TWO = tuple(map(SpecialExpr, [0, 1, 2]))
A, B, C, D, E, F, G, P, Q = tuple(map(Expr, 'ABCDEFGPQ'))

def sentence1() -> Expr:
    return conjoin([
        disjoin([A, B]),
        (~ A % disjoin([~ B, C])),
        disjoin(~ A, ~B, C)
    ])

def sentence2() -> Expr:
    return conjoin([
        C % disjoin([B, D]),
        A >> (conjoin([~B, ~D])),
        ~conjoin([B, ~C]) >> A,
        ~D >> C
    ])

```

```python
def sentence3() -> Expr:
    PacmanAlive_0 = PropSymbolExpr('PacmanAlive', time=0)
    PacmanAlive_1 = PropSymbolExpr('PacmanAlive', time=1)
    PacmanBorn_0 = PropSymbolExpr('PacmanBorn', time=0)
    PacmanKilled_0 = PropSymbolExpr('PacmanKilled', time=0)
    return conjoin([
        PacmanAlive_1 % disjoin([conjoin([PacmanAlive_0, ~PacmanKilled_0]),
           conjoin(~PacmanAlive_0, PacmanBorn_0)]),
        ~conjoin([PacmanAlive_0, PacmanBorn_0]),
        PacmanBorn_0
    ])

def findModelCheck() -> Dict[Any, bool]:
    class dummyClass:
        def __init__(self, variable_name: str = 'A'):
            self.variable_name = variable_name

        def __repr__(self):
            return self.variable_name
    return {dummyClass('a'): True}

def entails(premise: Expr, conclusion: Expr) -> bool:
    return findModel(conjoin([premise, ~conclusion])) == False

def plTrueInverse(assignments: Dict[Expr, bool], inverse_statement: Expr) ->
    bool:
    return pl_true(~inverse_statement, assignments)

def atLeastOne(literals: List[Expr]) -> Expr:
    return disjoin(literals)  # 已经能够保证是合取式

from itertools import *

def atMostOne(literals: List[Expr]) -> Expr:
    return conjoin([disjoin([~atom1,~atom2]) for atom1,atom2 in combinations
       (literals, 2)])

def exactlyOne(literals: List[Expr]) -> Expr:
    return conjoin( [atLeastOne(literals) ,atMostOne(literals)])
```

2.4. 实验结果

成功通过 Question1 和 Question2 的全部测试，实验结果截图见图 2.1。这里所使用的到的测试用例

```
Question q1
===========

*** PASS: test_cases\q1\correctSentence1.test
***     PASS
*** PASS: test_cases\q1\correctSentence2.test
***     PASS
*** PASS: test_cases\q1\correctSentence3.test
***     PASS
*** PASS: test_cases\q1\entails.test
***     PASS
*** PASS: test_cases\q1\findModelCheck.test
***     PASS
*** PASS: test_cases\q1\findModelSentence1.test
***     PASS
*** PASS: test_cases\q1\findModelSentence2.test
***     PASS
*** PASS: test_cases\q1\findModelSentence3.test
***     PASS
*** PASS: test_cases\q1\plTrueInverse.test
***     PASS

### Question q1: 2/2 ###
```

```
Question q2
===========

*** PASS: test_cases\q2\atLeastOne.test
***     PASS
*** PASS: test_cases\q2\atLeastOneCNF.test
***     PASS
*** PASS: test_cases\q2\atMostOne.test
***     PASS
*** PASS: test_cases\q2\atMostOneCNF.test
***     PASS
*** PASS: test_cases\q2\exactlyOne.test
***     PASS
*** PASS: test_cases\q2\exactlyOneCNF.test
***     PASS

### Question q2: 2/2 ###
```

图 2.1: Question1&2 实验结果

较为基础，检查了上述函数是否能够返回正确的结果。其中，对于 atLeastOne,atMostOne 和 exactlyOne 这三个函数的检测方式是检验包含不同数量的 True 的模型是否满足由函数返回的表达式。

3. 第二部分

3.1. 问题概述

该部分包含第 3、4、5 小题。

该部分中要求我们通过完成 pacmanSuccessorAxiomSingle 和 pacphysicsAxioms 来对吃豆人游戏规则进行建模。在此基础之上完成 checkLocationSatisfiability，其功能是给定上一时刻的吃豆人位置，判断下一时刻吃豆人是否有可能（不）达到指定位置，如果有可能，返回一个满足的模型，否则返回 False。然后，我们需要利用逻辑来帮助吃豆人规划从起始位置到目标位置的路线。最后，需要利用逻辑来帮助吃豆人规划收集所有食物的方案。

3.2. 算法设计

参考 SLAMSuccessorAxiomSingle，pacmanSuccessorAxiomSingle 用于描述吃豆人如何在时刻 t 时到达 (x, y) 处。由于这里不允许吃豆人停止，该事件发生当且仅当吃豆人上一时刻在相邻位置，并且在上一时刻朝着该位置移动。

按照题目要求，pacphysicsAxioms 中需要包含以下内容：

1. 对于所有的坐标，“有墙”蕴含着“吃豆人不在这里”
2. 在每一时刻吃豆人恰好在一个位置（这可以通过 exactlyOne 函数实现）
3. 在每一时刻吃豆人恰好执行一个动作（这可以通过 exactlyOne 函数实现）
4. sensorModel 返回的语句
5. successorAxioms 返回的语句

按照题目要求，在 checkLocationSatisfiablity 中，首先向知识库 KB 中添加以下内容：

1. pacphysics_axioms 返回值
2. 吃豆人初始位置
3. 吃豆人执行了 action0
4. 吃豆人执行了 action1

这里 action0 和 action1 分别是在时刻 0 和时刻 1 执行。这里的 action1 的用途只是为了确保由 SAT 返回的模型和 autograder 中设定的一致。由于需要返回具体的模型，所以最后需要调用 findModel，将 KB 以及询问的内容，即吃豆人在时刻 1（不）位于指定位置，传入。

positionLogicPlan 算法的伪代码在下一页中给出。核心思想是：每一时刻，添加对于吃豆人位置的约束，然后判断吃豆人在当前时刻达到目标位置，如果能，则返回对应的行动，否则，添加该时刻中对于吃豆人动作的约束，以及转移公理，从而完成了从这一时刻到下一时刻的过渡。

foodLogicPlan 算法的整体逻辑以及伪代码和 positionLogicPlan 的类似。区别在于，知识库 KB 初始化时，需要添加食物的初始信息。同时，判断给定时刻是否存在解的条件时该时刻的吃豆人收集掉了所有

Algorithm 3.2.1: positionLogiPlan

```
KB = []
KB.append(pacmanInitialLocation)
for t in range (50) do
    print (t)// 因为运行时间较长，用于获取当前进度
    KB.append(pacmanInExactlyOneLocationSentence(t))// 吃豆人恰好在一个位置
    result = findModel(KB+pacmanInGoalPositionSentence(t))// 是否存在模型，其中吃豆人到达目标位置
    if result then // 存在解，则从中提取行动序列
        return extractActionSequence(result)
    end if
    KB.append(pacmanTakeExactlyOneActionSentenc(t))// 吃豆人恰好执行一个动作
    forall coordinate in non_wall_coords do
        KB.append(pacmanSuccessorAxiomsSingle(coordinate,t+1))// 转移公理
    end forall
end for
```

食物，即此时地图中没有任何食物存在。并且，两个时刻之间的状态不仅仅由 pacmanSuccessorAxiom 进行约束，还需要利用 foodSuccessorAxiom 对食物的状态进行约束，即：给定一个位置，如果在上一时刻没有食物，在这一时刻也没有食物；如果在上一时刻有食物，那么如果这一时刻吃豆人在该位置上，这一时刻便没有食物了，否则仍然有食物。

3.3. 算法实现

在 pacphysicsAxioms 函数中，只有当 t 不为 0 时，才有可能向 KB 中添加这一时刻的转移公理，这是因为对于初始时刻而言，并没有上一时刻，自然不受转移公理的约束。

在 checkLocationSatisfiability,positionLogicPlan 和 foodLogicPlan 函数中，查询某一结论是否存在模型满足它时，需要将 KB 和结论合取在一起，然后传入 findModel 函数中。这里需要注意的是，结论并不属于 KB 的一部分，所以不能将其添加进 KB 中，只能采用临时变量，即运算符 + 返回的结果，作为参数传入 findModel 函数中。

在 positionLogicPlan 和 foodLogicPlan 函数中，并没有调用先前实现的 pacphysicsAxioms 函数，而是将其拆分开来。主要原因是因为 pacphysicsAxiom 函数相当于把时间的每一刻视为一个整体，但是在这两个函数中，我们在每一刻中执行的操作是有先后顺序的。由于转移公理描述的是给定时刻受上一时刻的约束，所以在循环体内，传入转移公理的时刻都是 $t+1$，而非 t，因为下一轮循环便进入了 $t+1$ 时刻。

以下为算法的具体实现，其中略去了已给的部分代码。

```python
def pacmanSuccessorAxiomSingle(x: int, y: int, time: int, walls_grid: List[List[bool]] = None) -> Expr:
    # 略去已给代码
    return PropSymbolExpr(pacman_str, x, y, time=time) % disjoin([possible_cause for possible_cause in possible_causes])

```

```python
def pacphysicsAxioms(t: int, all_coords: List[Tuple], non_outer_wall_coords: List[Tuple], walls_grid: List[List] = None, sensorModel: Callable = None, successorAxioms: Callable = None) -> Expr:
    pacphysics_sentences = []

    notInWall = [PropSymbolExpr(wall_str, x, y) >> ~PropSymbolExpr(pacman_str, x, y, time=t) for x, y in all_coords]
    inExactOne = exactlyOne([PropSymbolExpr(pacman_str, x, y, time=t) for x, y in non_outer_wall_coords])
    doExactOne = exactlyOne([PropSymbolExpr(direction, time=t) for direction in DIRECTIONS])
    pacphysics_sentences += notInWall
    pacphysics_sentences.append(inExactOne)
    pacphysics_sentences.append(doExactOne)
    if sensorModel:
        pacphysics_sentences.append(sensorModel(t, non_outer_wall_coords))
    if t and successorAxioms: # 初始时刻并不受上一时刻的约束
        pacphysics_sentences.append(successorAxioms(t, walls_grid, non_outer_wall_coords))

    return conjoin(pacphysics_sentences)

def checkLocationSatisfiability(x1_y1: Tuple[int, int], x0_y0: Tuple[int, int], action0, action1, problem):
    # 略去已给代码

    map_sent = [PropSymbolExpr(wall_str, x, y) for x, y in walls_list]
    KB.append(conjoin(map_sent))

    KB.append(pacphysicsAxioms(0, all_coords, non_outer_wall_coords, walls_grid, None, allLegalSuccessorAxioms))
    KB.append(pacphysicsAxioms(1, all_coords, non_outer_wall_coords, walls_grid, None, allLegalSuccessorAxioms))
    KB.append(PropSymbolExpr(pacman_str, x0, y0, time=0))
    KB.append(PropSymbolExpr(action0, time=0))
    KB.append(PropSymbolExpr(action1, time=1))

    inx1_y1 = findModel(conjoin(KB + [PropSymbolExpr(pacman_str, x1, y1, time=1)]))
    notInx1_y1 = findModel(conjoin(KB + [~PropSymbolExpr(pacman_str, x1, y1, time=1)]))
    return inx1_y1, notInx1_y1
```

```
36
37  def positionLogicPlan(problem) -> List:
38      # 略去已给代码
39      KB = []
40      KB.append(PropSymbolExpr(pacman_str, x0, y0, time=0))
41      for t in range(0, 50):
42          print(t)
43          KB.append(exactlyOne([PropSymbolExpr(pacman_str, x, y, time=t) for x
                , y in non_wall_coords]))
44          result = findModel(conjoin(KB + [PropSymbolExpr(pacman_str, xg, yg,
                time=t)]))
45          if result:
46              return extractActionSequence(result, actions)
47          KB.append(exactlyOne([PropSymbolExpr(action, time=t) for action in
                actions]))
48          KB += [pacmanSuccessorAxiomSingle(x, y, t + 1, walls_grid) for x, y
                in non_wall_coords]
49
50  def foodSuccessorAxiomSingle(x: int, y: int, time: int, walls_grid: List[
        List[bool]] = None) -> Expr:
51      now, last = time, time - 1
52      return conjoin([
53          ~PropSymbolExpr(food_str, x, y, time=last) >> ~PropSymbolExpr(
                food_str, x, y, time=now),  # 没有食物的位置不会生成食物
54          conjoin(
55              [PropSymbolExpr(food_str, x, y, time=last), PropSymbolExpr(
                    pacman_str, x, y, time=now)]) >> ~PropSymbolExpr(
56              food_str, x, y, time=now),
57          # 移动到有食物的位置后，食物便消失了
58          conjoin(
59              [PropSymbolExpr(food_str, x, y, time=last), ~PropSymbolExpr(
                    pacman_str, x, y, time=now)]) >> PropSymbolExpr(
60              food_str, x, y, time=now),
61          # 如果没有吃豆人移动到有食物的位置，那么该位置仍然有食物
62      ])
63
64  def foodLogicPlan(problem) -> List:
65      # 略去已给代码
66      KB = []
67      KB.append(PropSymbolExpr(pacman_str, x0, y0, time=0))
68      KB += [PropSymbolExpr(food_str, x, y, time=0) for x, y in food]
69      for t in range(0, 50):
```

```
70          print(t)
71          KB.append(exactlyOne([PropSymbolExpr(pacman_str, x, y, time=t) for x
                , y in non_wall_coords]))
72          result = findModel(conjoin(KB + [~PropSymbolExpr(food_str, x, y,
                time=t) for x, y in food]))
73          if result:
74              return extractActionSequence(result, actions)
75          KB.append(exactlyOne([PropSymbolExpr(action, time=t) for action in
                actions]))
76          KB += [pacmanSuccessorAxiomSingle(x, y, t + 1, walls) for x, y in
                non_wall_coords]
77          KB += [foodSuccessorAxiomSingle(x, y, t + 1, walls) for x, y in food
                ]
```

3.4. 实验结果

3.4.1. Question3

成功通过 Question3 的全部测试，实验结果截图见图 3.1。这里所使用的到的测试用例较为基础，检查了上述函数是否能够返回正确的结果。

3.4.2. Question4

成功通过 Question4 的全部测试，实验结果截图见图 3.2,3.3。所用到的测试样例便是将吃豆人放在迷宫场景中进行实战，三个测试样例中的迷宫大小递增。

3.4.3. Question5

成功通过 Question5 的全部测试，实验结果截图见图 3.4,3.5。所用到的测试样例便是将吃豆人放在迷宫场景中进行实战，两个测试样例中的迷宫大小递增，食物数量也递增。

```
Question q3
===========

*** Testing checkLocationSatisfiability
[LogicAgent] using problem type LocMapProblem
Ending game
Average Score: 0.0
Scores:        0.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** PASS: test_cases\q3\location_satisfiability1.test
*** Testing checkLocationSatisfiability
[LogicAgent] using problem type LocMapProblem
Ending game
Average Score: 0.0
Scores:        0.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** PASS: test_cases\q3\location_satisfiability2.test
*** Testing pacphysicsAxioms
*** PASS: test_cases\q3\pacphysics1.test
*** Testing pacphysicsAxioms
*** PASS: test_cases\q3\pacphysics2.test
*** PASS: test_cases\q3\pacphysics_transition.test
***     PASS

### Question q3: 4/4 ###
```

图 3.1: Question3 实验结果

```
Path found with total cost of 2 in 0.0 seconds
Nodes expanded: 0
Pacman emerges victorious! Score: 508
Average Score: 508.0
Scores:        508.0
Win Rate:      1/1 (1.00)
Record:        Win
*** PASS: test_cases\q4\positionLogicPlan1.test
***     pacman layout:          maze2x2
***     solution score:         508
***     solution path:          West South
```

```
Path found with total cost of 8 in 0.4 seconds
Nodes expanded: 0
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
*** PASS: test_cases\q4\positionLogicPlan2.test
***     pacman layout:          tinyMaze
***     solution score:         502
***     solution path:          South South West South West West South West
[LogicAgent] using problem type PositionPlanningProblem
```

图 3.2: Question4 实验结果

```
Path found with total cost of 19 in 43.7 seconds
Nodes expanded: 0
Pacman emerges victorious! Score: 491
Average Score: 491.0
Scores:        491.0
Win Rate:      1/1 (1.00)
Record:        Win
*** PASS: test_cases\q4\positionLogicPlan3.test
***     pacman layout:          smallMaze
***     solution score:         491
***     solution path:          East East South South West South South West West South West West West West West West West West

### Question q4: 4/4 ###
```

图 3.3: Question4 实验结果

```
Path found with total cost of 7 in 0.1 seconds
Nodes expanded: 0
Pacman emerges victorious! Score: 513
Average Score: 513.0
Scores:        513.0
Win Rate:      1/1 (1.00)
Record:        Win
*** PASS: test_cases\q5\foodLogicPlan1.test
***     pacman layout:          testSearch
***     solution score:         513
***     solution path:          West East East South South West West East
[LogicAgent] using problem type FoodPlanningProblem
```

图 3.4: Question5 实验结果

```
Path found with total cost of 27 in 8.6 seconds
Nodes expanded: 0
Pacman emerges victorious! Score: 573
Average Score: 573.0
Scores:        573.0
Win Rate:      1/1 (1.00)
Record:        Win
*** PASS: test_cases\q5\foodLogicPlan2.test
***     pacman layout:          tinySearch
***     solution score:         573
***     solution path:          South South West East East East East North North North North West West West West West West East East East South South West West West South South West

### Question q5: 3/3 ###
```

图 3.5: Question5 实验结果

4. 第三部分

4.1. 问题概述

该部分包含第 6、7、8 小题。

该部分要求我们利用逻辑来帮助智能体实现定位、地图绘制功能，并最终实现 SLAM 功能。

4.2. 算法设计

在定位任务中，智能体在初始阶段便会生成一个探索的方案，我们只需要按照该方案执行每一步动作，解析每次行动后得到的感知，将其加入知识库 KB 中，并尝试进行定位即可。“尝试定位”是指遍历所有非外侧围墙的位置，如果依据现有的知识库可以推断出智能体（不）在该位置，则将该结论加入知识库中；吃豆人在某一在满足知识库 KB 的模型中在特定位置，则该位置是吃豆人可能存在的位置。

在制图任务中，智能体在初始阶段也会生成一个探索的方案，我们只需要按照该方案执行每一步动作，解析每次行动后得到的感知，将其加入知识库 KB 中，并尝试进行制图即可。“尝试制图”是指遍历所有非外侧围墙的位置，如果依据现有的知识库可以推断出在该位置（不）是墙壁，则将该结论加入知识库中。如果一个位置既无法确定墙壁在该位置，也无法确定墙壁不在该位置，则该位置是否存在墙壁待定。

而 SLAM，即同时定位和制图，便只需要将定位和制图组合在一起即可，在初始状态，智能体知道它的初始位置，但是并不知道各个墙壁的位置。同时，智能体的感知器只能获取到智能体邻近区域的墙壁数量。

Algorithm 4.2.1: localization(agent)

```
KB = []
forall wall in walls do // 在定位任务中，智能体事先知道所有墙壁位置
    KB.append(wall.wallSentence())
end forall
for t in agent.num_timesteps do // 实现规划好的计划
    KB.append(gatherPacInfo(t))// 收集和吃豆人相关的信息，包含游戏规则、转移公理、感知器公理等
    possible_locations = updatePacLocation(t,KB)// 更新关于位置的信息
    agent.moveToNextState(agent.actions[t])// 执行计划的下一步
    yield possible_locations
end for
```

4.3. 算法实现

该部分的算法实现中依赖以下 4 个辅助函数：

1. gatherPacInfo：用于收集和吃豆人相关的各种信息，例如基本游戏规则对应的约束、解析智能体的感知得到的逻辑语句、智能体的行为

Algorithm 4.2.2: mapping(agent)

```
known_map.initialize()// 初始化已知地图（所有位置都待确定，除外边框）
KB = []
KB.append(pacmanInitialLocation)
for t in agent.num_timesteps do // 实现规划好的计划
    KB.append(gatherPacInfo(t))// 收集和吃豆人相关的信息，包含游戏规则、转移公理、感知器公理等
    updateWalllocation(KB,known_map)// 更新关于各位置墙壁的信息
    agent.moveToNextState(agent.actions[t])// 执行计划的下一步
    yield known_map
end for
```

Algorithm 4.2.3: slam(agent)

```
known_map.initialize()// 初始化已知地图（所有位置都待确定，除外边框）
KB = []
KB.append(pacmanInitialLocation)
for t in agent.num_timesteps do // 实现规划好的计划
    KB.append(gatherPacInfo(t))// 收集和吃豆人相关的信息，包含游戏规则、转移公理、感知器公理等
    updateWalllocation(KB,known_map)// 更新关于各位置墙壁的信息
    possible_locations = updatePacLocation(t,KB)// 更新关于位置的信息
    agent.moveToNextState(agent.actions[t])// 执行计划的下一步
    yield (known_map,possible_location)
end for
```

2. addIfEntail：判断已有知识库 KB 是否蕴含给定结论，如果是，则将该结论添加入知识库中，并放回 True，否则返回 False

3. updatePacLocation：遍历所有非外墙位置，如果已有知识库能判断吃豆人（不）在该位置，则将该信息加入知识库中。同时，通过寻找满足的模型来判断吃豆人是否有可能在该位置中

4. updateWallLocation：遍历所有非外墙位置，如果已有知识库能判断该位置（不）是墙壁，则将该信息加入知识库中，并在 known_map 中进行相应的标记。

```python
def gatherPacInfo(t: int, all_coords: List[Tuple], non_outer_wall_coords: List[Tuple], pacphysics_axioms: Callable,
    agent, walls_grid: List[List] = None,
    sensorModel: Callable = None, successorAxioms: Callable = None,
    perceptProcessing: Callable = None) -> Expr:
    new = []
    new.append(pacphysics_axioms(t, all_coords, non_outer_wall_coords, walls_grid, sensorModel, successorAxioms))
    new.append(PropSymbolExpr(agent.actions[t], time=t))
    percepts = agent.getPercepts()
    new.append(perceptProcessing(t, percepts))
    return conjoin(new)

def addIfEntail(KB: list, conclusion: Expr):
    if entails(conjoin(KB), conclusion):
        KB.append(conclusion)
        return True
    return False

def updatePacLocation(t: int, KB: List, non_outer_wall_coords: List[Tuple]):
    possible_locations = []
    for x, y in non_outer_wall_coords:
        addIfEntail(KB, PropSymbolExpr(pacman_str, x, y, time=t))
        addIfEntail(KB, ~PropSymbolExpr(pacman_str, x, y, time=t))
        if findModel(conjoin(KB + [PropSymbolExpr(pacman_str, x, y, time=t)])):
            possible_locations.append((x, y))
    return possible_locations

def updateWallLocation(KB: List, non_outer_wall_coords: List[Tuple], known_map:List[List]):
    for x, y in non_outer_wall_coords:
        if addIfEntail(KB, PropSymbolExpr(wall_str, x, y)):
            known_map[x][y] = 1
        if addIfEntail(KB, ~PropSymbolExpr(wall_str, x, y)):
```

```
32              known_map[x][y] = 0
```

下面是各算法的具体实现，其中略去了部分的已给代码。

```
def localization(problem, agent) -> Generator:
    # 略去已给代码
    KB = []
    KB += [PropSymbolExpr(wall_str, x, y) if (x, y) in walls_list else ~
       PropSymbolExpr(wall_str, x, y) for x, y in all_coords]
    for t in range(agent.num_timesteps):
        KB.append(gatherPacInfo(t, all_coords, non_outer_wall_coords,
           pacphysicsAxioms, agent, walls_grid, sensorAxioms,
           allLegalSuccessorAxioms, fourBitPerceptRules))
        possible_locations = updatePacLocation(t, KB, non_outer_wall_coords)
        agent.moveToNextState(agent.actions[t])
        yield possible_locations

def mapping(problem, agent) -> Generator:
    # 略去已给代码
    KB.append(PropSymbolExpr(pacman_str, pac_x_0, pac_y_0, time=0))
    for t in range(agent.num_timesteps):
        KB.append(gatherPacInfo(t, all_coords, non_outer_wall_coords,
           pacphysicsAxioms, agent, known_map, sensorAxioms,
           allLegalSuccessorAxioms, fourBitPerceptRules))
        updateWallLocation( KB, non_outer_wall_coords,known_map)
        agent.moveToNextState(agent.actions[t])
        yield known_map

def slam(problem, agent) -> Generator:
    # 略去已给代码
    KB.append(PropSymbolExpr(pacman_str, pac_x_0, pac_y_0, time=0))
    for t in range(agent.num_timesteps):
        KB.append(gatherPacInfo(t, all_coords, non_outer_wall_coords,
           pacphysicsAxioms, agent, known_map, SLAMSensorAxioms,
           SLAMSuccessorAxioms, numAdjWallsPerceptRules))
        updateWallLocation( KB, non_outer_wall_coords,known_map)
        possible_locations = updatePacLocation(t, KB, non_outer_wall_coords)
        agent.moveToNextState(agent.actions[t])
        yield (known_map, possible_locations)
```

4.4. 实验结果

成功通过 Question6,Question7 和 Question8 的全部测试，实验结果截图分别见图 4.1,4.2,4.3。这里的测试样例均是让吃豆人在规模较小的迷宫中执行对应的任务。

```
Question q6
===========

[LogicAgent] using problem type LocalizationProblem
rendering walls beforehand
*** PASS: test_cases\q6\localizationLogic1.test
[LogicAgent] using problem type LocalizationProblem
rendering walls beforehand
*** PASS: test_cases\q6\localizationLogic2.test

### Question q6: 4/4 ###
```

图 4.1: Question6 实验结果

```
Question q7
===========

[LogicAgent] using problem type MappingProblem
*** PASS: test_cases\q7\mappingLogic1.test
[LogicAgent] using problem type MappingProblem
*** PASS: test_cases\q7\mappingLogic2.test

### Question q7: 3/3 ###
```

图 4.2: Question7 实验结果

```
Question q8
===========

[LogicAgent] using problem type SLAMProblem
*** PASS: test_cases\q8\SLAMLogic1.test
[LogicAgent] using problem type SLAMProblem
*** PASS: test_cases\q8\SLAMLogic2.test

### Question q8: 4/4 ###
```

图 4.3: Question8 实验结果

5. 总结与分析

在实验的过程中，问题主要出现在后半部分的智能体的规划算法的实现上。例如在实现 foodLogicPlan 时，在测试过程中，算法返回的方案过短，经过排查后发现是因为 foodSuccessorAxiom 中，并没有考虑全所有的三种情况：1）之前无食物；2）之前有食物，吃豆人当前在该位置；3）之前有食物，吃豆人当前不在该位置。这导致约束过弱，导致 SAT 在求满足的模型时可以不受约束地对食物相关的语句进行赋值，例如没有考虑第三种情况，那么即使吃豆人当前不在该位置，SAT 返回的模型中该位置上的食物也可以被设置为已收集，从而导致吃豆人过早地认为收集食物的任务已经完成了。

在测试的过程中，我还直观地感受到了基于逻辑的规划算法的缺点——耗时过长。基于逻辑的规划算法虽然强大，能够帮助智能体进行路径规划、制图、SLAM 等功能，且相应的源代码都十分模块化，但是随着时间的推移，知识库 KB 中包含的语句数量快速增长，导致耗时较长。即使是在一个较小的迷宫场景下，想要完成一个任务都需要耗时将近半分钟，这在充满不确定性以及多变性的现实场景中往往是不可接受的。