2024 AI - HW#3 Knowledge

生醫電資所 碩一 r12945040 郭思言

Show your autograder result for each question:

- **Overall**

```
Finished at 14:37:51

Provisional grades
==================
Question q1: 10/10
Question q2: 10/10
Question q3: 10/10
Question q4: 10/10
Question q5: 10/10
------------------
Total: 50/50

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

- **Q1**

```
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/entailsLong.test
***     PASS
*** PASS: test_cases/q1/findModelSentence1.test
***     PASS
*** PASS: test_cases/q1/findModelSentence2.test
***     PASS
*** PASS: test_cases/q1/findModelSentence3.test
***     PASS
a.__dict__ is: {'op': 'A', 'args': ()}
*** PASS: test_cases/q1/findModelUnderstandingCheck.test
***     PASS
*** PASS: test_cases/q1/plTrueInverse.test
***     PASS

### Question q1: 10/10 ###
```

- **Q2**

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

*** PASS: test_cases/q2/atLeastOne.test
***     PASS
*** PASS: test_cases/q2/atLeastOneCNF.test
***     PASS
*** PASS: test_cases/q2/atLeastOneEff.test
***     PASS
*** PASS: test_cases/q2/atMostOne.test
***     PASS
*** PASS: test_cases/q2/atMostOneCNF.test
***     PASS
*** PASS: test_cases/q2/atMostOneEff.test
***     PASS
*** PASS: test_cases/q2/exactlyOne.test
***     PASS
*** PASS: test_cases/q2/exactlyOneCNF.test
***     PASS
*** PASS: test_cases/q2/exactlyOneEff.test
***     PASS

### Question q2: 10/10 ###
```

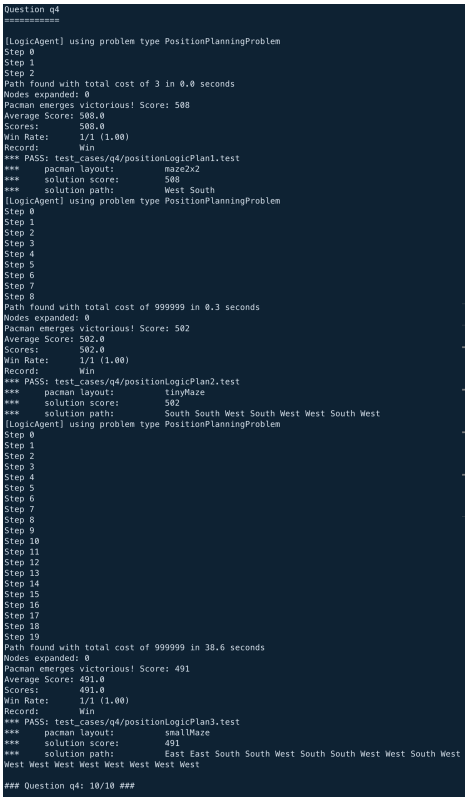
- **Q3**

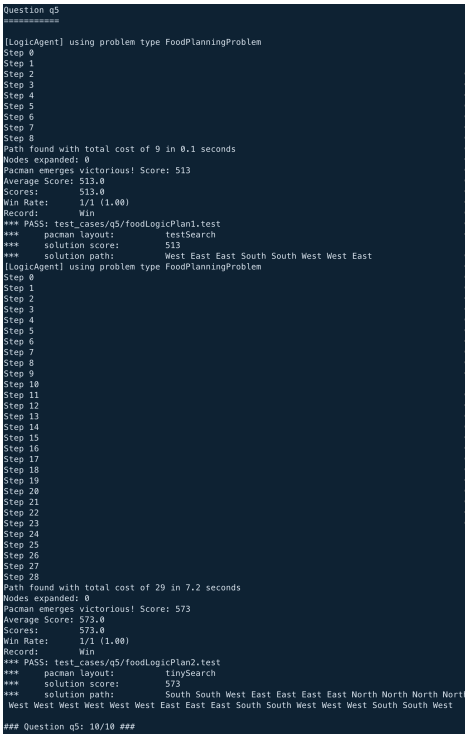
```
Question q3
===========

*** 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: 10/10 ###
```

- **Q4**
- **Q5**





Describe your algorithm for each question in the report.

- **Q1 – Logic Warm-up**
  1. **sentence1():**
     - This function creates an `Expr` instance that represents the proposition that the given three sentences are true.
     - Uses the `disjoin` function to represent logical OR (`|`) and the `%` operator to represent logical equivalence (`<=>`).
     - The three sentences are:
       1. A OR B
       2. NOT A IF AND ONLY IF (NOT B OR C)
       3. NOT A OR NOT B OR C
  2. **sentence2():**
     - This function creates an `Expr` instance that represents the proposition that the given four sentences are true.
     - Uses the `disjoin` function to represent logical OR (`|`) and the `>>` operator to represent logical implication (`=>`).
     - The four sentences are:
       1. C IF AND ONLY IF (B OR D)
       2. A IMPLIES ((NOT B) AND (NOT D))
       3. (NOT (B AND (NOT C))) IMPLIES A

4. (NOT D) IMPLIES C

3. **sentence3():**
    - Creates an `Expr` instance that encodes the given three English sentences as propositional logic without any simplification.
    - The three sentences are:
        1. Pacman is alive at time 1 if and only if Pacman was alive at time 0 and it was not killed at time 0 or it was not alive at time 0 and it was born at time 0.
        2. Pacman cannot both be alive at time 0 and be born at time 0.
        3. Pacman is born at time 0.
4. **findModelUnderstandingCheck():**
    - Creates a lower-cased `Expr` instance and attempts to mimic the output of `findModel(Expr('a'))` if lower-cased expressions were allowed.
5. **entails(premise: Expr, conclusion: Expr):**
    - Checks if the conjunction of the premise and the negation of the conclusion is unsatisfiable using `findModel`.
    - If the conjunction is unsatisfiable, it means the premise entails the conclusion, and the function returns True. Otherwise, it returns False.
6. **plTrueInverse(assignments: Dict[Expr, bool], inverse_statement: Expr):**
    - Uses the `pl_true` function from `logic.py` to evaluate the negation of the `inverse_statement` with the given assignments.

- **Q2 – Logic Workout**
    1. **atLeastOne(literals: List[Expr]) → Expr:**
        - Uses the `disjoin` function to create a disjunction (logical OR) of all the literals in the input list, ensuring that at least one of them is true.
    2. **atMostOne(literals: List[Expr]) → Expr:**
        - Uses `itertools.combinations` to generate all pairs of literals in the input list and creates a disjunction of negated pairs, ensuring that at most one of the literals in each pair is true.
    3. **exactlyOne(literals: List[Expr]) → Expr:**
        - Uses the `atLeastOne` and `atMostOne` functions to create an `Expr` instance that represents the conjunction (logical AND) of at least one and at most one, ensuring that exactly one of the literals is true.

- **Q3 – Pacphysics and Satisfiability**
    1. **pacmanSuccessorAxiomSingle(x: int, y: int, time: int, walls_grid: List[List[bool]] = None) → Expr:**
        - This function generates an expression defining the sufficient and necessary conditions for Pacman to be at (x,y) at time t.

- It considers all possible causes for Pacman to be at (x,y) at time t based on the previous position at time t-1 and the action taken to move to (x,y).
- The return statement constructs an `Expr` instance representing the logical relationship between the possible causes.

2. **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:**
   - This function generates a set of physics axioms for timestep t.
   - It constructs logical expressions for the following conditions:
     - If a wall is at (x, y), then Pacman is not at (x, y) at timestep t.
     - Pacman is at exactly one of the squares at timestep t.
     - Pacman takes exactly one action at timestep t.
     - Results of calling the `sensorModel` function, if provided.
     - Results of calling the `successorAxioms` function, if provided.
   - The expressions are added to `pacphysics_sentences` and then conjoined to form the final `Expr` instance.
3. **checkLocationSatisfiability(x1_y1: Tuple[int, int], x0_y0: Tuple[int, int], action0, action1, problem) → Tuple[Dict[Expr, bool], Dict[Expr, bool]]:**
   - This function checks the satisfiability of Pacman's location at (x1, y1) at time t = 1 given its location at (x0, y0) at time t = 0 and actions taken.
   - It creates a knowledge base (KB) containing:
     - Known walls on the grid.
     - Physics axioms for t = 0 and t = 1.
     - Pacman's location at time t = 0.
     - Pacman's actions at t = 0 and t = 1.
   - It then queries the SAT solver with two models: one where Pacman is at (x1, y1) at time t = 1 and another where Pacman is not at (x1, y1) at time t = 1.
   - The function returns these two models as a tuple.

- **Q4 – Path Planning with Logic**

1. **positionLogicPlan(problem) → List:**
   - This function takes an instance of a `PositionPlanningProblem` and returns a list of actions that lead Pacman to the goal.
   - It incrementally adds knowledge to the knowledge base (KB) and queries for a model at each timestep to determine Pacman's actions.
   - The KB is initialized with the initial position of Pacman.
   - For each timestep (up to a maximum of 50):

    - It adds constraints to the KB to ensure that Pacman is at exactly one of the possible locations and takes exactly one action.
    - It adds transition model sentences using `pacmanSuccessorAxiomSingle` for all possible Pacman positions.
    - It queries the SAT solver with a goal assertion asserting that Pacman is at the goal at the current timestep.
    - If a model is found, it extracts the action sequence from the model and returns it as the plan.
  - The function iterates through timesteps until a plan is found or until the maximum number of timesteps is reached.

  This algorithm uses propositional logic to plan Pacman's sequence of actions leading to the goal, incrementally building up knowledge about Pacman's possible positions and actions at each timestep.

- **Q5 – Eating All the Food**

  1. **foodLogicPlan(problem) → List:**
     - This function takes an instance of a `FoodPlanningProblem` and returns a list of actions that lead Pacman to eat all of the food on the board.
     - It initializes the KB with Pacman's initial position and the initial positions of the food.
     - For each timestep (up to a maximum of 50):
       - It adds constraints to the KB to ensure that Pacman is at exactly one of the possible locations and takes exactly one action.
       - It adds transition model sentences using `pacmanSuccessorAxiomSingle` for all possible Pacman positions.
       - It adds food successor axioms using `foodSuccessorAxiomSingle` to model the relation between food at timestep t and t+1.
       - It queries the SAT solver with a goal assertion asserting that all food have been eaten.
       - If a model is found, it extracts the action sequence from the model and returns it as the plan.
     - The function iterates through timesteps until a plan is found or until the maximum number of timesteps is reached.

     This algorithm uses propositional logic to plan Pacman's sequence of actions leading to the goal of eating all of the food on the board. It incrementally builds up knowledge about Pacman's possible positions, actions, and the state of the food at each timestep.