

# Principles of Programming Languages

## Spring 2024 - Project 1

### 1 Introduction

In this project, you will develop a simulation for a farm environment where animals can move around, consume food, and reproduce. The game takes place in a simplified, farm-themed world where reaching sufficient energy points from foods allows animals to reproduce.

### 2 Knowledge Base

We have built the basic mechanics for you in `cmpefarm.pro` and `farm.pro`. These files should remain unchanged. The farm map is provided in `map.txt`. You can change it to test your farm environment in different settings.

The main data structure for this project is given in `state/4` predicate. After you load your main file (`main.pro`), you can check the current state with the following query:

?- `state`(Agents, Objects, Time, TurnOrder).

`Agents` = agent \_dict{0:agents{children:0, energy point:0, subtype:chicken, type:herbivore, x:4, y:4}, ...},

`Objects` = object dict{0:object{subtype:corn, type:food, x:2, y:1}, ...},

`TimeOrder` = 1,

`Turn` = [0,1]

```
. M . . . . . F .  
    . . . . . F . . .  
    . . . H G . . . C .  
    . . . M . . F . . .  
. F . . M . . . . W
```

This state is generated from the map file `farm.txt`. Each character in the `farm.txt` represents different types of agents and objects:

- Agents
  - C: Cow.
  - H: Chicken.
  - W: Wolf.
- Objects
  - G: Grass.
  - F: Grain.
  - M: Corn.

Each agent is characterized by six distinct attributes: type, subtype, x, y, energy point, children. Agents are able to move through the map and can eat food to increase their energy points. If the agent's energy point exceeds a certain point, the agent reproduces and the number of agents increases.

- type: Agent's type (herbivore/carnivore).
- subtype: Species of the agent.

- x: X coordinate of the agent.
- y: Y coordinate of the agent.
- energy \_point: Agent's energy for reproduction.
- children: Number of children the agent has.

Each object is represented by four attributes, subtype, type, x, y, and they are held in an object dictionary.

- type: Object's type.
- subtype: Object's kind.
- x: X coordinate of the object.
- y: Y coordinate of the object.

The agents can move in a certain direction, eat and reproduce. Cows have the ability to move in four directions: up, down, left and right, allowing them to navigate the game map within these specific constraints. Chickens in the game are restricted to moving diagonally; they can move up-right, up-left, down-right, and down-left. Lastly, wolves can move in every direction.

Moreover, cows in the game are able to eat grass and grain, but they cannot consume corn. Chickens can eat grain and corn, but they are not able to eat grass. If the agent reaches the location of the food, it does not automatically consume it, instead wait for the eat command.

### 3 Predicates

#### 3.1 agents distance(+Agent1, +Agent2, -Distance) 10 points

This predicate will compute the Manhattan distance(D) between two agents.

$$D([x_1, y_1], [x_2, y_2]) = |x_1 - x_2| + |y_1 - y_2| \quad (1)$$

**Examples:**

?- state(Agents, , , ), agents distance(Agents.0, Agents.1, Distance)

Distance = 5.

?- state(Agents, , , ), agents distance(Agents.1, Agents.2, Distance) Distance = 4.

All test cases will contain a valid Agent1 and Agent2

#### 3.2 number of agents(+State, -NumberOfAgents) 10 points

This predicate finds the total number of agents in a State and unifies it with NumberOfAgents.

State is a list in the form of [Agents, Objects, Time, TurnOrder].

**Examples:**

?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], number of agents(State, NumberOfAgents).

NumberOfAgents = 3

?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], number of agents(State, NumberOfAgents).

NumberOfAgents = 6

### 3.3 value of \_farm(+State, -Value) 10 points

This predicate calculates the total value of all products on the farm. You need to consider the values of agents (animals) and objects (foods). You can find the values of products in farm.pro.

#### Examples:

?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], value\_of\_farm(State, Value).

Value = 740

?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], value\_of\_farm(State, Value). Value = 520

### 3.4 find food coordinates(+State, +AgentId -Coordinates) 10 points

This predicate will find the coordinates of the foods consumable by the specific Agent at the given State and unifies the list of coordinates with Coordinates. If there is no such object that can be eaten by the agent, the predicate should be false.

#### Examples:

?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], find\_food\_coordinates(State, 0, Coordinates).

Coordinates = [[2, 1], [9, 1], [6, 2], [4, 4], [7, 4], [2, 5], [5, 5]].

?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], find\_food\_coordinates(State, 1, Coordinates).

Coordinates = [[9, 1], [6, 2], [5, 3], [7, 4], [2, 5]]

?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], find\_food\_coordinates(State, 4, Coordinates). false.

### 3.5 find nearest agent(+State, +AgentId, -Coordinates, -NearestAgent) 10 points

This predicate finds the nearest agent and unifies the agent's coordinate with Coordinates in the [X,Y] form, and unifies the agent's dictionary with NearestAgent.

You cannot use the -> operator for this or any associated helper predicates.

#### Examples:

?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], find\_nearest\_agent(State, AgentId, Coordinates, NearestAgent).

Coordinates = [4,3]

NearestAgent = agents{children:0, energy point:0, subtype:chicken, type:herbivore, x:4, y:3} ?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], find\_nearest\_agent(State, AgentId, Coordinates, NearestAgent).

Coordinates = [4,3]

NearestAgent = agents{children:0, energy point:0, subtype:chicken, type:herbivore, x:4, y:3}

### 3.6 find nearest food(+State, +AgentId, -Coordinates, -FoodType, -Distance) 10 points

This predicate finds the nearest **consumable** food by the Agent and unifies the object's coordinate with Coordinates in the [X,Y] form, unifies the kind of the food with FoodType, and unifies the Manhattan distance between the food and the agent with Distance. If there is no such object that can be eaten by the agent, the predicate should be false.

**Examples:**

?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], **find nearest food**(State, AgentId, Coordinates, FoodType, Distance).

**Coordinates** = [2,5]

**FoodType** = grain

**Distance** = 5

**3.7 move to coordinate(+State, +AgentId, +X, +Y, -ActionList, +DepthLimit) 10 points**

This predicate will find a series of actions that will get the Agent to a specific [X,Y] coordinate and unify this list of actions with ActionList. The number of actions is limited with the DepthLimit.

**Examples:**

?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], **move to \_coordinate**(State, 0, 1, 5, ActionList, 4).

**ActionList** = [move down, move down, move down, move \_down]

?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], **move to \_coordinate**(State, 0, 2, 5, ActionList, 3). **false**.

**3.8 move to nearest food(+State, +AgentId, -ActionList, +DepthLimit) 10 points**

This predicate will find a series of actions that will get the Agent to the closest food that can be consumed by the Agent and unify this list of actions with ActionList. The number of actions is limited with the DepthLimit.

**Examples:**

?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], **move to \_nearest food**(State, 0, ActionList, 5).

**ActionList** = [move down, move right, move down, move down, move down]

?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], **move to \_nearest food**(State, 0, ActionList, 2). **false**.

**3.9 consume \_all(+State, +AgentId, -NumberOfMovements, -Value, -NumberOfChildren, +DepthLimit) 20 points**

This predicate will find a sequence of actions that will guide the Agent to every consumable food item, beginning with the closest one.

- After reaching the first item, the agent should **consume** the item.
- The agent should find the next nearest consumable item and move towards it.
- If the nearest item cannot be reached by the agent, the agent should move towards to the next closest item.
- The number of actions is limited by the DepthLimit; however, you are expected to reach the item using the smallest number of movements possible.
- There will be no consumable items at the same distance from the agent.
- You should first check whether the specific agent can consume the item and can move to the item's location.

Unify the total number of movements with NumberOfMovements and farm's new value with Value and the total number of children of the agent with NumberOfChildren.

**Examples:**

```
?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], consume all(State, 1, NumberOfMovements, Value, NumberOfChildren).
```

NumberOfMovements = 17

Value = 1630

NumberOfChildren = 3

```
?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], consume all(State, 0, NumberOfMovements, Value, NumberOfChildren).
```

NumberOfMovements = 11

Value = 800

NumberOfChildren = 2

### 3.10 Printing states

You can use make one action/4 with print state/1 to view the new state after the action.

Moreover, you can use make one \_action \_print/4 and make series of actions print/3 to make some actions and view the final state.

For example,

```
?- state(Agents, Objects, Time, TurnOrder), State=[Agents, Objects, Time, TurnOrder], make one action(Action, State, AgentId, NewState), print state(NewState).
```

This will help you see the actions of the agent, and hopefully allow you to debug more easily.