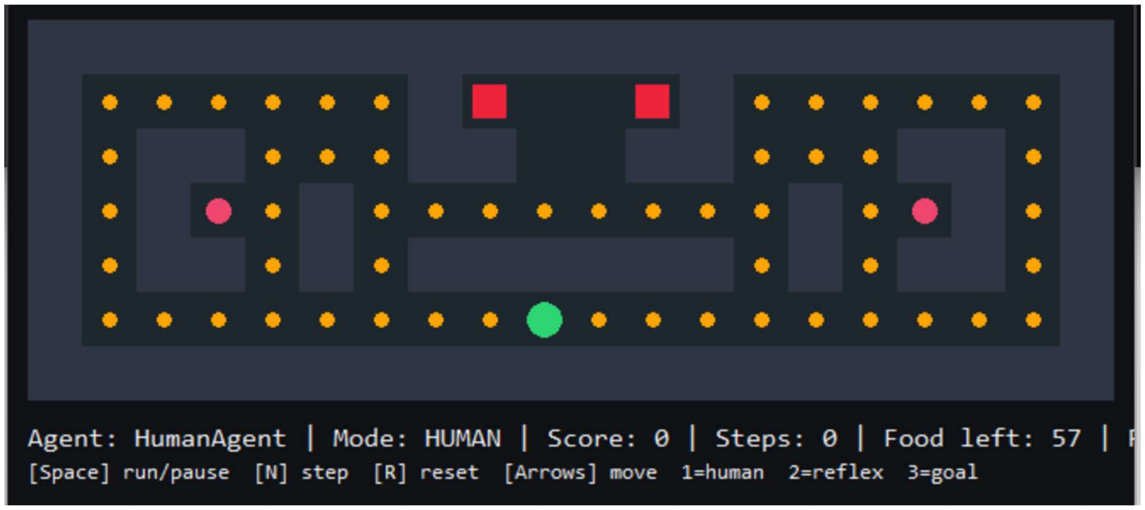
Phan Ngọc Hạnh Nhi

2131209002

CSW 330-Lab1

Assignment 1

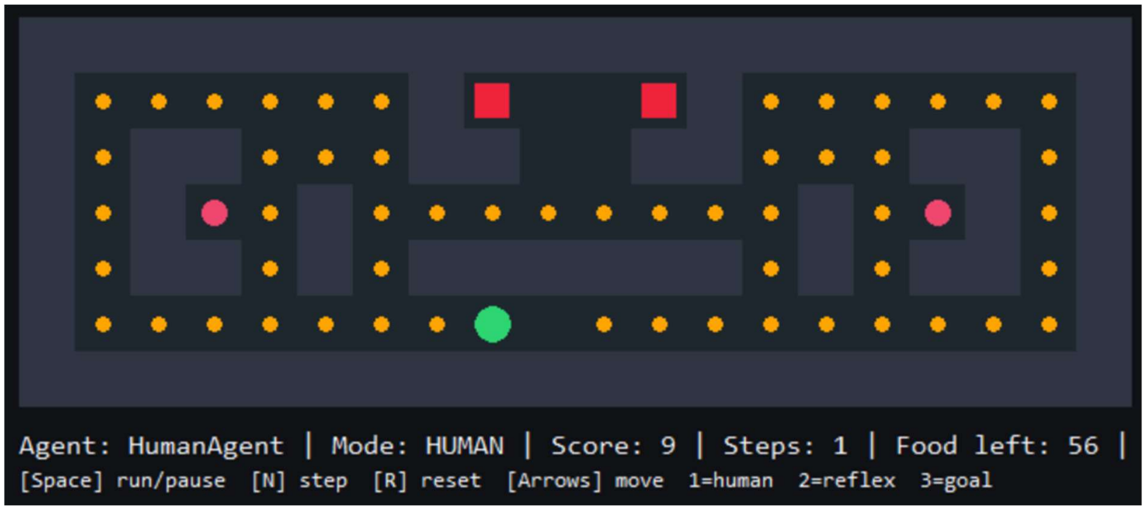
1. Run human mode



```
Agent: HumanAgent | Mode: HUMAN | Score: 0 | Steps: 0 | Food left: 57 | F
[Space] run/pause  [N] step  [R] reset  [Arrows] move  1=human  2=reflex  3=goal
```

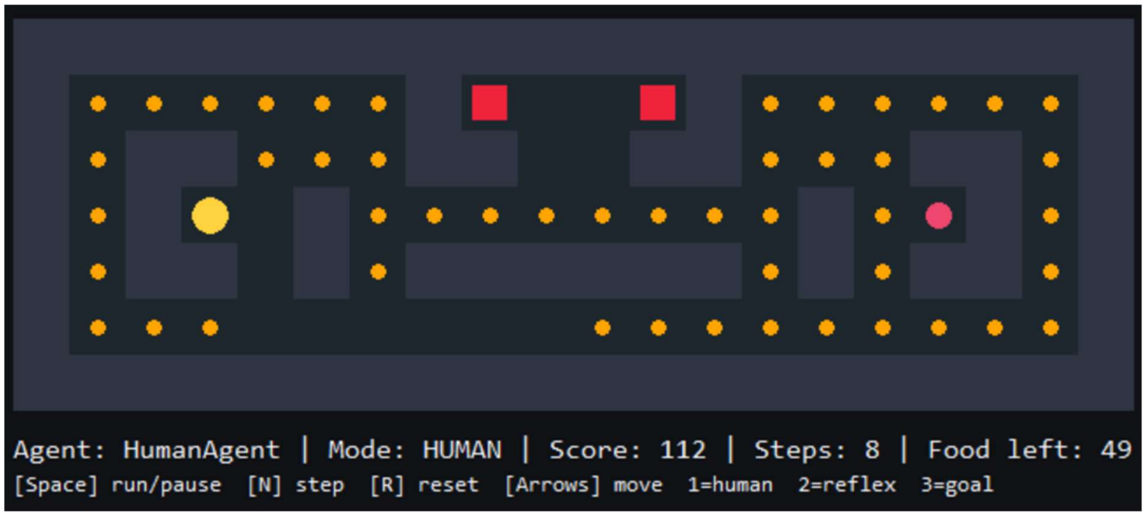
2. Observe the rules

- Food . : +10 points.



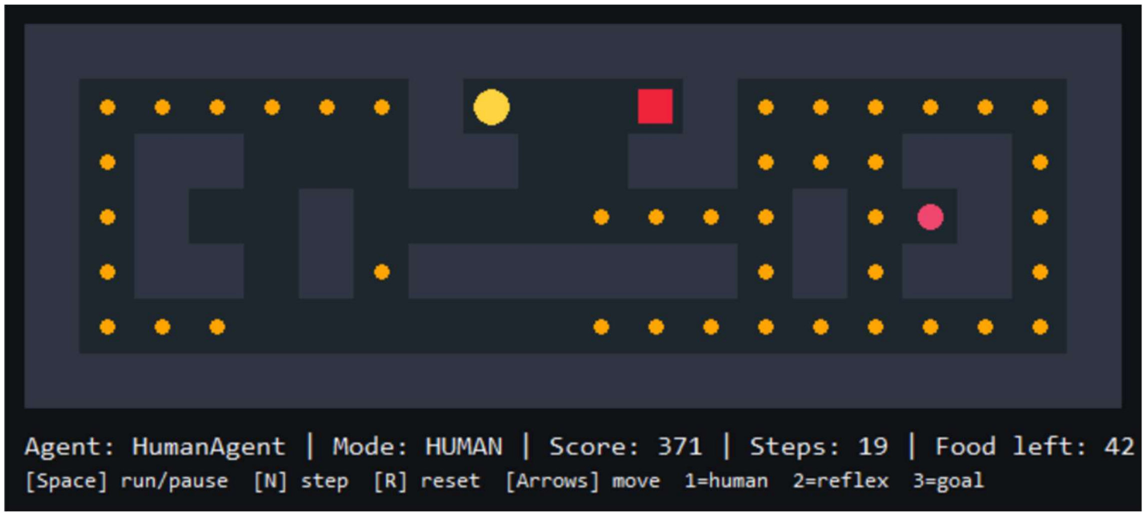
```
Agent: HumanAgent | Mode: HUMAN | Score: 9 | Steps: 1 | Food left: 56 |
[Space] run/pause  [N] step  [R] reset  [Arrows] move  1=human  2=reflex  3=goal
```

- Power pellet o : +50 points and power mode for 15 steps.



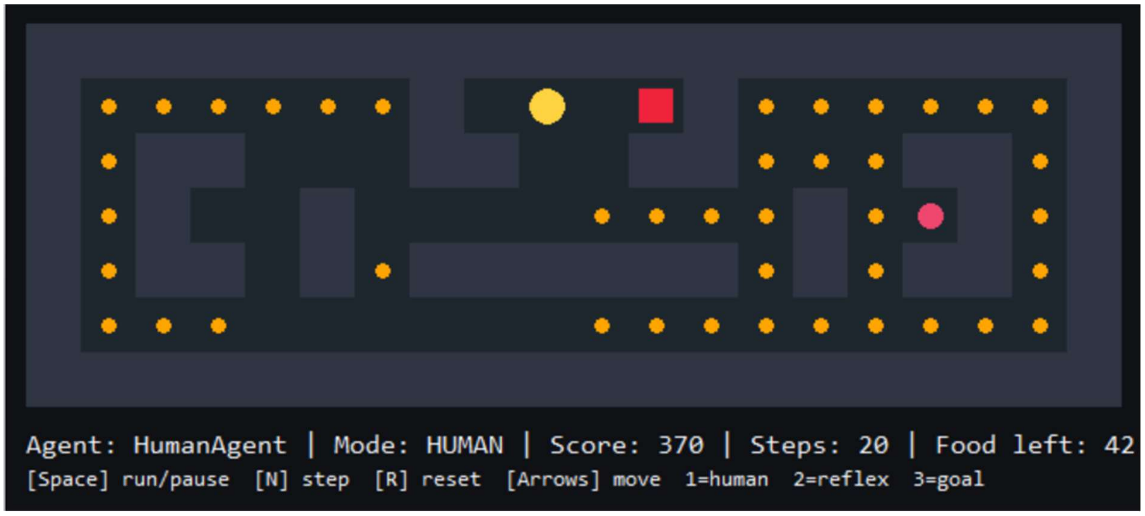
```
Agent: HumanAgent | Mode: HUMAN | Score: 112 | Steps: 8 | Food left: 49
[Space] run/pause  [N] step  [R] reset  [Arrows] move  1=human  2=reflex  3=goal
```

- Ghost G collision: +200 if in power mode (ghost removed), −500 otherwise.



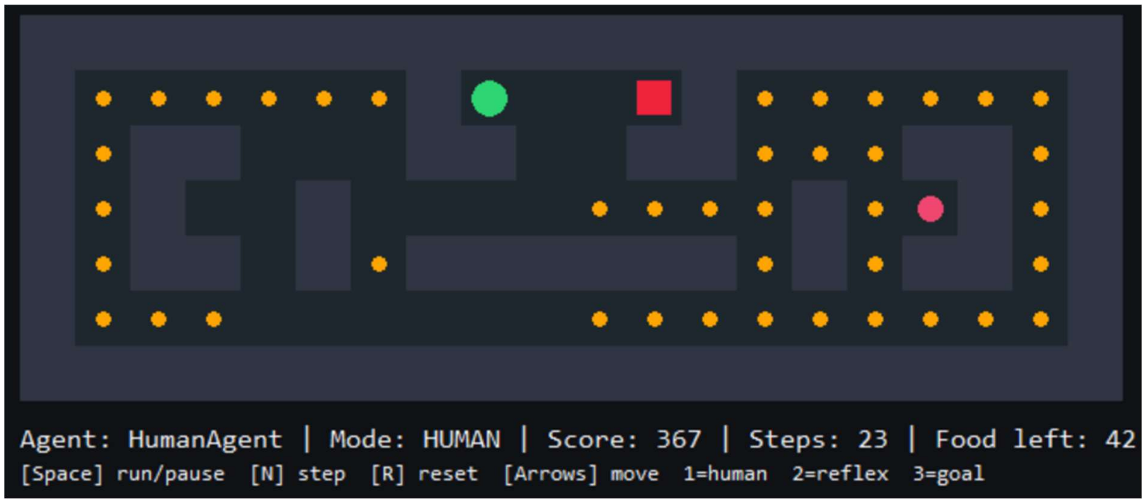
```
Agent: HumanAgent | Mode: HUMAN | Score: 371 | Steps: 19 | Food left: 42
[Space] run/pause  [N] step  [R] reset  [Arrows] move  1=human  2=reflex  3=goal
```

- Every step: −1 point (step penalty).



```
Agent: HumanAgent | Mode: HUMAN | Score: 370 | Steps: 20 | Food left: 42
[Space] run/pause  [N] step  [R] reset  [Arrows] move  1=human  2=reflex  3=goal
```

- Bumping into a wall % : you remain in place (optional bump cost = 0 by default).



```
Agent: HumanAgent | Mode: HUMAN | Score: 367 | Steps: 23 | Food left: 42
[Space] run/pause  [N] step  [R] reset  [Arrows] move  1=human  2=reflex  3=goal
```

3. **Answer these questions in your report:**

   a) What triggers power mode, and when does it end?

   - Trigger: Power Mode is activated when Pac-Man eats a power pellet.

     The score increases by +50 points, Pac-Man changes color, and the status line shows "Power: ON."

   - End: Power Mode ends automatically after 15 moves or when the internal timer expires. Once it ends, Pac-Man returns to normal color and the status changes to "Power: off."

   b) How does total score evolve over time during exploration?

   - The score increases when Pac-Man eats food . (+10) and power pellets o (+50).
   - During Power Mode, each ghost eaten adds +200 points.
   - Every move incurs a −1 step penalty, so the score fluctuates slightly - rising with food, dropping slowly between meals.
   - Colliding with a ghost outside Power Mode causes -500 points, a sharp drop.

     Overall: the score tends to rise steadily during exploration, spike when eating ghosts, and decline gradually due to movement penalties.

   c) What ends an episode (all pellets vs. step limit)?

   - The episode ends when all food and power pellets are eaten (when food_remaining = 0). Initially, I thought there was no limit on the number of steps, since Pac-Man could continue moving freely. However, after checking the source code in main.py, I noticed that there is a default step limit of 200 (ap.add_argument("--max-steps", type=int, default=200)).

     Therefore, the game can also end when the maximum step count (200 steps by default) is reached.

   d) One limitation you experienced with manual control.

- One limitation of manual control is that I had to learn and remember the keys for each action (for example, *Space* to run/pause and arrow keys to move). The key mapping is simple, but at first I was not familiar with it, so I felt a bit confused and could not move properly several times. After some practice, I got used to the control scheme, but it still required attention and slowed down my reactions during the game.
- Manual control also has other limitations: it is easy to hit walls or get stuck due to slow reactions, players cannot plan ahead to avoid ghosts-resulting in frequent collisions and score loss-and humans struggle to efficiently collect all food before reaching the step limit.
- Summary: Manual control is intuitive but inefficient, as it lacks the strategic planning and speed of automated agents.

Assignment 2

1. Open engine.py and locate where the environment handles:

- Eating food . and power pellet o:

```
    # Consume food / power if present and not yet eaten
        if self.grid[nr][nc] == '.' and (nr, nc) not in self.eaten:
            self.eaten.add((nr, nc))
            if (nr, nc) in self.food:
                self.food.remove((nr, nc))
            reward += config.FOOD_REWARD

        elif self.grid[nr][nc] == 'o' and (nr, nc) not in
self.eaten:
            self.eaten.add((nr, nc))
            if (nr, nc) in self.power:
                self.power.remove((nr, nc))
            reward += config.POWER_REWARD
            self.power_mode = True
            self._power_left = config.POWER_STEPS
```

- Entering and leaving power mode:

```
  self.power_mode = True
            self._power_left = config.POWER_STEPS
```

```
  # Power mode countdown
        if self.power_mode:
            self._power_left -= 1
            if self._power_left <= 0:
                self.power_mode = False
```

- Colliding with ghosts G

```
  # Ghost collision (static ghosts for Lab 1)
        if (nr, nc) in self.ghosts:
            if self.power_mode:
                # Eat the ghost: reward and remove it
```

```
                    reward += config.GHOST_REWARD
                    self.ghosts.remove((nr, nc))
                else:
                    reward += config.GHOST_PENALTY
```

- Updating score , steps , and eaten

```
    # Bookkeeping
            self.steps += 1
            self.score += reward
```

2. Open ui.py and confirm:

- How walls % , empty spaces, food . , power o , ghosts G , and Pac-Man are drawn

```
    # Tiles & pellets
            for r in range(self.env.H):
                for c in range(self.env.W):
                    x0 = pad + c * cell
                    y0 = pad + r * cell
                    x1 = x0 + cell
                    y1 = y0 + cell
                    ch = self.env.grid[r][c]

                    # base tile
                    fill = config.COLOR_WALL if ch == "%" else
config.COLOR_EMPTY
                    self.canvas.create_rectangle(x0, y0, x1, y1,
fill=fill, width=0)

                    # food '.'
                    if ch == "." and (r, c) not in self.env.eaten:
                        cx, cy = (x0 + x1) // 2, (y0 + y1) // 2
                        self.canvas.create_oval(
                            cx - 4, cy - 4, cx + 4, cy + 4,
fill=config.COLOR_FOOD, width=0)

                    # power 'o'
                    if ch == "o" and (r, c) not in self.env.eaten:
                        cx, cy = (x0 + x1) // 2, (y0 + y1) // 2
                        self.canvas.create_oval(
                            cx - 7, cy - 7, cx + 7, cy + 7,
fill=config.COLOR_POWER, width=0)

            # ghosts 'G' (static in Lab 1)
            for (gr, gc) in self.env.ghosts:
                x0 = pad + gc * cell + 6
                y0 = pad + gr * cell + 6
                x1 = x0 + cell - 12
                y1 = y0 + cell - 12
                self.canvas.create_rectangle(x0, y0, x1, y1,
fill=config.COLOR_GHOST, width=0)

            # pac-man (color indicates power mode)
```

```
        pr, pc = self.env.P
        x0 = pad + pc * cell + 6
        y0 = pad + pr * cell + 6
        x1 = x0 + cell - 12
        y1 = y0 + cell - 12
        pac_color = config.COLOR_PAC_PWR if self.env.power_mode
else config.COLOR_PAC
        self.canvas.create_oval(x0, y0, x1, y1, fill=pac_color,
width=0)
```

- Why Pac-Man changes color in power mode

```
pac_color = config.COLOR_PAC_PWR if self.env.power_mode else
config.COLOR_PAC
self.canvas.create_oval(x0, y0, x1, y1, fill=pac_color, width=0)
```

- What the information panel reports each frame (agent, mode, score, steps, food left, power state)

```
 text1 = (
            f"Agent: {type(self.agent).__name__} | Mode:
{self.mode.upper()} | "
            f"Score: {self.env.score} | Steps: {self.env.steps} |
Food left: {self.env.food_remaining()} "
            f"| Power: {'ON' if self.env.power_mode else 'off'} |
Ghosts: {len(self.env.ghosts)}"
        )
```

3. Write a short explanation:

- How the environment determines termination

```
def done(self, max_steps: int) -> bool:
        return self.food_remaining() == 0 or self.steps >=
max_steps
```

  - The environment ends when:
  - All food and power pellets are eaten (`self.food_remaining() == 0`), or
  - The total number of steps exceeds the maximum limit (`self.steps >= max_steps`, default = 200).

- Why the world is designed to be deterministic

  The world is deterministic — the same action always results in the same outcome. This approach has several advantages:

  1. Easy to understand: predictable outcomes help visualize and reason about behavior.
  2. Easy to debug: repeating the same input yields the same output, simplifying testing.
  3. Ideal for reinforcement learning: the stable environment helps agents learn consistent state–reward relationships.

- What env.neighbors(r,c) returns and how it prevents walking through walls env.neighbors(r, c) returns a dictionary of valid moves from position (r, c):

```
{
    'up': (6, 3),
    'down': (7, 3),
    'left': (7, 2),
    'right': (7, 4)
}
```

The logic:

out[a] = (nr, nc) if self.passable(nr, nc) else (r, c)

This ensures:

- If the next position (nr, nc) is passable, Pac-Man moves there.
- If not (it's a wall % or out of bounds), Pac-Man stays in place (r, c).

The helper function:

```
def passable(self, r: int, c: int) -> bool:
    return self.in_bounds(r, c) and self.grid[r][c] != '%'
```

It checks whether the position is inside the map and not a wall,

so Pac-Man cannot move through walls or outside the grid.

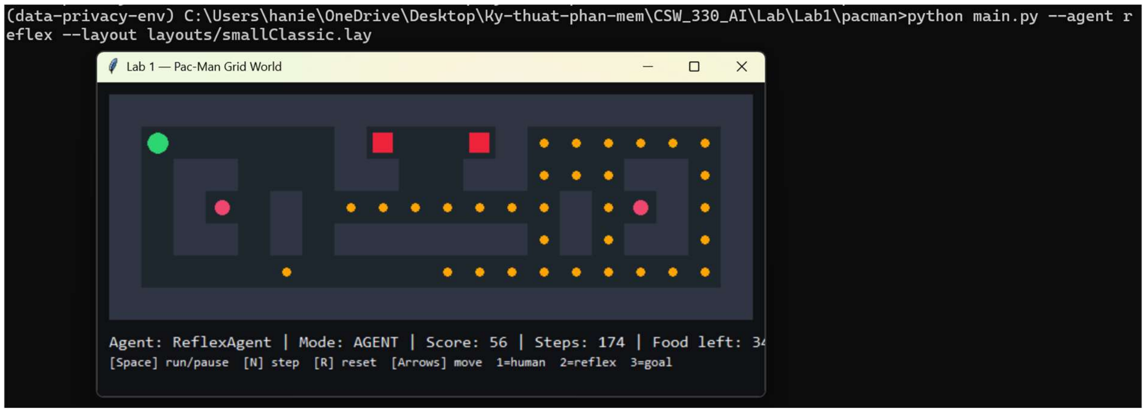
Assignment 3

Compare simple reflex agent and goal-based agent through experiments.

Step 1: Execute both agents

1. Run each configuration separately:

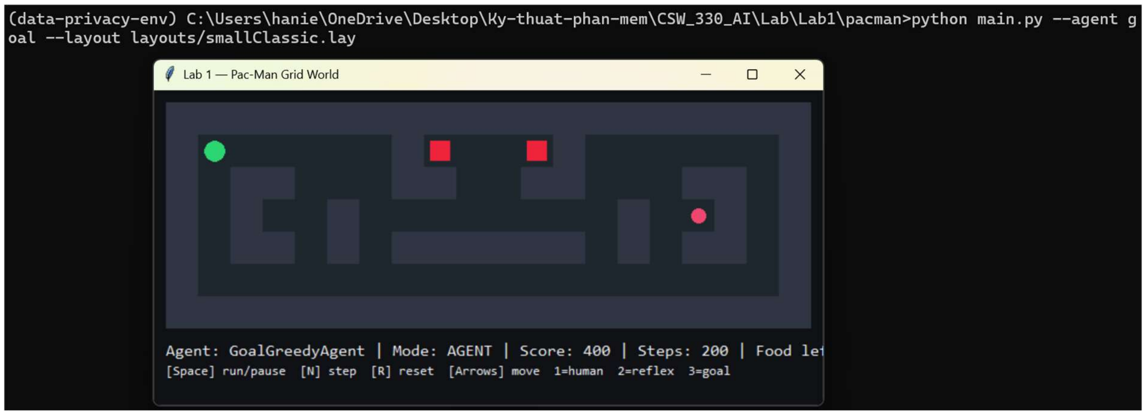
python main.py --agent reflex --layout layouts/smallClassic.lay



```
(data-privacy-env) C:\Users\hanie\OneDrive\Desktop\Ky-thuat-phan-mem\CSW_330_AI\Lab\Lab1\pacman>python main.py --agent r
eflex --layout layouts/smallClassic.lay
```

```
Agent: ReflexAgent | Mode: AGENT | Score: 56 | Steps: 174 | Food left: 34
[Space] run/pause  [N] step  [R] reset  [Arrows] move  1=human  2=reflex  3=goal
```

python main.py --agent goal --layout layouts/smallClassic.lay

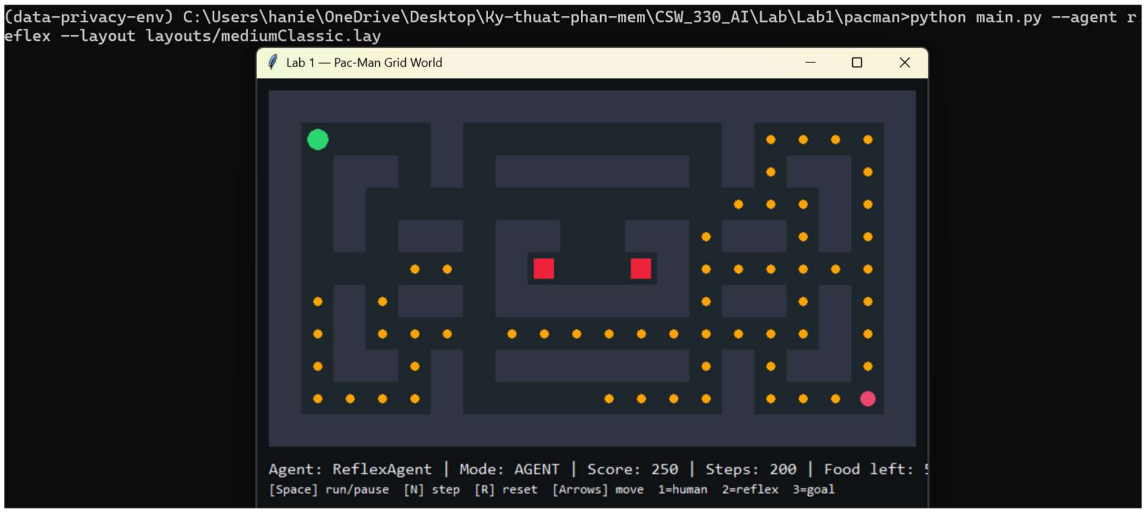


```
(data-privacy-env) C:\Users\hanie\OneDrive\Desktop\Ky-thuat-phan-mem\CSW_330_AI\Lab\Lab1\pacman>python main.py --agent g
oal --layout layouts/smallClassic.lay
```

```
Agent: GoalGreedyAgent | Mode: AGENT | Score: 400 | Steps: 200 | Food lef
[Space] run/pause  [N] step  [R] reset  [Arrows] move  1=human  2=reflex  3=goal
```

2. Then repeat on a more complex map:

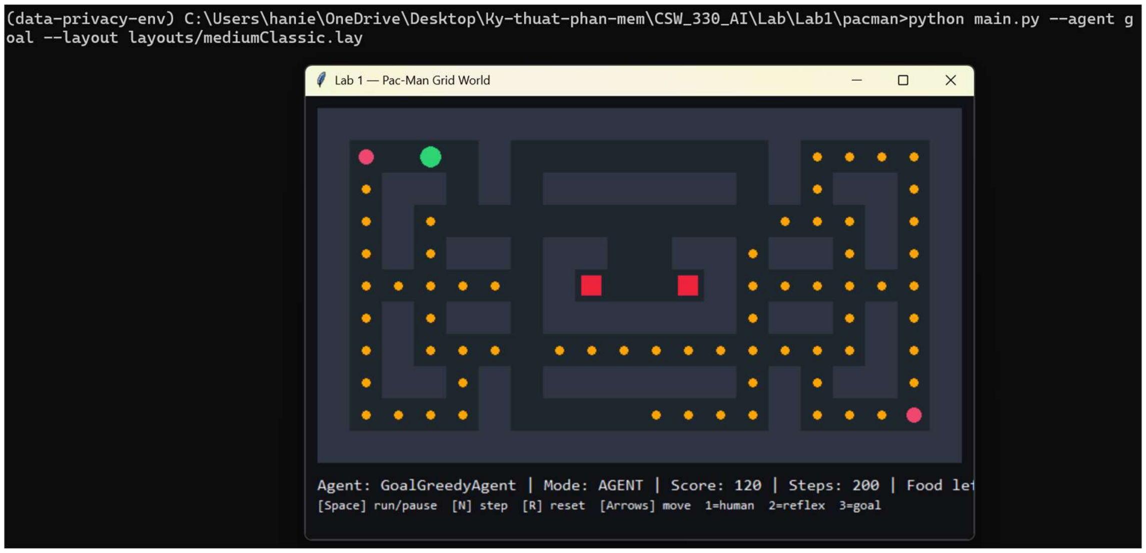
python main.py --agent reflex --layout layouts/mediumClassic.lay



```
(data-privacy-env) C:\Users\hanie\OneDrive\Desktop\Ky-thuat-phan-mem\CSW_330_AI\Lab\Lab1\pacman>python main.py --agent r
eflex --layout layouts/mediumClassic.lay
```

```
Agent: ReflexAgent | Mode: AGENT | Score: 250 | Steps: 200 | Food left: 5
[Space] run/pause  [N] step  [R] reset  [Arrows] move  1=human  2=reflex  3=goal
```

python main.py --agent goal --layout layouts/mediumClassic.lay



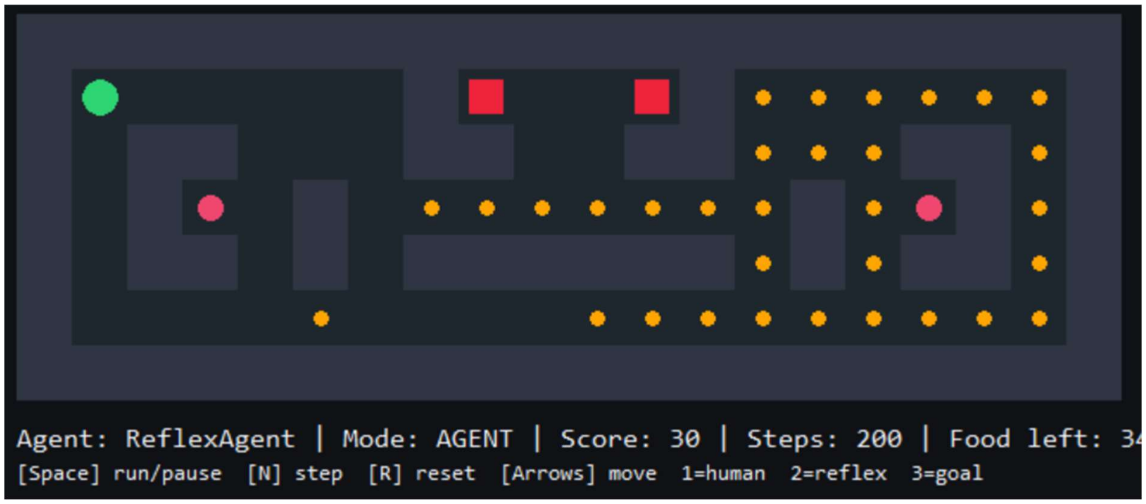
```
(data-privacy-env) C:\Users\hanie\OneDrive\Desktop\Ky-thuat-phan-mem\CSW_330_AI\Lab\Lab1\pacman>python main.py --agent g
oal --layout layouts/mediumClassic.lay
```

```
Agent: GoalGreedyAgent | Mode: AGENT | Score: 120 | Steps: 200 | Food lef
[Space] run/pause  [N] step  [R] reset  [Arrows] move  1=human  2=reflex  3=goal
```

Step 2: Collect data (15 points)

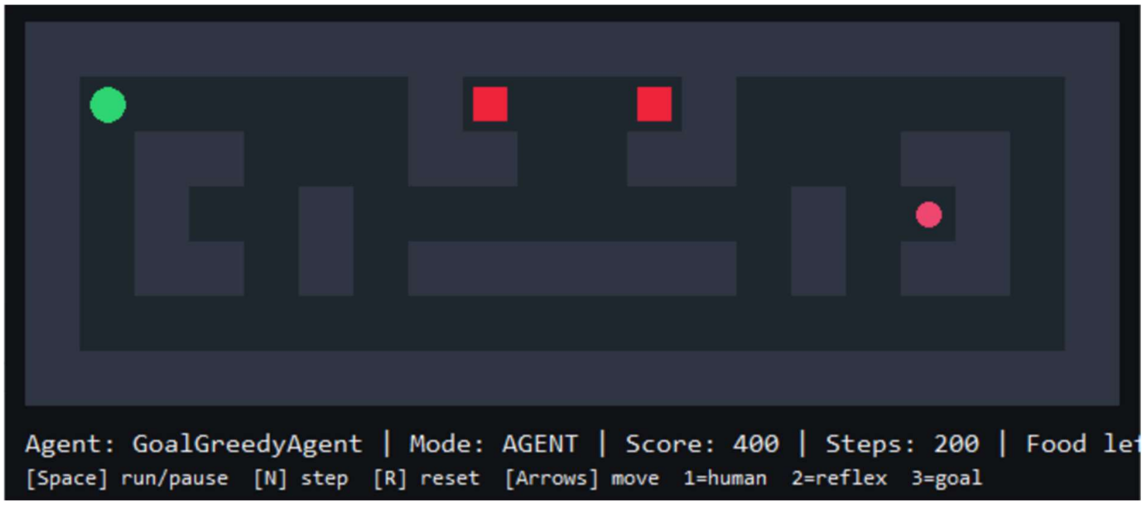
1. For each layout and each agent type, run five independent simulations.

- smallClassic – Reflex Agent



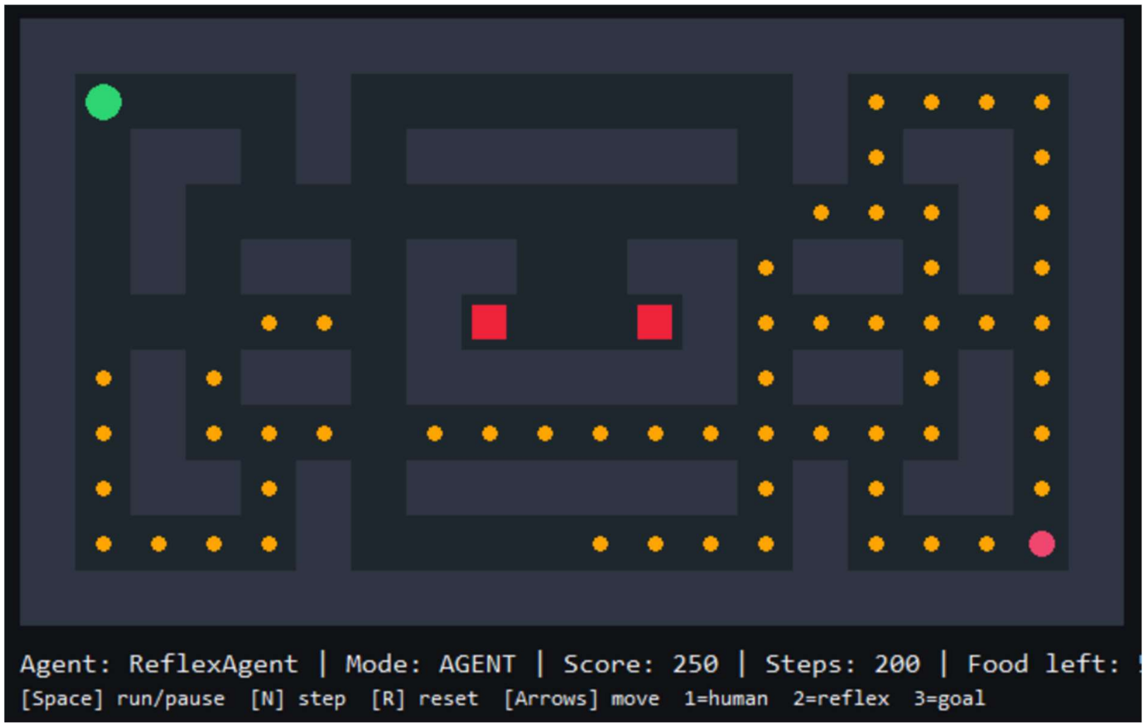
```
Agent: ReflexAgent | Mode: AGENT | Score: 30 | Steps: 200 | Food left: 34
[Space] run/pause  [N] step  [R] reset  [Arrows] move  1=human  2=reflex  3=goal
```

- smallClassic – Goal-Based Agent

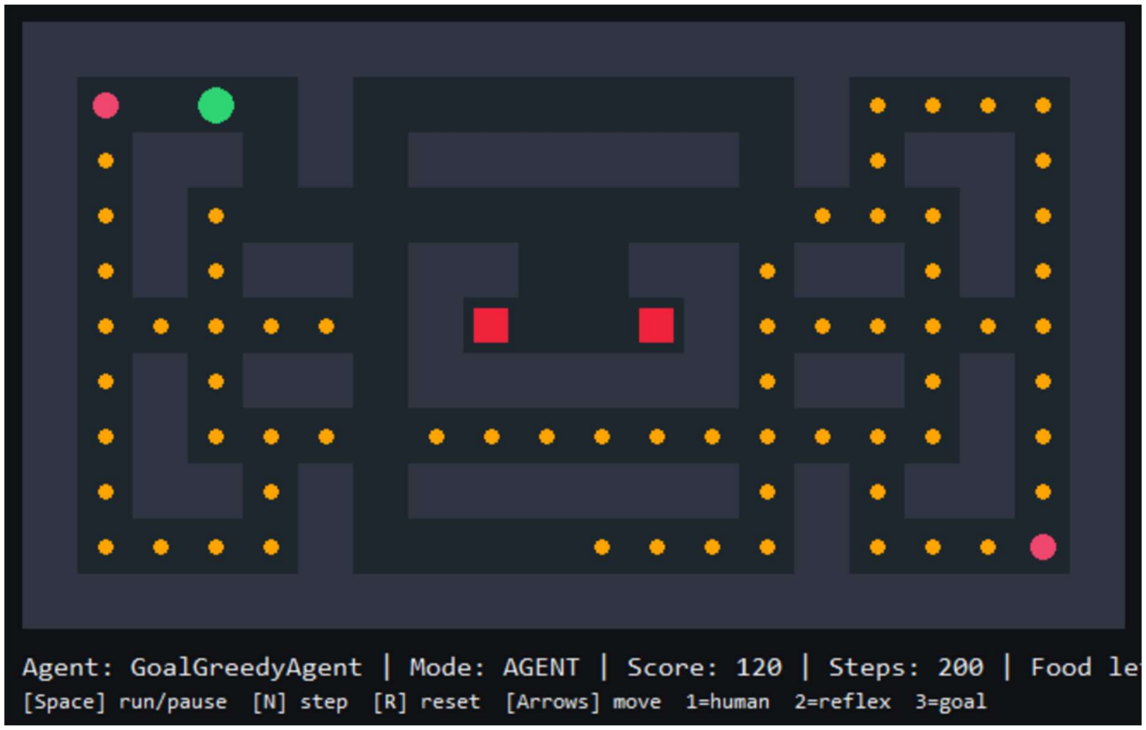


```
Agent: GoalGreedyAgent | Mode: AGENT | Score: 400 | Steps: 200 | Food lef
[Space] run/pause  [N] step  [R] reset  [Arrows] move  1=human  2=reflex  3=goal
```

- mediumClassic – Reflex Agent


```
Agent: ReflexAgent | Mode: AGENT | Score: 250 | Steps: 200 | Food left:
[Space] run/pause  [N] step  [R] reset  [Arrows] move  1=human  2=reflex  3=goal
```

- mediumClassic – Goal-Based Agent


```
Agent: GoalGreedyAgent | Mode: AGENT | Score: 120 | Steps: 200 | Food le
[Space] run/pause  [N] step  [R] reset  [Arrows] move  1=human  2=reflex  3=goal
```

2. Record at the end of each run: total score, number of steps, food remaining, and ghosts eaten.

Layout	**Agent Type**	**Average Score**	**Average Steps Taken**	**Total Food Remaining**	**Total Ghosts Eaten**

smallClassic	Reflex	30	200	32	0
smallClassic	GoalBased	400	200	0	0
mediumClassic	Reflex	250	200	55	0
mediumClassic	GoalBased	120	200	63	0

Step 3: Observations (20 points)

- Does the Reflex Agent loop or get stuck?

  Yes. In both smallClassic and mediumClassic layouts, the Reflex Agent tends to get stuck when it reaches a boundary or corner. It often loops back and forth along the wall instead of finding a new path to the remaining food.
- Does the Goal-Based Agent reduce distance to food more consistently?

  Yes. In the smallClassic layout, the Goal-Based Agent moves more efficiently and consistently reduces the distance to nearby food. However, in the mediumClassic layout, the Goal-Based Agent initially follows a similar path to the Reflex Agent but eventually gets stuck earlier, failing to reach all food.
- Behavior near ghosts and power pellets
  - SmallClassic

    Reflex Agent: When near power pellets, it ignores them and continues straight to collect food instead. It tends to move toward the left side (away from the ghost side) and eventually gets stuck near the corner.

    Goal-Based Agent: It collects most of the food first, then moves to the corners to pick up power pellets. However, it still avoids corners that contain ghosts and does not actively approach them.
  - MediumClassic

    Reflex Agent: The power pellet lies along its normal route, so it picks it up incidentally while collecting food. It does not move toward corners or attempt to chase ghosts.

    Goal-Based Agent: Before reaching the power pellet, it starts looping in place, suggesting it got stuck in a local loop. It also avoids corners with ghosts, similar to the Reflex Agent.
- Which finishes faster or leaves less food?
  - SmallClassic: The Goal-Based Agent finishes faster and leaves less food, while the Reflex Agent gets stuck before finishing.
  - MediumClassic: The Reflex Agent leaves slightly less food, but both agents fail to finish the map completely.

Step 4: Discussion

- How the Reflex Agent makes decision when it fails to reach food?

```
# Rule 1: Eat if adjacent food exists
        for a in self.eat_priority:
            nr, nc = nbrs[a]
            if (nr, nc) != (r, c) and env.grid[nr][nc] == '.' and
(nr, nc) not in env.eaten:
                return a

        # Rule 2: Otherwise move to any passable neighbor
        for a in self.move_priority:
            nr, nc = nbrs[a]
            if (nr, nc) != (r, c):
                return a

        # Rule 3: If surrounded by walls, select the first
available move in the priority list.
        return self.move_priority[0]
```

  - When there is no food adjacent to its position, the Reflex Agent has no plan or memory.
  - It simply reacts to immediate surroundings following fixed rules: eat nearby food if possible, otherwise move randomly to a passable cell.
  - As a result, when food is far away, the agent tends to wander aimlessly, loop around walls, or get stuck indefinitely.

- How the Goal-Based Agent differs conceptually (goal selection + greedy step).

```
def act(self, env: GridWorld) -> Action:
        self._perceive_update(env)
        if self.target and self.target in env.eaten:
            self.target = None
        if self.target is None:
            self._select_target(env)

        a = self._greedy_step(env)
        if a:
            return a
        return self._explore_step(env)
```

  - Unlike the Reflex Agent, the Goal-Based Agent maintains an explicit goal — typically the nearest uneaten food.
  - It first selects a target (_select_target), then moves toward it using a greedy step (_greedy_step). If the target is eaten or no longer valid, it dynamically selects a new goal.
  - This makes its behavior purpose-driven, more systematic, and generally more efficient than the Reflex Agent.

- Compare efficiency (score/steps) and robustness (ability to finish).

Aspect	**Reflex Agent**	**Goal-Based Agent**
Decision Type	Rule-based and reactive (local view only)	Goal-oriented and greedy toward a target
Efficiency	Uses more steps and often wanders unnecessarily	Takes fewer steps, moves directly toward food
Robustness	Easily gets stuck or loops near walls	Adapts better, can reach distant goals
Completion	Often fails to finish when food is far	More likely to complete maps with scattered food