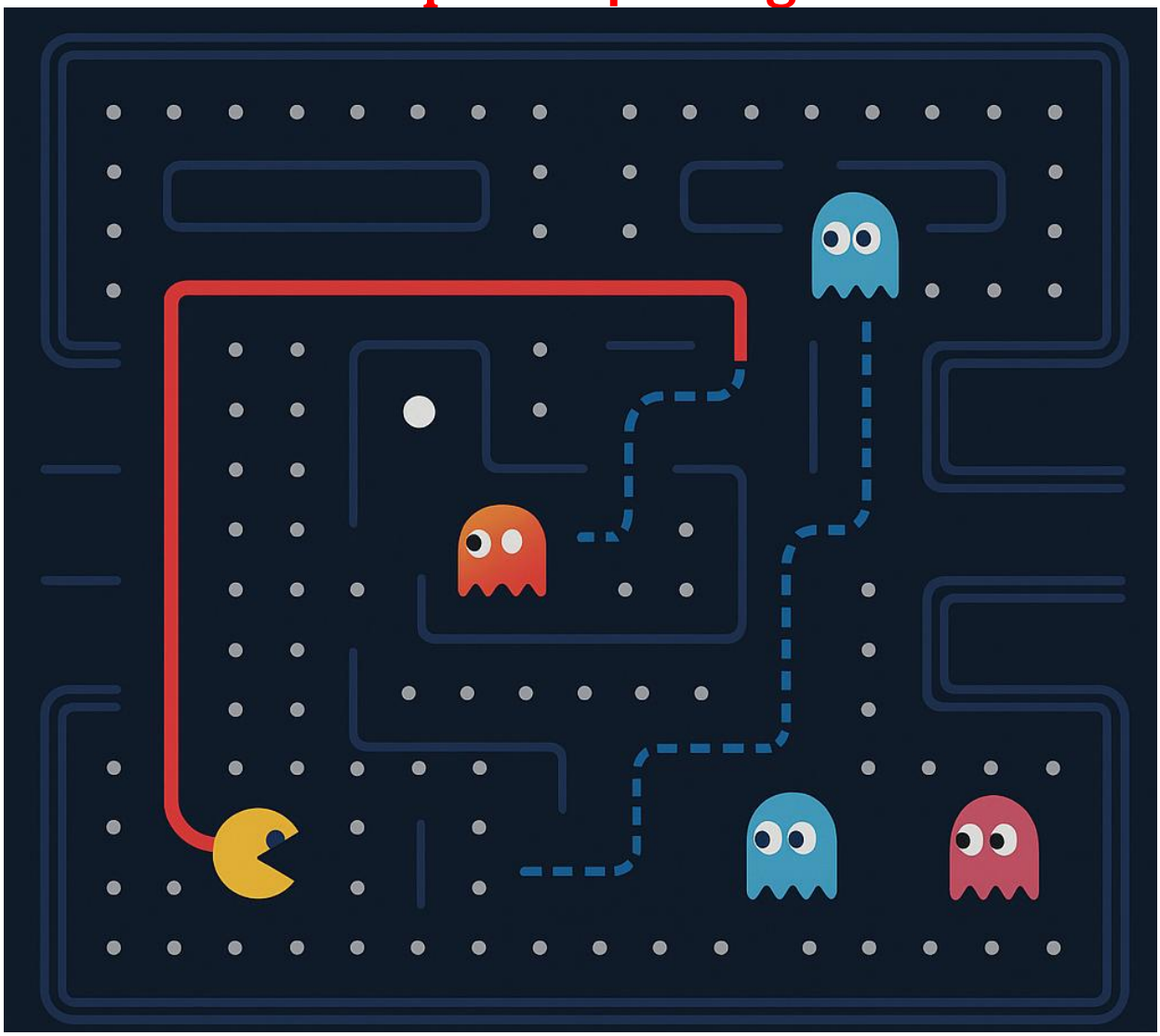
# Đề tài 2: Xây dựng mô phỏng trò chơi Pac-Man sử dụng các thuật toán tìm kiếm và hành vi phản xạ thông minh



**Chú thích:**

**: Đường đi tối ưu để ăn thức ăn**

**: Đường đi tránh ma hoặc để đuổi ma khi có sức mạnh**

## 1. Mục tiêu bài toán:

- Mô phỏng một game Pac-Man có thể tự động điều hướng nhờ các thuật toán tìm kiếm thông minh.
- Pac-Man có thể:
  - Tự tìm đường ăn hết thức ăn (dot và power pellet).
  - Tránh ma đang truy đuổi bằng thuật toán heuristic.
  - Chủ động ăn ma khi ở trạng thái có sức mạnh (sau khi ăn chấm lớn), có tính toán thời gian màu xanh.
- So sánh hiệu quả các thuật toán tìm kiếm: BFS, DFS, UCS, A*, heuristic,...
- Tích hợp một agent thông minh (ReflexAgent) ra quyết định theo tình huống.

## 2. Ý tưởng để giải quyết bài toán

### 2.1. Mô hình hóa bản đồ Pac-Man

- Biểu diễn mê cung dạng lưới 2D (grid).
- Mỗi ô có thể là:
  - Tường (#)
  - Đường trống ( )
  - Chấm thường (.)
  - Chấm lớn (o)
  - Vị trí ma (G)
  - Vị trí Pac-Man (P)

### 2.2. Cài đặt các thuật toán tìm kiếm cơ bản

- breadthFirstSearch: Tìm đường gần nhất đến dot (đơn giản).
- depthFirstSearch: Dùng cho so sánh (hiệu quả thấp).
- uniformCostSearch: Tối ưu khi có trọng số (ví dụ: né ma có thể là chi phí cao).

- aStarSearch: Dùng để đến đích (dot/goal) nhanh và chính xác.

### 2.3. Xây dựng các bài toán mở rộng (Problem definitions)

- CornersProblem: Pac-Man phải đi qua 4 góc.
- FoodSearchProblem: Tìm đường ăn hết tất cả thức ăn.
- findPathToClosestDot: Tìm dot gần nhất và đến ăn.

### 2.4. Tích hợp hành vi thông minh

- ReflexAgent:
  + Tránh ma nếu đang không có sức mạnh.
  + Ăn ma nếu vừa ăn chấm lớn và thời gian còn lại đủ.
  + Ra quyết định dựa vào đánh giá trạng thái: gần dot? gần ma? còn thời gian màu xanh bao lâu?

### 2.5. Quản lý trạng thái Power Pellet (chấm lớn)

- Khi ăn chấm lớn, thiết lập biến đếm thời gian màu xanh (sức mạnh ăn ma).
- Nếu thời gian gần hết mà đang đuổi ma → dừng lại hoặc tránh xa.
- Nếu còn thời gian dài → tìm đường đến ma gần nhất để ăn.

### 2.6. Giao diện và mô phỏng

- Dùng pygame, matplotlib, hoặc console để hiển thị.
- Hiển thị Pac-Man, ma, thức ăn, đường đi.
- Cho phép chuyển đổi giữa các thuật toán để so sánh.

### 2.7. Đánh giá và thử nghiệm

- So sánh:
  + Đường đi của từng thuật toán.
  + Số bước để hoàn thành màn chơi.
  + Số lần ăn ma thành công khi có power pellet.

+ Số lần bị mất mạng do hết thời gian màu xanh.

**Câu hỏi 1:** Giải thích sự khác nhau giữa breadthFirstSearch và depthFirstSearch. Trong bối cảnh Pac-Man, thuật toán DFS hay BFS thường được ưu tiên hơn? Tại sao?

---

**Câu hỏi 2:** Trong game Pac-Man, tại sao cần phải dùng thuật toán tìm kiếm thay vì di chuyển ngẫu nhiên hoặc theo một hướng cố định?

---

**Câu hỏi 3:** Trong hàm ReflexAgent, bạn sẽ thiết kế heuristic thế nào để Pac-Man vừa tránh ma, vừa biết khi nào nên đuổi ma (sau khi ăn chấm lớn), và không bị hết thời gian màu xanh giữa đường?

---

**Câu hỏi 4:** Khi sử dụng A* để tìm đường đi đến thức ăn hoặc góc bản đồ, bạn sẽ thiết kế hàm heuristic (cornersHeuristic, foodHeuristic) như thế nào để đảm bảo tính khả thi và hiệu quả?

---