Contents

[Project 2 3](#_Toc184487329)

[Giới thiệu bài toán 3](#_Toc184487330)

[Phương pháp sử dụng 3](#_Toc184487331)

[Cài đặt 3](#_Toc184487332)

[Reflex Agent 3](#_Toc184487333)

[Kết quả 3](#_Toc184487334)

[Kết luận. 3](#_Toc184487335)

# Project 2

## Giới thiệu bài toán

Pacman là một bài toán cổ điển trong lĩnh vực trí tuệ nhân tạo, nơi nhân vật Pacman cần di chuyển trong mê cung để thu thập các viên thức ăn trong khi tránh các bóng ma. Bài toán yêu cầu tìm kiếm một chiến lược hiệu quả để đạt được mục tiêu này

## Phương pháp sử dụng

Các thuật toán bao gồm Reflex Agent, Minimax, Alpha-Beta Pruning,

Expectimax, and the Evaluation Function.

## Cài đặt

### Reflex Agent

Trong trò chơi Pacman, nhiệm vụ chính của Pacman là ăn hết thức ăn, tránh các ghost, và tối đa hóa điểm số. Để đạt được điều này, ReflexAgent được thiết kế như một tác nhân phản ứng nhanh, dựa vào trạng thái hiện tại của trò chơi để đưa ra hành động tốt nhất

**Mô tả thuật toán:**

a. Cách chọn hành động (getAction)

Hàm getAction được sử dụng để chọn hành động tốt nhất tại mỗi bước. Quy trình hoạt động như sau:

1. Thu thập các hành động hợp lệ: Sử dụng gameState.getLegalActions() để lấy danh sách các hành động mà Pacman có thể thực hiện.
2. Tính điểm đánh giá: Gọi evaluationFunction để đánh giá điểm số cho mỗi hành động dựa trên trạng thái tiếp theo.
3. Chọn hành động tốt nhất:

* Xác định điểm số cao nhất.

- Trong trường hợp có nhiều hành động có điểm số cao nhất, chọn ngẫu nhiên một hành động từ danh sách.

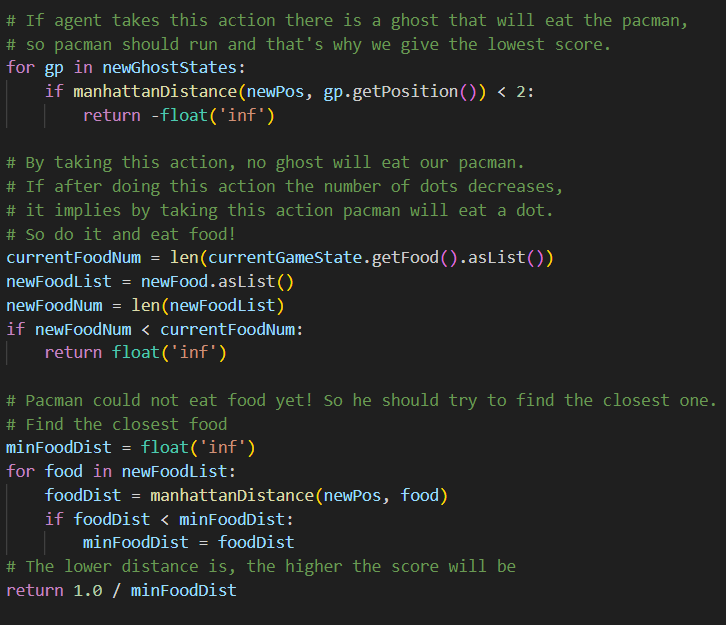
b. Hàm đánh giá trạng thái (evaluationFunction)

Hàm evaluationFunction là trái tim của ReflexAgent, xác định mức độ "tốt" của một hành động. Nó đánh giá dựa trên các tiêu chí sau:

1. Tránh ghost: Nếu Pacman di chuyển đến vị trí quá gần một ghost (khoảng cách Manhattan < 2 ô), trạng thái này sẽ bị phạt nặng với điểm là -∞.
2. Ưu tiên ăn thức ăn: Nếu hành động dẫn đến việc Pacman ăn một thức ăn, trạng thái này sẽ được thưởng điểm rất cao với giá trị +∞.
3. Tìm thức ăn gần nhất: Nếu không thể ăn thức ăn ngay lập tức, trạng thái sẽ được đánh giá dựa trên khoảng cách đến thức ăn gần nhất. Điểm số tỷ lệ nghịch với khoảng cách này (càng gần, điểm càng cao).

c. Công thức đánh giá

- Khoảng cách Manhattan: Được dùng để tính khoảng cách giữa hai điểm trong không gian lưới.



### Minimax

MinimaxAgent được thiết kế để tối ưu hóa chiến lược chơi, không chỉ dựa vào trạng thái hiện tại mà còn dự đoán các hành động tương lai của đối thủ (ghost). Bằng cách áp dụng thuật toán Minimax, agent có thể đưa ra các quyết định thông minh nhằm tối đa hóa điểm số và giảm thiểu rủi ro từ ghost

**Mô tả thuật toán (Algorithm Description)**

1. Mục tiêu của MinimaxAgent

Mục tiêu chính của MinimaxAgent là:

* Maximizing Pacman’s score: Pacman cố gắng tối đa hóa điểm số của mình.

- Minimizing ghosts’ advantage: Ghost cố gắng giảm lợi thế của Pacman bằng cách tối thiểu hóa điểm số Pacman có thể đạt được.

b. Quy trình hoạt động của Minimax

1. Điều kiện dừng:

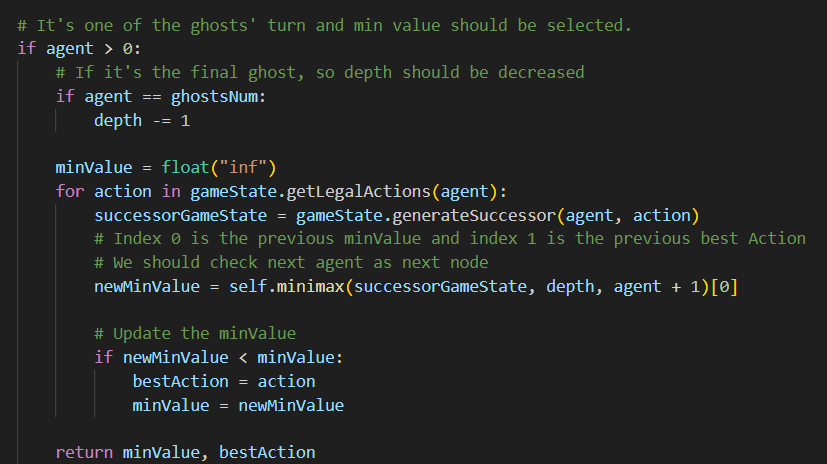
* Trò chơi kết thúc (Pacman thắng/thua).
* Độ sâu tối đa của cây tìm kiếm (do người dùng quy định) đạt đến.

2. Phân tích các tác nhân:

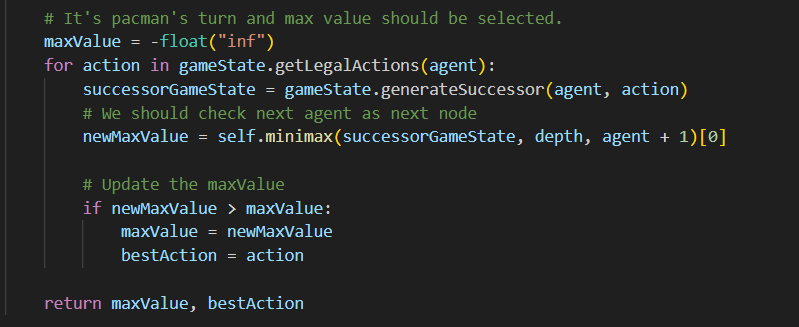
* Pacman (agent 0): Pacman là tác nhân Max, cố gắng tối đa hóa giá trị trạng thái.
* Ghosts (agents ≥ 1): Ghosts là tác nhân Min, cố gắng tối thiểu hóa giá trị trạng thái của Pacman.

**Công thức chính**:

* **Min node**:



* Max node:



### AlphaBeta-Prunning

AlphaBetaAgent được xây dựng dựa trên thuật toán Minimax, nhưng bổ sung cắt tỉa Alpha-Beta (Alpha-Beta Pruning), một kỹ thuật tối ưu hóa giúp giảm số lượng trạng thái cần phải kiểm tra trong quá trình tìm kiếm, từ đó giảm độ phức tạp tính toán

**Mô tả thuật toán:**

Thuật toán sử dụng hai tham số alpha và beta để cắt tỉa các nhánh không cần thiết của cây minimax:

* Alpha: Giá trị tốt nhất mà tác nhân Max có thể hy vọng đạt được.
* Beta: Giá trị tốt nhất mà tác nhân Min có thể hy vọng đạt được.

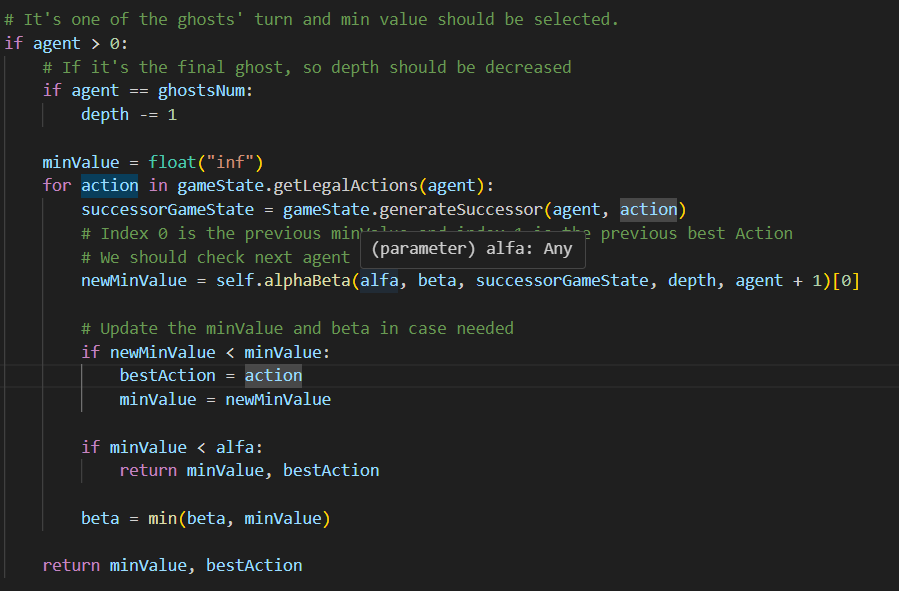
Nếu tại một điểm trong quá trình tính toán, alpha ≥ beta, nghĩa là không cần tiếp tục kiểm tra nhánh đó vì giá trị tốt nhất có thể đạt được đã bị giới hạn

**Quy trình tìm kiếm (Alpha-Beta):**

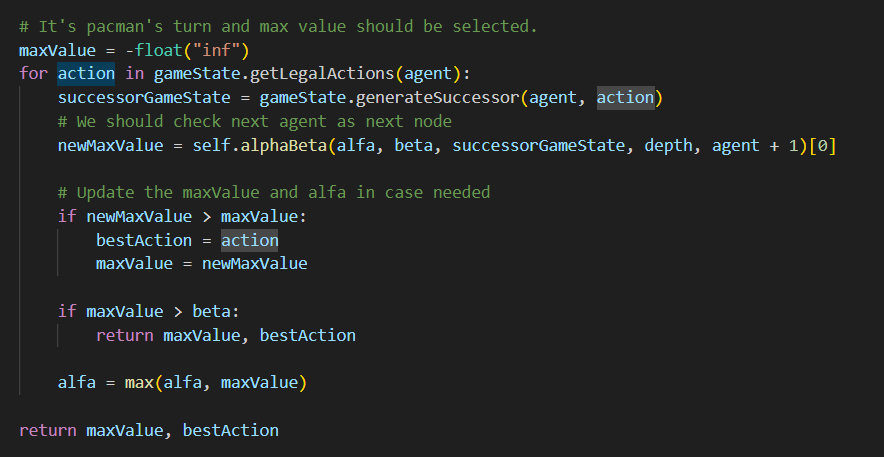
* Max Node (Pacman): Tại đây, alpha được cập nhật nếu giá trị tìm được lớn hơn alpha hiện tại.
* Min Node (Ghost): Tại đây, beta được cập nhật nếu giá trị tìm được nhỏ hơn beta hiện tại.

Nếu giá trị tìm được tại Max Node vượt quá beta hoặc tại Min Node nhỏ hơn alpha, việc tìm kiếm sẽ bị cắt bỏ (pruning).

**Min node:**



**Max node:**

****

### ExpectimaxAgent

ExpectimaxAgent là một thuật toán tìm kiếm tối ưu, thay thế chiến lược Minimax bằng Expectimax để xử lý các trạng thái có yếu tố ngẫu nhiên, như hành động không thể dự đoán của ghost. Expectimax là một thuật toán tìm kiếm dựa trên cây quyết định, được sử dụng trong các tình huống có yếu tố ngẫu nhiên. Expectimax khác với Minimax ở chỗ:

* Với Minimax, ghost được coi như các đối thủ "tối ưu", luôn chọn hành động để gây bất lợi tối đa cho Pacman.
* Với Expectimax, ghost được coi như các tác nhân hành động ngẫu nhiên, chọn hành động dựa trên xác suất phân phối (trong trường hợp này là đều).

**Thuật toán:**

**Điều kiện dừng**:

* Nếu độ sâu tìm kiếm đạt giới hạn hoặc trò chơi kết thúc (Pacman thắng/thua), trả về giá trị trạng thái bằng evaluationFunction

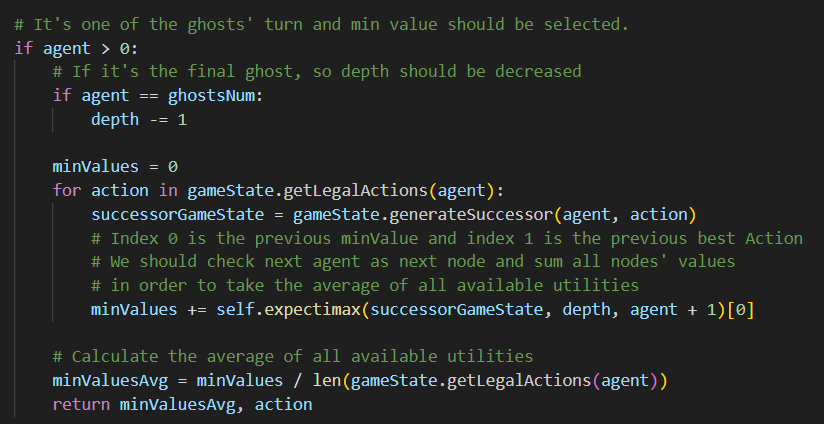
**Tác nhân hiện tại**:

* **Pacman (agent 0)**: Tìm giá trị tối đa từ tất cả các hành động khả thi.
* **Ghost (agents ≥ 1)**: Tính giá trị kỳ vọng (expected value) dựa trên tất cả các hành động hợp lệ.

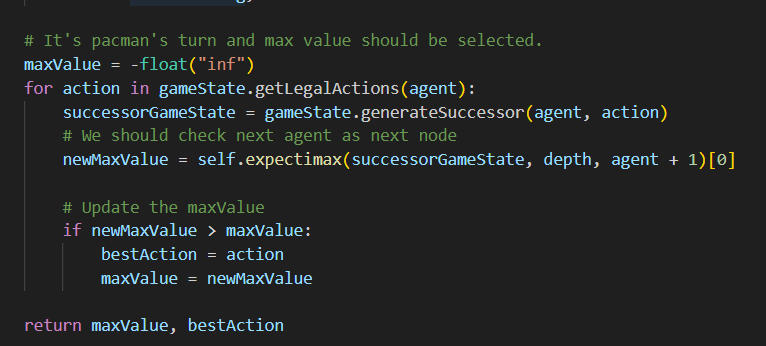
**Tính toán giá trị**:

* Với Pacman: Chọn hành động đem lại giá trị lớn nhất.
* Với Ghost: Tính trung bình cộng giá trị các trạng thái kế tiếp, giả sử mỗi hành động được chọn với xác suất bằng nhau

Nếu là một trong các lượt của ghost

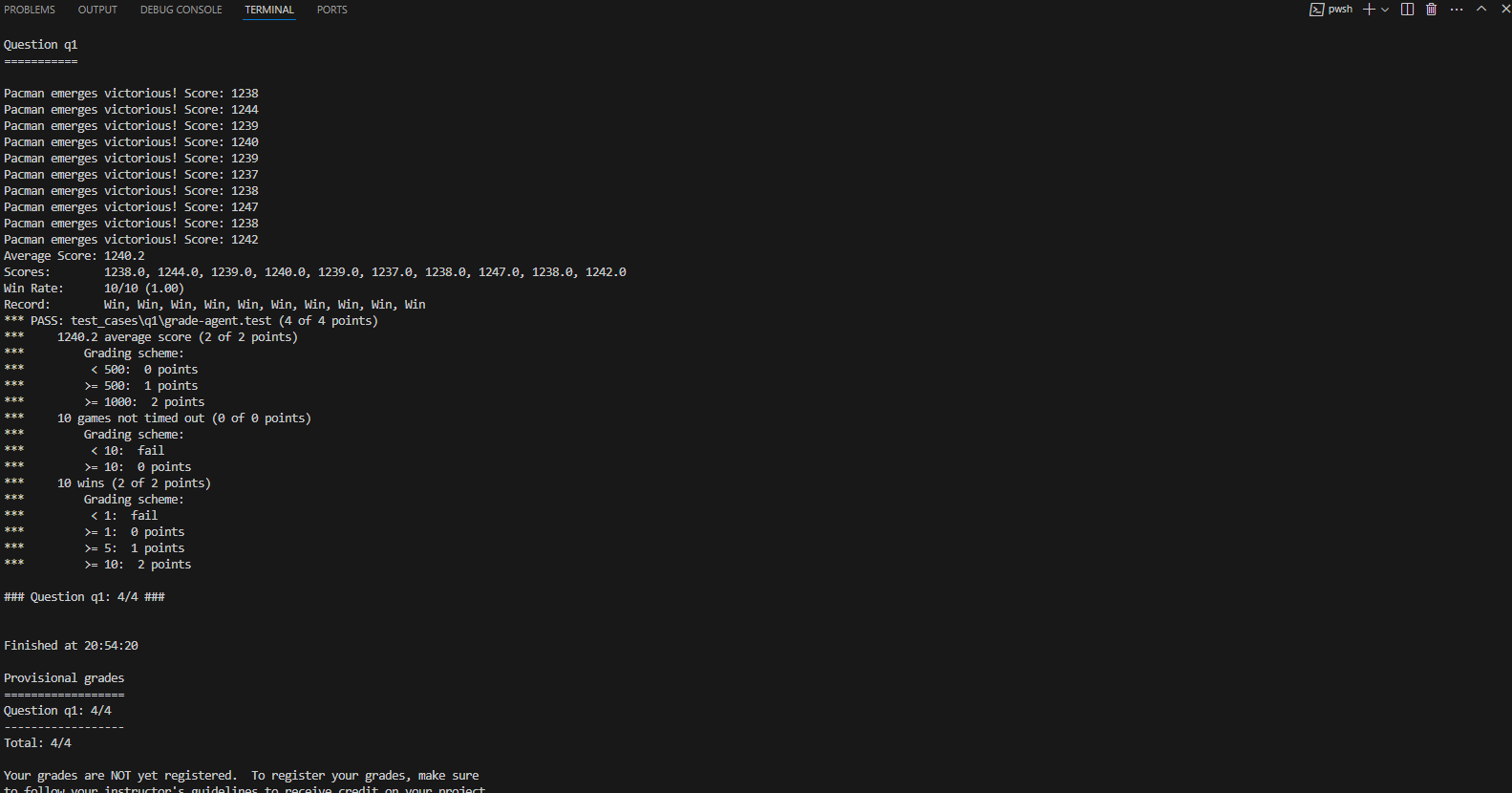


Nếu là lượt của pacman:

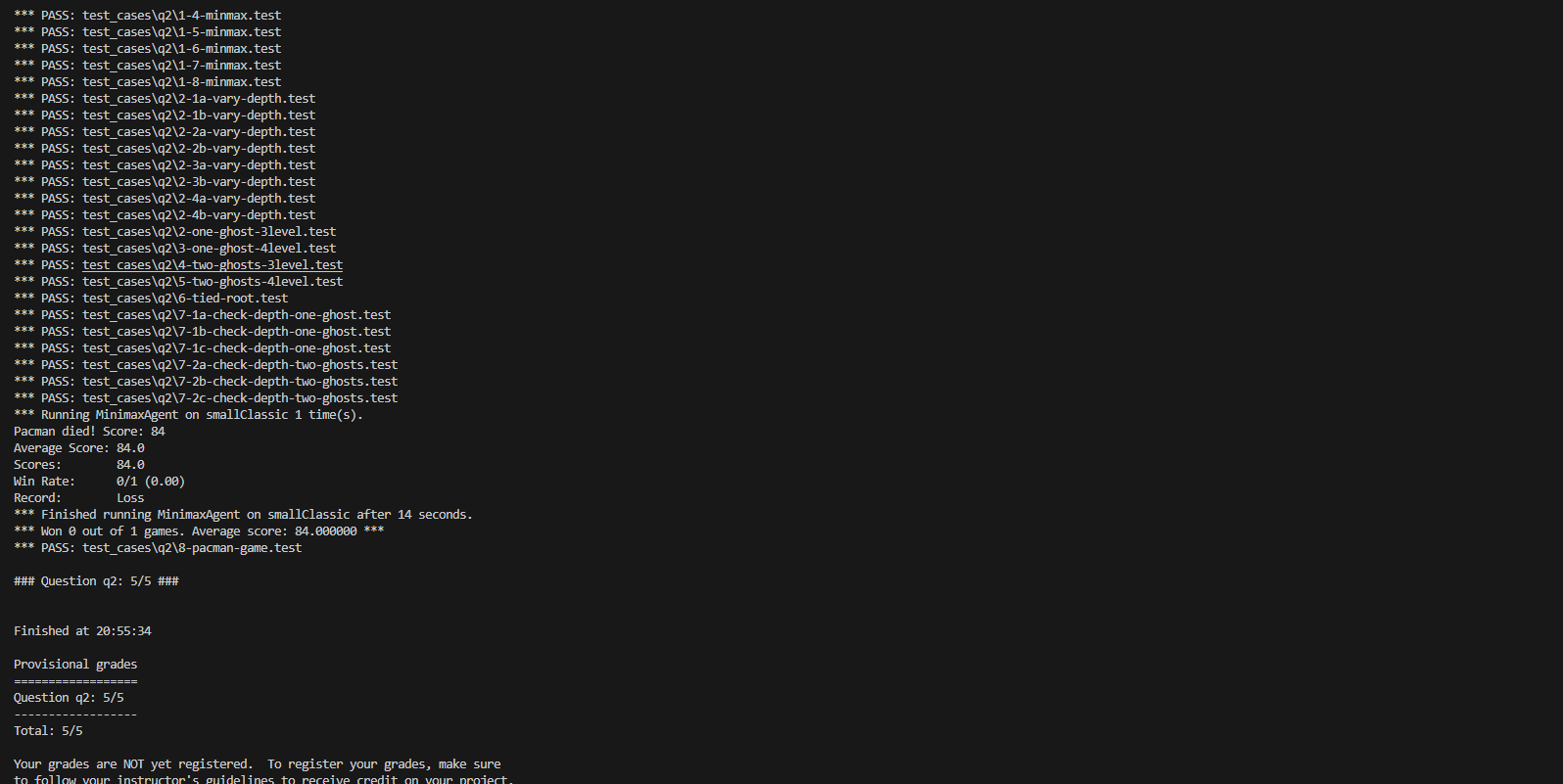


## Kết quả

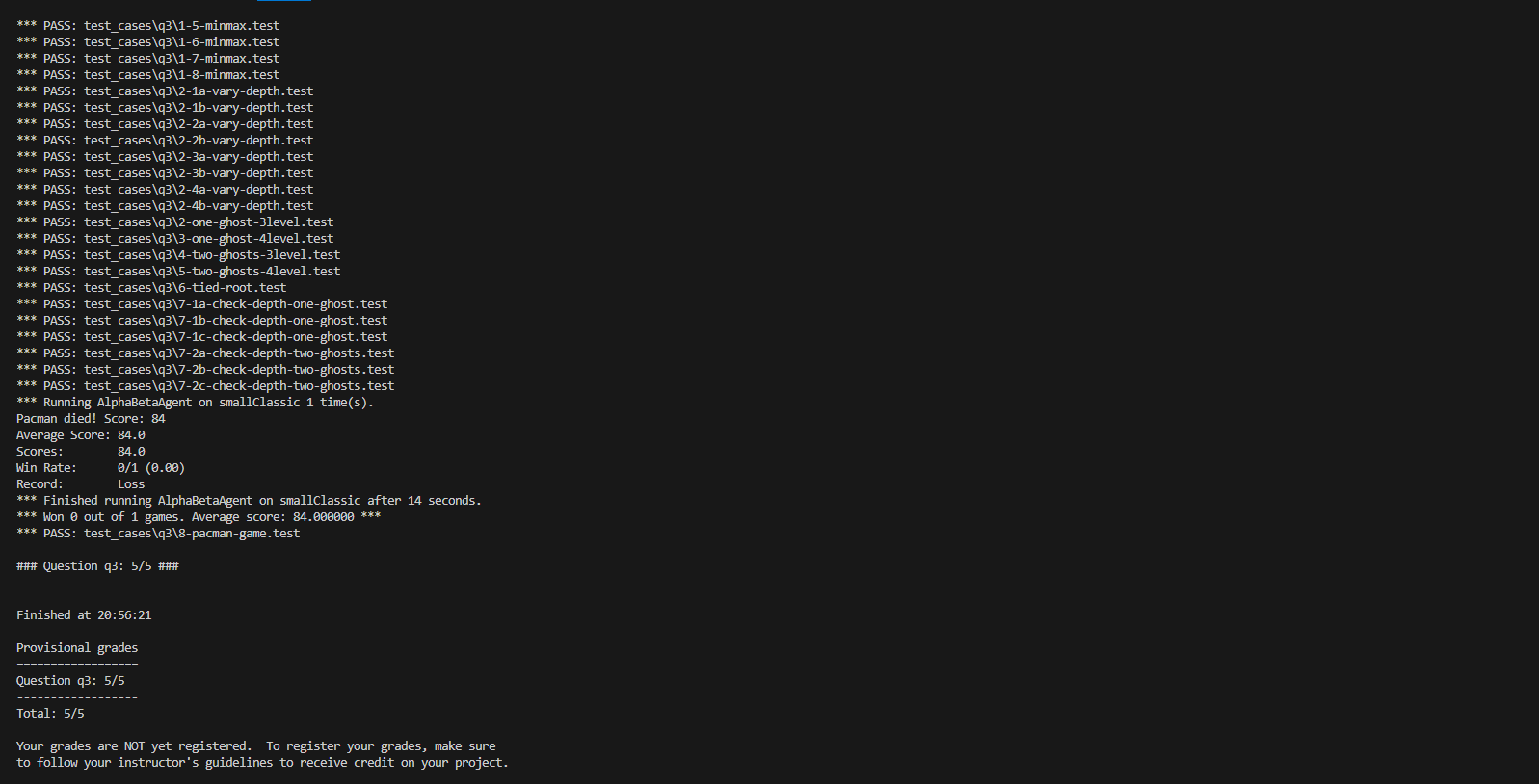
* Kết quả q1



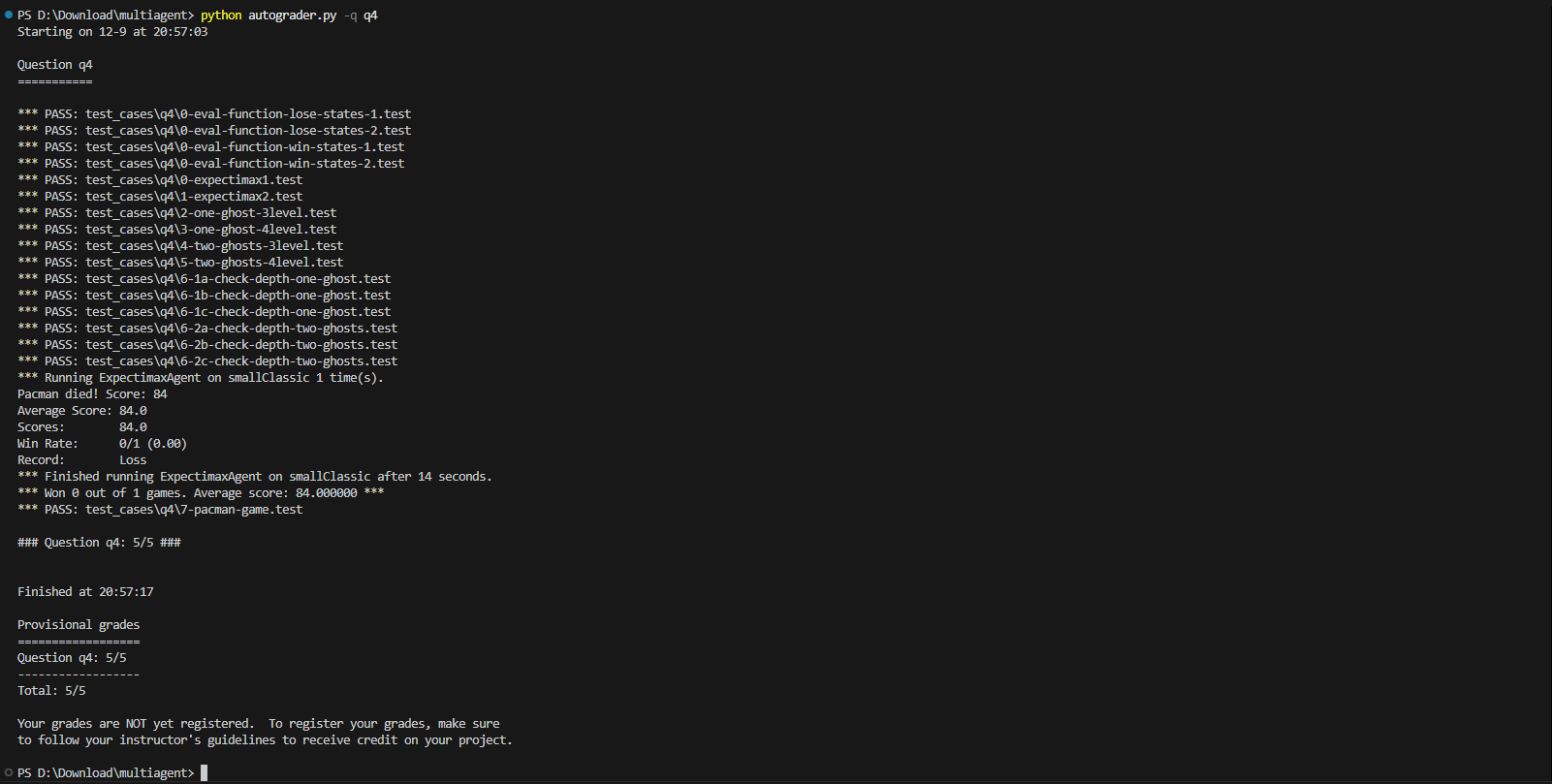
* Kết quả q2



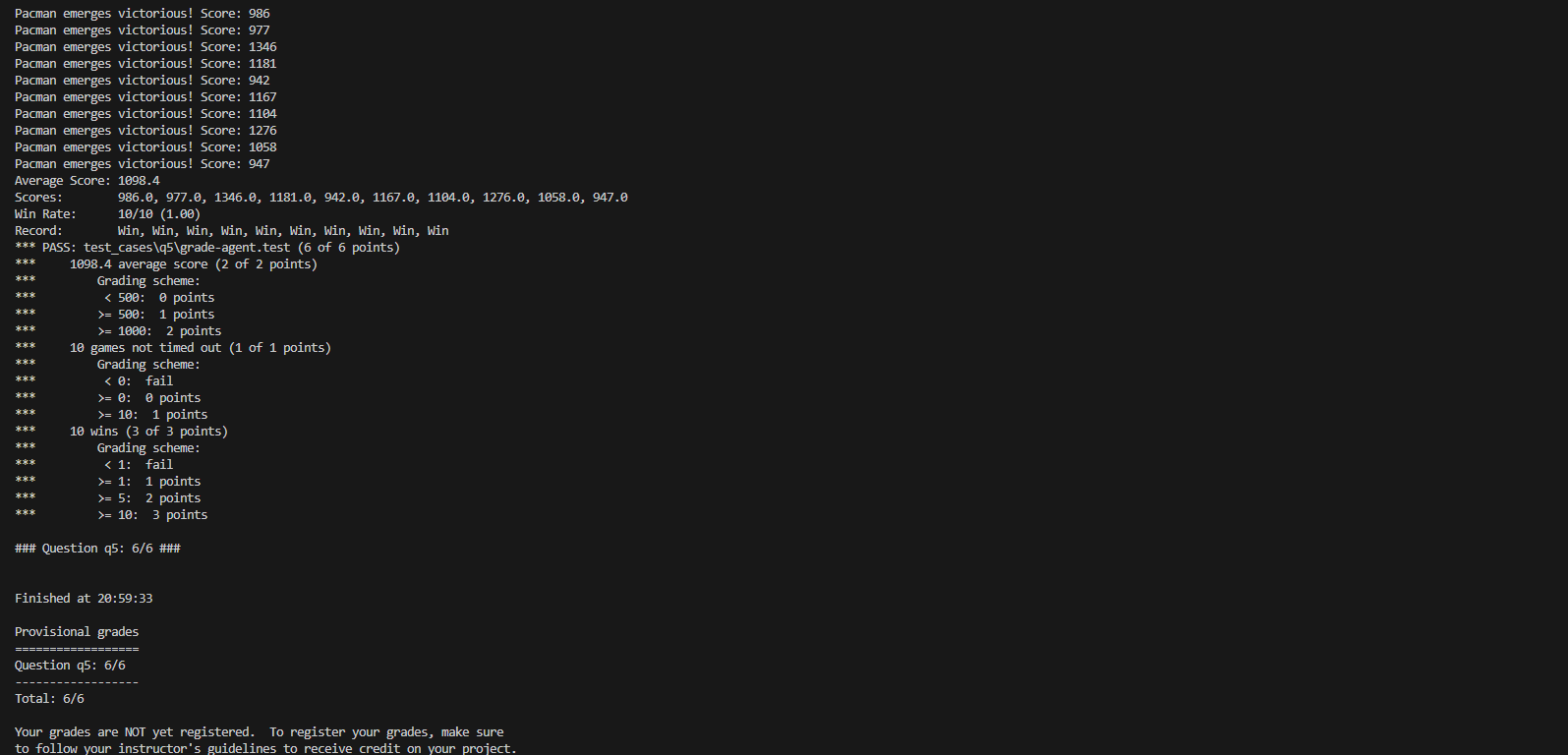
* Kết quả q3



* Kết quả q4



* Kết quả q5



## Kết luận.

* Kết quả toàn bộ dự án

