Ho Chi Minh City, Viet Nam
Faculty of Computer Science
University of Information Technology

Evaluation functions for Minimax/AlphaBeta/Expectimax

Tăng Gia Hân – 22520394

3 May 2024

1 Thiết kế hàm lượng giá

Mục tiêu của trò chơi, đó là làm sao để có thể đạt được điểm số cao nhất có thể và ăn hết toàn bộ food để kết thúc trò chơi.

Mỗi hành động mà Pacman thực hiện sẽ ảnh hưởng đến điểm số và điểm kết thúc của trò chơi, do đó ta cần thiết kế hàm lượng giá dựa trên các kết quả mà hành động này mang lại.

Object	Score
Time per second	-1
Food	10
Eat ghost	200
Win	500
Lost	-500
Capsule	0

Table 1:Điểm số tương ứng với mỗi hành động của trò chơi

2 Ý tưởng thiết kế hàm lượng giá

Hàm lượng giá được thiết kế dựa trên các đặc trưng sau:

- Lấy score hiện tại của trò chơi sử dụng **currentScore = currentGameState.getScore()** với hệ số **(2)** nhằm chọn các state dẫn đến việc gia tăng điểm số, giúp cải thiện khả năng sống sót và chiến thắng của Pacman. Tuy nhiên, nếu chỉ có một hệ số này, sẽ dẫn đến tình trạng Pacman không di chuyển khi cây tìm kiếm có độ sâu thấp và các không gian trạng thái xung quanh không giúp gia tăng điểm số.

- Tiếp theo, để tối đa điểm số đạt được thì việc tránh kéo dài thời gian và chiến thắng là quan trọng nhất, để đạt được thì ta cần phải kết thúc trò chơi càng nhanh càng tốt. Do đó thêm số lượng thức ăn còn lại vào hàm lượng giá với hệ số -4 khuyến khích Pacman ăn thật nhiều thức ăn càng sớm càng tốt: **len(foodList) (-4)**

- Khoảng cách đến thức ăn là một yếu tố quan trọng ảnh hưởng đến quyết định di chuyển của Pacman. Do độ sâu tìm kiếm vẫn có hạn, nên nếu thức ăn ở xa, sẽ tiếp tục xảy ra hiện tượng Pacman bị đói. Thêm khoảng cách đến thức ăn **(distanceToClosestFood) (-1.5)** vào hàm lượng

giá với hệ số -1.5 để khuyến khích Pacman di chuyển đến các điểm thức ăn gần nhất. Điều này cũng có nghĩa là khoảng cách của Pacman đến thức ăn gần nhất càng lớn thì điểm càng âm.

- Tuy nhiên, một thách thức tiếp tục đối diện là khả năng thất bại, vì Pacman của chúng ta hoạt động trong một môi trường Adversarial, khiến cho va chạm với Ghost có thể dẫn đến kết thúc trò chơi với kết quả thua cuộc. Do đó, việc tránh xa Ghost là cần thiết để tránh rơi vào tình thế này. Chính vì vậy, đặc trưng **distanceToClosestActiveGhost** được sử dụng để đảm bảo rằng Pacman không tiếp xúc quá gần với Ghost, giảm nguy cơ bị bắt và tăng khả năng sống sót. Được tính như sau: **(-2)*(1/distanceToClosestActiveGhost)**. Điều này có nghĩa là khoảng cách đến Ghost hoạt động gần nhất càng lớn thì điểm càng âm.

- Việc tiêu thụ Capsule tuy sẽ không giúp gia tăng điểm số, nhưng sau hành động đó có thể mang lại lợi ích lớn cho Pacman trong trường hợp gặp phải Ghost, Ghost sẽ trở thành dạng sợ hãi, cho phép Pacman ăn chúng trong một khoảng thời gian nhất định, đồng thời nhận được điểm thưởng lớn. Tuy nhiên, việc tiêu thụ Capsule không nên được thực hiện một cách vô ý và không cần thiết, vì nếu Capsule và Ghost đang ở quá xa, việc tiêu thụ sẽ không hiệu quả và có thể làm mất đi một lượng vô ích. Do đó, sử dụng đặc trưng **(numberOfCapsulesLeft) (-20)** với hệ số âm rất cao -20 giúp Pacman chỉ tập trung vào việc tiêu thụ Capsule khi chúng ở gần đủ để đảm bảo hiệu quả và tối ưu hóa điểm số.

- Khi tiêu thụ Capsule, các Ghost sẽ trở thành bị dính hiệu ứng đặc biệt và có thể bị tiêu diệt bởi Pacman trong khoảng thời gian là 40 giây và nhận được điểm thưởng lớn. Đặc trưng **effectGhostDistance** với hệ số **(-2)** được chọn để đo lường khoảng cách từ vị trí hiện tại của Pacman đến Ghost khi chúng ở trong trạng thái sợ hãi (scared). Đặc trưng này giúp Pacman ưu tiên di chuyển đến các Ghost ở gần nhất khi chúng bị hiệu ứng của Capsule, nhằm tối đa hóa cơ hội tiêu thụ và nhận điểm số cao.

Công thức lượng giá:

score = 2 * currentScore – 1.5 * distanceToClosestFood – 2 * (1./distanceToClosestActiveGhost)

– 2 * effectGhostDistance – 20 * numberOfCapsulesLeft – 4 * (len(foodlist))

3 Thống kê thực nghiệm

Để việc so sánh kết quả các thuật toán được công bằng, mỗi thuật toán được đo 5 lần ở mỗi map, với các random seed tương tự có dạng 22520394 + số thứ tự lần chơi :

```
def runGames(layout, pacman, ghosts, display, numGames, record, numTraining=0, catchExceptions=False, timeout=30):
    import __main__
    __main__.__dict__['_display'] = display

    rules = ClassicGameRules(timeout)
    games = []

    for i in range(numGames):
        random.seed(22520394 + i)
        beQuiet = i < numTraining
        if beQuiet:
```

Figure 1: Mỗi lần thực nghiệm với 1 random seed khác nhau.

Kết quả thực nghiệm 5 lần ở các map khác nhau, đều được giới hạn depth = 2 với các thuật toán đã cài đặt với better evaluation function và scoreEvaluationFunction. Sau khi chạy lệnh **" .\script.bat "** sẽ tạo file **result.txt** chứa kết quả của quá trình thực thi.

3.1 So sánh giữa các giải thuật với hàm lượng giá mới

Ta có thể so sánh các giải thuật dựa trên bảng tổng hợp sau:

betterEvaluation	MinimaxAgent		AlphaBetaAgent		ExpectimaxAgent	
Layout	Score	game_win	Score	game_win	Score	game_win
trappedClassic	324.4	4	324.4	4	324.4	4
contestClassic	1100.4	3	1100.4	3	1141.0	2
powerClassic	1397.2	3	1397.2	3	2255.0	4
capsuleClassic	-157.2	0	-157.2	0	-150.8	1
mediumClassic	1281.0	4	1474.0	4	1043.8	3
smallClassic	705.2	4	705.2	4	1040.0	5
minimaxClassic	-291.2	1	-291.2	1	-291.2	1
testClassic	557.6	5	562.4	5	355.6	3
trickyClassic	1222.6	2	1222.6	2	983.2	0
originalClassic	1464.8	4	1464.8	4	961.0	2
openClassic	1235.6	5	1235.6	5	1235.6	5

Table 2: Kết quả tổng hợp với hàm lượng giá betterEvaluationFunction

Ta có thể thấy trong các trường hợp thuật toán Minimax và Alpha–Beta Pruning có phần tốt hơn, giải quyết được nhiều layout và số bàn thắng hơn.

Trong 4 bản đồ (contestClassic, powerClassic, capsuleClassic, smallClassic) BetterEvaluationFunction cho Expectimax có điểm số tốt hơn so với Minimax và Alpha–Beta Pruning. Điều này cho thấy Pacman khi áp dụng thuật toán Expectimax có xu hướng di chuyển thông minh và thích ứng tốt hơn với các con Ghost ở trạng thái sợ hãi (chuyển màu trắng và trở thành một điểm thức ăn), tức là Pacman có xu hướng rượt theo ăn các con Ghost màu trắng để tăng mạnh điểm số.

Ta có thể thấy thuật toán Minimax và Alpha–Beta Pruning cho kết quả gần như giống nhau trên mọi Layout trò chơi. Nguyên nhân là do 2 thuật toán này tương tự nhau, nhưng thời gian xử lý của Alpha–Beta Pruning tốt hơn rất nhiều so với Minimax do cơ chế pruning. Ta có thể theo dõi đồ thị dưới đây:

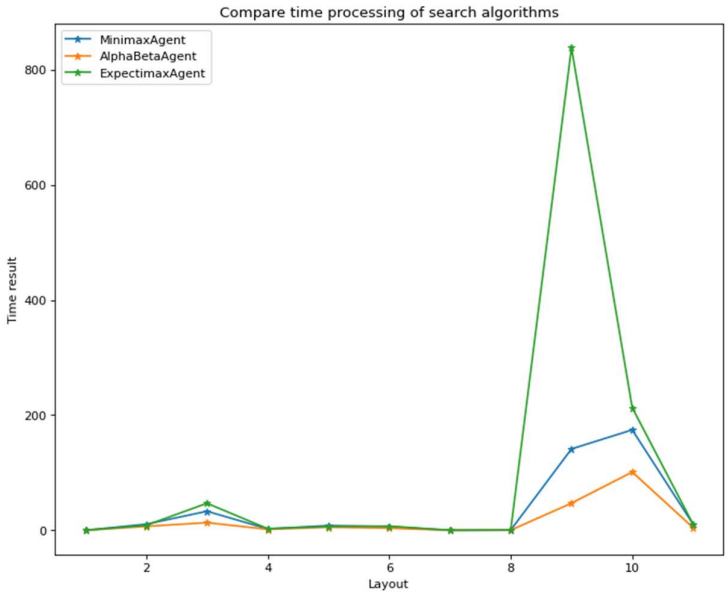

Figure 2: So sánh thời gian xử lý giữa các thuật toán.

3.2 So sánh hàm lượng giá cũ với hàm lượng giá mới

Sau đây là kết quả tổng hợp trên hàm lượng giá **scoreEvaluationFunction**:

scoreEvaluationFunction	MinimaxAgent		AlphaBetaAgent		ExpectimaxAgent	
Layout	**Score**	**game_win**	**Score**	**game_win**	**Score**	**game_win**
trappedClassic	325.2	4	325.2	4	325.2	4
contestClassic	-133.4	0	-133.4	0	126.0	0
powerClassic	228.4	0	228.4	0	22.4	0
capsuleClassic	-468.6	0	-468.6	0	-471.8	0
mediumClassic	85.0	2	85.0	2	-531.6	0
smallClassic	-272.8	0	-272.8	0	8.2	1
minimaxClassic	-291.2	1	-291.2	1	111.6	3
testClassic	0	0	0	0	221.6	4
trickyClassic	686.4	0	686.4	0	361.6	0
originalClassic	-96.4	0	-96.4	0	1.6	0
openClassic	0	0	0	0	-1061.2	0

Table 3: Kết quả tổng hợp với hàm lượng giá scoreEvaluationFunction

Ta có thể quan sát đồ thị dưới đây để có cái nhìn trực quan hơn:

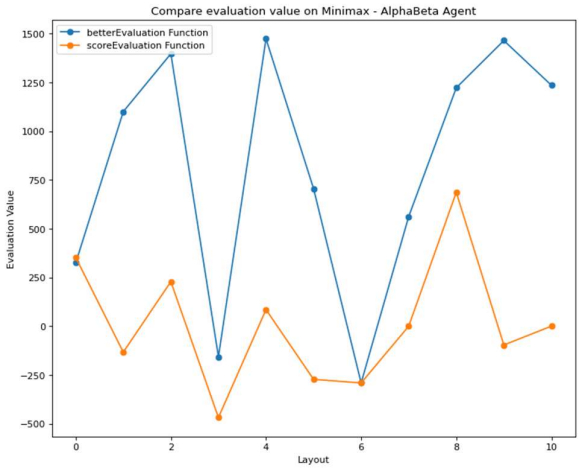

Figure 3: So sánh kết quả trên thuật toán Minimax - AlphaBeta Agent

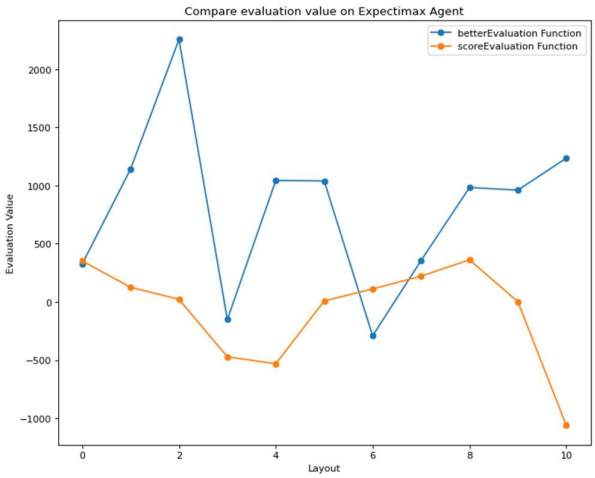

Figure 4: So sánh kết quả trên thuật toán Expectimax Agent

Ta có thể thấy, hàm BetterEvaluationFunction tự xây dựng tuy chưa đạt đến ngưỡng tối ưu nhất, nhưng đã tốt hơn nhiều so với hàm lượng giá được thiết kế sẵn ScoreEvaluationFunction, có nhiều layout mà hàm lượng giá ScoreEvaluationFunction vẫn chưa giải quyết được nhưng với hàm lượng giá mới đã có thể giải quyết khá tốt. Ta vẫn có thể thấy một số hạn chê ở hàm lượng giá BetterEvaluationFunction, như việc chưa đạt được 5/5 ở các lần thử nghiệm và kết quả phân bố khá rộng, chênh lệch điểm số khá nhiều giữa các lần thử nghiệm.

4 Kết luận

Trong các thuật toán, ta có thuật toán Minimax và AlphaBeta tuy có kết quả thực nghiệm gần như là giống nhau, nhưng ở AlphaBeta sẽ cho thời gian xử lí thấp hơn so với Minimax nếu các nhánh có thứ tự sắp xết tốt hơn và độ sâu lớn hơn nhờ vào bước cải tiến Pruning so với minimax. Điều này làm cho AlphaBeta trở thành lựa chọn ưu tiên hơn trong nhiều tình huống thực tế.

Tuy nhiên, trong khi Minimax và AlphaBeta tập trung vào việc tối ưu hóa chi phí thời gian, thuật toán Expectimax tập trung vào việc ước lượng giá trị kỳ vọng của các trạng thái, phù hợp với các trò chơi mà sự không chắc chắn là một phần quan trọng. Mặc dù Expectimax có thể cho kết quả tốt trên nhiều loại bản đồ, nhưng vẫn có những trường hợp không thực sự tốt và thời gian tìm kiếm của nó cũng có thể chậm hơn so với Minimax và AlphaBeta.

Ngoài ra, việc thiết kế hàm lượng giá tốt có thể làm tăng đáng kể kết quả cho các thuật toán tìm kiếm. Tuy nhiên độ sâu càng thấp, giúp tối ưu bộ nhớ thì lại làm cho hàm ước lượng trở nên phức tạp hơn, mất nhiều thời gian để tính toán hơn. Do đó, việc đánh đổi này cũng nên được xem xét kĩ hơn để có được hiệu suất tối ưu nhất.

5 Demo kết quả

Video demo quá trình tìm lời giải của Pacman được đính kèm cùng với file báo cáo.

Link truy cập:

https://drive.google.com/file/d/14cyg0seUHWDpjG8WLUZYACknHk29DWmc/view?usp=sharing