Bài tập 4 - Evaluation functions for Minimax/AlphaBeta/Expectimax

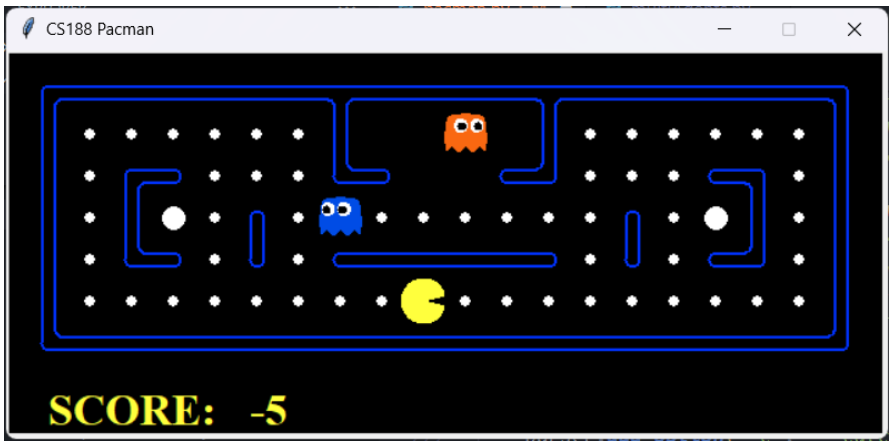
Lý Nguyên Thùy Linh - 22520766

April 2024

Link video clip ván chơi cao điểm nhất

Mục lục

1 pacman.py



Hình 1: Game Pacman

Trong Pacman, người chơi điều khiển một nhân vật hình tròn có tên là Pac-Man (agent), đi qua một mê cung để ăn các viên thức ăn và tránh né các con ma (ghost agents).

Trò chơi sẽ kết thúc nếu rơi vào một trong hai trường hợp sau:

- Chiến thắng: nếu như agent Pacman ăn hết viên thức ăn.
- Thua: nếu agent Pacman bị bắt bởi agent Ghost.

Ngoài ra, bản đồ còn xuất hiện các viên capsule có vai trò tăng sức mạnh Pacman trong một khoảng thời gian nhất định. Khi ăn được viên capsule, Pacman có khả năng tiêu diệt ghost agent và thu về điểm thưởng.

Mục tiêu của trò chơi là điều khiển Pacman ăn hết tất cả các viên thức ăn trong mê cung trong khi tránh né các con ma, và cố gắng đạt được điểm số cao nhất có thể.

Yêu cầu của bài tập 4 là vận dụng các thuật toán multiAgents search và thiết kế hàm lượng giá để đánh giá các giá trị trung gian, trạng thái hành động của Pacman.

2 Các thuật toán Multi - Agent search

2.1 Minimax

Class MinimaxAgent có 3 hàm chính:

- def minimax: hàm duyệt qua tất cả các hành động hợp lệ của Pacman, sau đó gọi hàm minValue để đánh giá giá trị của từng hành động, cuối cùng là chọn hành động có giá trị cao nhất và trả về hành động đó.

```
def minimax(state):
    bestValue, bestAction = None, None
    print(state.getLegalActions(0))
    value = []
    for action in state.getLegalActions(0):
        value = max(value,minValue(state.generateSuccessor(0, action), 1, 1))
        succ  = minValue(state.generateSuccessor(0, action), 1, 1)
        value.append(succ)
        if bestValue is None:
            bestValue = succ
            bestAction = action
        else:
            if succ > bestValue:
                bestValue = succ
                bestAction = action
    print(value)
    return bestAction
```

Hình 2: hàm minimax

- def minValue và def maxValue: lần lượt được sử dụng để đánh giá giá trị tối thiểu và giá trị tối đa của một trạng thái sau khi quan sát đối thủ thực hiện hành động của mình. Sau đấy sử dụng hàm lượng giá **evaluationFunction** để đánh giá trạng thái.

```
def minValue(state, agentIdx, depth):
    if agentIdx == state.getNumAgents():
        return maxValue(state, 0, depth + 1)
    value = None
    for action in state.getLegalActions(agentIdx):
        succ = minValue(state.generateSuccessor(agentIdx, action), agentIdx + 1, depth)
        if value is None:
            value = succ
        else:
            value = min(value, succ)

    if value is not None:
        return value
    else:
        return self.evaluationFunction(state)


def maxValue(state, agentIdx, depth):
    if depth > self.depth:
        return self.evaluationFunction(state)
    value = None
    for action in state.getLegalActions(agentIdx):
        succ = minValue(state.generateSuccessor(agentIdx, action), agentIdx + 1, depth)
        if value is None:
            value = succ
        else:
            value = max(value, succ)

    if value is not None:
        return value
    else:
        return self.evaluationFunction(state)
```

Hình 3: hàm minValue và maxValue

2.2 Alpha - Beta Pruning

Thuật toán Alpha - Beta Pruning là kĩ thuật dùng để giảm số lượng nodes trên cây Minimax bằng cách cắt tỉa bớt các nhánh mà chắc chắn rằng có giá trị score tệ hơn các nhánh đã được lượng giá trước đó. Sự khác biệt giữa minimax và alpha-beta pruning nằm ở giá trị alpha và beta được thêm vào các hàm của alpha-beta pruning hỗ trợ cho việc tỉa nhánh.

```
def minValue(state, agentIdx, depth, alpha, beta):
    if agentIdx == state.getNumAgents():
        return maxValue(state, 0, depth + 1, alpha, beta)
    value = None
    for action in state.getLegalActions(agentIdx):
        succ = minValue(state.generateSuccessor(agentIdx, action), agentIdx + 1, depth, alpha, beta)
        if value is None:
            value = succ
        else:
            value = min(value, succ)
        if value <= alpha: return value
        beta = min(beta, value)

    if value is not None:
        return value
    else:
        return self.evaluationFunction(state)


def maxValue(state, agentIdx, depth, alpha, beta):
    if depth > self.depth:
        return self.evaluationFunction(state)
    value = None
    for action in state.getLegalActions(agentIdx):
        succ = minValue(state.generateSuccessor(agentIdx, action), agentIdx + 1, depth, alpha, beta)
        if value is None:
            value = succ
        else:
            value = max(value, succ)
        if value >= beta: return value
        alpha = max(alpha, value)

    if value is not None:
        return value
    else:
        return self.evaluationFunction(state)
```

Hình 4: Alpha-Beta Pruning

- alpha tương ứng với giá trị Max tốt nhất, beta tương ứng với giá trị Min tốt nhất
- nếu $value \leq alpha$ hoặc $value \geq beta$, khi đó node cha (MAX hoặc MIN) của các node value sẽ không bao chờ chọn node này, vì thế các hàm sẽ trả về giá trị value, tỉa hết toàn bộ các cây con của node này.

2.3 Expectimax

Thuật toán Expectimax cũng tương tự với Minimax, thay đổi ở hàm minValue thành expValue, tức thay vì các agent ghost luôn chọn giá trị thấp nhất đối với agent Pacman, thì bây giờ các agent ghost lựa chọn hành động có yếu tố ngẫu nhiên (do agent Pacman không biết được hành động của agent ghost).

```
def expValue(state, agentIdx, depth):
    if agentIdx == state.getNumAgents():
        return maxValue(state, 0, depth + 1)
    value = 0
    legalActions = state.getLegalActions(agentIdx)
    numActions = len(legalActions)
    for action in legalActions:
        succ = expValue(state.generateSuccessor(agentIdx, action), agentIdx + 1, depth)
        value += succ / numActions
    return value
```

Hình 5: expValue

3 Hàm lượng giá

3.1 Evaluation Function

Hàm lượng giá default trong file *multiAgents.py* chỉ đơn giảm là trả về giá trị score của trạng thái.

```
def scoreEvaluationFunction(currentGameState):
    """
    This default evaluation function just returns the score of the state.
    The score is the same one displayed in the Pacman GUI.

    This evaluation function is meant for use with adversarial search agents
    (not reflex agents).
    """
    return currentGameState.getScore()
```

Hình 6: scoreEvaluationFunction

3.2 betterEvaluationFunction

Dựa trên những ý tưởng có sẵn của betterEvaluationFunction1 có sẵn trong file *multiAgents.py*, hàm betterEvaluationFunction2 sẽ đánh giá trạng thái dựa trên các yếu tố sau:

- Khoảng cách Mahattan giữa agent ghost gần nhất và agent Pacman (1)

  Để chắc chắn về khả năng sống sót của Pacman, khoảng cách (1) phải càng xa càng tốt.

- Khoảng cách Mahattan giữa agent Pacman và viên thức ăn gần nhất (2)

  Để thỏa mãn điều kiện chiến thắng (ăn hết thức ăn), khoảng cách (2) phải càng gần càng tốt.

- Khoảng cách Mahattan giữa agent Pacman và viên capsule gần nhất (3)

  Khoảng cách này chiếm trọng số không cao vì việc ăn được viên capsule không thực sự cần thiết (không nằm trong điều kiện chiến thắng). Tuy nhiên, vì ăn capsule và tiêu diệt được ghost sẽ cộng khá nhiều điểm, nên betterEvaluationFunction2 có thêm thuộc tính scaredTime (4) (thời gian ghost đang trong trạng thái *sợ hãi* sau khi Pacman ăn được viên Capsule).

- Sau khi Pacman ăn được viên capsule, khoảng cách (1) sẽ đổi dấu, tức khoảng cách giữa Pacman và Ghost càng gần càng tốt, Pacman sẽ có xu hướng *rượt* ghost để tiêu diệt chúng.

  Trọng số của thuộc tính này sẽ phụ thuộc vào thời gian scaredTime của agent ghost, tức nếu thời gian scaredTime còn càng nhiều, trọng số càng lớn.

  Tuy nhiên, trọng số của thuộc tính scaredTime vẫn không nên cao hơn (2) vì (2) mới là điều kiện chiến thắng, nếu trọng số của (4) quá cao, sẽ có thể khiến Pacman chỉ chăm lo rượt đuổi các agent ghost và *quên mất* điều kiện chiến thắng, mà điểm lại giảm dần theo thời gian của màn chơi, vì thế có khả năng số điểm không những không cải thiện mà còn bị giảm.

```python
def betterEvaluationFunction2(currentGameState):
    """
    Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
    evaluation function (question 5).
    """
    "*** YOUR CODE HERE ***"
    #util.raiseNotDefined()
    newPos = currentGameState.getPacmanPosition()
    newFood = currentGameState.getFood()
    foodList = newFood.asList()
    newGhostStates = currentGameState.getGhostStates()
    newCapsules = currentGameState.getCapsules()
    newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]
    closestGhostDis = float("inf")
    closestFoodDis = float("inf")
    closestCapDis  = float("inf")

    for ghost in newGhostStates:
        dis = manhattanDistance(newPos, ghost.getPosition())
        if closestGhostDis > dis:
            closestGhost = ghost
            closestGhostDis = dis

    for food in foodList:
        dis = manhattanDistance(newPos, food)
        if dis < closestFoodDis:
            closestFoodDis = dis

    if newCapsules:
        for caps in newCapsules:
            dis = manhattanDistance(newPos, caps)
            if dis < closestCapDis:
                closestCapsule = dis
    else:
        closestCapsule = 0
```

```python
if closestCapsule:
    closest_capsule = -2 / closestCapsule
else:
    closest_capsule = 100

if closestGhost.scaredTimer > 0:
    # print(f"scared time closest-dis: {closestGhost.scaredTimer}, {closestGhostDis} ")
    ghost_distance =  closestGhostDis #càng gần scared ghost điểm càng cao
    ghost_distance *= (closestGhost.scaredTimer/10)  # số điểm biến thiên theo thời gian scared của ghost
    foodScores = 10 / closestFoodDis
else:
    if closestGhostDis:
        ghost_distance = -2 / closestGhostDis
    else:
        ghost_distance = -500
    if closestFoodDis:
        foodScores = 3 / closestFoodDis # khoang cach giua food gan nhat và ghost gần nhất càng xa nhau thì càng ưu tiên ăn food
    else: # ngược lại thì ưu tiên né ghost và move to another food
        foodScores= 0


score =  foodScores + ghost_distance -10  * len(foodList) + closest_capsule

# print(newScaredTimes, score, sep=" ")
# return -2 * closestFood + ghost_distance - 10 * len(foodList) + closest_capsule
# print(score)
return score
```

Hình 7: betterEvaluationFunction2

4 Thực nghiệm

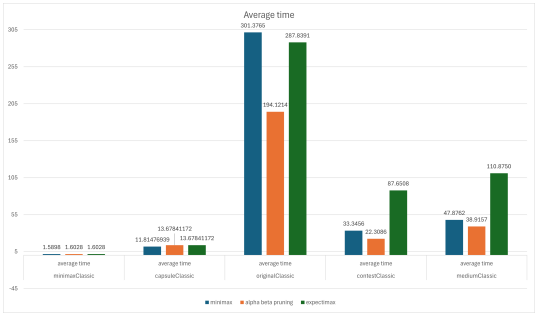
Thực hiện chơi Pacman trên 5 map khác nhau, mỗi map chơi 5 lần với 5 random seed (mssv + [0, 4]), mỗi lần chơi với 3 thuật toán multi - agent search và 2 hàm lượng giá.

Kết quả thu được thể hiện bảng dưới:

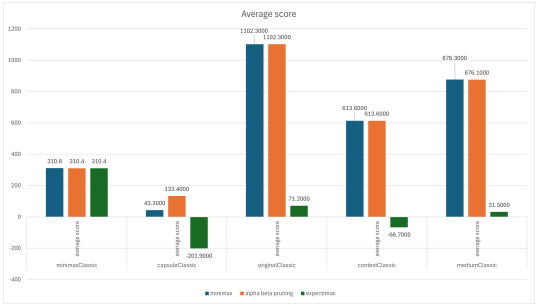
random seed, function / layouts, algorithms			minimaxClassic			capsuleClassic			originalClassic			contestClassic			mediumClassic		
			minimax	alpha beta pruning	expectimax	minimax	alpha beta pruning	expectimax	minimax	alpha beta pruning	expectimax	minimax	alpha beta pruning	expectimax	minimax	alpha beta pruning	expectimax
22520766	evaluation function	res	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss
		scores	-492.0000	-492.0000	-498.0000	-692.0000	-692.0000	-503.0000	772.0000	772.0000	777.0000	221.0000	221.0000	-46.0000	-474.0000	-474.0000	419.0000
		time (s)	1.3024	1.9020	1.8670	38.2131	27.4267	11.6116	377.1561	224.9930	848.6878	26.6853	16.3153	60.8032	97.4180	81.0572	67.7909
	better evaluation function	res	Win	Win	Loss	Win	Win	Loss	Win	Win	Loss	Win	Win	Loss	Win	Win	Loss
		scores	507.0000	507.0000	-506.0000	1390.0000	1390.0000	-466.0000	2308.0000	2308.0000	1021.0000	1357.0000	1357.0000	-375.0000	1680.0000	1680.0000	524.0000
		time (s)	1.8716	1.8562	2.0953	15.8212	12.4778	1.6659	263.8606	185.8798	124.7496	35.7401	26.4853	12.2686	18.3597	15.2321	9.7653
22520767	evaluation function	res	Win	Win	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss
		scores	512.0000	512.0000	-562.0000	-562.0000	339.0000	187.0000	-637.0000	-637.0000	-616.0000	-55.0000	-55.0000	189.0000	189.0000	187.0000	-616.0000
		time (s)	1.7144	1.5260	18.1650	11.8240	51.0926	36.2254	579.4810	370.8151	566.9828	19.7089	11.1848	29.7493	25.0154	36.2254	566.9828
	better evaluation function	res	Win	Win	Loss	Win	Win	Loss	Win	Win	Loss	Loss	Loss	Loss	Win	Win	Loss
		scores	515.0000	515.0000	-502.0000	1211.0000	1211.0000	-487.0000	2168.0000	2168.0000	69.0000	373.0000	373.0000	-451.0000	1454.0000	1454.0000	167.0000
		time (s)	1.4764	1.5199	1.8551	11.2963	10.2119	1.8159	218.8192	158.4242	67.6168	28.4760	22.2567	12.8846	17.9677	15.2101	10.6744
22520768	evaluation function	res	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Win	Win	Loss
		scores	-495.0000	-495.0000	-500.0000	-443.0000	-443.0000	421.0000	-11.0000	-11.0000	-11.0000	851.0000	851.0000	903.0000	1664.0000	1664.0000	-272.0000
		time (s)	1.4338	1.3673	1.8341	4.7060	4.1735	65.6129	106.7290	75.0458	188.0703	38.2828	26.1304	350.6567	50.6315	39.8994	190.4369
	better evaluation function	res	Win	Win	Loss	Win	Win	Loss	Win	Win	Loss	Win	Win	Loss	Win	Win	Loss
		scores	513.0000	513.0000	-497.0000	995.0000	995.0000	-466.0000	2208.0000	2208.0000	-267.0000	1194.0000	1194.0000	-430.0000	1515.0000	1515.0000	121.0000
		time (s)	1.5326	1.5355	1.6693	11.7227	10.5544	1.6981	194.2561	135.3815	29.1412	30.5046	22.6239	9.2500	16.2036	12.7264	12.3643
22520769	evaluation function	res	Win	Win	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Win	Win	Loss	Win	Win	Loss
		scores	513.0000	513.0000	-499.0000	-444.0000	-444.0000	-165.0000	-156.0000	-156.0000	-157.0000	1483.0000	1483.0000	95.0000	1422.0000	1422.0000	110.0000
		time (s)	1.5347	1.4606	1.7979	4.9837	4.1309	50.5489	167.8981	116.1097	262.4618	49.9152	31.8422	191.9404	44.8550	37.6228	97.2303
	better evaluation function	res	Win	Win	Loss	Loss	Loss	Loss	Win	**Win**	Loss	Loss	Loss	Loss	Win	Win	Loss
		scores	513.0000	509.0000	-510.0000	-274.0000	-274.0000	-459.0000	2324.0000	**2324.0000**	-131.0000	157.0000	157.0000	-278.0000	1303.0000	1303.0000	327.0000
		time (s)	1.7257	1.6922	2.3069	5.8618	5.0318	6.4119	268.5902	**177.2981**	70.5132	41.7847	28.8629	26.2568	14.3826	12.3652	12.6414
22520770	evaluation function	res	Win	Win	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss	Loss
		scores	513.0000	513.0000	-499.0000	-469.0000	-469.0000	398.0000	166.0000	166.0000	165.0000	-33.0000	-33.0000	103.0000	-1472.0000	-1472.0000	-578.0000
		time (s)	1.5258	1.4296	1.6303	6.4073	5.5767	18.4843	416.6158	237.1112	598.8184	26.7132	15.1329	166.2996	173.5691	122.7820	129.0811
	better evaluation function	res	Win	Win	Loss	Loss	Loss	Loss	Win	Win	Loss	Loss	Loss	Loss	Win	Win	Loss
		scores	509.0000	509.0000	-501.0000	-279.0000	-279.0000	-479.0000	1881.0000	1881.0000	-138.0000	588.0000	588.0000	-377.0000	1482.0000	1482.0000	113.0000
		time (s)	1.7802	1.7388	1.7881	7.3114	6.1078	2.0438	420.3587	260.1560	121.3494	35.6448	22.2517	16.3990	20.3596	16.0368	11.7823

Hình 8: Kết quả thực nghiệm

4.1 Nhận xét các thuật toán Multi-Agent Search



(a) Thời gian trung bình qua các map của 3 thuật toán



(b) Điểm trung bình qua các map của 3 thuật toán

		minimax	alpha beta pruning	expectimax
minimaxClassic	res	8 Wins / 10	8 Wins / 10	0 Wins / 10
	average time	1.5898	1.6028	1.6028
	average score	310.8	310.4	310.4
capsuleClassic	res	3 Wins / 10	3 Wins / 10	3 Wins / 10
	average time	11.81476939	13.67841172	13.67841172
	average score	43.3000	133.4000	-201.9000
originalClassic	res	5 Wins / 10	5 Wins / 10	0 Wins / 10
	average time	301.3765	194.1214	287.8391
	average score	1102.3000	1102.3000	71.2000
contestClassic	res	3 Wins / 10	3 Wins / 10	0 Wins / 10
	average time	33.3456	22.3086	87.6508
	average score	613.6000	613.6000	-66.7000
mediumClassic	res	7 Wins / 10	7 Wins / 10	0 Wins / 10
	average time	47.8762	38.9157	110.8750
	average score	876.3000	876.1000	31.5000

(c) Bảng so sánh 3 thuật toán

Hình 9: So sánh 3 thuật toán Multi-Agent Search

Qua các biểu đồ trong 9, ta thấy rằng trong hầu hết các trường hợp, thuật toán alpha-beta pruning thường có thời gian chạy nhanh nhất nhờ vào khả năng cắt tỉa nhánh thừa của nó.

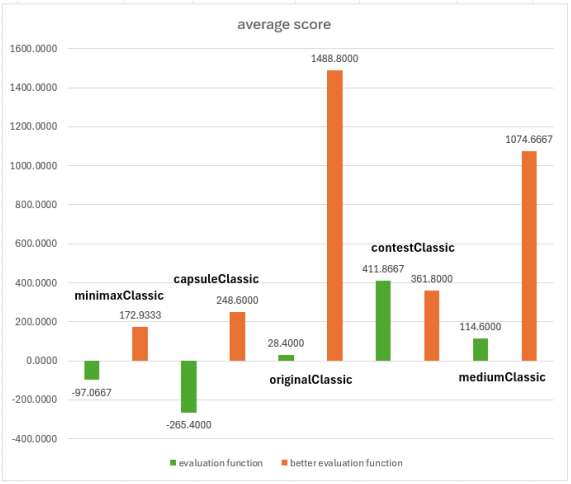
Tuy nhiên, trên map capsuleClassic, thời gian chạy của alpha-beta pruning lại cao hơn so với hai thuật toán còn lại một chút. Điều này có thể do kết quả của cây trò chơi thường rơi vào node cuối cùng, khiến cho alpha-beta pruning không thể cắt tỉa nhánh nào, đồng thời phải mất thêm thời gian

tính toán logic của các tham số alpha, beta. Điều này dẫn đến việc thời gian chạy của nó trở nên lâu hơn so với hai thuật toán còn lại.

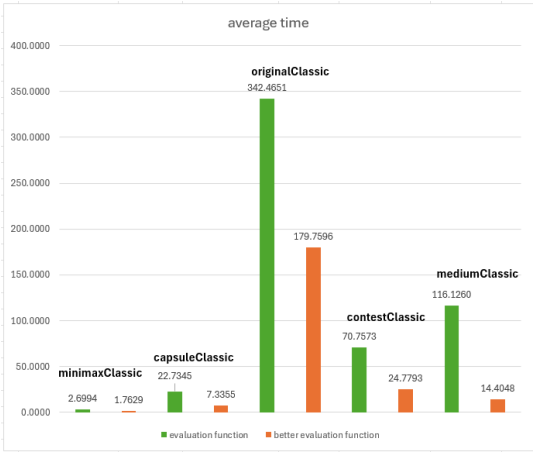
Về điểm số, chúng ta biết rằng thuật toán minimax luôn đưa ra lời giải tối ưu (nếu chạy đủ lâu). Do đó, trong hầu hết các trường hợp, điểm số của minimax thường cao hơn so với alpha-beta pruning. Tuy nhiên, trên map capsuleClassic, điểm số của alpha-beta pruning lại cao hơn. Điều này có thể được giải thích bởi sự đa dạng của map này, với nhiều viên capsules, từ đó dẫn đến việc thuật toán alpha-beta pruning có thể tận dụng hàm betterEvaluation2 để lấy được nhiều điểm hơn.

Về thời gian và điểm số, thuật toán expectimax thường thể hiện kém hơn do ảnh hưởng của yếu tố ngẫu nhiên, khiến cho Pacman gặp khó khăn trong việc dự đoán trạng thái tiếp theo.

4.2 Nhận xét hai hàm lượng giá: scoreEvaluationFunction và betterEvaluationFunction2



(a) Thời gian trung bình qua các map của 2 hàm lượng giá



(b) Điểm trung bình qua các map của 2 hàm lượng giá

	minimaxClassic		capsuleClassic		originalClassic		contestClassic		mediumClassic	
	evaluation function	better evaluation function	evaluation function	better evaluation function	evaluation function	better evaluation function	evaluation function	better evaluation function	evaluation function	better evaluation function
res	6 Wins / 15	10 Wins / 15	0 Wins / 15	6 Wins / 15	0 Wins / 15	10 Wins / 15	1 Wins / 15	2 Wins / 15	4 Wins / 15	10 Wins / 15
average score	-97.0667	172.9333	-265.4000	248.6000	28.4000	1488.8000	411.8667	361.8000	114.6000	1074.6667
average time	2.6994	1.7629	22.7345	7.3355	342.4651	179.7596	70.7573	24.7793	116.1260	14.4048

(c) Bảng so sánh hàm lượng giá

Hình 10: So sánh 2 hàm lượng giá

Dựa vào các biểu đồ và bảng số liệu từ 10, có thể thấy rằng trong phần lớn các trường hợp, hàm betterEvaluationFunction2 cho kết quả tốt hơn rất rõ rệt so với scoreEvaluationFunction. Trong 15 trận đấu, trung bình số trận thắng của betterEvaluationFunction2 là 7.6, trong khi đó của scoreEvaluationFunction chỉ là 2.2. Không chỉ vậy, thời gian chạy trung bình và điểm trung bình của betterEvaluationFunction2 cũng có hiệu năng cao hơn nhiều so với scoreEvaluationFunction, đặc biệt là trên bản đồ originalClassic. Trên bản đồ này, điểm trung bình của betterEvaluationFunction2 là 1488.8 so với chỉ 28.4 của scoreEvaluationFunction. Trong khi đó, ở bản đồ mediumClassic, thời gian chạy trung bình của betterEvaluationFunction2 chỉ là 14.4 giây, trong khi scoreEvaluationFunction lại cần tới 116.126 giây để hoàn thành.

4.3 Những điểm chưa khắc phục được

Với ý tưởng ban đầu là việc Pacman có khả năng *rượt đuổi* Pacman khi ghost agents đang trong thời gian scaredTime, và trọng số của thuộc tính này sẽ phụ thuộc vào scaredTime còn lại của ghost agents. Tuy nhiên, khi thực nghiệm thì Pacman agent ở hàm betterEvaluationFunction2 chỉ có khả năng "không né tránh scared ghost"chứ chưa thể rượt đuổi tiêu diệt scared ghost. Điểm chỉ có thể cao

nếu scared ghost vô tình chạy tới Pacman và Pacman *thụ động* tiêu diệt chúng. Điều này có thể được giải thích do trọng số chưa được khởi tạo đúng cách.

Bên cạnh đó, do Pacman chỉ có thể nhìn được trước 3 depth (**depth=3**), nên trong vài trường hợp, dù vẫn còn viên thức ăn trên map nhưng Pacman vẫn bất động. Điều này có thể được cải thiện bằng cách điều chỉnh trọng số của thuộc tính 3.2 cao lên, kéo khoảng cách 3.2 ngày càng gần càng tốt. Một cách khác là tăng giá trị tham số depth > 3 giúp Pacman có thể nhìn trước được nhiều bước đi hơn. Hoặc có thể tạo thêm nhiều thuộc tính khác, giúp phong phú hơn hàm lượng giá.