Artificial Intelligence based Pacman Game

Himakar Sai Chowdary Maddipati, Aravind Kundurthi, Pothula Mahima Raaj, Kapudasi Srilatha, Ravikishan Surapaneni

***Abstract:** This paper is about implementing pacman game with AI.The Game Pac-Man is a very challenging video game that can be useful in conducting AI(Artificial Intelligence) research. Here,the reason we have implemented various AI algorithms for pacman game is that it helps us to study AI by using visualizations through which we can understand AI more ef- fectively.The main aim is to build an intelligent pacman agent which finds optimal paths through the maze to find a particular goal such as a particular food position,escaping from ghosts.For that, we have implemented AI search algorithms like Depth first search,Breadth first search,A*search,Uniform cost search.We have also implemented multi-agents like Reflex agent,Minimax agent,Alpha-beta agent.Through these multiagent algorithms,we can make pacman to react from its environmental conditions and escape from ghosts to get high score.We have also done the visualization part of the above AI algorithms by which anyone can learn and understand AI algorithms easily.For visualisation of algorithms,we have used python libraries matplotlib and Networkx.*

***Keywords** : Artificial Intelligence,search algorithms,multi agents,pacman.*

I. INTRODUCTION

Pac-man is one of the most popular arcade games in the world. The player controls Pac-Man, who must eat all the dots inside an enclosed maze while avoid- ing four colored ghosts. Eating large flashing dots called power pellets causes the ghosts to turn blue, allowing Pac-Man to eat them for bonus points.Here,the main aim is to score points by collecting food from maze and escaping from the ghosts which are roaming around the maze.If the ghost captures the pacman,then the game is over. So,first we will design an intelligent pacman agent which will find optimal paths through the maze to reach the goal state,eating all dots,escaping from ghosts in minimum no.of steps. To design this intelligent agent, we implemented several search algorithms.

The search algorithms are categorized into 2 types: Uninformed search algorithms(In this we will cover DFS,BFS ,UCS),Informed search algorithms(In this, we will implement A* search).The main difference between uninformed and informed is that the uninformed search algorithms are not given any information about the problem,whereas informed search algorithms are given information about the problem.

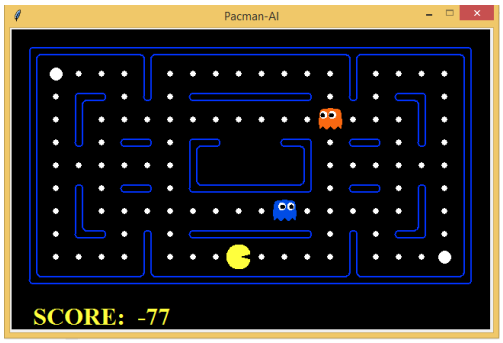

Fig. 1. Interface of Pacman game.

we will compare the performance of above stated algorithms for pacman to find the optimal path. We have also provided visualisation for the above said algorithms,which helps us in understanding the working of algorithms clearly.For visualisation we have used matplotlib,NetworkX libraries of python.We have also implemented multiagent algorithms like Reflex agent,Minimax-agent,Alpha-beta agent. By using these algorithms,the pacman agent will try to escape from the ghost agents and eat all the food in the maze to win the game.

II. RELATED WORK

Sebastian Thrun implemented artificial intelligence for chess game by designing NeuroChess program which learns how to play chess game from the final outcome of all the games by itself. NeuroChess learns various board evaluation functions,which are implemented using artificial neural networks. César Villacís, Walter Fuertes, Mónica Santillán, Hernán Aules, Ana Tacuri,Margarita Zambrano and Edgar Salguero proposed a case study of an optimized Tic-Tac-Toe.They have implemented AI for a video game known as Tic-Tac-Toe. The model was implemented by using several AI algorithms and a graphical UI including Semiotics; this provided an attractive and nice environment. Its main purpose is stimulating cognitive development of children. Veenus Chhabra and Kuldeep Tomar implemented artificial intelligence for ludo game. Ludo is a game that is played by two,three or four players.In this game,the players race against their four tokens from initial to end state according to rolling of a dice.They have implemented ludo using Q-learning which is a type of reinforcement learning.

Revised Manuscript Received on June 25, 2020.

***** Correspondence Author

Himakar Sai Chowdary Maddipati* ,Department of Computer Science and Engineering,,VRSiddhartha Engineering College, Kanuru, India. Email: maddipati.himakar@gmail.com

Aravind Kundurthi ,Department of Computer Science and Engineering,,VRSiddhartha Engineering College, Kanuru, India. Email: aravindkundurthi005@gmail.com

Pothula Mahima Raaj , Department of Computer Science and Engineering College,VR Siddhartha Engineering College ,Kanuru, India. Email: mahi.raaj1999@gmail.com

Kapudasi Srilatha , Department of Computer Science and Engineering ,VR Siddhartha Engineering College,Kanuru,India.Email: ksrilatha2222@gmail.com

Ravikishan Surapaneni , Department of Computer Science and Engineering, VR Siddhartha Engoineering College, Kanuru, India.Email: suraki@vrsiddhartha.ac.in

Sergey Karakovskiy and Julian Togelius implemented Mario AI benchmark,benchmark for reinforcement learning algo- rithms and game AI techniques.The benchmark is for classic game Super Mario Bros, and completely open source.The benchmark was used in various competitions ,international conferences, and students from various parts of the world implemented various different solutions to beat the benchmark.

III. SEARCH ALGORITHMS

Uninformed search is a class of search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree.It is also called blind search.

A. Depth First Search

Depth-first search always tries to expand the deepest node in the stack of the search tree. Depth-first search algorithm is a search algorithm which uses a stack data Structure(LIFO data structure). Through stack, the most recently generated node is chosen for expansion.

Algorithm for DFS:

/*

function dfs(problem) returns solution or failure
initialize the stack using the initial state of the problem
initialize the visited list to null
loop do
if the stack is empty then return failure
choose a leaf node and remove it from the stack
if node contains a goal state then return the node as a solution
add the node to the visited list
expand chosen node and add the resulting nodes to the stack only if they are not in the visited list

*/

The results of running depth first search algorithm are shown in the above table.

TABLE IRESULTS OF DFS

Maze	Cost	Nodes Expanded	Score
tinyMaze	10	14	500
mediumMaze	130	144	380
bigMaze	210	390	300

*s-seconds.

But DFS does not provide us the best solution as the solutions shown above are not least cost solutions.The visual-ization of DFS algorithm is shown in the Fig.2 below.Python libraries like NetworkX,Matplotlib are used to create the graphs of the nodes that are expanded during the search process.Also,visualization of stack is shown.

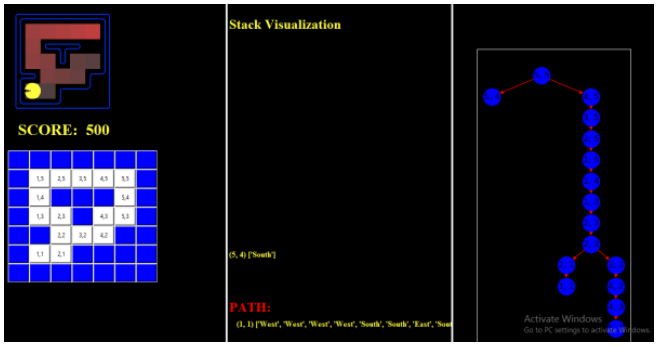

Fig. 2. Visualization of DFS

B. *Breadth First Search*

In Breadth-first search algorithm, the root node is expanded first and then all the children of the root node are expanded,then their successors, and so on.Here all nodes are expanded level by level,which means that all the nodes of a particular level will be expanded before going to the next level .BFS uses a queue (FIFO data structure) for the frontier.

Breadth First Search returns a least cost solution in terms of effort taken by Pacman to reach the food dot and results are shown in Table II. But here, the expanded nodes are relatively large(269 and 620 for mediumMaze and bigMaze respectively), which consumes a huge amount of time to find the best solution.

TABLE II RESULTS OF BFS

Maze	Cost	Nodes Expanded	Score
tinyMaze	8	68	502
mediumMaze	68	268	442
bigMaze	210	618	300

*s-seconds.

The visualization of BFS algorithm is shown in the Fig.3 below.Python libraries like NetworkX,Matplotlib are used to create the graphs of the nodes that are expanded during the search process.Also,visualization of queue is shown.

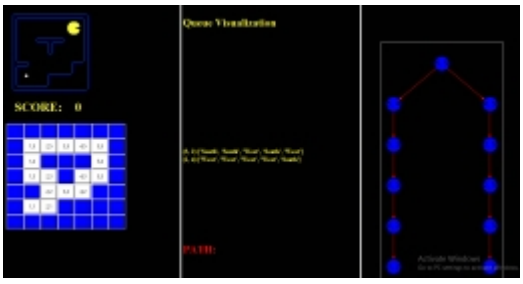

Fig. 3. Visualization of BFS

C. Uniform Cost Search

The main property of Uniform Cost Search algorithm is,instead of expanding the deepest node, it tries to expand the node having the least path cost. This is done by using the priority queue data structure in which the items are ordered by cost.

The priority queue stores items in the form of(c1,c2,i) where c1 is the cost associated with the node,c2 is the count specifying the number of items in the priority queue and i is the item denoting the state and actions associated with the node.Nodes are arranged in the ascending order of their costs and in each iteration,the node with minimum cost is removed out of the queue and its successor nodes are expanded.The cost of each node is calculated based on the distance between the current state and the initial state.

We have run the UCS algorithm for tinymaze,mediummaze and bigmaze.The results are as shown below.

TABLE III RESULTS OF UCS

Maze	Cost	Nodes Expanded	Score
tinyMaze	8	15	502
mediumMaze	68	269	442
bigMaze	210	620	300

*s-seconds.

From the results ,we can observe that when compared to DFS and BFS,UCS takes more time and cost to expand the nodes and find the goal state.When compared to BFS, the number of nodes expanded are slightly more,the cost and score remains same but the time taken by UCS is more.

The visualization of UCS algorithm is also implemented using NetworkX and Matplotlib python libraries,which is shown in the diagram below.

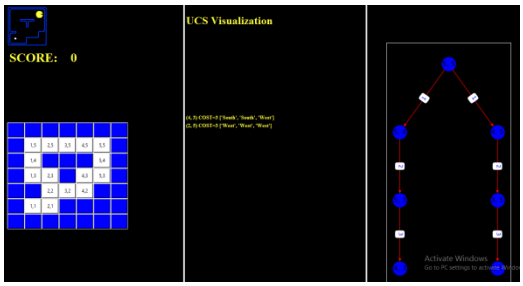

Fig. 4. Visualisation of Uniform Cost Search.

D. A* Search

A* search is a type of informed search algorithm which knows information about the problem beforehand. It evaluates nodes by adding x(n), the cost to reach the node, and y(n), the cost to get from the node to the goal: f (n) = x(n) + y(n). As x(n) is the path cost from the start node to node n, and y(n) is the cost of the optimal path from n to the goal, we have f(n) =total cost of the cheapest solution through n.

So, if we should find the cheapest solution,we should consider the node with the lowest value of f(n).

The only difference between A* search and UCS is UCS considers the cost of reaching the node whereas A* Search considersthe sum of cost of reaching the node and cost of reaching the goal node from that node(x+y) instead of x.Here,the heuristic function used is Manhattan distance heuristic.This heuristic is used to find which node or state is closer to the goal state.

We have also compared the performance of A* Search and UCS algorithms on tinymaze,mediummaze and big maze and shown them below in the figure.

TABLE IV UCS AND A* SEARCH COMPARISON

Algorithm	Maze	Nodes Expanded	Cost	Time
UCS	bigMaze	620	210	7.3s
A*	bigMaze	549	210	0.6s
UCS	mediumMaze	269	68	1.6s
A*	mediumMaze	221	68	0.1s

s-seconds.

Out of these,A* Search is the most efficient one in terms of nodes expanded and also cost in finding the goal state.

For implementing A* Search in python,we have used the priority queue data structure same as in UCS algorithm.But the main difference here is,in A* Search we have used Manhattan Heuristic.The visualization of A* Search algorithm is also implemented using NetworkX and Matplotlib python libraries,which is shown in the diagram below.

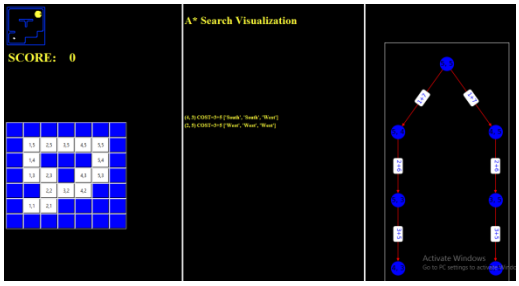

Fig. 5. Visualisation of A* Search.

IV. MULTIAGENTS

A. Reflex Agent

We create here a reflex agent which chooses at its turn a random action from the legal ones.Note that this is different from the random search agent, since a reflex agent does not build a sequence of actions, but chooses one action and executes it. The random reflex agent appears in listing below:
class RandomAgent(Agent):

/*This gets the action to be performed using this agent*/ def getAction (self , gameState):

/*Possible directions in which pacman can move */ legalMoves = gameState . getLegalActions ()

/* Pick randomly among the legal */ chosenIndex=random.choice(range(0,len(legalMoves))) return legalMoves [chosenIndex]

This reflex agent chooses its current action based only on its current perception. The Reflex Agent gets all its legal actions, computes the scores of the states reachable with these actions and selects the states that results into the state with the maximum score. In case more states have the maximum score, it will choose randomly one The Reflex agent algorithm has worked successfully for all the three mazes(testClassic,openClassic,mediumClassic).The resultant scores for various mazes are as shown below:

TABLE V RESULTS OF REFLEX AGENT

Maze Form	Ghosts	Pellets count	Score
testClassic	1	8	562
openClassic	1	86	1247
mediumClassic	2	98	1253

B. Minimax Agent

In case the world where the agent plans ahead includes other agents which plan against it, adversarial search can be used. One agent is called MAX and the other one MIN. Utility(s; p) (called also payo function or objective function) gives the nal numeric value for a game that ends in terminal state s for player p. For example, in chess the values can be +1, 0, ½. The game tree is a tree where the nodes are game states and the edges are moves. Max's actions are added first. Then, for each resulting state, the action of Min's are added, and so on. A game tree can be seen in below figure.

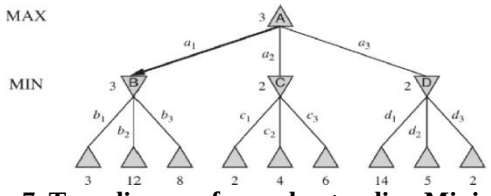

Fig. 7. Tree diagram for understanding Minimax algorithm

Optimal decisions in games must give the best move for MAX in the initial state, then MAX's moves in all the states resulting from each possible response by MIN, and so on.Minimax value ensures optimal strategy for MAX. The algorithm for computing this value is given in below listings. For the agent in Fig.7,the best move is a1 and the minimax value of the game is 3.

MINIMAX(s) =
=UTILITY(s) if TERMINAL-TEST(s)
=max(a ∈ACTIONS)(s)
MINIMAX(RESULT(s,a)) if PLAYER(s)=
MAX
=min(a ∈ACTIONS)(s)
MINIMAX(RESULT(s,a)) if
PLAYER(s)=MIN

The following are the functions that provide a brief overview about MINIMAX algorithm:

```
function MINIMAX-DECISION(state) returns an action
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v = -INFINITY
  for each a in ACTIONS( state ) do
    v = MAX(v , MIN-VALUE(RESULT( state , a ) ) )
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return
  UTILITY(state) v = -INFINITY
  for each a in ACTIONS( state ) do
    v = MIN(v ,MAX-VALUE(RESULT(state,a)))
  return v
```

Result(state; a) is the state which results from the application of action a in state. Minimax algorithm generates the entire game search space. Imperfect real-time decisions involve the use of cuto-test based on limiting the depth for the search. When the CUTOFF test is met, the tree leaves are evaluated using an heuristic evaluation function instead of the utility function.

H-MINIMAX(s,d) =
=EVAL(s) if CUTOFF-TEST(s, d)
=max(a ∈Actions)(s) H-MINIMAX(RESULT(s,a), d+1) if
PLAYER(s)= MAX
=min(a ∈Actions)(s) H-MINIMAX(RESULT(s,a), d+1) if
PLAYER(s)=MIN

The Minimax algorithm has worked successfully for all the mazes(testClassic,openClassic,mediumClassic).The scores for various mazes are as shown below:

TABLE VI RESULTS OF MINIMAX AGENT

Maze Form	Ghosts	Pellets count	Score
testClassic	1	8	558
openClassic	1	86	1088
mediumClassic	2	98	1660

C. Alpha-beta pruning

In order to limit the number of game states from the game tree, alpha-beta prunning can be applied, where
a = the value of the best (highest value) choice there is so far at any choice point along the path for MAX
b = the value of the best (lowest-value) choice there is so far at any choice point along the path for MIN
The following are the functions that provide a brief overview about ALPHA BETA pruning procedure:

```
function ALPHA-BETA-SEARCH(state) returns an
  action v = MAX-VALUE( state , -INF,+INF)
  return the action in ACTIONS( state ) with value v
function MAX-VALUE(state,α,β) returns a utility
  value if TERMINAL-TEST(state) then return
  UTILITY(state) v=-INF
  for each i in ACTIONS( state ) do
    v=MAX(v,MIN-VALUE(RESULT(s,i),
    α,β))
    if v >= β then return v
    α=MAX
  (α,v) return
  v
function MIN-VALUE(state,α,β) returns a utility value
  if TERMINAL-TEST(state) then return
  UTILITY(state) v=-INF
  for each i in ACTIONS( state ) do
    v=MIN(v,MAX-VALUE(RESULT(s,i),
    α,β))
    if v <=α then return v
    β=MIN(
  β,v) return
  v
```

The Alpha-Beta pruning algorithm has worked successfully for all the three mazes(testClassic,openClassic,mediumClassic).The statistics for various mazes are as shown below:

TABLE VII RESULTS OF ALPHA-BETA PRUNING

Maze Form	Ghosts	Pellets count	Score
testClassic	1	8	548
openClassic	1	86	907
mediumClassic	2	98	1470

V. CONCLUSION

We have successfully implemented search algorithms like Depth First Search,Breadth first search,Uniform Cost Search,A* Search and also implemented MultiAgents like Reflex Agent,MiniMax Agent,Alpha-Beta Agent.We have also compared the performance of the above search algorithmsOut of DFS and BFS,BFS is better in terms of its performance.Of Uniform Cost Search and A* Search,A* is better in terms of its performance.We have also compared the performance of the multiagent algorithms and conclude that Reflex agent is best in terms of its performance compared to the other multiagents.Finally,we are also successful in implementing visualization to all the search algorithms.

REFERENCES

1. Russell, S., Norvig, P. (2010). Artificial Intelligence: A Modern Approach. Upper Saddle River, NJ: Prentice Hall.
2. Jamey Pittman.The pac-man dossier. http://home.comcast.net/ jpittman2/pacman/pacmandossier.html, 2011.
3. Y. Shoham and K. Leyton-Brown (2008) Multiagent systems: algorithmic, game theoretic, and logical foundations,Cambridge University Press.
4. P. Stone and M. Veloso (2000) Multiagent systems: a survey from a machine learning perspective.
5. Masataro Asai and Alex S Fukunaga."Tiebreaking strategies for A* search: How to explore the nal frontier". In AAAI, pages 673-679, 2016.
6. Game:Pacman,http://www.ling.helsinki.fi/kit/2006s/clt230/livewires/ gam es/pacman.pdf
7. M. Wooldridge (2002) :An introduction to multiagent systems : John Wiley and Sons.
8. D. E. Knuth and R. W. Moore (1975): An analysis of alpha-beta pruning.
9. D. Koller and B. Milch (2003) : Multi-agent influence diagrams for representing and solving games.

AUTHORS PROFILE

Himakar Sai Chowdary Maddipati, is pursuing 4/4 B.Tech in computer science of engineering from VR Siddhartha engineering college,Kanuru,India.His areas of interest include Artificial Intelligence,Machine learning,Virtual Reality.

Aravind Kundurthi, is pursuing 4/4 B.Tech in computer science of engineering from VR Siddhartha engineering college,Kanuru,India.His areas of interest include Python,C,Machine learning.

Pothula Mahima Raaj, is pursuing 4/4 B.Tech in computer science of engineering from VR Siddhartha engineering college,Kanuru,India.Her areas of interest include Business analytics and Cyber security.

Kapudasi Srilatha, is pursuing 4/4 B.Tech in computer science of engineering from VR Siddhartha engineering college,Kanuru,India.Her areas of interest include Java programming,Python,Web technologies.

Ravikishan Surapaneni, is working as Associate Professor in CSE department of VR Siddhartha Engineering College since 20 years.His areas of interest include Data analytics and has published more than 20 papers in various reputed journals.