Cs 5100 Foundations of Artificial Intelligence

Pacman vs Ghosts Framework:

Problem Statement:

Creating an AI controller for the ghosts/pacman using the MS Pacman v/s Ghosts framework.

Algorithms Implemented:
I implemented the following Blind Search Algorithms for Pacman for the first submission for the final project:

1. Breadth First Search
2. Depth First Search
3. Iterative Deepening Depth First Search

Implementation:

Tree.java: The tree that will be created based on the game state, for the Pacman/ Ghost to search through for the best move using either of the algorithms can be found in "MoveController" package. The formation of tree takes O(n^2) complexity which is quite an overhead and needs to be worked on so that the deeper trees can be constructed faster – which will make the searching better.

Evaluation.java: This class is the core of the entire logic. This file controls the various constants for the tree generation, includes a method for evaluating every node (possible game state) and also it contains the logic for the next best move. The heuristics to determine the best moves is determined by Boolean variable COMPLETE_LEVEL. If the variable is set true, the Pacman goes for LEVEL COMPLETION by following pills rather than following the edible ghosts. If the Ghost is 20 or less than 20 units nearer to the pacman, it changes it's course of direction to move away from ghost. If the edible ghost is 100 or less units nearer to the pacman it goes nearer to the ghost to eat it (only if COMPLETE_LEVEL is set false).

BFSController.java: Uses Breadth First Search to traverse through the tree. It gets the best move by calling getBestMove() function from Evaluation class.

DFSController.java: Uses Depth First Search to traverse through the tree. It gets the best move by calling getBestMove() function from Evaluation class.

iterativeDeepeningController.java: Uses Iterative Deepening Depth First Search to traverse through the tree. It gets the best move by calling getBestMove() function from Evaluation class.

Comparison and Limitations:

Implementation Comparison and Limitations: Currently amongst all these algorithms Iterative Deepening performs more optimally in both the scenario where the heuristics is focused on collecting points. Iterative Deepening performs well since, due to balanced depth of tree time isn't much of the overhead at the moment and also space isn't the overhead for pacman to find the best possible moves. Since, none of the algorithms have been able to complete the level of of yet no comments can be made in that scenario as of now. Moreover, this is just based on few test runs of mine and in future proper data modelling and machine learning algorithms can be applied to find the most optimal solution.

Theoretical Performance Comparison:

Evaluation	**Breadth First**	**Depth First**	**Iterative Deepening**
Time	B^D	B^M	B^D
Space	B^D	BM	BD
Optimal?	Yes	No	Yes
Complete?	Yes	No	Yes

Where B is Branching Factor, M is the Maximum Depth and D is the Depth of the solution.