This handout includes space for every question that requires a written response. Please feel free to use it to handwrite your solutions (legibly, please). If you choose to typeset your solutions, the `README.md` for this assignment includes instructions to regenerate this handout with your typeset LaTeX solutions.

1.a

The minimax algorithm is a recursive algorithm used to choose an optimal move for a player assuming that the opponent is also playing optimally. It is a depth-first search algorithm for generating and exploring the game tree.

For the context of Pac-Man with multiple adversaries (the ghosts), the minimax algorithm would extend as follows:

Let $V_{\text{minimax}}(s, d, a)$ be the minimax value of a state s at depth d for agent a. The function $V_{\text{minimax}}(s, d, a)$ can be defined recursively. The depth d decreases with each level (each agent's move), and when it reaches zero, the evaluation function is used to determine the value of the terminal state. We consider agent a to be the maximizing agent if $a = 0$ (Pac-Man), and a minimizing agent if $a \neq 0$ (ghosts).

The recurrence for $V_{\text{minimax}}(s, d)$ can be written as follows:

$$V_{\text{minimax}}(s, d) = \begin{cases} \text{Utility}(s) & \text{if IsEnd}(s) \text{ is true} \\ \text{Eval}(s) & \text{if } d = 0 \\ \max_{a \in \text{Actions}(s)} V_{\text{minimax}}(\text{Succ}(s, a), d - 1) & \text{if Player}(s) = a_0 \\ \min_{a \in \text{Actions}(s)} V_{\text{minimax}}(\text{Succ}(s, a), d - 1) & \text{otherwise} \end{cases}$$

- n is the number of ghosts (minimizing agents).
- a_0 represents Pac-Man (the maximizing agent).
- The depth d is the number of moves to look ahead in the game, and $d = 0$ represents that we are at the terminal nodes of our search depth.
- IsEnd(s) checks if state s is a terminal state (end of the game).
- Utility(s) provides the utility of that terminal state.
- Eval(s) function is used to evaluate the worth of the non-terminal state s when the depth limit is reached.
- Actions(s) gives us the set of all possible actions from state s.
- Succ(s, a) gives us the successor state after action a is taken from state s.
- Player(s) returns the player that is to move in state s.

This recurrence relation is used to calculate the minimax decision from the current game state by considering each possible move of Pac-Man and the ghosts in turn, looking ahead d moves in the future.

3.a

Pac-Man thrashing around next to a dot without eating it can be attributed to a few factors inherent in the minimax algorithm and its application in a complex environment like the larger boards of "openClassic" and "mediumClassic":

1. **Local Optima**: The evaluation function used in minimax might not sufficiently reward the action of eating a dot if the state evaluation does not look far enough ahead or if it doesn't incorporate the long-term benefits of eating dots. This can lead to Pac-Man preferring to move in ways that seem locally optimal but don't contribute to the long-term goal of clearing the board.
2. **Lack of Lookahead**: If the depth of the minimax search is not deep enough, Pac-Man may not see the consequences of eating a dot. As a result, he may avoid it due to the immediate risk of running into a ghost, even if eating the dot could be advantageous in the long run.
3. **Ties in Evaluation**: If the evaluation function returns the same value for multiple actions, Pac-Man may choose any of them arbitrarily. If the algorithm does not break ties in a way that promotes goal-directed behavior, Pac-Man may seem to move randomly.
4. **Non-Strategic Evaluation Function**: The evaluation function may not consider strategic elements such as cornering ghosts after eating a power pellet or clearing dots in a particular area to open up the board. Instead, it might focus on survival, which would explain why Pac-Man avoids risks even when they come with potential rewards.
5. **Ghosts' Predicted Moves**: The minimax algorithm anticipates the ghosts' moves as well. If the ghosts' optimal strategy involves cutting off certain paths, Pac-Man might avoid those paths even if there are dots to be eaten, which could result in seemingly irrational movement patterns.
6. **Computation Constraints**: On larger boards, the computational resources required to perform deep minimax searches increase exponentially. This often necessitates limiting the depth, which means Pac-Man is making decisions based on less information, potentially leading to suboptimal play.

In sum, the thrashing behavior is a combination of the limitations of the evaluation function, the depth of search, the way ties are broken, and the strategic complexity of the game itself, especially on larger boards where computational resources are a constraint.