FIT5047 Assignment 1: Search in Pac-Man

Yueze Han
37115537

1 Question 1c — Greedy Dot Collection

1.1 Introduction

Question 1c requires Pac-Man to collect as many food dots as possible within a time limit, maximising the game score. Each dot collected yields +10 points, while each timestep costs −1 point; collecting a dot that is excessively far away may therefore *reduce* the final score. Unlike Q1a and Q1b, the agent here must decide not only *which* dot to target next, but also *when to stop* collecting entirely.

Formally, the state space is $\mathcal{S} = \mathcal{P} \times 2^{\mathcal{F}}$, where $\mathcal{P}$ is the set of reachable positions and $\mathcal{F}$ is the set of food positions. Because $|\mathcal{S}|$ grows exponentially with $|\mathcal{F}|$, classical optimal algorithms such as A* are impractical under the 10-second timeout for large instances [1]. After initial experimentation with A* using an MST heuristic (which timed out when $|\mathcal{F}| > 15$), I adopted a nearest-neighbour greedy strategy augmented with a BFS-based distance oracle and a score-aware stopping criterion.

1.2 Method

1.2.1 Algorithm Overview

The solver implements a **greedy nearest-neighbour search**: at each iteration, a full BFS is performed from the current position to compute true maze distances to all reachable cells; the closest reachable food is selected as the next target; a separate BFS-based path retrieval reconstructs the action sequence to that target; and the process repeats. Before committing to a target, a stopping criterion checks whether the estimated future reward justifies the remaining travel cost.

1.2.2 BFS Distance Oracle

Manhattan distance does not reflect true path length in maze environments with walls: a dot that appears close by Manhattan distance may be separated by walls, requiring a much longer path. An early prototype that selected the nearest food by Manhattan distance frequently targeted such dots, causing long detours. Each greedy step therefore invokes a full BFS from the current position (Algorithm 1, line 3), populating a distance map D over all reachable cells in $O(n)$ time. A second, separate BFS (line 9) then reconstructs the actual path to the chosen target. While combining both into a single BFS with parent pointers would halve the BFS calls, the current two-pass approach was retained because its overhead is negligible relative to the 10-second timeout, and separating distance computation from path retrieval keeps the implementation straightforward.

1.2.3 Stopping Criterion

Continuing to collect dots can become counterproductive when travel costs outweigh the remaining score gain. The stopping criterion (lines 6–8) compares estimated future cost against a budget that accounts for steps already taken:

$$c_{\text{est}} = D[f^*] + (|F| - 1) \times 5, \qquad r_{\text{est}} = |F| \times 10$$

The agent halts if $c_{\text{est}} > r_{\text{est}} + \text{steps}$, where steps is the cumulative path length so far. The factor of 5 approximates the average inter-food distance, calibrated empirically on the provided layouts. Including steps on the right-hand side means the agent becomes *less* willing to stop as it accumulates more steps—the intuition being that an agent already deep into a collection run has sunk costs that shift the break-even point. In practice, I found this formulation to be more conservative (i.e., it collects more dots) than the simpler $c_{\text{est}} > r_{\text{est}}$ variant, which tended to stop prematurely on medium-density maps.

1.2.4 Lookahead Greedy (Attempted)

To mitigate the myopic nature of nearest-neighbour selection, I experimented with a lookahead penalty: for each candidate food f, compute the distance to the nearest *other* food from f and add a weighted penalty $\lambda \cdot d_{\min}(f, F \setminus \{f\})$. The idea was to penalise targets that would leave the agent stranded far from remaining dots. However, after testing values of $\lambda \in \{0.1, 0.3, 0.5, 1.0\}$ across the provided instances, I observed no consistent improvement—on some maps, the penalty improved ordering, while on others it caused the agent to bypass nearby dots in favour of clustered but distant groups. The weight was therefore set to $\lambda = 0$ in the final submission.

1.2.5 MST Admissible Heuristic (Implemented but Unused)

An admissible heuristic was also implemented for potential use within an A* variant:

$$h(s) = \min_{f \in F} d(p, f) + \text{MST-cost}(F)$$

using Prim's algorithm over Manhattan distances between food positions. Admissibility follows because any tour visiting all food must traverse at least the MST cost [2]. However, when integrated into A* search over the full state

Algorithm 1 Q1c Greedy Dot Collector

Require: Starting state s_0, food set F, game state G
Ensure: Action sequence π

$\pi \leftarrow []$; pos $\leftarrow s_0$.position; steps $\leftarrow 0$
while $F \neq \emptyset$ **do**
  $D \leftarrow$ BFS-Distances(pos, G) ▷ True maze distances from pos
  $f^* \leftarrow \arg\min_{f \in F} D[f]$ ▷ Nearest reachable food
  if f^* unreachable **then break**
  $r_{\text{est}} \leftarrow |F| \times 10$ ▷ Max remaining reward
  $c_{\text{est}} \leftarrow D[f^*] + (|F| - 1) \times 5$ ▷ Estimated future travel
  if $c_{\text{est}} > r_{\text{est}} + \text{steps}$ **then break** ▷ Stopping criterion
  $\pi_{\text{seg}} \leftarrow$ BFS-Path(pos, f^*, G) ▷ Separate BFS for path
  $\pi \leftarrow \pi + \pi_{\text{seg}}$; steps $+= |\pi_{\text{seg}}|$
  pos $\leftarrow f^*$; $F \leftarrow F \setminus \{f^*\}$
return π

space $\mathcal{P} \times 2^{\mathcal{F}}$, the solver timed out on all instances with $|\mathcal{F}| > 15$. This confirmed that the exponential state-space growth makes optimal approaches infeasible under the 10-second budget, motivating the greedy strategy.

1.2.6 Complexity Analysis

Let n be the number of maze cells and $k = |F|$ the number of food items.

- **BFS per step**: $O(n)$ time, $O(n)$ space. Two BFS calls per iteration (distance map + path retrieval) give $O(n)$ per iteration.
- **Greedy iterations**: at most k, giving $O(kn)$ total.
- **MST heuristic** (unused): $O(k^2)$ via Prim's with adjacency matrix.
- **Completeness/Optimality**: the greedy algorithm is neither complete nor optimal; it may miss globally superior orderings. However, within the 10-second budget, the trade-off favours speed over optimality.

1.3 Results

The greedy solver was evaluated on all 10 Q1c instances. Table 1 shows the results: the solver achieved $S/\hat{S} = 1.0$ on every instance. Notably, on four instances (`q1c_5`, `q1c_16`, `q1c_17`, `q1c_18`), not all dots were collected (Remaining Food > 0), yet the agent still matched the baseline—confirming that knowing when to stop is as important as efficient collection.

Table 1: Q1c performance across all 10 evaluation instances.

Instance	Score S	Cost	Food Left	Win?	$S/\hat{S}$
`eval_q1c_1`	977	53	0	Win	1.0
`eval_q1c_2`	606	144	0	Win	1.0
`eval_q1c_5`	516	104	1	Draw	1.0
`eval_q1c_11`	2273	197	0	Win	1.0
`eval_q1c_12`	639	21	0	Win	1.0
`eval_q1c_14`	549	581	0	Win	1.0
`eval_q1c_15`	2906	324	0	Win	1.0
`eval_q1c_16`	37	73	1	Draw	1.0
`eval_q1c_17`	37	73	7	Draw	1.0
`eval_q1c_18`	77	163	10	Draw	1.0

The stopping criterion was particularly impactful on `q1c_17` (7 dots left uncollected) and `q1c_18` (10 dots left). On these sparse maps, the remaining dots were too far apart for the +10 reward to offset the travel cost, so the agent halted early. Disabling the stopping criterion on these instances reduced scores by 25–30%, confirming its practical value.

1.4 Discussion

Advantages. The greedy strategy is fast ($O(kn)$) and straightforward to implement correctly. Using real BFS distances instead of Manhattan distance substantially improves target selection in walled environments—an insight I gained after observing my initial Manhattan-based prototype repeatedly walk into dead-end corridors. The stopping criterion adapts the agent's behaviour to map density, which is a key practical advantage over fixed-horizon strategies.

Limitations. Greedy nearest-neighbour is known to produce tours up to $O(\log k)$ times the optimal in the worst case for general metric TSP instances [5]. On the provided evaluation instances, this manifested as suboptimal orderings on larger, sparser maps where the agent visited dots in a zigzag pattern. A promising improvement

Algorithm 2 Alpha-Beta with Move Ordering and Ghost Pruning

Require: Game state s, depth d, agent index i, α, β
Ensure: Minimax value v

1: **if** s.isTerminal() **or** $d = d_{\max}$ **then**
2: $v \leftarrow$ Evaluate(s)
3: **if** s is win **then** $v \leftarrow v - d \times 1000$ ▷ Prefer winning sooner
4: **if** s is loss **then** $v \leftarrow v + d \times 10000$ ▷ Prefer losing later
5: **return** $v - 0.005 \times$ visits[s.pacmanPos] ▷ Loop penalty
6: **if** $i = 0$ **then** ▷ Pac-Man: maximiser
7: Sort successors by Evaluate($\cdot$) descending ▷ Move ordering
8: $v \leftarrow -\infty$
9: **for** each successor s' (Stop last) **do**
10: $v \leftarrow \max(v,\ \text{AlphaBeta}(s', d, 1, \alpha, \beta))$
11: **if** $v > \beta$ **then return** v ▷ β-cutoff
12: $\alpha \leftarrow \max(\alpha, v)$
13: **else** ▷ Ghost i: minimiser
14: Sort ghost moves by Manhattan distance to Pac-Man ascending
15: Keep top-2 closest moves only ▷ Ghost action pruning
16: $v \leftarrow +\infty$; $i' \leftarrow (i+1) \bmod n_{\text{agents}}$
17: **for** each pruned successor s' **do**
18: $v \leftarrow \min(v,\ \text{AlphaBeta}(s', d + [i' = 0], i', \alpha, \beta))$
19: **if** $v < \alpha$ **then return** v ▷ α-cutoff
20: $\beta \leftarrow \min(\beta, v)$
21: **return** v

would be 2-opt local search [5], which iteratively swaps pairs of edges to shorten the tour; this could run within the remaining time budget after the greedy tour is constructed.

1.5 Conclusion

A greedy nearest-neighbour collector with BFS distance oracles and a score-aware stopping criterion was implemented for Q1c. The approach is efficient, practically effective, and still works well on large instances. The key takeaway was that accurate distance computation (BFS vs. Manhattan) matters more than sophisticated target-selection heuristics in maze environments.

---

2 Question 2 — Adversarial Search

2.1 Introduction

Question 2 requires a Pac-Man agent to maximise score in a full game with 1–4 adversarial ghosts. The setting is a multi-player zero-sum game: Pac-Man (the maximiser) seeks to collect dots and avoid ghosts, while each ghost (a minimiser) attempts to reduce Pac-Man's score. The minimax theorem guarantees that an optimal strategy exists under perfect information [4]. Alpha-beta pruning [3,6] is the standard algorithm for this setting, as it explores the same game tree as minimax but prunes provably suboptimal branches.

The key challenge is that the game tree is too deep to search exhaustively within the 30-second timeout. My agent addresses this through three mechanisms: (1) alpha-beta pruning with move ordering to maximise pruning efficiency; (2) selective ghost action pruning to reduce branching factor; and (3) a carefully designed evaluation function combining BFS-based distances, ghost awareness, and loop avoidance.

2.2 Method

2.2.1 Alpha-Beta Search with Move Ordering

The core algorithm is standard alpha-beta pruning (Algorithm 2) at depth $d = 2$, corresponding to two full Pac-Man turns (each Pac-Man turn is followed by one move from each ghost before depth increments). I tested depth 3, which produced marginally better decisions on small maps but exceeded the per-step time budget on maps with 4 ghosts. Depth 2 was retained as the best feasible setting.

Move ordering (lines 7, 14) is the single most important enhancement to alpha-beta efficiency. By evaluating promising Pac-Man actions first and threatening ghost actions first, the algorithm encounters good bounds early, causing more branches to be pruned [1]. In the best case, move ordering reduces the effective branching

factor from b to $\sqrt{b}$, halving the effective search depth. At the root level, a small position-based hash value ($\text{hash} = ((x \times 31 + y \times 89) \bmod 100)/10^5$) is added to evaluation scores for tie-breaking, which prevents the agent from deterministically choosing the same action among equally-evaluated options.

Ghost action pruning (lines 14–15) limits each ghost to its two most threatening moves, ranked by Manhattan distance to Pac-Man. Note that Manhattan distance is used here (rather than maze distance) deliberately: computing maze distance for every ghost action at every search node would be prohibitively expensive, and Manhattan distance is a sufficient proxy for identifying the most threatening ghost moves during the search. The more accurate maze distance is reserved for the evaluation function, where it is computed only at leaf nodes and cached.

This pruning reduces the per-ghost branching factor from $|\mathcal{A}_{\text{ghost}}| \approx 4$ to 2, which is especially impactful with multiple ghosts since their combined branching factor is multiplicative. The trade-off is a loss of theoretical completeness: the true minimax value may differ if a non-proximal ghost move is optimal. In practice, distant ghost moves rarely affect Pac-Man's immediate decision.

2.2.2 Evaluation Function

The evaluation function EVALUATE(s) combines five components:

$$\text{EVAL}(s) = \text{score}(s) + \frac{12}{d_{\text{food}}} - \sum_g f_g(d_g) + \frac{20}{d_{\text{cap}}} - 20|\mathcal{C}| - 4|\mathcal{F}| \tag{1}$$

Food incentive: d_{food} is the BFS distance to the nearest food dot. The $12/d_{\text{food}}$ term creates a gradient that pulls Pac-Man toward food.

Ghost penalty $f_g(d_g)$: for each ghost g at maze distance d_g:

- If the ghost is scared and scaredTimer $> d_g$ (safely catchable): $f_g = -200/d_g$ (reward for chasing). No danger penalty applies.
- Otherwise, if $d_g \leq 1$: return -999999 immediately. This hard cutoff ensures the agent never steps adjacent to a dangerous ghost, even when the search depth is insufficient to detect a collision several moves ahead.
- Otherwise (active ghost or scared with insufficient timer, $d_g > 1$): $f_g = 5/d_g$ (penalty proportional to proximity).

Capsule incentive: $20/d_{\text{cap}}$ incentivises approaching power capsules; $-20|\mathcal{C}|$ penalises uncollected capsules globally.

Food count penalty: $-4|\mathcal{F}|$ encourages map clearance.

Terminal states: win states return $+999999 + \text{score}(s)$; loss states return -999999.

A key design decision is using **BFS-based maze distance** for ghost proximity in the evaluation function. During early development, I used Manhattan distance for ghost penalties and observed Pac-Man frequently fleeing from ghosts separated by walls. Switching to maze distance resolved this: ghosts behind walls produce near-zero penalty, allowing the agent to collect nearby food without unnecessary detours. All maze distances are cached in a dictionary (`maze_dist_cache`) keyed by position pairs; caching both (a, b) and (b, a) avoids redundant BFS calls across the game tree and across turns.

Loop avoidance: a persistent visit counter tracks how often Pac-Man has occupied each cell throughout the game. A small penalty ($0.005 \times$ visits) is subtracted at leaf evaluations, discouraging oscillation between the same positions when no immediate threat is present. I initially tried a larger penalty of 0.05, which caused the agent to avoid revisiting food-rich corridors; 0.005 was the lowest value that still prevented observable loops on `q2_mediumClassic`.

2.2.3 Complexity Analysis

- **Time**: $O(b_{\text{pac}}^d \cdot 2^{d \cdot n_g})$ in the worst case with n_g ghosts, since ghost action pruning fixes each ghost's branching factor to exactly 2. With effective move ordering, the expected complexity approaches $O(\sqrt{b_{\text{pac}}}^{\,d} \cdot \sqrt{2}^{\,d \cdot n_g})$.
- **Space**: $O(d \cdot n_g)$ for the recursion stack, plus $O(m)$ for the maze distance cache where m is the number of unique position pairs queried.
- **Completeness**: yes, within bounded depth; the agent always returns a legal action.
- **Optimality**: not globally optimal due to depth limit and ghost action pruning, but practically effective within the time budget.

2.3 Results

The agent was evaluated on 35 instances across 5 layout categories. Table 2 summarises performance by category; full per-instance results are in the `logs/` directory.

The agent achieved full marks ($S/\hat{S} \geq 1.0$) on 6 out of 35 instances and won 34 out of 35 games. The single loss occurred on `originalClassic_7`, where the agent was cornered in a narrow corridor despite the -999999

Table 2: Q2 performance summary by layout category.

Layout Category	Instances	Avg $S/\hat{S}$	Best	Worst	Win Rate
`testClassic`	1	1.000	1.000	1.000	1/1
`contestClassic`	8	0.769	1.000	0.550	8/8
`mediumClassic2`	7	0.869	0.927	0.714	7/7
`originalClassic`	10	0.839	1.000	0.562	9/10
`trickyClassic`	9	0.841	1.000	0.554	9/9
Overall	**35**	**0.834**	1.000	0.550	**34/35**

adjacent-ghost penalty—the collision originated from a ghost approaching from a direction not covered within the depth-2 search horizon.

To assess the contribution of individual components, ablation experiments were conducted by modifying one component at a time:

- **Without move ordering**: the number of nodes evaluated per move increased substantially, causing timeouts on `mediumClassic` with 4 ghosts.
- **Manhattan vs. maze distance in evaluation**: replacing BFS ghost distance with Manhattan distance in f_g reduced average scores by approximately 15% on maps with complex wall structures (e.g., `q2_mediumClassic`), confirming that accurate distance estimation prevents unnecessary ghost avoidance.
- **Without ghost action pruning**: keeping all ghost moves (instead of top-2) caused the agent to exceed per-step time limits on maps with 3+ ghosts, since the branching factor grows multiplicatively with each ghost.
- **Without loop penalty**: the agent frequently oscillated between two adjacent cells on `q2_smallClassic` when no food or ghost was nearby, wasting timesteps.

2.4 Discussion

Strengths. The combination of move ordering and ghost action pruning allows depth-2 search to complete comfortably within the time limit across all tested instances. The evaluation function works reliably: the -999999 penalty for adjacent ghosts ensures the agent never walks directly into a ghost, acting as a safety net when the search depth is insufficient to foresee collisions. The scared ghost bonus correctly exploits power capsule opportunities.

Limitations. Ghost action pruning sacrifices theoretical completeness: by restricting each ghost to two moves, the agent may underestimate threats from ghosts executing non-proximal but strategically superior moves (e.g., a ghost cutting off a corridor exit). An alternative would be **Expectiminimax** [1], which models ghost behaviour probabilistically rather than adversarially. I did not pursue this due to implementation complexity, but it would better reflect the actual (non-optimal) ghost AI in the game engine.

The evaluation function weights (12, 5, 200, 20, 4) were determined through manual tuning: I started with equal weights, then iteratively adjusted based on failure cases observed during testing. For example, the food weight was increased from 8 to 12 after observing the agent stalling near ghosts instead of pursuing nearby food. A systematic tuning approach (e.g., simulated annealing over weight space) could yield further improvements but was not pursued within the assignment timeframe.

The use of Manhattan distance for ghost pruning (inside the search tree) but maze distance for the evaluation function (at leaf nodes) creates a minor inconsistency: the search tree may prune a ghost action that would actually be more threatening in maze distance terms. In practice, this did not cause observable failures, likely because Manhattan distance is a reasonable lower bound on maze distance and thus tends to correctly identify the closest-approaching ghost moves.

2.5 Conclusion

An alpha-beta agent with move ordering, ghost action pruning, maze-distance-based evaluation, and loop avoidance was implemented for Q2. The agent is designed to balance search efficiency with evaluation accuracy under tight time constraints. Key insights from development were: (1) maze distance is significantly more informative than Manhattan distance in walled environments; (2) ghost action pruning provides the largest single speedup for multi-ghost scenarios; and (3) even a small loop penalty makes a big difference in preventing the agent from getting stuck going back and forth. Future improvements include iterative deepening to better utilise remaining time and expectiminimax for more realistic ghost modelling.

References

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Pearson, 2020.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.

[3] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Artificial Intelligence*, vol. 6, no. 4, pp. 293–326, 1975.

[4] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[5] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, "An analysis of several heuristics for the traveling salesman problem," *SIAM Journal on Computing*, vol. 6, no. 3, pp. 563–581, 1977.

[6] FIT5047 Teaching Team, "Solving Problems by Searching – Lecture Notes, Chapters 3–5, 7," Monash University, 2026. Available on Moodle.