Mansoura University
Faculty of Computers and Information
Department of Computer Science

Clever Crossing: An AI-Driven River Crossing Puzzle Solver

Submitted by

. HABIBA IBRAHIM ELAWADY
. EMAN EL-SAYED HELAL
. ABDULRAHMAN SAEED ELRAIS

Supervisors:

DR/ SARA EL-SAYED EL-METWALLY

Signature:

DR/ MAI MOHAMED

Signature:

Date Submitted
30/5/2025

1- INTRUDCTION

When we look at our River Crossing Puzzle through the PEAS lens, we begin by defining exactly how we'll judge success. In "Clever Crossing," the **performance measure** is straightforward: the agent should always find the shortest valid sequence of crossings, minimizing the total number of moves. Along the way, we also track how many states the agent expanded and how much memory it consumed, since those metrics give us insight into the efficiency of our implementation.
Next, we consider the **environment** in which the agent operates. Here, the world is simple and fully transparent—there are two riverbanks ("left" and "right"), and four fixed entities: the farmer, the wolf, the goat, and the cabbage. Every action deterministically moves the farmer across the river, either alone or accompanied by exactly one of the other three items. No surprises or hidden variables exist; as soon as the agent inspects a state, it knows exactly which crossings are legal and which would leave the wolf alone with the goat or the goat alone with the cabbage.
To change the world, our agent relies on its **actuators**, which in this case amount to a single capability: commanding the boat to cross the river. That command might ferry the farmer by himself or with one designated item. Because the boat's capacity is limited, each move must respect that constraint and then immediately update the positions of the farmer and any accompanying cargo.
Finally, the agent's **sensors** consist of perfect state-recognition: at any point it can read the exact location ("left" or "right") of each of the four entities. This full observability means the agent never needs to guess or infer hidden information—every potential move's legality can be checked on the spot by our validation function. Altogether, this PEAS formulation keeps our design sharply focused on finding the fewest-move solution in a small yet fully understood world.

2. Project Objectives

1- Implement BFS to find the optimal sequence of moves.

2-Develop modular, well-documented code with unit tests.

3-Provide a CLI example for demonstration purposes

3. PEAS Analysis

Dimension	Details
Performance Measure	- Shortest solution length (fewest moves) - Number of states expanded - Memory usage
Environment	- Two riverbanks ("left" and "right") - Fixed entities: farmer, wolf, goat, cabbage - Deterministic, fully observable
Actuators	- Move the farmer alone or with one item across the river
Sensors	- Detect locations (left/right) of all entities

4. ODESA Framework

When we examine “Clever Crossing” through the ODESA framework, we first articulate our **Objective**: we need an agent that reliably transports the farmer, wolf, goat, and cabbage from the left bank to the right without ever leaving forbidden pairings alone—specifically, wolf with goat or goat with cabbage—and does so in the fewest possible crossings. The **Data** backing that objective is elegantly minimal: each world configuration is stored as a four-element tuple of “left” or “right” for the farmer, wolf, goat, and cabbage, along with implicit definitions of the starting state (("left","left","left","left")) and goal state (("right","right","right","right")). Our **Environment** is equally straightforward—two riverbanks, a deterministic boat that carries the farmer alone or with one passenger, and no hidden variables or randomness. From there, our **Solution** strategy is to treat this as a state-space search problem: we run a breadth-first search over the implicit graph of valid tuples so that the very first time we encounter the goal state, we know we’ve used the minimum number of moves. Underpinning all of this is our **Agent**, a model-based searcher that fully knows the transition rules (via our game-mechanics module), systematically explores each legal successor, and then executes the plan it has confidently computed offline—guaranteeing both optimality and clarity of design.
o4-mini

5. Agent Type

Model-Based Search Agent: Maintains an explicit model of states and transitions, has full knowledge of the environment, and plans using BFS to generate an optimal move sequence.
In our “Clever Crossing” project, the agent we design is fundamentally a model-based search agent, which means it doesn’t stumble forward one move at a time in the dark—instead, it builds and uses a complete picture of how the world behaves before it ever takes a single step. From the very beginning, this agent holds in memory the full state-transition model: it knows precisely how moving the farmer alone or with any one of the wolf, goat, or cabbage alters the configuration of the four entities on the riverbanks. Armed with this knowledge, it then performs an exhaustive, breadth-first exploration of all possible states, systematically expanding one layer of moves at a time until it uncovers the goal configuration where everyone safely occupies the opposite shore.

6. Problem Formulation

When we cast the River Crossing Puzzle into a formal search problem, we begin by representing each configuration as a simple four-element tuple—one entry for the farmer, wolf, goat, and cabbage—each marked either “left” or “right” to indicate its riverbank. Actions in this world are equally straightforward: at any step, the farmer may cross alone or take exactly one of the other three items with him, and this is implemented by a move function that flips the relevant positions in the tuple. Our transition model then filters out any resulting state that violates the puzzle’s safety rules—specifically, those in which the wolf and goat or goat and cabbage end up unsupervised on the same bank. Finally, we treat the search space as an implicit graph whose nodes are these valid states and whose edges represent single crossings, assigning each move a uniform cost of one.

7. Timeline

Week	Milestone
1	Requirements finalization & design docs
2	game_mechanics.py development & tests
3	game_solver.py development & tests
4	Documentation, demo scripts, review

8. Conclusion

In closing, “Clever Crossing” demonstrates how a well-structured, model-based search agent can elegantly and efficiently solve a classic river-crossing puzzle. By defining a simple yet complete state representation, enforcing the domain’s safety constraints through our mechanics module, and applying a breadth-first search over the resulting state graph, we guarantee both correctness and optimality. The separation of concerns—model definition, legal-move generation, and planner execution—yields a design that is not only easy to understand and verify but also readily extensible to more complex variants or heuristic enhancements. Ultimately, this project serves as both a clear pedagogical example of AI search principles in action and a reusable codebase for future experimentation and teaching.