ShoverWorld – Plan and Challenge Mode

1. Introduction

This phase of the ShoverWorld project grows directly from the environment created in Phase 1, but introduces a goal-directed Player AI and a bespoke challenge configuration to test intelligent planning under resource constraints. The fundamental environment mechanics are unchanged, but the default parameters and reward structure have been adjusted to the Phase 2 specs.

ShoverWorld is a fixed-size (13 × 9) grid-world consisting of sliding box tiles with lava and immovable obstacles. The player manipulates the environment by clicking on grid squares and dropping down directional or special actions. All interactions sap or recharge energy based on the action's success. The world now also explicitly models delay-based energy loss, promoting efficient and timely decision making.

Your goal here is to eliminate every box on the grid with enough energy left. To do so, it has a search-based Player AI implemented using A*. This agent formulates the environment as a deterministicplanning problem and calculates an entire plan of actions that transition from the initial state to a goal state. A specially crafted challenge map is supplied to guarantee the AI has to think about box chains, perfect squares, lava, and energy trade-offs instead of resorting to trivial solutions.

2. environment.py

The ShoverWorld environment remains a custom Gymnasium-compatible class, specifying a deterministic rule-based grid world. For Phase 2, the environment parameters are at their project default settings to provide a consistent comparison across evaluations.

The grid is 13×9 and the player starts each episode with 1000 units of energy. Directional push actions use energy based on two variables. First, a base force of 10 diminishes the player's energy relative to the length of a push chain. Second, it takes an initial force of 40 to break the inertia of box chains at rest — the price for getting something started.

The environment rewards destructive actions. Every box pushed off the board—usually by lava—awards 40 energy, incentivizing tactical elimination instead of mindless pushing. Also, turning a complete n × n perfect square of boxes into an obstacle grants $10n^2$ energy, motivating the agent to purposely create and take advantage of square formations.

There is also a timing penalty. The environment times the gap between successive actions, subtracting an energy unit for each 200ms, incentivizing efficient play and penalizing fiddling.

The objective state is obvious–if the grid boxes count hits zero, the episode terminates with a victory. On the other hand, if the player's energy hits zero with at least 1 box remaining, then the game ends in failure. As in Phase 1, the environment handles perfect-square aging, stationary box tracking, lava destruction, and termination logic internally, providing deterministic, reproducible transitions for planning-based agents.

3. A* Strategy Player (strat_player_live_hud.py)

The Strategy Player used in this phase is an objective-based planning agent that runs an A* search to find a full plan before performing any actions. Instead of a reactive agent that decides moves one at a time, this player plans out the entire future sequence of actions, explicitly modeling how each action alters the grid and player's available energy.

State Representation

The state of each node in the A* search is a complete snapshot of the world, containing:

The current grid configuration (positions of boxes, obstacles, lava, and empty cells)

The remaining energy level

Implicit information like box still states and squares

States are considered immutable and are serialized to hashable representations so they can be easily stored in a visited set. This stops the search from cycling through previously visited configurations, and also prevents infinite loops from reversible moves.

- Action Generation
  From a any state, the agent produces all possible legal actions that can be performed:\
  To push a box in any of the four directions
  Causing a legitimate full square of boxes to convert into an obstacle
  So that box chains could interact with lava tiles to delete boxes
  Instead of hardcoding validity rules, the agent depends on the environment's internal transition logic to establish whether an action is legal. Each candidate action is run using a duplicate of the environment, so that the resulting state captures all movement rules, energy costs, and rewards. Expired deeds of course lead to no state growth.

- Cost Function
  The A* cost function based mainly on energy. Tasks that waste significant energy—like pushing long runs of boxes or constantly jumping-starting inertia—lead to elevated path costs. Energy bonuses from box elimination or square conversion lower the effective cost of arriving at a state, causing these actions to appear desirable in planning.
  This formulation also gives the agent a natural preference for plans that clear boxes quickly, take advantage of square bonuses, and reduce wasteful moves. Timing penalties are not explicitly modeled when planning, but are implicitly minimized by producing compact action sequences.

- Heuristic Design
  A domain specific heuristic directs the A* search to likely states. The heuristic approximates the distance a state is from the goal by looking at
  The amount of boxes left on the grid
  The incidence of prospective or actual perfect squares
  The spatial clotting of boxes, influencing how readily they can be withdrawn

States with fewer boxes and more rigid arrangements get lesser heuristic scores. Although the heuristic is not strictly admissible, it greatly enhances search efficiency, and it always results in good solutions quite quickly.

- Search Process
  For instance, for A*, the frontier is a priority queue sorted by path cost + heuristic value. States are expanded in ascending order of estimated total cost. The search ends once it generates a state with no boxes on the grid.
  Upon reaching a goal state, the agent rebuilds the action sequence by following parent pointers back to the start state. This results in a full, linearized plan of action that will succeed when implemented from the initial state.

- Execution and Visualization
  After planning, the world is rewound and the determined action sequence is carried out one step at a time. A live HUD displays:
  The current grid
  Remaining energy
  Number of boxes left
  The action being executed

  This visualization verifies that the planner's hallucinated transitions correspond to the true environment dynamics and provides a lucid view of how tactical maneuvers—like forming squares and deploying lava—help resolve the puzzle.

4. Challenge Design

The challenge design phase concentrates on specifying an initial map setup that provides a non-trivial test of the Player AI's planning quality. The challenge map still respects the 13 × 9 grid but gives free reign to box/obstacle/lava placement.

This challenge map is not trivial. Boxes are scattered over the grid in both sparse and dense configurations, with obstacles arranged to limit direct navigation and compel indirect approaches. Some areas will yield perfect squares only after strategic rearrangement, so that the AI sometimes needs to think several moves ahead to unlock high reward moves. Lava tiles are strategically placed to reward accurate chain pushes and punish haphazard movement.

This construction defeats greedy or purely local approaches. Indeed, solving the map demands coordinated application of pushing mechanics, square-based transformations, and energy-conscious sequencing. We selected the map such that the Player AI constructed in this phase can solve it.

5. Execution Flow

The execution process starts up by initializing the environment with the default Phase 2 parameters and loads in the challenge map. The A* planner is then called to generate a complete plan from

the start to the goal. Planning is all done offline before execution, based on deterministic environment simulation instead of online feedback.

Once a plan is discovered, the context is reset and the action plan is followed stepwise. At runtime, the live HUD draws the grid and shows metrics including timestep count, energy remaining, boxes remaining, and recent interactions. The episode ends automatically on success or failure as per the goal definition.

6. Conclusion

Phase 2's ShoverWorld project shows how classical planning can be incorporated into a tricky, rule-based grid world. With the environment parameters fixed, and by injecting explicit energy rewards, penalties, and delay costs, the problem becomes a planning problem, rather than a purely reactive problem.

Our A* Strategy Player, which can effectively reason over long action sequences, balance energy expenditure against rewards, and exploit environment mechanics (e.g., perfect squares, lava destruction). The challenge map proves the agent's competence to solve non-trivial configurations that demand foresight and coordinated action.

Together, environment, planning agent, and challenge design establish a well-defined and reproducible setting for investigating goal-directed behavior in grid worlds.

Maryam Soleimani Bahare Motamedi