AI Report

- A description of the implementation of GenGrid.

  GenGrid is a method that takes as input the grid represented as a string.This string is then manipulated to extract the needed positions of Ironman,Thanos and the four stones.However,due to the fact that we have four stones,we also split over the commas of the stones using a for loop.This is done using the predefined function split().Where we do splitting over the semi colon.After saving each position of X and Y in a variable,we concatenate them into one global string that will be written to a new .pl file.This is achieved in the main method which calls the GenGrid and gives it as input our example grid.Consequently,the global string knowledgeBase gets updated and is written to a new file using the FileOutputStream library.

- A discussion of the syntax and semantics of the action terms and predicate symbols you employ.
  1. **inside(X, Y)** predicate makes sure that ironman moves inside the grid not making invalid movements, where X and Y are ironman's coordinates.
  2. **ironman((X, Y), [], result(A, S))** predicate moves ironman to thanos position where X and Y are ironman's coordinates, [] is the empty list as no stones are collected yet, A is the action, S is the set of states
  3. **ironman((X, Y), [(Sx, Sy)|T], result(A, S))** predicate moves ironman to all possible stones position where X and Y are ironman's coordinates,[(Sx, Sy)|T] is the stones that are collected (Sx,Sy) are the coordinates of the recently collected stone and T is the tail of the list, A is the action, S is the set of states.
  4. **won(result(A,S))** predicate which gets the final solution such that ironman collects all stones and moves to thanos. A will be snapped and S is the set of states such that ironman collects all stones and moves to thanos.
  5. **snapped(S)** predicate is the best case query where it calls **snapped(S,1)**,meaning that it will only have a depth of 1.
  6. **snapped(S, X)** predicate is the expansion of snapped(S) where S is the set of states and X is the current depth.
- A discussion of your implementation of the successor-state axioms.

  The first one is ironman which has the following two definitions:

  1. **ironman((X, Y), [], result(A, S))**
  2. **ironman((X, Y), [(Sx, Sy)|T], result(A, S))**

These two predicates represent the status of ironman and the collected stones, As (X, Y) is the position of ironman and the implementation of the predicate handles the movement of ironman in the 4 directions in the grid. And the second argument in the predicate is representing

the list of the collected stones so far. In the first one it's written as ([]) which means that ironman is still in the beginning of the game and he did not collect so far. And the second one it handles the state of the list depending on the constraint of 'collect' action.

The second one is won which has the following definition:

won(result(A,S))

First it gets the coordinates (X,Y) for thanos then gets the list of available stones and outputs them in variable ST then calls **ironman((X,Y),ST,S)** where S will be the states of which ironman will go collect all stones then moves to thanos's cell. After which this is satisfied result(A,S) will be equal to result(snap,S)..

- A description of the predicate **snapped(S)** used to query the KB to generate the plan.

  snapped(S) predicate used IDS which has a best case of a depth of one. If it didn't get the answer it will keep expanding unless it reached the maximum depth by prolog. Moreover it calls the successor-state axioms **won(result(A,S))** which gets the solution as described in the point above.

- At least two running examples from your implementation.

- First Example using the example **grid = 5,5;1,2;3,4;1,1,2,1,2,2,3,3**
  S = result(snap, result(right, result(collect, result(down, result(right, result(collect, result(right, result(collect, result(down, result(collect, result(left, s0))))))))))

- Second Example using another example **grid = 4,4;0,0;3,3;1,1,1,2,2,2,2,3**
  S = result(snap, result(down, result(collect, result(right, result(collect, result(down, result(collect, result(right, result(collect, result(down, result(right, s0))))))))))