# Section 1 – Dynamic Programming Pseudocode

Function DP\_search(mine)

Setup/Initialise cache (maxsize = None)

Function Recursive\_Search(State)

Best payoff = payoff(state)

Best action list = []

Best final state = state

For each child action in all valid actions

Child state = result(state, child action)

Child payoff, child action list, child final state = Recursive\_Search(child state)

If child payoff is better than best payoff

Best payoff = child payoff

Best action list = child action + child action list

Best final state = child final state

Return best payoff, best action list, best final state

Print(cache info)

best payoff, best action list, best final state = Recursive\_Search(initial state)

Print(cache info)

return best payoff, best action list, best final state

# Section 2 – Branch and Bound Pseudocode

Branch\_Bound(mine)

Node = initial mine node

Frontier = Priority que of nodes ordered by payoff

Frontier += node

Best node = node

While frontier

Node = next node in frontier

For child node in all child nodes

If child node is not in frontier

Add child node to frontier

Else

If child node payoff is better than current payoff

delete incumbent node

best node = child node

add child node to frontier

return best payoff, best action list, best final state

# Section 3 – Testing Methodology

* Used same case as provided to compare outputs
* Printed cache data to compare hits and miss even if time wasn’t same
* Used timings
* Made other test mines ie when optimal is near surface, deep, optimal under layer of negatives
* Ran through on debugging initially to check data types and correct indexing
* Also had asserts throughout to test computations

# Section 4 – Performance and Limitations

-the DP exels at small mines or mines where states will overlap but its time grows exponentially for larger more complex mines such as 3d

- will be slower the deeper the optimal solution is