Sokoban-Game-Proposal

Luis Hsu(22609779), Yao Chung Chang(55965465)

1 Current results

In the first beginning, let's see the result we've done currently.
This is one random picked map from the DeepMind 'boxoban-levels' datasets

```
##########
# ###    #
# @## $. #
# $. .$$ #
#   # .  #
##########
```

Reference: `https://github.com/deepmind/boxoban-levels`

The result under map is the training process of our convolution neural network containing average loss and difference of loss between two adjacent epoch.

```
[0] Avg loss: 1.14988 abs: 1.14988
[1] Avg loss: 1.02034 abs: 0.12954
[2] Avg loss: 0.958701 abs: 0.0616429
[3] Avg loss: 0.935158 abs: 0.0235434
[4] Avg loss: 1.14121 abs: 0.206053
[5] Avg loss: 0.913932 abs: 0.227278
[6] Avg loss: 0.913879 abs: 5.29289e-05
[7] Avg loss: 0.913874 abs: 5.36442e-06
[8] Avg loss: 0.930584 abs: 0.0167106
[9] Avg loss: 0.927268 abs: 0.0033167

[0] Avg loss: 0.896039 abs: 0.896039
[1] Avg loss: 0.892526 abs: 0.00351292
[2] Avg loss: 0.892595 abs: 6.87242e-05
[3] Avg loss: 0.892594 abs: 4.17233e-07
```

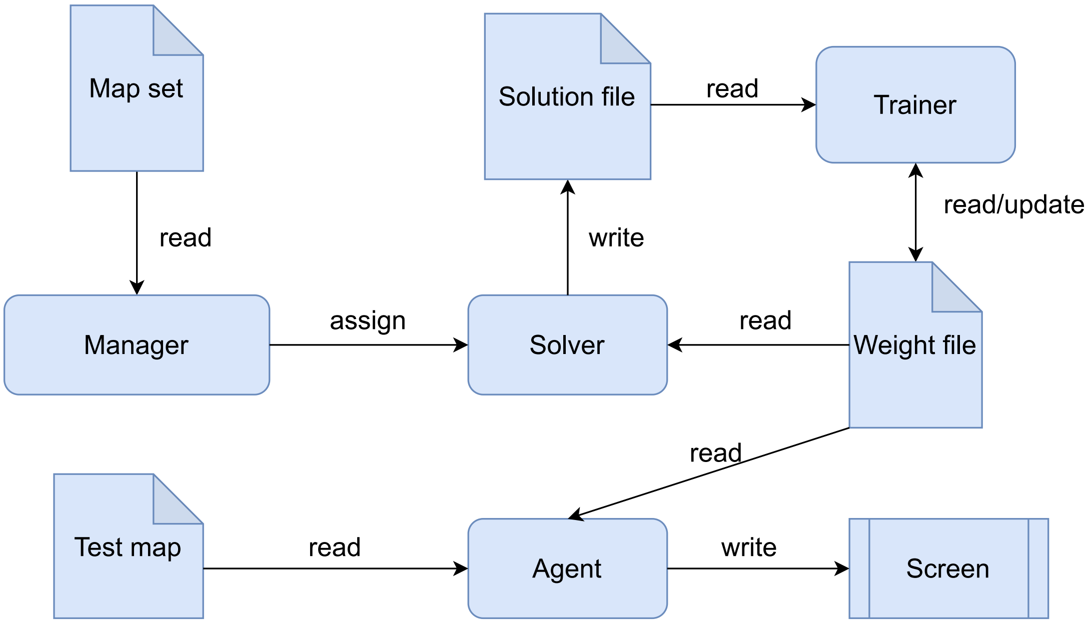
The logs below is the exploring process of our AI agent.

```
MaxIter: 99 Restart: 17852 a: 133.615 b: 1.16327 r: 1 R: 102 a/R: 1.30995 b*T: 0
MaxIter: 100 Restart: 17856 a: 133.63 b: 1.16162 r: 0.998583 R: 103 a/R: 1.29738 b*T: 0
MaxIter: 101 Restart: 17879 a: 133.716 b: 1.16 r: 1 R: 102 a/R: 1.31094 b*T: 0
MaxIter: 102 Restart: 17882 a: 133.727 b: 1.15842 r: 0.965405 R: 103 a/R: 1.29832 b*T: 0
MaxIter: 103 Restart: 17909 a: 133.828 b: 1.15686 r: 0.970192 R: 104 a/R: 1.28681 b*T: 0

=== Finished ===
total steps = 47

== Solution ==
Left, Down, Down, Right, Right, Up, Right, Right,
Down, Right, Right, Up, Left, Down, Left, Up,
Left, Left, Down, Left, Left, Up, Right, Right,
Right, Right, Down, Right, Right, Up, Left, Left,
Left, Right, Right, Right, Right, Up, Up, Left,
Left, Left, Down, Up, Right, Down, Down
```

```
=== Final state ===
##########
# ###    #
#  ##  % #
#  % %@  #
#   # %  #
##########
```

2 System overview

After having a glance of our current result, it's to look into the whole system.
Here comes the block diagram:



- Solver: This program first receives the sokoban map from Manager and then will read the weight file that containing our CNN model. After solving the map, it will write the solution to Solution file.
- Trainer: This program first reads the Weight file to get the CNN model and then read the solution file generated by solver. After that, it starts training by using existing solutions. Finally, it updates the newest model to Weight file.
- Manager: This program first reads the maps in the Map set file and the assigns the map to solver one by one.
- Agent: This program first read one map from Test map file, then read the CNN model from Weight file and start to solve the map. After solving the map, it will show the solution on terminal screen.

3 Algorithms

- MAKE_STATE(map): Make new state from map
- NEXT_STATE(map, direction): Make next state from map and direction
- DISTANCE(state): Calculate the distance between root state and the state
- IS_WIN(state): Check whether the state is winning state

Algorithm 1 The main solve algorithm

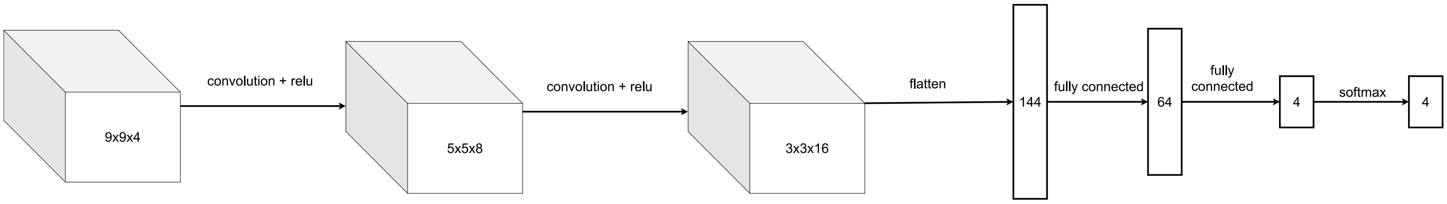
```
function SOLVE(map) returns policy, a sequence of move direction
    Input: map, the map to solve
    Statics: states, a set of states    iteration_limit, the current limit of iteration
    $states \leftarrow MAKE\_STATE(map)$
    $iteration\_limit \leftarrow 1$
    while  do
        if $current\_state$ is new then
            for direction in [Up, Down, Left, Right] do
                $next\_state \leftarrow NEXT\_STATE(map, direction)$
                if $next\_state$ exists then
                    if $next\_state$ hit wall or unmovable box then
                        continue
                    end if
                    if $DISTANCE(current\_state) + 1 < DISTANCE(next\_state)$ then
                        $current\_state[direction] \leftarrow next\_state$
                    end if
                else
                    $states \Leftarrow next\_state$
                    $current\_state[direction] \leftarrow next\_state$
                    if $IS\_WIN(next\_state)$ then
                        $policy \Leftarrow direction$
                        return policy
                    end if
                end if
            end for
            if $current\_state$ is dead node then
                clean dead nodes and restart from initial state
            end if
            $next\_direction \leftarrow DECIDE(current\_state)$
            $current\_state \leftarrow next\_state$
            if reach iteration_limit then
                $iteration\_limit \Leftarrow iteration\_limit + 1$
                restart from initial state
            end if
        end if
    end while
```

```
function DECIDE(state) returns direction, the preferred next direction
    Input: state, the current state
    Statics: possibilities, a set of the possibilities that get to each direction
    $confidences \leftarrow CONFIDENCE(state)$
    $suggestions \leftarrow SUGGEST(state.map)$
    for direction in state do
        $possibilities[direction] \leftarrow (1-\gamma)\frac{confidences[direction]}{SUM(confidences)} + \gamma\frac{suggestions[direction]}{SUM(suggestions)}$
    end for
    return a direction randomly chosen by the possibility of possibilities
```

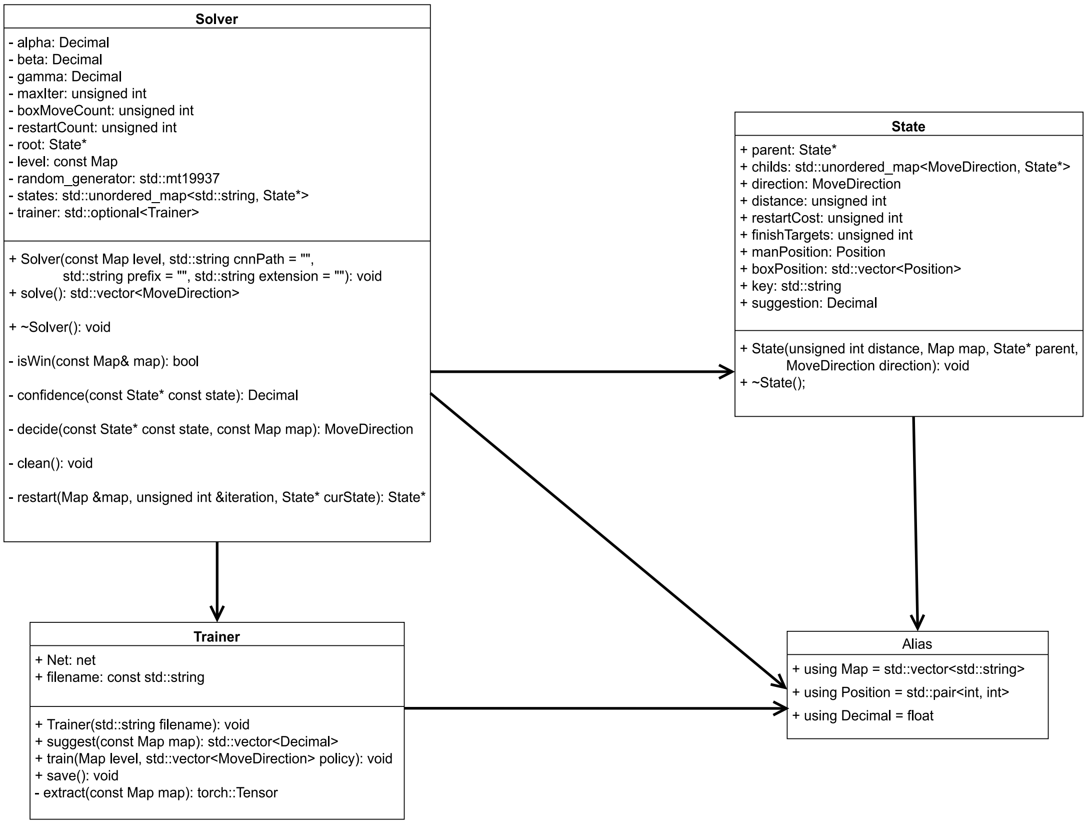
```
function CONFIDENCE(state) returns confidences, a set of decimal numbers represent confidences of each actions
    Input: state, the current state
    $result = 0$
    if the $restart\_cost$ of $current\_state < 0$ then
        $result = result + \alpha/\ restart\_cost$
    else
        $result = result + 1$
    end if
    if $finish\_targets = 0$ then
        $result = result + \beta * finish\_targets$
    end if
    return $result$
```

The neural network model used as SUGGEST(map)

Suggestion is to get the probability of next move for each direction. It can be regarded as a classification problem to classify the map into each directions. In this system, a neural network is used.



4 Software architecture


Solver
- alpha: Decimal
- beta: Decimal
- gamma: Decimal
- maxIter: unsigned int
- boxMoveCount: unsigned int
- restartCount: unsigned int
- root: State*
- level: const Map
- random_generator: std::mt19937
- states: std::unordered_map<std::string, State*>
- trainer: std::optional<Trainer>
+ Solver(const Map level, std::string cnnPath = "", std::string prefix = "", std::string extension = ""): void
+ solve(): std::vector<MoveDirection>
+ ~Solver(): void
- isWin(const Map& map): bool
- confidence(const State* const state): Decimal
- decide(const State* const state, const Map map): MoveDirection
- clean(): void
- restart(Map &map, unsigned int &iteration, State* curState): State*
State
+ parent: State*
+ childs: std::unordered_map<MoveDirection, State*>
+ direction: MoveDirection
+ distance: unsigned int
+ restartCost: unsigned int
+ finishTargets: unsigned int
+ manPosition: Position
+ boxPosition: std::vector<Position>
+ key: std::string
+ suggestion: Decimal
+ State(unsigned int distance, Map map, State* parent, MoveDirection direction): void
+ ~State();
Trainer
+ Net: net
+ filename: const std::string
+ Trainer(std::string filename): void
+ suggest(const Map map): std::vector<Decimal>
+ train(Map level, std::vector<MoveDirection> policy): void
+ save(): void
- extract(const Map map): torch::Tensor
Alias
+ using Map = std::vector<std::string>
+ using Position = std::pair<int, int>
+ using Decimal = float

- Solver: This class is responsible for solving the map by utilizing the algorithm described above.
- State: The instance of this class represents a state when walking in the map.
- Trainer: This class defines the structure of convolution neural network that help us speed up the solving process.
- Alias: The aliases of some type definition.