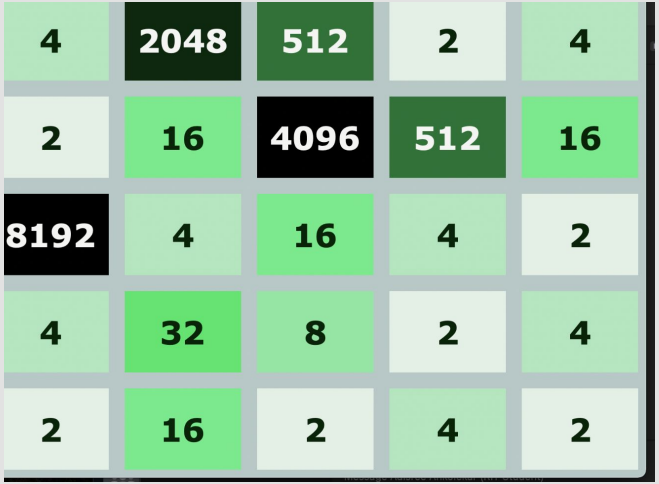
The Project: 2048 MDP

-Ashnil Vazirani (RIT | SE)

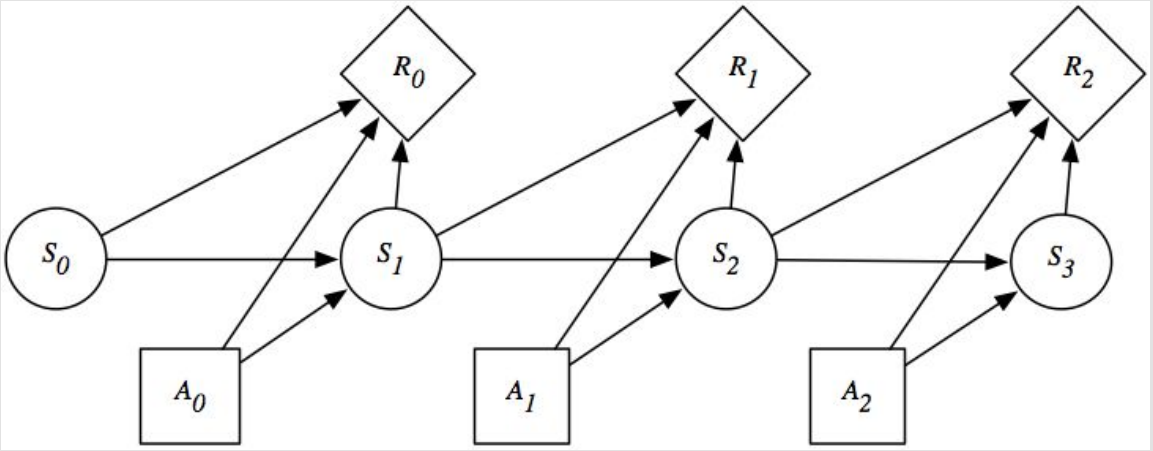
Problem statement:

- MDP grid problem
- Traverse from start to goal state
- Label after each action
- Actions:
  - mergeUP
  - mergeDOWN
  - mergeLEFT
  - mergeRIGHT
- Nxn grid world.!



Ideology

- Agent traverses the world with respect to the current state.
- Check for current tile values
- Init: move to merge with row/column
- Iterate: keep larger number at the corners
- Cleaning robot agent

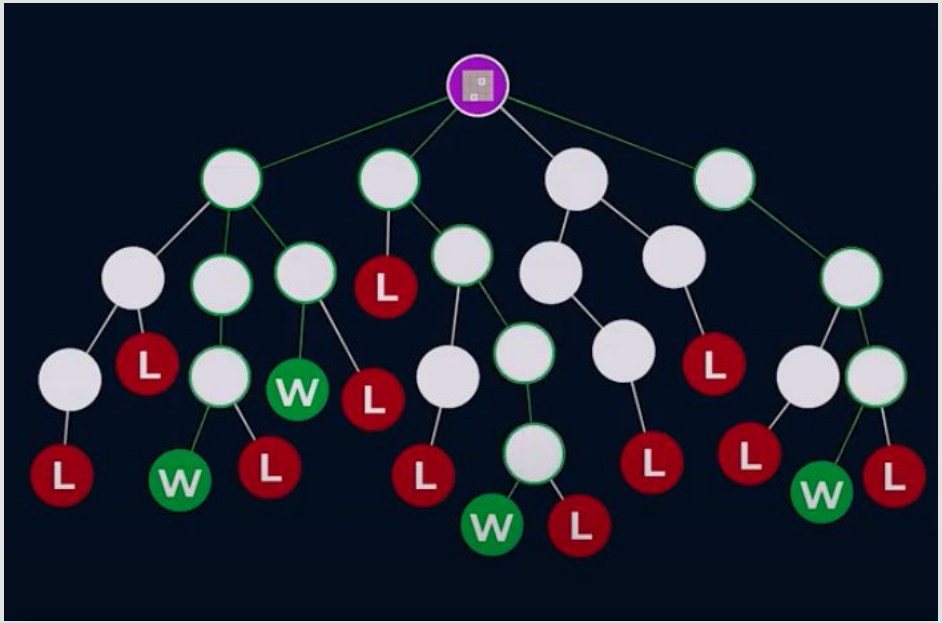


A Markov Decision Process (MDP) is a 5-tuple $\langle S, A, P, R, s_0 \rangle$, where each element is defined as follows:

- S: a set of states.
- A: a set of actions.
- $P(S_{t+1}|S_t, A_t)$: the dynamics.
- $R(S_t, A_t, S_{t+1})$: the reward. The agent gets a reward at each time step (rather than just a final reward).
  - $R(s, a, s')$ is the reward received when the agent is in state s, does action a and ends up in state s'.
- s_0: the initial state.

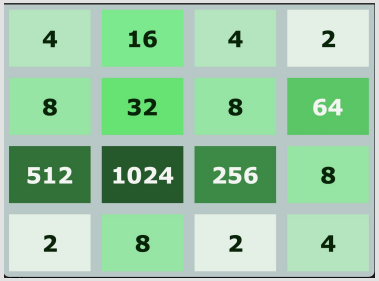
Algorithm: Monte Carlo VS Alpha-Beta

- Exploration problem
- Iterate the problem in a linear fashion
- Built a decision tree
- Search along the path of decision tree
- Idealise a move for a state by selecting the path that lies within the unit circle

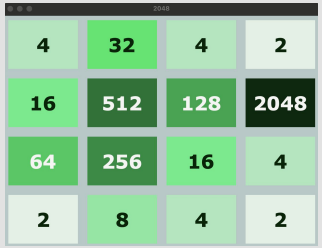


Approach:

- Incremental:
  - Agent takes on step
  - Action based on present state
  - User involvement
  - Agent is not solely responsible
  - Does not explore instead just exploits current state



- Iterative:
  - Agent is left to learn in the environment
  - Creates a decision tree
  - Decides for number of searches per move and the search length
  - searchesPerMove: branches of the tree
  - searchLength: lookahead through the depth of tree



Experiment:

- Implemented with python
- UI with tkinter library
- Keyboard inputs
- NxN grid world setup
- Metric:
  - Iteration time
  - Thing time
  - Outcomes per episode
- Output Plots:
  - Iteration time in seconds
  - Overall thing time to understand the knowledge module
- Compare optimization

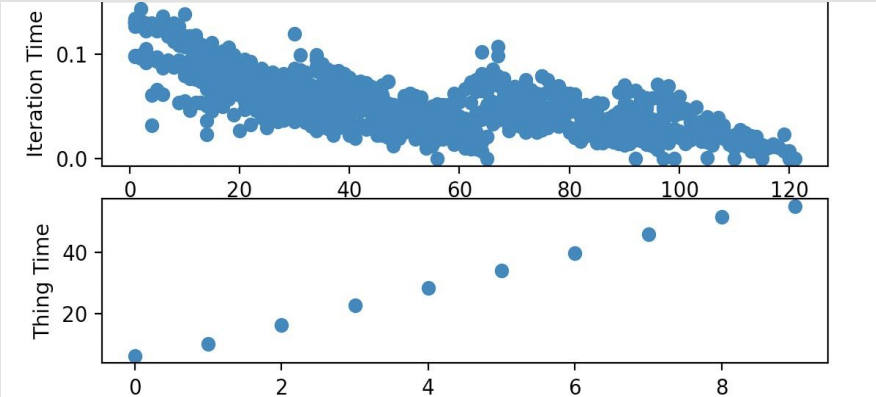
Hardware Overview:

Model Name: MacBook Pro
Model Identifier: MacBookPro17,1
Chip: Apple M1
Total Number of Cores: 8 (4 performance and 4 efficiency)
Memory: 8 GB
System Firmware Version: 6723.101.4
OS Loader Version: 6723.101.4
Hardware UUID: 93974ED8-93E0-511F-9806-9297F6423882
Provisioning UDID: 00008103-000D31CC0A62001E
Activation Lock Status: Enabled

Comparisons:

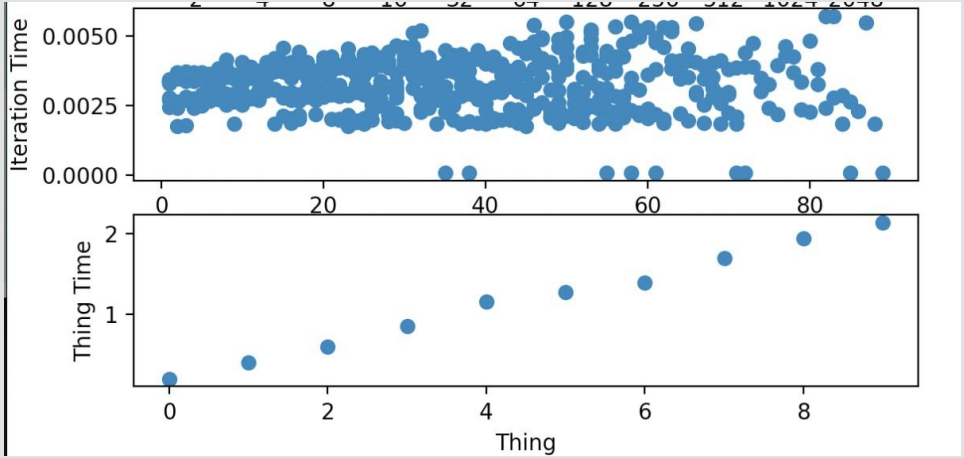
- Conform the self adaptive nature of an agent:
  - Reduces/Optimize on the number of searches per move and search length
- Behaviour of agent with respect to grid size:
  - Increases number of and iterations and thing(response) time
- Outcome of episodes:
  - Identify number of successful episodes
- Contrast the implication of time:
  - Exploration agent VS adaptive agent
- Iterative VS Incremental
  - Iterative: impact of having knowledge module

Results:



- Ideal plot for grid world
- Iteration Time up: 0.1 seconds
- Thing time up: knowledge built

- Optimised
- Iteration time: max 0.0050
- Thing time: 2 seconds



Self Adaptive agent:

- Decide to the convergence point
- Idealise the number of searches per move and search length with respect to the move number
  - numberOfSearches = 10 * (1+floor(moveNumber / 200))
  - searchLength = 4 * (1+floor(moveNumber / 200))
- HyperParameters to tuned with a max of 200 searches and searchLenght with 4 possible moves

Algorithm:

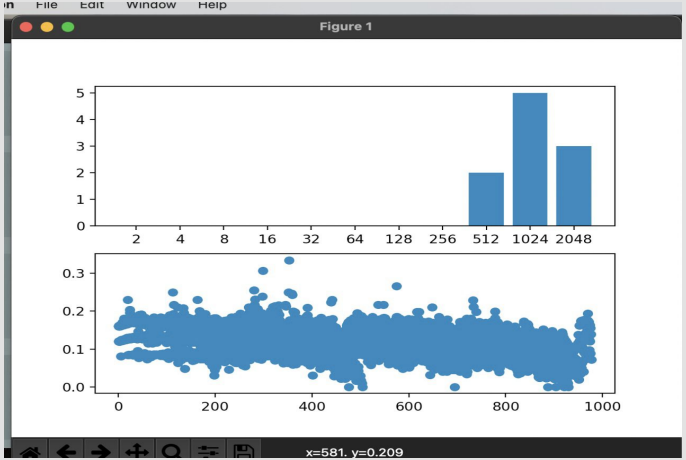
```
for move in possibleMoves:
    if firstMove:
        idealFirstMove = makeFirstMove(board)
        If idealFirstMove is valid:
            board = makeMove(idealFirstMove)
    Else:
        for searchnumber in numberOfSearches:
            moveNumber = 1
            tempBoard = currentBoard
            while gameValid and moveNumber < lengthOfSearch
                current, gameValid, score = oneRandomStep(currentBoard)
                if(gameValid):
                    currentBoard = placeNewTile(currentBoard)
                    moveScore[moveNumber] = score
bestMove = np.argmax(moveScore)
movePlaced = possibleMoves[bestMove]
board = makeMove(idealFirstMove)
```

Self Adaptive agent:

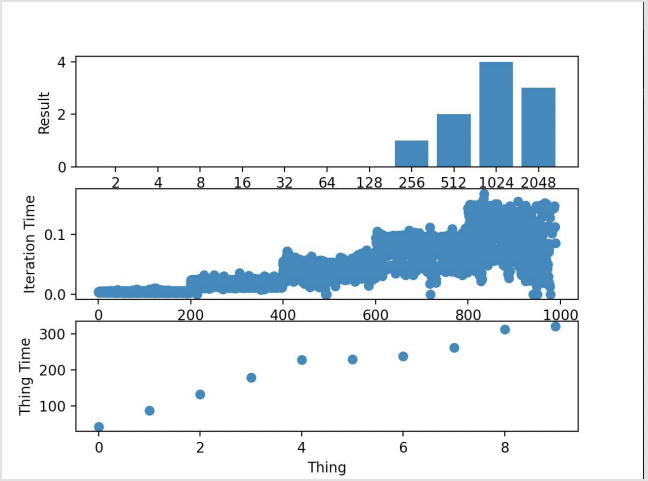
- Decide to the convergence point
- Idealise the number of searches per move and search length with respect to the move number
  - numberOfSearches = 10 * (1+floor(moveNumber / 200))
  - searchLength = 4 * (1+floor(moveNumber / 200))
- HyperParameters to tuned with a max of 200 searches and searchLenght with 4 possible moves

```
while not moveMade and len(move_order) > 0:
        move_index = np.random.randint(0,
len(move_order))
        move = move_order[move_index]
        board, move_made, score  = move(board)
        if move_made:
            return board, True, score
        move_order.pop(move_index)
    return board, False, score
```

Episode Comparison:

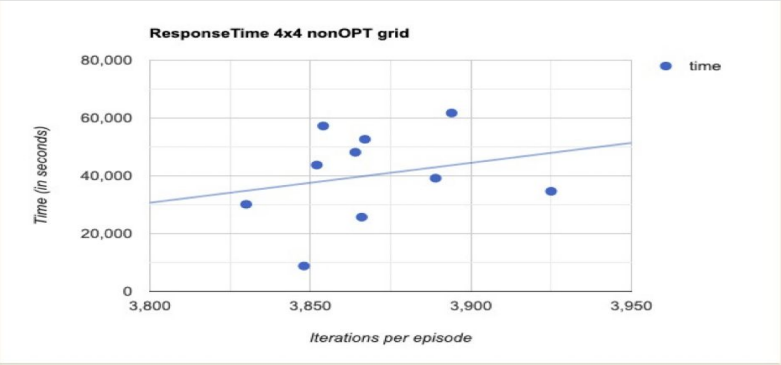


- Linear exploration episode result
- Density in plot of time iteration corresponds number of iterations performed
- Time in seconds

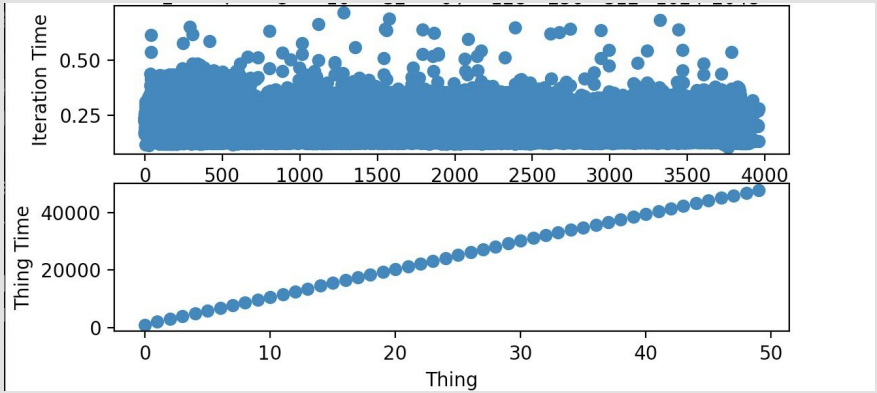


- Ineffective with number of episode results
- Time for an iteration increases as number of iterations -> in order to adapt the knowledge module.
- Thing time goes up for episodes since the knowledge module keep to growing

Self-Adaptive and Scalable Comparison:

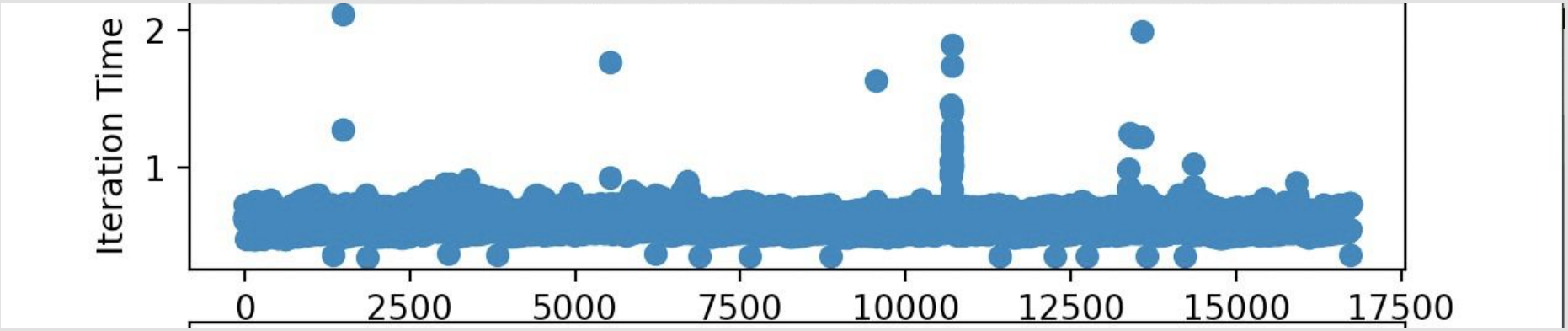


- 10 episodes of 4x4 exploration agent
- Thing time to 60,000 seconds max
- Iteration
  - (3848, 8798.442965984344)
  - (3866, 25715.07758617401)
  - (3830, 30145.487763166428)
  - (3925, 34652.23674607277)
  - (3889, 39175.38327431679)
  - (3852, 43747.438895225525)
  - (3864, 48152.35827612877)
  - (3854, 57247.99343729019)
  - (3867, 52655.974182128906)
  - (3894, 61737.6257622242)



- 50 episodes of 6x6 adaptive agent
- Thing time to 40,000 seconds max
- Iteration time to 0.75 seconds max

Scalability:



- One episode 12x12 grid to result
- Iterations: 16873
- Iteration time: 0.612 seconds

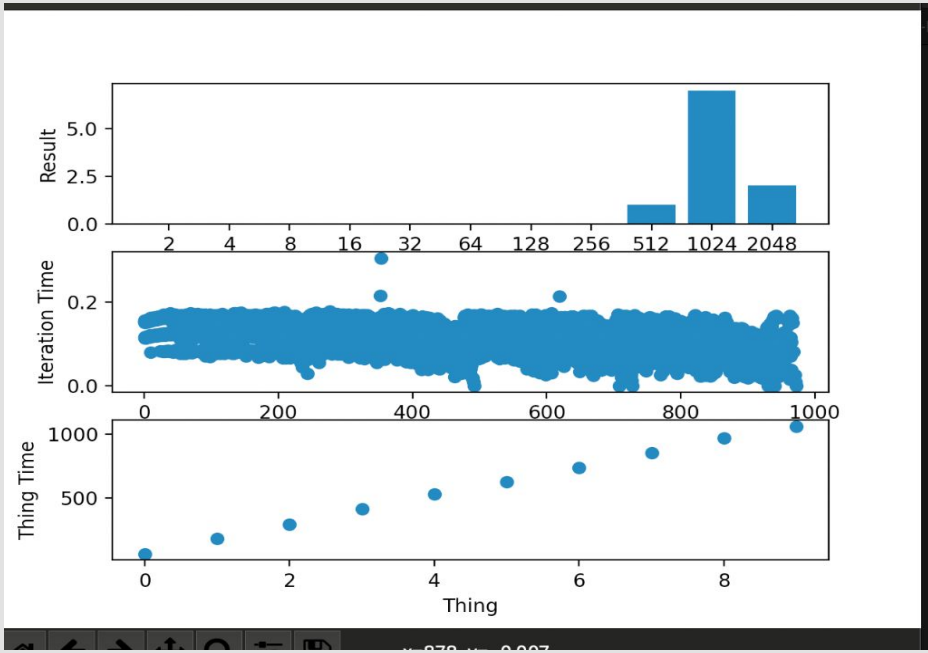
Work:

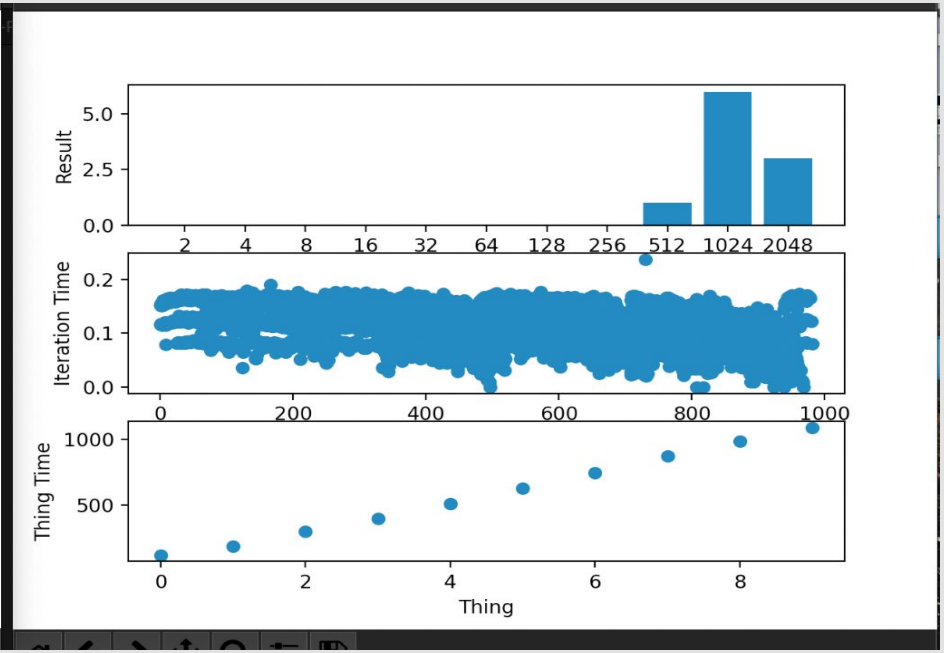
- Peer Review:
  - Scalability with the grid world! ✓
  - Result comparison ✓
  - Time to solve the problem ✓

- Future Work:
  - Optimise the knowledge module.
  - Add a convergence point for knowledge module.
  - Gain maximum score by optimising
  - Imply this MDP problem with a control architecture approach.

Discussion.!

Execution (this morning):





Graphs: Exploration VS Adaptive
AverageIteration: 828, 421