Artificial Intelligence
6 – Online MDP Solving
(2048 and Expectimax)

Arthur Bit-Monnot

INSA 4IR

Markov Decision Processes

States and actions:

- S: set of states
- $Actions(s)$: set of actions available in state s
  - a state with no available actions is a **terminal** state

Transition model:

- $P(s'|s,a)$: probability of reaching state s' after taking action a in state s
  - **markovian**: only depends on current state and action (and not on history)

Reward model:[1]

- $R(s,a,s')$: reward received after taking action a in state s and reaching state s'
  - 1 for reaching the goal, -1 for reaching the negative goal
  - -0.04 for each other transition (action cost, encouraging to reach the goal quickly)

[1] a negative reward can be interpreted as a cost

2048 as an MDP

- **States**: all possible board configurations
- **actions**: subset of {up, down, left, right} (the ones that change the board)
- **Transition model**:
  - first move all tiles in the direction of the action (merging identical tiles)
  - the uniformely select an empty time
  - place a 2 or 4 tile with respective probabilities 0.9 and 0.1
- **reward**: the value of any newly merged tile

Section 1

State Space

Game Dynamics

At each turn the sum of all tiles is incremented by 2 or 4.

- merging two tiles of value x and y gives a tile of value $x + y$ (no changes)
- a new tile is added with value 2 or 4

How much space do I need to produce a single tile?

Assume I just got the SUM 2048 (2^11)

What's the representation of those sums?

- $2048 = 2^{11} =$ 0b100000000000
  - $2048 = 2^{10} + 2^{10}$
  - $2048 = 2^{10} + 2^9 + 2^9$

Minimum number of tiles

Just before having the SUM 2048, I had either:

- the SUM 2046 $(2^{11} - 2)$
- the SUM 2044 $(2^{11} - 4)$

What the most compact representation of those sums?

- $2046 = 2^{11} - 2 =$ 0b011111111110
- $2044 = 2^{11} - 4 =$ 0b011111111100

Minimum number of tiles

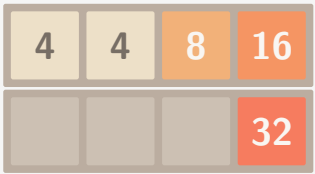
Theorem

To produce the tile 2^N, I need at least the space for $N-1$ tiles.

- $N-2$ *to represent the sum* 2^N-4
- $+1$ *free space to where a new tile (4) will be added*

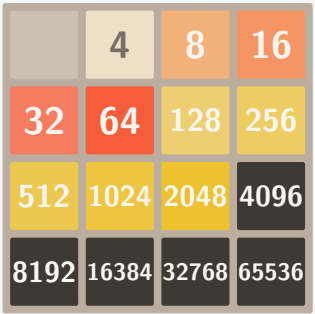
32 (2^5) requires 4 a board of size at least 4

- $32 = 16+8+4+4$
- $2^5 - 4 = 2^4 + 2^3$



Biggest tile

The board has 16 spaces, so we can in the best case produce a tile of value 2^{17} (131072).



In situation (SUM $= 2^{17} - 4$), we can either

- get a new tile of value 2
  - GAME OVER
- get a new tile of value 4
  - we will produce the tile 2^{17}

Even in the best situation above, we have 90% chance of losing the game before getting the tile 2^{17}.

Winning Conditions

What are the expectations of the players?

- humans (experienced players):
  - 2048 (common)
  - 4096 (challenging)
  - 8192 (hard)
- machines:
  - 8192 (common)
  - 16384 (challenging)
  - 32768 (hard)
  - 65536 (never seen personally[2])

[2] A very involved algorithm, with heavy offline computation for end-games, claims to reach 65536 with a probability of 5.8%

Game length

How many turns are necessary to reach the SUM 32768?

- each turn we add a tile of value 2 or 4
  - the average increment is $2.2 = 2 \times 0.9 + 4 \times 0.1$
- Reaching the SUM 32768 requires on average: 14894.5 turns (32768 / 2.2)

State space size

How many possible states are there in the game?

State space size

How many possible states are there in the game?

Having N distinct tiles on the board, how many possible states are there?

- $N = 1$: 16 possible states
- $N = 2$: 16 x 15 possible states
- $N = 3$: 16 x 15 x 14 possible states

State space size

How many possible states are there in the game?

Having N distinct tiles on the board, how many possible states are there?

- $N = 1$: 16 possible states
- $N = 2$: 16 x 15 possible states
- $N = 3$: 16 x 15 x 14 possible states

$$\frac{16!}{(16-N)!}$$

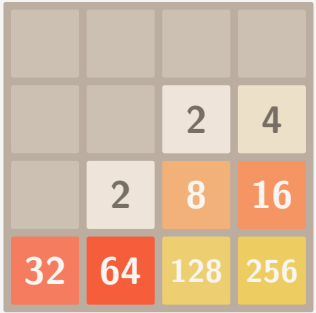
- $N = 8$: 5×10^8
- $N = 12$: 9×10^{12}

Clearly, we can't store all possible states in memory.

Section 2

State representation

State representation



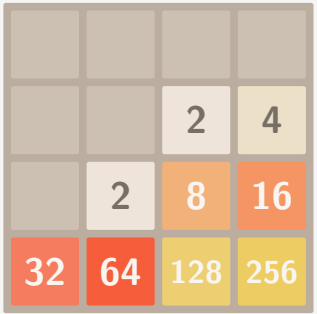
How to encode this in a computer?

- 4 by 4 matrix
- of integers

What's the number of bits necessary to store the integers?

- 2048 -> 11 bits (two bytes)
- 4096 -> 12 bits (two bytes)
- 8192 -> 13 bits (two bytes)
- 16384 -> 14 bits (two bytes)
- 32768 -> 15 bits (two bytes)
- 65536 -> 16 bits (two bytes)
- 131072 -> 17 bits (three bytes)

State representation

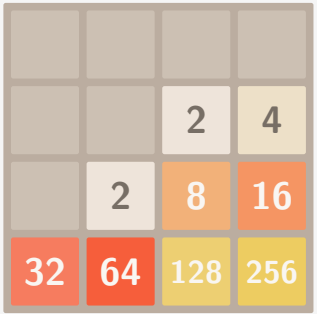


If we settle for 16 bits, we can represent up to 65536.
Total memory for the board:

- 16 bits x 16 tiles = 256 bits = 32 bytes
- 4 machine words (64 bits)

Could we do better?

State representation



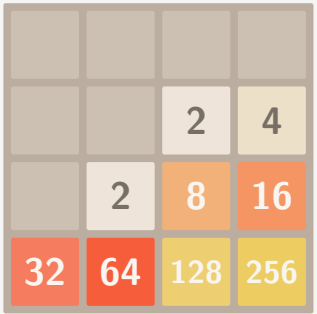
If we settle for 16 bits, we can represent up to 65536.
Total memory for the board:

- 16 bits x 16 tiles = 256 bits = 32 bytes
- 4 machine words (64 bits)

Could we do better?

- tiles are powers of 2
  - $16 = 2^4$
  - $2048 = 2^{11}$
  - $4096 = 2^{12}$
  - $65536 = 2^{16}$
- What if we stored the exponent?
  - the number x in the matrix represents the tile 2^x
  - only 4 bits necessary to store the exponent
- Memory for the board:
  - 4 bits x 16 tiles = 64 bits = 8 bytes

State representation



Yet, 4 bits are annoying to manipulate.

- a CPU is only allowed to manipulate (load/store) bytes
- this course is not about bit twiddling

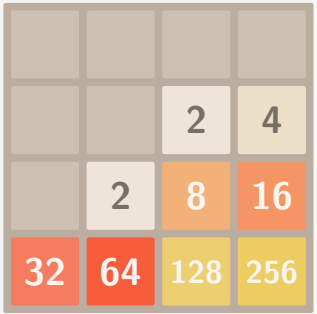
Lets use:

- one byte for each tile
- with the exponent trick

Board:

- 4 x 4 matrix
- each with one 8-bits unsigned integer (u8 in Rust)
- the number x in the matrix represents the tile 2^x
- total size of the board: 16 bytes

State representation



Encoding of the board:

```
[[0,0,0,0],
 [0,0,1,2],
 [0,1,3,4],
 [5,6,7,8]]
```

Section 3

Utility estimation

Performance measure

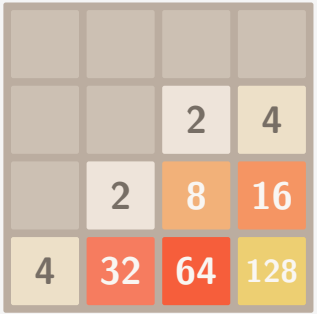
Your score is the sum of all tiles that you have produced over the entire game.

Could we estimate the score at a given state?

Performance measure

Your score is the sum of all tiles that you have produced over the entire game.

Could we estimate the score at a given state?



- score(2) = 0 (just popped)
- score(4) = 4 (assuming we merged 2 and 2)[a]
- score(8) = 8 + 2 x score(4) = 16
- score(16) = 16 + 2 x score(8) = 48
- score(32) = 32 + 2 x score(16) = = 128
- score(64) = 64 + 2 x score(32) = 320
- score(128) = 128 + 2 x score(64) = 768
- ...

[a] the real average score for a 4 tile is 4×0.9 as it would just have appear 10% of the time

Score(board) = 4 + 4 + 48 + 128 + 320 + 768 = 1272

Performance measure

Observations:

1. the score associated with a state is only dependent on the tiles on the board.

Performance measure

Observations:

1. the score associated with a state is only dependent on the tiles on the board.
2. for any configuration of the board with a large SUM, the large tiles on the board are the same.

Performance measure

Observations:

1. the score associated with a state is only dependent on the tiles on the board.
2. for any configuration of the board with a large SUM, the large tiles on the board are the same.
3. the large tiles are the ones that contribute the most to the score

Performance measure

Observations:

1. the score associated with a state is only dependent on the tiles on the board.
2. for any configuration of the board with a large SUM, the large tiles on the board are the same.
3. the large tiles are the ones that contribute the most to the score
4. the SUM increases linearly with the number of turns

Performance measure

Observations:

1. the score associated with a state is only dependent on the tiles on the board.
2. for any configuration of the board with a large SUM, the large tiles on the board are the same.
3. the large tiles are the ones that contribute the most to the score
4. the SUM increases linearly with the number of turns

$\Rightarrow$ the number of turns is an almost perfect proxy for the score

- our performance measure: the number of turns played

Solved?

$$\mathit{perf}(b) = \frac{\mathit{SUM}(b)}{2.2}$$

$$U(b) = \begin{cases} \mathit{perf}(b) & \text{if } terminal(b) \\ \max_a \sum_{b'} P(b'|b,a) \times U(b') & otherwise \end{cases}$$

Solved?

$$\mathit{perf}(b) = \frac{\mathit{SUM}(b)}{2.2}$$

$$U(b) = \begin{cases} \mathit{perf}(b) & \text{if } terminal(b) \\ \max_a \sum_{b'} P(b'|b,a) \times U(b') & otherwise \end{cases}$$

Intractable, requires exploring all possible state sequences until a terminal state

Utility estimation vs Performance measure

What should the utility function estimate?

Utility estimation vs Performance measure

What should the utility function estimate?

- NOTHING!
- it should be such that the action with the best expected utility is the one with the best expected performance measure
- only a mean to **compare** states

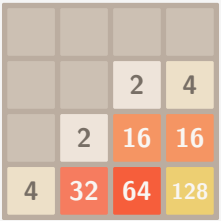
Utility estimation vs Performance measure

What should the utility function estimate?

- NOTHING!
- it should be such that the action with the best expected utility is the one with the best expected performance measure
- only a mean to **compare** states

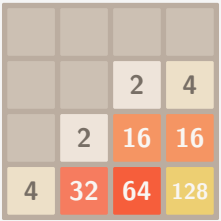
Utility function: combination of numerical values **correlating** with the *life expectancy*

Utility estimation



Some simple metrics:

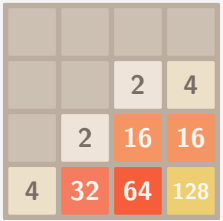
Utility estimation

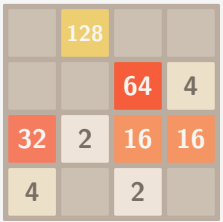


Some simple metrics:

- number of empty tiles
- number adjacent tiles with the same value

Utility estimation

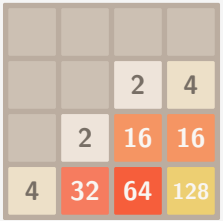




Some simple metrics:

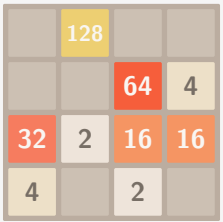
- number of empty tiles
- number adjacent tiles with the same value

Utility estimation



Some simple metrics:

- number of empty tiles
- number adjacent tiles with the same value



Need to account for:

- the structure of the board
- emphasizing large tiles

Monotonicity (naive counting)

monotonicity: count the number of increases and decreases in the sequences

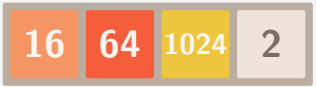
- increases: 3
- decreases: 0
- monotonicity penalty: 0 (min)

- increases: 2
- decreases: 1
- monotonicity penalty: 1 (min)

Differential Monotonicity

Weighting: to emphasize the importance of large tiles[3]

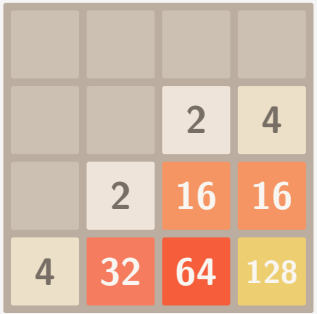
- increases: $(4-2)+(16-4)$
- decreases: $(16-8)$
- monotonicity penalty: 8 (min)



- increases: $(64-16)+(1024-64)$
- decreases: $(1024-2)$
- monotonicity penalty: 1022 (min)

[3] Slide presents the main idea but implementations may more complex

Monotonocity (board)



Monotocity measure for the board:

- sum of the monotonicity measure for each row and column
- favors placing large tiles in a corner

Utility measure

$$\begin{aligned}\hat{U}(b) = w_1 &\times \textit{empty-tiles}(b) \\ + w_2 &\times \textit{monotonicity}(b) \\ + w_3 &\times \textit{adjacent-tiles}(b)\end{aligned}$$

How to set the weights w_1, w_2, and w_3?

- trial and error (by hand)
- trial and error (optimization algorithm)[4]
- machine learning (reinforcement learning)

[4] Someone did it for us https://github.com/nneonneo/2048-ai

Section 4

Algorithms

Structure of a turn

A turn is composed of:

- the application of an action a, selected by the agent
  - $s' \leftarrow result(s, a)$
- the addition of a random time
  - $s'' \leftarrow add\_tile(s')$ with probability $P(s''|s')$

Greedy

- **Greedy algorithm**: at each turn, choose the action that maximizes the approximated utility of the resulting state

$$\textit{greedy}(s) = \arg\max_a \hat{U}(result(s, a))$$

Complexity: $O(1)$ (at most 4 evaluations of the utility function)

Greedy

- **Greedy algorithm**: at each turn, choose the action that maximizes the approximated utility of the resulting state

$$\textit{greedy}(s) = \arg\max_a \hat{U}(result(s, a))$$

Complexity: $O(1)$ (at most 4 evaluations of the utility function)

Surely we can do better than that!

From Bellman to Expectimax

Recall the Bellman equation:

$$U(s) = \max_{a \in Actions(s)} \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma U(s') \right]$$

For 2048, we can simplify it (no rewards nor discounts)

$$U(s) = \max_{a \in Actions(s)} \sum_{s'} P(s'|s,a) U(s')$$

From Bellman to Expectimax

Expectimax: approximates the gathering the average utility of the states reachable with a fixed number of actions.

Utility estimated by EXPECTIMAX, with lookahead (depth) d

$$U_{EM}(s,d) = \begin{cases} \hat{U}(s) & \text{if } d = 0 \\ \max_a \sum_{s'} P(s'|s,a) U_{EM}(s', d-1) & otherwise \end{cases}$$

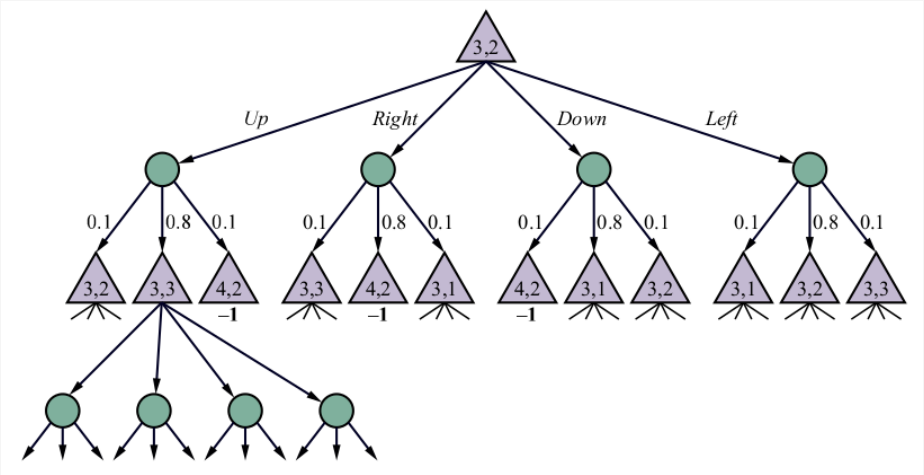
Expectimax algorithm

$$expectimax(s, d) = \arg\max_a eval\_chance(result(a), d-1)$$

$$eval\_chance(s) = \begin{cases} \hat{U}(s) & \text{if } d = 0 \\ \sum_{s'} P(s'|s) \times eval\_max(s', d-1) & otherwise \end{cases}$$

$$eval\_max(s, d) = \max_a eval\_chance(result(s, a), d-1)$$

Expectimax: tree



Expectimax: complexity

- Time complexity (size of the explored tree): $O(b^d)$
  - b: branching factor (number of actions x number of possible new tiles)
  - d: depth of the tree (number of turns)
- Space complexity: $O(d)$ (depth of the tree, like DFS)

Expectimax: complexity

- Time complexity (size of the explored tree): $O(b^d)$
  - b: branching factor (number of actions x number of possible new tiles)
  - d: depth of the tree (number of turns)
- Space complexity: $O(d)$ (depth of the tree, like DFS)

Typical values for 2048:

- 3 available actions
- 6 empty tiles (less in late game)
  - 12 possible new tiles (2 or 4)
- $b = 3 \times 12 = 36$ (a lot of variations across the game)

Expectimax: complexity

- Time complexity (size of the explored tree): $O(b^d)$
  - b: branching factor (number of actions x number of possible new tiles)
  - d: depth of the tree (number of turns)
- Space complexity: $O(d)$ (depth of the tree, like DFS)

Typical values for 2048:

- 3 available actions
- 6 empty tiles (less in late game)
  - 12 possible new tiles (2 or 4)
- $b = 3 \times 12 = 36$ (a lot of variations across the game)

Explored states:

- $d = 1$: $36^1 = 36$
- $d = 2$: $36^2 = 1296$
- $d = 3$: $36^3 = 46656$
- $d = 4$: $36^4 = 1679616$
- $d = 5$: $36^5 = 60466176$

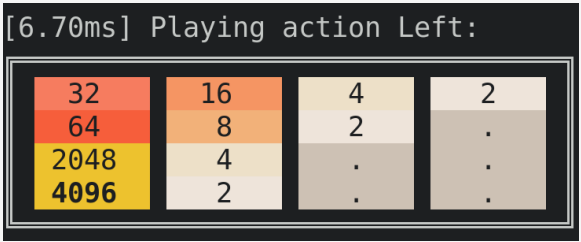
Next week: 2048 lab

Given:

- game logic
- utility function implementation
- algorithm evaluation code

To do:

- implement the greedy algorithm
- implement the Expectimax algorithm
- evaluate the performance of the algorithm
- improve and compete?



Going further

- **transposition table**: caching the results of the eval_max/eval_chance functions
  - exploit redundancy in the search tree
- **iterative deepening**: gradually increase the depth of the search
  - deeper search on states with lower branching factor (typical in end games)
- **approximation**: early termination the search
  - on very unlikely states (e.g. we got 6 tiles of value 4 in a row)
  - on states with a very low utility (e.g. recovery is unlikely and this branch will never be selected)
- **heuristic**: tuning the heuristic utility function
  - offline by hand or optimization (has already been done for you)
  - online with reinforcement learning (out of scope for this course)
    - The heuristic is, e.g., a neural network that is trained while playing the game[5]

[5] https://arxiv.org/abs/1604.05085