Report for final project

Video link: https://youtu.be/gZeJ1pgTIuo

Problem description

The project we did was to develop a game playing agent to have a decent performance in the game 2048.

In general, we used adversarial search. The AI simply performs maximizer over all possible moves, followed by expectation over all possible tile generations. The problem is formulated as follows:

- **States**: A state description specifies the value of each of tiles on the board. The board has 16 tiles and each tile can be empty, 2, 4, 8, 16, 32, 64, 128, 256, 512 and 1024. Plus, the termination state is that one tile on the board is value 2048. Thus, there are $16 * 15^{12} + 16^{12}$, namely $16 * 15^{12} + 2^{48}$ possible board states. Note that this size of state space is an upper bound.
- $\boldsymbol{s_0}$: The initial state. Random two tiles are assigned with either 2 or 4. In our project, 90% is for a 2 and 10% for a 4.
- **ACTIONS(s)**: Each state s has four actions: Left, Right, Up and Down.
- **RESULT(s, a)**: The transition model. Given a state and action, this returns the resulting state: Tiles on the board slide as far as possible in the direction as action indicates until they are stopped by either another tile or the edge of the grid. Two tiles of the same number collide while moving, they will merge into a tile with the total value of the two tiles that collided. The resulting tile cannot merge with another tile again in the same move.
- **TERMINAL- TEST**: This checks whether there is a tile on the board with value of 2048
- **UTILITY(s, p)**: This defines the final numeric value for a game that ends in terminal state s for a player p.

In perspective of computation, the input of the algorithm is a state and the output is an action.

Approach

In 2048 problem, what we need to decide is the move for every step. Generally speaking, we have four choices which are up, down, left and right. However, sometimes one or more options may not be valid, this is because these actions cannot make any change on the 4*4 grid. In this case, we only treat the moves which can change the 4*4 grid as legal actions.

Every time we act a legal action, a new tile with value 2 or 4 will be generated in random empty spots on the board. This is just like we are playing game against another player. The job of the other player is randomly spawning tiles to fill the grid and make us not able to achieve 2048 (game over). So the solution of the game is quite similar to other classic games such as Chess or Tic-tac-toe, which apply adversarial search.

Now we need to consider which adversarial search fits this game best. Apparently, we cannot apply minimax search on this game, because the other player doesn't play optimally. On the contrary, we can find the other player actually makes a move randomly, hence this situation can be handled by an extension to the minimax algorithm that evaluates the other player's move by taking the average utility of all its children (possible moves), that is, the expectimax. Usually, it is not feasible to consider the whole game tree, so we need to cut the search off at some point and apply a heuristic evaluation function that estimates the utility of a state.

According to the analysis mentioned above, we are using the following algorithm: Every time to find the optimal move to achieve the goal, we calculate the score of each legal actions, and get the one with maximum score. When calculating the score of one action, it is just like the other player's turn playing game. The other player may spawn a new tile with value 2 or 4 on any empty cell of the grid. So we just calculate the average scores of all possible moves weighted by their probabilities. This is how we perform expecti. To calculate the score of every possible moves of the other player, we apply a Max value function, which calculates the score of every actions and get the max score. Expecti and Max parts run in an interleaving way and the depth will be increased by one after finishing any parts. When the depth reaches the maximum given depth which is a constant number, the actual score will be calculated by an evaluation function.

Below is pseudo code of our Expecti-Max Search:

```
function EXPECTI-MAX-SEARCH(state) returns an action
    return arg max$_{action \in \text{ACTIONS(s)}}$ EXPECTI-VALUE(RESULT(state, action))
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  $v \leftarrow -\infty$
  for each action in ACTIONS(state) do
      $v \leftarrow$ MAX($v$, EXPECTI-VALUE(RESULT(state, action)))
  return $v$
function EXPECTI-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  $v \leftarrow 0$
  for each action in ACTIONS(state) do
      $v \leftarrow v$ + MAX-VALUE(RESULT(state, action)) * PROBABILITY(action)
  return $v$
```

It should be clear that the performance of a game-playing program depends strongly on the quality of its evaluation function. An inaccurate evaluation function will guide the agent toward positions that turn out to be lost. In this case, to develop the game agent, we come up with three strategies which engage ones that a good human player usually apply. What we are doing is trying to get to the states we believe it's optimal. To evaluate a state whether good or not, we have several rules below:

First, bigger valued tiles should be clustered in a corner. It will typically prevent smaller valued tiles from getting orphaned and will keep the board very organized which can ensure that the values of the tiles are all either increasing or decreasing along both the left/right and up/down directions.

Second, the value difference between adjacent tiles should be as small as possible. By minimizing the difference, we can try our best to let tiles with same value be adjacent so that they can be merged in the future.

Third, the number of free tiles should be as large as possible. We know that the game would be over when the whole board is occupied with no tiles can be merged. So we should prevent options from running out quickly by maximizing the number of free tiles.

According to the three strategies mentioned above, we come up with a weighted linear evaluation function. Mathematically, it can be expressed as

$$\mathrm{EVAL(s)} = w_{corner} * f_{corner} + w_{adjacency} * f_{adjacency} + w_{empty} * f_{empty}$$

where each w is a weight and each f is a feature of state s.

Results

To quantitatively characterize how the algorithm worked, we ran tests in the following way:
Test arguments include:

- *Terminal depth*
  DEPTH = 5, when the number of empty cells is more than 9
  DEPTH = 6, when the number of empty cells is between 5 to 9
  DEPTH = 7, when the number of empty cells is less than 5
  Note that the depth starts from 0.
- *Weights of each evaluation feature*
  $W_{corner} = 1.0$, which weighs the corner feature
  $W_{adjacency} = 2.0$, which weighs the adjacency feature
  $W_{empty} = 3.0$, which weighs the number of the empty cells feature

Under these circumstances, 95 out of 100 times the game agent can reach the 2048 tile successfully. The win rate is 95% with an average score of 20308, which is satisfying though not perfect. As for the rare fails, the reason is that the randomly

spawned tile spawned in an extremely bad position which makes our game agent have no option to avoid game over.
To get the number of nodes expanded, it is necessary to take both depth and the number of empty tiles into consideration. However, since the depth depends on the number of empty tiles, we treat different number of empty tiles as different scenarios and approximately calculate the expanded nodes. The result is illustrated in the table below:

The number of empty cells	Depth	Expanded node
14	5	14400
13	5	12544
12	5	10816
11	5	9216
10	5	7744
9	6	64000
8	6	46656
7	6	32768
6	6	21952
5	6	13824
4	7	32000
3	7	16384
2	7	6912
1	7	2048
0	7	256

Conclusion

How to develop a game agent to solve 2048(video game) is presented in this report. The algorithm applied is adversarial search, more specifically, expectimax. Besides, evaluation function is used to approximate the true utility of a state without doing a complete search so that the efficiency can be improved to some extent. The win rate 95% shows that the game agent performs well, but we believe there is still room for improvement, e.g. the weights in evaluation function.

References

2048(video game). (n.d). Retrieved from https://en.wikipedia.org/wiki/2048_(video_game)
Optimal Algorithm for the game 2048. (n.d.). Retrieved from http://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048
"5 Adversarial Search." Artificial Intelligence - A Modern Approach. 3rd ed. New Jersey: Pearson Education, 2010. 180-208. Print.