COMP30024 Artificial Intelligence

Project: Part A

03 April 2022

Leo Xinqi Yu, Mengyuan Huang

Data Structure:

i. Cell unit: The basic unit here is a class named Cell, that essentially is a linked list that storing the pointer of a cell's parent cell, for backtracking purpose.

```python
# Author: Leo
# a cell
class Cell():
    def __init__(self, im_red=None, coord=None, parent=None, g_val=0, h_val=0, f_val=0, shown_as=None):
        self.coord = coord
        self.parent = parent
        self.g_val = g_val
        self.h_val = h_val
        self.f_val = f_val
        self.im_red = im_red
        self.shown_as = shown_as
        self.is_empty = False
```

ii. Frontier: The frontier is implemented with a plain array, using its append() and pop(index to the best-cost cell) to simulate the behaviour of a priority queue.

iii. Seen/Checked: The seen and checked cells are stored as well in a plain array, with in() checking a cell's existence.

Complexities:

i. Time complexity:

Best case: The best case occurs when no obstacles/enemies exist in the board, or no obstacles/enemies are placed on the shortest path of the no-obstacle condition. In this case, the heuristic error is zero, where leads to a time complexity of $O(bd)$ [1].

Worst case: The worst case is brought by having a misdirecting heuristic value in every step. It ends up with searching all cells who has an equal or smaller depth than the final path, i.e., $O(b^d)$.

Average case, considering how a Cachex game usually develops, the heuristic error in each round gets larger, in general, since obstacles are kept building by the opponent. However, the empty cells (total search space) are linearly decreasing, hence the increasing time complexity will only cause a limited effect.

ii. Space complexity:
A* keeps all explored cells in the memory, the big-O expression is the same as above (corresponding time complexity). In this particular case, Cachex. Since the map is a

[1] In the complexity section: b = the branching factor, particularly, 6. d = the depth of the search path.

regular hex map, map information can be generated (and checked with reachability) on-the-fly. Hence it saves the effort of saving all cells of the map in memory.

Heuristic:

Manhattan (L1) distance. It is admissible since it describes the shortest path between two cells, calculated as $h(C_1C_2) = |q_1 - q_2| + |q_1 - q_2|$. Time complexity $O(1)$. Due to the succinctness of the L1 calculation, the global time complexity is reduced to the minimum based on the concurrent searching strategy (in Complexity section).

Challenge:

Now we push the new condition introduced from the ‘Challenge’ section into consideration. For effectively utilising the existing pieces in the board, a modification is added to the stage of calculating the g value when pushing a cell’s neighbours into the frontier. When expanding a cell, if a neighbour is of the same colour, then it’s taken as a stepping stone with its g value (current path cost) assigned same as its parent cell (instead of +1), heuristic value and f value (cost) are calculated as usual.