CSC384H1F

Assignment 3: CSP Battleship

Junrui Chen - chenj591

Due: November 3, 2022

Heuristic Function

Code. The following is my code for my heuristic function and how to use the heuristic value:

```python
class Constraint:
    """A constraint that some variables should follow.
    About the number of grids assigned for each row/column.

    === Attributes ===
    scope:
        A set of the coordinates of variables that this constraint is over.
    r_or_c:
        Row or column this constraint is for.
        0 for row and 1 for column.
    index:
        Which row/column this constriant is for.
    value:
        The number of grids having ship in this row/column.

    === Representation Invariants ===
    - r_or_c == 0 or r_or_c == 1
    """
    scope: set[tuple[int]]
    r_or_c: int
    index: int
    value: int

    def __init__(self, N: int, r_or_c: int, index: int, value: int) -> None:...

    def __str__(self) -> str:...

    def difference(self) -> int:
        """The difference between the value and the size
        of scope of this Constraint."""
        return len(self.scope) - self.value
```

```python
class priority_queue:
    """A priority queue ordered by difference between the value and the size
    of scope of a Constraint.

    Stores data ordered by the difference. When removing an item from the
    priority queue, the item with the smallest differnece is the one that
    is removed.

    === Private attributes ===
    heap:
        The items stored in this priority queue, which is a min-heap. Each item
        stored as a tuple, where the first element is the difference and the
        second element is a tuple reprensenting a Constriant.
    """
    lst: list

    def __init__(self) -> None:...

    def is_empty(self) -> bool:...

    def enqueue(self, diff: int, c: tuple) -> None:
        """Add a new element to the priority queue."""
        heapq.heappush(self.lst, (diff, c))

    def dequeue(self) -> Optional[tuple]:
        """Remove and return the element the priority queue."""
        if self.is_empty():
            return None
        else:
            return heapq.heappop(self.lst)[1]
```

```
410     def PickUnassignedVariable(self) -> tuple[tuple[int], tuple[int]]:
411         mrv = priority_queue()
412         for i in self.constraints:
413             mrv.enqueue(self.constraints[i].difference(), i)
414         c = self.constraints[mrv.dequeue()]
415         while len(c.scope) == 0:
416             c = self.constraints[mrv.dequeue()]
417         v = c.scope.pop()
418         c.scope.add(v)
419         return ((c.r_or_c, c.index), v)
```

Description. Since the type of propagation for my CSP solver is Forward Checking, I use Minimum Remaining Values Heuristics to help for tracking.

Each constraint is set as a limitation of the number of grids containing (parts of) ships in the corresponding row/column. Thus, the heuristic function is the size of scope for a constraint minus this limit. The function Constraint.difference() is used to compute the heuristic value.

Obviously, the smaller the heuristic value is, the less times we need to take to to choose grids which will be padded with ships, especially when the heuristic value is 0.

Use a priority queue which is a min-heap to store tuples that can represent each constraint, ordering by the heuristic value.

Every time State.PickUnassignedVariable() is called, a new priority queue is built based on the current set of constraints. Then it takes the one with lowest heuristic value, return the tuple representing this constraint and random variable in the scope of this constraint. If the scope of this constraint is empty, it continues to dequeue constraints until there exists variables in the scope of the constraint it just dequeued.