Planning paths for RISK: Uniform Cost Search

This assignment involves planning paths in the game of RISK. A path in this context is a sequence of territories that an attacking player could conquer in order to occupy a target territory. Such a path will start at a territory that is owned by the attacking player and then proceeds through a sequence of territories owned by opposing players that ends at the target territory. A minimum cost path will correspond in some manner to the difficulty of conquering the entire sequence of territories. In this assignment you will get to define your own cost function for these paths. You should first download the `RISK-AI.zip` file and unzip in on your computer. The provided code has been tested with Python 3.

The folder you are provided contains (in addition to files for the final RISK AI assignment, which are described in that handout):

- `RISK_Search_Handout.pdf` - This file that you are reading
- `risk_search.py` - This is the file that you will modify to implement the search function. It is provided a logfile and a state index, as well as source and destination territories. It will then run the search and print out the resulting path. It can be run by typing:

  ```
  python risk_search.py logfile state_index source_territory
                        destination_territory
  ```

  from the command line. The source territory must belong to the player whose turn it is to act in the give state, while the destination territory must belong to an opponent.

Part 1 [60 points]

The first part of this assignment is to complete the code so uniform cost search can plan a path from between the given territories. Starter code has been provided with the assignment

- **Task 1 [15 points]** Implement the `get_successors( )` function. You can find where to do this by searching for "Task 1.1" in the code. This function takes in a territory id number and the current state and needs to return a list of all of the neighboring territory id numbers that are not owned by the current player in the given state. Helpful tips for this:

  - A list of the territories that border territory number `t` are stored in `state.board.territories[t].neighbors`
  - The player id of the player that owns territory `t` in a state is given by `state.owners[t]`
  - The current player in a state is given by `state.current_player`

- **Task 2 [45 points]** Implement the main loop of the uniform cost search function. You can find this by searching for "Task 1.2" in the code. In this loop you need to:
  - Extract the lowest cost node from the fringe (use `fringe.get( )`)
  - Check to see if this node's state is the goal of the search. To do this compare the `cur_node.id` to the goal. These are integers that indicate a territory number. If it is the goal, return the node.
  - Add this node's id to the list of expanded states.
  - Then get all of the successors of this node's state (use the `get_successors( )` function that you implemented in Task 1.)
  - For each successor, make sure it already hasn't been expanded, and then create a new `SearchNode` object which has this successor as its id. Pass 1 in as the step cost.
  - Add this new node to the fringe. To do this use `fringe.put( node )`.

Part 2 [20 points]

In this part of the assignment you will evaluate your code and make sure it is working. To do this we first need some game states that we can plan paths in

To do this we will do the following:

- Use the `play_risk_ai.py` to play a match between two different agents. We suggest trying matches between different combinations of random_ai, donothing_ai, and attacker_ai, which are provided in the `ai` folder. This script will automatically perform a match of specified length between the specified AI's and optionally save the game log file. To run the script you should type:

  ```
  python play_risk_ai.py -w ai\random_ai.py A ai\random_ai.py B
  ```

  This will play a 5 game match between two random_ai agents, named A and B. The logfiles will be written out (`-w` option) to the `matches` folder. To run a match between different agents you can simply change the random_ai script path to point to another agent.

- Then watch the logs using `risk_game_viewer.py` - This is the script that allows you to view games played using the `play_risk_ai.py` script. It just needs to be passed in the full path to the logfile. Run it by typing:

  ```
  python risk_game_viewer.py LOG_FILE
  ```

from the command line.

- When you see a state that has potential paths that could be planned, pause the viewer, then take a screenshot. Notice the state number in the display, you will need to pass this into the search program. Then get the names of the two territories (make sure you notice who the current player in the state is).
- Then run your search code and see if it produces the lowest cost path. The provided code initially simply counts the number of territories that must be passed through to get the cost.
- Include in your write-up several examples, with screenshots of the state and paths found, to demonstrate that your code is working.

Part 3 [20 points]

In this section you will modify the cost function that is used in the search. You will do this by modifying the code you wrote in Task 2 above. We would like the cost of the path to reflect in some manner the difficulty of conquering that path. Come up with a method of measuring this cost, explain how you chose it in your write-up, and modify the code to calculate and us it (instead of the step cost of 1).

Find at least two states where the planned path is different using the two cost methods and show what paths were planned and discuss which you think might be better for a RISK agent.

You are required to hand in the following for this assignment:

1. Your code with the required parts implemented or changed.
2. Required screenshots and discussion showing that the code is working correctly for the different parts of the assignment.
3. At least one substantial paragraph describing what you learned from these experiments, and how you could use this in a RISK AI.

1 Extra credit [20 points]

Modify the code to find the best possible starting territory for reaching the destination (in terms of minimum cost). Turn in your code and a description of how you did this.