Assignment 2 - Simple agent and environment simulator for game of RISK

Implementation:

Phase1:

We implement the environment simulator, and several simple (non-AI) agents

Our environment simulator consists of Main Classes

- **Territory Class:**
  - An object that represents each vertex(country) in the Risk Map
  - Each object has an identifier , number of armies , owner Agent , List of neighbor Territories and the Continent its belongs to (Partition cell)

- **Continent Class**
  - An object that represents each Partition Cellin the Risk Map
  - Each object has an identifier , Value and List of Territories as its Members that belongs to it

- **Semi-Continent Class**
  - An object that represents how many territories Some agent needs to have a continent
  - Each object and an integer represent the difference between the owner semi-Continents and the continents
  - You can add and removes territories as the attacking turns goes

- **InputReader Class**
  - An object that used to read the input and build the environment using the above Classes

several simple (non-AI) agents

- Agent
    - An Object that represent each agent
    - Has an identifier, Continents if exists, territories, semi-Continents, bonus armies to be place next Turn if exists and enemy that it plays against
- A human agent
    - Inherits from Agent Class
    - read the next move from the user.
    - Place bonus as user input too

- A completely passive agent
    - Inherits from Agent Class
    - it never attacks, and always places all its additional armies on the vertex that has the fewest armies, breaking ties by favoring the lowest-numbered vertex.

- An aggressive agent
    - Inherits from Agent Class
    - always places all its bonus armies on the vertex with the most armies, and greedily attempts to attack so as to cause the most damage - i.e. to prevent its opponent getting a continent bonus (the largest possible).
    - Always a set of possible attacks is created and we can check which will cause the most damage as it attack with all armies except for one

- A nearly pacifist agent
    - It places its armies like the completely passive agent, then conquers only one vertex (if it can), such that it loses as few armies as possible
    - It attack with a territory with max armies with least attacking armies

Phase 2:

Implementing an intelligent agent that plays against a completely passive agent, and attempts to win in as few turns as possible.

Using a heuristic evaluation function such that :

Heuristic = total value of continent that the agent didn't own yet + total number of armies that the opponent owns + the current bonus of opponent.

We implement 2 Classes that is used in all AI Agents

- **AgentState Class**
  - An object that represent the Search State in the agent
  - Each object has 2 agents, heuristic function, parent state to be able to form the path after reaching the goal state and an attack object
  - Most important Functions:
    - getNeighbors(): is used to calculate the neighbor states depending upon the All Possible Attacks that the agent can do

- **Attack Class**
  - An object that represent the attack process and it contains of the Attacking agent Territory and the Attacked enemy Territory and the Attacking armies that is equal to the Attacked + 1
  - Such that each opposing player loses (#enemy Armies) and the attacking player must move at least 1 army to the enemy Territory , but must leave at least 1 army in his Attacking Territory

AI Agents:

- A greedy agent

    - picks the move with best immediate heuristic value discussed above using the h(x) function
    - it Places the bonus armies such that to make the most damage as to prevent the passive agent from holding the largest valued continent until next turn if it is not the case it tries to make the most damage by placing the armies in the territory that has the higher armies so it can attack all its neighbors as the passive agent is never attacking
    - Build the path by using a stack of AgentState and GetNeighbors function try to trying to do few turns as possible.

- An agent using A* search
    - Same strategy in placing the armies as greedy
    - Same strategy in build the path but using the function h(x)+G(x)(that represents the depth of a State in the path)

- An agent using real-time A*.
    - Same strategy in placing the armies as greedy
    - Same strategy in build the path but using the function h(x)+G(x)(that represents the depth of a State in the path)
    - Goal to reduce the A* Algorithm execution time
    - Limits the search horizon of A* and select an action in constant time.
    - Make decision about next move in real-world without a complete plan (path) to reach goal state
    - We set a Search depth of 5 States to perform the action while pruning the whole state that has f(x) larger than f(CurrentState) and return the calculate path then start performing A* Search again using Current state as starter State

Performance measure:

- Astar

  when f = 1 ,P = 10

  when f = 100 ,P = 505

  when f = 10000 ,P = 550005

- RTAstar

  when f = 1 ,P = 10

  when f = 100 ,P = 505

when f = 10000 ,P = 550005

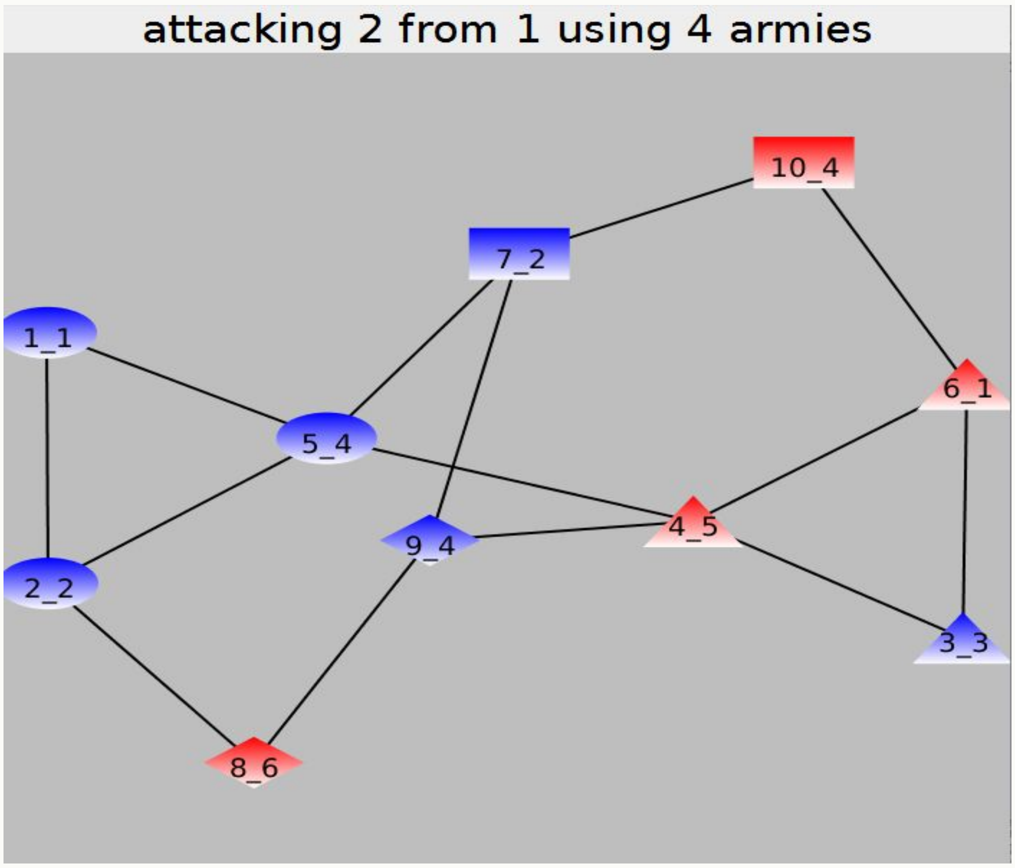
- Greedy

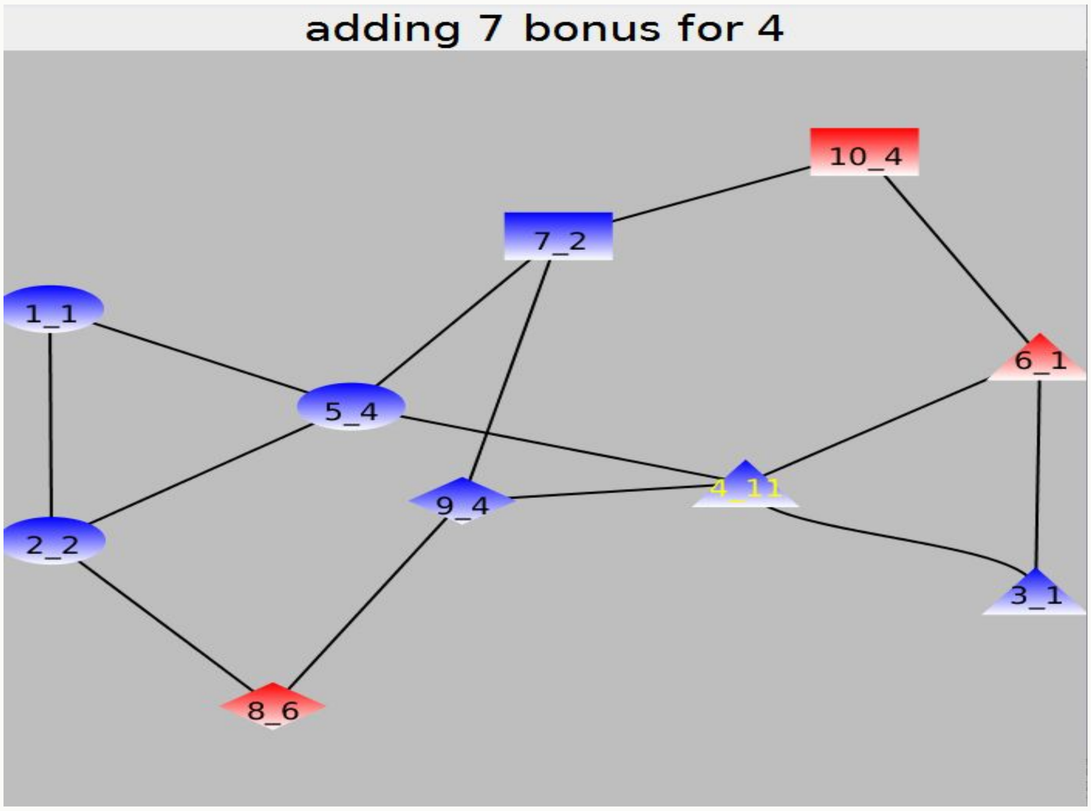
when f = 1 ,P = 10

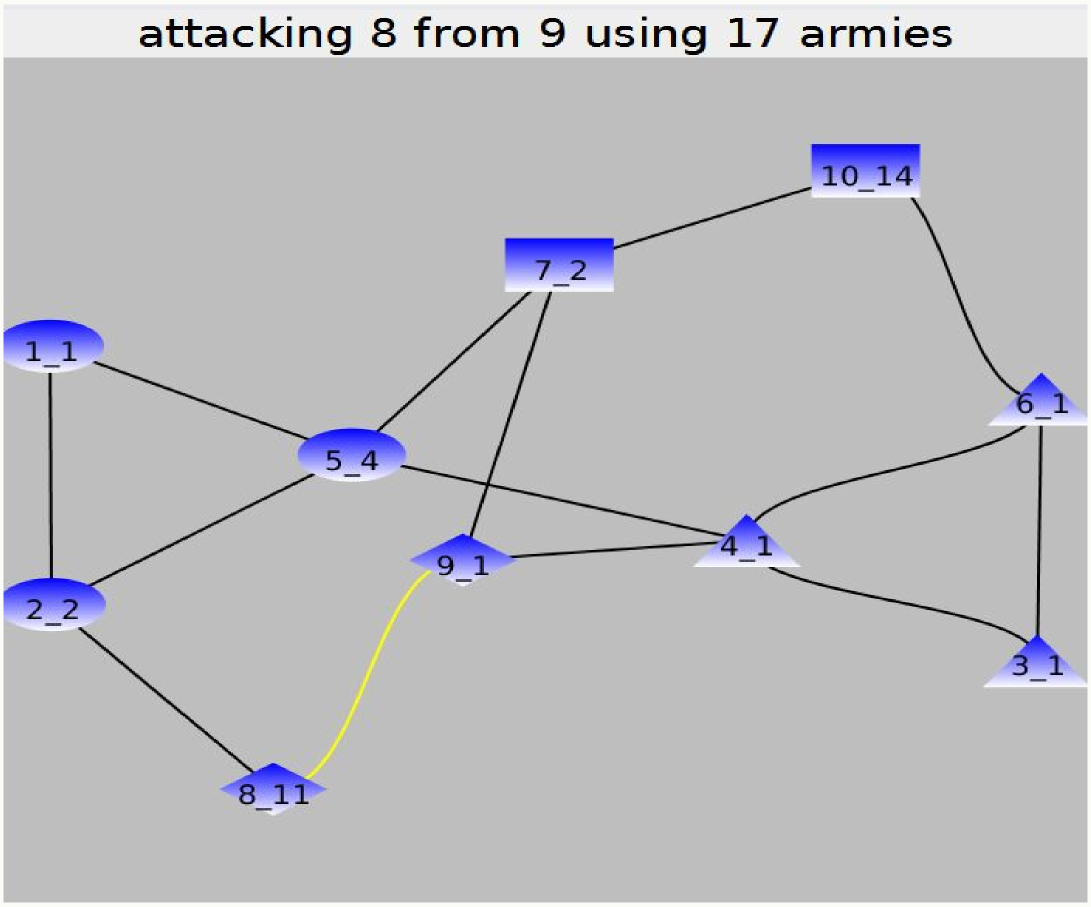
when f = 100 ,P = 505

when f = 10000 ,P = 550005

Screen shots:


attacking 2 from 1 using 4 armies


adding 7 bonus for 4


attacking 8 from 9 using 17 armies