**The semi-constants I chose to modify**

1. “bBarracksNum” in the “Building State, BuildingBarracks” function
2. “bRefineriesNum” in the “Building State, BuildingRefineries” function
3. “bGoldNum” in the “Building State, GatherGold” function
4. “bArchersNum” in the “Building State, BuildingArchers” function
5. “bSoldiersNum” in the “Building State, BuildingSoldiers” function
6. “bWorkersNum” in the “Building State, BuildingWorkers” function
7. “bAArchersNum” in the “Building State, AttackingWithArchers” function
8. “bASoldiersNum” in the “Building State, AttackingWithSoldiers” function
9. “aArchersNum” in the “Attacking State, BuildingArchers” function
10. “aSoldiersNum” in the “Attacking State, BuildingSoldiers” function

**My heuristics**

1. int myArmy = myArchers.Count + mySoldiers.Count + myWorkers.Count
2. int enemyArmy = enemyArchers.Count + enemySoldiers.Count + enemyWorkers.Count
3. int totalArmy = myArmy – enemyArmy
4. int totalWins = AgentNbrWins - agent.AgentNbrWins

**How I implemented the Hill Climb technique**

* The way I implemented the hill climb technique was that I found the difference in the previous army sizes verses the current army sizes of both agents and I would find the difference between the current number of wins for both agents. After this I would store that round’s army size difference and agent wins difference. Then, I would see if that round’s army size difference and agent wins difference is bigger than the previous. If this was the case for both calculations, then that would count as a win. If either was less than, then it would count as a loss and all the semi-constants would be evaluated based off certain parameters and changed accordingly. For example, the “bRefineriesNum” variable would only change if the number were anything that is not the desired number of refineries, which is 3.

**How I implemented the random restart technique**

* The random restart would occur based off specific rounds that the AIs are on. I chose round 25, 50, and 75 when running 100 rounds. On these rounds, it would randomly select a semi-constant to reset back to zero. This allows for a randomness to my AI to see if one of the semi-constants would allow for a more dominant win. Running this at these rounds also allowed for time for my AI to develop and local max out on the semi-constants

**Extra Credit**

* I implemented 5 more semi-constants. The way I did this was similar to the first five by multiplying certain heuristic equation by a number between zero and one. These semi-constants are like influencers. The higher the influence the more likely it is to accomplish that action.

**Discuss the results**

* One thing that really surprised me was that my AI would pick one of the mines to set up at randomly. The thing is that my AI would build its army and immediately attack the enemy resulting in a very quick victory.
* I also noticed that my AI, when learning, would prioritize the building semi-constants. It would get those higher quickly over the semi-constants I had for my attacking heuristics.
* I also learned that there would be some rounds that the AI would not build a single soldier or build them very late.
* Another thing I learned was that my AI would only ever attack with archers unless it was in the win state. In this case it would attack with both archers and soldiers. The AI would also learn that it did not need soldiers, even if I had set the value to a number greater than zero.
* I also learned that my AI would win through passivity. It would never attack or never finish off the enemy but collect more gold and have a bigger army then the enemy.