GeoContest

Simulating Strategies of Conquest

Nils Weidmann, CIS Room E.3
weidmann@icr.gess.ethz.ch
January 25, 2005

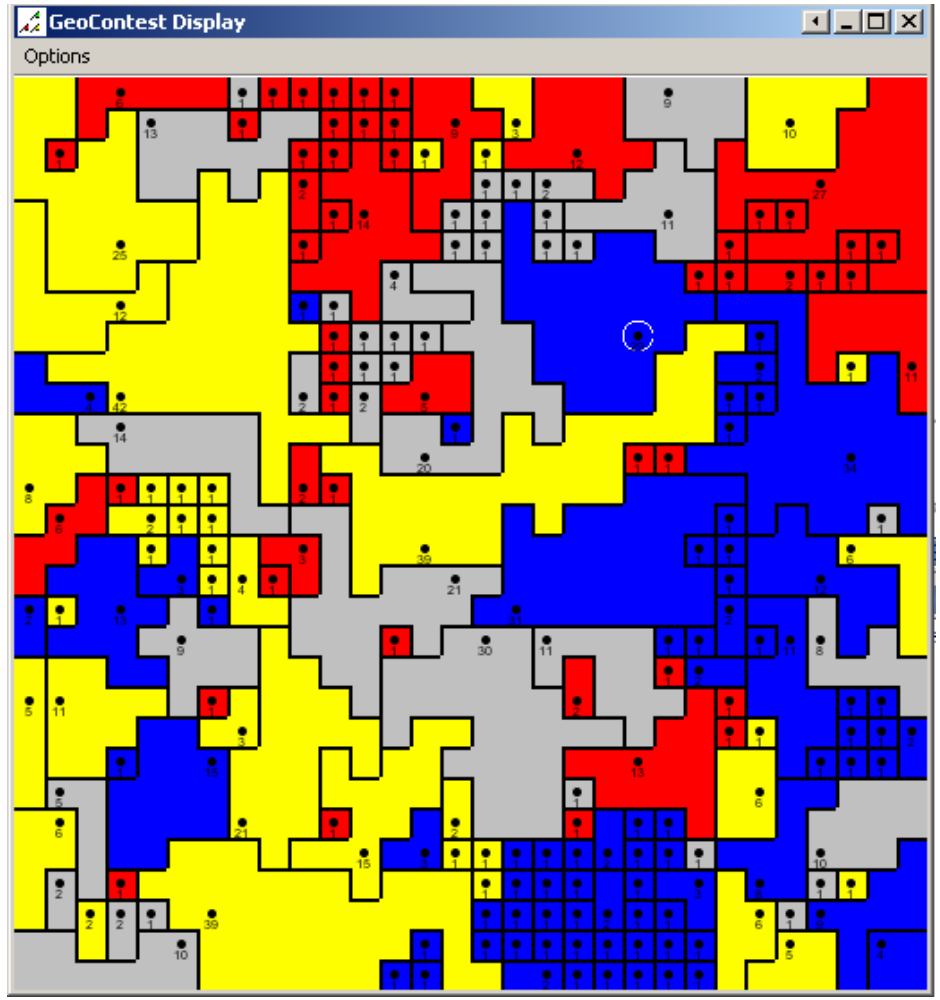
GeoContest

- Simplified version of Geosim
- Designed to examine ”conquest strategies” in a geopolitical system
- Tournament setting: Mix of strategies placed in a territorial grid, see how a strategy ”takes over the world”
- Submit your own strategies

! *GeoContest is still a prototype – comments welcome!*

Setting

- Collection of sovereign actors (marked by •)
- Actors try to conquer adjacent states
- Conquest on province level
- An actor's resources are determined by the area covered
- In each time step, an actor decides which of its neighbors it is going to attack

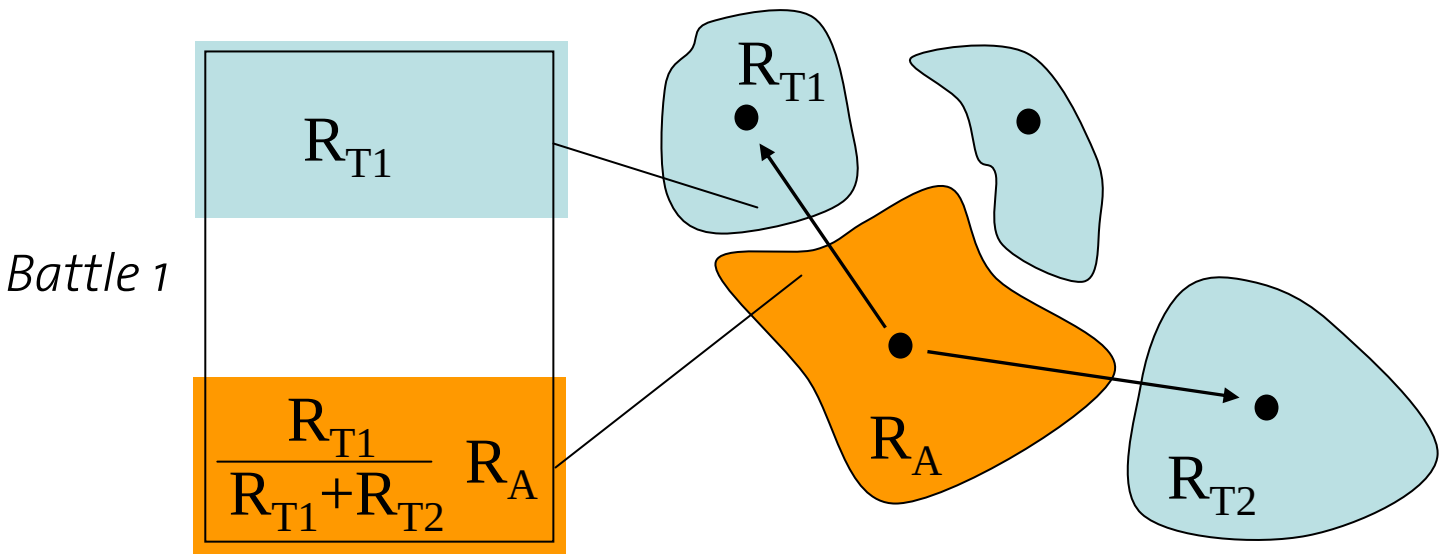


A single time step

- *Sequential activation*: We randomly pick one actor (probability proportional to the areal extension of the actor: Big actors more likely to be picked)
- Actor decides which of its neighbors it wants to attack
- For each of the offensive relations, a „battle path“ is selected
- Battles are carried out with each of the selected targets
- If attacker wins, the province being fought over is assigned to its territory, otherwise attacker loses its province to the target (no draw)

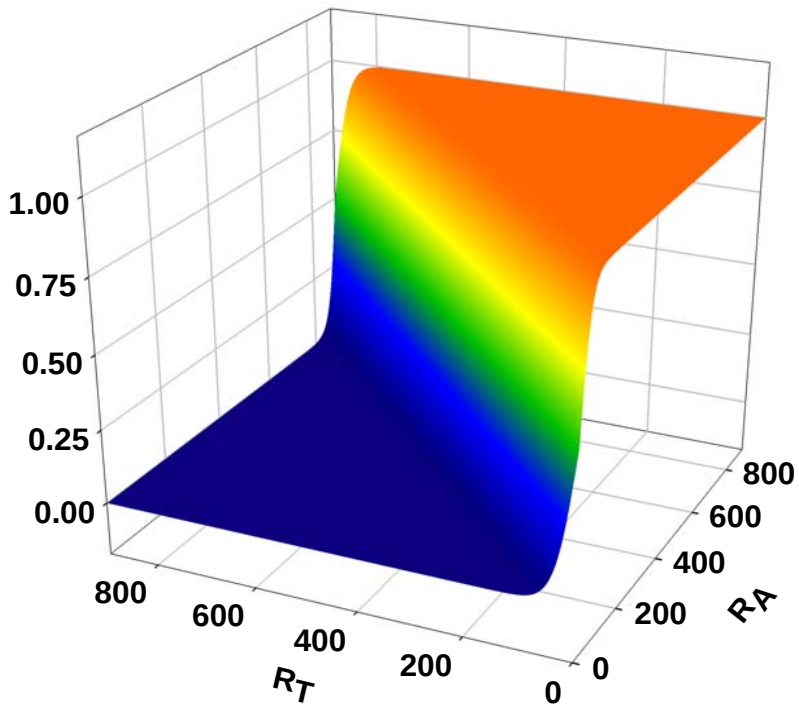
Resources and battles

- Battles are won by stronger states
- But what if an actor attacks several targets at a time?
- Battle resource allocation fixed:



Who wins?

- Depends on the resources R_A, R_T assigned to the fronts
- Probability of attacker winning the battle:



$$p_A(R_A, R_T) = \frac{1}{1 + e^{-k(\frac{R_A - R_T}{R_{\max}})}}$$

with $R_{max} = 900$, $k=50$

Model parameters

- World size: 30x30 cells
- Initial number of actors: 100
- Four strategies in a game, equal number of actors per strategy (25)
- Evaluation: Run for 10,000 time steps, see what strategy covers the largest area
- Runner-up: Strategy that disappeared last or covers the second-largest area at t=10,000.

Tournament setup

- Random split of strategies into groups of four
- Randomly assign first and second rank strategies from each group to new games
- Number of levels depends on the number of participants...

Demo runs

- AlwaysAttacking vs. NeverAttacking
- Cautious, LessCautious, Random1, AlwaysAttacking

Implementing a strategy

- Each strategy object assigned to an actor
- Extend abstract class `Strategy`
- Most important methods:
  - `ArrayList decide(ArrayList neighbors)`
    Takes a list of neighbors. Returns a decision vector, i.e. a list of Doubles D, with D_i=1 if neighbor i should be attacked and D_i=0 otherwise
  - **`void`** `feedback(ArrayList result)`
    Takes a feedback vector of battle results. Required for reinforcement strategies.

Implementing a strategy (cont'd)

- Important fields you might want to use:
  - `ActorAccessor actor` – reference to the actor the strategy is part of
  - `ModelAccessor model` – reference to the model
  - `int type` – the strategy type
  - `java.util.Random random` – the strategy's RNG

ActorAccessor


```
public ArrayList getNeighborStates();
```
ActorAccessor list of neighbors

```
public double getResourceLevel();
```
The actor's resource level

```
public int getNumPolNeighbors();
```
Number of sovereign neighbors

```
public int getNumProvinces();
```
Number of provinces being part of this state

```
public ActorAccessor getGovt();
```
ActorAccessor of the capital

```
public int getX();
public int getY();
```
Coordinates on the grid

```
public boolean isSovereign();
public boolean isAtom();
```

```
public int getStrategyType();
```
the type of the strategy – to recognize actors using the same strategy

ModelAccessor


```
public int getInitPolarity();
```
Initial polarity of the model (100)

```
public int getWorldXSize();
public int getWorldYSize();
```
Grid size (30x30)

```
public int getInitNumStrategies();
```
Number of participating strategies (at the beginning)

```
public int getNumActorsWith(int type);
```
Number of actor with a given strategy

```
public int getNumSovActors();
```
Number of sovereign states

```
public double getWinProbability(double attRes, double tarRes);
```
Probability for an attacker to win a battle with the given resource constellation

Example: AlwaysAttackingStrategy


```
public class AlwaysAttackingStrategy extends Strategy {

    public ArrayList decide(ArrayList neighbors) {

          ArrayList decision = new ArrayList();

          for (int i = 0; i < neighbors.size(); i++) {
                    decision.add(new Double(1.0));
        }

          return decision;
    }

    public void feedback(ArrayList result) {
    }

    public String getName() {
          return "AlwaysAttackingStrategy";
    }

    public String getAuthor() {
          return "Nils Weidmann";
    }
}
```

The decision-making method

We attack all neighbors

We‘re not interested in feedback

Demonstration: AlwaysAttacking vs. NeverAttacking

Example: CautiousStrategy.decide(..)


```
public ArrayList decide(ArrayList neighbors) {

    ArrayList decision = new ArrayList();
    for (int i=0; i<neighbors.size(); i++) {
            decision.add(new Double(0.0));
    }

    // we find out the smallest target size
    double smallestSize = Double.MAX_VALUE;
    for (int i = 0; i < neighbors.size(); i++) {
            ActorAccessor a =  (ActorAccessor) neighbors.get(i);
            if (a.getResourceLevel() < smallestSize) {
                    smallestSize = a.getResourceLevel();
            }
    }

    // we randomly pick one of the smallest actors
    int randIndex;
    ActorAccessor bestActor;
    do {
            randIndex = random.nextInt(neighbors.size());
            bestActor = (ActorAccessor) neighbors.get(randIndex);
    }
    while (bestActor.getResourceLevel() > smallestSize);

    // we attack the randomly chosen actor
    decision.set(randIndex, new Double(1.0));

    return decision;
}
```

Initialize the decision vector

Note: random is the Java built-in RNG (from java.util)

SimpleFeedbackAgent


```
public class SimpleFeedbackStrategy extends Strategy {

    // the strategy's memory
    private int[] memory;

    // the current list of opponents
    private ArrayList last;


    // since we have to initialize the memory,
    // we need to override Strategy.init(..)
    public void init(ActorAccessor actor, ModelAccessor model, int
         type) {
         super.init(actor, model, type);
         memory = new int[model.getInitNumStrategies()];
         for (int i = 0; i < memory.length; i++) {
                 memory[i] = 0;
         }
    }
```

The number of victories we have achieved against each strategy

We store the targets of the current move

SimpleFeedbackAgent (cont'd)


```
public ArrayList decide(ArrayList neighbors) {

        // we determine the maximal number of victories we have achieved
        int maxNumVictories = 0;

        // we store the configuration
        last = new ArrayList(neighbors);

        for (int i = 0; i < memory.length; i++) {
                  if (memory[i] > maxNumVictories) {
                            maxNumVictories = memory[i];
                  }
        }

        ArrayList decision = new ArrayList();

        // attack if we have been successful fighting this strategy
        // otherwise, attack with small probability
        for (int i=0; i<neighbors.size(); i++) {
                  ActorAccessor currActor = (ActorAccessor) neighbors.get(i);
                  if (memory[currActor.getStrategyType()] == maxNumVictories) {
                            decision.add(new Double(1.0));
                  }
                  else if (random.nextDouble() < 0.3) {
                            decision.add(new Double(1.0));
                  }
                  else decision.add(new Double(0.0));
        }
        return decision;
}
```

SimpleFeedbackAgent (cont'd)


```
    public void feedback(ArrayList feedback) {

          // we update our memory
          for (int i = 0; i < feedback.size(); i++) {
                Double res =  (Double) feedback.get(i);
                if (res.intValue() == Model.BATTLE_WIN) {
                      ActorAccessor a = (ActorAccessor) last.get(i);
                      memory[a.getStrategyType()]++;
                }
          }
    }

    public String getName() {
          return "SimpleFeedbackStrategy";
    }

    public String getAuthor() {
          return "Nils Weidmann";
    }

} // end class SimpleFeedbackAgent
```

Online resources

- The GeoContest web site: http://www.icr.ethz.ch/research/geocontest
- Download the code
- Browse the documentation
- Look at the implemented strategies

- We‘re looking forward to your submissions!