

**ETH**

Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

INTERNATIONAL CONFLICT RESEARCH

# Advanced Computational Modeling of Social Systems

## GeoContest V.2

Nils Weidmann  
International Conflict Research  
Swiss Federal Institute of Technology (ETH)  
[weidmann@icr.gess.ethz.ch](mailto:weidmann@icr.gess.ethz.ch)

# Model Setting

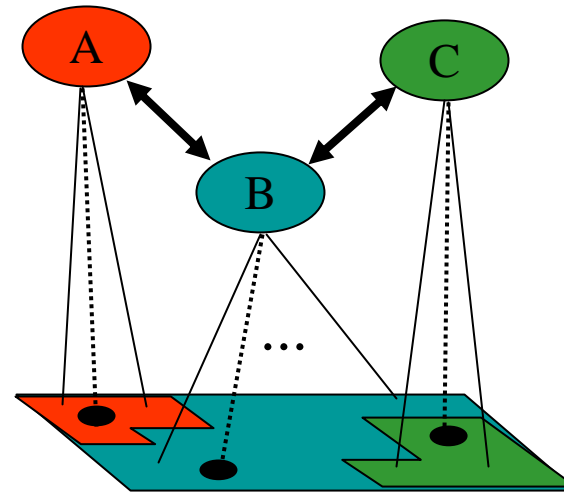
Two layers:

States

Actor  
Relation

Space

SpatialUnit



# A single time step

- *Parallel activation:*

All actors

- decide on which front to attack (decision made by strategy)
  - allocate resources to their active fronts (proportional to opponent size)
  - fight battles along their relations
- Model carries out structural change based on battle results

# Accessors

- Model objects are hidden from a strategy by their accessor objects
- Accessor objects control all operations on the model objects made by a strategy
- Accessors available: ModelAccessor, ActorAccessor, RelationAccessor

# Java 1.5 new features

- Generics: Typed collections

```
ArrayList<Actor> actors = new ArrayList<Actor>();  
actors.add(new Actor());  
Actor anActor = actors.get(i);
```

No cast needed: safe  
and efficient.

- For-each loop: Iterate over list elements

```
for (Actor active : actors) {  
    active.step();  
}
```

# RelationAccessor

```
public ActorAccessor owner();
```

The owner of the relation (relations are directed)

```
public ActorAccessor other();
```

The target actor

```
public HistoryEntry prevResult();
```

The result of the previous interaction along this relation

```
public List<HistoryEntry> getHistory();
```

The complete history

Relation.Action:  
COOPERATE,  
DEFECT

```
public void setAction(Relation.Action action);
```

Sets the action for the next move intended by the owner

# History

- A relation history is an ordered collection of HistoryEntries
- Each HistoryEntry stores
  - The owner's action: COOPERATE, DEFECT
  - The other's (=target) action: COOPERATE, DEFECT
  - The battle outcome: WIN, LOSE, DRAW or NO (= no battle)

**Relation.BattleOutcome:  
WIN, LOSE, DRAW, NO**

# ActorAccessor

```
public ArrayList<ActorAccessor> neighbors();
```

A list of the actor's neighbors

```
public double resources();
```

The resources the actor has at its disposal

```
public int numProvinces();
```

The number of provinces (N.B. equal to resources ( ))

```
public ArrayList<RelationAccessor> relations();
```

The relations of this actor

```
public RelationAccessor getRelationWith(ActorAccessor other);
```

Pick a relation to a specified actor – **null** if there is none

```
public int strategyType();
```

The strategy type

```
public int getX(); public int getY();
```

The position of the capital

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

Actor is atomic iff it has just one province

# ModelAccessor

```
public int numStrategies();
```

The number of strategies taking part

```
public int numActors(int s);
```

The number of actors of a given strategy s

```
public int worldXSize();
```

```
public int worldYSize();
```

The grid size

```
public double winProb(double attRes, double  
targetRes);
```

The probability of winning for an attacker with resources attRes against a target actor with resources targetRes

# Example: AlwaysAttackingStrategy

```
public class AlwaysAttackingStrategy extends Strategy {  
  
    public void decide() {  
        for (RelationAccessor rel : actor.relations()) {  
            rel.setAction(Relation.Action.DEFECT);  
        }  
    }  
  
    public String getName() {  
        return "AlwaysAttackingStrategy";  
    }  
  
    public String getAuthor() {  
        return "Nils Weidmann";  
    }  
}
```

# Example: TitForTatStrategy

```
public class TitForTatStrategy extends Strategy {  
  
    public void decide() {  
  
        for (RelationAccessor rel : actor.relations()) {  
            if (rel.getHistory().isEmpty()) {  
                rel.setAction(Relation.Action.COOPERATE);  
            }  
            else rel.setAction(rel.prevResult().otherAction());  
        }  
    }  
  
    public String getName() {  
        return "TitForTatStrategy";  
    }  
  
    public String getAuthor() {  
        return "Nils Weidmann";  
    }  
  
}
```

# Demo runs

- AlwaysAttacking vs. NeverAttacking
- Cautious vs. Random1
- > what results do you expect?
  
- KantianStrategy vs. CautiousStrategy
- > different outcomes demonstrate impact of activation regime!

# Possible extensions

- Symmetric (D,D) and asymmetric (C,D; D,C) attack decisions are treated the same way: Battle. -> Change this?
- Resources allocated by strategy

If you are interested in submitting a GeoContest V.2 strategy, please let me know (*[weidmann@icr.gess.ethz.ch](mailto:weidmann@icr.gess.ethz.ch)*).