Toward Simulating Networked Societies with Formal Institutions Using AI Agents (Supplementary Appendix)

Michael Richards, Daniel Cowser, Daniel Nielson, Jacob W. Crandall

This supplementary appendix contains detailed information regarding the model, experiments, and results presenting in the main paper. The paper has four sections:

1. Details related to the simulation model presented in the paper
2. Supporting details related to the experiments and results performed with already-established government institutions (first two results sections)
3. Supporting details related to the experiments and results performed with other institutions (last two results sections)
4. Code commands used to conduct the experiments.

Supplementary code provided to run all the experiments is also provided.

1 Model Details (SM 1)

We provided additional details about the model used in our experiments in three parts:

1.1 The parameters and settings used in the JHG

1.2 Implementation details of institutional players

1.3 Implementation details of non-institutional players (society members)

1.1 Parameters and setting used in the JHG (Economic Game)

We used the C++ version of the code provided by Skaggs et al. [1] in our implementation of the JHG. We used the following parameter settings:

- Wealth update coefficient: $\alpha = 0.20$
- Fame vs. fortune coefficient: $\beta = 1$ – This means that capital exchanged between the players is modeled fully as some form of a material good.
- $c_{\text{keep}} = 0.95$ – Keeping coefficient, specifying the wealth multiplier for keeping tokens – same for all players and interactions
- $c_{\text{give}} = 1.3$ – The trade coefficient used, specifying the wealth multiplier for giving tokens – same for all players and interactions
- $c_{\text{take}} = 1.6$ – Attack coefficient used, specifying the wealth multiplier for attacking (stealing) – same for all players and interactions

Previous authors, in naming and introducing the JHG, have referred to the currency of the JHG *popularity* (seemingly to be consistent with the theme of grade-school). We instead use the term *wealth* throughout this paper.

Parameter	Possible values	Description
Fee schedule	{Progressive, Regressive, Flat}	Defines the taxes/fees the institution asks of players
% Power	$[0, 100]$	Specifies the target percentage of the wealth the institution should hold in society. It also specifies the magnitude of fees – fees are set so that if all players pay the specified fees, the institution will hold this percentage of societal wealth.
Redistribution	{Progressive, Regressive, Proportional, Merit}	Specifies the proportion of redistribution tokens that go to each player
Remuneration	{Yes, No}	Specifies whether or not the institution reimburses members who are victims of attack. Membership is determined by fee payment and lack of attacking other society members. If *Yes*, then the institution gives back the amount that the member lost in the attack(s).
Punish nonpayment	{Yes, No}	Specifies whether the institution punishes a player for not being taxes/fees
Punish attacks	{Yes, No}	Specifies whether the institution punishes a player for attacking a player with membership in the institution
Theft threshold	≥ 0	Specifies the amount of loss a player must experience in an attack before the institution considers remuneration and punishment
Punish strength	≥ 0	Specifies the number of times the institutions punishes a member for a crime. For example, if 1.0, then the institution punishes for the extent of the crime. If 2.0, then the institution punishes double the amount of the crime.
Extractive	{Yes, No}	Specifies whether or not the institution attacks non-member players without cause.

Table 1: Parameters of institutional players.

1.2 Institutional Players

Institutional players followed scripted behavior that is controlled by the set of parameters listed in Table 1. The full math for these institutional players is spelled out in the supplied code, specifically files `Institution.h` (for government institutions) and `Coop.h` for non-government players. Both players follow the same general algorithm, but slightly different implementations are given for these agents. Differences are due to government institutions being designed to reflect attributes of actual governments while non-government institutions are designed to be efficient and gain power quickly so as to attract contributions from society members.

Both forms of institutional players carry out the following steps:

- Determine fees/taxes and broadcast them to the players. The way this is done is specified in the function `broadcastNextTaxes`. In Progressive fee schedules, more wealthy players are asked to pay a higher percentage of their tokens in taxes/fees than less wealthy players. Regressive is the opposite: less wealthy players are asked to pay a higher percentage of their tokens in taxes/fees then more wealthy players. A Flat tax/fee scheme indicates that all players are asked to pay the same percentage of tokens in taxes/fees. The magnitude of taxes/fees is controlled by the value of the parameter *% Power*.
- The institutional player then determines how to allocate their tokens (see the `allocateTokens` function for details. The following order of allocations is followed until no tokens remain to be allocated

1. Defend self: The institutional player predicates how much it will be attacked in the round based on token allocations by other players in previous rounds. It then keeps enough tokens to block the expected amount of attacks.

   See the function `defendSelf` for details.

2. Punish crime: If the institutional player still has unallocated tokens, it allocates tokens to attack crime if either *Punish nonpayment* or *Punish attacks* are set to *Yes*.

   See the function `punishCrimeFunc` for details.

3. Extract: If *Extractive* = *Yes* and the institutional player is not punishing any crimes in this round, then (with $\frac{1}{3}$ probability) it attacks another player that does not have membership in its coop.

   See the function `extractFunc` in `Coop.h` for details.

4. Remunerate victim: If the institutional player still has unallocated tokens and the parameter *Remuneration* is set to true, it allocates tokens to restore wealth that was stolen from members who have previously been attacked.

   See the function `remunerateVictimsFunc` for details.

5. Redistribution: If the institutional player still has unallocated tokens, it gives tokens to members according to the parameter *Redistribution*. Progressive redistribution gives more tokens to less wealth players than more wealthy players, while Regressive does the opposite. Proportional gives back to players at the amount they tend to pay in fees. Merit redistribution dictates redistribution amounts on how much a player is contributing to society based on effort. This is computed by the percentage of taxes/fees paid plus the percentage of tokens given out by the player (as opposed to keeping or attacking – which hurts society). Because merit-based redistribution is not based on how much is given but simply the percentage of tokens, this redistribution schedule tends to help the poorer members of society more than the rich (its more like Progressive redistribution), but requires players to be trying to help society.

   Government players (`Institution.h`) based redistribution amounts on the current wealth of the player, while non-government institutional players (`Coop.h`) base redistribution amounts on the influence matrix of the JHG. This difference is because governments sometimes based redistribution on income levels, while other (perhaps smaller) institutions can more easily base it more readily on actual contributions made.

   See the function `doReimbursementsFunc` for details.

We note three differences in our implementations of already-established governments (`Institution.h`) and non-government (`Coop.h`) institutions:

1. Our already-established governments use a fixed *% Power* parameter throughout the simulation. Other institutional players use an adaptive one. The fee starts out at the specified level, which is typically high to allow the institution to gain strength from nothing in early rounds of the game. When the institution and its members have become strong relative to the rest of society or nobody is attacking it or its members, it slowly reduces *% Power* down to 20%. This reduces the institution's power. When the institution or its members lose power relative to society or it or its members get attacked, it begins to raise it fees again.

   See `broadcastNextTaxes` in `Coop.h` for details.

2. Non-government institutional players only attack another player if its wealth is greater than the other player's wealth.

3. Non-government players also redistribute all remaining tokens (to help build the strength of its members), while already-established governments hold back some tokens to help it maintain its target wealth.

These differences help non-government institutions (1) gain power quickly, (2) effectively distribute wealth to its members, and (3) thrive in the presence of other institutions.

1.3 Members of Society

Non-institutional players in our simulations follow hCAB strategies [2] (with the exception of the society with bad actors, which also include CATs [1] – see the main paper). We use the 100 hcab parameterizations computed by Skaggs and Crandall [2] to defined the behavior of our agents.

Our society was tuned to the current institutional structure by conducting training simulations. During training, the probability that each hcab parameterization is chosen to represent an agent in society is proportional to its past performance (based on economic growth) in prior training games it has been part of in which the agent had the same social class. This same mechanism is used to select agents in evaluations after training.

The social class of a player in society is determined as follows. Let W_i denote the initial wealth of the player and let $\bar{W}$ denote the mean initial wealth of all non-institutional players. Then, player i is determined to be *lower-class* if $W_i < 0.5\bar{W}$, player i is *upper-class* if $W_i \geq 1.5\bar{W}$, and *middle-class* otherwise. Thus, three sets of parameterizations (one for lower-class, one for middle-class, and one for upper-class) are kept. All parameterizations are evaluated during training in each class.

Formally, let Z be the set of all parameterizations. Then, the probability that parameterization $z \in Z$ is selected for an agent in a particular class is: $\frac{G(z)}{\sum_{y \in Z} G(y)}$, where $G(y)$ is the average economic growth of parameterization y in past training runs in which the parameterization was involved (for that class). See the function `playRandomGame_choice` for details.

2 Study details with already-established governments (SM 2)

We detail the optimization functions, genetic algorithm, and additional results in this section.

2.1 Optimization Functions

In this subsection, we describe the objective functions used to compute the four governments for each condition. Let

$$W = \frac{\bar{W}}{500}, \tag{1}$$

where $\bar{W}$ is the average wealth of non-institutional players after 30 rounds. Furthermore, let

$$G = max\left(0, \frac{0.5 - giniIndex}{0.5}\right), \tag{2}$$

where $giniIndex$ is the wealth Gini Index of the society after 30 rounds. The fitness of a society with respect to each objective function is then given by:

- *Prioritize Government Wealth*: For this function, *Fitness* is simply the wealth of the government player after 30 rounds
- *Prioritize Equality*: $Fitness = 0.2W^2 + 0.8 * G^2$
- *Prioritize Wealth*: $Fitness = 0.8W^2 + 0.2 * G^2$
- *Balanced Priorities*: $Fitness = min(W, G)$

2.2 Genetic Algorithm

We evolved the optimized governments over 20 generations, beginning with a gene pool of 25 governments (parameters selected randomly). In each generation, each of the 25 governments was evaluated in 25 20-round games . The fitness function for a government was then the average fitness (computed using the objective function) in this 25 games. After each generation, the gene pools for each generation thereafter were constructed using the process defined in the function `evolveGovs` in `playInst.cpp` in the supplied code. This function uses selection, crossover, and mutation to produce new governments.

To speed up computation, we conducted these optimizations in 10-player simulations (with 2 CATs) rather than 20-player simulations (with 4 CATs).

	Prioritize Equality	Balanced Priorities	Prioritize Ave Wealth	Prioritize Gov Wealth
	Uniform Initial Wealth			
Fee Schedule	Progressive	Flat	Progressive	Flat
% Power	0.0	3.0	1.5	49.5
Redistribution	Regressive	Regressive	Regressive	Merit
Remuneration	Yes	No	Yes	No
Punish nonpayment	No	No	No	Yes
Punish attacks	No	No	No	No
Theft Tolerance	3.0	–	22.5	–
Punishment strength	–	–	–	2.40
	Uniform Initial Wealth, 20% CATs			
Fee Schedule	Progressive	Progressive	Flat	Regressive
% Power	29.5	30.5	27.0	47.0
Redistribution	Progressive	Progressive	Merit	Merit
Remuneration	No	Yes	No	No
Punish nonpayment	Yes	Yes	Yes	Yes
Punish attacks	Yes	Yes	No	No
Theft Tolerance	14.5	9.5	–	–
Punishment strength	0.96	1.48	3.32	3.32
	Unequal Initial Wealth			
Fee Schedule	Flat	Flat	Progressive	Flat
% Power	39.0	24.5	5.5	47.5
Redistribution	Progressive	Merit	Regressive	Merit
Remuneration	Yes	No	No	Yes
Punish nonpayment	Yes	Yes	No	Yes
Punish attacks	No	Yes	Yes	No
Theft Tolerance	34.0	15.5	30.5	45.5
Punishment strength	3.28	3.12	3.56	2.12

Table 2: Optimized government policies for each population type and objective function. A '–' indicates the value has no effect given other policy settings. *Extractive* was set to *No* for all of these governments.

2.3 Results

Table 2 gives the full set of parameter computed by the genetic evolution for each condition and objective function (only a partial set of parameters was given in the main paper).

Table 3 provides additional results regarding societies with already-established government players. *Player Wealth* specifies the mean wealth of non-institutional players after 30 rounds over all simulations. *Gov. Wealth* specifies the mean wealth of the government player after 30 rounds over all simulations. *Total Wealth* specifies the mean wealth of all players after 30 rounds. *Gini Index* specifies the mean Gini Index over all simulations after 30 rounds. The other columns specify the average number of tokens kept, taken (i.e., attacks), and given by non-institutional and government players throughout the 30 rounds over all simulations.

Figure 4 in the main paper describes how populations that begin with different wealth distributions produce different economic growth (error bars in that figure specify the standard error of the mean over the 50 simulations conducted for each initial wealth condition). In the figure, economic growth is given by the total wealth in society (sum of all player wealth) divided by the initial wealth in society.

Table 4 lists the initial wealth of all 20 players for all different starting conditions.

3 Experiment details with non-government institutions (SM 3)

The last two results sections in the main paper describe experiments in which societies have binary choices between two institutions. We define the parameters of each of these institutions in Tables 5-9.

Government	Player Wealth	Gov. Wealth	Total Wealth	Gini Index	% Keep (Player)	% Take (Player)	% Give (Player)	% Taxes Paid	% Keep (Gov)	% Take (Gov)	% Give (Gov)
Condition 1: Uniform Initial Wealth											
No Government	389±6	—	7781±103	0.139±0.005	10.9	2.8	86.2	—	—	—	—
Prioritize Equality	375±5	42± 3	7547± 98	0.125±0.005	11.8	3.1	85.2	26.0	21.2	0.0	78.8
Balanced Priorities	353±6	178± 3	7239±120	0.154±0.006	11.5	3.5	85.0	32.3	60.0	0.0	40.0
Priority Ave. Wealth	382±6	96± 3	7729±116	0.129±0.005	11.4	2.8	85.8	25.4	49.1	0.0	50.9
Prioritize Gov. Wealth	194±7	3050±10	6926± 23	0.179±0.006	8.1	1.6	90.3	95.3	41.6	6.8	57.7
Condition 2: Uniform Initial Wealth, 20% Bad Actors (CATs)											
No Government	88±3	—	1754± 57	0.516±0.013	28.5	17.8	53.7	—	—	—	—
Prioritize Equality	182±2	1411±17	5043± 66	0.252±0.004	23.5	12.9	63.6	88.1	12.8	10.2	77.0
Balanced Priorities	206±3	1692±19	5810± 77	0.247±0.003	25.0	9.5	65.5	89.5	16.2	8.3	75.5
Priority Ave. Wealth	259±2	1763±10	6944± 40	0.320±0.004	25.6	2.2	72.2	78.0	20.3	7.0	72.7
Prioritize Gov. Wealth	187±1	2737± 9	6483± 23	0.346±0.005	24.3	1.4	74.3	70.7	37.7	3.7	58.6
Condition 3: Unequal Initial Wealth											
No Government	287±8	—	5733± 68	0.656±0.010	32.5	3.8	63.7	—	—	—	—
Prioritize Equality	204±2	2329±22	6402± 63	0.057±0.003	14.2	3.5	82.2	97.9	29.2	1.1	69.7
Balanced Priorities	304±3	1844±19	7923± 87	0.198±0.004	9.8	1.8	88.4	98.2	19.1	2.5	78.4
Priority Ave. Wealth	307±6	247±14	6379±125	0.599±0.007	26.7	3.3	69.9	23.5	50.7	17.7	31.6
Prioritize Gov. Wealth	202±1	3015±18	7063± 43	0.164±0.005	6.8	1.3	91.9	97.6	39.9	0.9	59.2

Table 3: Mean values for each condition and government. ± values show standard error of the mean.

4 Code commands used to conduct the experiments (SM 4)

This section outlines the specific commands invoked to run the various experiments reported in the paper from the supplied code. Before doing this, the code must be compiled in the folder `GeneSimulation_cpp`: `g++ playInst.cpp -o jhginst`

Note that the code was created and run on MacOS Sequoia. Other libraries, etc. may need to be included for other operating systems. It can run effectively on standard and current laptops and desktops.

- Commands used to evolve governments were as follows:

  `./jhginst evolveGov 25 20 25 [condition]-[objective]`

  Here, `condition` is *equal* for Condition 1, *cats* for Condition 2, and *exp* for Condition 3. Also `objective` is 20 for *prioritize equality*, 50 for *balanced priorities*, 80 for *prioritize ave. wealth*, and *gov* for *prioritize gov. wealth*.

- Commands used to tune and then evaluate the various already-established government societies were as follows (each two-step sequences to tune and then evaluate):

  - No government in Condition 1:

    `./jhginst tune3 30 equal_20 hcabs_enone hcabs_20 100`

    `./jhginst play3 30 equal_20 hcabs_enone hcabs_20 100`

  - Prioritize equality in Condition 1:

    `./jhginst tuneGov 30 equal_21 hcabs_e2080 govment_20 gov_equal-20-80 200`

    `./jhginst playGov 30 equal_21 hcabs_e2080 govment_20 gov_equal-20-80 100`

  - Balanced priorities in Condition 1:

    `./jhginst tuneGov 30 equal_21 hcabs_e5050 govment_20 gov_equal-50-50 200`

    `./jhginst playGov 30 equal_21 hcabs_e5050 govment_20 gov_equal-50-50 100`

  - Prioritize ave. wealth in Condition 1:

    `./jhginst tuneGov 30 equal_21 hcabs_e8020 govment_20 gov_equal-80-20 200`

    `./jhginst playGov 30 equal_21 hcabs_e8020 govment_20 gov_equal-80-20 100`

  - Prioritize gov. wealth in Condition 1:

    `./jhginst tuneGov 30 equal_21 hcabs_egpow govment_20 gov_equal-GovPower 200`

    `./jhginst playGov 30 equal_21 hcabs_egpow govment_20 gov_equal-GovPower 100`

  - No government in Condition 2:

    `./jhginst tune3 30 equal_20 hcabs_cnone hcabs16_cats4 100`

    `./jhginst play3 30 equal_20 hcabs_cnone hcabs16_cats4 100`

Player	Gini 0.000	Gini 0.90	Gini 0.158	Gini 0.253	Gini 0.361	Gini 0.460	Gini 0.533	Gini 0.589	Gini 0.665	Gini 0.721	Gini 0.790	Gini 0.855
p1	100	86	75	60	43	27	16	7	3	1	1	1
p2	100	86	75	60	44	28	17	8	4	2	1	1
p3	100	86	76	61	45	30	19	10	5	2	1	1
p4	100	87	76	62	46	31	20	12	6	3	1	1
p5	100	87	77	63	48	33	22	14	7	4	1	1
p6	100	87	77	64	48	35	24	16	9	4	1	1
p7	100	88	79	66	51	38	29	21	12	7	2	1
p8	100	88	80	68	54	42	32	25	15	8	2	1
p9	100	90	82	71	58	47	38	32	20	12	4	1
p10	100	91	84	74	63	52	45	39	26	17	6	1
p11	100	92	86	78	69	60	54	49	35	23	10	2
p12	100	94	90	84	77	70	66	62	46	33	16	3
p13	100	97	94	91	87	83	80	78	62	47	25	6
p14	100	100	100	100	99	99	99	99	83	67	40	12
p15	100	104	107	111	116	120	124	126	112	96	65	24
p16	100	109	116	126	136	147	154	160	151	138	106	51
p17	100	116	128	145	164	182	195	205	206	200	174	109
p18	100	125	144	170	200	227	247	263	282	291	286	231
p19	100	136	164	202	246	286	315	338	386	424	472	493
p20	100	151	190	244	306	363	404	436	530	621	786	1059

Table 4: The starting wealth of all 20 players in each condition for Figure 4 in the main paper.

Institution 1 – Merit		**Institution 2 – Proportional**	
Fee Schedule	Flat	Fee Schedule	Flat
% Power	80	% Power	80
Redistribution	Merit	Redistribution	Proportional
Remuneration	Yes	Remuneration	Yes
Punish nonpayment	No	Punish nonpayment	No
Punish attacks	Yes	Punish attacks	Yes
Theft threshold	0.01	Theft threshold	0.01
Punish strength	4.0	Punish strength	4.0
Extractive	No	Extract	No

Table 5: Institution parameters for merit vs proportional redistribution.

Institution 1 – Remuneration		**Institution 2 – No Remuneration**	
Fee Schedule	Flat	Fee Schedule	Flat
% Power	50	% Power	50
Redistribution	Proportional	Redistribution	Proportional
Remuneration	Yes	Remuneration	No
Punish nonpayment	No	Punish nonpayment	No
Punish attacks	Yes	Punish attacks	Yes
Theft threshold	0.01	Theft threshold	0.01
Punish strength	4.0	Punish strength	4.0
Extractive	No	Extract	No

Table 6: Institution parameters for comparison between remuneration and no remuneration.

Institution 1 – Low Fees		**Institution 2 – High Fees**	
Fee Schedule	Flat	Fee Schedule	Flat
% Power	30	% Power	80
Redistribution	Proportional	Redistribution	Proportional
Remuneration	Yes	Remuneration	Yes
Punish nonpayment	No	Punish nonpayment	No
Punish attacks	Yes	Punish attacks	Yes
Theft threshold	0.01	Theft threshold	0.01
Punish strength	4.0	Punish strength	4.0
Extractive	No	Extract	No

Table 7: Institution parameters for comparison between lower and higher initial fees (power).

Institution 1 – More Vengeful		**Institution 2 – Less Vengeful**	
Fee Schedule	Flat	Fee Schedule	Flat
% Power	50	% Power	50
Redistribution	Proportional	Redistribution	Proportional
Remuneration	Yes	Remuneration	Yes
Punish nonpayment	No	Punish nonpayment	No
Punish attacks	Yes	Punish attacks	Yes
Theft threshold	0.01	Theft threshold	0.01
Punish strength	8.0	Punish strength	1.0
Extractive	No	Extract	No

Table 8: Institution parameters for comparison between higher (more vengeful) and lower punishments (less vengeful).

Institution 1 – Extractive		**Institution 2 – Not Extractive**	
Fee Schedule	Flat	Fee Schedule	Flat
% Power		% Power	
Redistribution	Proportional	Redistribution	Proportional
Remuneration	Yes	Remuneration	Yes
Punish nonpayment	No	Punish nonpayment	No
Punish attacks	Yes	Punish attacks	Yes
Theft threshold	0.01	Theft threshold	0.01
Punish strength	4.0	Punish strength	4.0
Extractive	Yes	Extract	No

Table 9: Institution parameters for comparison between extractive and not extractive.

  - Prioritize equality in Condition 2:
    `./jhginst tuneGov 30 equal_21 hcabs_c2080 gov_hcabs16_cats4 gov_cats-20-80 200`
    `./jhginst playGov 30 equal_21 hcabs_c2080 gov_hcabs16_cats4 gov_cats-20-80 100`
  - Balanced priorities in Condition 2:
    `./jhginst tuneGov 30 equal_21 hcabs_c5050 gov_hcabs16_cats4 gov_cats-50-50 200`
    `./jhginst playGov 30 equal_21 hcabs_c5050 gov_hcabs16_cats4 gov_cats-50-50 100`
  - Prioritize ave. wealth in Condition 2:
    `./jhginst tuneGov 30 equal_21 hcabs_c8020 gov_hcabs16_cats4 gov_cats-80-20 200`
    `./jhginst playGov 30 equal_21 hcabs_c8020 gov_hcabs16_cats4 gov_cats-80-20 100`
  - Prioritize gov. wealth in Condition 2:
    `./jhginst tuneGov 30 equal_21 hcabs_cgpow gov_hcabs16_cats4 gov_cats-GovPower 200`
    `./jhginst playGov 30 equal_21 hcabs_cgpow gov_hcabs16_cats4 gov_cats-GovPower 100`
  - No government in Condition 3:
    `./jhginst tune3 30 exp_20 hcabs_snone hcabs_20 600`
    `./jhginst play3 30 exp_20 hcabs_snone hcabs_20 100`
  - Prioritize equality in Condition 3:
    `./jhginst tuneGov 30 exp_21 hcabs_s2080 govment_20 gov_exp-20-80 1200`
    `./jhginst playGov 30 exp_21 hcabs_s2080 govment_20 gov_exp-20-80 100`
  - Balanced priorities in Condition 3:
    `./jhginst tuneGov 30 exp_21 hcabs_s5050 govment_20 gov_exp-50-50 1200`
    `./jhginst playGov 30 exp_21 hcabs_s5050 govment_20 gov_exp-50-50 100`
  - Prioritize ave. wealth in Condition 3:
    `./jhginst tuneGov 30 exp_21 hcabs_s8020 govment_20 gov_exp-80-20 1200`
    `./jhginst playGov 30 exp_21 hcabs_s8020 govment_20 gov_exp-80-20 100`
  - Prioritize gov. wealth in Condition 3:
    `./jhginst tuneGov 30 exp_21 hcabs_sgpow govment_20 gov_exp-GovPower 1200`
    `./jhginst playGov 30 exp_21 hcabs_sgpow govment_20 gov_exp-GovPower 100`

- Commands used to tune and then evaluate societies with binary pairs of institutions all use the following two commands:

  `./jhginst tuneDuelingCoops 80 exp_22 hcabs_coops_current coops_20 2000`

  `./jhginst playDuelingCoops 80 exp_22 hcabs_coops_current coops_20 100`

  Before running these commands, however, the coops need to be set up. This can be done as follows:

  - For Configuration 2:
    Change `Coops/coop1_merit.txt` to `Coops/coop1.txt` and `Coops/coop2_proportional.txt` to `Coops/coop2.txt`
  - For Configuration 3:
    Change `Coops/coop1_remuneration.txt` to `Coops/coop1.txt` and `Coops/coop2_noremuneration.txt` to `Coops/coop2.txt`
  - For Configuration 4:
    Change `Coops/coop1_lowpower.txt` to `Coops/coop1.txt` and `Coops/coop2_highpower.txt` to `Coops/coop2.txt`
  - For Configuration 5:
    Change `Coops/coop1_vengeful.txt` to `Coops/coop1.txt` and `Coops/coop2_notvengeful.txt` to `Coops/coop2.txt`
  - For Configuration 6:
    Change `Coops/coop1_extractive.txt` to `Coops/coop1.txt` and `Coops/coop2_notextractive.txt` to `Coops/coop2.txt`

References

[1] J. Skaggs, M. Richards, M. Morris, M. A. Goodrich, and J. W. Crandall. Fostering collective action in complex societies using community-based agents. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Jeju, Korea, 2024.

[2] J. B. Skaggs and J. W. Crandall. Modeling human behavior in a strategic network game with complex group dynamics. *arXiv:2505.03795*, 2025.