Mechanismen der Polarisierung von Parteiensystemen:

Ideologische Dynamiken im Kontext von Angebot, Nachfrage und institutionellen Rahmenbedingungen

Der Quellcode des agentenbasierten Simulationsmodells

Wichtig: Der Quellcode ist leicht verändert im Vergleich zum Original, damit die Darstellung im DIN-A4 Format möglich ist. Dies betrifft in erster Linie die Kommentierung. Um den Code technisch nachzuvollziehen, eignet sich die Darstellung in NetLogo wesentlich besser als die hier vorliegende Version im DIN-A4 Format (ebenfalls zum Download verfügbar). Diese Version dient lediglich dem Referenzieren innerhalb der Publikation (Schmitt, Johannes: „Mechanismen der Polarisierung von Parteiensystemen: Ideologische Dynamiken im Kontext von Angebot, Nachfrage und institutionellen Rahmenbedingungen“). Weil Zeilen in NetLogo teilweise nicht als eine Zeile auf DIN A4 darstellbar sind, umfassen eine Spalte mehr als eine Zeile.

Der Quellcode besteht aus sechs einzelnen Dateien:

1. main: Zeile 1 bis 79
2. breeds_and_globals.nls: Zeile 80 bis 154
3. setup.nls: Zeile 155 bis 552
4. functions.nls: Zeile 553 bis 851
5. partie.nls: Zeile 852 bis 1119
6. voters.nls: Zeile 1120 bis 1275
7. elections.nls: Zeile 1276 bis 1289
8. coalitions.nls: Zeile 1290 bis 1547
9. output.nls: Zeile 1548 bis 1587

main

```
1   __includes ["breeds_and_globals.nls" "setup.nls" "functions.nls" "voters.nls" "parties.nls"
    "election.nls" "output.nls" "coalition.nls"]
2
3   to setup
4     ; Setup inialisiert immer einen neuen Durchlauf
5
6     ; Für den neuen Durchlauf wird alles zurückgesetzt
7     clear-all
8     reset-timer
9     reset-ticks
10    file-close-all
11
12    ; Variablen werden initialisiert
13    set emergency-coalition "none"
14    set district-relevants (list)
15    ini-measures
16    set error-code "none"
```

```
17
18      ; Setup für den Durchlauf wird aufgesetzt - […]
19      if random-setup? [random-setup]
20      random-seed used-seed
21      rebalance-input
22      setup-competition
23
24      ; Ausgangspunkt wird berechnet
25      calculate-election-result
26      create-coalition-at-start
27      set cabinet-opposition cabinet-opposition-status
28      set relevant-patches relevant-patches with [votes > 0]
29      if error-code != "none" [stop]
30      set  start-cabinet-is-electoral-connected?  coalition-is-connected-given-any-party?  sort
      parties with [cabinet = 1]
31
32      set polarization-sd-seats-begin polarization-sd-seats?
33      set polarization-sd-votes-begin polarization-sd-votes?
34      set polarization-range-begin    polarization-range?
35      set enp-seats-begin enp-seats?
36      set enp-votes-begin enp-votes?
37
38      if save-party-and-polarization-stats? [
39        file-open "runstats.txt"
40        file-print file-stats-header
41      ]
42    end
43
44    to go
45      if not any? parties [
46        set error-code "no party in party list"
47        stop
48      ]
49
50      ; Parteien positionieren sich neu
51      ask parties [party-positioning]
52
53      ; Das aktuelle Ergebnis an der Urne wird berechnet
54      calculate-election-result
55
56      ; ... und auch angezeigt
57      if display? [show-election-results]
58
59      ; Der Output wird gemessen und es geht ggf. eine Runde weiter
60      measure-parties
61      tick
62
63      if save-party-and-polarization-stats? [file-print print-to-file-stats]
64
65      ; ... außer es ist die maximale Anzahl an Iterationen erreicht. […]
66      if (ticks > max-ticks?) [
67        measure-output
68        set polarization-sd-seats-end polarization-sd-seats?
69        set polarization-sd-votes-end polarization-sd-votes?
70        set polarization-range-end    polarization-range?
71        set enp-seats-end enp-seats?
72        set enp-votes-end enp-votes?
73        file-close-all
74        stop
75      ]
76
77      ; Wenn ein substanzieller Fehler (Bug) aufgetreten ist, wird […]
78      if error-code != "none" [stop]
79    end
```

breeds_and_globals.nls

```
80    globals [
81      error-code                      ; dokumentiert Fehler im Modell
82      district-relevants              ; Variable zur Erstellung von Distrikten […]
83      relevant-patches                ; speichert alles relevanten Patches
```

```
84      parties                          ; speichert alle Parteien
85      issue-types                      ; Liste der Issue-Typen
86      issue-relevance                  ; Liste der Relevanz der Issues
87      issue-none-valence               ; Liste der Issues, welche nicht Valenz sind
88      number-of-issues                 ; Anzahl der Issues
89      number-of-coalitions             ; Variable zur Berechnung des Koalitionspotentials
90
91      start-cabinet-is-electoral-connected? ;
92
93      max-xcor!                        ; Maximale Position auf der X-Achse
94      max-ycor!                        ; Maximale Position auf der Y-Achse
95      min-xcor!                        ; Minimale Position auf der X-Achse
96      min-ycor!                        ; Minimale Position auf der Y-Achse
97      max-distance!                    ; Maximal mögliche Distanz
98
99      party-color-list                 ; Speichert ein Farbset für die Parteien
100     time-consumption                 ; verbrauchte Zeit für den Durchlauf
101     emergency-coalition              ; speichert, ob die Backup-Option […]
102
103     cabinet-opposition               ; Speichert als Output, welche Partei zum […]
104     parties-xcor-positions           ; Speichert als Output die X-Position der Parteien
105     parties-ycor-positions           ; Speichert als Output die Y-Position der Parteien
106     parties-seats                    ; Speichert als Output die Sitze der Parteien
107     parties-votes                    ; Speichert als Output die Stimmen der Parteien
108     parties-salience                 ; Speichert als Output die Betonungen der Parteien
109     parties-position                 ; Speichert als Output die Positionwn der Parteien
110
111     polarization-sd-votes-begin
112     polarization-range-begin
113     polarization-sd-seats-begin
114
115     polarization-sd-votes-end
116     polarization-range-end
117     polarization-sd-seats-end
118
119     enp-seats-begin
120     enp-votes-begin
121     enp-seats-end
122     enp-votes-end]
123
124   ; turtles sind die Parteien im Modell
125   turtles-own [
126     cabinet                          ; Ist die Partei Teil des aktuellen Kabinettes […]
127     votes-total                      ; Wählerstimmenanteil der Partei (0-1)
128     seats-total                      ; Sitzanteil der Partei (0-1)
129     utility                          ; Aktueller Nutzen der Partei (0-1)
130     policy-pref-x                    ; Präferenz der Partei auf der X-Achse
131     policy-pref-y                    ; Präferenz der Partei auf der Y-Achse
132     coalition-potential              ; Koalitionspotential der Partei
133     prefered-coalition               ; Die präferierte Koalition der Partei
134     issues-position-p                ; Parteiposition hinsichtlich der Issues
135     issues-salience-p                ; Standardisierte Salienz der Themen […]
136     issues-salience-p-not-standard   ; Technische Variablen zur unstandisierten […]
137     old-utility                      ; Variablen für reaktive Parteien
138     old-salience                     ; Variablen für reaktive Parteien
139     old-position                     ; Variablen für reaktive Parteien
140     control                          ; Technische Variable, […]
141     votes-control]                   ; Technische Variable, […]
142
143   ; patches sind die Wähler im Modell
144   patches-own [
145     heuristic-v                      ; Speichert [..] Aspekte der Wählerheuristik […]
146     non-policy-factor                ; Speichert die non-policy-Gewichtung […]
147     votes                            ; Die Anzahl der Wähler auf der Issue
148     issues-position-v                ; Die Position der Wähler hinsichtlich der Issues
149     issues-relevance-v               ; Die Relevanz der verschiedenen Issues […]
150     parties-utility                  ; Liste über den Nutzen […]
151     parties-utility-temp             ; Technische Variable, welche […]
152     district-voters                  ; Wählerstimmen nach Distrikten unterteilt […]
153     temp                             ; technische Variablen, welche in einigen […]
```

```
154     party-elected]                    ; gewählte Partei durch die Wähler auf dem Patch
```

setup.nls

```
155   to random-setup
156     ; initiert die Parameter für einen zufälligen Durchlauf des Modells
157
158     set max-xcor? item (random 3)[10 20 30]
159     set max-ycor? item (random 10)[10 10 10 10 10 10 10 10 20 30] ;max?
160     set max-ticks? 40 + random 41
161     set proaktiv? TRUE ;item (random 2) [TRUE FALSE]
162     set information-lvl? 1 ;(1 + random 3)
163     if not proaktiv? [set max-ticks? max-ticks? * information-lvl?]
164     set ban-leap-frogging? item (random 2) [TRUE FALSE]
165
166     set number-of-left-right-issues? 1
167     set left-right-relevance? random_100_but_gt_1 * number-of-left-right-issues?
168
169     let what-additional-dimension random 4
170
171     ifelse what-additional-dimension = 1 [set number-of-second-dimension-issues? 1][set number-
      of-second-dimension-issues? 0]
172     set second-dimension-relevance? random_100 * number-of-second-dimension-issues?
173
174     ifelse what-additional-dimension = 2 [set number-of-uncorrelated-issues? 1][set number-of-
      uncorrelated-issues? 0]
175     set uncorrelated-relevance? random_100 * number-of-uncorrelated-issues?
176
177     ifelse what-additional-dimension = 3 [set number-of-valence-issues? 1][set number-of-
      valence-issues? 0]
178     set valence-relevance? random_100 * number-of-valence-issues?
179
180     set relevance-lr? random_100
181     set relevance-sd? random_100 / 2
182     if what-additional-dimension = 0 [
183       set relevance-lr? 0
184       set relevance-sd? 0
185     ]
186
187     set voter-modal? 1 + random 2
188     set voter-sd? 75 + random 131
189     set voter-sd? voter-sd? * (max-xcor? / 100)
190     if  voter-modal? = 2 [set voter-sd? voter-sd? * .75]
191     set voter-n? 20000 + random 30001
192
193     set electoral-system? item (random 2)["Proportional" "Plurality"]
194     ifelse electoral-system? = "Proportional" [set electoral-threshold? ((random 11) / 100)][set
      electoral-threshold? 0]
195     ifelse electoral-system? = "Proportional" [set number-of-districts? 1][set number-of-
      districts? 20 + random 31]
196     ifelse electoral-system? = "Proportional" [set district-homogenity? 0][set district-
      homogenity? ((random 101) / 100 )]
197
198     set number-of-parties? 2 + random 11
199     set center-cabinet? item (random 2)[TRUE FALSE]
200     set cabinet-cooperation? random 3
201     set minimum-size-coalition? item (random 3)[0.4 0.5 0.6]
202     set stay-connected? item (random 2)[TRUE FALSE]
203     set minimize-range? item (random 2)[TRUE FALSE]
204
205     set vote-seeking? random_100_but_gt_1
206     ifelse random 2 = 1[set office-seeking? random_100_but_gt_1][set office-seeking? 0]
207     ifelse random 2 = 1[set policy-seeking? random_100_but_gt_1][set policy-seeking? 0]
208     set votes-seats? random_100
209     if  random 2 = 1 [set votes-seats? item (random 2) [0 1]]
210
211     set relevance-positioning? random_100
212     if random 2 = 1 [set relevance-positioning? 1]
213     set policy-movement? (5 + random 10) / 100
214
```

```
    ifelse random 2 = 1 [set strategic-voting-chance? random_100][set strategic-voting-chance? 0]

    ; Non-Voting Utility ---------------------------
    ifelse random 2 = 1 [
      set non-voting-utility? 0
      set nvu-lr? 0
    ][
      set non-voting-utility? (random 51) / 100
      set nvu-lr? precision (1 - (random 201) / 100) 2
    ]

    ; Cabinet-Evaluation? ---------------------------
    ifelse random 2 = 1 [
      set cabinet-evaluation? 0
      set ce-lr? 0
    ][
      set cabinet-evaluation? (0.5 - ((random 101) / 100)) * 0.8
      set ce-lr? precision (1 - (random 201) / 100) 2
    ]

    ; Policy-Weight? -------------------------------
    set policy-weight? random_100_but_gt_1

    ifelse random 2 = 1 [
      set pw-lr? 0
    ][
      set pw-lr? precision (1 - (random 201) / 100) 2
    ]

    ; Non-Policy Term Weight? ----------------------
    ifelse random 2 = 1 [
      set non-policy-weight? 0
      set npw-lr? 0
      set npw-distance? 0
      set npw-sd? 0
    ][
      set non-policy-weight? (random 101) / 100
      set npw-lr? precision (1 - (random 201) / 100) 2
      set npw-distance? (random 51) / 100
      set npw-sd? (random 31) / 100
    ]

    let balance non-policy-weight? + policy-weight?
    if (balance = 0) [set balance 1]
    set policy-weight? precision (policy-weight? / balance) 2
    set non-policy-weight? 1 - policy-weight?
    set used-seed new-seed
  end

  to rebalance-input
    let sum_x left-right-relevance? + second-dimension-relevance? + uncorrelated-relevance? + valence-relevance?
    if (sum_x > 0)[
      set left-right-relevance?       precision (left-right-relevance? / sum_x) 3
      set second-dimension-relevance? precision (second-dimension-relevance? / sum_x) 3
      set uncorrelated-relevance?     precision (uncorrelated-relevance? / sum_x) 3
      set valence-relevance?          precision (1 - (left-right-relevance? + second-dimension-relevance? + uncorrelated-relevance?)) 3
      if (valence-relevance? < 0) [set valence-relevance? 0]
    ]

    set sum_x vote-seeking? + office-seeking? + policy-seeking?
    if (sum_x > 0)[
      set vote-seeking?   precision (vote-seeking? / sum_x) 3
      set office-seeking? precision (office-seeking? / sum_x) 3
      set policy-seeking? precision (1 - (vote-seeking? + office-seeking?)) 3
      if (policy-seeking? < 0) [set policy-seeking? 0]
    ]
  end
```

```
282
283   to setup-competition
284     ; Erstellt den Parteienwettbewerb für einen Durchlauf
285     ; max-xcor? und max-ycor? grenzen den Wettbewerbsbereich ein
286     set relevant-patches patches with [pxcor <= max-xcor? and pycor <= max-ycor?]
287     if not is-patch-set? relevant-patches[
288       set error-code "no patch set"
289       print "stop"
290       stop
291     ]
292
293     ; create parties, voters and issues
294     create-parties number-of-parties?
295     create-voter-distribution voter-sd? voter-n? voter-modal?
296     issue-setup
297     create-voter-heuristic
298     if electoral-system? = "Proportional" [
299       set number-of-districts? 1
300       set district-homogenity? 0
301     ]
302     if electoral-system? = "Plurality" [
303       create-districts
304     ]
305     set max-xcor! max [pxcor] of relevant-patches
306     set max-ycor! max [pycor] of relevant-patches
307     set min-xcor! min [pxcor] of relevant-patches
308     set min-ycor! min [pycor] of relevant-patches
309     set parties turtles with [who < 100]
310     if not is-agentset? parties [
311       set error-code "no parties"
312       stop
313     ]
314     set max-distance! (sqrt(((max-xcor! - min-xcor!)^2) + ((max-ycor! - min-ycor!)^2)))
315     voter-nonpolicy-setup
316   end
317
318   to create-parties [#party-n]
319
320     if display? [
321       set party-color-list [red blue green yellow brown grey orange lime turquoise cyan sky
      violet magenta pink]
322       set-default-shape turtles "triangle 2"
323     ]
324
325     create-turtles #party-n [
326       let x -1
327       let y -1
328       while [(count turtles with [xcor = x and ycor = y] > 0 or x < 0 or x > max-xcor? or y < 0
      or y > max-ycor?) and error-code != "time-out party placement"][
329         set x normal-rnd-value (max-xcor? / 2) (max-xcor? / 4) 0 max-xcor? 0
330         set y normal-rnd-value (max-ycor? / 2) (max-ycor? / 4) 0 max-ycor? 0
331         if timer > 30 [
332           set error-code "time-out party placement"
333         ]
334       ]
335       setxy x y
336       set control 0
337       set size 4
338       set utility 1
339       set prefered-coalition (list)
340       ifelse max-xcor? = 0 [set policy-pref-x 0][set policy-pref-x normal-rnd-value xcor (max-
      xcor? * 0.05) 0 max-xcor? 0]
341       ifelse max-ycor? = 0 [set policy-pref-y 0][set policy-pref-y normal-rnd-value ycor (max-
      ycor? * 0.05) 0 max-ycor? 0]
342       if display? [set color item who party-color-list]
343     ]
344   end
345
346   to create-voter-distribution [#sd #n #modal]
347     let sd-x (#sd / 100) * max-xcor?
```

```
348
349     ifelse #modal = 2 [
350       ask relevant-patches [
351         let x1 (pxcor - (max-xcor? * 0.8)) / sd-x
352         let y1 (pycor - (max-ycor? / 2)) / sd-x
353         set votes round ((#n * exp (-0.5 * ( x1^2 + y1^2)) / (2 * pi * sd-x^2)) / 2)
354         set x1 (pxcor - (max-xcor? * 0.2)) / sd-x
355         set y1 (pycor - (max-ycor? / 2)) / sd-x
356         set votes votes + round ((#n * exp (-0.5 * ( x1^2 + y1^2))/ (2 * pi * sd-x^2)) / 2)
357       ]
358     ][
359       ask relevant-patches [
360         let x1 (pxcor - (max-xcor? / 2)) / sd-x
361         let y1 (pycor - (max-ycor? / 2)) / sd-x
362         set votes round (#n * exp (-0.5 * ( x1 ^ 2 + y1 ^ 2)) / (2 * pi * sd-x ^ 2))
363       ]
364     ]
365
366     if display? = true [
367       show-voter-distribution
368     ]
369   end
370
371   to issue-setup
372     set issue-types (list)
373     set issue-relevance (list)
374     set issue-none-valence (list)
375
376     let sum-of-relevance left-right-relevance? * number-of-left-right-issues? + number-of-
      second-dimension-issues? * second-dimension-relevance? + number-of-valence-issues? * valence-
      relevance? + number-of-uncorrelated-issues? * uncorrelated-relevance?
377     if (sum-of-relevance = 0)[set sum-of-relevance 1]
378     let i 0
379
380     repeat number-of-valence-issues? [
381       set issue-types lput "valence" issue-types
382       set issue-relevance lput (valence-relevance? / sum-of-relevance) issue-relevance
383       set i i + 1
384     ]
385
386     repeat number-of-left-right-issues? [
387       set issue-types lput "first-dimension" issue-types
388       set issue-relevance lput (left-right-relevance?/sum-of-relevance) issue-relevance
389       set issue-none-valence lput i issue-none-valence
390       set i i + 1
391     ]
392
393     repeat number-of-second-dimension-issues? [
394       set issue-types lput "second-dimension" issue-types
395       set issue-relevance lput (second-dimension-relevance?/sum-of-relevance) issue-relevance
396       set issue-none-valence lput i issue-none-valence
397       set i i + 1
398     ]
399
400     repeat number-of-uncorrelated-issues? [
401       set issue-types lput "uncorrelated" issue-types
402       set issue-relevance lput (uncorrelated-relevance? / sum-of-relevance) issue-relevance
403       set issue-none-valence lput i issue-none-valence
404       set i i + 1
405     ]
406
407     set number-of-issues number-of-left-right-issues? + number-of-second-dimension-issues? +
      number-of-valence-issues? + number-of-uncorrelated-issues?
408     set i 0
409
410     ask relevant-patches [
411       set i 0
412       set issues-relevance-v (list)
413       set issues-position-v (list)
414       let lr-p pxcor / max [pxcor] of relevant-patches
```

```
415       let lr-w (abs(lr-p - 0.5) * 2)
416       let lr_v (abs((pxcor / max-xcor?) - 0.5) * 2)
417
418       repeat number-of-issues [
419         if (item i issue-types) = "valence"            [set issues-position-v lput 1 issues-
      position-v]
420         if (item i issue-types) = "first-dimension"  [set issues-position-v lput (((pxcor / max
      [pxcor] of relevant-patches) * 2) - 1) issues-position-v ]
421         if (item i issue-types) = "second-dimension" [set issues-position-v lput (((pycor / max
      [pycor] of relevant-patches) * 2) - 1) issues-position-v]
422         if (item i issue-types) = "uncorrelated"     [set issues-position-v lput ((((random 101)
      / 100) * 2) - 1) issues-position-v]
423         let m (item i issue-relevance)
424         if (item i issue-types) = "first-dimension"   [set m estimate-value (item i issue-
      relevance) relevance-lr? lr_v relevance-sd? FALSE]
425         set issues-relevance-v lput m issues-relevance-v
426         set i i + 1
427       ]
428
429         let s sum issues-relevance-v
430         if s = 0 [set s 1]
431         let l2 issues-relevance-v
432         set issues-relevance-v (list)
433         let i-2 0
434         repeat number-of-issues [
435           set issues-relevance-v lput ((item i-2 l2) / s) issues-relevance-v
436           set i-2 i-2 + 1
437         ]
438     ]
439
440     ask turtles [
441       set i 0
442       set issues-salience-p (list)
443       set issues-position-p (list)
444       set issues-salience-p-not-standard (list)
445
446       repeat number-of-issues [
447         if (item i issue-types) = "valence" [set issues-position-p lput 1 issues-position-p]
448         if (item i issue-types) = "first-dimension"  [set issues-position-p lput (((pxcor / max
      [pxcor] of relevant-patches) * 2) - 1) issues-position-p]
449         if (item i issue-types) = "second-dimension" [set issues-position-p lput (((pycor / max
      [pycor] of relevant-patches) * 2) - 1) issues-position-p]
450         if (item i issue-types) = "uncorrelated"     [set issues-position-p lput ((((random 101)
      / 100) * 2) - 1) issues-position-p]
451         set issues-salience-p lput (item i issue-relevance) issues-salience-p
452         set i i + 1
453       ]
454       set issues-salience-p-not-standard issues-salience-p
455     ]
456
457     if display? [
458       set i 0
459       output-print "Structure of Issues:"
460       repeat number-of-issues [
461         output-type "Issue " output-type (i + 1) output-type ": " output-type (item i issue-
      types) output-type " " output-type (precision (item i issue-relevance) 2) output-print "."
462         set i i + 1
463       ]
464     ]
465   end
466
467   to create-voter-heuristic
468     ask relevant-patches [
469       let lr_v (abs((pxcor / max-xcor?) - 0.5) * 2)
470       set heuristic-v (list)
471       let  pw (estimate-value-without-error     policy-weight?  pw-lr? lr_v FALSE)
472       if (pw = 0)[set pw policy-weight?]
473       let npw (estimate-value-without-error non-policy-weight? npw-lr? lr_v FALSE)
474       let balance pw + npw
475       if (balance = 0) [set balance 1]
```

```
476       set pw precision (pw / balance) 2
477       set npw 1 - pw
478       set heuristic-v lput pw heuristic-v
479       set heuristic-v lput npw heuristic-v
480       set heuristic-v lput (((1 + (random 100)) / 100) <= strategic-voting-chance?) heuristic-
      v
481       set heuristic-v lput (estimate-value-without-error non-voting-utility? nvu-lr? lr_v FALSE)
      heuristic-v
482       set heuristic-v lput (estimate-value-without-error cabinet-evaluation? ce-lr? lr_v TRUE)
      heuristic-v
483     ]
484   end
485
486   to create-districts
487     let counter (list)
488     let i 1
489     while [i <= number-of-districts?][
490       set counter lput 0 counter
491       set i i + 1
492     ]
493
494     let mean-district-votes round (voter-n? / number-of-districts?)
495     ask relevant-patches [
496       set temp 0
497       set district-voters counter
498     ]
499     set i 1
500     let y 0
501     let random-list (range 1 (number-of-districts? + 1))
502     let bedingung true
503     while [bedingung][
504       set random-list shuffle random-list
505       set y item 0 random-list
506       ifelse length random-list = 1 [
507         set random-list (range 1 (number-of-districts? + 1))
508       ][
509         set random-list remove-item 0 random-list
510       ]
511       ifelse ((random 100) / 100) >= district-homogenity? [
512         ask one-of relevant-patches with [temp = 0][
513           set district-voters replace-item (y - 1) district-voters ((item (y - 1) district-
      voters) + 1)
514           if sum district-voters >= votes [set temp 1]
515         ]
516       ][
517         let stand_value (max-xcor? / number-of-districts?) * y
518         ask min-one-of relevant-patches with [temp = 0][distance (patch stand_value (random max-
      ycor?)) ][
519           set district-voters replace-item (y - 1) district-voters ((item (y - 1) district-
      voters) + 1)
520           if sum district-voters >= votes [set temp 1]
521         ]
522       ]
523       ifelse is-number? count relevant-patches with [temp = 0][
524         if not (count relevant-patches with [temp = 0] > 0)[set bedingung false]
525       ][
526         set bedingung false
527       ]
528     ]
529   end
530
531   to voter-nonpolicy-setup
532     ask relevant-patches [
533       set non-policy-factor (list)
534       set non-policy-factor (non-policy 1 npw-sd? npw-distance?)
535     ]
536   end
537
538   to-report non-policy [#v #sd #distance]
539     let i 0
```

```
    let l (list)
    repeat count turtles [
      ask turtle i [
        let x1 (#v * (1 - (party-voter-distance self myself)))
        if x1 > 1 [set x1 1]
        if x1 < 0 [set x1 0]
        set x1 normal-rnd-value x1 #sd 3 1 0
        set l lput x1 l
      ]
     set i i + 1
    ]
    report l
  end
```

functions.nls

```
  to-report normal-rnd-value [#mean #sd #precision #max #min]
    let x-value #min - 100
    let count-iterations 0
    while [(x-value < #min or x-value > #max) and count-iterations < 100][
      set x-value precision (random-normal #mean #sd) #precision
      set count-iterations count-iterations + 1
    ]
    if count-iterations = 100 [set x-value #mean]
    report x-value
  end

  to show-voter-distribution
      ask relevant-patches [set pcolor scale-color red votes 0 max [votes] of relevant-patches]
  end

  to show-district-distribution [#id]
    let max-value max-district-voters-of-i #id
    ask relevant-patches [set pcolor scale-color red (item #id district-voters) 0 max-value]
  end

  to-report estimate-value2 [#cons #lr-b #lr #error-b]
    let error_v random_100
    let cons_b (1 - #lr-b - #error-b)
    report (cons_b * #cons + #lr-b * #lr + #error-b * error_v)
  end

  to-report estimate-value [#cons #lr-b #lr #error-b #-1to1]
    let max_distance .5
    if (#-1to1) [set max_distance 1]
    let lr (#lr * 2) - 1

    let output #cons + #lr-b * (lr * max_distance)

    if output > 1 [set output 1]
    if (    #-1to1)[if output < -1 [set output -1]]
    if (not #-1to1)[if output < 0  [set output  0]]

    let error_v random_100
    set output (1 - #error-b) * output + #error-b * error_v
    report output
  end

  to-report estimate-value-without-error2 [#cons #lr-b #lr #-1to1]
    let cons_b (1 - abs(#lr-b))
    let lr_componente #lr-b * #lr
    if (#-1to1) [set lr_componente #lr-b * (1 - (#lr * 2))]
    let output (cons_b * #cons + lr_componente)

    if output > 1 [set output 1]
    if (    #-1to1)[if output < -1 [set output -1]]
    if (not #-1to1)[if output < 0  [set output  0]]
    report output
  end

  to-report estimate-value-without-error [#cons #lr-b #lr #-1to1]
```

```
608     let max_distance .5
609     if (#-1to1) [set max_distance 1]
610     let lr (#lr * 2) - 1
611
612     let output #cons + #lr-b * (lr * max_distance)
613
614     if output > 1 [set output 1]
615     if (    #-1to1)[if output < -1 [set output -1]]
616     if (not #-1to1)[if output < 0  [set output  0]]
617     report output
618   end
619
620   to-report random_100
621     report (random 101) / 100
622   end
623
624   to-report random_100_but_gt_1
625     report (1 + (random 100)) / 100
626   end
627
628   to-report max-district-voters-of-i [#item]
629     let i (list)
630     ask relevant-patches [set i lput item #item district-voters i]
631     report max i
632   end
633
634   to-report sum-district-voters-of-i [#item]
635     let i (list)
636     ask relevant-patches [set i lput item #item district-voters i]
637     report sum i
638   end
639
640   to-report sum-district-voters
641     let x 1
642     ask relevant-patches [
643       ifelse is-list? x [set x (map + x district-voters)][set x district-voters]
644     ]
645     report x
646   end
647
648   to-report party-voter-distance [#x #y]
649     let distance-between 0
650     let pos-x [issues-position-p] of #x
651     let pos-y [issues-position-v] of #y
652     let rel-x [issues-salience-p] of #x
653     let rel-y [issues-relevance-v] of #y
654     let rel (map + rel-y rel-x)
655     let rel_stand (map ([[x] -> x / sum rel]) rel)
656     let dis (map ([[x y] -> (abs(x - y) / 2) ]) pos-y pos-x)
657     set distance-between sum (map * dis rel)
658     report distance-between
659   end
660
661   to show-issue-position [#i]
662     ask relevant-patches [
663       let p item #i issues-position-v
664       set p p * 0.5
665       ifelse p > 0 [set pcolor scale-color green (3 * p) 2 0][
666         ifelse p < 0 [set pcolor scale-color red ((3 * -1 * p)) 2 0][set pcolor white]
667       ]
668     ]
669   end
670
671   to show-issue-relevance [#i]
672     ask relevant-patches [
673       let p item #i issues-relevance-v
674       set p p * 0.5
675       ifelse p > 0 [
676         set pcolor scale-color green (3 * p) 2 0
677       ][
```

```
678         ifelse p < 0 [set pcolor scale-color red ((3 * -1 * p)) 2 0][set pcolor white]
679       ]
680     ]
681   end
682
683   to show-non-policy-factor-of-party [#i]
684     let mean_non_policy 0
685     ask relevant-patches [
686       let p item #i non-policy-factor
687       set p p * 0.5
688       set mean_non_policy mean_non_policy + item #i non-policy-factor
689       ifelse p > 0 [
690         set pcolor scale-color green (3 * p) 2 0
691       ][
692         ifelse p < 0 [set pcolor scale-color red ((3 * -1 * p)) 2 0][set pcolor white]
693       ]
694     ]
695     print mean_non_policy / count relevant-patches
696   end
697
698   to utility-factor-of-patches [#i]
699     ask relevant-patches [
700       let p item #i heuristic-v
701       set p p * 0.5
702       ifelse p > 0 [
703         set pcolor scale-color green (3 * p) 2 0
704       ][
705         ifelse p < 0 [set pcolor scale-color red ((3 * -1 * p)) 2 0][set pcolor white]
706       ]
707     ]
708   end
709
710   to voters-issue-relevance [#i]
711     ask relevant-patches [
712       let p item #i issues-relevance-v
713       set p p * 0.5
714       ifelse p > 0 [
715         set pcolor scale-color green (3 * p) 2 0
716       ][
717         ifelse p < 0 [set pcolor scale-color red ((3 * -1 * p)) 2 0][set pcolor white]
718       ]
719     ]
720   end
721
722   to-report give-xcor
723     let i 0
724     let output (list)
725
726     repeat count parties [
727       ask turtle i [set output lput xcor output]
728       set i i + 1
729     ]
730     report output
731   end
732
733   to-report test-xcor-relationship [#xcors #xcor]
734     let output (list)
735     foreach (#xcors) [[x] -> set output lput (relationship-numbers x #xcor) output]
736     report output
737   end
738
739   to-report relationship-numbers [#x #y]
740     let output 0
741     if #x > #y [set output  1]
742     if #x < #y [set output -1]
743     report output
744   end
745
746   to-report compare-two-lists [#list1 #list2]
747     let output TRUE
```

```
748     foreach (range length #list1) [[x] -> if item x #list1 != item x #list2 [set output FALSE]]
749     report output
750   end
751
752   to show-party-voter-distance [#i]
753     ask relevant-patches [
754       let x 0
755       ask turtle #i [set x party-voter-distance self myself]
756       set pcolor scale-color red (3 * x) 2 0
757     ]
758   end
759
760   to show-proximity-issue-utility [#i]
761     ask relevant-patches [
762       let x proximity-issue-utility #i
763       set pcolor scale-color green (x) 2 0
764     ]
765   end
766
767   to-report give-lr-issues
768     let i 0
769     if number-of-issues > 1 [if item 1 issue-types = "first-dimension" [set i 1]]
770     let y 0
771     let output (list)
772     repeat count parties [
773       ask turtle y [set output lput (item i issues-position-p) output]
774       set y y + 1
775     ]
776     report output
777   end
778
779   to-report polarization-range?
780     ifelse any? turtles with [seats-total > 0][
781       report (max [xcor] of turtles with [seats-total > 0] - min [xcor] of turtles with [seats-
      total > 0]) / max-xcor?
782     ][
783       report 0
784     ]
785   end
786
787   to-report enp-seats?
788     let output 0
789     ask turtles [set output output + (seats-total ^ 2)]
790     report 1 / output
791   end
792
793   to-report enp-votes?
794       let output 0
795     ask turtles [set output output + (votes-total ^ 2)]
796     report 1 / output
797   end
798
799   to-report polarization-range?2
800     report (max [xcor] of turtles with [seats-total > 0]) / max-xcor? - (min [xcor] of turtles
      with [seats-total > 0]) / max-xcor?
801   end
802
803   to-report polarization-sd-seats?
804     let center_of_gravity 0
805     ask turtles with [seats-total > 0][set center_of_gravity center_of_gravity  + (xcor / max-
      xcor?) * seats-total]
806     let polarization 0
807     ask turtles with [seats-total > 0][
808       set polarization polarization +((center_of_gravity - (xcor / max-xcor?)) ^ 2) * seats-
      total
809     ]
810     set polarization sqrt polarization
811     report polarization
812   end
813
```

```
814  to-report polarization-sd-votes?
815    let center_of_gravity 0
816    ask turtles with [votes-total > 0][set center_of_gravity center_of_gravity  + (xcor / max-
     xcor?) * votes-total]
817
818    let polarization 0
819    ask turtles with [votes-total > 0][set polarization polarization + ((center_of_gravity -
     (xcor / max-xcor?)) ^ 2) * votes-total]
820    set polarization sqrt polarization
821    report polarization
822  end
823
824  to-report file-stats-header
825    let print-string ""
826    let i 0
827    repeat count turtles [
828      set i i + 1
829      set print-string (word print-string "party" i "xcor;")
830      set print-string (word print-string "party" i "ycor;")
831      set print-string (word print-string "party" i "seats;")
832      set print-string (word print-string "party" i "votes;")
833    ]
834    set print-string (word print-string "tick;")
835    set print-string (word print-string "polarization")
836    report print-string
837  end
838
839  to-report print-to-file-stats
840    let print-string ""
841    let i 0
842    repeat count turtles [
843      ask turtle i [
844        set print-string (word print-string (precision xcor 4) ";" (precision ycor 4) ";"
     (precision seats-total 4) ";" (precision votes-total 4) ";")
845      ]
846      set i i + 1
847    ]
848    set print-string (word print-string ticks ";")
849    set print-string (word print-string (precision ((polarization-sd-seats? * 2 + polarization-
     range?) / 2) 4))
850    report print-string
851  end
```

parties.nls

```
852  to party-positioning
853    ifelse proaktiv? [issue-based-positioning-proaktiv][issue-based-positioning-reaktiv]
854  end
855
856  to issue-based-positioning-reaktiv
857    no-display
858    let test-leap-frogging (list)
859    if ban-leap-frogging? [set test-leap-frogging (test-xcor-relationship give-xcor [xcor] of
     turtle who)]
860
861    if old-utility > utility [
862      set issues-salience-p-not-standard old-salience
863      set issues-position-p old-position
864    ]
865
866    set old-salience issues-salience-p-not-standard
867    set old-position issues-position-p
868    set old-utility utility
869
870      ifelse ((1 + random 100)/ 100) <= relevance-positioning? [
871        ; positioning
872        if (empty? issue-none-valence) [
873          set error-code "non none-valence issues but positioning"
874          stop
875        ]
876
```

```
877         let i one-of issue-none-valence
878         let p random 2
879         if p = 0 [set p -1]
880
881         let issues-p-temp issues-position-p
882         let new-issue-emphasize ((item i issues-position-p) + (policy-movement? * p))
883         if  new-issue-emphasize < -1 [set new-issue-emphasize -1]
884         if  new-issue-emphasize > 1 [set new-issue-emphasize 1]
885
886         set issues-position-p replace-item i issues-position-p new-issue-emphasize
887         set-position-based-on-issues
888         if ban-leap-frogging? [
889           if not (compare-two-lists (test-xcor-relationship give-xcor [xcor] of turtle who)
    test-leap-frogging) [
890             set issues-position-p old-position
891             set-position-based-on-issues
892           ]
893         ]
894       ][
895         ; relevance
896         let i random number-of-issues
897         let p random 2
898         if p = 0 [set p -1]
899
900         let issues-p-temp issues-salience-p-not-standard
901         let new-issue-emphasize ((item i issues-salience-p-not-standard) + (policy-movement? *
    p))
902         if new-issue-emphasize < 0 [set new-issue-emphasize 0]
903         if new-issue-emphasize > 1 [set new-issue-emphasize 1]
904
905         set issues-salience-p-not-standard replace-item i issues-salience-p-not-standard new-
    issue-emphasize
906         rebalance-relevance
907         set-position-based-on-issues
908         if ban-leap-frogging? [
909           if not (compare-two-lists (test-xcor-relationship give-xcor [xcor] of turtle who)
    test-leap-frogging) [
910             set issues-salience-p-not-standard old-salience
911             rebalance-relevance
912             set-position-based-on-issues
913           ]
914         ]
915       ]
916
917     display
918   end
919
920   to issue-based-positioning-proaktiv
921     no-display
922     let best-utility calculate-utility
923     let test-leap-frogging (list)
924     let leap-frogging-utility-weight 1
925     if ban-leap-frogging? [set test-leap-frogging (test-xcor-relationship give-xcor [xcor] of
    turtle who)]
926
927     repeat (information-lvl?) [
928       ifelse ((1 + random 100)/ 100) <= relevance-positioning? [
929         ; positioning
930         if (empty? issue-none-valence) [
931           set error-code "non none-valence issues but positioning"
932           stop
933         ]
934
935         let i one-of issue-none-valence
936         let p random 2
937         if p = 0 [set p -1]
938         let issues-p-temp issues-position-p
939         let new-issue-emphasize ((item i issues-position-p) + (policy-movement? * p))
940         if  new-issue-emphasize < -1 [set new-issue-emphasize -1]
941         if  new-issue-emphasize > 1 [set new-issue-emphasize 1]
```

```
942         set issues-position-p replace-item i issues-position-p new-issue-emphasize
943         set-position-based-on-issues
944         calculate-election-result
945         if ban-leap-frogging? [
946           if not (compare-two-lists (test-xcor-relationship give-xcor [xcor] of turtle who) test-leap-frogging) [set leap-frogging-utility-weight -1]
947         ]
948
949         ifelse (calculate-utility * leap-frogging-utility-weight) > best-utility [
950           set best-utility calculate-utility
951         ][
952           set issues-position-p issues-p-temp
953           set-position-based-on-issues
954           calculate-election-result
955         ]
956       ][
957         ; relevance
958         let i random number-of-issues
959         let p random 2
960         if p = 0 [set p -1]
961
962         let issues-p-temp issues-salience-p-not-standard
963         let new-issue-emphasize ((item i issues-salience-p-not-standard) + (policy-movement? * p))
964         if new-issue-emphasize < 0 [set new-issue-emphasize 0]
965         if new-issue-emphasize > 1 [set new-issue-emphasize 1]
966
967         set issues-salience-p-not-standard replace-item i issues-salience-p-not-standard new-issue-emphasize
968         rebalance-relevance
969         set-position-based-on-issues
970         calculate-election-result
971
972         if ban-leap-frogging? [
973           if not (compare-two-lists (test-xcor-relationship give-xcor [xcor] of turtle who) test-leap-frogging) [set leap-frogging-utility-weight -1]
974         ]
975
976         ifelse (calculate-utility * leap-frogging-utility-weight) > best-utility [
977           set best-utility calculate-utility
978         ][
979           set issues-salience-p-not-standard issues-p-temp
980           rebalance-relevance
981           set-position-based-on-issues
982           calculate-election-result
983         ]
984       ]
985     ]
986     display
987   end
988
989   to-report calculate-utility
990     ; list that indicates the following aspects: […]
991     let recent-utility 0
992
993     let policy-utility 0
994     if  policy-seeking? > 0 [set policy-utility policy-seeking-utility]
995
996     let vote-utility 0
997     if  vote-seeking? > 0   [set vote-utility   vote-seeking-utility]
998
999     let office-utility 0
1000    if  office-seeking? > 0 [set office-utility office-seeking-utility]
1001
1002    ifelse cabinet-cooperation? < 2 or cabinet = 0 [
1003      set recent-utility (policy-seeking? * policy-utility) + (vote-seeking? * vote-seeking-utility) + (office-seeking? * office-utility) ;[…]
1004    ][
1005      set recent-utility (policy-seeking? * policy-utility) + ((office-seeking? + vote-seeking?) * office-seeking-utility)
```

```
1006    ]
1007    report recent-utility
1008  end
1009
1010  to-report policy-seeking-utility
1011    report 1 - ((sqrt(((xcor - policy-pref-x) ^ 2) + ((ycor - policy-pref-y) ^ 2))) / max-
      distance!)
1012  end
1013
1014  to-report vote-seeking-utility
1015    report ((1 - votes-seats?) * votes-total) + (votes-seats? * seats-total)
1016  end
1017
1018  to-report office-seeking-utility
1019    let u coalition-potential
1020    if (cabinet = 1 and stay-connected?) [set u (u + is-connected-weight) / 2]
1021    if (cabinet = 1 and minimize-range?) [set u (u * .99 + (1 - range_cabinet) * .1)]
1022    report u
1023  end
1024
1025  to set-position-based-on-issues
1026    let lr -1
1027    let val -1
1028    let second-dim -1
1029
1030    if item 0 issue-types = "first-dimension" [set lr 0]
1031    if number-of-issues > 1 [if item 1 issue-types = "first-dimension" [set lr 1]]
1032
1033    if item 0 issue-types = "valence"  [set val 0]
1034    if number-of-issues > 1 [if item 1 issue-types = "valence"  [set val 1]]
1035
1036    if item 0 issue-types = "second-dimension" [set second-dim 0]
1037    if number-of-issues > 1 [if item 1 issue-types = "second-dimension" [set second-dim 1]]
1038
1039    let salience item lr issues-salience-p
1040    if val != -1 [set salience salience + item val issues-salience-p]
1041    ifelse salience > 0 [
1042      set salience (item lr issues-salience-p) / salience
1043      set lr salience * (item lr issues-position-p)
1044      set xcor ((lr + 1) / 2) * max-xcor?
1045    ][
1046      set xcor 0
1047    ]
1048
1049    ifelse second-dim = -1 [
1050      set ycor precision (max-ycor! / 2) 0
1051    ][
1052        set salience item second-dim issues-salience-p
1053        if val != -1 [set salience salience + item val issues-salience-p]
1054        ifelse salience > 0 [
1055          set salience (item second-dim issues-salience-p) / salience
1056          set second-dim salience * (item second-dim issues-position-p)
1057          set ycor ((second-dim + 1) / 2) * max-ycor?
1058        ][
1059          set ycor 0
1060        ]
1061    ]
1062  end
1063
1064  to rebalance-relevance
1065    let sum-relevance sum issues-salience-p-not-standard
1066    if sum-relevance = 0 [set sum-relevance 1]
1067    let i 0
1068    repeat number-of-issues [
1069      set issues-salience-p replace-item i issues-salience-p ((item i issues-salience-p-not-
      standard) / sum-relevance)
1070      set i i + 1
1071    ]
1072  end
1073
```

```
1074  to-report party-distance [#x #y]
1075    let distance-between 0
1076    let pos-x [issues-position-p] of #x
1077    let pos-y [issues-position-p] of #y
1078    let rel-x [issues-salience-p] of #x
1079    let rel-y [issues-salience-p] of #y
1080    let rel (map + rel-y rel-x)
1081    let sum-rel sum rel
1082    if sum-rel = 0 [set sum-rel 1]
1083    let rel_stand (map ([[x] -> x / sum-rel]) rel)
1084    let dis (map ([[x y] -> (abs(x - y) / 2) ]) pos-y pos-x)
1085    set distance-between sum (map * dis rel)
1086    report distance-between
1087  end
1088
1089  to calculate-coalition-potential
1090    ask parties [
1091      set prefered-coalition (list)
1092      let prefered-coalition-temp give-preferred-coalition minimum-size-coalition? true false
1093      if ((cabinet = 0 or cabinet-cooperation? < 1) and is-list? prefered-coalition-temp) [set
      prefered-coalition prefered-coalition-temp]
1094    ]
1095    set number-of-coalitions number-of-prefered-coalitions
1096    if cabinet-cooperation? >= 1 [set number-of-coalitions number-of-coalitions + 1]
1097    ask parties [
1098      let my_n 0
1099      foreach ([prefered-coalition] of parties) [[x] -> if member? self x [set my_n my_n + 1]]
1100      ifelse number-of-coalitions > 0 [set coalition-potential my_n / number-of-coalitions][set
      coalition-potential 0]
1101    ]
1102    if cabinet-cooperation? >= 1 [
1103      ask parties with [cabinet = 1][
1104        set coalition-potential (votes-seats? * sum [seats-total] of parties with [cabinet = 1]
      + (1 - votes-seats?) * sum [votes-total] of parties with [cabinet = 1])
1105      ]
1106    ]
1107  end
1108
1109  to-report is-connected-weight
1110    let output 0
1111    if coalition-is-connected? sort parties with [cabinet = 1] [set output 1]
1112    report output
1113  end
1114
1115  to-report number-of-prefered-coalitions
1116    let j 0
1117    ask parties [ifelse empty? prefered-coalition [set j j + 0][set j j + 1]]
1118    report j
1119  end
```

voters.nls

```
1120  to voting-proportional
1121    let list-modifier (list)
1122    let i 0
1123    if strategic-voting-chance? > 0 [
1124      repeat count parties[
1125        ask turtle i [
1126          let modifier 0
1127          if votes-total < electoral-threshold? and (max [votes-total] of parties) != 0  [set
      modifier -9999]
1128          set list-modifier lput modifier list-modifier
1129        ]
1130        set i i + 1
1131      ]
1132    ]
1133
1134    ask parties [set votes-total 0]
1135
1136    ask relevant-patches [
1137      if item 2 heuristic-v [set parties-utility (map + parties-utility list-modifier)]
```

```
      if (max parties-utility) >= (item 3 heuristic-v) [
        let min-party turtle (position (max parties-utility) parties-utility)
        set party-elected min-party
        ask min-party [set votes-total votes-total + [votes] of myself]
      ]
    ]

    set i sum [votes-total] of parties
    if i > 0 [
      ask parties [
        set votes-total votes-total / i
        set seats-total 0
      ]
    ]

    let j sum [votes-total] of parties with [votes-total >= electoral-threshold?]
    if j > 0 [
       ask parties with [votes-total >= electoral-threshold?][
         set seats-total votes-total / j
       ]
    ]
  end

  to voting-plurality
    let i 0
    ask parties [
      set votes-control 0
      set votes-total 0
      set seats-total 0
    ]

    let district-relevants-new (list)

    if display? [
      ask relevant-patches [
        if (max parties-utility) >= (item 3 heuristic-v) [
          set party-elected turtle (position (max parties-utility) parties-utility)
        ]
      ]
    ]

    repeat number-of-districts? [
      ask relevant-patches [
        set parties-utility-temp parties-utility
        if item 2 heuristic-v and ticks > 0 [set parties-utility-temp (map + parties-utility-
  temp (item i district-relevants))]
        if (max parties-utility-temp) >= (item 3 heuristic-v) [
          let min-party turtle (position (max parties-utility-temp) parties-utility-temp)
          let k item i district-voters
          ask min-party [
            set votes-control votes-control + k
            set votes-total votes-total + k
          ]
        ]
      ]

      if strategic-voting-chance? > 0 [
        let second-most-votes [votes-total] of parties
        set second-most-votes sort second-most-votes
        set second-most-votes (item (count parties - 2) second-most-votes) * 0.95
        let list-modifier (list)
        let n 0
        repeat count parties[
          ask turtle n [
            let modifier 0
            if votes-total < second-most-votes [set modifier -9999]
            set list-modifier lput modifier list-modifier
          ]
          set n n + 1
        ]
```

```
        set district-relevants-new lput list-modifier district-relevants-new
      ]
      ask one-of parties with [votes-control = max [votes-control] of parties][
        set seats-total seats-total + 1
      ]
      set i i + 1
      ask parties [set votes-control 0]
    ]
    let i-1 sum [votes-total] of parties
    let i-2 sum [seats-total] of parties
    ask parties [
      set seats-total seats-total / i-2
      set votes-total votes-total / i-1
      set votes-control 0
    ]
    set district-relevants district-relevants-new
  end

  to update-only-one-party [#i]
    set parties-utility replace-item #i parties-utility (voting-utility #i)
  end

  to update-voting-utility
    set parties-utility (list)
    let i 0
    repeat count turtles [
      set parties-utility lput (voting-utility i) parties-utility
      set i i + 1
    ]
  end

  to-report voting-utility [#i]
    ;0: Policy-Weight / 1: Non-Policy-Weight / 2: Strategigc Voting? / 3: Non-Voting[.]
    report ((proximity-issue-utility #i) * (item 0 heuristic-v)) + ((item #i non-policy-factor)
  * (item 1 heuristic-v))
  end

  to-report proximity-issue-utility [#j]
    let positions-p [issues-position-p] of turtle #j
    let relevance-p [issues-salience-p] of turtle #j
    let positions-v issues-position-v
    let relevance-v issues-relevance-v
    let i 0
    let utility-list (list)
    repeat length relevance-v [
      let g 0
      ifelse (item i relevance-p) > (item i relevance-v) [set g (item i relevance-v)][set g
  (item i relevance-p)]
      let u abs((item i positions-p) - (item i positions-v))
      set utility-list lput (((2 - u) / 2) * g) utility-list
      set i i + 1
    ]
    report sum utility-list
  end

  to cabinet-evaluation
    let j [who] of parties with [cabinet = 1]
    if cabinet-evaluation? != 0 [
      ask relevant-patches [
        let i 0
        repeat count parties with [cabinet = 1][
          let x item (item i j) non-policy-factor
          set x x + ((item 4 heuristic-v) / max-ticks?)
          if x > 1 [set x 1]
          if x < 0 [set x 0]
          set non-policy-factor replace-item (item i j) non-policy-factor x
          set i i + 1
        ]
      ]
    ]
```

```
1275  end
```

elections.nls

```
1276  to calculate-election-result
1277    ask relevant-patches [update-voting-utility]
1278    ifelse electoral-system? = "Proportional" [voting-proportional][voting-plurality]
1279    ask parties [
1280      set utility calculate-utility
1281      set-position-based-on-issues
1282    ]
1283    calculate-coalition-potential
1284    cabinet-evaluation
1285  end
1286
1287  to show-election-results
1288    if display? [ask relevant-patches [ifelse party-elected = 0 [set pcolor grey][set pcolor
      scale-color ([color] of party-elected) 0.4 -1 1]]]
1289  end
```

coalitions.nls

```
1290  to-report cabinet-opposition-status
1291    let i 0
1292    let output (list)
1293    repeat count parties [
1294      ask turtle i [set output lput cabinet output]
1295      set i i + 1
1296    ]
1297    report output
1298  end
1299
1300  to-report give-preferred-coalition [#minimum-size #cooperation-check #policy-based?]
1301        let prefered-coalition-temp (list)
1302        set prefered-coalition-temp lput self prefered-coalition-temp
1303        ask parties [set control 0]
1304        let coalition-size seats-total
1305        set control 1
1306        let coalition-condition true
1307        let false-coalition false
1308        ifelse coalition-size < #minimum-size [
1309          while [coalition-size < #minimum-size and (any? parties with [control = 0 and seats-
      total > 0])][
1310            ifelse #policy-based? [
1311              ask min-one-of parties with [control = 0 and seats-total > 0][party-distance myself
      self][
1312                set prefered-coalition-temp lput self prefered-coalition-temp
1313                set coalition-size coalition-size + seats-total
1314                set control 1
1315                if cabinet = 1 and cabinet-cooperation? >= 1 and #cooperation-check [set false-
      coalition true]
1316              ]
1317            ][
1318              ask min-one-of parties with [control = 0 and seats-total > 0][abs([xcor] of myself
      - [xcor] of self)][
1319                set prefered-coalition-temp lput self prefered-coalition-temp
1320                set coalition-size coalition-size + seats-total
1321                set control 1
1322                if cabinet = 1 and cabinet-cooperation? >= 1 and #cooperation-check [set false-
      coalition true]
1323              ]
1324            ]
1325          ]
1326          if coalition-size < #minimum-size [set false-coalition true]
1327          if parties with [control = 1] = nobody  [set false-coalition true]
1328          if not coalition-condition or false-coalition or seats-total <= 0 [set prefered-
      coalition-temp (list)]
1329        ]
1330        [
1331          set prefered-coalition-temp lput self prefered-coalition-temp
1332        ]
1333    report prefered-coalition-temp
1334  end
```

```
1335
1336  to-report coalition-traits [#coalition]
1337    let output-list (list)
1338    set output-list lput what-kind-of-cabinet? #coalition output-list
1339    set output-list lput differences-with-cabinet? #coalition output-list
1340    report output-list
1341  end
1342
1343  to-report coalition-is-connected? [#coalition]
1344    let output TRUE
1345    let opposition parties with [not member? self #coalition and seats-total > 0]
1346    if any? opposition and not empty? #coalition [
1347      let min-lr-cabinet min [xcor] of parties with [member? self #coalition]
1348      let max-lr-cabinet max [xcor] of parties with [member? self #coalition]
1349      ask opposition [
1350        if xcor < max-lr-cabinet and xcor > min-lr-cabinet [set output FALSE]
1351      ]
1352    ]
1353    report output
1354  end
1355
1356  to-report coalition-is-connected-given-any-party? [#coalition]
1357    let output TRUE
1358    let opposition parties with [not member? self #coalition]
1359
1360    if any? opposition and not empty? #coalition [
1361
1362      let min-lr-cabinet min [xcor] of parties with [member? self #coalition]
1363      let max-lr-cabinet max [xcor] of parties with [member? self #coalition]
1364
1365      ask opposition [
1366        if xcor < max-lr-cabinet and xcor > min-lr-cabinet [set output FALSE]
1367      ]
1368    ]
1369    report output
1370  end
1371
1372  to-report range_cabinet
1373    report ((max [xcor] of parties with [cabinet = 1] - min [xcor] of parties with [cabinet =
       1]) / max-xcor!)
1374  end
1375
1376  to-report differences-with-cabinet? [#coalition]
1377    let cabinet-parties parties with [member? self #coalition]
1378    let output 999
1379    let differences 0
1380    if count cabinet-parties = 1 [set output 0]
1381    if count cabinet-parties > 1 [
1382      ask cabinet-parties [
1383        ask cabinet-parties [
1384          set differences differences + (party-distance myself self)
1385        ]
1386      ]
1387      set output (differences / count parties) / 2
1388    ]
1389    report output
1390  end
1391
1392  to-report what-kind-of-cabinet? [#coalition]
1393    let output "none"
1394    let opposition parties with [not member? self #coalition and seats-total > 0]
1395    ifelse any? opposition and not empty? #coalition [
1396      let min-lr-cabinet min [xcor] of parties with [member? self #coalition]
1397      let max-lr-cabinet max [xcor] of parties with [member? self #coalition]
1398      let min-lr-opposit min [xcor] of opposition
1399      let max-lr-opposit max [xcor] of opposition
1400      ifelse max-lr-opposit >= max-lr-cabinet and min-lr-opposit <= min-lr-cabinet [set output
       "center"][set output "wing"]
1401    ][ifelse not empty? #coalition [set output "no coalition"][set output "no opposition"]]
1402    report output
```

```
1403  end
1404
1405  to show-a-prefered-coalition-of-party [#i]
1406    let coalition ""
1407    ask turtle #i [set coalition give-preferred-coalition .5 FALSE FALSE]
1408    ask turtle #i [type "seats: " print precision seats-total 3]
1409    ask parties with [not member? self coalition and seats-total > 0][set color blue]
1410    ask parties with [not member? self coalition and seats-total = 0][set color violet]
1411    ask parties with [    member? self coalition and seats-total > 0][set color red]
1412    ask turtle #i [set color brown]
1413    print "-------------------------------------"
1414    print what-kind-of-cabinet? coalition
1415    print coalition-is-connected? coalition
1416    print differences-with-cabinet? coalition
1417    print coalition-traits coalition
1418  end
1419
1420  to-report give-coalition-at-start [#type #minimum-size]
1421    let possible-coalitions (list)
1422    let possible-coalitions-type (list)
1423    let possible-coalitions-diff (list)
1424    let final-cabinet (list)
1425
1426    ask parties with [seats-total > 0][
1427      let coalition give-preferred-coalition #minimum-size FALSE FALSE
1428      set possible-coalitions lput coalition possible-coalitions
1429      set possible-coalitions-type lput what-kind-of-cabinet? coalition possible-coalitions-
      type
1430      set possible-coalitions-diff lput differences-with-cabinet? coalition possible-
      coalitions-diff
1431    ]
1432
1433    let stop-condition not member? "wing" possible-coalitions-type and not member? "center"
      possible-coalitions-type
1434    if stop-condition [
1435        let emergency create-emergency-coalition #type
1436        ifelse is-agentset? emergency [
1437          set final-cabinet emergency
1438        ][
1439          set error-code "no coalition possible"
1440          print "no coalition possible"
1441          set stop-condition true
1442        ]
1443    ]
1444
1445    ifelse member? #type possible-coalitions-type and not stop-condition [
1446      let i 0
1447      let minimal-differences 999
1448
1449      repeat length possible-coalitions-type [
1450        if item i possible-coalitions-type = #type [
1451          if item i possible-coalitions-diff < minimal-differences [
1452            set minimal-differences item i possible-coalitions-diff
1453            set final-cabinet item i possible-coalitions
1454          ]
1455        ]
1456        set i i + 1
1457      ]
1458    ][
1459      if not stop-condition [
1460        ;Hier
1461        let emergency create-emergency-coalition #type
1462        ifelse is-agentset? emergency [
1463          set final-cabinet emergency
1464        ][
1465          set error-code "no coalition possible"
1466          print "no coalition possible"
1467          set stop-condition true
1468        ]
1469      ]
```

```
1470    ]
1471    ifelse not stop-condition [report final-cabinet][report FALSE]
1472  end
1473
1474  to-report create-emergency-coalition [#type]
1475    let party-n count turtles with [seats-total > 0]
1476    let output "none"
1477    if not (#type = "center" and party-n < 3)[
1478      if (#type = "center") [
1479        let min-x min [xcor] of turtles with [seats-total > 0]
1480        let max-x max [xcor] of turtles with [seats-total > 0]
1481        if ((count turtles with [xcor != min-x and xcor != max-x and seats-total > 0]) > 0)[
1482          let relevant-parties turtles with [xcor != min-x and xcor != max-x and seats-total >
      0]
1483          ask parties [set control 0]
1484          ask one-of relevant-parties [set control 1]
1485          let anker one-of relevant-parties with [control = 1]
1486          repeat ((count relevant-parties) - 1) [
1487            if (sum ([seats-total] of relevant-parties with [control = 1])) <= 0.5 [
1488              ask anker [
1489                ask min-one-of relevant-parties with [control = 0][party-distance myself
      self][set control 1]
1490              ]
1491            ]
1492          ]
1493          set output turtles with [control = 1]
1494          set emergency-coalition sum [seats-total] of output
1495        ]
1496      ]
1497
1498      if (#type = "wing") [
1499        ask parties [set control 0]
1500        let test [xcor] of turtles with [seats-total > 0]
1501        if is-list? test and length test > 0 [
1502          let min-x min [xcor] of turtles with [seats-total > 0]
1503          let max-x max [xcor] of turtles with [seats-total > 0]
1504          let anker one-of parties with [xcor = min-x or xcor = max-x]
1505          ask anker [set control 1]
1506          let relevant-parties turtles with [xcor != min-x and xcor != max-x and seats-total >
      0]
1507
1508          repeat count relevant-parties [
1509            if (sum ([seats-total] of relevant-parties with [control = 1])) <= 0.5 [
1510              ask anker [
1511                ask min-one-of relevant-parties with [control = 0][party-distance myself
      self][set control 1]
1512              ]
1513            ]
1514          ]
1515          set output turtles with [control = 1]
1516          set emergency-coalition sum [seats-total] of output
1517        ]
1518      ]
1519    ]
1520    report output
1521  end
1522
1523  to create-coalition-at-start
1524    let searched-cabinet "test"
1525    ifelse center-cabinet? [set searched-cabinet "center"][set searched-cabinet "wing"]
1526    let coalition give-coalition-at-start searched-cabinet 0.5 ;minimum-size-coalition?
1527    ask parties [
1528        set cabinet 0
1529        set shape "triangle"
1530      ]
1531    if is-list? coalition [
1532      ask parties with [member? self coalition][
1533        set cabinet 1
1534        set shape "circle"
1535      ]
```

```
1536    ]
1537    if is-agentset? coalition [
1538      ask coalition [
1539        set cabinet 1
1540        set shape "circle"
1541      ]
1542    ]
1543    if not is-agentset? coalition and not is-list? coalition [
1544      set error-code "no coalition reported"
1545      stop
1546    ]
1547  end
```

output.nls

```
1548  to measure-output
1549    set time-consumption timer
1550  end
1551
1552  to ini-measures
1553    set parties-xcor-positions (list)
1554    set parties-ycor-positions (list)
1555    set parties-seats (list)
1556    set parties-votes (list)
1557    set parties-position (list)
1558    set parties-salience (list)
1559  end
1560
1561  to measure-parties
1562    let i 0
1563    let l1 (list)
1564    let l2 (list)
1565    let l3 (list)
1566    let l4 (list)
1567    let l5 (list)
1568    let l6 (list)
1569    while [i < count parties][
1570      ask turtle i [
1571        set l1 lput xcor l1
1572        set l2 lput ycor l2
1573        set l3 lput seats-total l3
1574        set l4 lput votes-total l4
1575
1576        set l5 lput issues-position-p l5
1577        set l6 lput issues-salience-p l6
1578      ]
1579      set i i + 1
1580    ]
1581    set parties-xcor-positions lput l1 parties-xcor-positions
1582    set parties-ycor-positions lput l2 parties-ycor-positions
1583    set parties-seats lput l3 parties-seats
1584    set parties-votes lput l4 parties-votes
1585    set parties-position lput l5 parties-position
1586    set parties-salience lput l6 parties-salience
1587  end
```