```
;;
============================================================
=================
;; CVC Exploration Simulation    v12
;; Master's Thesis — Keio Business School (March 2026)
;;
============================================================
=================
;; v11 -> v12 changes:
;;    1. D-near & D-far merged into single D-boundary
;;         Near zone: d <= D-boundary,    Far zone: d > D-boundary
;;         No middle zone. Cleaner conceptually.
;;
;;    2. Success learning now distance-dependent
;;         Old: ΔA = (1-A) × ( η_s × succ_bar + η_f × (1-succ_bar) × d_bar)
;;         New: ΔA = (1-A) × d_bar × ( η_s × succ_bar + η_f × (1-succ_bar))
;;         "All learning is proportional to knowledge distance"
;;         Cohen & Levinthal (1990): absorptive capacity grows from the gap
;;         between existing and new knowledge.
;;
;;    3. Parameter rebalancing from theoretical derivation:
;;         lambda-plus    = 0.12   (free parameter α)
;;         lambda-minus = 0.15   (1.25× α, total negativity bias = 2.5 with κ=2)
;;         lambda-cf       = 0.075 (0.625× α, half of visible failure impact)
;;         W                    = 15       (2-3 quarterly reviews, Dushnitsky 2012)
;;
============================================================
=================

breed [capitalists capitalist]
breed [startups startup]

startups-own [
  su-value
  su-distance
  su-discovered?
```

```
  su-translated?
  su-visible-success?
  su-visible-failure?
  su-cost
  su-sr
  su-tick-born
]

patches-own [
  distance-norm
]

globals [
  A_t
  L_t
  O_t
  B_t
  cumulative-CF
  cumulative-SR
  survival-time
  N_t
  pos-signal-total
  neg-signal-total
  eta-s
  eta-f
  lambda-sr
  T-switch
  max-dist
  exited?
  logging?
  prev-mode
]

;;
================================================================
=================
```

```
;; SETUP
;;
============================================================
=================

to setup
  clear-all

  resize-world (- grid-half) grid-half (- grid-half) grid-half
  set-patch-size max list 1 (floor (330 / (2 * grid-half + 1)))

  set max-dist sqrt (max-pxcor ^ 2 + max-pycor ^ 2)

  ask patches [
    let raw-dist sqrt (pxcor ^ 2 + pycor ^ 2)
    set distance-norm raw-dist / max-dist

    ;; Grayscale: near = light, far = dark
    ifelse distance-norm <= D-boundary [
      set pcolor 9 - (distance-norm / D-boundary) * 2
    ] [
      let t (distance-norm - D-boundary) / (1 - D-boundary + 0.001)
      set pcolor 6 - t * 4
    ]

    ;; Zone boundary line
    if distance-norm <= D-boundary [
      if any? neighbors4 with [distance-norm > D-boundary] [
        set pcolor black
      ]
    ]
  ]

  ask patch 0 0 [
    set pcolor white
    set plabel "HQ"
```

```
  set plabel-color black
]

let agent-size ifelse-value (grid-half <= 20) [2] [3]

create-capitalists n-capitalists [
  setxy 0 0
  set color black
  set size agent-size
  set shape "person"
  pen-down
  set pen-size 1
]

set A_t A-init
set L_t L-init
set O_t 0
set B_t 0
set cumulative-CF 0
set cumulative-SR 0
set survival-time max-ticks
set N_t 0
set pos-signal-total 0
set neg-signal-total 0
set exited? false
set logging? false
set prev-mode ""

set eta-s (1 - 2 ^ (-1 / h-s))
set eta-f (1 - 2 ^ (-1 / h-f))
set lambda-sr (ln 2) / d-half
set T-switch round (tau * max-ticks)

let near-count count patches with [distance-norm <= D-boundary]
let far-count count patches with [distance-norm > D-boundary]
print ""
```

```
  print "========== SETUP COMPLETE =========="
  print (word "Grid: [-" grid-half ", " grid-half "]    Total patches: " count patches)
  print (word "D-boundary: " D-boundary)
  print (word "Near zone (d<=" D-boundary "): " near-count " patches")
  print (word "Far zone    (d>" D-boundary "):    " far-count " patches")
  print (word "Startup rate: " startup-rate "/tick    Lifetime: " startup-lifetime " ticks")
  print (word "CVC agents: " n-capitalists "    Max ticks: " max-ticks)
  print (word "T-switch: " T-switch "    eta-s: " precision eta-s 4 "    eta-f: " precision eta-f 4)
  print (word "lambda+: " lambda-plus "    lambda-: " lambda-minus "    lambda-cf: "
lambda-cf "    W: " W)
  print "====================================="

  reset-ticks
end

;;
============================================================
=================
;; STARTUP SPAWNING
;;
============================================================
=================

to spawn-startups
  create-startups startup-rate [
    move-to one-of patches
    let d [distance-norm] of patch-here

    let sigma (sigma-base + sigma-dist * d)
    set su-value exp (random-normal 0 sigma)
    set su-distance d
    set su-discovered? false
    set su-translated? false
    set su-visible-success? false
    set su-visible-failure? false
    set su-cost 0
```

```
    set su-sr 0
    set su-tick-born ticks

    set shape "dot"
    set size 0.8
    set color gray + 2
  ]
end

;;
============================================================
=================
;; GO
;;
============================================================
=================

to go
  if exited? [ stop ]
  if ticks >= max-ticks [
    log-tick
    stop-log
    print-summary
    stop
  ]

  let mode get-current-mode
  if mode != prev-mode and prev-mode != "" [
    flash-phase-change mode
  ]
  set prev-mode mode

  ;; === Step 1: New startups appear ===
  spawn-startups

  ;; === Step 2: CVC agents move and discover ===
```

```
ask capitalists [
  ifelse mode = "near" [
    set color black
  ] [
    set color white
  ]
  move-according-to mode

  let available startups-here with [not su-discovered?]
  if any? available [
    ask one-of available [
      set su-discovered? true
    ]
  ]
]

;; Gather newly discovered (discovered but not yet processed)
let discovered-now startups with [su-discovered? and not su-translated?]
set N_t count discovered-now

;; === Step 3: Translation ===
ask discovered-now [
  let d su-distance
  let V su-value

  let P-succ (pi-param * A_t) / (A_t + d)
  set su-translated? true

  ifelse random-float 1 < P-succ [
    ;; Success
    set su-visible-success? (random-float 1 < q)
    let g exp (- lambda-sr * d)
    let beta-val (1 + b-param * d)
    set su-sr (V * g * beta-val * L_t)

    ifelse V > 5 [
```

```
        set shape "star"
        set color white
        set size 1.5
      ] [
        set shape "circle"
        set color white
        set size 1
      ]
    ] [
      ;; Failure
      set su-visible-failure? (random-float 1 < (kappa * q))
      set su-sr 0

      set shape "x"
      set color gray - 2
      set size 1
    ]

    set su-cost (c0 * d * (max list 0 (1 - gamma-param * L_t)))
]

;; === Aggregate ===
let successful-now discovered-now with [su-sr > 0]
let successes count successful-now
let total-succ-value sum [su-value] of successful-now
let total-cost sum [su-cost] of discovered-now
let total-SR sum [su-sr] of discovered-now
let sum-d sum [su-distance] of discovered-now

;; === SQRT-BASED signals ===
let pos-count count discovered-now with [su-visible-success?]
let neg-count count discovered-now with [su-visible-failure?]
let pos-signal sqrt pos-count
let neg-signal sqrt neg-count

;; === Cashflow ===
```

```
let CF_t (total-succ-value - total-cost)
set cumulative-CF cumulative-CF + CF_t
set cumulative-SR cumulative-SR + total-SR

;; === Signal totals ===
set pos-signal-total pos-signal-total + pos-count
set neg-signal-total neg-signal-total + neg-count

;; === Legitimacy update ===
let cf-penalty ifelse-value (CF_t < 0) [lambda-cf] [0]
set L_t clip01 (rho * L_t
                        + lambda-plus * pos-signal
                        - lambda-minus * neg-signal
                        - cf-penalty)

;; === ACAP update (v12: d_bar applied to ALL learning) ===
if N_t > 0 [
  let succ-bar (successes / N_t)
  let d-bar (sum-d / N_t)
  ;; Learning is proportional to knowledge distance
  ;; Near: small d_bar -> slow learning (familiar territory)
  ;; Far:   large d_bar -> fast learning (novel territory)
  let delta-A ((1 - A_t) * d-bar * (eta-s * succ-bar + eta-f * (1 - succ-bar)))
  set A_t clip01 (A_t + delta-A)
]

;; === Option stock ===
set O_t O_t + successes

;; === Clean up old startups ===
ask startups with [not su-discovered? and (ticks - su-tick-born) >= startup-lifetime] [
  die
]

;; === Exit check ===
ifelse L_t < L-min [
```

```
    set B_t B_t + 1
  ] [
    set B_t 0
  ]

  if B_t >= W [
    set survival-time ticks
    log-tick
    stop-log
    print-summary
    set exited? true
    ask capitalists [ set color gray set size 1 ]
    stop
  ]

  log-tick
  tick
end

;;
================================================================
=================
;; VISUALS
;;
================================================================
=================

to flash-phase-change [new-mode]
  let agent-size ifelse-value (grid-half <= 20) [2] [3]
  ifelse new-mode = "far" [
    print (word ">>> PHASE SWITCH: Far exploration starts (tick " ticks ")")
    ask capitalists [ set size (agent-size + 1) set color white ]
  ] [
    print (word ">>> PHASE SWITCH: Near exploration starts (tick " ticks ")")
    ask capitalists [ set size (agent-size + 1) set color black ]
  ]
```

```
  ask capitalists [ set size agent-size ]
end

;;
=============================================================
=================
;; STRATEGY / MOVEMENT
;;
=============================================================
=================

to-report get-current-mode
  if strategy = "near-only"              [ report "near" ]
  if strategy = "far-only"               [ report "far"    ]
  if strategy = "staged-near-far"    [
    ifelse ticks < T-switch [ report "near" ] [ report "far" ]
  ]
  if strategy = "reverse-far-near" [
    ifelse ticks < T-switch [ report "far" ] [ report "near" ]
  ]
  report "near"
end

to move-according-to [mode]
  ifelse mode = "near" [
    move-near
  ] [
    move-far
  ]
end

to move-near
  ;; Active search in near zone via networks, events, accelerators
  let targets startups with [su-distance <= D-boundary and not su-discovered?]
  ifelse any? targets [
    move-to one-of targets
```

```
  ] [
    ;; No undiscovered startups — random walk within near zone
    let old-x xcor
    let old-y ycor
    rt random 360
    fd 1
    if [distance-norm] of patch-here > D-boundary [
      setxy old-x old-y
      facexy 0 0
      rt (random 61) - 30
      fd 1
      if [distance-norm] of patch-here > D-boundary [
        setxy old-x old-y
      ]
    ]
  ]
end

to move-far
  ;; Active search in far zone via scouting, thematic funds
  let targets startups with [su-distance > D-boundary and not su-discovered?]
  ifelse any? targets [
    move-to one-of targets
  ] [
    let far-patches patches with [distance-norm > D-boundary]
    if any? far-patches [
      move-to one-of far-patches
    ]
  ]
end

;;
=============================================================
=================
;; UTILITIES
;;
```

```
============================================================
==================

to-report clip01 [x]
  report max list 0 (min list 1 x)
end

to-report current-mode-label
  report get-current-mode
end

to-report discoveries-pct
  let total-ever (ticks + 1) * startup-rate
  ifelse total-ever > 0 [
    report (count startups with [su-discovered?] / total-ever) * 100
  ] [
    report 0
  ]
end

to-report active-startups
  report count startups with [not su-discovered?]
end

;;
============================================================
==================
;; CSV DATA LOGGING
;;
============================================================
==================

to start-log
  let filename (word "cvc_" strategy "_g" grid-half "_" date-and-time-safe ".csv")
  file-open filename
  file-print
```

```
"tick,strategy,grid_half,mode,A_t,L_t,O_t,B_t,N_t,cumulative_CF,cumulative_SR,pos_signals,neg_signals,discovered_pct,active_startups,survival_time"
  set logging? true
  print (word "LOG STARTED: " filename)
end

to stop-log
  if logging? [
    file-close
    set logging? false
    print "LOG SAVED."
  ]
end

to log-tick
  if logging? [
    file-print (word
      ticks ","
      strategy ","
      grid-half ","
      get-current-mode ","
      precision A_t 6 ","
      precision L_t 6 ","
      O_t ","
      B_t ","
      N_t ","
      precision cumulative-CF 4 ","
      precision cumulative-SR 4 ","
      pos-signal-total ","
      neg-signal-total ","
      precision discoveries-pct 2 ","
      active-startups ","
      survival-time
    )
  ]
end
```

```
to-report date-and-time-safe
  let raw date-and-time
  let result ""
  let i 0
  while [i < length raw] [
    let ch item i raw
    ifelse (ch = " " or ch = ":" or ch = ".") [
      set result (word result "-")
    ] [
      set result (word result ch)
    ]
    set i i + 1
  ]
  report result
end

;;
============================================================
==================
;; RESULT SUMMARY
;;
============================================================
==================

to print-summary
  print ""
  print "========== SIMULATION RESULT =========="
  print (word "Strategy:         " strategy)
  print (word "Market size:      grid-half=" grid-half " (" count patches " patches)")
  print (word "D-boundary:       " D-boundary)
  print (word "Startup rate:     " startup-rate "/tick, lifetime=" startup-lifetime " ticks")
  print (word "Survival:         " survival-time " / " max-ticks " ticks")
  ifelse survival-time < max-ticks [
    print "Outcome:          *** EXIT (shut down) ***"
  ] [
```

```
    print "Outcome:              SURVIVED"
  ]
  print "--------------------------------------"
  print (word "ACAP (A):            " precision A_t 4)
  print (word "Legitimacy (L): " precision L_t 4)
  print (word "Options (O):      " O_t)
  print "--------------------------------------"
  print (word "Cumul. CF:           " precision cumulative-CF 2)
  print (word "Cumul. SR:           " precision cumulative-SR 2)
  print "--------------------------------------"
  print (word "Pos signals:       " pos-signal-total)
  print (word "Neg signals:       " neg-signal-total)
  print (word "Discovered:        " precision discoveries-pct 1 "%")
  print "======================================="
  print ""
end
```