Agent-Based Opinion Dynamics Simulator — Capabilities (Updated)

Version — updated with repeating events (repeat-event / event-pace).

1. Conceptual Overview

This simulator implements an agent-based model of opinion dynamics on a dynamic network. Agents hold a continuous opinion in [-1, 1], a prevalence (0–99) that encodes how strongly their opinions are represented (interpreted as the number/weight of 'memes'), and an influence score in [0,1]. Opinion transmission occurs along links subject to (i) prevalence gaps, (ii) polarization penalties, (iii) group alignment, and (iv) reward-based learning for successful emitters. Optional 'memes' dynamics allow opinions and prevalence to emerge endogenously from accumulating pro/anti meme stocks. Meta-influencers (e.g., yellow agents or agents with influence=1) can be protected from sign flips via the metablock veto.

2. Architectural Highlights

• Networked interactions: link creation/removal; homophily vs. bridging links.

• Reward mechanism: successful emitters receive a transmission bonus (tx-bonus).

• Memes module (optional): opinion = balance(pro-memes vs. anti-memes); prevalence scales with total memes.

• Meta veto (metablock): prevents sign changes for meta-influencers while still allowing magnitude strengthening.

• External events: global or partial shocks with optional repetition (repeat-event) at a defined pace (event-pace).

• CSV logging per tick and multi-run support.

3. Controls and Parameters (Key Widgets)

3.1 Transmission & Adoption

• prevalence-weight (≥0): amplifies the role of prevalence gaps in adoption probability.

• adoption-floor (0–1): minimum penalty level to avoid extreme polarization lock-in.

• polarization-factor (≥0): scales the penalty from ideological distance (abs difference on |opinion|).

3.2 Group Effects

• group-impact-mode: 'all' or 'k-nearest' to compute alignment.

• group-k (≥1): neighborhood size when using k-nearest.

• group-impact-weight (0–1) and group-impact-alpha (≥0.1): weight and nonlinearity of group alignment.

3.3 Reward Mechanics

• reward-step (>0): additive increment to tx-bonus upon each successful influence.

• reward-cap (≥ reward-step): maximum accumulated tx-bonus.

• reward-scope: 'both', 'left-only', or 'right-only'.

• reward-prev-delta (≥0): optional immediate increase of target prevalence upon success.

• reward-decay (≥0): per-tick decay of tx-bonus towards zero.

3.4 Meta-Influencers & Veto

• metablock (on/off): when on, meta-influencers cannot change the sign of their opinion. If adoption proposes a sign flip, the model preserves the current sign and increases magnitude to the max(|old|, |new|) — 'reinforcement without inversion'.

3.5 Memes Module (Optional)

• use-memes? (on/off): activates meme accumulation dynamics.

• meme-max (>0): soft cap on total meme stock (pro+anti).

• meme-gain (>0): increment added to the meme stock aligned with the emitter's sign.

• meme-anti-leak (≥0): reduces the opposite meme stock upon reinforcement (soft competition).

• meme-decay (≥0): per-tick decay of both meme stocks.

Derived quantities: opinion = (pro − anti)/(pro + anti); prevalence ∝ (pro + anti).

3.6 Events: One-Off, Partial, and Repeating

• event (button): triggers a shock that nudges opinions and optionally prevalence for eligible agents.

• auto_event (switch): if on, fires the event automatically when iter == tick-event.

• tick-event (number): the tick at which the next auto event occurs.

• event-pace (integer ≥1): the step used to schedule the next event when repeat-event is on.

• repeat-event (switch): if on, after firing at tick-event the simulator re-schedules the next event at tick-event + event-pace.

• event-prob-max (0.0–1.0): fraction of agents sampled to actually apply the event at a trigger (partial shocks).

• event_size: shift applied to opinion; prev_change: optional shift applied to prevalence.

• Filters: meme_set plus bounds (low_meme–high_meme, low-prev–high-prev) to target sub-populations.

Recommended Scheduling Logic

When auto_event is ON: if iter == tick-event, run event; if repeat-event is ON, set tick-event = tick-event + max(1, round(event-pace)). When auto_event is OFF: do not modify tick-event automatically; optionally add a button 'schedule-next-event' that sets tick-event = iter + max(1, round(event-pace)).

3.7 Network Formation

• link-formation-threshold (%): homophily radius for link creation.

• bridge-prob (0–1): chance to prioritize creating a cross-sign (bridging) link.

• linksup / linksdown / prob: number and probability knobs for link churn per tick.

• link-removal-threshold (%): ideological distance above which links are eligible for removal.

• show-links? (switch): toggles link visibility in the view.

4. Data & Outputs

• CSV export per tick: key aggregates (medians by sign, link churn, inversion %, etc.).

• 'Values' and 'File' outputs: per-agent streaming suitable for debugging or analysis.

• Multi-run support: try / nb_try with automatic per-run CSV naming (basename-<try>.csv).

5. Example Use Cases

A. Repeating Media Shock (Partial)

Turn auto_event ON, set tick-event=50, repeat-event ON, event-pace=25, event-prob-max=0.15. This applies a 15% partial shock at ticks 50, 75, 100, … enabling stepped changes. Combine with prev_change>0 to simulate increased salience.

B. Meta Reinforcement without Sign Flip

Turn metablock ON. When meta agents receive conflicting influence, they will not cross zero; instead, their magnitude increases to the maximum of pre/post attempts.

C. Meme-Driven Drift

Turn use-memes? ON, set meme-gain>0 and a small meme-decay, and keep reward-step moderate. Opinions and prevalence emerge from accumulated pro/anti memes; use event-prob-max<1 to target subsets.

6. Good Practices

• Normalize event-pace to integers ≥1 to avoid rescheduling stalls.

• Initialize defaults for repeat-event and event-pace when widgets are absent.

• Avoid updating tick-event when auto_event is OFF, to respect manual control.

• Provide a 'schedule-next-event' button for explicit scheduling in manual mode.

• Cap probabilities to [0,1] and constrain opinions to [-1,1].

Appendix — Code Suggestions for Repeating Events

1) Globals & Defaults

- Add tick-event to globals if missing.
- In setup, set defaults:

```
  if not is-boolean? repeat-event [ set repeat-event false ]
  if not is-number? event-pace   [ set event-pace 10 ]
  if not is-number? event-prob-max [ set event-prob-max 1.0 ]
  if event-pace < 1 [ set event-pace 1 ]
```

2) Scheduling in go

```
if auto_event [
 if iter = tick-event [
  event
  if repeat-event [ set tick-event (tick-event + max 1 round event-pace) ]
 ]
]
```

3) Optional manual scheduler button

```
to schedule-next-event
 set tick-event (iter + max 1 round event-pace)
end
```