Université de Montréal

Simulation d'une campagne de vaccination

Carla Danahe Gomez Saucedo (20341379)
CarlaGomez1

Maimouna Modibo Koné (20234378)
mkone

Mia Turmel (20277557)
miaturmel

Travail présenté à Timothée Poisot dans le cadre du cours
BIO 2045 « Simuler le Vivant »

Source du projet

2026-04-08

Table des matières

Introduction

Présentation du modèle

Implémentation

Les packages nécessaires pour simuler le code.

```
using CairoMakie ## Pour créer des graphiques.
CairoMakie.activate!(px_per_unit=6.0) ## Activation de CairoMakie comme moteur d'affichage
graphique.
```

px_per_unit=6.0 augmente la résolution des figures.

```
using StatsBase ## Fournit des fonctions statistiques (tirage aléatoire avec probabilités).
using Random ## Génère des nombres aléatoires.
import UUIDs ## Permet de générer des ID uniques.
```

Code

```
Random.seed!(2045) ## Garantit des résultats reproductibles.
```

```
Random.TaskLocalRNG()
```

Base.@kwdef mutable struct permet de créer une structure mutable qui peut être initialisée avec des valeurs spécifiques. Ici, la taille du Landscape est établie.

```
Base.@kwdef mutable struct Landscape
    xmin::Int64 = -50
    xmax::Int64 = 50
    ymin::Int64 = -50
    ymax::Int64 = 50
end
```

```
Main.var"##277".Landscape
```

Ici, les caractéristiques de départ de l'agent (individus de la population) sont établies.

```
Base.@kwdef mutable struct Agent
    # Position
    x::Int64 = 0
    y::Int64 = 0
    # Horloge interne de l'agent
    clock::Int64 = 21
    # Indique si l'agent est infectieux
    infectious::Bool = false
    # Indique si l'agent est vacciné
    vaccinated::Bool = false
    # Indique quand le vaccin prend effet (pour ce code, il prend effet après 2 jours)
    vax_timer::Int64 = 2
    # Indique si l'agent a été détecté comme infecté
```

```
    is_detected::Bool = false
    # ID unique attribué à chaque agent
    id::UUIDs.UUID = UUIDs.uuid4()
end
```

```
Main.var"##277".Agent
```

Ici, les caractéristiques de l'événement de transmission de l'infection sont établies.

```
Base.@kwdef struct InfectionEvent
    # ID de l'agent ayant transmis l'infection
    from::UUIDs.UUID
    # ID de l'agent ayant reçu l'infection
    to::UUIDs.UUID
    # Moment où la transmission a eu lieu dans la simulation
    time::Int64
    # Position où la transmission a eu lieu
    x::Int64
    y::Int64
end
```

```
Main.var"##277".InfectionEvent
```

Pour ce code, la population est un vecteur contenant plusieurs agents.

```
const Population = Vector{Agent}
```

```
Vector{Agent} (alias for Array{Main.var"##277".Agent, 1})
```

Ceci permet de générer un agent aléatoire dans un paysage (Landscape) donné et de lui assigner une position aléatoire à l'intérieur des limites du Landscape.

```
Random.rand(::Type{Agent}, L::Landscape) = Agent(
    x = rand(L.xmin:L.xmax),
    y = rand(L.ymin:L.ymax)
)
```

Ceci permet de déplacer un agent (A) dans le Landscape (L)

```
"""
    move!(A::Agent, L::Landscape; torus=false)

Déplace un agent aléatoirement d'un pas de taille 1 dans les deux dimension x et y.
Le sens du déplacement est aléatoirement défini entre -1 et 1 sur les deux axes.
Si activé, le keyword de la fonction permet que l'agent réapparait de l'autre coté de la
lattice
s'il en dépasse les bordures.

# Arguments et keyword:
A::Agent : identité de l'agent qui subit le déplacement
L::Landscape : lattice sur laquelle l'agent se déplace
keyword : permet de définir la lattice comme un environnement toroidal.
    - Si true : l'environnement est toroidal et si un agent dépasse les limite du landscape,
```

```
        il revient de l'autre coté.
    - Si false : l'agent est contraint aux limites du landscape

# Retour:
La fonction retourne la position de l'agent modifiée.
"""
function move!(A::Agent, L::Landscape; torus=false)
    # Déplacement limité de l'agent
    A.x += rand(Int64(-1):Int64(1))
    A.y += rand(Int64(-1):Int64(1))
    # Grâce à la fonction torus, si l'agent atteint la bordure du Landscape, il est renvoyé de
l'autre côté
    if torus
        A.y = A.y < L.ymin ? L.ymax : A.y
        A.x = A.x < L.xmin ? L.xmax : A.x
        A.y = A.y > L.ymax ? L.ymin : A.y
        A.x = A.x > L.xmax ? L.xmin : A.x
    # Sinon, l'agent reste à l'intérieur des limites du Landscape
    else
        A.y = clamp(A.y, L.ymin, L.ymax)
        A.x = clamp(A.x, L.xmin, L.xmax)
    end
    return A
end
```

```
Main.var"##277".move!
```

Vérifie l'état de l'agent (infectieux ou en santé)

```
"""
    isinfectious(agent::Agent)

Indique si un agent est infectieux ou non à l'aide de valeurs Booléennes.

# Arguments
agent::Agent : identité de l'agent dont on vérifie l'état d'infection

# Retour
La fonction retourne une valeur Booléennes : true si l'agent est infectieux, false si non.
"""
isinfectious(agent::Agent) = agent.infectious

"""
    ishealthy(agent::Agent)

Indique si un agent est sain ou non à l'aide de valeurs Booléennes, en comparant l'état de
l'agent au contraire de 'agent.infectious'.

# Arguments
agent::Agent : identité de l'agent dont on vérifie l'état d'infection

# Retour
La fonction retourne une valeur Booléennes : true si agent.infectious == false, false si
agent.infectious == true.
"""
ishealthy(agent::Agent) = !agent.infectious
```

```
Main.var"##277".ishealthy
```

Filtre la population pour ne garder que les agents infectieux ou sains

```
"""
    infectious(pop::Population)

Retourne les agents infectieux d'une population en filtrant la population contenant tous les
agents pour ne garder que les
agents infectieux.

# Arguments
pop::Population = la population contenant tous les agents (infectieux et sains)

# Retour
La fonction retourne une collection contenant uniquement les agents infectieux de la
population.
"""
infectious(pop::Population) = filter(isinfectious, pop)

"""
    healthy(pop::Population)

Retourne les agents sains d'une population en filtrant la population contenant tous les agents
pour ne garder que les
agents sains.

# Arguments
pop::Population = la population contenant tous les agents (infectieux et sains)

# Retour
La fonction retourne une collection contenant uniquement les agents sains de la population.
"""
healthy(pop::Population) = filter(ishealthy, pop)
```

```
Main.var"##277".healthy
```

incell(target, pop) analyse les agents de la population et renvoie seulement ceux qui sont aux mêmes coordonnées x et y

```
"""
    incell(target::Agent, pop::Population)

Identifie un agent spécifique, puis vérifie les agents qui occupent la même cellule dans le
Landscape que cet agent d'intéret.
La fonction retourne ensuite toutes les positions sur les axes x et y de ces agents.

# Arguments
target::Agent : un agent spécifique à qui la fonction compare la position avec les autres
agents de la population
pop::Population : la population contenant tous les agents (infectieux et sains)

# Retour
La fonction retourne une collection contenant uniquement les agents dont les coordonnées sur
les axes x et y sont les même
    que l'agent target.
"""
incell(target::Agent, pop::Population) = filter(ag -> (ag.x == target.x && ag.y == target.y),
pop)
```

```
Main.var"##277".incell
```

Crée le Landscape où vivent les agents

```
"""
    simulation(; maxlength::Int64=1000, population_size::Int64=3750, intervention=true)

# Description
1. La fonction établie le Landscape dans lequel la population évolue.
2. Génération de la population contenant tous les agents au départ.
3. Échantillonnage d'un individu spécifique qui sera le premier individu infectieux.
4. Établissement du budget de départ et des couts relier aux interventions.
5. À chaque tick, ou pas de temps, les agents de la population subissent un déplacement,
    des agents infectieux sont choisi, la propagation de la maladie est générée et une
    horloge interne de 21 jours est initié qui décompte le temps avant qu'un agent infectieux
    meurt.
6. Les agents qui meurt à la fin du délai de 21 jours sont supprimés de la population.
7. L'intervention est simulée : une groupe de 20 agents sont sélectionnés aléatoirement,
    ils se font tester et certains se font vacciner.

# Arguments ou keywords
maxlength::Int64=1000 : nombre maximal de ticks ou pas de temps simulés
population_size::Int64=3750 : taille de la population initiale
intervention=true : active ou non l'intervention. Si true, l'intervention est activée et les
RAT et la vaccination sont mis en place.
    Si false, l'intervention n'a pas lieu.

# Retour
La fonction retourne retourne un NamedTuple qui contient:
    - tick = le nombre total de ticks simulés
    - S = historique desvagents sains
    - I = historique des agents infectieux
    - D = historique des morts cumulées
    - V = historique des agents vaccinés
    - budget_hist = l'historique du budget
    - morts_totaux = nombre total de morts
    - budget_restant = montant restant du budget initial
    - survivants = nombre d'agents restant dans la population
    - events = historique des évènements d'infection
"""
function simulation(; maxlength::Int64=1000, population_size::Int64=3750, intervention=true)
    L = Landscape()
    # Un agent est choisi au hasard pour qu'il soit infectieux au départ
    population = [rand(Agent, L) for _ in 1:population_size]
    rand(population).infectious = true

    # Budget et coûts de départ
    budget::Float64 = 21000.0
    cout_vax::Float64 = 17.0
    cout_rat::Float64 = 4.0
    morts_totaux::Int64 = 0
    tick::Int64 = 0

    # Stocke les infos de la simulation à chaque étape (nombres de sains, infectés, morts,
etc. à chaque tick)
    S = Int64[]
    I = Int64[]
    D = Int64[]
    V = Int64[]
    budget_hist = Float64[]
    events = InfectionEvent[]

    # Boucle principale de la simulation (un tick correspond à une étape)
    while !isempty(infectious(population)) && tick < maxlength
        tick += 1

        # Déplacement des agents
        for agent in population
            move!(agent, L; torus=false)
        end
```

```
        # Établir un ordre aléatoire des infectieux
        infectieux_du_jour = Random.shuffle(infectious(population))
        for agent in infectieux_du_jour
            # Déterminer tous les voisins sains sur la même cellule
            neighbors = healthy(incell(agent, population))
            for v in neighbors
                est_protege = v.vaccinated && v.vax_timer <= 0
                if !est_protege && !v.infectious && rand() <= 0.4
                    # Si non protégé, non infectieux et probabilité < 0.4, l'agent devient
infectieux
                    v.infectious = true
                    # Durée de l'infection et enregistrement de l'événement
                    v.clock = 21
                    push!(events, InfectionEvent(
                        from = agent.id,
                        to = v.id,
                        time = tick,
                        x = v.x,
                        y = v.y
                    ))
                end
            end
        end

        # Délai avant l'effet du vaccin
        for agent in population
            if agent.vaccinated && agent.vax_timer > 0
                agent.vax_timer -= 1
            end
        end

        # Compte à rebours jusqu'à la mort
        for agent in population
            est_protege = agent.vaccinated && agent.vax_timer <= 0
            if agent.infectious && !est_protege
                agent.clock -= 1
            end
        end

        # Supprime les agents morts
        morts_step = count(a -> a.clock <= 0, population)
        morts_totaux += morts_step
        population = filter(a -> a.clock > 0, population)

        # Intervention (vaccination et détection)
        if intervention && morts_totaux > 0 && budget > 0 && !isempty(population)
            n = min(Int64(20), Int64(length(population)))
            cibles = StatsBase.sample(population, n; replace=false)

            # Détecte les infectieux avec 95 % de chance (test RAT)
            for a in cibles
                if budget >= cout_rat
                    budget -= cout_rat
                    a.is_detected = a.infectious && rand() <= 0.95
                end
            end

            # Vaccination des agents sains restants (délai de 2 jours avant que le vaccin
devienne actif)
            a_vacciner = filter(a -> !a.infectious && !a.vaccinated, cibles)
            for a in a_vacciner
                if budget >= cout_vax
                    budget -= cout_vax
                    a.vaccinated = true
                    a.vax_timer = 2
```

```
                end
            end
        end

        # Enregistre les données à chaque tick
        push!(S, Int64(length(healthy(population))))
        push!(I, Int64(length(infectious(population))))
        push!(D, morts_totaux)
        push!(V, Int64(count(a -> a.vaccinated, population)))
        push!(budget_hist, budget)
    end

    # Retourne les résultats de la simulation
    return (
        tick = tick,
        S = S,
        I = I,
        D = D,
        V = V,
        budget_hist = budget_hist,
        morts_totaux = morts_totaux,
        budget_restant = budget,
        survivants = Int64(length(population)),
        events = events
    )
end
```

```
Main.var"##277".simulation
```

Fonction pour répéter la simulation plusieurs fois et extraire les données pour comparaison

```
"""
    replicate_simulations(nrep::Int64; intervention=true)

La fonction effectue plusieurs répétition de la simulation d'infection et collecte des
statistiques pour l'analyse.

# Arguments
nrep::Int6 : nombre de répétitions de la simulation à effectuer
intervention=true : active ou non l'intervention. Si true, l'intervention est activée et les
RAT et la vaccination sont mis en place.
    Si false, l'intervention n'a pas lieu.

# Retour
La fonction retourne retourne un NamedTuple qui contient:
    - results : résultats complets de chaque simulation
    - morts : nombre total de morts par simulation
    - survivants : nombre de survivants par simulation
    - budgets : budgets dépensés par simulation
    - durations : durées de chaque simulation (en nombre de tick)
    - mean_morts : moyenne du nombre de morts
    - std_morts : écart-type du nombre de morts
    - mean_survivants : moyenne du nombre de survivants
    - std_survivants : écart-type des survivants
    - mean_budget : moyenne du budget dépensé
    - std_budget : écart-type du budget dépensé
    - mean_duration : durée moyenne des simulations
    - std_duration : écart-type de la durée des simulations
"""
function replicate_simulations(nrep::Int64; intervention=true)
    results = [simulation(intervention=intervention) for _ in 1:nrep]
    morts = [r.morts_totaux for r in results]
    survivants = [r.survivants for r in results]
```

```
    budgets = [21000.0 - r.budget_restant for r in results]
    durations = [r.tick for r in results]
    # Moyennes et écarts-types pour comparaison
    return (
        results = results,
        morts = morts,
        survivants = survivants,
        budgets = budgets,
        durations = durations,
        mean_morts = mean(morts),
        std_morts = std(morts),
        mean_survivants = mean(survivants),
        std_survivants = std(survivants),
        mean_budget = mean(budgets),
        std_budget = std(budgets),
        mean_duration = mean(durations),
        std_duration = std(durations)
    )
end
```

```
Main.var"##277".replicate_simulations
```

Simulations avec et sans intervention et collecte des événements

```
resultats_sans = simulation(intervention=false);
resultats_avec = simulation(intervention=true);
events = resultats_avec.events;

rep_sans = replicate_simulations(Int64(30), intervention=false);
rep_avec = replicate_simulations(Int64(30), intervention=true);
```

Interprétation des résultats comparatifs et statistiques

```
println("=== COMPARAISON SIMPLE ===")
println("Sans intervention - morts: ", resultats_sans.morts_totaux)
println("Avec intervention - morts: ", resultats_avec.morts_totaux)
println("Réduction de mortalité: ", resultats_sans.morts_totaux - resultats_avec.morts_totaux)
println("Coût de la campagne: ", round(21000.0 - resultats_avec.budget_restant, digits=2))

println()
println("=== RÉPLICATIONS (30) ===")
println("Sans intervention - morts moyens: ", round(rep_sans.mean_morts, digits=2), " ± ",
round(rep_sans.std_morts, digits=2))
println("Avec intervention - morts moyens: ", round(rep_avec.mean_morts, digits=2), " ± ",
round(rep_avec.std_morts, digits=2))
println("Sans intervention - survivants moyens: ", round(rep_sans.mean_survivants, digits=2),
" ± ", round(rep_sans.std_survivants, digits=2))
println("Avec intervention - survivants moyens: ", round(rep_avec.mean_survivants, digits=2),
" ± ", round(rep_avec.std_survivants, digits=2))
println("Budget moyen utilisé: ", round(rep_avec.mean_budget, digits=2), " ± ",
round(rep_avec.std_budget, digits=2))
println("Durée moyenne avec intervention: ", round(rep_avec.mean_duration, digits=2), " ± ",
round(rep_avec.std_duration, digits=2))
```

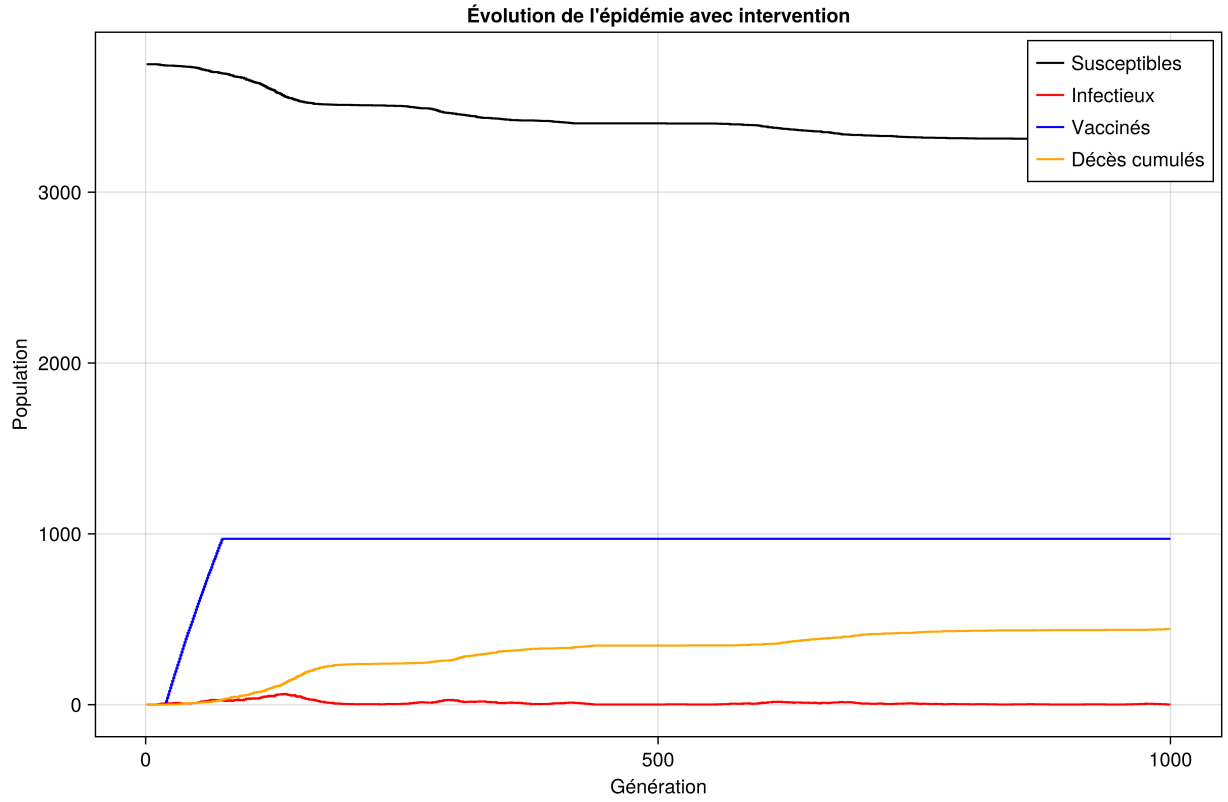
```
=== COMPARAISON SIMPLE ===
Sans intervention - morts: 2808
Avec intervention - morts: 443
Réduction de mortalité: 2365
Coût de la campagne: 20999.0
```

```
=== RÉPLICATIONS (30) ===
Sans intervention - morts moyens: 2485.57 ± 996.02
Avec intervention - morts moyens: 627.4 ± 421.22
Sans intervention - survivants moyens: 1264.43 ± 996.02
Avec intervention - survivants moyens: 3122.6 ± 421.22
Budget moyen utilisé: 19347.63 ± 5366.95
Durée moyenne avec intervention: 748.8 ± 379.38
```

Graphiques pour visualiser les résultats

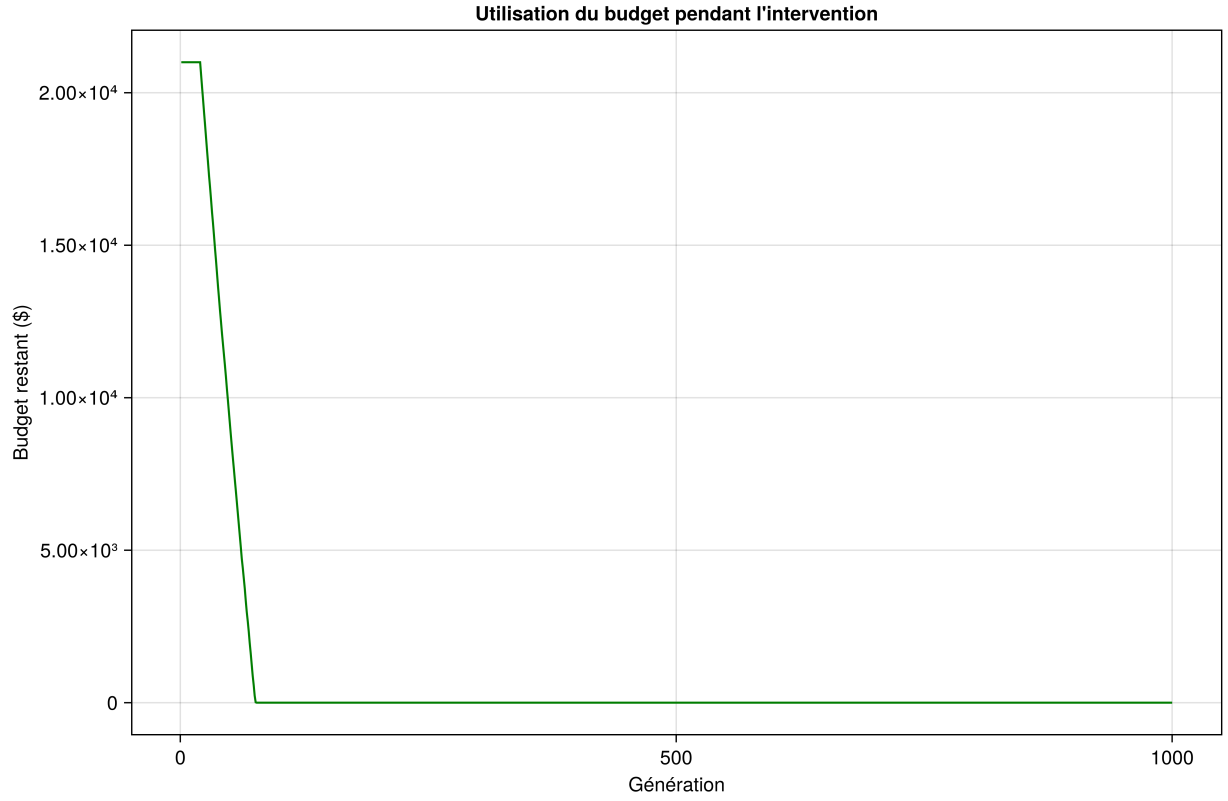
Évolution de la population

```
f1 = Figure(size=(900, 600))
ax1 = Axis(f1[1, 1],
    xlabel="Génération",
    ylabel="Population",
    title="Évolution de l'épidémie avec intervention"
)
stairs!(ax1, 1:length(resultats_avec.S), resultats_avec.S, label="Susceptibles", color=:black)
stairs!(ax1, 1:length(resultats_avec.I), resultats_avec.I, label="Infectieux", color=:red)
stairs!(ax1, 1:length(resultats_avec.V), resultats_avec.V, label="Vaccinés", color=:blue)
stairs!(ax1, 1:length(resultats_avec.D), resultats_avec.D, label="Décès cumulés",
color=:orange)
axislegend(ax1)
current_figure()
```



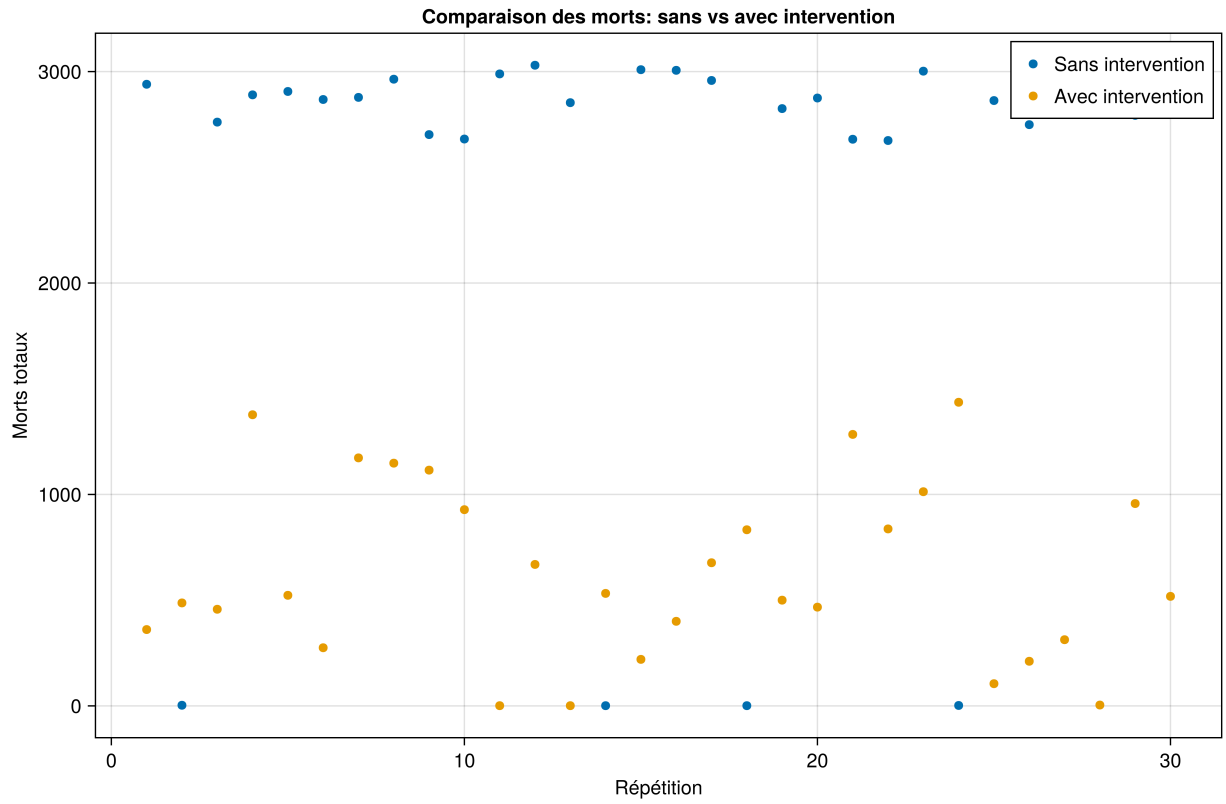
Budget restant au fil du temps

```
f2 = Figure(size=(900, 600))
ax2 = Axis(f2[1, 1],
    xlabel="Génération",
    ylabel="Budget restant (\$)",
    title="Utilisation du budget pendant l'intervention"
)
lines!(ax2, 1:length(resultats_avec.budget_hist), resultats_avec.budget_hist, color=:green)
current_figure()
```



Comparaison des morts sur toutes les réplications

```
f3 = Figure(size=(900, 600))
ax3 = Axis(f3[1, 1],
    xlabel="Répétition",
    ylabel="Morts totaux",
    title="Comparaison des morts: sans vs avec intervention"
)
scatter!(ax3, 1:length(rep_sans.morts), rep_sans.morts, label="Sans intervention")
scatter!(ax3, 1:length(rep_avec.morts), rep_avec.morts, label="Avec intervention")
axislegend(ax3)
current_figure()
```



Graphiques basés sur les événements d'infection

```
if !isempty(events)
    infxn_by_uuid = countmap(getfield.(events, :from))
    nb_inxfn = countmap(collect(values(infxn_by_uuid)))

    # Positions et temps des infections
    t = getfield.(events, :time)
    xs = getfield.(events, :x)
    ys = getfield.(events, :y)
end
```

```
443-element Vector{Int64}:
   9
   8
   9
   8
   6
  10
   7
  12
  12
   9
   4
   5
  10
  14
  13
  13
  13
  11
  13
```

```
12
12
13
11
13
17
13
17
 9
18
13
10
10
16
21
20
21
16
 7
22
17
20
18
13
17
15
14
26
16
 7
27
22
10
23
 5
 7
23
22
24
 6
19
29
20
 4
24
28
 6
17
11
28
28
 6
24
30
14
28
10
 7
 7
11
23
12
23
11
28
11
```

```
24
23
10
28
28
12
30
29
10
10
14
11
 7
 9
 8
11
10
30
13
10
11
25
16
 9
 5
26
26
33
34
12
28
10
24
28
 8
13
24
33
 4
 9
14
 3
27
 2
 4
35
12
26
15
 3
12
32
24
34
28
25
 1
 4
 1
34
 6
33
 3
15
35
31
```

```
 6
37
31
33
25
 5
18
 7
 7
 7
30
11
11
33
26
34
 8
34
34
30
26
 8
26
 6
35
 7
 3
13
24
12
 7
11
 8
29
17
14
33
 4
28
26
16
23
25
29
30
31
12
 6
39
 9
34
 3
20
 3
11
 8
37
35
 4
23
 8
16
 4
34
 8
 7
```

```
 19
  6
 38
  4
 22
 22
 34
 39
 11
 11
  7
  3
 23
  6
  1
  6
 19
  3
  8
  4
 20
 -2
 -2
 -2
 -3
  3
  0
 -1
  9
  9
  0
-15
-14
-15
-17
 -3
-14
 -3
-14
-17
 -3
-16
-17
-19
 -6
-15
-19
-20
-15
-20
-15
-15
-14
-21
-14
-17
-15
-22
-17
-17
-14
-13
-12
-20
-11
-13
```

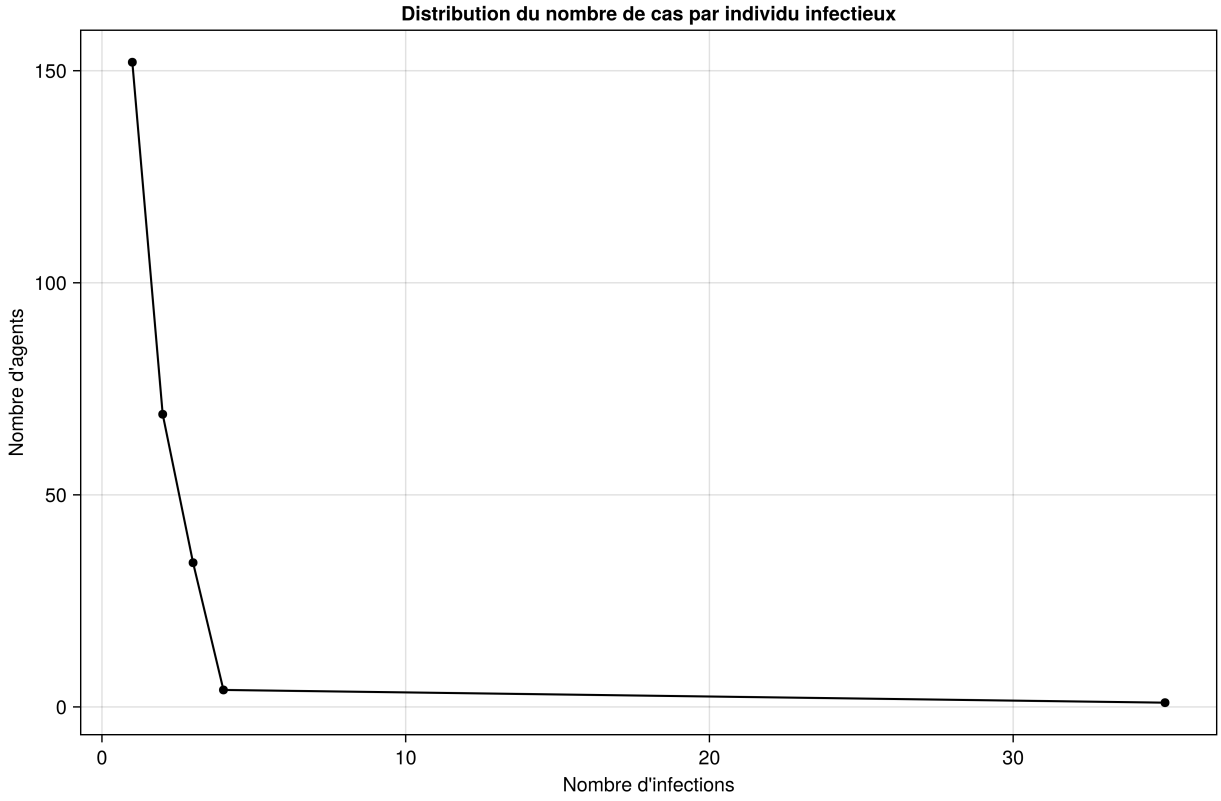
```
-17
-17
-24
-24
-19
 -6
-15
-23
-11
-20
-15
-10
-16
-13
-12
-13
-14
 -5
-22
-16
-15
 -5
 -5
-12
-12
-17
 -5
-15
-16
-15
-19
-18
-17
-14
-14
-15
-16
-15
-14
-17
-17
-19
-18
-18
-17
-20
-22
-20
-23
-19
-25
-26
-25
-23
-23
-23
-22
-27
-24
-24
-25
-23
-22
-19
-19
-20
```

```
-18
-23
-18
-25
-23
-25
-24
-26
-21
-24
-27
-24
-26
-27
-23
-22
-24
-19
-26
-23
-18
-31
-26
-16
-32
-16
-18
-33
-15
-32
-34
-18
-32
-35
-36
-34
-10
-36
-34
-34
-31
-34
-39
-31
-36
-42
-32
-35
-34
-41
-11
-31
-31
-36
-30
-36
-37
-29
-36
-27
-47
-35
-36
-36
-32
-32
```

```
 -9
 -9
-10
-36
 -6
 -7
 -4
 -8
 -4
 -3
 -6
 -9
 -7
-14
-10
-14
-10
-10
-11
-11
-10
 -9
 -9
-19
-18
-14
-17
-14
```

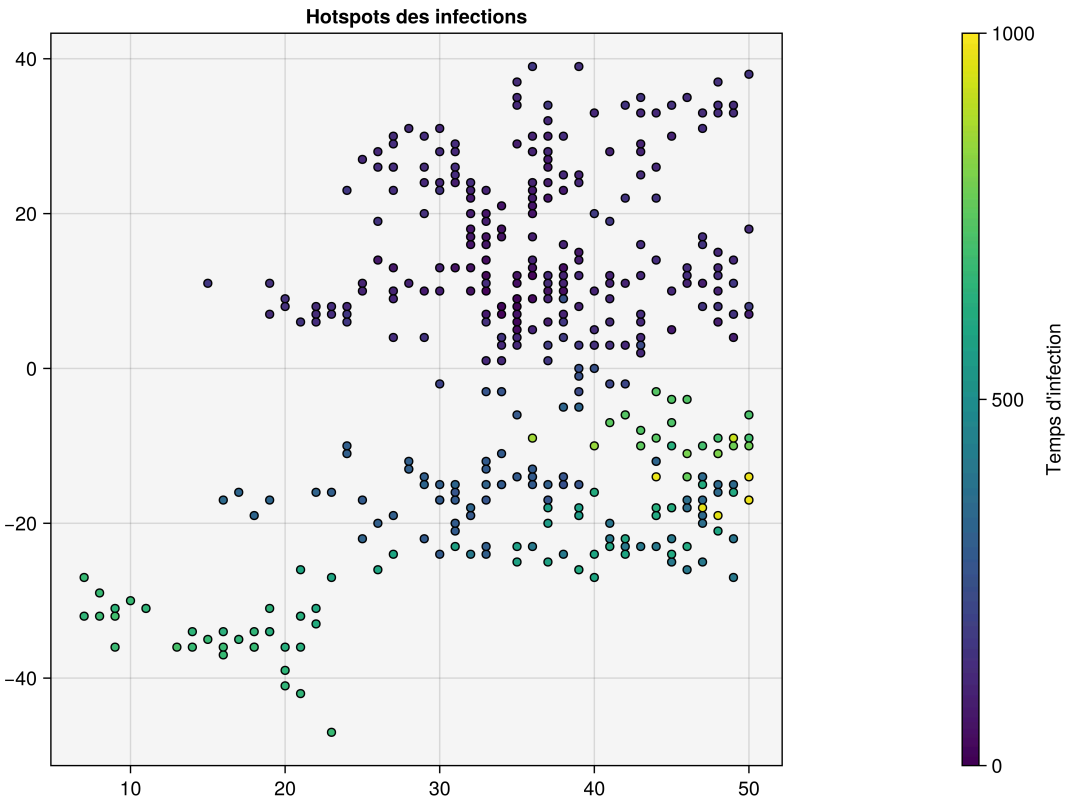
Nombre de cas par individu infectieux

```
f4 = Figure(size=(900, 600))
ax4 = Axis(f4[1, 1],
    xlabel="Nombre d'infections",
    ylabel="Nombre d'agents",
    title="Distribution du nombre de cas par individu infectieux"
    )
xvals = sort(collect(keys(nb_inxfn)))
yvals = [nb_inxfn[x] for x in xvals]
scatterlines!(ax4, xvals, yvals, color=:black)
current_figure()
```



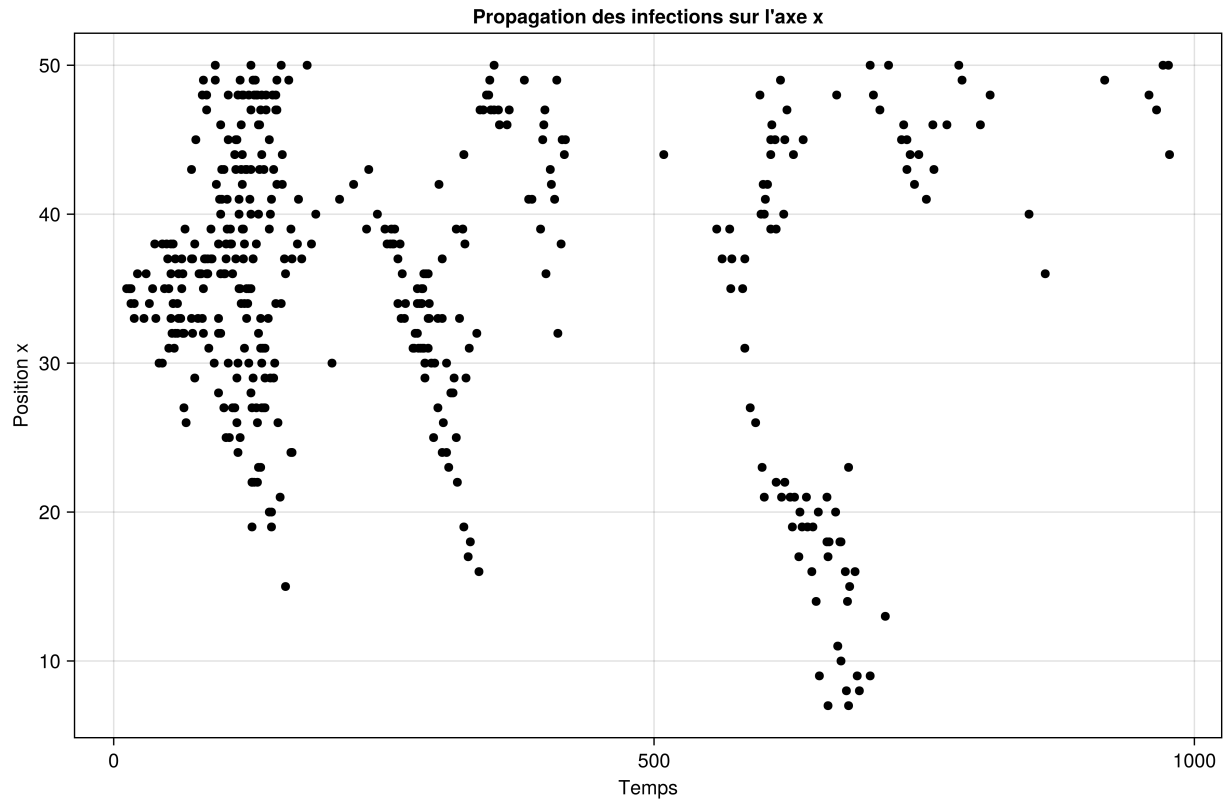
Hotspots des infections

```
f5 = Figure(size=(900, 600))
ax5 = Axis(f5[1, 1],
     aspect=1,
    backgroundcolor=:grey97,
    title="Hotspots des infections"
    )
hm = scatter!(
    ax5,
    xs,
    ys,
    color=t,
    colormap=:viridis,
    strokecolor=:black,
    strokewidth=1,
    colorrange=(0, max(1, resultats_avec.tick)),
    markersize=8
    )
Colorbar(f5[1, 2], hm, label="Temps d'infection")
current_figure()
```



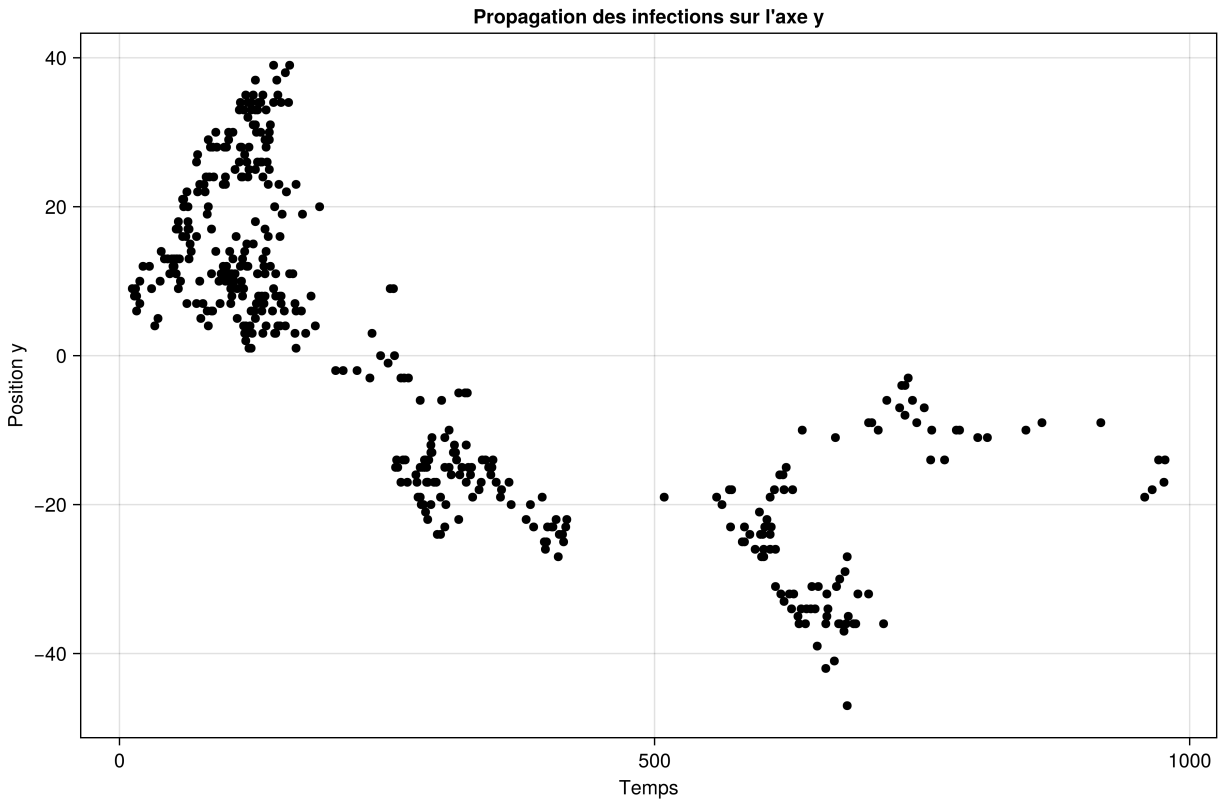
Propagation des infections sur l'axe x

```
f6 = Figure(size=(900, 600))
ax6 = Axis(f6[1, 1],
    xlabel="Temps",
    ylabel="Position x",
    title="Propagation des infections sur l'axe x"
    )
scatter!(ax6, t, xs, color=:black)
current_figure()
```



Propagation des infections sur l'axe y

```
f7 = Figure(size=(900, 600))
ax7 = Axis(f7[1, 1],
    xlabel="Temps",
    ylabel="Position y",
    title="Propagation des infections sur l'axe y"
    )
scatter!(ax7, t, ys, color=:black)
current_figure()
```



Résultats

=== COMPARAISON SIMPLE === Sans intervention - morts: 2808 Avec intervention - morts: 443 Réduction de mortalité: 2365 Coût de la campagne: 20999.0

=== RÉPLICATIONS (30) === Sans intervention - morts moyens: 2485.57 ± 996.02 Avec intervention - morts moyens: 627.4 ± 421.22 Sans intervention - survivants moyens: 1264.43 ± 996.02 Avec intervention - survivants moyens: 3122.6 ± 421.22 Budget moyen utilisé: 19347.63 ± 5366.95 Durée moyenne avec intervention: 748.8 ± 379.38

Discussion

1. Types & Structs Changement : clock augmenté de 20 ⟶ 21 Ajout : vaccinated, vax_timer, is_detected Changement : Lattice ±50 au lieu de ±25
2. Population Generation Changement : génération avec comprehension
3. Movement Changement : torus=false par défaut, clamp
4. Infection Ajout : Protection vaccin Changement : reset clock à 21
5. Survie / Mortalité Changement : décrément seulement infectieux non protégés Chnagement : suivi morts_totaux
6. Intervention Nouveau : détection RAT + vaccination + budget
7. Statistiques Ajout : suivi V_count et budget_count
8. Réplications Nouveau : réplications + moyennes/écarts-types

9. Plotting Ajout : plots V/D, budget, hotspots, propagation x/y

On peut aussi citer des références dans le document “references.bib”, qui doit être au format BibTeX. Les références peuvent être citées dans le texte avec “@” suivi de la clé de citation. Par exemple: ermentrout1993cellular – la bibliographie sera ajoutée automatiquement à la fin du document.

Le format de la bibliographie est American Physics Society, et les références seront correctement présentées dans ce format. Vous ne devez/pouvez pas éditer la bibliographie à la main.