# "Algorithmique et programmation"
# TP Programmation orientée-objet et modules
# (compétences 4, 7 et 8)

**Remarque préliminaire** : Vous devez utiliser git tout au long de ce TP. Il est conseillé de faire un commit à chaque fois que vous avez écrit *et testé* une nouvelle fonction ou méthode, et/ou un morceau de programme principal.

Le but de cet exercice est de faire une modélisation spatiale, stochastique et multi-agents d'une épidémie d'infection virale. En moyenne, les simulations devraient suivre la même dynamique que celle du modèle mathématique classique "SIR" (Susceptible - Infectious - Recovered).

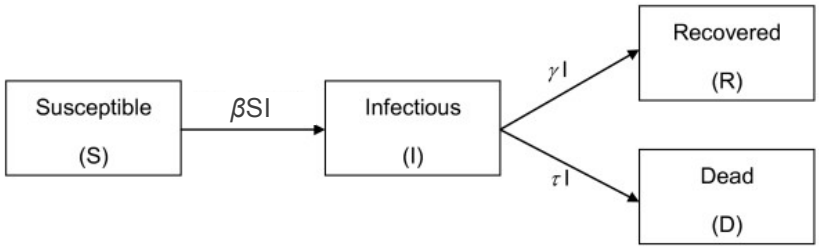


Figure 1 : Le modèle SIR et ses quatre compartiments. *S*, *I*, *R* et *D* représentent respectivement les nombres d'individus suceptibles, infectieux, guéris et décédés. Le modèle suppose que les individus guéris sont alors immunisés et ne peuvent pas être infectés à nouveau. Figure adaptée de Lin et al. (2010). *BMC Infectious Diseases* **10**: 32.

Ici, nous allons considérer un environnement fini et discret : une grille de taille w × h. Les individus se déplaceront sur cette grille selon un mouvement aléatoire : à chaque pas de temps, un individu se déplacera sur l'une des quatre cases voisines, choisie au hasard. On imposera sur la grille des conditions toriques : ce qui sort d'un côté rentre de l'autre. Chaque agent pourra être dans l'un des 4 états suivants : *susceptible*, *infectious*, *recovered* ou *dead*. Il pourra y avoir plusieurs argents sur la même case de la grille.

Un agent susceptible se trouvant sur la même case qu'un ou plusieurs agent(s) infectieux aura une probabilité $p_i$ d'être infecté à son tour. A chaque pas de temps, un agent infectieux aura une probabilité $p_r$ de devenir *recovered* (et ne plus être infectieux), et une probabilité $p_d$ de succomber à l'infection.

1. Créez un fichier `agent.py` qui contiendra le code d'une classe Agent. Chaque agent aura les attributs suivants : x, y, state, $p_i$, $p_r$, $p_d$. Ecrivez le code de la méthode __init__ de cette classe, qui prendra en arguments les valeurs de tous les attributs.

2. Testez votre code en ajoutant dans `agent.py` un 'programme principal', sous la classe Agent. Placez ce programme dans un `if __name__ == "__main__"`.

3. Ajoutez à la classe Agent une méthode move, qui prendra les dimensions de la grille en paramètres. Ajoutez aussi les instructions de test de cette méthode et vérifiez qu'elle fonctionne bien.

4. Ajoutez à la classe Agent une méthode recover_or_die. Ajoutez aussi les instructions de test de cette méthode et vérifiez qu'elle fonctionne bien.

5. Créez un fichier world.py qui contiendra le code d'une classe World. Cette classe aura les attributs suivants : w, h, pop (liste d'agents). Ecrivez le code de la méthode __init__ de cette classe, qui prend en paramètres les dimensions de la grille, le nombre désiré d'agents, la proportion d'agents initialement infectés et les trois probabilités de changements d'état. La méthode __init__ doit créer les agents en les plaçant aléatoirement sur la grille.

6. Testez votre code en ajoutant dans world.py un 'programme principal', sous la classe World. Placez ce programme dans un if __name__ == "__main__".

7. Ajoutez à la classe World une méthode move, qui effectue le mouvement pour tous les agents. Ajoutez les instructions de test de cette méthode et vérifiez qu'elle fonctionne bien.

8. Ajoutez à la classe World une méthode infect, qui reçoit une liste d'agents se trouvant sur une même case. Elle vérifie s'il y en a au moins un d'infecté, et si tel est le cas, résout l'infection (i.e infecte les susceptibles si le tirage est défavorable). Ajoutez les instructions de test de cette méthode et vérifiez qu'elle fonctionne bien.

9. Ajoutez à la classe World une méthode contact, qui trouve sur la grille les agents au même endroit. Pour cela, une liste (de taille w × h) de listes vides est créée, et chaque agent de la liste est rajouté dans la sous-liste de l'endroit où il se trouve. La fonction contact renvoie cette grille. Ajoutez les instructions de test de cette méthode et vérifiez qu'elle fonctionne bien.

10. Ajoutez à la classe World une méthode recover_or_die, qui lance la méthode du même nom de chaque agent. Ajoutez les instructions de test de cette méthode et vérifiez qu'elle fonctionne bien.

11. Ajoutez à la classe World une méthode run, qui lance le mouvement des agents, puis récupère la grille de contact. Chaque sous-liste qui possède plus de deux agents est envoyée à la méthode infect. On appelle ensuite recover_or_die, ce qui termine le traitement. Ajoutez les instructions de test de cette méthode et vérifiez qu'elle fonctionne bien.

12. Ajoutez à la classe World une méthode stats, qui renvoie le nombre d'agents sains, infectés, résistants et morts (sous forme de tuple).

13. Testez la dynamique du système avec w = h = 100 et 1000 agents dont 20% sont initialement infectieux. On prendra $p_i = 1.0$, $p_r = 0.01$ et $p_d = 0.001$, ce qui signifie qu'il faut en moyenne 100 pas de temps en moyenne pour devenir résistant et 1000 pour en mourir. Par contre, le moindre contact entraîne l'infection. On fera la simulation jusqu'à ce qu'il n'y ait plus d'agents infectieux ou que 10000 pas de temps aient été calculés. Utilisez gnuplot pour tracer l'évolution de l'infection au cours du temps. A-t-on une épidémie ?

14. Comparer avec les mêmes paramètres mais pr = 0.001. Que se passe-t-il ?

15. Faire des essais avec ces deux mêmes jeux de paramètres, mais en prenant une infection initiale de 10%, 5% et 1%. Cela change-t-il quelque chose ?

16. On suppose maintenant que l'infection est très mortelle : $p_m = 0.1$. Que se passe-t-il ? Est-ce évident ?

17. (Bonus) Vous devez en principe retrouver sensiblement la même dynamique qu'avec les équations mathématiques. Mais l'intérêt de votre modèle multi-agents est qu'on peut facilement le rendre plus complexe pour s'approcher de situations plus réalistes. Réalisez au choix l'une des modifications suivantes :

    - Rendre les agents hétérogènes, avec par exemple deux sous-groupes d'agents dont les probabilités de guérison sont différentes.
    - Ajouter des obstacles dans l'environnement, par exemple un mur vertical séparant la grille en deux, avec seulement un trou au milieu. Il faudra alors ajouter un attribut dans la classe World (liste des positions des obstacles), s'assurer que les agents ne soient pas créés sur les obstacles, et empêcher le mouvement d'un agent qui voudrait se rendre sur un obstacle.