IA04 – Systèmes Multi-Agents

Simulation multi-agent (version 0.1)

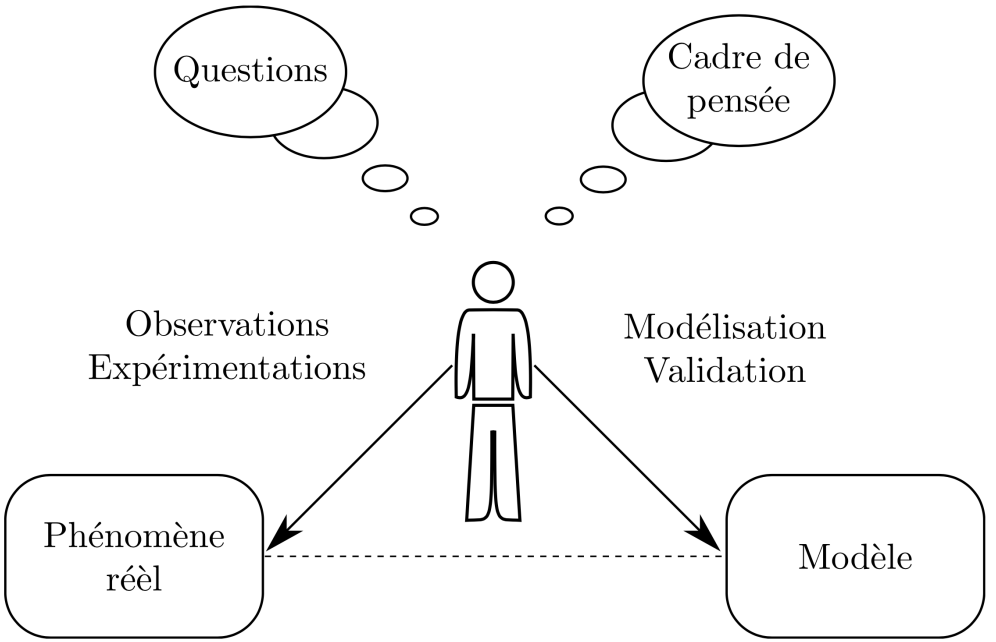
Sylvain Lagrue

sylvain.lagrue@hds.utc.fr

I. Introduction

La simulation informatique

Idée : étudier un *phénomène* réel à l'aide d'un *modèle* (dépendant d'une *théorie*) en utilisant un programme informatique (le *simulateur*). Une interface graphique peut aider à mieux comprendre le phénomène en affichant et en interagissant avec le modèle (*IHM*).



Crédits :

Thèse de Julien Siebert, Approche multi-agent pour la multi-modélisation et le couplage de simulations

I. Introduction

Pourquoi simuler ?

- Pour prédire (comment va évoluer le système), décrire (comment fonctionne le système), prescrire (quel comportement avoir dans quelle situation)

Les problèmes sous-jacents

- Prévoir l'état final d'un système connaissant son état initial (problème direct)
- Déterminer les paramètres d'un système connaissant un ou plusieurs couples (état initial - état final) (problème inverse)
- Pédagogie (simulation d'entraînement)
- Jeux

Les trois (principaux) types de simulation

- La *simulation continue* : le système est représenté sous forme d'équations différentielles (non soluble analytiquement)
- La *simulation discrète* : le temps est discrétisé et le système est soumis à une succession d'événements
- La *simulation par agents* : la simulation est segmentée en différentes

I. Introduction

Pourquoi la simulation multi-agent ?

- Permet de mieux représenter des comportements autonomes et "rationnels"
- Résolution de problèmes complexes
- Test de mécanismes qui convergent par design (ex. mécanismes d'enchères)

Pourquoi la simulation multi-agent ?

- Parce que c'est votre projet !!!!

Exemples de projets

Organisation de structures

- Simulation d'un fast-food
- Simulation d'un hôtel (clients, employés)
- Salle d'urgence à l'hôpital
- Test embarquement d'avion
- Parc d'attractions

Sciences humaines, macro-économie

- Simulation d'une salle de marché (avec API des cours en direct)
- Simulation d'un modèle économique de plateformes de streaming
- Impact des nouvelles technologies sur le marché de l'emploi

Exemples de projets

Simulation de fast-food

Exemples de projets

Marchés financiers

Exemples de projets

Salle d'urgence à l'hôpital

Exemples de projets

Panique...

- Incendie à BF
- Attaque de zombies
- Évacuation d'un avion
- Simulation de manifestations
- Panique dans une salle de cinéma

Simulations d'écosystèmes

- Meute de loups
- Écureuils vs extension de ville
- Pêche et poissons
- Les Schtroumpfs
- Bernard l’hermite

Exemples de projets

Véhicules

- Drones dans des grottes
- Embouteillages et style de conduite
- Simulation réseau RATP

Santé

- Propagation de maladies et vaccins
- Microrobots dans le corps humain

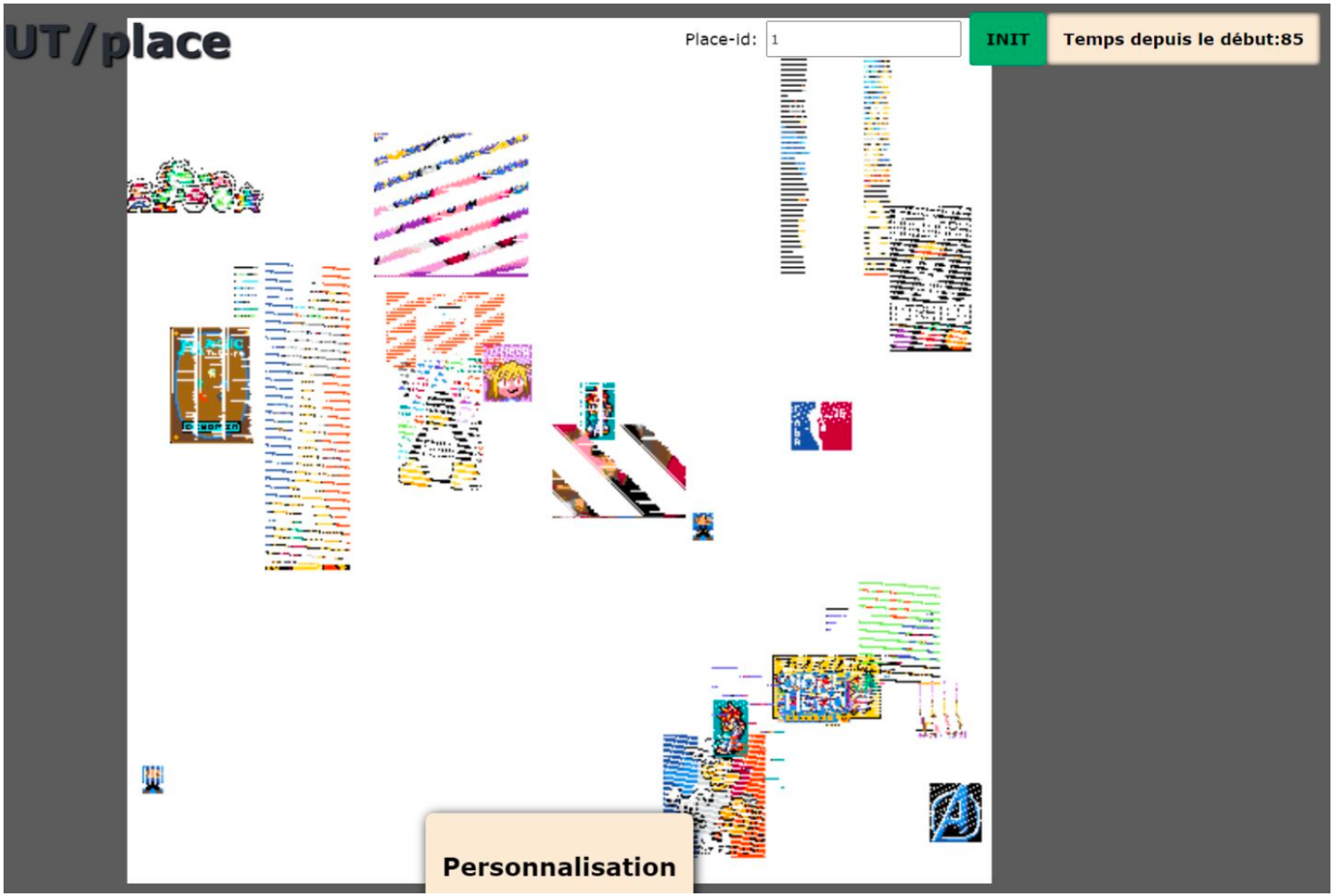
Exemples de projets

Sports/Jeux

- Pixel War
- Conspiration
- Who is Undercover
- Tower defense
- Texas Hold’em
- Battle royal/Hunger Games
- Secret Hitler
- Simulateur de decks de Pokémons
- Kah Lanto
- Football manager

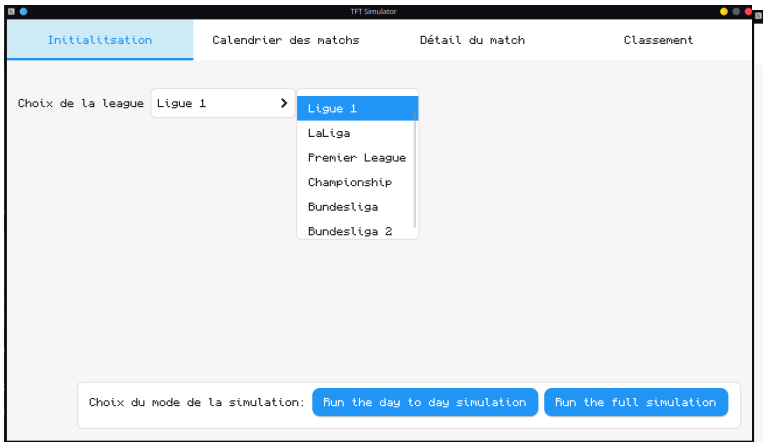
Exemples de projets

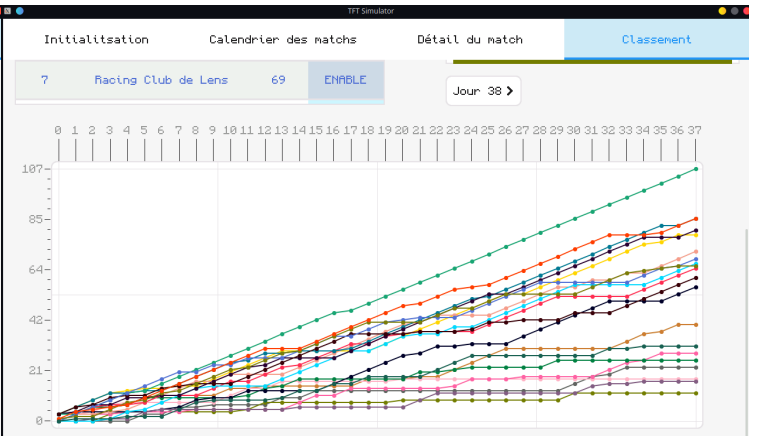
Pixel War



Exemples de projets

Football manager





II- Simulation

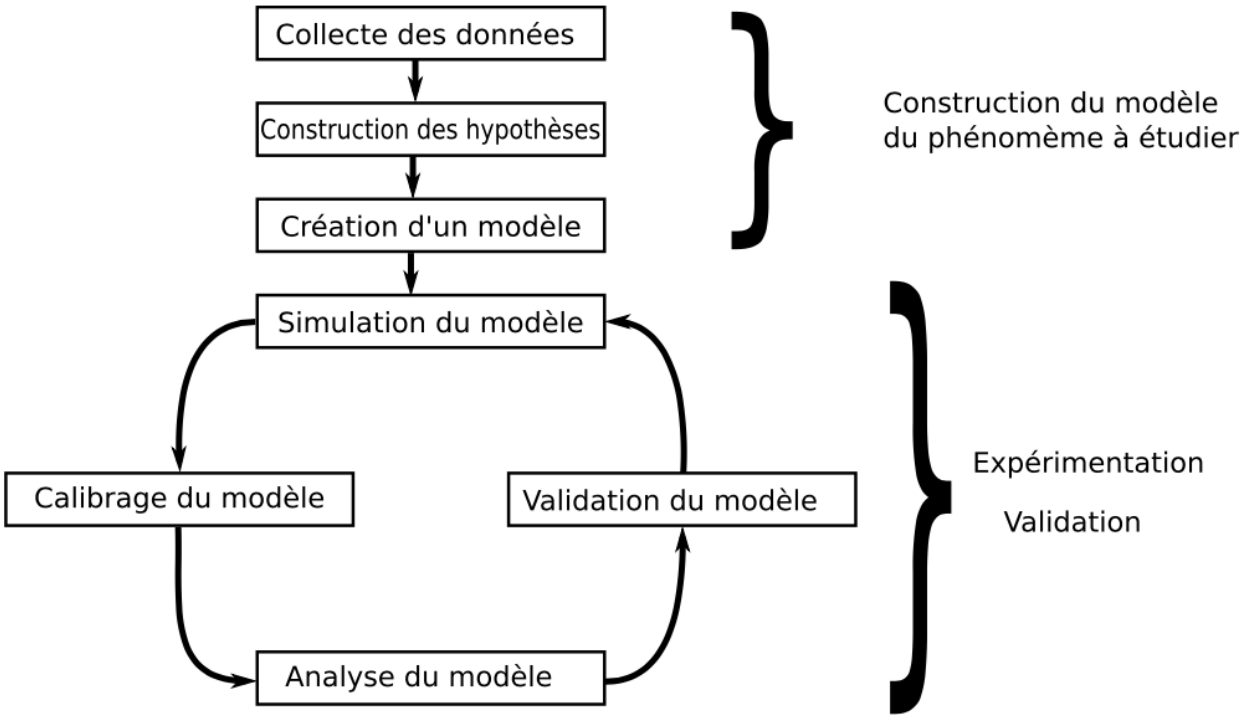
Définitions de base

> *Simulation : démarche scientifique qui consiste à réaliser une reproduction artificielle, appelée modèle, d'un phénomène réel que l'on désire étudier, à observer le comportement de cette reproduction lorsqu'on en fait varier certains paramètres, et à en induire ce qui se passerait dans la réalité sous l'influence de variations analogues.*

- *Système* : phénomène réel à étudier
- *Modèle* : représentation simplifiée d'un phénomène pour pouvoir l'étudier
- *Logiciel de simulation* : outil informatique qui permet de simuler numériquement un modèle.

II- Simulation

Cycle de création d'une simulation



Crédits : thèse de Benoît Calvez : Le calibrage de modèles à base d'agents pour la simulation de systèmes complexes

III- Simulation multi-agent

- Chaque agent représente un individu ou un groupe d'individus
- Idéalement elle est asynchrone, mais peut nécessiter des points de synchronisation

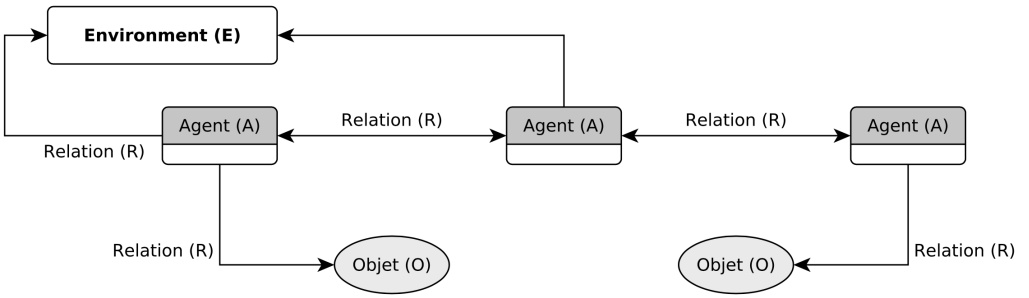
Exemples de frameworks de simulation multi-agent

Plateforme	**Langage de programmation**
GAMA	GAML
Jason	Java, AgentSpeak
MadKit	Java, C/C++, Pyhton
MasOn	Java
NetLogo	Logo
RePast	Java, C#, C++, Lisp, Prolog, Python
Swarm	Java

III- Simulation multi-agent

Modélisation d'un système multi-agent [Ferber,1995] :

- Un environnement E
- Un ensemble d'objets O situés dans E (ces objets peuvent être perçus, créés, détruits et modifiés par les agents)
- Un ensemble d'agents A avec ($A \subseteq O$) représentant les entités actives du système
- Un ensemble de relations (R) qui unissent des objets (et donc des agents) entre eux



Crédits : Thèse d'Emmanuel Hermellin : Modélisation et implémentation de simulations multi-agents sur architectures massivement parallèles

III- Simulation multi-agent

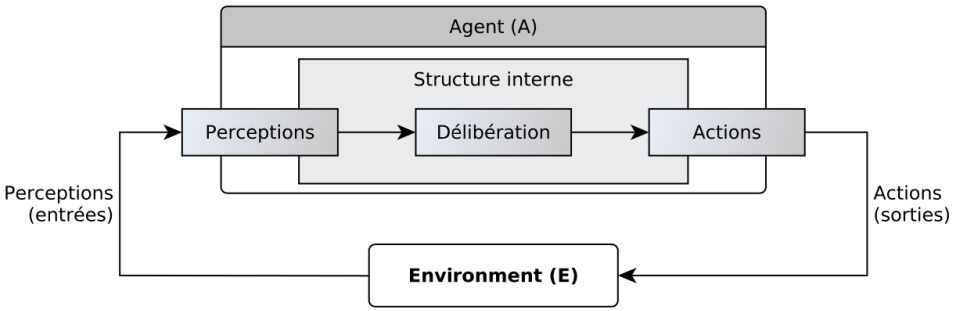
Code Go

```
type SimulationMA struct {
    env  *Environment
    agts []Agent
    objects []Objects
}


func (simu *SimulationMA) Run()
func (simu *SimulationMA) log()
func (simu *SimulationMA) print()
```

III- Simulation multi-agent

La boucle Perception/Délibération/Action



Crédits : Thèse d'Emmanuel Hermellin : Modélisation et implémentation de simulations multi-agents sur architectures massivement parallèles

Code Go

```
type Agent interface {
    Start()
    Percept(Environment)
    Deliberate()
    Act(*Environnent)
}


func (simu *SimulationMA) run()
```

III- Simulation multi-agent

La boucle de simulation

Définition : Boucle de simulation est une boucle infinie qui permet de réaliser les différentes actions nécessaires à la mise en œuvre d'un logiciel de simulation

Principales propriétés: le pas de simulation, la fréquence d'affichage et le temps de latence/réponse (dans le cadre d'une IHM)

Code Go

```
func (simu *SimulationMA) Run() {
    step := 0

    for {
        for _, agt := range simu.agts {
            agt.Percept(simu.env)
            agt.Deliberate()
            agt.Act(simu.env)
        }
        step += 1
        simu.log()   // On sauvegarde l'état courant de la simulation
        simu.print() // On "affiche le résultat actuel"
    }
}
```

III- Simulation multi-agent

Avantages de la modélisation

- Extrêmement simple
- Déterministe
- Efficace

Problèmes de cette modélisation

- MonoThread
- Déterministe (les mêmes agents commencent toujours en premiers
- Ne gère pas les communications entre agents (encapsulation dans Act ?)
- Ne profite pas de la parallélisation
- Omniscience des agents
- "Triche" possible des agents sur l'environnement

III- Simulation multi-agent

Solutions simples

- Mélanger le slice d'agents (`func Shuffle(n int, swap func(i, j int))` du package `range`) où à la main :

```
func Shuffle[T any](slice []T) {
    for i := range slice {
        j := rand.Intn(i + 1)
        slice[i], slice[j] = slice[j], slice[i]
    }
}
```

- Lancer des agents dans des goroutines et utiliser des points de synchronisation (`WaitGroup`) Attention à environnement !)
- Utiliser des copies (locales) d'environnement
- etc.

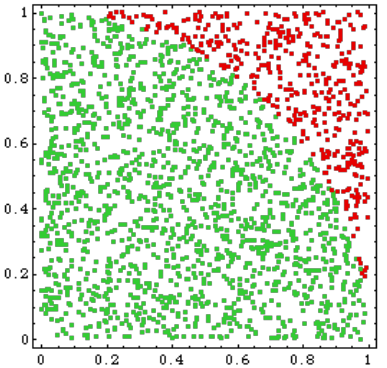
IV. Un petit exemple concret : calcul multi-agent de π

Principe

- On considère un quart de cercle de rayon 1
- On choisit un point au hasard (x, y) dans le carré $[(0, 0), (1, 1)]$
- On détermine s'il est dans le quart de cercle de rayon 1 c-à-d

$$x^2 + y^2 \leq 1$$

- On recommence n fois



$$\pi = 4.\frac{m}{n}$$

Avec m le nombre de points à l'intérieur du cercle.

IV. Un petit exemple concret : calcul multi-agent de π

Du code !

https://gitlab.utc.fr/lagruesy/ia04/-/tree/main/demos/simulations

- version 1 : boucle simple
- version 2 : workers et goroutines
- version 3 : les agents sont des goroutines
- version 4 : toutes les agents sont indépendants
- version 4 web : une API permet de faire le lien avec un client web

Améliorations possibles

- version 5 : l'environnement est un microservice
- version 6 : l'environnement gère des environnements différents (4 1/4 de cercles ou autre)
- version 7 : de la communication entre agents ?
- version 8 : un front permet d'afficher le résultat en direct et le cercle

Remarques sur les GUI

- client lourds vs. client légérs

Exemples de clients lourds

- basés sur des toolkits connus : gotk3, Qt,...
- Pur Go : Fyne,...
- Moteurs de jeux 2D: ebiten,...
- ...

plus d'exemples : https://github.com/go-graphics/go-gui-projects et https://github.com/avelino/awesome-go#advanced-console-uis

Remarques sur les GUI

Exemples de clients légers

- javascript vanilla
- React
- View.js
- Angular
- Svelte
- Elm
- ...

Remarques sur les GUI

Une architecture MVC like pour la gestion d'application web ou de GUI

V. Modélisation

Utilisation de l'approche objet/UML

- Ensemble d'agents/objets/Environnement = diagramme de classe
- Comportement d'un agent = diagramme d'état transition
- Communication entre agents = diagramme de séquence
- Interaction avec l'utilisateur = diagramme des cas d'utilisation

V. Modélisation

Les questions à se poser pour concevoir une simulation SMA

1. Déterminer la question à laquelle on veut répondre
2. Identifier les différents agents
3. Identifier l'environnement et les différents objets les composant
4. Comment gérer l'interaction entre l'environnement et les agents (avec les problèmes de concurrence) ?
5. Comment les agents communiquent-ils ?
6. Comment gérer la simulation (tour par tour, synchronisation, etc.) ?

V. Modélisation

Exercice : le chat et les souris

> *Des souris et un chat peuplent un monde. Le but des souris est de trouver du fromage et de se reproduire tandis que le but du chat est de manger des souris.* [sujet]