Projet IA04

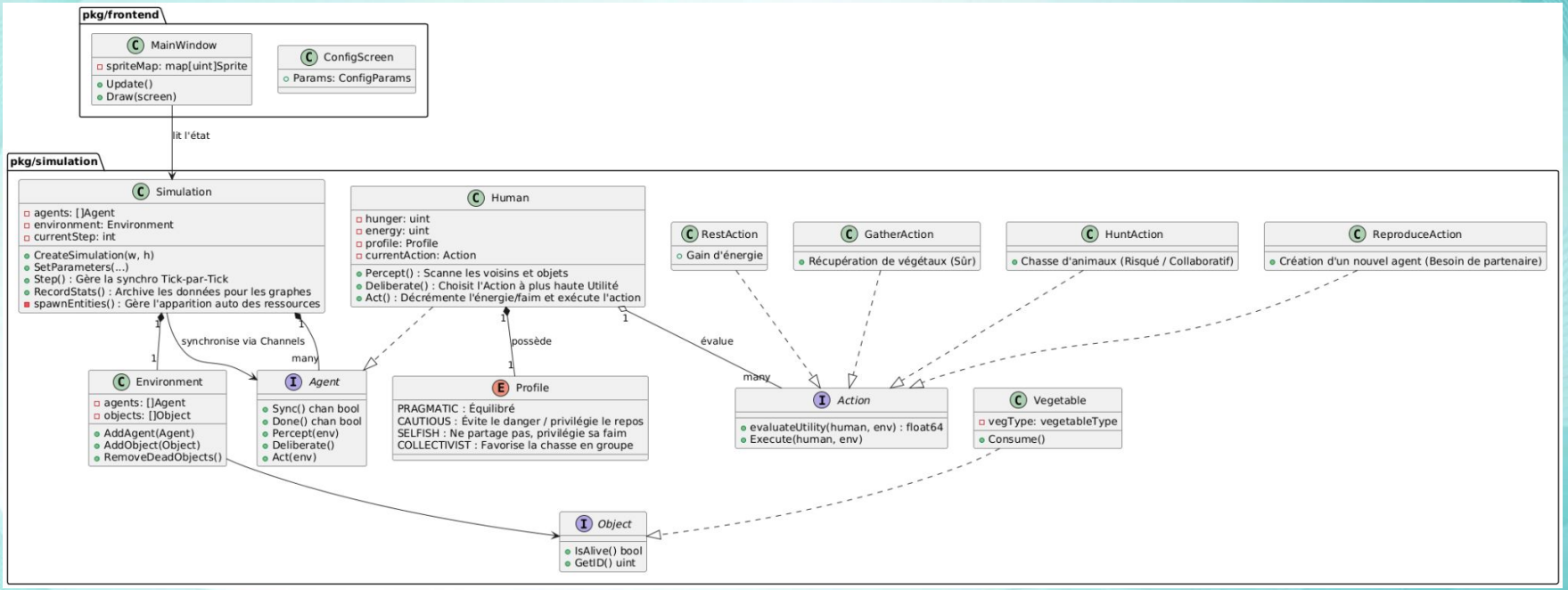
Simulation d'une Tribu Préhistorique

Maxime Gautrot

Problématique

Comment les profils individuels influencent-ils la survie collective du groupe et de ses membres ?

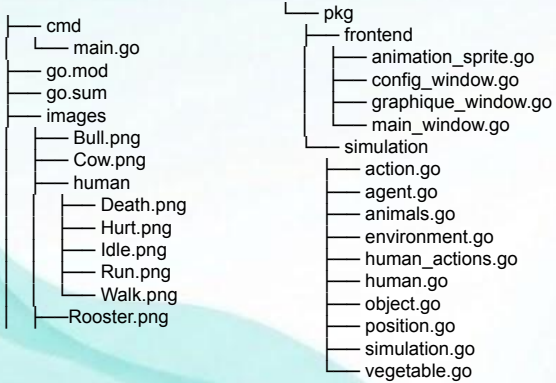
Architecture - UML



Architecture - Code

Technologie

- Backend : GoLang
- Frontend : Ebiten v2



- **Concurrence Native**: Utilisation des **Goroutines**
- **Communication par Channels**: Utilisation de `chan bool` pour la synchronisation stricte des cycles de vie ("Tick-par-Tick") sans partage de mémoire
- **Performance**

- **Client Lourd**: Choix d'une application native pour garantir une fluidité d'affichage.
- **Gestion des Spritesheets**
- **Interface Interactive**: Écran de configuration (`ConfigScreen`) et écran de statistiques (`GraphScreen`).

Architecture - Code

```
type Agent interface {

        // Cycle de vie (IA)
        Percept(env *Environment)
        Deliberate()
        Act(env *Environment)

        // Concurrence (Goroutines)
        Start(env *Environment)
        Sync() chan bool // Signal de début de tour
        Done() chan bool // Signal de fin de tour }
```

- Signal Sync : Le moteur Simulation envoie un booléen sur syncChan
- Phase IA : L'agent se débloque, exécute Percept(), Deliberate() et Act()
- Signal Done : Une fois l'action terminée, l'agent envoie un signal sur doneChan pour dire qu'il est prêt pour le tour suivant
- Attente du Moteur : La simulation attend que TOUS les agents aient répondu avant de passer au tick T+1

```
type Simulation struct {
        maxSteps int
        MaxAnimals int
        MaxPlants int

        currentStep int
        isRunning bool
        agents []Agent
        environment Environment

        lambdaAnimals float64
        lambdaPlants  float64

        InitHumans int
        InitAnimals int
        InitPlants int

        nextAnimalTime float64
        nextPlantTime  float64

        distPragmatic float64
        distCautious float64
        distSelfish  float64
        distCollectivist float64

        History []TurnData
        globalIDCounter uint
}
```

```
type AgentParams struct {
        id uint
        name string
        health int
        alive bool
        sprite Sprite

        syncChan chan bool
        doneChan chan bool
        stopChan chan bool
}
```

```
const (
        Selfish     Profile = iota
        Collectivist
        Pragmatic
        Cautious
)
```

```
type Human struct {
        AgentParams
        hunger uint
        profile Profile
        strategyType string
        energy  uint
        currentAction  Action
        visibleAgents  []Agent
        visibleObjects []Object
        tickCounter  int
        actionDuration int
}
```

L’IA des Agents

Le cycle de l'IA est structuré en trois étapes distinctes exécutées à chaque tick de synchronisation :

- Perception: L'agent scanne son environnement dans un rayon de vision pour identifier les ressources et les autres agents
- Délibération: L'agent évalue toutes les actions possibles (Rest, Gather, Hunt, Reproduce)
- Action: L'agent exécute l'action ayant obtenu le score d'utilité le plus élevé

Chaque action possède sa propre logique de scoring, pondérée par l'état interne de l'agent et son profil :

- Besoins vitaux : L'utilité de Gather ou Hunt augmente proportionnellement au niveau de faim
- Facteurs environnementaux : La distance de la cible et la dangerosité (nombre de chasseurs requis) réduisent le score
- Filtres de reproduction : L'action Reproduce n'est envisageable que si l'énergie est haute et que l'environnement est riche en nourriture

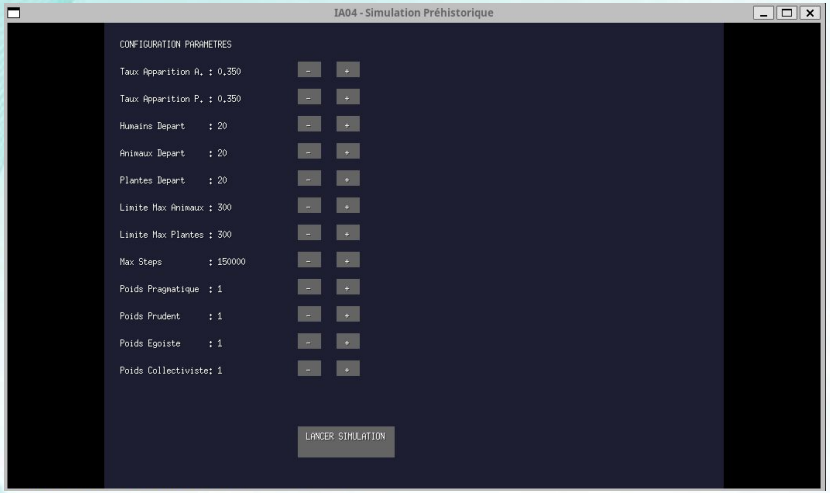
L’IA des Agents

```
hu.TargetID = closest.GetID()

        utility := (float64(h.hunger) * 1.5) - (minDist * 0.1)
        risk := float64(closest.GetPeopleNeeded()) * 10

        switch h.profile {
        case Cautious:
                utility -= risk * 1.25
        case Selfish:
                utility -= (float64(closest.GetPeopleNeeded()) - 1) * 15
        case Collectivist:
                utility += 50
        case Pragmatic:
                if h.energy > 200 {
                        utility += 50
                }
                utility += float64(currentHunters) * 15
        }

        return math.Max(0, utility)
```

Le profil modifie dynamiquement les priorités de l'agent via des multiplicateurs :

- Prudent (Cautious) : Bonus d'utilité pour le repos et forte pénalité sur les actions risquées comme la chasse
- Égoïste (Selfish) : Réduit l'utilité des actions collaboratives si le gain individuel n'est pas immédiat
- Collectiviste (Collectivist) : Reçoit un bonus d'utilité pour la chasse
- Pragmatique (Pragmatic) : Favorise l'action la plus efficace selon ses réserves d'énergie actuelles

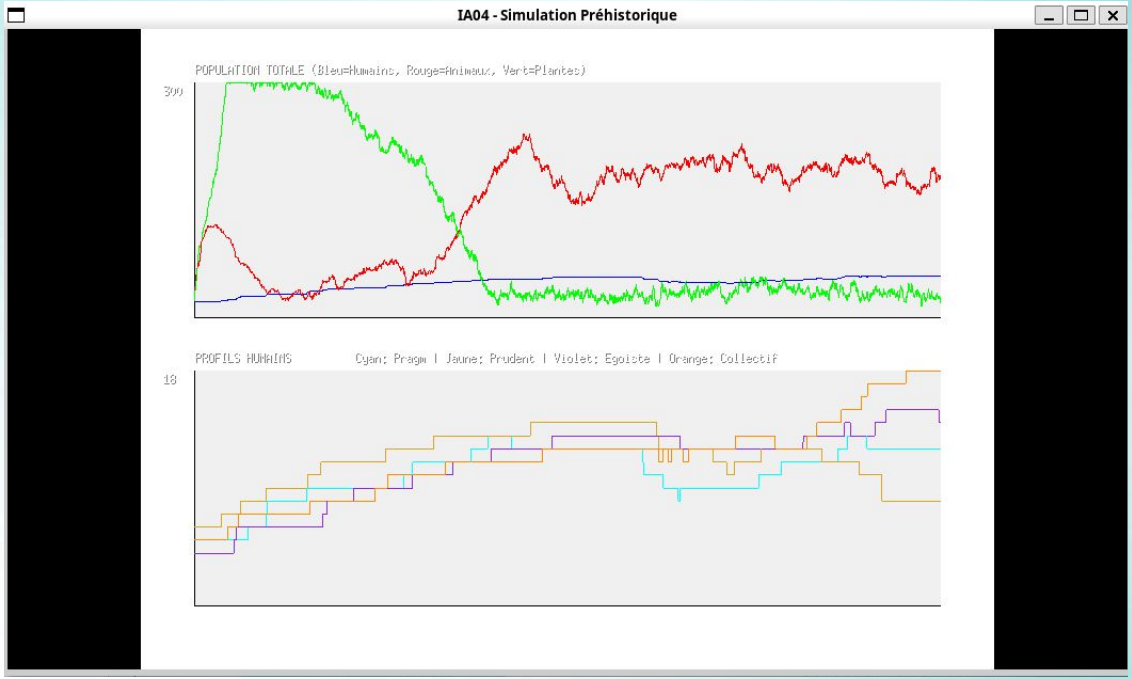
Le Rendu



panneau de configuration

Simulation

Le Rendu



Graphiques et Résultats

Problèmes et Axes d'Améliorations

Problèmes

- Complexité Algorithmique de la Perception : La détection des voisins dans Percept() se fait par un balayage linéaire de tous les agents O(N^2)
- Concurrence et Verrous (Contention): L'utilisation d'un Mutex global sur l'environnement peut créer un goulot d'étranglement
- Équilibrage des Paramètres : Les constantes de métabolisme (faim, énergie) et les poids d'utilité sont difficiles à équilibrer pour éviter soit une extinction immédiate, soit une surpopulation incontrôlable

Améliorations possibles

- Partitionnement Spatial: Remplacer la recherche linéaire par une structure de données spatiale pour optimiser la vision des agents.
- Communication Directe: Implémenter un système de passage de messages entre agents pour permettre une véritable coopération négociée.
- Apprentissage: Remplacer les poids d'utilité fixes par un apprentissage simple, permettant aux profils d'évoluer selon leurs succès passés
- Gestion du Terrain: Ajouter des obstacles (murs, rivières) ou des zones de ressources spécifiques

Merci pour votre attention