

# Rendu IA troupe rapide

---

## Table des matières

- [Rendu IA troupe rapide](#)
  - [Table des matières](#)
  - [1. Informations générales](#)
  - [2. Localisation du code](#)
  - [3. Architecture et stratégie](#)
    - [3.1 Cycle décisionnel](#)
    - [3.2 Schéma de décision](#)
      - [3.2.1 Évaluation d'objectifs \(priorités décroissantes\)](#)
      - [3.2.2 Comportements d'états avec délégation de navigation](#)
    - [3.3 Comportements par état](#)
    - [3.4 Services de support](#)
  - [4. Algorithmes implémentés](#)
  - [5. Mise en route et observation](#)
    - [5.1 Cloner le dépôt et choisir la bonne branche](#)
    - [5.2 Préparer l'environnement Python](#)
    - [5.3 Lancer le jeu](#)
    - [5.4 Interagir avec les unités et observer l'IA](#)
  - [6. Réglages "avancés" utiles](#)

## 1. Informations générales

- **Nom / Prénom** : Lambert Romain
- **Jeu** : Galad Island
- **Unité ciblée** : troupe rapide ennemie (Zasper / Scout)
- **Branche Git** : [IA\\_LAMBERT](#)

## 2. Localisation du code

- **Répertoire principal** : [src/ia\\_troupe\\_rapide](#)
- **Fichiers clés** :
  - [config.py](#) — paramètres équilibrage (danger, pathfinding, pondérations d'objectifs, debug).
  - [processors/rapid\\_ai\\_processor.py](#) — boucle principale ECS, machine à états, coordination multi-unités.
  - [states/](#) — comportements concrets (attaque, fuite, navigation, suivi du druide, etc.).
  - [services/](#) — briques transverses (pathfinding pondéré, carte de danger dynamique, prédiction, évaluation d'objectifs, coordination).
- **Point d'intégration** : [src/ia\\_troupe\\_rapide/integration.py](#) est invoqué depuis [src/game.py](#) ([ensure\\_ai\\_processors](#)).

## 3. Architecture et stratégie

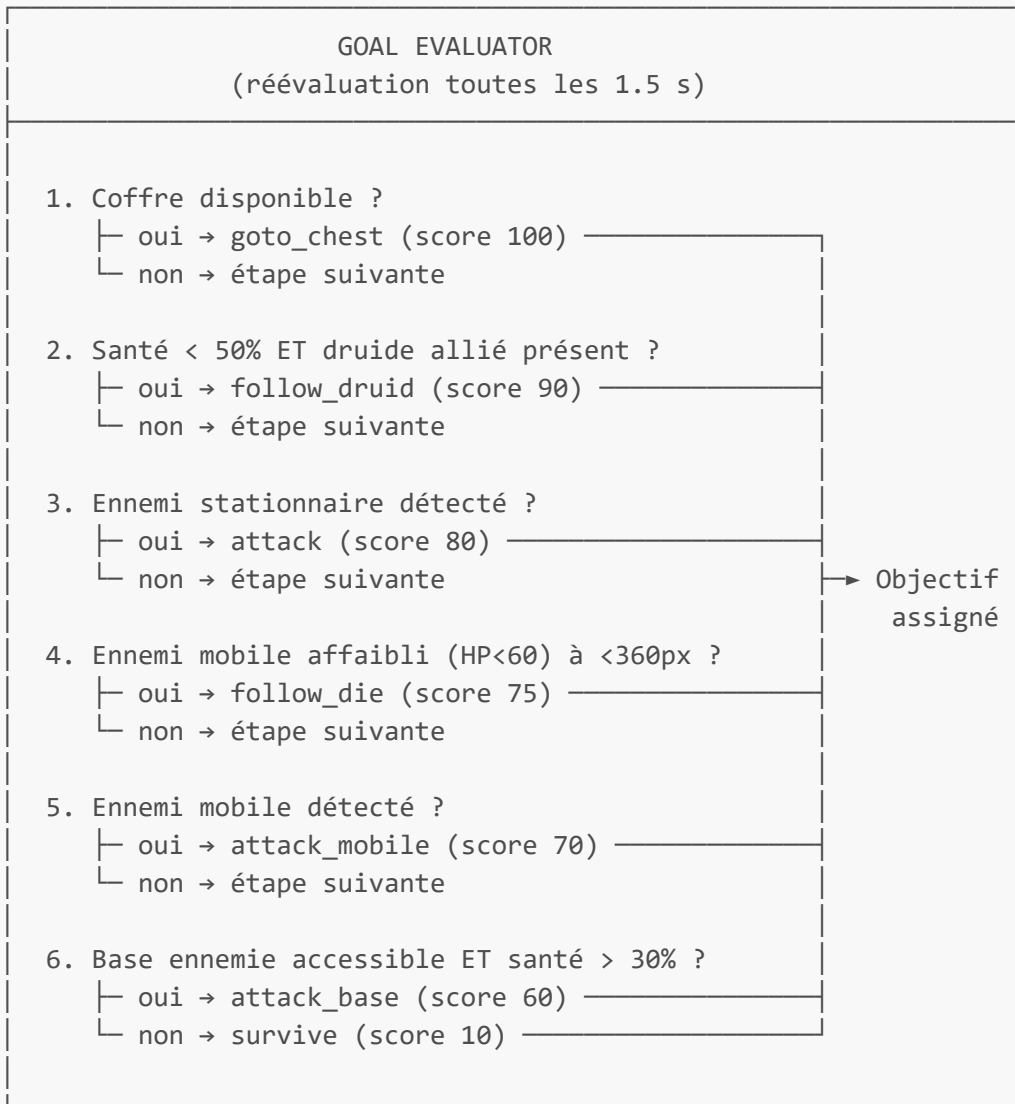
### 3.1 Cycle décisionnel

1. **Collecte de contexte** : `AIContextManager` (services/context.py) agrège chaque tick la santé, la position, les composants SpeScout, etc.
2. **Évaluation d'objectifs** : `GoalEvaluator.evaluate` scanne coffres, druides alliés, ennemis détectés et casse la base selon priorités configurées.
3. **Machine à états** : `StateMachine` (fsm/machine.py) sélectionne l'état comportemental en fonction des transitions globales (danger, navigation en cours, suivi du druide) et locales (objectif atteint, cible perdue...).
4. **Actions unitaires** : les états (`states/*.py`) pilotent navigation, tir continu, activation d'invulnérabilité et partage d'informations via `share_channel`.

## 3.2 Schéma de décision

Le système décisionnel se compose de trois couches interdépendantes :

### 3.2.1 Évaluation d'objectifs (priorités décroissantes)



### 3.2.2 Comportements d'états avec délégation de navigation

## ÉTATS COMPORTEMENTAUX

## IDLE

- Annule navigation active
- S'éloigne des zones dangereuses ( $\text{danger} > \text{seuil\_alerte}$ )
- Vitesse réduite

## GOTO (état de navigation délégué)

- Suit chemin A\* pondéré (recalcul toutes les 0.6s)
- Coordonne avec autres IA (évitement collisions)
- Complete → retour à `nav_return_state`
- Peut être interrompu par transitions globales

## ATTACK

- Sélectionne ancre de tir (85% portée optimale)
- Force arrêt (vitesse → 0)
- Tir continu (`_try_continuous_shoot`)
- Si base bloquée → repositionnement orbital via GOTO
- Priorité basse : cède si coffre détecté

## FLEE

- Calcule point sûr (bonus -10 près base amie)
- Active invincibilité SpeScout si  $\text{vie} < 30\%$
- Navigation prioritaire via GOTO (`nav_return="Flee"`)
- Hystérésis:  $\text{seuil\_sortie} (0.3) < \text{seuil\_entrée} (0.5)$

## FOLLOW\_DRUID

- Orbite autour druide allié (rayon 160px)
- Prédiction position druide (0.8s)
- Navigation via GOTO (`nav_return="FollowDruid"`)
- Sortie si  $\text{santé} \geq 50\%$

## FOLLOW\_TO\_DIE

- Poursuite agressive cible affaiblie
- Prédiction courte portée (0.4s)
- Tir continu pendant poursuite
- Sortie si cible perdue ou éliminée

NOTE: Le système de navigation est hybride - les états demandent une navigation (`start_navigation`), l'état GOTO l'exécute, puis retourne à l'état appelant (`nav_return_state`) une fois la destination atteinte.

### 3.3 Comportements par état

- **IdleState** : annule la navigation et quitte les zones dangereuses.

- **GoToState** : suit les chemins pondérés, recalcule toutes les 0,6 s si la cible s'éloigne et suit des points de passage optimisés.
- **AttackState** : choisit une "ancree" de tir à portée optimale, ajuste la direction, force l'arrêt, tire en boucle et repositionne autour des bases adverses si l'accès direct est bloqué.
- **FleeState** : sélectionne un point sûr (bonus autour de la base amie), déclenche l'invincibilité SpeScout si la vie est basse et maintient une navigation prioritaire.
- **FollowDruidState** : orbite autour du druide allié à 160 px pour regagner de la santé.
- **FollowToDieState** : poursuit agressivement une cible affaiblie en utilisant la prédiction courte portée.
- **AttackState** et **FollowToDieState** profitent du tir continu imposé par `RapidUnitController._try_continuous_shoot`.

### 3.4 Services de support

- **DangerMapService** : met à jour une carte de chaleur (décroissance exponentielle, signaux pour : projectiles, tempêtes, bandits).
- **PathfindingService** : A\* pondéré sur grille sous-tuillée (pour plus de détails), coûts ajustés (îles, mines, nuages, danger), `numba` pour l'heuristique.
- **PredictionService** : extrapolation cinématique sur 0,8 s et vecteurs d'évitement de projectiles.
- **CoordinationService** : évitement entre IA, attribution des coffres, diffusion de danger global.
- **IAEventBus** : relaye apparitions/disparitions (coffres, tempêtes) depuis `RapidTroopAIProcessor`.

## 4. Algorithmes implémentés

- **Carte de danger dynamique** (`services/danger_map.py`) : grille float32 avec décroissance `decay_per_second`, impulsions circulaires, bonus négatif autour de la base amie, clamps à `max_value_cap`.
- **Pathfinding pondéré** (`services/pathfinding.py`) : A\* avec heuristique euclidienne (`_heuristic_numba`), sous-tuiles (facteur 2), périmètre d'obstacles (`sliding_window_view`), marges forcées et tolérance de navigation.
- **Évaluation d'objectifs** (`services/goals.py`) : décision déterministe priorisant coffres > soins > attaques statiques > poursuite > base > survie.
- **Prédiction & tir** (`services/prediction.py`, `states/attack.py`) : projection directionnelle simple pour viser les cibles mobiles et sécuriser la cadence de tir.
- **Coordination multi-unités** (`services/coordination.py`) : champ d'évitement vectoriel et rotation de rôles pour limiter les collisions.

## 5. Mise en route et observation

### 5.1 Cloner le dépôt et choisir la bonne branche

- **Récupérer le dépôt** (depuis un terminal placé dans le dossier cible) :

```
git clone https://github.com/Fydyr/Galad-Islands.git
cd Galad-Islands
git checkout IA_LAMBERT
```

Remarque : les instructions ci-dessous supposent que vous restez à la racine du dépôt.

## 5.2 Préparer l'environnement Python

- **Créer un environnement virtuel** (recommandé avant de lancer `setup_dev.py`).

- Windows (PowerShell) :

```
python -m venv .venv
.venv\Scripts\Activate.ps1
```

- Linux :

```
python3 -m venv .venv
source .venv/bin/activate
```

Si vous oubliez cette étape, `setup_dev.py` détecte l'absence de `venv` et propose d'en créer un automatiquement.

- **Installer les dépendances :**

- Méthode conseillée (installe `pip`, `Commitizen`, `requirements` et `hooks`) :

```
python setup_dev.py # python3 sous Linux
```

- Alternative manuelle si vous ne souhaitez pas exécuter le script :

```
pip install --upgrade pip
pip install -r requirements.txt
pip install -r requirements-dev.txt
```

## 5.3 Lancer le jeu

- **Démarrer Galad Islands :**

```
python main.py # python3 sous Linux si nécessaire
```

- **Dans le menu principal**, cliquer sur « Jouer ». Le chargement initial place automatiquement deux Scouts sur la carte :

- Un Scout allié contrôlable manuellement (aucune IA attachée).
- Un Scout ennemi piloté par `RapidTroopAIProcessor`.

## 5.4 Interagir avec les unités et observer l'IA

- **Contrôler les unités alliées :**
  - Sélection via clic gauche puis déplacement avec **ZQSD** (configuration par défaut) ou les raccourcis reconfigurés dans les options.
  - Les unités alliées que vous créez ne sont **pas** reliées à l'IA rapide.
- **Analyser le comportement IA :**
  - L'ennemi Scout actif suit toutes les transitions d'état décrites dans la section 3.
  - Appuyer sur **F3** pour afficher l'overlay debug (chemins, état, fps) afin de suivre la navigation et les décisions.
- **Changer de camp si besoin :**
  - Touche **T** (ou bouton d'interface en bas à droite) pour basculer vers le camp adverse.
  - La boutique (icône en bas à gauche) permet de faire apparaître d'autres unités. Seuls les Scouts ennemis possèdent la logique IA rapide ; les autres unités restent manuelles.
- **Relancer la scène :** revenir au menu principal via **Échap** puis « Jouer » pour réinitialiser la situation si nécessaire.

## 6. Réglages "avancés" utiles

- **assets/ia\_troupe\_rapide/config.json** (ou **config/ia\_troupe\_rapide.json** si présent) permet de surcharger les paramètres sans modifier le code.
- Les seuils de danger (**DangerSettings**) contrôlent l'entrée/sortie de l'état **Flee** et l'agressivité globale.
- **PathfindingSettings.island\_perimeter\_weight** et **blocked\_margin\_weight** ajustent l'évitement des obstacles (utile pour tester des cartes alternatives).
- **AISettings.tick\_frequency** régule la fréquence de décision (par défaut 10 Hz).