AIBOOTCAMP 2018

The Inclusive JellyFishes

Guillaume BUCHLE, Iannick LANGEVIN, Alexys DUSSIER,
Redha MAOULY-VILATIMO, Pierrick HUMBERT

DDJV - cohorte 14

UNIVERSITÉ DE SHERBROOKE

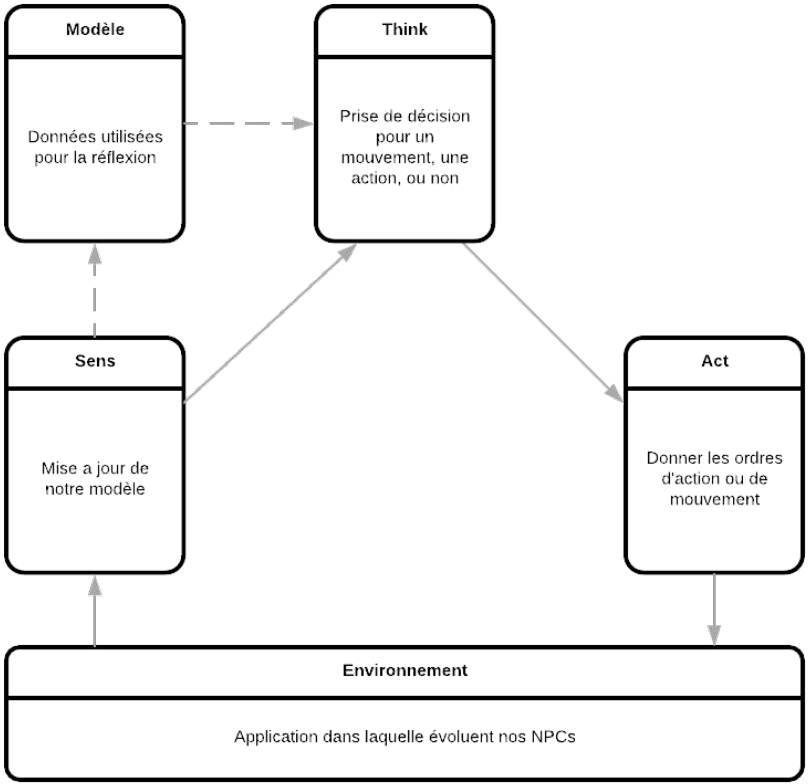
Introduction

- Concours entre les étudiants
- Résolution de niveaux par des NPCS
- But :
  - Application des concepts vu en cours
  - Trouver des solutions personnelles
  - Avoir une réflexion adaptée au Game AI

Plan

- Notre Architecture
- Les différentes problématiques
- Les techniques de résolutions
  - Recherche du Plus Court Chemin
  - Gestion du Comportement
  - Calcul de Score et Heuristiques
  - Flood Fill
  - Gestion des portes
- Optimisations
- Limites et Améliorations envisagées
- Résultats et Conclusion

Notre architecture



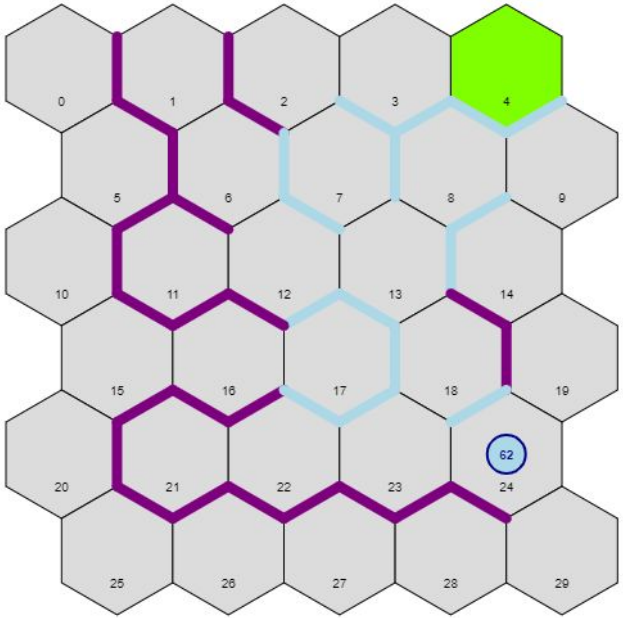
Les différentes problématiques

- Problème à résoudre par le Bot
  - Recherche du Chemin le plus Court
  - Gestion de la vision
  - Gestion des portes
  - Collaborer entre NPCs

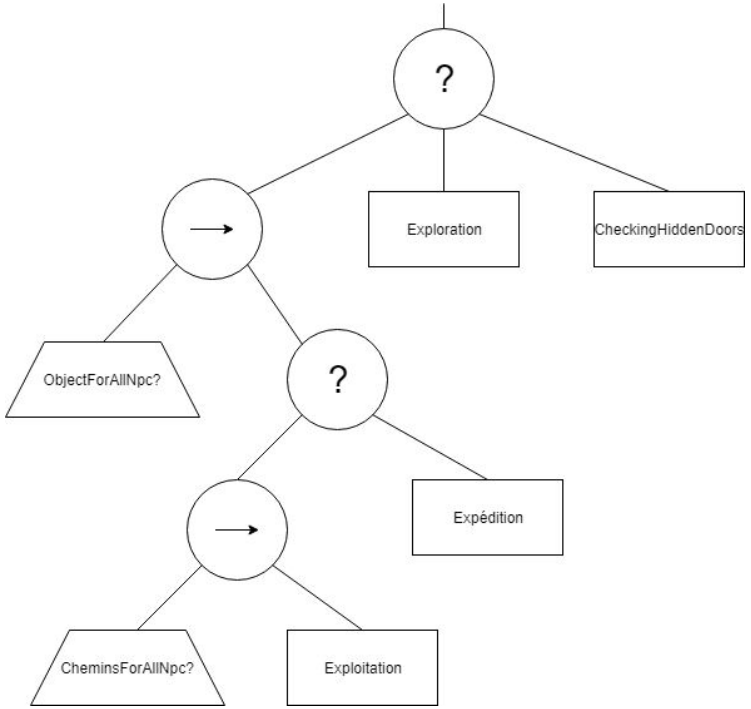
- Problème à résoudre par le Développeur
  - Avoir un Bot qui réussi le challenge
  - Avoir un Bot suffisamment rapide
  - S’adapter à la puissance du serveur cible

Recherche du Plus Court Chemin

- Algorithme A*



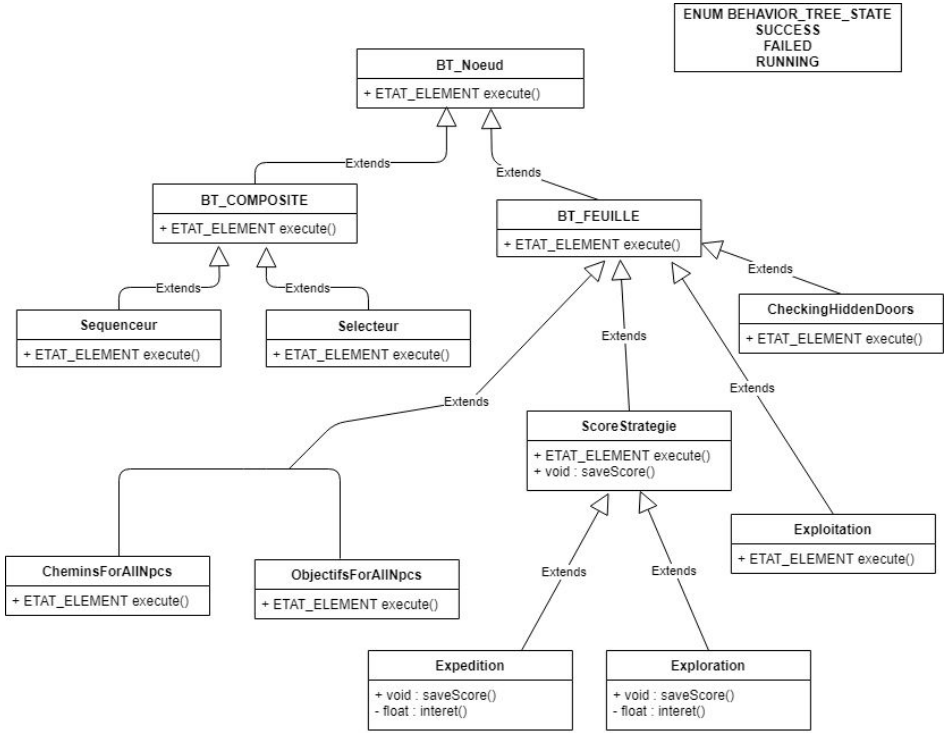
Gestion du comportement : Behavior Tree



4 Phases :

- Exploration
  - Recherche des buts
- Expédition
  - Recherche des chemins des buts
- Exploitation
  - Les NPC se dirigent vers les buts
- Recherche de Portes Cachées

Gestion du comportement

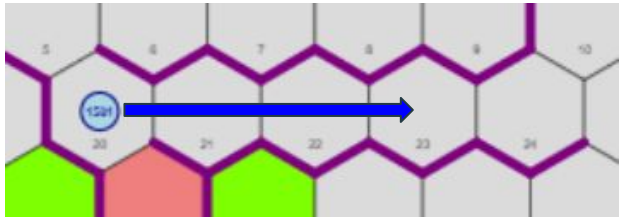


Heuristiques d’exploration et d'expédition

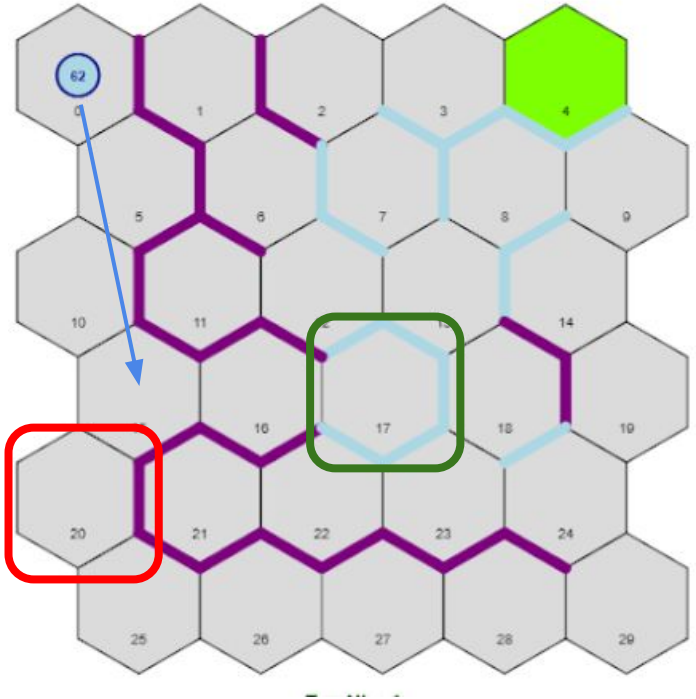
- Définir un score par tuile
- Sélectionner un objectif par NPC
- Deux heuristiques

Source : FlatIcon

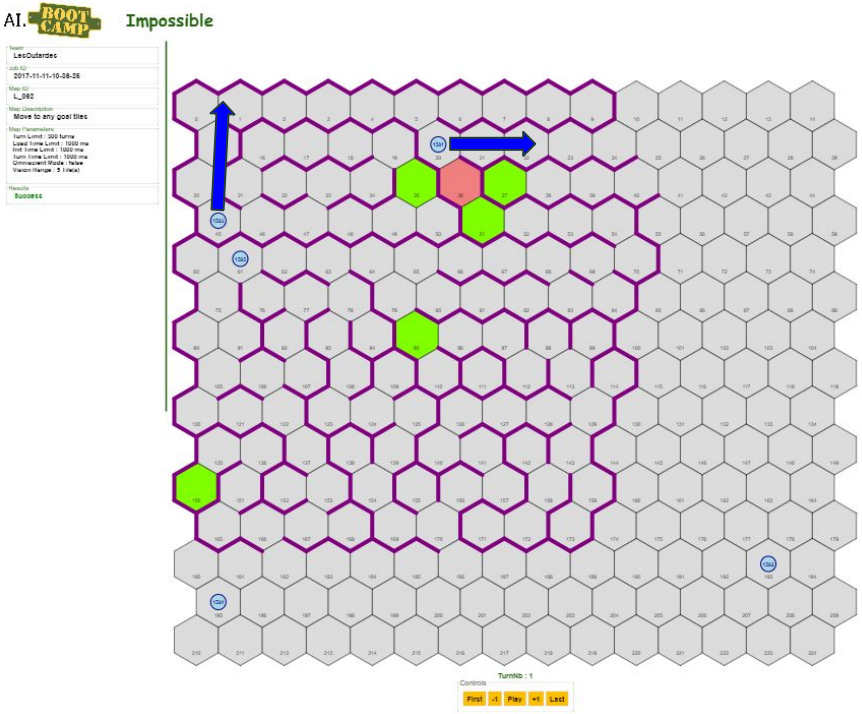
Exploration



- Nombre de mouvements nécessaires
- Degré d’ouverture de la tuile



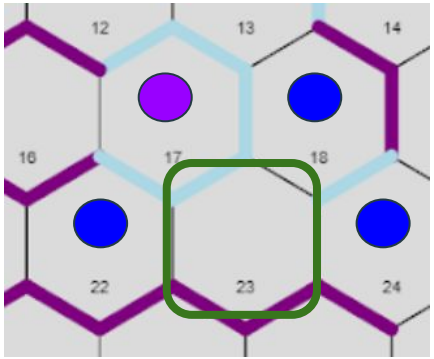
Exploration



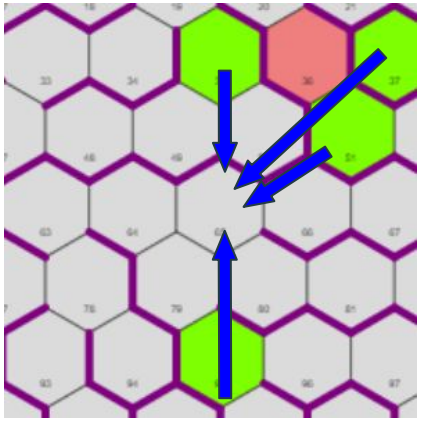
- Chaque NPC reste dans une zone
- Maximiser la découverte
- Éviter la concurrence

Exploration

- L’intérêt d’une tuile
  - tuiles visibles mais inaccessibles
  - tuiles accessibles

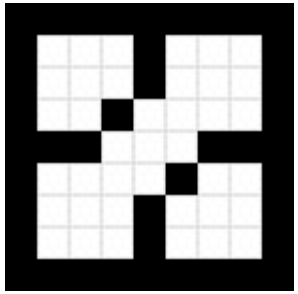


Expédition



- Score quasi-identique à l'exploration
- Ajout d'un paramètre
  - distance moyenne de la tuile à tous les objectifs

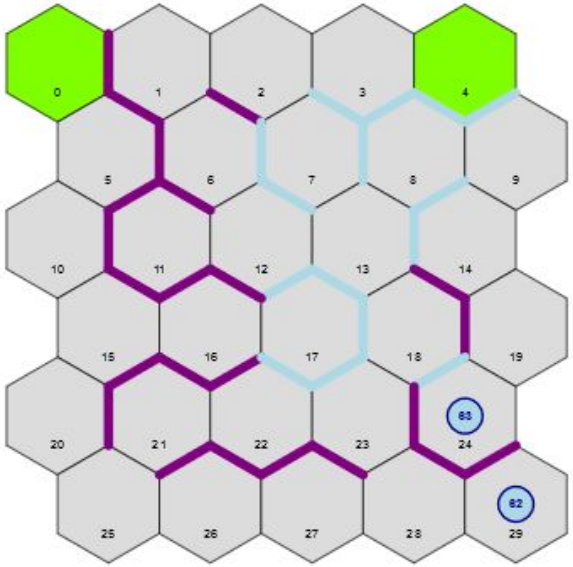
Flood Fill



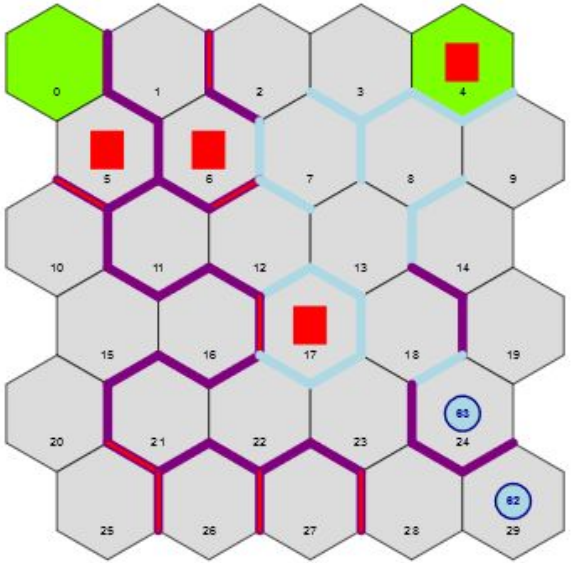
Source : Wikipedia

- Algorithme de remplissage
- Permet de connaître les tuiles connexes
- Stocke le nombre de déplacements nécessaires pour aller de A à B
- Chaque NPC stocke son vecteur de cases accessibles
- Rafraîchissement périodique du FloodFill (Période = 1 Tour)
- L’algorithme A* trouve toujours un chemin

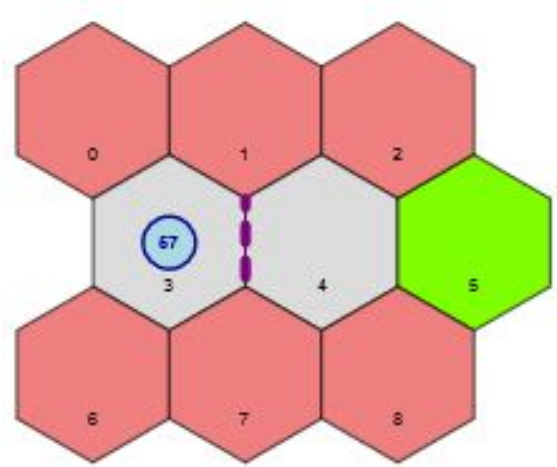
Gestion des Portes : Principe Fondamental



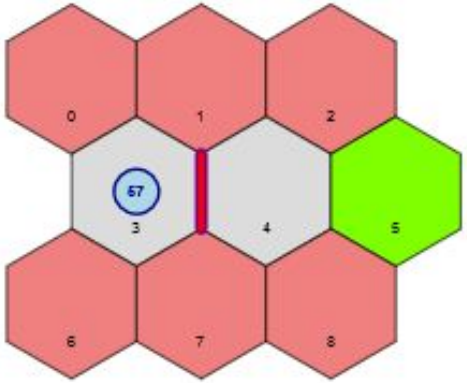
La Transparence



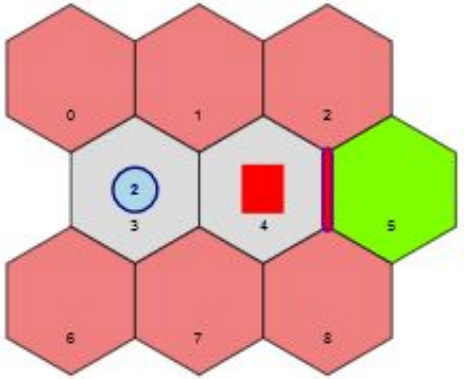
Gestion des Portes : Notre porte préférée



Gestion des Portes : Types de portes



Porte à poignée



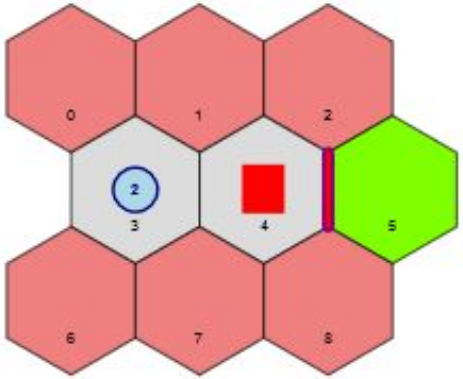
Porte à switch

Gestion des Portes : Les portes à poignées

- Ignorées dans le parcours du FloodFill
- Ignorées dans la recherche avec A*
- Rajoute juste un poids de 1 au chemin

Gestion des Portes : Les portes à switch

Les portes à switch : Le cas transparent



Les portes à switch : Ce que l’on veut

- Une seule chose
- Savoir si on peut passer une porte
- Et c’est tout

Les portes à switch : Les Contraintes

- Composante d’un Chemin
- Caractérise le besoin de passer par une porte
- Est orientée

Les portes à switch : Résoudre une Contrainte

- Lui associer un Npc non-occupé
- Trouver un chemin du Npc à un switch de la porte

Les portes à switch : Algorithme Récursif

- Pour résoudre une contrainte il faut un Chemin
- Pour parcourir ce Chemin il faut résoudre une contrainte
- Pour résoudre une contrainte il faut un Chemin
- Pour parcourir ce Chemin il faut résoudre une contrainte
- Pour résoudre une contrainte il faut un Chemin
- Pour parcourir ce Chemin il faut résoudre une contrainte
- Pour résoudre une contrainte il faut un Chemin
- Pour parcourir ce Chemin il faut résoudre une contrainte

Les portes à switch : Les Cas de Base

- Si aucun autre Npc ne peut activer un switch
- Si un Npc à besoin de la même contrainte plusieurs fois
- Si un autre Npc trouve un Chemin jusqu'à un switch
- Si plusieurs trouvent, on garde le Chemin le plus rapide

Les portes à switch : Répartir les Contraintes

- Le Npc le plus éloigné de son objectif est prioritaire

Gestion des portes : Conclusion

Outils de Débogage

- Logger
  - garder une trace écrite du déroulement du programme
- Profiler
  - mesurer le temps d'exécution d’une méthode
  - écrire dans un logger du texte, notamment sa durée d’exécution
  - visualisation graphique via l’outils “tracing” de Google Chrome

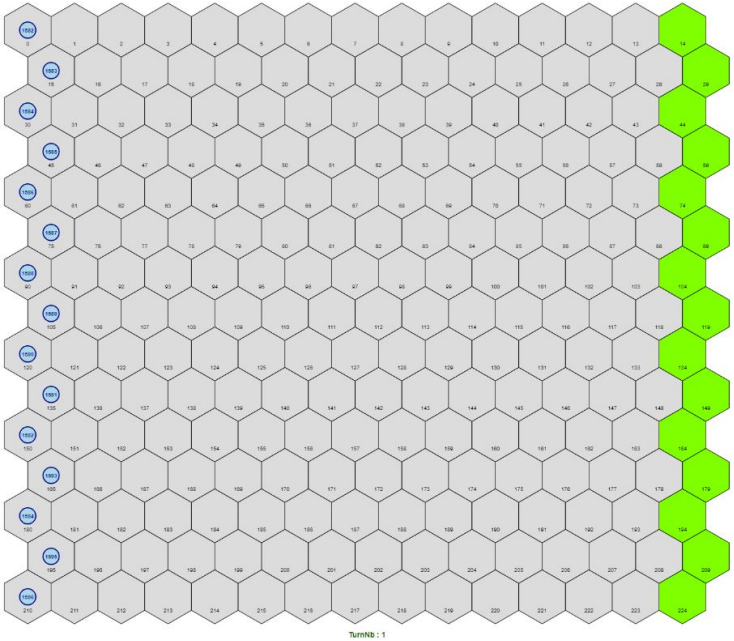
Optimisations

- Optimisations mineurs
  - Retour de fonctions
  - Éviter les copies sauf si nécessaire
  - Bon choix de structures de données
  - A*
  - Réaffectation d'objectif
- Utilisation du *MultiThreading*
  - Nombre de tours alloués VS Nombre de tours nécessaires
  - Identifier les goulots d'étranglement -----> concurrence?
  - Dépendance dans notre code

Optimisations

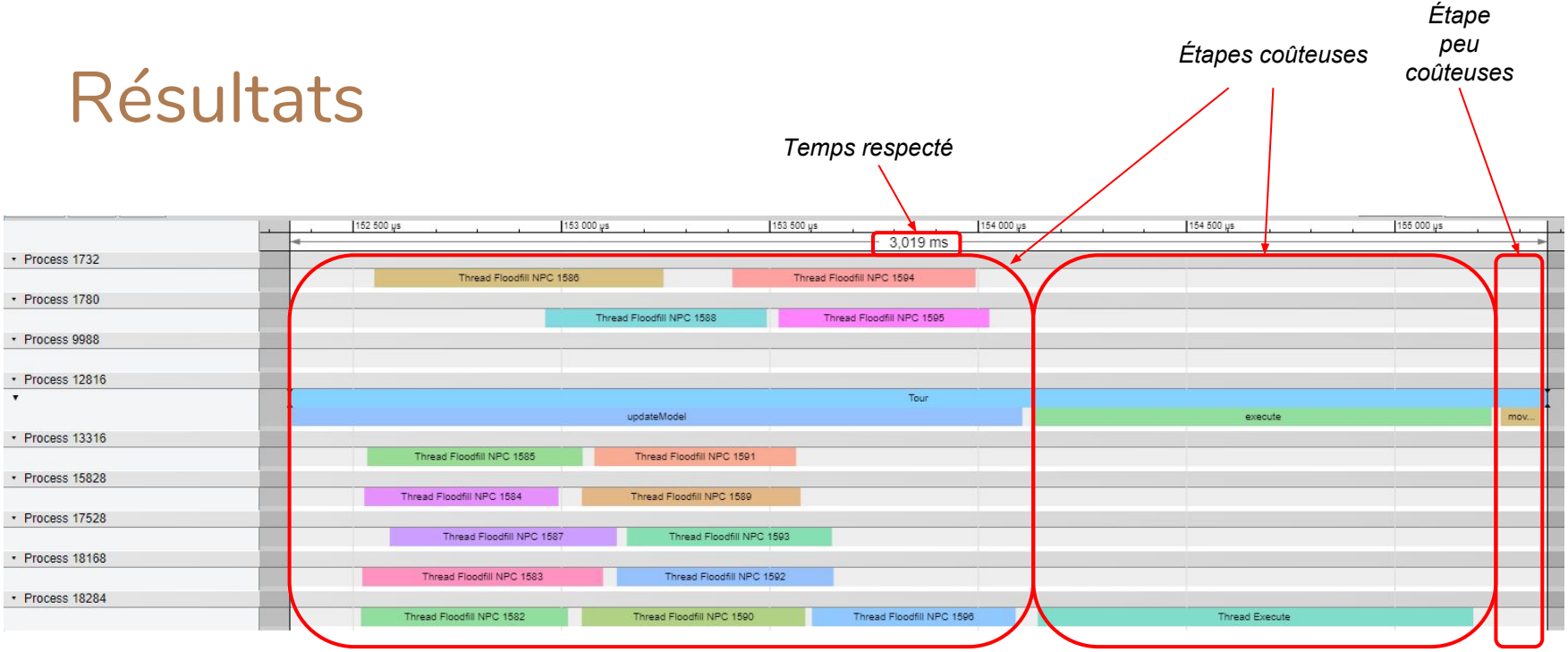
```
1     enum TaskState { NEED_TO_RUN, RUNNING, READY };
2
3     TaskState StateFloodFill; // ~1/2 timeAllowed
4     TaskState StateExecute;   //  1/4 timeAllowed
5
6     if (StateFloodFill::NEED_TO_RUN && StateExecute::NEED_TO_RUN)
7        updateModel();
8
9     if (StateFloodFill::READY && StateExecute::NEED_TO_RUN)
10       execute();
11
12    if (StateFloodFill::READY && StateExecute::READY)
13       moveNpc();
14
15    //END TURN
```

Optimisations



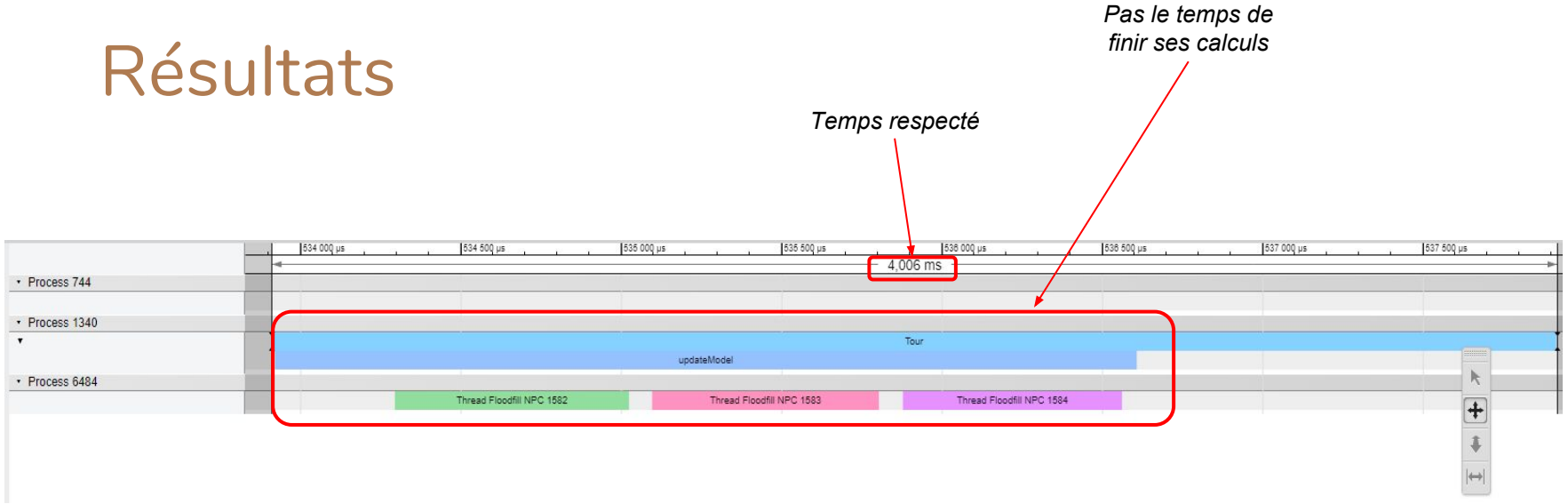
Déplacement #	Nombre de tours requis
1	2
2	2
3	2
4	2
5	2
6	3
7	2
8	2
9	3
10	3
11	2
12	2
13	2
14	2

Résultats



Performance sur le PC de développement

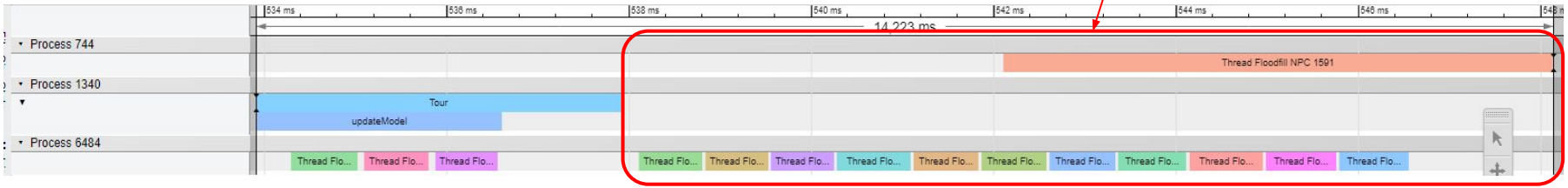
Résultats



Performance sur le Serveur : tour “1”

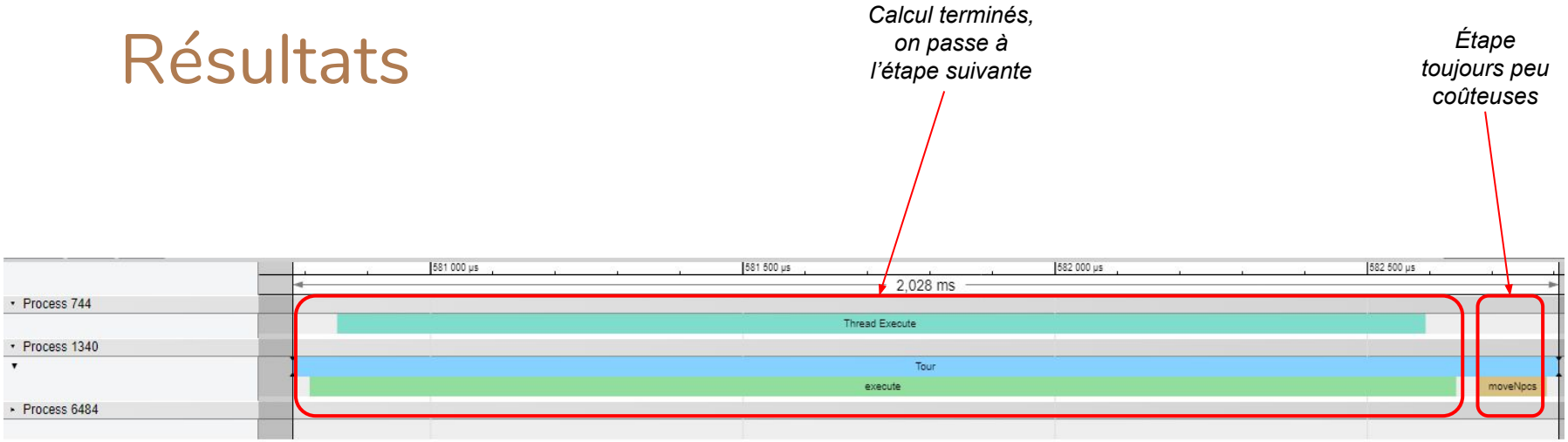
Résultats

Calcul pendant le temps serveur



Performance sur le Serveur : tour “1” et temps serveur

Résultats



Performance sur le Serveur : tour “2”

Limites

- Bot coûteux en temps de calcul
- Forte dépendance de nos modules/outils rendant difficile la modification du modèle
- Bot n’exploitant peut-être pas suffisamment les spécificité du serveur cible

Améliorations envisagées

- Nouvelle optimisation de l'algorithme A* et du floodfill
- Moins de dépendance inter-modules pour faciliter l'adaptation du modèle
- Architecture mieux pensée pour faire du multi-threading

Conclusion

- Compétences techniques acquises :
  - Arbre de comportement
  - Flood Fill
  - Algorithmes de recherches de chemins
- Travail en Équipe :
  - Comprendre du code étranger
  - Répartition du Travail
  - Utilisations d'outils de débogage
- Expérimentation des enjeux de l'IA dans le jeu-vidéo
  - Temps alloué très court
  - Problème complexes à résoudre
  - Machine(s) cible(s) de puissance(s) ou configuration(s) différente(s)

Questions ?