IA pour les jeux vidéos

Objectifs

- Connaitre les algos d'IA classique
- Implémenter un Algo d'IA en C++
- Renforcer les bonnes habitudes en C++
- Un peu de veille techno

Plan du cours

- Mercredi Après midi : Introduction + Cours State Machine
- Jeudi Matin : Projet StateMachine
- Jeudi Après midi : Cours GOAP
- Vendredi Matin : Projet GOAP
- Vendredi Après midi : Mini soutenance

Le contenu du cours / Sujet de projet

- Machine à état
  - État / Condition
  - Sub State Machine
- GOAP
  - Action / Précondition / Effet
  - GOAP Planner
- Behaviour Tree
  - Selector / Sequence
  - Sub Tree
- Monte Carlo (domineering)
  - MinMaxAlgo
  - MonteCarlo + Environnement de test

Rendu

- C++
- Rendu par GIT avant le 11/09 23h59m59s
- M’envoyer par mail le groupe + le lien git
  - emerick.lecomte.pro@gmail.com
- Machine a état
  - Les classes : Etat – Transition - StateMachine
  - SubState Machine
- GOAP

Critère d’évaluation

- Fonctionnement
- Performance
- Rigueur (C++)
- Clarté du code (n’oubliez pas les commentaires si besoin !)

C++ bonne pratique

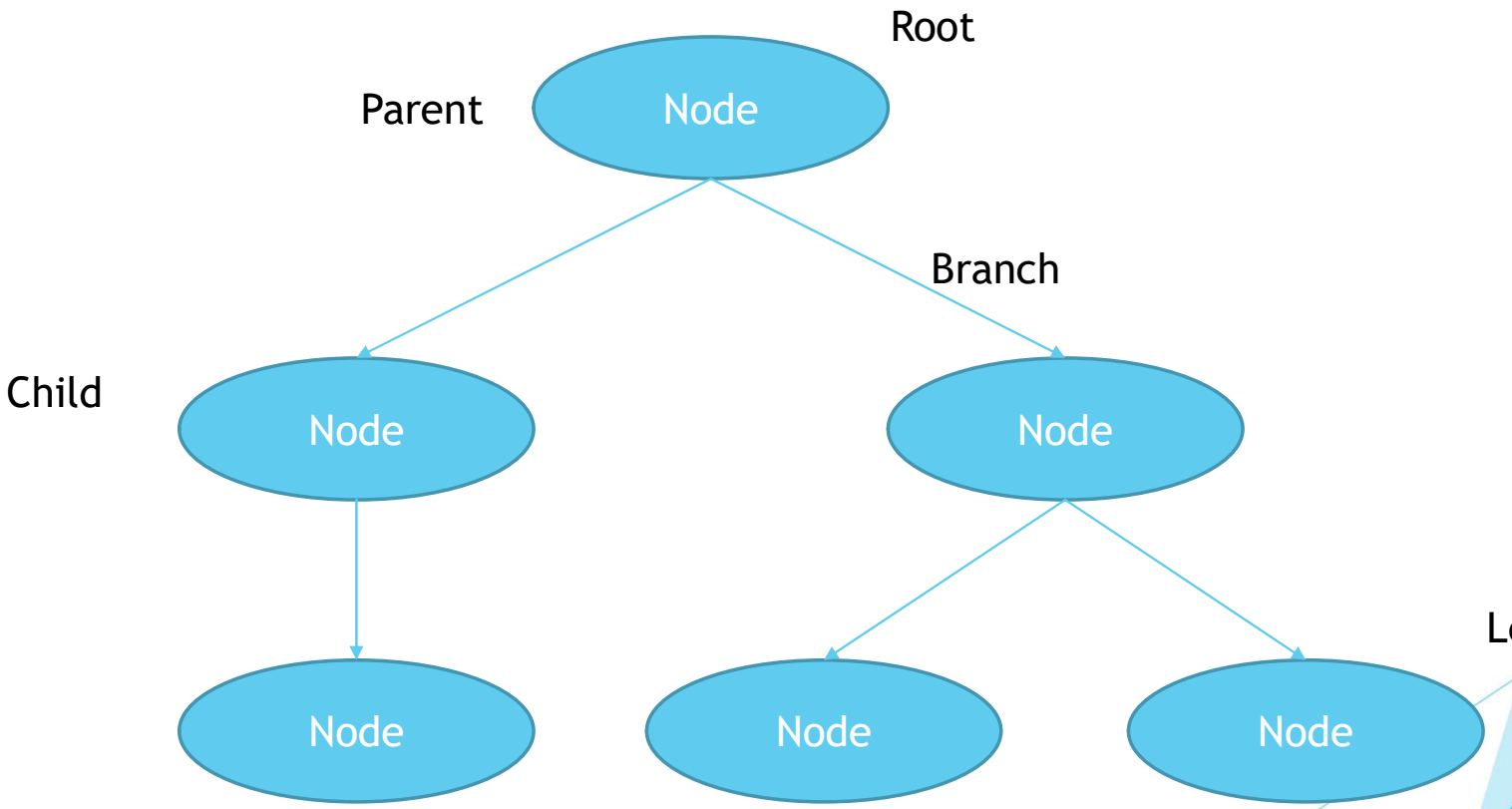
- Reserve un vecteur si on peut connaitre sa taille
- Utiliser Assert si l'on ne veut pas de certaines valeurs
  - Exemple : division par 0 assert(div > 0);
- Const , const et plein de const tout le temps
- Attention aux copies, passer par référence si possible
- Limité le nombre d'appelle de fonction
  - Notamment dans les boucles

C++ bonne pratique

- Alignement des bool
- Être tolérant dans l'intolérance (surtout si ECS)
- Penser à utiliser unsigned si nécessaire
- Nommage de fonction et de paramètre
- Faire attention aux new / delete

Question ?

Rappel : Tree


Root
Parent
Node
Branch
Child
Node
Node
Node
Node
Node
Leaf
Depth

Finite State Machine

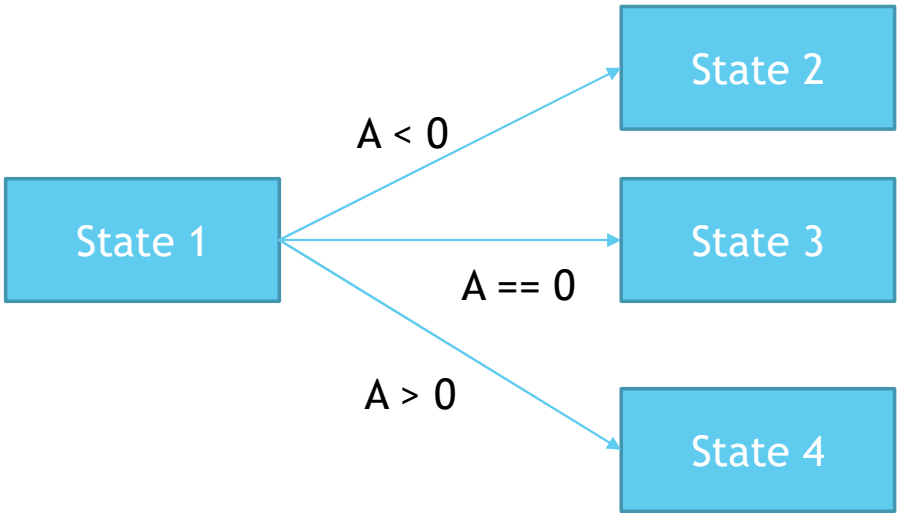
Machine à état fini – Automate fini

Utilisation

- Animation
- Machine à état de groupe
- Machine à état d’unité
- SD2 envoyer des ordres au gameplay

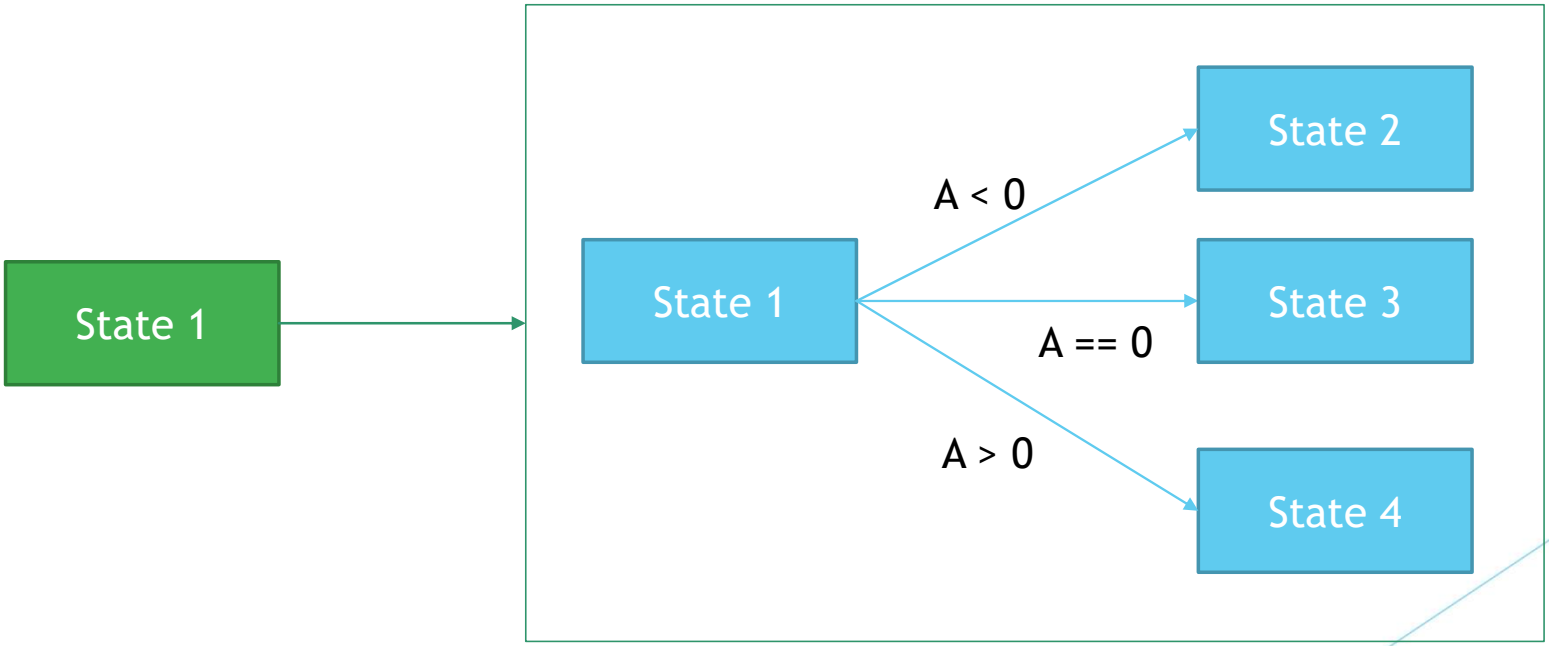
Finite State Machine

- Etat
- Condition



Finite State Machine

- Sub State Machine
  - Machine à état imbriqué



Classes

- State (virtual ou utiliser des lambdas)
- Transition (virtual ou utiliser des lambdas)
- StateMachine

Pseudo Code

```
void CreateStateMachine() { //Stocker tout ça dans un objet StateMachine
	State* start = new State("Start State"); //Ne pas oublier le destroy !
	State* end = new State("End State");
	Transition* transition = new Transition_IsTrue();
	start->AddTransition(transition, end);
}

void StateMachine::ProcessState() {
	State* currentState = this->GetCurrentState();
	for(int i=0; i < currentState->TransitionList.size(); ++i)
	{
		if (currentState->Transition[i].Process())
			ChangeState(currentState->Transition[i].GetEndState());
	}
}
```

Finite State Machine

- Pros
  - Simple à implémenter
  - Réutilisable
  - Facile à designer
  - Facilement compréhensible
- Cons
  - Peut devenir très gros
  - Difficilement réglable (si gros)
  - Difficilement représentable

Goap

Goal Oriented action Planning

GOAP

- Technique similaire au state machine
- Suite d’action et de condition
- Permet d’évaluer le meilleur chemin en fonction du contexte
- Utilisé dans F.E.A.R.
  - Permet au ennemies de s’adapter a la stratégie du joueur
  - Permet des techniques d’encerclement

GOAP - Exemple

- But : avoir 1000 bois

OU

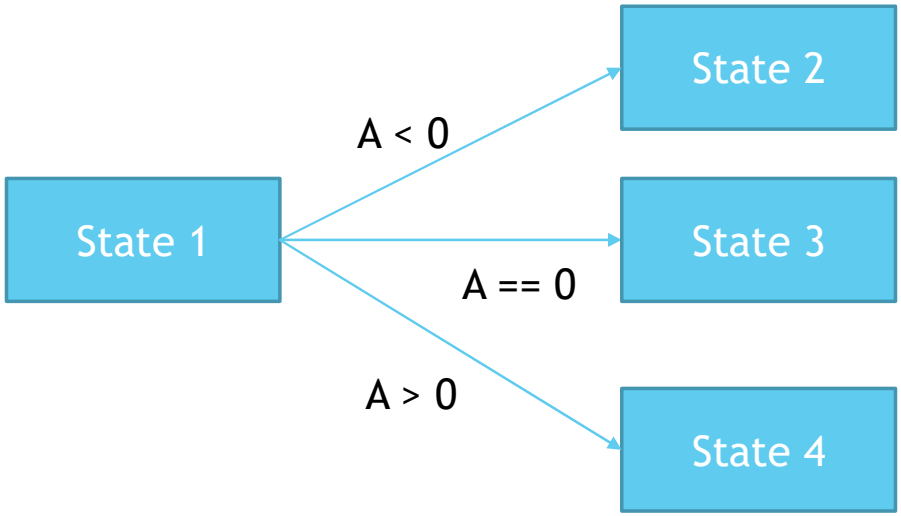
Requis : Envoyer des villageois couper du bois
Requis : Avoir des villageois
Requis : Produire des villageois
Requis : Avoir 100 de nourriture
Requis : Réassigner des villageois

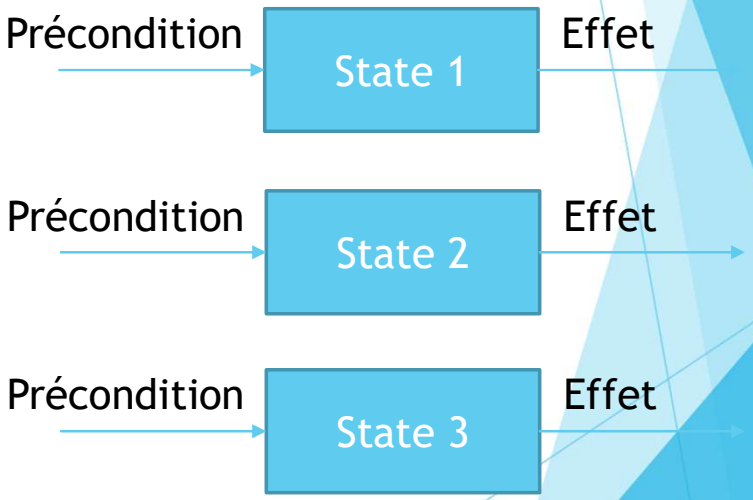
Requis : Envoyer des villageois couper du bois
Requis : Réassigner des villageois

GOAP

- Action
  - Créer des villageois
- Préconditions
  - Avoir de la nourriture
- Effets
  - Villageois possible à assigner à la récolte de bois

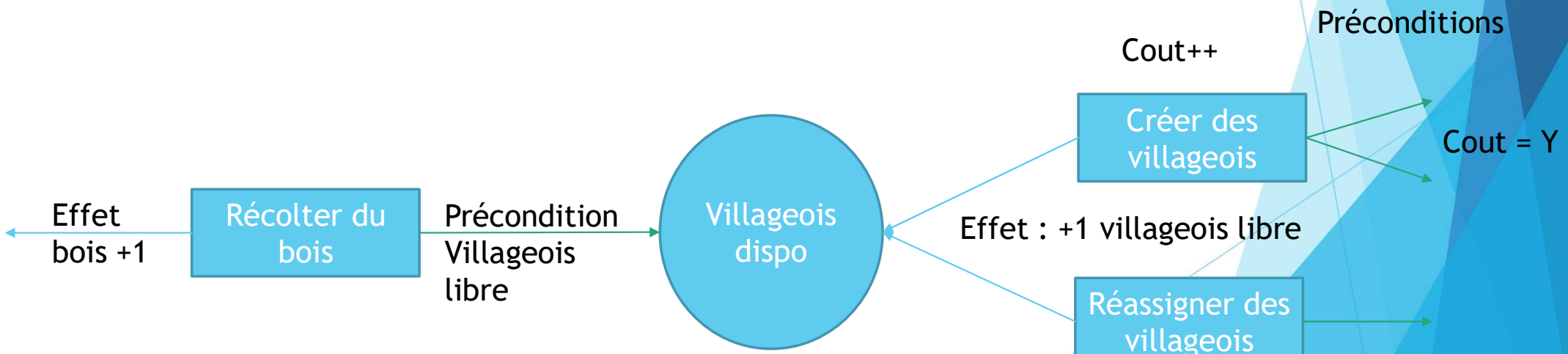
Différence StateMachine vs GOAP





GOAP Planner

- État actuel
- Évaluer les différents chemins et le moins cher



Pseudo Code

```
void CreateAction()
{
	Action* chopWoodAction = new Action("ChopWood");
	Action* createSbire = new Action("Créer des villageois");
	Action* reassignSbire = new Action("Réassigner des villageois");
	createSbire->AddEffect("Avoir des villageois disponible");
	reassignSbire->AddEffect("Avoir des villageois disponible");
	chopWoodAction >AddPrecondition("Avoir des villageois disponible");
}
```

GOAP

- Pros
  - Facile à implémenter
  - Plus lisible que les machines à état
  - Facile à designer
  - Des résultats souvent plus impressionnants (ex: F.E.A.R)
- Cons
  - Plus couteux, surtout sur un grand nombre d'unités
  - Plus long à paramétrer et donc plus d'erreurs humaines