Travail Personnel Approfondi: CoreWar

Chaid Akacem Jasmine Baudin Hugo

Université de Caen

21 avril 2019

- CoreWar
- Développer une machine virtuelle
- Construire des programmes performants

Deux classes pour effectuer la lecture

Classe `Reader`

- Lecture basique d'un fichier texte
- Pré-traitement

Classe `Parser`

- Création d'un programme exécutable
- Découpage d'une ligne
- Vérification
- Conversion

- Découpage de la ligne
- Traitement différencié selon opérateur

Figure – Schéma général d'une ligne RedCode

Exemple traitement différencié

La ligne `ADD #0 1` est orienté vers le traitement par défaut.
La ligne `JMP 5` est orienté vers le traitement pour les opérateurs à un champ.

Les instructions

- Instruction : Classe abstraite
- Trois méthodes principales

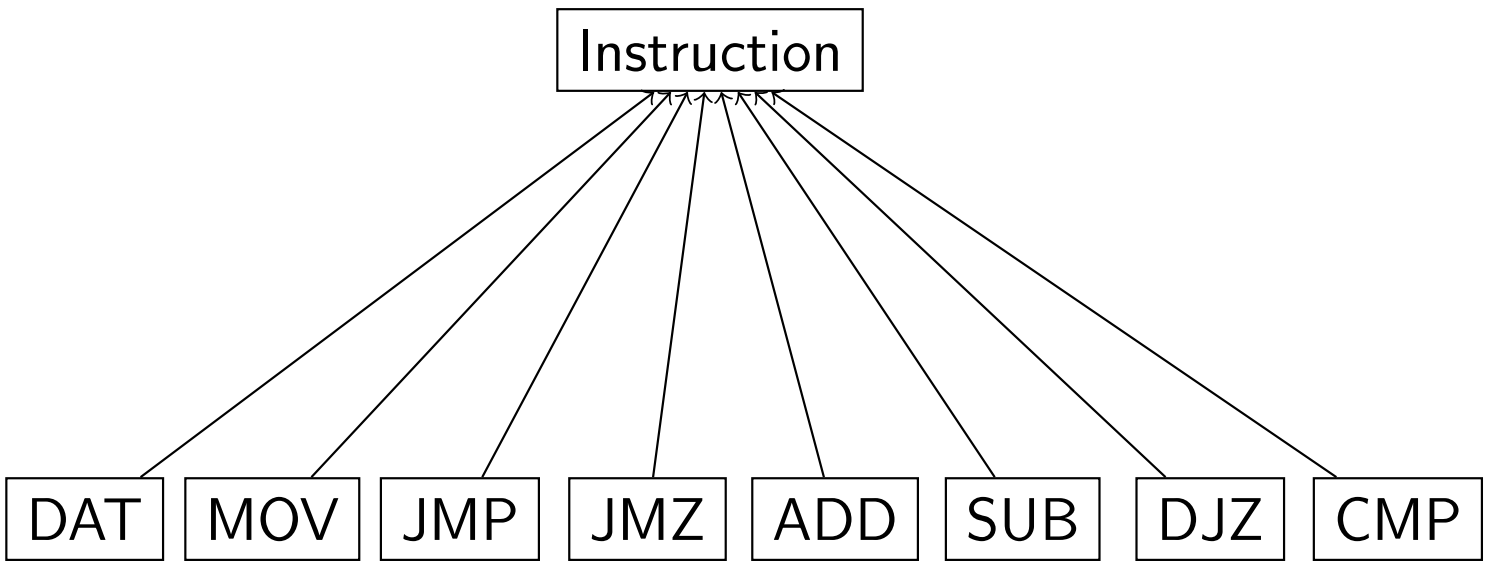

Figure – Héritage de l'instruction

Une gestion des erreurs pendant la lecture

Erreurs de syntaxe

- Ligne non conforme
- Opérateur/Mode d'adressage non reconnu
- Adresse qui n'est pas un nombre

Instructions non exécutables

- Ligne syntaxiquement correcte
- Nécessite qu'un objet Instruction soit instancié
- Bloquée avant son exécution

Exemple Instruction non exécutable

La ligne `ADD 0 1` n'est pas exécutable
La ligne `DAT 0` n'est pas exécutable (Raison de *cohérence*)

La partie CoreWar : deux objets principaux

La classe `VirtualMachine`

- Machine virtuelle du logiciel
- Exécution
- Stockage
- Insertion des programmes

La classe `Warrior`

- Un programme à exécuter

Deux initialisations de la mémoire

- Avec des données (Ligne `DAT #0`)
- Avec les deux programmes qui vont se combattre
  1. Placement du pointeur du premier programme
  2. Placement du premier programme
  3. Placement du pointeur du second programme selon les contraintes
  4. Placement du second programme

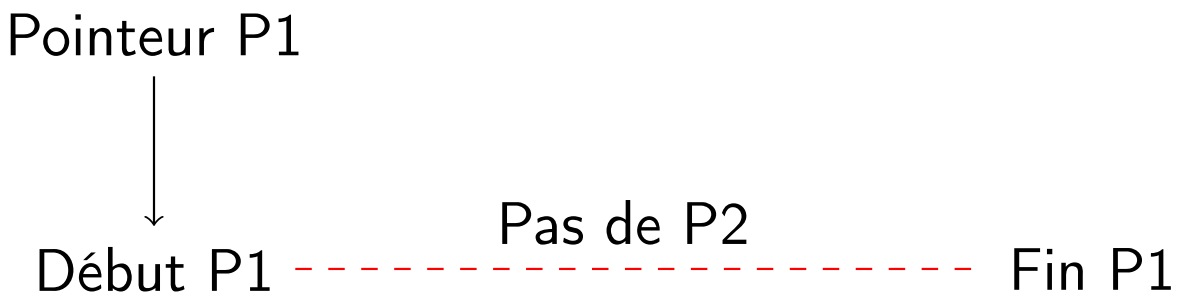

Figure – Contraintes de placement

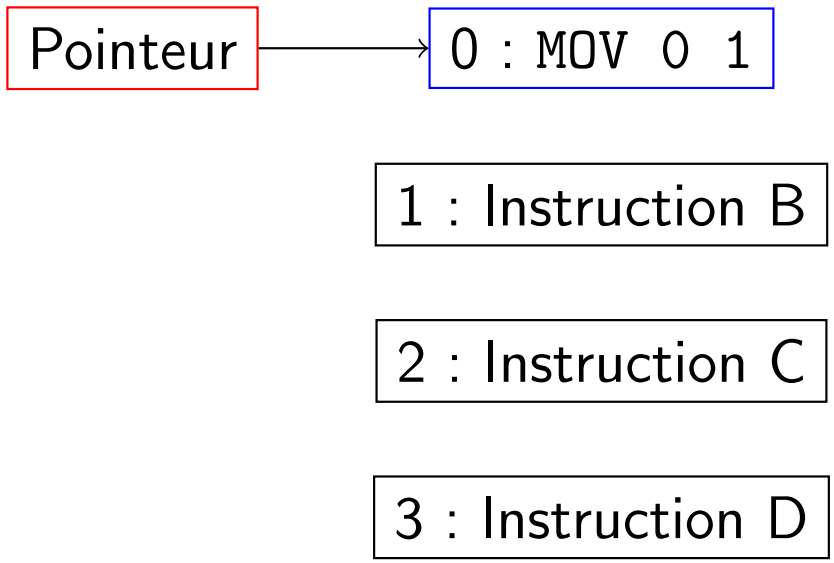

Figure – Boucle de jeu

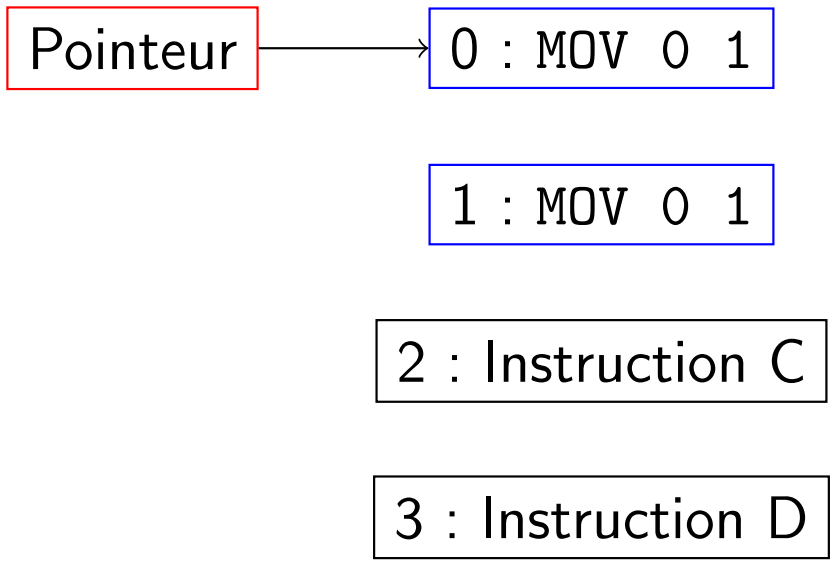

Figure – Boucle de jeu

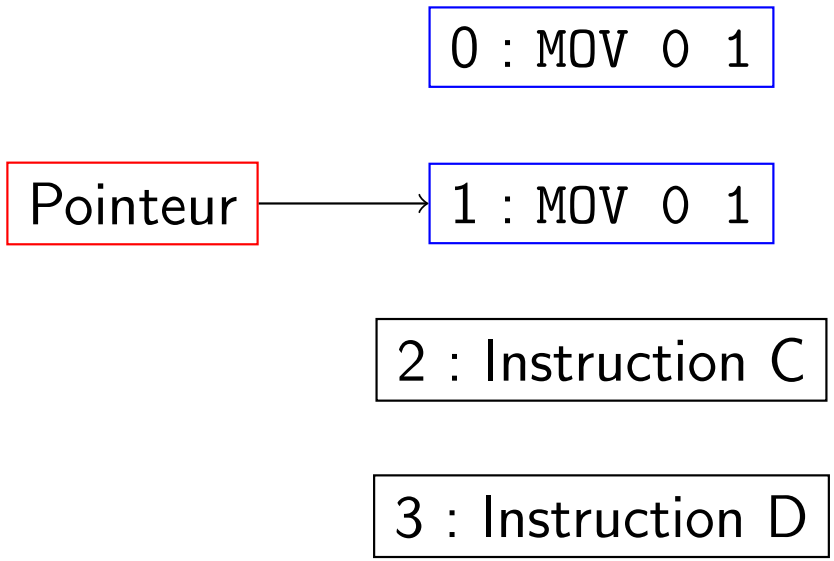

Figure – Boucle de jeu

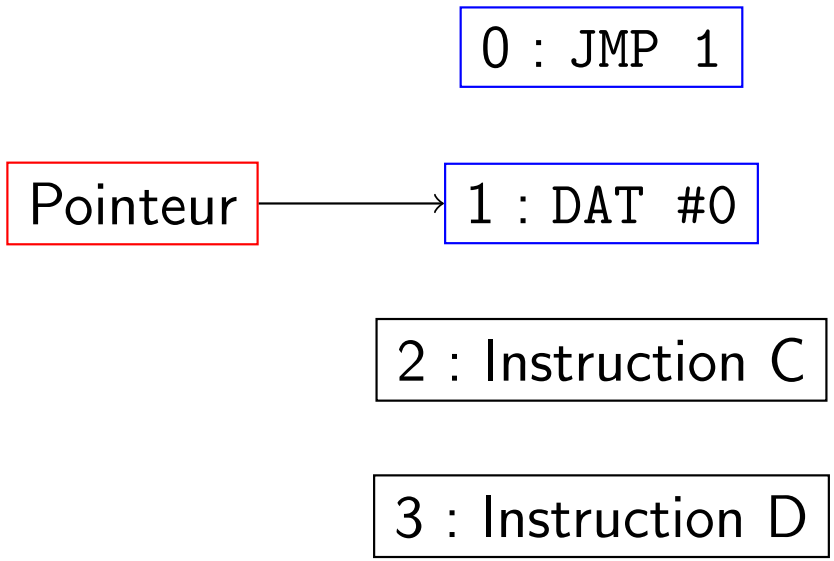

Figure – Boucle de jeu

- Créer un programme performant
- Analogue à la biologie
  - Sélection
  - Mutations
  - Croisement des individus sélectionnés
- Début aléatoire

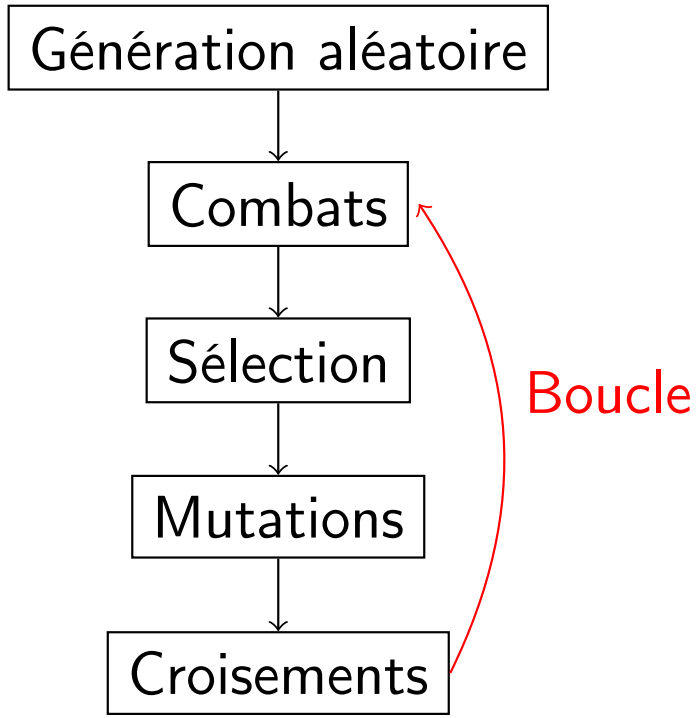

Figure – Déroulement de l'algorithme

Partie Combat

- Méthode `fight`
- Adaptation au problème
- Statistiques sur les Warriors

Notre implémentation : *Tournoi*

- Complexité raisonnable
- Les meilleurs confrontés aux meilleurs

Partie Sélection

- Méthode `doSelection`
- Garder la meilleure partie de la population
- A partir des statistiques générées

Notre implémentation : *Les 20 meilleurs*

- Adapté à notre implémentation du combat
- Complexité d'un algorithme de tri

Partie Evolution

- Méthode `mutAllWarriors`
- Outils de base
- Réutilisation avec un seul outil, muni des probabilités
- Intervient sur les programmes

Notre implémentation

Outils de base

- Remplacement d'une ligne
- Délétion d'une ligne
- Rajout d'une ligne

Le tout encapsulé dans une classe qui va utiliser les trois outils successivement.

Partie Croisement

- Méthode `cross` et `crossAll`
- Nombre de croisements définis
- Prend les instructions des meilleurs programmes
- Créé de nouveaux programmes

Notre implémentation

- Sélection aléatoire de deux programmes
- Sélection aléatoire de ligne

Conclusion

- Un CoreWar complétement orienté objet
- Une gestion des erreurs avant exécution
- Une génération de programme modulable selon les besoins