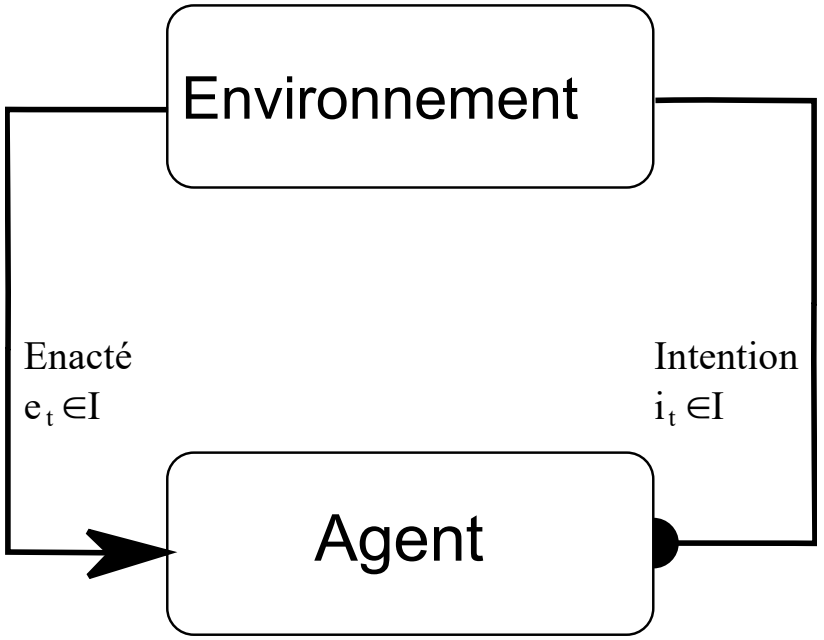
Cours-TD d'introduction à l'Intelligence Artificielle

Partie VIII

L'apprentissage Développemental

Simon Gay

Introduction à l'Intelligence Artificielle

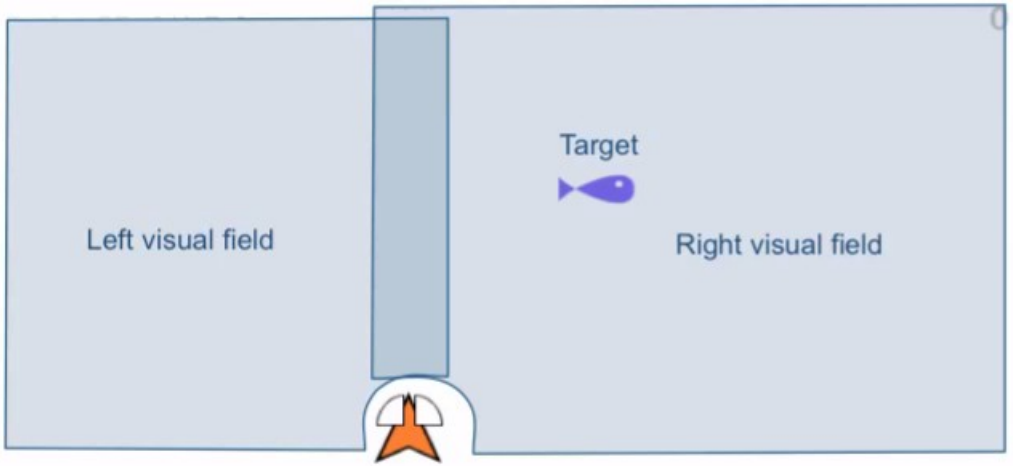


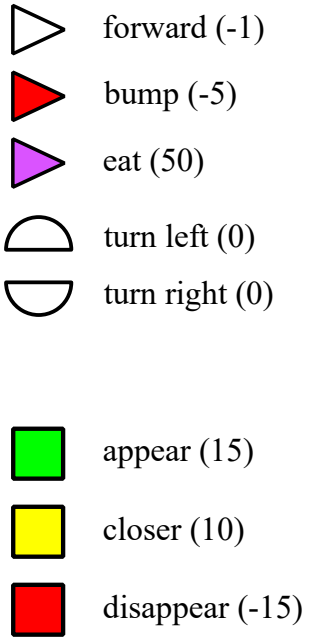
Passons à la pratique !

Introduction à l'Intelligence Artificielle

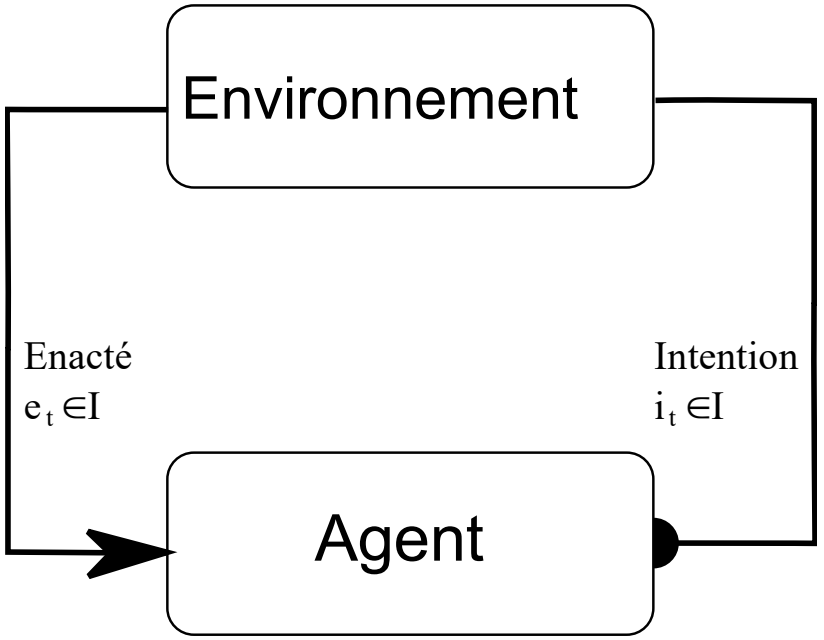
- **Sujet d'étude**
  - Un agent chasseur
    - Deux champs de visions (trois régions) → apparition, rapprochement, disparition
    - 5 interactions primitives,
    - 3 événements et 3 régions possibles pour forward, turn left et turn right

      → total de 32 interactions (5 primitives + 3 x 3 x 3 interactions additionnelles)





Introduction à l'Intelligence Artificielle



Commençons à coder !

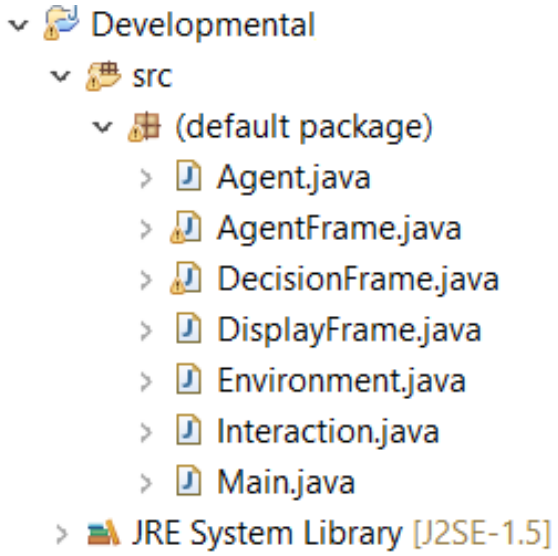
Introduction à l'Intelligence Artificielle

Les parties purement 'algorithmique' ou 'structure' peuvent être copiées depuis ce support

Essayez quand même de comprendre ce que vous copiez !

Introduction à l'Intelligence Artificielle

- **L'environnement de travail**
  - Ouvrez Eclipse
  - Créez un nouveau projet Java 'developmental'
    - Clic droit dans package explorer → new → Java project
  - Télécharger l'archive developmental.zip sur le github, puis importez les 7 classes java dans votre nouveau projet.



Introduction à l'Intelligence Artificielle

- **L'environnement de travail**
  - Étudions les classes à notre disposition :

    - Main : classe principale, contient la fonction main( ), qui exécute la boucle d'échange entre l'agent et l'environnement. → rien à faire

```
while (true){

    // gestion de la pause et du pas-à-pas
    while (env.pause && !env.step){
        try {Thread.sleep(20);}
        catch (InterruptedException e) {e.printStackTrace();}
    }
    env.step=false;

    intended=agent.intention();
    System.out.print(intended.name);

    enacted=list[env.intend(intended)];
    System.out.println(" -> "+enacted.name);
    agent.setResult(enacted);

    display.repaint();

    try {Thread.sleep(50);}
    catch (InterruptedException e) {e.printStackTrace();}
}
```

Introduction à l'Intelligence Artificielle

- **L'environnement de travail**
  - Étudions les classes à notre disposition :

    - Main : définit également la liste des interactions

      → rien à faire !

```
Environment env=new Environment();

// define list of interactions
Interaction[] list=new Interaction[32];
list[ 0]=new Interaction(">  " ,0);          list[12]=new Interaction("^  ",-1);
list[ 1]=new Interaction("X  ",-10);         list[13]=new Interaction("v  ",-1);
list[ 2]=new Interaction("E  ",50);

list[ 3]=new Interaction(">+C",5);           list[14]=new Interaction("^+C",5);        list[23]=new Interaction("v+C",5);
list[ 4]=new Interaction(">+L",5);           list[15]=new Interaction("^+L",5);        list[24]=new Interaction("v+L",5);
list[ 5]=new Interaction(">+R",5);           list[16]=new Interaction("^+R",5);        list[25]=new Interaction("v+R",5);

list[ 6]=new Interaction(">*C",5);           list[17]=new Interaction("^*C",0);        list[26]=new Interaction("v*C",0);
list[ 7]=new Interaction(">*L",5);           list[18]=new Interaction("^*L",0);        list[27]=new Interaction("v*L",0);
list[ 8]=new Interaction(">*R",5);           list[19]=new Interaction("^*R",0);        list[28]=new Interaction("v*R",0);

list[ 9]=new Interaction(">-C",-5);          list[20]=new Interaction("^-C",-5);       list[29]=new Interaction("v-C",-5);
list[10]=new Interaction(">-L",-5);          list[21]=new Interaction("^-L",-5);       list[30]=new Interaction("v-L",-5);
list[11]=new Interaction(">-R",-5);          list[22]=new Interaction("^-R",-5);       list[31]=new Interaction("v-R",-5);
```

Introduction à l'Intelligence Artificielle

- **L'environnement de travail**
  - Étudions les classes à notre disposition :

    - Interaction : classe définissant une interaction entre agent et environnement. Pour l'instant, une interaction est définie par un nom et une valeur de satisfaction. Une fonction de comparaison et de recherche vous sont fournies.

      → adapter pour pouvoir utiliser des schemes sensorimoteur

    - Agent : classe contenant les mécanismes d'apprentissage et décisionnel. Pour l'instant, pas d'apprentissage et sélection aléatoire

      → implémenter les mécanismes de décision

```
// get intended primary interaction
public Interaction intention(){

    int rand=(int)(Math.random()*interList.size());

    return interList.get(rand);
}
```

Introduction à l'Intelligence Artificielle

- **L'environnement de travail**

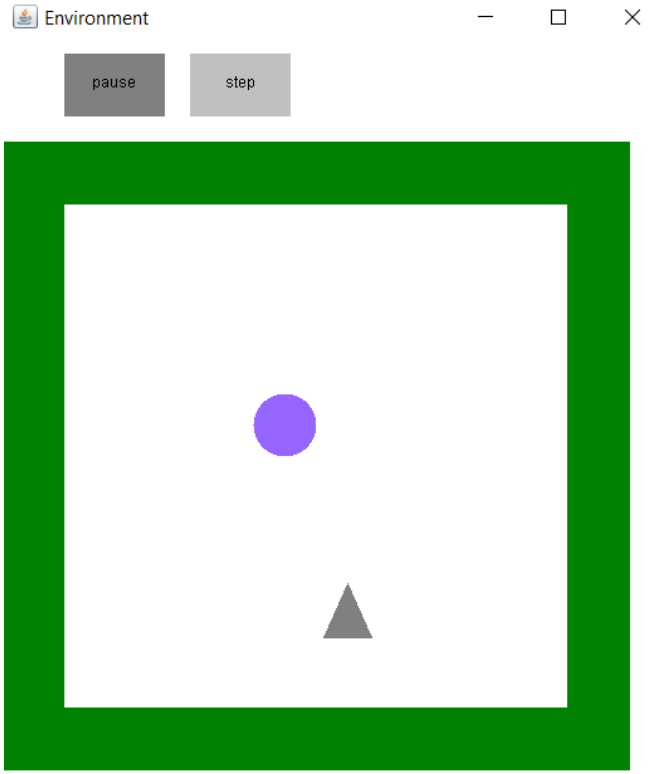
  - Classe Agent : pour l'instant, elle contient une timeline pour stocker les interactions énactées. Le constructeur copie la liste des interactions primitives dans la liste des schemes

```
public ArrayList<Interaction> interList;

public ArrayList<Interaction> timeline;


public Agent(Interaction[] list){

    interList=new ArrayList<Interaction>();
    for (int i=0;i<list.length;i++){
        interList.add(list[i]);
    }

    timeline=new ArrayList<Interaction>();


public void setResult(Interaction enacted){

    // add to timeline
    timeline.add(0,enacted);
    while (timeline.size()>10) timeline.remove(timeline.size()-1);
```

Introduction à l'Intelligence Artificielle

- **L'environnement de travail**
  - Étudions les classes à notre disposition :

    - DisplayFrame : classe pour l'afficheur de l'environnement. Gère les boutons de fonction et l'interface utilisateur

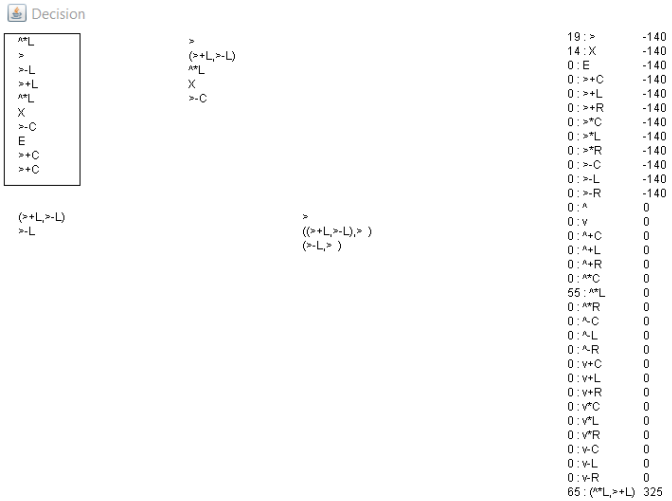


Introduction à l'Intelligence Artificielle

- **L'environnement de travail**
  - Étudions les classes à notre disposition :

    - AgentFrame : affiche la liste des interactions disponibles, code fourni, à ré-activer quand les schemes seront fonctionnels

    - DecisionFrame : affiche les timeslines, scopes et propositions du système décisionnel. Code fourni, à réactiver progressivement



Introduction à l'Intelligence Artificielle

- **Les schemes**

  - Il faut définir une même classe pour les interaction primitives et les schemes composites
  - Une solution :
    - Une classe Interaction pour les interactions primitives
    - Une classe Composite qui hérite de Interaction pour les schemes

      → La classe Interaction doit contenir tous les attributs et fonctions des deux types d'interactions

→ Créez une classe Composite qui hérite de Interaction :

```
public class Composite extends Interaction{

    public Composite(Interaction pre, Interaction post) {
        super("("+pre.name+","+post.name+")", pre.valence+post.valence);
    }

}
```

Introduction à l'Intelligence Artificielle

- **Les schemes**

  - Nous allons adapter la classe Interaction pour pouvoir utiliser des schemes :

```
public class Interaction {

    public String name="";
    public int valence=0;

    public int enactions=1;   // count enactions

    public boolean primary=true;
    public int length=1;

    public Interaction pre=null;
    public Interaction post=null;
```

Introduction à l'Intelligence Artificielle

- **Les schemes**

  - On ajoute les fonctions pour manipuler les sous-séquences :

```
public Interaction(String n, int val){
     name=n;
     valence=val;
}

public boolean isActive(ArrayList<Interaction> scope){ // context is in scope
     return true;
}

public Interaction getProposition(){  // proposition part of the scheme
     return this;
}

public int getPropositionValue(){     // utility value of the proposition
     return 0;
}

public Interaction getNextPrimary(){  // next primitive of the scheme
     return this;
}
```

Introduction à l'Intelligence Artificielle

- **Les schemes**

  - On ajoute les fonctions équivalentes pour les composites :

```
public Composite(Interaction pre, Interaction post) {
      super("("+pre.name+","+post.name+")", pre.valence+post.valence);

      this.pre=pre;
      this.post=post;

      primary=false;
      length=pre.length+post.length;
}

public boolean isActive(ArrayList<Interaction> scope){// search scheme in scope list
      boolean found=false;
      int i=0;
      while (!found && i<scope.size()){
            if (pre.isEqual(scope.get(i))) found=true;
            else i++;
      }
      return found;
}

public Interaction getProposition(){    // get proposition part of the scheme
      return post;
}
```

Introduction à l'Intelligence Artificielle

- **Les schemes**

  - Pour *getPropositionValue*, il faut utiliser le nombre d'enaction du scheme

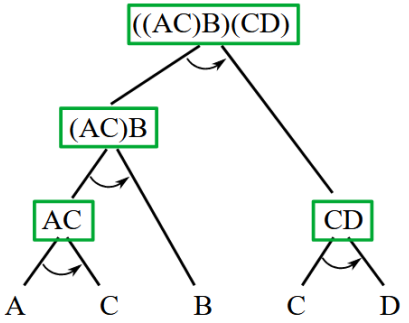
  - Dans Composite, nous pouvons calculer la valeur de la proposition :

```
public Interaction getProposition(){  // get proposition part of the scheme
    return post;
}

public int getPropositionValue(){
    return post.valence*this.enactions;
}
```

Introduction à l'Intelligence Artificielle

- **Les schemes**

  - Pour *getNextPrimary,* c'est plus compliqué : il faut un moyen de progresser dans l'exécution du scheme.
    - Rappel : l'exécution se fait récursivement



  - Nous allons utiliser une variable pour indiquer l'avancée dans le scheme :
    - 0 → initialisation / partie contexte
    - 1 → partie contexte terminée / partie proposition
    - 2 → parie proposition terminée / scheme énacté avec succès
    - -1 → échec du scheme

Introduction à l'Intelligence Artificielle

- **Les schemes**

  - On ajoute dans *Interaction* une variable pour indiquer l'avancement :

```
public Interaction pre=null;
public Interaction post=null;

public int status=0; // 0= pre, 1= post, 2= completed, -1= failed
```

  - Puis on ajoute la version *Composite* de la fonction *getNextPrimary*

```
public int getPropositionValue(){
    return post.valence*this.enactions;
}

public Interaction getNextPrimary(){
    if (status==0){ // if scheme is enacting context
        return pre.getNextPrimary();
    }
    else{           // if scheme is enacting proposition
        return post.getNextPrimary();
    }
}
```

Introduction à l'Intelligence Artificielle

- **Les schemes**

  - On doit maintenant gérer la mise à jour et la mise à zéro du statut :
    - Pour les interactions primitives, c'est simple : si l'interaction enactée est la même, le statut passe à 2, sinon, il passe à -1, et on ajoute à la liste si elle n'y est pas déjà

```
public void step(Interaction enacted, ArrayList<Interaction> list){

     if (this.isEqual(enacted)){
          status=2;
     }
     else{
          status=-1;

          // set alternative interaction in global list
          int id=enacted.isIncluded(list);
          if (id==-1){
               id=list.size();
               list.add(enacted);
          }
     }
}

public void reset(){     // reset status
     status=0;
}
```

Introduction à l'Intelligence Artificielle

- **Les schemes**

  - Avec les Composites, il y a plus de cas à gérer :

```
public void step(Interaction enacted, ArrayList<Interaction> list){

    if (status==0){ // enaction of context part
        pre.step(enacted, list); // step forward in sub-scheme

        if (pre.status==2){ // if pre-sequence is completed,
            status=1;       // move to post-sequence
            post.reset(); // prepare post-sequence for enaction
        }
        else if (pre.status==-1){    // if pre-sequence failed,
            status=-1;               // scheme enaction is a failure
        }
    }
    else{           // enaction of proposition part
        post.step(enacted, list); // step forward in sub-scheme

        if (post.status==2){     // if post-sequence is completed,
            status=2;            // scheme is completed
        }
        else if (post.status==-1){   // if post-sequence failed,
            status=-1;               // scheme enaction is a failure
        }
    }
}
```

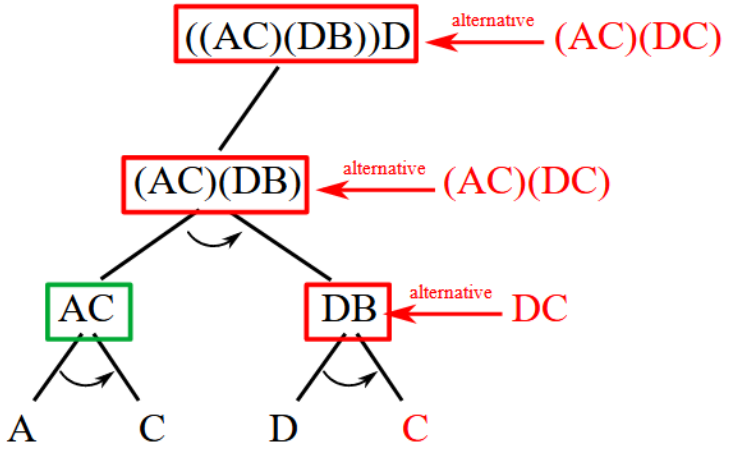
Ne pas oublier :

```
public void reset(){
    status=0;
    pre.reset();
    post.reset();
}
```

Introduction à l'Intelligence Artificielle

- **Les schemes (on y est presque)**

  - Il faut gérer les schemes alternatifs en cas d'échec
    - Construction récursive des schemes alternatifs



  - Pour chaque scheme :
    - Une liste des schèmes alternatifs
    - Un pointeur pour mémoriser le scheme alternatif actuel (pour le niveau 'au dessus')

Introduction à l'Intelligence Artificielle

- **Les schemes (on y est presque)**

  - Dans *Interaction*, on ajoute le pointeur et la liste d'alternatives

```
public int status=0; // 0= pre, 1= post, 2= completed, -1= failed

public Interaction alternativeScheme=null;
public ArrayList<Interaction> alternativeList;
```

  - et on initialise la liste dans le constructeur

```
public Interaction(String n, int val){
    name=n;
    valence=val;

    alternativeList=new ArrayList<Interaction>();
}
```

Introduction à l'Intelligence Artificielle

- **Les schemes (on y est presque)**

  - Pour les Interactions primitives : dans la fonction *step*, on enregistre simplement, en cas d'échec, l'interaction énactée. On l'ajoute à la liste si elle n'y est pas déjà.

```
if (this.isEqual(enacted)){
    status=2;
}
else{
    status=-1;

    // set alternative interaction in global list
    int id=enacted.isIncluded(list);
    if (id==-1){
        id=list.size();
        list.add(enacted);
    }

    // record the alternative scheme
    alternativeScheme=list.get(id);

    // add alternative to list
    id=alternativeScheme.isIncluded(alternativeList);
    if (id==-1) alternativeList.add(alternativeScheme);
}
```

Introduction à l'Intelligence Artificielle

- **Les schemes (on y est presque)**

  - Pour les Composites, le pointeur et la liste sont hérités. On traite en premier le cas d'un échec de la partie contexte : on récupère l'alternative de ce même contexte, et on l'ajoute à la liste d'alternatives (si il n'est pas déjà présent)

    → Ce scheme alternatif a déjà été enregistré dans la liste globale par le sous-scheme contexte

```
if (pre.status==2){ // if pre-sequence is completed,
    status=1;       // move to post-sequence
    post.reset(); // prepare post-sequence for enaction
}
else if (pre.status==-1){   // if pre-sequence failed,
    status=-1;              // scheme enaction is a failure

    // record the alternative scheme from pre-sequence current
    // alternative (already exists in global list)
    alternativeScheme=list.get(pre.alternativeScheme.isIncluded(list));

    // add alternative to list
    int id=alternativeScheme.isIncluded(alternativeList);
    if (id==-1) alternativeList.add(alternativeScheme);
}
```

Introduction à l'Intelligence Artificielle

- **Les schemes (on y est presque)**

  - Si c'est la partie proposition qui échoue, on construit l'alternative à partir du contexte et de l'alternative de la proposition

    → il faut ajouter ce scheme alternatif dans la liste globale (si inconnu)

```
else if (post.status==-1){  // if post-sequence failed,
    status=-1;              // scheme enaction is a failure

    // construct the alternative scheme and add it to alternative list
    alternativeScheme=new Composite(this.pre, post.alternativeScheme);

    // add created alternative (if needed)
    int id=alternativeScheme.isIncluded(list);
    if (id==-1){
        id=list.size();
        list.add(alternativeScheme);
    }

    // record the alternative scheme
    alternativeScheme=list.get(id);

    // add alternative to list
    id=alternativeScheme.isIncluded(alternativeList);
    if (id==-1) alternativeList.add(alternativeScheme);
}
```

Introduction à l’Intelligence Artificielle

- **Les schemes**

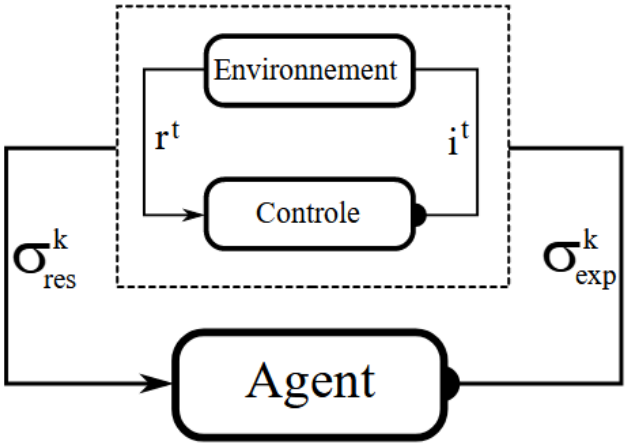
  - Les classes Interaction et Composite sont désormais prêtes pour l’utilisation
    - Vous pouvez ré-activer le code de l’afficheur AgentFrame pour afficher l’ensemble des schemes enregistrés par l’agent.

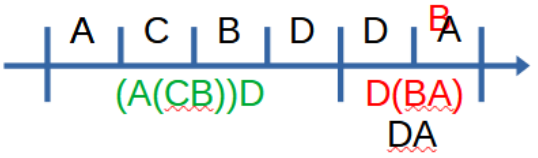
  - Passons à l’agent et à son système décisionnel !

Introduction à l'Intelligence Artificielle

- **Le système décisionnel**

  - Nous allons préparer le cycle de décision à deux niveaux





  - Il nous faut :
    - Une deuxième timeline
    - Une fonction intermédiaire pour la décision

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : une seconde timeline**
  - Dans Agent, on ajoute et initialise un deuxième ArrayList d'interactions :

```
public ArrayList<Interaction> timeline;
public ArrayList<Interaction> enactionLine;
```

  - Dans le constructeur :

```
timeline=new ArrayList<Interaction>();
enactionLine=new ArrayList<Interaction>();
```

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : une double décision**

  - Le principe est le suivant :
    - tant que le scheme n'est pas terminé, le contrôleur (décision 1) effectue les interactions primitives.
    - Quand le scheme se termine, en succès ou en échec, le contrôleur transmet le scheme effectué à l'agent, qui intègre le résultat.
    - Le contrôleur demande ensuite un nouveau scheme à l'agent (décision 2)

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : une double décision**

  - Nous allons intégrer le contenu de la fonction intention dans une nouvelle fonction :

```
// decision of the agent
private void decision(){

     int rand=(int)(Math.random()*interList.size());

     intendedScheme=interList.get(rand);
}

// get intended primary interaction
public Interaction intention(){
     if (intendedScheme==null){
          decision();
     }
     return intendedScheme.getNextPrimary();
}
```

  - À noter que pour le moment, il n'y a pas de changement

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : une double décision**

  - De même, on récupérera le résultat dans une double fonction, l'une pour intégrer le résultat de l'interaction primitive, l'autre pour le scheme complet

```
private void learn(Interaction enacted){

    if (intendedScheme.isEqual(enacted)) System.out.println("scheme is a success");
    else System.out.println("scheme is a failure");

    enactedScheme=enacted;    // record enacted scheme

    // add to scheme line
    enactionLine.add(0,enacted);
    while (enactionLine.size()>5) enactionLine.remove(enactionLine.size()-1);

    // remove intended scheme to trigger new decision
    intendedScheme=null;
}

////////////////////////////////////////////////////////////////////
public void setResult(Interaction enacted){
```

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : une double décision**

```
////////////////////////////////////////////////////////////
public void setResult(Interaction enacted){

    // add to timeline
    timeline.add(0,enacted);
    while (timeline.size()>10) timeline.remove(timeline.size()-1);

    // move forward in the intended scheme
    intendedScheme.step(enacted, interList);


    if (intendedScheme.status==2){ // case of a success
        learn(intendedScheme);
    }
    else if (intendedScheme.status==-1){// case of failure
        learn(intendedScheme.alternativeScheme);
    }

    displayList.repaint();
    displayDecision.repaint();
}
```

  - À noter : vous pouvez activer l'affichage de la seconde timeline dans *DecisionFrame*

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : les scopes**

  - Les scopes sont des ArrayList d'Interaction
    - Ils sont mis à jour à la fin de la fonction *learn*
    - Le second scope est une copie du premier

```
private Interaction intendedScheme=null;
private Interaction enactedScheme=null;

public ArrayList<Interaction> scope1;
public ArrayList<Interaction> scope2;
```

  - Dans le constructeur :

```
timeline=new ArrayList<Interaction>();
enactionLine=new ArrayList<Interaction>();

scope1=new ArrayList<Interaction>();
scope2=new ArrayList<Interaction>();
```

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : les scopes**
  - Les scopes sont des ArrayList d'Interaction
    - Ils sont mis à jour à la fin de la fonction *learn*
    - Le second scope est une copie du premier
  - À la fin de *learn*

```
// define new scopes
scope2.clear();
for (int i=0;i<scope1.size();i++) scope2.add(scope1.get(i)); // copy previous scope

scope1.clear();  // prepare new scope
scope1.add(enactedScheme);
if (!enactedScheme.primary) scope1.add(enactedScheme.post);


// remove intended scheme to trigger new decision
intendedScheme=null;
```

  - À noter : il manque les schemes mis à jour (on va y revenir). Vous pouvez également activer l'affichage des scopes dans *DecisionFrame*

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : Un premier mécanisme d'apprentissage**

  - Nous allons implémenter le premier mécanisme d'apprentissage (dans *learn)* : la construction de schemes à partir du contenu du scope actuel et du scheme enacté
    - On définira une liste des schemes mis à jour pour les ajouter au scope

```
enactionLine.add(0,enacted);
while (enactionLine.size()>5) enactionLine.remove(enactionLine.size()-1);

// learn/update schemes
enactedScheme.enactions++;    // increment enaction count
ArrayList<Integer> updated=new ArrayList<Integer>(); // get list of updated schemes

// first mechanism
for (int i=0;i<scope1.size();i++){
    Interaction temp=new Composite(scope1.get(i),enactedScheme);

    if (temp.length<10){

        int id=temp.isIncluded(interList);
        if (id==-1) interList.add(temp);
        else{
            interList.get(id).enactions++;
            if (interList.get(id).enactions>treshold) updated.add(id);
        }
    }
}
```

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : Un premier mécanisme d'apprentissage**

  - On n'oublie pas d'ajouter les schemes mis à jour dans les scopes :

```
scope1.clear();  // prepare new scope
scope1.add(enactedScheme);
if (!enactedScheme.primary) scope1.add(enactedScheme.post);

for (int i=0;i<updated.size();i++){
    scope1.add(interList.get(updated.get(i)));
}

// remove intended scheme to trigger new decision
intendedScheme=null;
```

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : Un second mécanisme d'apprentissage**

  - Le second mécanisme fait intervenir le stream scheme, constitué des deux derniers schemes énactés, et utilise le scope précédent (scope 2)

  - Ajoutez un pointeur d'Interaction pour stocker le *stream sheme* :

```
private Interaction intendedScheme=null;
private Interaction enactedScheme=null;

public Interaction streamScheme=null;
```

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : Un second mécanisme d'apprentissage**

  - Le second mécanisme fait intervenir le stream scheme, constitué des deux derniers schemes énactés, et utilise le scope précédent (scope 2)

  - Juste après le premier mécanisme : (1/2)

```
// second mechanism
if (enactionLine.size()>=2){
    // define stream scheme
    Interaction temp=new Composite(enactionLine.get(1), enactionLine.get(0));

    if (temp.length<10){ // limit size of schemes

        // search for existing scheme
        int id=temp.isIncluded(interList);
        if (id==-1){
            interList.add(temp);
            streamScheme=interList.get(interList.size()-1);
        }
        else streamScheme=interList.get(id);

    ...
```

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : Un second mécanisme d'apprentissage**

  - Le second mécanisme fait intervenir le stream scheme, constitué des deux derniers schemes énactés, et utilise le scope précédent (scope 2)

  - Juste après le premier mécanisme : (2/2)

```
            else streamScheme=interList.get(id);

            if (streamScheme.enactions>treshold){
                // create new schemes with scope2 and stream scheme
                for (int i=0;i<scope2.size();i++){

                    temp=new Composite(scope2.get(i),streamScheme);

                    id=temp.isIncluded(interList);
                    if (id==-1) interList.add(temp);
                    else{
                        interList.get(id).enactions++;
                        if (interList.get(id).enactions>treshold) updated.add(id);
                    }
                }
            }
        }
    }
```

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : Un second mécanisme d'apprentissage**

  - Le second mécanisme fait intervenir le stream scheme, constitué des deux derniers schemes énactés, et utilise le scope précédent (scope 2)

  - Juste après le premier mécanisme : (2/2)

```
            else streamScheme=interList.get(id);

            if (streamScheme.enactions>treshold){
                // create new schemes with scope2 and stream scheme
                for (int i=0;i<scope2.size();i++){

                    temp=new Composite(scope2.get(i),streamScheme);

                    id=temp.isIncluded(interList);
                    if (id==-1) interList.add(temp);
                    else{
                        interList.get(id).enactions++;
                        if (interList.get(id).enactions>treshold) updated.add(id);
                    }
                }
            }
        }
    }
```

Introduction à l’Intelligence Artificielle

- **Le système décisionnel : Un second mécanisme d’apprentissage**
  - Le mécanisme d’apprentissage est maintenant complet.
  - Cependant, pour être un vrai agent développemental, l’agent doit être dans le contrôle de son propre apprentissage : il faut maintenant lui permettre de sélectionner son action pour fermer la boucle d’interaction

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : la sélection de l'action**

  - La sélection de l'action va consister à proposer des schemes, et à calculer leur valeur d'utilité dans le contexte du scope.
    - Le calcul de cette valeur consiste à multiplier le nombre d'énaction d'un scheme proposé par le nombre d'énaction (on additionnera le nombre d'énaction pour les différentes propositions d'un scheme)

  - Cette implémentation propose de définir trois listes :
    - Une liste pour les schemes proposés
    - Une liste pour donner le nombre d'énaction d'un scheme
    - Une liste pour donner la valeur d'utilité calculée (valeurs finales).

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : la sélection de l'action**

  - On prépare les trois listes :

```
public ArrayList<Interaction> scope1;
public ArrayList<Interaction> scope2;

public ArrayList<Interaction> propositionList;
public ArrayList<Integer> propositionNb;
public ArrayList<Integer> propositionValues;
```

  - Dans le constructeur :

```
scope1=new ArrayList<Interaction>();
scope2=new ArrayList<Interaction>();

propositionList=new ArrayList<Interaction>();
propositionNb=new ArrayList<Integer>();
propositionValues=new ArrayList<Integer>();
```

  - Dans DecisionFrame, activez l'affichage des décisions

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : la sélection de l'action**

  - Dans la fonction decision, on réinitialise les trois listes et on ajoute les interactions primitives avec une valeur de 0 :

```
// decision of the agent
private void decision(){

    System.out.println("====================================");

    // define new proposition list
    propositionList.clear();
    propositionNb.clear();

    propositionValues.clear();

    // set primary interactions
    for (int i=0;i<32;i++){
        propositionList.add(interList.get(i));
        propositionNb.add(0);
    }
```

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : la sélection de l'action**

  - On ajoute ensuite les propositions des schemes composites.
    - Un scheme peut proposer sa partie proposition si son contexte est dans le scope.
    - Si le scheme n'a pas dépassé le *regularity sensibility threshold*, et que sa proposition n'est pas primitive, le scheme proposera le contexte de sa proposition
  - Cas proposition 'complète' :

```
// add scheme propositions
for (int i=32;i<interList.size();i++){
    if (interList.get(i).isActive(scope1)){

        if (interList.get(i).post.primary || interList.get(i).enactions>treshold){

            System.out.println(interList.get(i).name+" proposes "+interList.get(i).post.name);

            int value=interList.get(i).post.enactions; // enaction count

            int id=interList.get(i).post.isIncluded(propositionList);
            if (id==-1){
                propositionList.add(interList.get(i).post); // add value
                propositionNb.add(value);
            }
            else{
                propositionNb.set(id, propositionNb.get(id) + value); // increment value
            }

        }
        else{
```

...

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : la sélection de l'action**

  - On ajoute ensuite les propositions des schemes composites.
    - Un scheme peut proposer sa partie proposition si son contexte est dans le scope.
    - Si le scheme n'a pas dépassé le *regularity sensibility threshold*, et que sa proposition n'est pas primitive, le scheme proposera le contexte de sa proposition
  - Cas proposition 'partielle' :

```
        else{

            System.out.println(interList.get(i).name+" proposes "+interList.get(i).post.pre.name);

            int value=interList.get(i).post.enactions;

            int id=interList.get(i).post.pre.isIncluded(propositionList);
            if (id==-1){
                propositionList.add(interList.get(i).post.pre);
                propositionNb.add(value);
            }
            else{
                propositionNb.set(id, propositionNb.get(id) + value);
            }
        }
    }
}
```

Introduction à l'Intelligence Artificielle

- **Le système décisionnel : la sélection de l'action**

  - On doit maintenant ajouter les valeurs des alternatives :
    - Pour chaque proposition, on recherche si ses alternatives sont dans la liste des propositions.
    - Si c'est le cas, on ajoute la valeur de cette alternative à la valeur d'utilité
    - Les valeurs sont stockées dans la liste *propositionValues*

```
// merge propositions
for (int i=0;i<propositionList.size();i++){

     // get value of the proposition
     propositionValues.add(propositionNb.get(i) * propositionList.get(i).valence );

     // search alternatives in the list
     for (int j=0;j<propositionList.get(i).alternativeList.size();j++){

          for (int k=0;k<propositionList.size();k++){
               if (propositionList.get(i).alternativeList.get(j).isEqual(propositionList.get(k))){
                    // add value af the alternative to final utility value
                    propositionValues.set( i , propositionValues.get(i)
                                        + propositionNb.get(k) * propositionList.get(k).valence );

               }
          }
     }
}
```

Introduction à l'Intelligence Artificielle

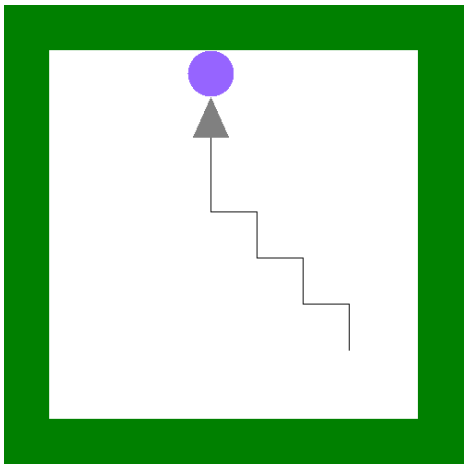
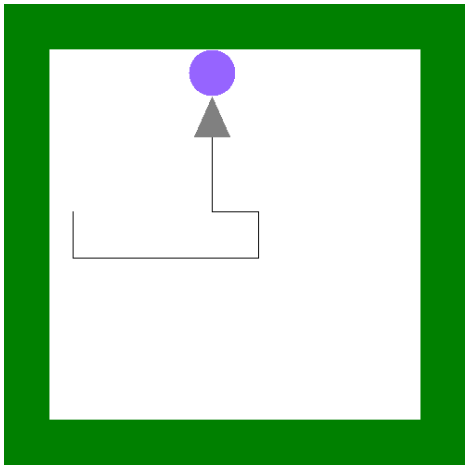
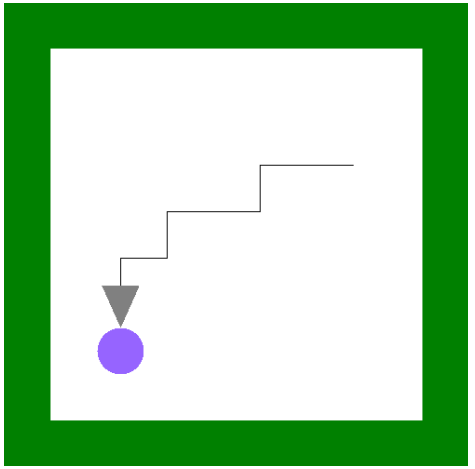
- **Le système décisionnel : la sélection de l'action**

  - Dernière étape : on cherche la meilleure proposition, on la réinitialise et on l'enregistre comme scheme intention (supprimez/commentez le *rand*) :

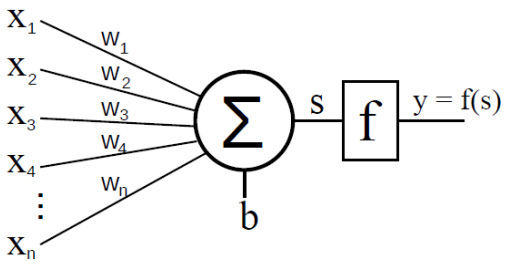
```
        // get scheme with greatest value
        int max=-100000;
        int idmax=-1;
        for (int i=0;i<propositionValues.size();i++){
            if (propositionValues.get(i)>max){
                max=propositionValues.get(i);
                idmax=i;
            }
        }

        intendedScheme=propositionList.get(idmax);
        intendedScheme.reset();

        System.out.println("intended scheme : "+intendedScheme.name);

    } // end of decision
```

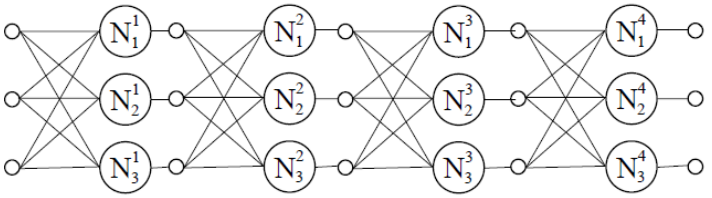
Introduction à l'Intelligence Artificielle

- **Un agent développemental**
  - L'agent est maintenant fonctionnel !
    - Testez différentes configurations de l'environnement (positions initiales de l'agent et de la cible) et essayez d'obtenir des comportements différents avec les mêmes paramètres (seuil de régularité et valeurs de satisfaction)

Introduction à l'Intelligence Artificielle





Conclusion