Manuel d'utilisation du système Ernest

Sommaire :

Liste des versions

Versions 2.8 : versions implémentant le méanisme de la mémoire péri-personnelle.

- 2.8.5 : mémoire péri-personnelle sans mécanisme d'apprentissage
- 2.8.6 : mémoire péri-personnelle avec mécanisme d'apprentissage
- 2.8.6.1 : mémoire péri-personnelle implémentant l'hypothèse vestibulair.

Versions 7 : versions implémantant le mécanisme de la mémoire spatiale extra-personnelle.

- 7.2 : mécanisme de la mémoire spatiale extra-personnelle sur un agent simulé. Système d'interaction simplifié, mémoire spatiale codée en dur.
- 7.2.2.1 : mécanismes d'apprentissage des signatures. Système d'interaction complet, pas de mécanisme d'exploitation de la mémoire.
- 7.2.8 : mécanisme de la mémoire spatiale extra-personnelle sur un agent simulé. Système d'interaction simplifié, mémoire spatiale agnostique
- 7.3 : mécanisme de la mémoire spatiale extra-personnelle sur un robot (mécanisme identique à la version 7.2). Système d'interaction simplifié, mémoire spatiale codée en dur.

Classes utilisées par chaque version

Version	**2.8.5**	**2.8.6**	**2.8.6.1**	**7.2**	**7.2.2.1**	**7.2.8**	**7.3**
//							
- display : ce package regroupe toutes les classes permettant l'affichage d'informations diverses sur l'agent selectionné.							
EnvFrame	**X**	**X**	**X**	**X**	**X**	**X**	**X**
EnvPanel	**X**	**X**	**X**	**X**	**X**	**X**	**X**
EfficiencyFrame	**X**	**X**	**X**	**X**		**X**	**X**
EfficiencyPanel	**X**	**X**	**X**	**X**		**X**	**X**
SpaceFrame	**X**	**X**	**X**	**X**		**X**	**X**
SpacePanel	**X**	**X**	**X**	**X**		**X**	**X**
ColliculusFrame	**X**	**X**	**X**				
ColliculusPanel	**X**	**X**	**X**				
PatternDataFrame		**X**	**X**				
PatternDataPanel		**X**	**X**				
ProbeFrame				**X**	**X**	**X**	
ProbePanel				**X**	**X**	**X**	
PredictionFrame				**X**		**X**	**X**
PredictionPanel				**X**		**X**	**X**
SignatureFrame					**X**		
SignaturePanel					**X**		
MemoryFrame					**X**		
MemoryPanel					**X**		

MovementFrame					**X**		
MovementPanel					**X**		
PlaceFrame						**X**	
PlacePanel						**X**	
CamFrame							**X**
CamPanel							**X**
MapFrame							**X**
MapPanel							**X**

//

- environment : ce package regroupe toutes les classes liées à la description et à la physique de l'environnement.

Block	**X**	**X**	**X**	**X**	**X**	**X**	
Environement	**X**	**X**	**X**	**X**	**X**	**X**	
EnvironementFrame	**X**	**X**	**X**	**X**	**X**	**X**	
EnvironementPanel	**X**	**X**	**X**	**X**	**X**	**X**	
MyFileFilter	**X**	**X**	**X**	**X**	**X**	**X**	
Robot	**X**	**X**	**X**	**X**	**X**	**X**	
Robot (physical)							**X**

//

- interface : ce package regoupe les classes qui décrivent le couplage entre l'agent et l'environnement. C'est notamment ici que sont définies les interactions primitives dont l'agent dispose.

Action	**X**	**X**	**X**	**X**	**X**	**X**	**X**
InteractionList	**X**	**X**	**X**	**X**	**X**	**X**	**X**
PrimitiveInteraction	**X**	**X**	**X**	**X**	**X**	**X**	**X**
Perception				**X**	**X**	**X**	**X**
PrimitiveAction				**X**	**X**	**X**	**X**
PrimitivePerception				**X**	**X**	**X**	**X**

//

- platform : ce package regroupe les classes liées à l'obsevation et l'étude de l'agent.

Agent	**X**	**X**	**X**	**X**	**X**	**X**	**X**
Display	**X**	**X**	**X**	**X**	**X**	**X**	**X**
ITracer	**X**	**X**	**X**	**X**	**X**	**X**	**X**
Observer	**X**	**X**	**X**	**X**	**X**	**X**	**X**
PrintableFrame	**X**	**X**	**X**	**X**	**X**	**X**	**X**
PrintablePanel	**X**	**X**	**X**	**X**	**X**	**X**	**X**
UserInterfaceFrame	**X**	**X**	**X**	**X**	**X**	**X**	**X**
XMLStreamTracer	**X**	**X**	**X**	**X**	**X**	**X**	**X**
Colliculus	**X**	**X**	**X**				
Node					**X**		
NodeList					**X**		

//

- sensorySystem : ce package regroupe les classes associées au système visuel de l'agent (version 7 uniquement)

Probe				**X**	**X**	**X**	
VisualSystem				**X**	**X**	**X**	
Calibration							**X**
ColorRecognition							**X**
Webcam							**X**

//

- spaceMemory : ce package regroupe les classes associées au mécanisme de décision.

Composite	**X**	**X**	**X**	**X**		**X**	**X**
Decision	**X**	**X**	**X**	**X**		**X**	**X**
SpaceMemory	**X**	**X**	**X**	**X**			**X**
EnvironementMemory	**X**	**X**	**X**			**X**	
MemoryElement	**X**	**X**	**X**				
SequenceMemory	**X**	**X**	**X**				
Signature				**X**		**X**	**X**
BlurPattern				**X**			**X**

//

-agnosticMemory : ce package regroupe les classes associées à la construction d'une mémoire spatiale agnostique.

Decision					**X**		
EnvironmentContext					**X**		
Signature					**X**		
SignatureList					**X**		
AgnosticMemory						**X**	
Interaction						**X**	
Object						**X**	
Place						**X**	
PlaceContext						**X**	
PlaceSignature						**X**	
PresenceSignature						**X**	

//

Main	**X**	**X**	**X**	**X**	**X**	**X**	**X**

Architecture du système

Le système implémentant l'agent est divisé en un ensemble de modules interconnectés, définissant chacun un package. La Figure 1 montre cette architecture. L'interface assure le déroulement du cycle de décision, en collectant l'interaction intention issue du système de décision, la transmettant à l'environnement afoin de la simuler, et récupère la ou les interactions énactées qu'elle transmet au système de décision. Le package platform offre à un utilisateur exterieur un esnsemble d'outils et d'interface lui permettant de contrôler l'exécution de la simulation et d'extraire des informations pertinentes sur l'agent. Le package Display contient un ensemble de système d'affichage permettant d'observer différents aspects de l'agent, notamment les structures qu'il génère.

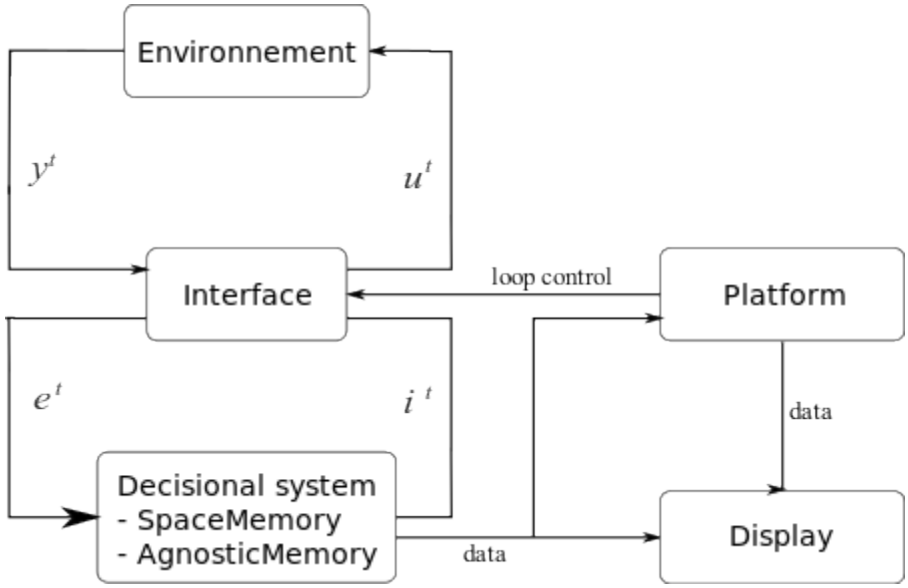

Figure 1 : architecture du système. L'environnement constitue le moteur physique de l'environnement dans lequel l'agent évolue. Le système de décision implémente les différents mécanismes de décision de l'agent. L'interface, comme son nom l'indique, fait le lien entre les interactions issues du système de décision et les actions et percepions issues de l'environnement. C'est l'interface qui contrôle le cycle de décision, et définie l'ensemble initial d'interaction primitive. Display contient une collection de système d'affichage destiné à afficher des informations pour un observateur exterieur. Platform contient un ensemble d'outils destinés à mettre en forme les informations sur l'agent, la gestion des traces, ainsi qu'une interface pour un observateur exterieur permettant le contrôle du cycle de décision.

I Environment

Ce package regroupe toutes les classes liées à la description et à la physique de l'environnement. Il n'y a pas de changement entre les versions de l'agent simulé. Dans la version robotique, seul le système de contrôle du robot est définie.

Block

Composant élémentaire de l'environnement, l'objet Block définie les propriétés d'une case de l'environnement : sa couleur, ses propriétés tactiles, sa visibilité (un objet peut être invisible pour l'agent) ainsi qu'un nom.

Environement

Définie l'environnement de l'agent. L'environnement par défaut est désigné par la variable *DEFAULT_BOARD*. Les différents types d'éléments sont également définis dans cette classe.

Paramètres :

- String DEFAULT_BOARD : nom du fichier de description de l'environnement par défaut.

Fonctions et méthodes principales :

- `public void init(String f) throws Exception` : cette méthode permet de construire l'environnement à partir du fichier spécifié. Voici un exemple de contenu :

```
w w w w w w
w - - - w w
w - w - - w
w - w w - w
w ^ - w - w
w w - - - w
w w w w w w
```

  Les symboles utilisables sont :
  - "^", ">", "v", "<" pour définir un agent et son orientation
  - "-" pour un espace vide
  - "*" pour une proie
  - "w", "g" pour un mur
  - "1", "2" pour une algue.

  Il est bien sûr possible de définir d'autres types de block et de symbole.

  Plusieurs types de blocs sont proposés par défaut :
  - *empty*, un epace vide ("-")
  - *wall1*, un mur vert utilisé pour tester nos mécanismes ("w")
  - *wall2*, un mur vert clair (non détecté par l'agent) ("g")
  - *alga1*, une algue rouge utilisée pour tester nos mécanismes ("1")
  - *alga2*, une algue verte (non détectée par l'agent) ("2")
  - *prey*, une proie mauve utilisée pour tester nos mécanismes ("*")

- `public void drawGrid()`
- `public void drawGrid(int c)`

  Ces deux méthodes permettent de redessiner l'environnement. Le paramètre "c" permet de modifier l'environnement dans la case sélectionnée :
  1 : selection de l'agent le plus proche du point sélectionné
  2 : ajoute ou supprime une proie dans la case sélectionnée
  3 : ajoute ou supprime un mur
  6 : ajoute ou supprime une algue.

EnvironementFrame (`extends PrintableFrame implements ActionListener`)
EnvironementPanel (`extends PrintablePanel implements MouseListener`)

Affichage de l'environnement et interface pour l'utilisateur. L'agent est représenté par un requin gris, les murs par des cases vertes, les proies par des poissons mauves, et les algues par des fleurs rouges. L'utilisateur peut ajouter ou supprimer des éléments :

- clic milieu permet d'ajouter ou de supprimer des proies
- clic droit permet d'ajouter ou de supprimer des murs
- SHIFT+clic droit permet d'ajouter ou supprimer des algues
- clic gauche sur un agent permet de sélectionner l'agent à observer (dans le cas d'un système avec plusieurs agents)

Différence entre les versions :

les versions 2.8.* permettent un affichage des interactions "toucher" sous forme de carrés dont la couleur dépend de l'interaction énactée. Les versions 7.2.* peuvent afficher les interactions "toucher" sous la forme de points rouges (ces intractions ne sont cependant pas utilisées par le mécanisme de décision). Les versions 7.2.* permettent également d'afficher la trace de l'agent.

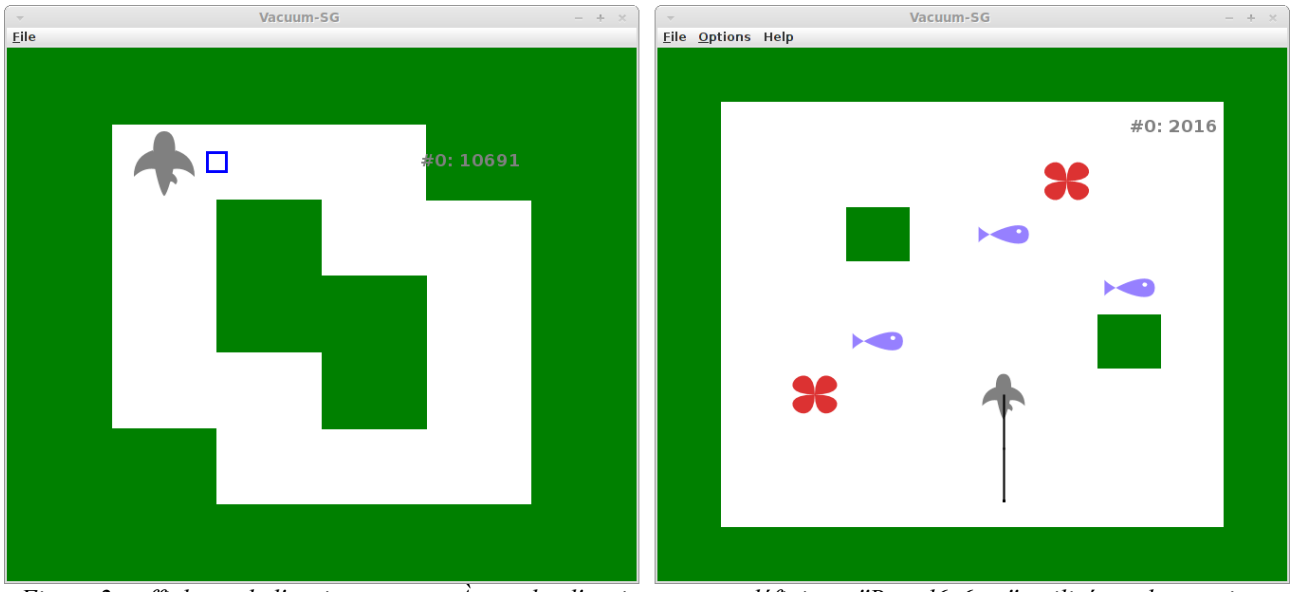

Figure 2 : affichage de l'environnement. À gauche, l'environnement défini par "Board6x6,txt", utilisé par les versions 2.8.. Le carré bleu et blanc indique que l'agent vient d'énacter "toucher un espace vide à droite". À droite, un environnement modifié de celui défini par "Board10x10.txt". Le trait noir montre la trace de l'agent. En haut à droite de l'afficheur est indiqué l'identifiant de l'agent observé et le nombre de cycle de décision effectué.*

MyFileFilter (`extends FileFilter`)

Filtre pour la selection des fichiers texte définissant l'environnement.

Robot

Interface entre l'environnement (physique ou simulé) et l'agent. Le robot reçoit les commandes motrices de l'interface et effectue l'action, puis met à jour ses capteurs. Dans le cas d'un agent robotique, la commande est envoyée par le port série défini par la variable "*file*", par défaut à "/dev/ttyUSB0".

Paramètres :

- boolean eatPrey (versions 7.2.*) : si 'true', les proies disparaissent lorsque l'agent les mange
- boolean replacePrey (versions 7.2.*) : lorsque l'agent mange une proie, une autre proie est ajoutée dans une case vide choisie aléatoirement.

Fonctions et méthodes principales :

- `public void move(double[] act)` : versions simulées
- `public void move(int msg)` : versions robotiques (7.3)

  Cette méthode permet d'envoyer des commandes motrices au robot. Dans le cas d'un agent simulé, la commande "act" est sous la forme d'un vecteur (x,y,theta,t) avec x et y les translations à effectuer, theta la rotation, et t une commande de capteur tactile utilisé par les interactions "toucher". Dans le cas d'un agent robotique, la commande "msg" est envoyée à un robot physique par un port série.
- `public void center()` : (agent simulé seulement) permet de finaliser l'interaction. Dans les versions 2.8.*, la position et l'oientation sont arrondies pour contraindre l'agent à une grille. Dans les versions 7.2.*, le contenu de la case sous l'agent est détecté pour définir le succès de l'interaction "manger".

Différences entre les versions :

les diférences sont principalement dûes aux systèmes sensoriels des différents agents. Les versions 2.8.5 et 2.8.6 sont identiques. La version 2.8.6.1 ajoute un système vestibulaire. Dans les versions 7.2.*, le système sensoriel est implémenté sous la forme d'un vecteur. La version 7.3 ne fait qu'envoyer et recevoir des messages au robot physique.

II SensorySystem (versions 7.*)

Ce package, spécifique aux versions 7, regroupe les modules sensoriels à longue portée de l'agent, notemment son système visuel. Ces modules peuvent également être utilisés pour définir des informations tactiles.

Probe (versions 7.2.*)

Moteur de rendu 2D sur 360°. L'image obtenue est ensuite interprétée pour fournir des valeurs d'entrée aux capteurs simulés, en particulier le système visuel de l'agent.

Fonctions et méthodes principales :

- `public void rendu()` : effectue un rendu de l'environnement et met à jour une rétine simulée utilisable par l'agent.

VisualSystem (versions 7.2.*)

Simule un flot optique à partir de la distance des objets.

Fonctions et méthodes principales :

- `public void opticFlow(PrimitiveInteraction inter)` : met à jour la rétine et simule le flux optique généré par l'énaction de l'interaction *inter*.

Calibration (version 7.3)

Contient la matrice de projection de l'image de la caméra sur une surface plane. Permet d'enregistrer et charger cette projection dans un fichier texte.

Paramètres :

- boolean load : si "true", charge un fichier de configuration.

ColorRecognition (version 7.3)

Contient une liste de couleurs de références fournies par l'utilisateur pour définir les couleurs primaires détectés par le système interactionnel de l'agent, et construit une image composée des couleurs primaires reconnues. L'image est ensuite projetée sur une surface plane.

Fonctions et méthodes principales :

- `public void setColor(int c)` : positionne la couleur primaire sélectionnée par l'utilisateur (1 : rouge, 2 : vert, 3 : bleu et 4 : absence de couleur)
- `public void addColor(Color c)` : ajoute une couleur de référence à la liste
- `public void removeLast()` : retire la dernière couleur de référence de la liste
- `public void getColorMap()` : construit l'image basée sur les couleurs primaires reconnues.
- `public void getMap()` : projète l'image pré-traitée issue de la webcam sur une surface plane.

Webcam (`implements CaptureCallback`) (version 7.3)

Lecture et récupération des images issues de la webcam.

Paramètres :

- int width, height : largeur et hauteur de l'image de la webcam (par défaut, 640x480)
- int std : paramètre de la webcam (par défaut, `V4L4JConstants.STANDARD_WEBCAM`)
- int channel : canal utilisé (par défaut, 0)
- String device : port utilisé par la webcam (par défaut, `"/dev/video0"`)

Fonctions et méthodes principales :

- `public void setImage()` : capture la dernière image prise par la webcam.
- `public void startCapture()` : connecte la webcam et lance la lecture.
- `public void cleanupCapture()` : déconnecte la webcam, ce qui permet de la débrancher sans arreter le système.

III Interface

Ce package regoupe les classes qui décrivent le couplage entre l'agent et l'environnement, et contrôle le cycle de décision. C'est notamment ici que sont définies les interactions primitives dont l'agent dispose. Dans les descriptions suivantes, le terme "robot" fait référence au système d'exécution de l'interaction dans l'environnement, défini par la classe *Robot* du package *environment*.

Action

Cette classe contrôle et exécute l'interaction entre l'agent et l'environnement. L'exécution d'un cycle de décision s'effectue tout d'abord en récupérant l'interaction primitive donnée par le système de décision. Les commandes motrices associées à chaque interaction primitive sont ensuite envoyées au robot à chaque cycle de simulation. Les cycles de simulation permettent de tester à intervalle régulier la validité du déplacement. Le cycle de décision peut notemment être interrompu en cas de collision, par exemple. Dans le cas des agents simulé, l'agent effectue 10 cycles de simulation par cycle de décision (nombre paramétrable via le paramètre "nbSubStep". Dans le cas de l'agent robotique, une seule commande motrice est envoyée, c'est alors le robot physique qui se charge d'exécuter l'interaction. Une fois l'exécution de l'interaction terminée, le mécanisme interprète les informations sensorielles du robot pour définir la ou les interactions énactées, et met à jour le système de décision de l'agent. Dans le cas des versions 7.2.*, c'est la classe perception qui interprète les capteurs du robot et détermine l'ensemble des interactions énactées.

Paramètres :

- int nbSubstep (versions 2.* et 7.2.*) : définit le nombre de cycle de simulation par cycle de décision (par défaut à 10).

Fonctions et méthodes principales :

- `public void act()` : méthode qui contrôle le cycle de décision de l'agent. Elle récupère l'interaction intention donnée par le système de décision de l'agent, puis applique les commandes motrices associées au robot. La ou les interactions énactées sont définies à partir de l'interaction intention et de l'état des capteurs du robot, puis sont transmises au système de décision de l'agent afin de mettre à jour ce dernier.
- `public PrimitiveInteraction recognizeEnacted(String action)` : (versions 2.8.* seulement, voir classe Perception pour les versions 7.*) cette fonction interprète l'état des capteurs du robot et l'action utilisée pour définir l'interaction énactée.

Différences entre les versions :

différences dans les interactions et les systèmes sensoriels utilisés. Les versions 2.8.* gèrent également l'interprétation de l'état des capteurs du robot. La version 7.3 ne gère pas l'exécution des commandes motrices.

Perception (`public class Perception`) (versions 7.* seulement)

Ce module interprète les états des capteurs et l'action utilisée pour définir l'ensemble des interactions énactées à partir des états des capteurs du robot et de l'interaction intention.

Fonctions et méthodes principales :

- `public void update()` : intègre les états des capteurs du robot sous la forme d'un vecteur.
- `public PrimitiveInteraction recognize(PrimitiveInteraction intended)` : cette fonction définit l'interaction primaire énactée.
- `public float[] recognizeEnactedEnsemble(PrimitiveInteraction enacted)` (versions 7.2, 7.2.2.1, 7.2.8, 7.3) : cette fonction retourne l'ensemble d'interaction énactées sous la forme d'un vecteur, d'après l'ensemble d'interaction défini pour la mémoire spatiale.
- `public float[] recognizeAgnosticEnactedEnsemble(PrimitiveInteraction enacted)` (versions 7.2.8 uniquement) : cette fonction est utilisé par la version 7.2.8 de l'agent pour obtenir l'ensemble d'interaction énactées sous la forme d'un vecteur, d'après l'ensemble d'interaction défini pour sa mémoire spatiale agnostique.

Différences entre les versions :

les versions diffèrent par leur système sensoriel et ainsi par l'interpretation effectuée. On peut noter que la version 7.2.8, qui utilise à la fois l'ensemble d'interaction utilisé par la mémoire codée en dur, et celui utilisé par la mémoire agnostique, utilise à la fois les fonctions *recognizeEnactedEnsemble* et *recognizeAgnosticEnactedEnsemble*.

InteractionList

C'est ici que les interactions entre l'agent et l'environnement doivent être définies. La fonction statique valence permet de calculer la valeur de satisfaction d'une interaction, selon des règles prédéfinies par le concepteur de l'agent. Si l'action et la perception sont séparés pour des raisons de simplification algorithmique, les mécanismes de décision de l'agent n'ont jamais accès à l'action et à la perception qui compose une interaction.

Fonctions et méthodes principales :

- `public static float valence(String a, String p)` : (versions 2.8.*)
- `public static float valence(PrimitiveAction a, PrimitivePerception p)` (versions 7.*)

  Cette fonction permet d'attribuer une valeur de satisfaction à chaque interaction primitve, en fonction de l'action *a* et de la perception *p* qui la compose.

Différences entre les versions :

- dans les versions 7.*, la liste des actions et des perceptions primitives sont également définies et utilisées pour la construction des interactions primitives.

PrimitiveAction (versions 7.* uniquement)

Définit une action primitive, composant d'une interaction : nom et identifiant.

PrimitivePerception (versions 7.* uniquement)

Définit une perception primitive, composant d'une interaction : nom et identifiant

PrimitiveInteraction

Définition des interactions primitives et leurs propriétés.

Différences entre les versions :

les versions 2.8.* définissent les actions et perceptions sous forme de chaines de caractères, tandis que les versions 7.* utilisent les structures *PrimitiveAction* et *PrimitivePerception*. La version 2.8.6.1 définit en outre une information vestibulaire (sous forme de chaine de caractère). Dans les versions 7.2, 7.2.8 et 7.3, les déplacements produits par l'énaction d'une interaction primitive, et utilisés par la mémoire spatiale codée en dur ou par le mécanisme de détection des instances d'objet, sont prédéfinis.

IV platform

ce package regroupe un ensemble d'outils destinés à l'analyse du comportmeent et des structures apprises par l'agent, et propose une interface pour un observateur externe.

Agent

Cette classe sert principalement de plaque tournante entre les différents mécanismes qui constituent l'agent. C'est dans cette classe que sont instanciés les différents modules d'affichage, que l'on peut activer ou désactiver par le biais d'un ensemble de variables. La liste des afficheurs disponibles varie en fonction des versions.

Paramètres :

Il est possible de paramétrer les afficheurs utilisés à l'aide d'un ensemble de variables dans *setDisplay*.

Fonctions et méthodes principales :

- `public void setDisplay()` : initialise les afficheurs sélectionnés.
- `public void act()` : lance un cycle de décision par le biais de la classe *Action*.

Différences entre les versions :

chaque version dispose de module et d'afficheurs qui lui sont propre. Les versions simulées permettent également de modifier la position et l'orientation de l'agent dans l'environnement.

Display

Gestionnaire de la liste des afficheurs.

ITracer (`public interface ITracer<EventElement>`)

Interface pour la génération des traces sur Abstract-Liste

Observer

Permet l'analyse des différentes propriétés de l'agent. Il est également possible de capturer des images des différents afficheurs lorsque la variable *picture* est à 'true'. Les captures se font tous les *capture_delay* cycles de décision, et sont enregistrées dans le répertoire spécifié par la variable *path*.

Paramètres :

- int length : longueur de la timeline (par défaut à 200)
- boolean picture : si à vrai, capture une image des afficheurs tous les *capture_delay* cycles de décision.
- int capture_delay : nombre de cycles entre chaque capture d'image (par défaut à 200).

Fonctions et méthodes principales :

- `public void updateObserver(PrimitiveInteraction inter)` : collecte les données relatives à l'agent pour le dernier cycle de décision.
- `public void trace()` : génère un obsel pour Abstract-Lite.

Différences entre les versions :

Les informations collectées dépendent des versions :

- Commun : valeur de satisfaction moyenne sur les 1, 10, 50 et 100 dernier cycles de décision
- 2.8.6 et 2.8.6.1 : nombre d'interactions fiables, toujours vraies, décorrélées, correctes.
- 7.2 et 7.2.8 : trace de l'agent dans l'environnement. La version 7.2.8 intègre également des variables partagées par les afficheurs
- 7.2.2.1 : trace de l'agent dans l'environnement, met à jour le graphe affichant la relation entre les signatures d'interaction

PrintableFrame (`extends JFrame implements Printable, KeyListener`)
PrintablePanel (`extends JPanel`)

Classe générique d'afficheur pouvant être exportées en PDF.

UserInterfaceFrame (`extends JFrame implements ActionListener`)

Interface pour un observateur exterieur. Elle permet de contrôler le déroulement des cycles de décision.

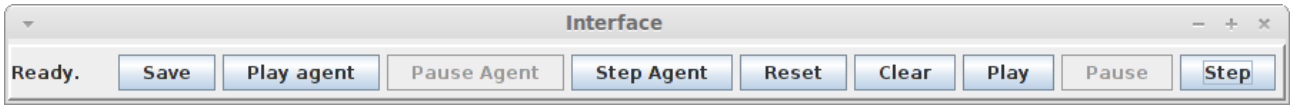

*Figure 3 : interface utilisateur des versions 7.2.**

Liste des boutons :

- Save : permet de sauvegarder les structures apprises par l'agent dans des fichiers texte.
- Play agent : lance la simulation de l'agent sélectionné
- Pause agent : met en pause l'agent sélectionné
- Step Agent : fait exécuter un cycle de décision à l'agent sélectionné
- Reset : réinitialise la simulation
- Clear (versions 7.2, 7.2.2.1 et 7.2.8) : efface la trace de l'agent.
- Play : lance la simulation pour l'ensemble des agents de l'environnement
- Pause : met la simulation en pause
- Step : fait exécuter un cycle de décision à l'ensemble des agents
- Camera On/Camera Off (version 7.3) : permet de connecter/déconnecter la webcam, opération nécessaire lorsque le cable est torsadé.

XMLStreamTracer (`implements ITracer<Element>`)

Permet l'envoi de traces.

Colliculus (versions 2.8.*)

Permet de définir les bundles d'interaction en fonction du type d'objet et de la position des objets qui affordent les interactions de l'agent. Cette classe permet également de déterminer l'objet réellement associé aux interactions.

Fonctions et méthodes principales :

- `public void updatePoints()` : met à jour la position de points représentant les signatures d'interaction dans le but de faire émerger des bundles. La position des points et leur distance par rapport aux autres dépend des signatures des interactions.
- `public int simulation(Composite p)` : calcule l'objet qui afforde une interaction composite p (l'agent n'a pas accès à cette information).

Différences entre les versions :

Les versions 2.8.6 et 2.8.6.1 tiennent compte de la fiabilité des interactions. Les coefficients de la version 2.8.6 sont également modifiés pour tenir compte du plus grand nombre d'interaction.

Node (version 7.2.2.1 seulement)
NodeList (version 7.2.2.1 seulement)

Système permettant de mettre en valeur les propriétés des signatures, en regroupant soit les signatures en fonction de la position de l'objet qui les affordent, soit en montrant la différence de position entre l'objet caractérisé par une interaction et la position de l'objet qui l'afforde. Un Node est un caractérisé par des coordonnées, un identifiant et un type d'interaction. NodeList gère l'ensemble des Nodes.

Fonctions et méthodes principales :

- `public void moveNode()` : simule un ensemble de contraintes dépendant des signatures d'interaction pour faire émerger des propriétés entre les interactions.

V Display

Ce package regroupe toutes les classes permettant l'affichage d'informations diverses sur l'agent selectionné. Comme l'utilisation des afficheurs dépend des versions de l'agent, les versions utilisant chaque afficheur est spécifié.

EnvFrame (`extends PrintableFrame`)
EnvPanel (`extends PrintablePanel`)

(toutes les versions)
Classes génériques de panels et de frames pour afficher diverses informations sur l'agent selectionné.

EfficiencyFrame (`extends EnvFrame`)
EfficiencyPanel (`extends EnvPanel`)

(Versions 2.8.*, 7.2, 7.2.8 et 7.3)
Affichage de la valeur de satisfaction au cours du temps, ainsi que la satisfaction moyenne sur les 10, 50 et 100 derniers cycles de décision.

Différences entre les versions :

Les versions prennent en considération des ensembles d'interactions différents. Les versions 2.8.* affichent les interactions "toucher".

SpaceFrame (`extends EnvFrame`)
SpacePanel (`extends EnvPanel implements MouseListener`)

(Versions 2.8.*, 7.2, 7.2.8 et 7.3)
Affiche diverses informations sur les signatures d'interaction, les contextes augmentés et la mémoire spatiale. Chaque version de l'agent dispose d'une version de l'afficheur qui lui est propre (voir section VII pour plus de détails).

ColliculusFrame (`extends EnvFrame`)
ColliculusPanel (`extends EnvPanel implements MouseListener,MouseMotionListener`)

(versions 2.8.*)
Ce système d'affichage permet de mettre en évidence les liens entre les signatures d'interactions, sous la forme de bundles d'interactions : d'une part, les bundles en fonction de la position de l'objet qui afforde les interactions, et d'autre part, en fonction du type d'objet qui afforde les interactions.

Différences entre les versions :

Les versions 2.8.6 et 2.8.6.1 ajoutent un bouton permettant de masquer les interactions considéréescomme non fiables par l'agent.

PatternDataFrame (`extends EnvFrame`)
PatternDataPanel (`extends EnvPanel`)

(versions 2.8.2 et 2.8.2.1)
Affiche l'évolution du nombre d'interaction total, du nombre d'interactions considérées comme fiables, toujours vraies et dé-corrélées, et du nombre d'interactions dont les signatures sont correctes d'un point de vue exterieur.

ProbeFrame (`extends EnvFrame`)
ProbePanel (`extends EnvPanel`)

(versions 7.2, 7.2.2.1 et 7.2.8)
Affiche l'environnement du point de vue de l'agent

PredictionFrame (`extends EnvFrame`)
PredictionPanel (`extends EnvPanel`)
(versions 7.2, 7.2.8 et 7.3)
Affiche sous forme de graphes la valeur de certitude de succès et d'échec des interactions primitives à chaque cycle de décision.

SignatureFrame (`extends EnvFrame`)
SignaturePanel (`extends EnvPanel implements MouseListener`)
(version 7.2.2.1)
Affiche le contexte interactionnel, les signatures d'interaction et les contextes prédits et augmentés.

MemoryFrame (`extends EnvFrame`)
MemoryPanel (`extends EnvPanel implements MouseListener`)
(version 7.2.2.1)
Affiche la relation entre les signatures d'interaction. Il est possible soit d'afficher la différence de position entre une interaction et le barycentre de sa signature pour mettre en évidence le déplacement produit par une interaction, soit de regrouper les interactions liées par une même signature pour afficher les bundles d'interactions.

Paramètres :
- int gap : distance entre deux interactions (par défaut, 8).

MovementFrame (`extends EnvFrame`)
MovementPanel (`extends EnvPanel implements MouseListener`)
(version 7.2.2.1)
Calcule et affiche les images de signatures définies par une séquence d'interaction choisie par l'utilisateur.

PlaceFrame (`extends EnvFrame`)
PlacePanel (`extends EnvPanel implements MouseListener`)
(version 7.2.8)
Affiche des informations sur les signatures de lieu des lieux composites, sur les éléments détectés et sur les objets stockés et suivi par la mémoire spatiale.

CamFrame (`extends EnvFrame`)
CamPanel (`extends EnvPanel implements MouseListener,MouseMotionListener`)
(version 7.3)
Affiche l'image issue de la webcam et l'image filtrée. Cet afficheur permet également à un utilisateur de sélectionner les couleurs de référence pour la détection des couleurs primaires.

MapFrame (`extends EnvFrame`)
MapPanel (`extends EnvPanel implements MouseListener,MouseMotionListener`)
(version 7.3)
Afficheur utilisé pour définir la matrice de projection de l'image webcam sur une surface plane.

VI Mécanismes de décision

Ce package regroupe l'ensemble des classes qui définissent les mécanismes de sélection de l'interaction intention à chaque cycle de décision. Les modules qui composent le système de décision diffèrent selon la version de l'agent. Chaque version sera donc décrite séparément.

Version 2.8.5

Cette version de l'agent a principalement pour but de tester le principe de la mémoire spatiale péri-personnelle et le comportement de l'agent lorsque celui-ci ne dispose que d'un mécanisme d'exploitation de la mémoire spatiale.

Composite

Description de l'interaction composite. Une interaction composite est constituée d'une séquence d'interactions primitives, d'une signature d'interaction, d'une valeur de satisfaction ainsi que de compteurs de succès et d'échec destinés à définir sa fiabilité.

Paramètres :

- int reliabilitySize : définit une limite de compteurs de succès et d'échecs prédits correctement et succéssivement, utilisés pour définir la fiabilité de l'interaction composite (par défaut à 10).

Fonctions et méthodes principales :

- `public float prediction(float[][] img)`: donne la prédiction de succès ou d'échec de l'interaction composite dans un contexte interactionnel *img* donné.
- `public void learn(float[][] img, boolean res)` : intègre le résultat *res* de l'interaction composite dans le contexte du cycle de décision t-1 *img*.
- `public Composite getNext(PrimitiveInteraction inter, ArrayList<Composite> pList)` : donne l'interaction composite *suivante*.
- `public int check(PrimitiveInteraction[] timeline, boolean increment)` : teste si l'interaction composite s'est terminée comme un succès (2), comme un échec (0) ou si elle n'a pas été énactée (1).
- `public boolean isValide(ArrayList<MemoryElement> list)` : retourne 'true' si toutes les sous-séquences du path de l'interaction composite sont présentes dans la liste *list*. Cette fonction permet de s'assurer que l'interaction composite est énactable d'après la liste *True* de la mémoire spatiale.
- `public boolean isInvalide(ArrayList<MemoryElement> list)` : retourne 'true' si une sous-séquence du path de l'interaction composite est présente dans la liste *list*. Cette fonction permet de s'assurer que l'énactabilité de l'interaction composite n'est pas invalidé par la liste *False* de la mémoire spatiale.

Decision

Il s'agit du mécanisme de décision qui détermine l'interaction primitive à énacter au cycle suivant. Le mécanisme vérifit dans un premier temps le résultat du cycle de décision suivant, puis, si l'énaction de l'interaction composite précédente s'est terminé ou a été interrompu, sélectionne, parmi les interactions considérées comme énactables, celle qui dispose de la valeur de satisfaction la plus élevée, et, si nécessaire, une interaction épistémique. Les interactions primitives qui composent l'interaction composite sont ensuite énactées une à une.

Fonctions et méthodes principales :

- `public PrimitiveInteraction decision()` : sélectionne les interactions composites et fourni l'interaction primitive intention à énacter au cycle de décision suivant.
- `public void learn(PrimitiveInteraction inter)` : intègre le résultat de l'énaction de la précédente interaction pour définir si l'énaction est un succès ou un échec.

SpaceMemory

Module principal de la mémoire spatiale. Ce module met à jour les autres composants, détecte et enregistre les interactions composites découvertes et calcule les prédictions de succès ou d'échec des interactions composites. Il est possible de sauvegarder les interactions composites et leur signatures dans un fichier texte.

Paramètres :

- int timeSize : définit la longueur maximale des ineractions composites (idéalement, 2 ou 3).
- boolean load : si à 'true', charge la liste des interactions composites et leurs signatures à partir d'un fichier texte.

Fonctions et méthodes principales :

- `public void nextStep(PrimitiveInteraction interaction)` : met à jour la mémoire spatiale et calcule les prédictions de succès ou d'échec des interactions composites.
- `public Composite getMissingSaccade(float[][] missing)` : détermine une interaction épistémique permettant d'obtenir l'information manquante indiquée par le vecteur *missing*.
- `public void save()` : enregistre les interactions composites et leur signatures dans un fichier texte.
- `public void load()` : charge les interactions composites et leur signatures depuis un fichier texte.

EnvironementMemory

Module qui construit le contexte environnemental E_t sous forme d'un vecteur (on utilise une matrice 3x2 pour des raisons de simplification algorithmique), ainsi que les contextes augmentés E_t' à E_t'''

Fonctions et méthodes principales :

- `public void updateEnvironment(PrimitiveInteraction inter)` : construit le contexte interactionnel et les contextes augmentés à partir de l'interaction énactée *inter* et de la mémoire spatiale.
- `public float[][] getMissingEnv(Composite p)` : détermine les informations du contexte interactionnel manquantes pour déterminer si l'interaction *p* est énactable ou non.

MemoryElement

Structure utilisée par la mémoire spatiale, contenant une interaction composite mise à jour à chaque cycle de décision, et l'interaction composite initialement détectée quand l'élément a été ajouté en mémoire. Cette structure permet notamment, lorsqu'une incohérence est détectée en mémoire, de savoir quelle interaction a pu être à l'origine de la présence de l'interaction composite impliquée dans l'incohérence.

Fonctions et méthodes principales :

- `public MemoryElement nextStep(PrimitiveInteraction inter, ArrayList<Composite> patternList)` : mise à jour de l'élément.

SequenceMemory

Mécanisme qui gère et met à jour la liste des interactions composites considérées comme possible (True) et non possible (False). La mémoire contient plusieurs listes True et False, définissant plusieurs niveaux de fiabilité.

Fonctions et méthodes principales :

- `public void updateSequences(ArrayList<Composite> compositeList, ArrayList<Float[]> predictions)` : met à jour les listes True et False en ajoutant les interactions composites considérées comme possible ou impossible.
- `public void updateMemory(PrimitiveInteraction inter)` : met à jour les séquences stockées en mémoire.
- `public ArrayList<Composite> getCandidates()` : détermine la liste des interactions considérées comme énactable.

Version 2.8.6

Cette version de l'agent a pour but de tester le principe de la mémoire spatiale péri-personnelle et le comportement de l'agent lorsque les signatures d'interactions sont apprises et renforcées par le biais d'un mécanisme d'apprentissage dédié.

Composite

Description de l'interaction composite. Une interaction composite est constituée d'une séquence d'interactions primitives, d'une signature d'interaction, d'une valeur de satisfaction ainsi que de compteurs de succès et d'échec destinés à définir sa fiabilité.

Paramètres :

- int reliabilitySize : définit le nombre de prédictions correctes consécutives de succès et d'échec pour qu'une interaction composite soit considérée comme fiable par l'agent (par défaut à 10).

Fonctions et méthodes principales :

- `public float prediction(float[][] img)`: donne la prédiction de succès ou d'échec de l'interaction composite dans un contexte interactionnel *img* donné.
- `public void learn(float[][] img, boolean res)` : intègre le résultat *res* de l'interaction composite dans le contexte du cycle de décision t-1 *img*.
- `public void learnMovement(ArrayList<Composite> pList)` : permet l'apprentissage des signatures de mouvement, formant un lien entre les interactions.
- `public Composite getNext(PrimitiveInteraction inter, ArrayList<Composite> pList)` : donne l'interaction composite *suivante*.
- `public int matchMap(float[][] map)` : retourne 1 si le contexte *map* contient l'ensemble de la signature de l'interaction composite, 2 si il contient partiellement la signature et -1 si il est incompatible.
- `public int check(PrimitiveInteraction[] timeline, boolean increment)` : teste si l'interaction composite s'est terminée comme un succès (2), comme un échec (0) ou si elle n'a pas été énactée (1).
- `public boolean isValide(ArrayList<MemoryElement> list)` : retourne 'true' si toutes les sous-séquences du path de l'interaction composite sont présentes dans la liste *list*. Cette fonction permet de s'assurer que l'interaction composite est énactable d'après la liste *True* de la mémoire spatiale.
- `public boolean isInvalide(ArrayList<MemoryElement> list)` : retourne 'true' si une sous-séquence du path de l'interaction composite est présente dans la liste *list*. Cette fonction permet de s'assurer que l'énactabilité de l'interaction composite n'est pas invalidé par la liste *False* de la mémoire spatiale.

Decision

Il s'agit du mécanisme de décision qui détermine l'interaction primitive à énacter au cycle suivant. Le mécanisme vérifit dans un premier temps le résultat du cycle de décision suivant, puis, si l'énaction de l'interaction composite précédente s'est terminé ou a été interrompu, sélectionne l'interaction composite suivante. Le module teste tout d'abord si une interaction peut être testée dans le contexte courant. Sinon, le module récupère la liste des interactions énactables et sélectionne celle dont la valeur de satisfaction est la plus élevée. Lorque cela est nécessaire, une interaction épistémique est sélectionnée. Les interactions primitives qui composent l'interaction composite sont ensuite énactées une à une.

Fonctions et méthodes principales :

- `public PrimitiveInteraction decision()` : sélectionne les interactions composites et fourni l'interaction primitive intention à énacter au cycle de décision suivant. En cas d'échec, la fiabilité des interactions impliquées dans l'erreur est réduite ou mise à zéro.
- `public void learn(PrimitiveInteraction inter)` : intègre le résultat de l'énaction de la précédente interaction pour définir si l'énaction est un succès ou un échec.
- `public void setSaccade(Composite s)` : permet à d'autres modules de définir une interaction épistémique.

SpaceMemory

Module principal de la mémoire spatiale. Ce module met à jour les autres composants, détecte et enregistre les interactions composites découvertes et calcule les prédictions de succès ou d'échec des interactions composites. Il est possible de sauvegarder les interactions composites et leur signatures dans un fichier texte.

Paramètres :

- int timeSize : définit la longueur maximale des ineractions composites (idéalement, 2 ou 3).
- boolean computeM : active l'apprentissage des signatures de mouvement (non utilisées par l'agent)
- boolean load : si à 'true', charge la liste des interactions composites et leurs signatures à partir d'un fichier texte.

Fonctions et méthodes principales :

- `public void nextStep(PrimitiveInteraction interaction)` : met à jour la mémoire spatiale et calcule les prédictions de succès ou d'échec des interactions composites.
- `public Composite getMissingSaccade(float[][] missing)` : détermine une interaction épistémique permettant d'obtenir l'information manquante indiquée par le vecteur *missing*.
- `public void save()` : enregistre les interactions composites et leur signatures dans un fichier texte.
- `public void load()` : charge les interactions composites et leur signatures depuis un fichier texte.

EnvironementMemory

Module qui construit le contexte environnemental E_t sous forme d'un vecteur (on utilise une matrice 3x2 pour des raisons de simplification algorithmique), ainsi que les contextes augmentés E_t' à E_t'''

Fonctions et méthodes principales :

- `public void updateEnvironment(PrimitiveInteraction inter)` : construit le contexte interactionnel et les contextes augmentés à partir de l'interaction énactée *inter* et de la mémoire spatiale.
- `public float[][] getMissingEnv(Composite p)` : détermine les informations du contexte interactionnel manquantes pour déterminer si l'interaction *p* est énactable ou non.

MemoryElement

Structure utilisée par la mémoire spatiale, contenant une interaction composite mise à jour à chaque cycle de décision, et l'interaction composite initialement détectée quand l'élément a été ajouté en mémoire. Cette structure permet notamment, lorsqu'une incohérence est détectée en mémoire, de savoir quelle interaction a pu être à l'origine de la présence de l'interaction composite impliquée dans l'incohérence.

Fonctions et méthodes principales :

- `public MemoryElement nextStep(PrimitiveInteraction inter, ArrayList<Composite> patternList)` : mise à jour de l'élément.

SequenceMemory

Mécanisme qui gère et met à jour la liste des interactions composites considérées comme possible (True) et non possible (False). La mémoire contient plusieurs listes True et False, définissant plusieurs niveaux de fiabilité.

Fonctions et méthodes principales :

- `public void updateSequences(ArrayList<Composite> compositeList, ArrayList<Float[]> predictions)` : met à jour les listes True et False en ajoutant les interactions composites considérées comme possible ou impossible.
- `public void updateMemory(PrimitiveInteraction inter)` : met à jour les séquences stockées en mémoire.
- `public ArrayList<Composite> getCandidates1()` : fournit, lorsque c'est possible, une interaction considérée comme non fiable pouvant être testée dans le contexte courant. Si une interaction épistémique est nécessaire, elle sera définie et envoyée au module de décision.
- `public ArrayList<Composite> getCandidates2()` : détermine la liste des interactions considérées comme énactable.

Version 2.8.6.1

Cette version est similaire à la versions 2.8.6 et ajoute un système vestibulaire à l'agent, donnant pour chaque interaction primitive une information sur le mouvement (sans signification a priori) produit par l'énaction de cette interaction primitive. Le but est ici de montrer l'amélioration des performances de l'agent lorsque celui-ci dispose d'organes sensoriels donnant des informations sur ses déplacements dans l'espace.

Composite

Description de l'interaction composite. Une interaction composite est constituée d'une séquence de mouvements et d'une interaction finale, d'une signature d'interaction, d'une valeur de satisfaction ainsi que de compteurs de succès et d'échec destinés à définir sa fiabilité.

Paramètres :

- int reliabilitySize : définit le nombre de prédictions correctes consécutives de succès et d'échec pour qu'une interaction composite soit considérée comme fiable par l'agent (par défaut à 10).

Fonctions et méthodes principales :

- `public float prediction(float[][] img)`: donne la prédiction de succès ou d'échec de l'interaction composite dans un contexte interactionnel *img* donné.
- `public void learn(float[][] img, boolean res, PrimitiveInteraction inter)` : intègre le résultat *res* de l'interaction composite dans le contexte du cycle de décision t-1 *img*. Un verrou est ensuite posé sur l'interaction composite, afin qu'elle ne puisse plus être testée tant que l'agent n'effectue pas un mouvement différent de celui associé à l'interaction *inter*.
- `public Composite getNext(PrimitiveInteraction inter, ArrayList<Composite> pList)` : donne l'interaction composite *suivante*.
- `public float getValence(ArrayList<MemoryElement> listT, ArrayList<PrimitiveInteraction> primitives)` : calcule la séquence d'interactions primitives énactable et compatible avec l'interaction composite la plus satisfaisante pour l'agent, dans le but de définir la valeur de satisfaction de l'interaction composite.
- `public int matchMap(float[][] map)` : retourne 1 si le contexte *map* contient l'ensemble de la signature de l'interaction composite, 2 si il contient partiellement la signature et -1 si il est incompatible.
- `public int check(PrimitiveInteraction[] timeline, boolean increment)` : teste si l'interaction composite s'est terminée comme un succès (2), comme un échec (0) ou si elle n'a pas été énactée (1).
- `public boolean isValide(ArrayList<MemoryElement> list)` : retourne 'true' si toutes les sous-séquences du path de l'interaction composite sont présentes dans la liste *list*. Cette fonction permet de s'assurer que l'interaction composite est énactable d'après la liste *True* de la mémoire spatiale.
- `public boolean isValide(ArrayList<MemoryElement> list, ArrayList<Composite> pList, float[][] map, boolean t)` : teste l'énactabilité de l'interaction composite d'après la liste list et les interactions composites considérées comme possible (si t est à 'true') ou seulement non impossible (si t est à 'false').
- `public boolean isInvalide(ArrayList<MemoryElement> list)` : retourne 'true' si une sous-séquence du path de l'interaction composite est présente dans la liste *list*. Cette fonction permet de s'assurer que l'énactabilité de l'interaction composite n'est pas invalidé par la liste *False* de la mémoire spatiale.

Decision

Il s'agit du mécanisme de décision qui détermine l'interaction primitive à énacter au cycle suivant. Le mécanisme vérifit dans un premier temps le résultat du cycle de décision suivant, puis, si l'énaction de l'interaction composite précédente s'est terminé ou a été interrompu, sélectionne l'interaction composite suivante. Le module teste tout d'abord si une interaction peut être testée dans le contexte courant. Sinon, le module récupère la liste des interactions énactables et sélectionne celle dont la valeur de satisfaction est la plus élevée. Lorque cela est nécessaire, une interaction épistémique est sélectionnée. Les interactions primitives qui composent l'interaction composite sont ensuite énactées une à une.

Fonctions et méthodes principales :

- `public PrimitiveInteraction decision()` : sélectionne les interactions composites et fourni l'interaction primitive intention à énacter au cycle de décision suivant.
- `public void learn(PrimitiveInteraction inter)` : intègre le résultat de l'énaction de la précédente interaction pour définir si l'énaction est un succès ou un échec.
- `public void setSaccade(Composite s)` : permet à d'autres modules de définir une interaction épistémique.

SpaceMemory

Module principal de la mémoire spatiale. Ce module met à jour les autres composants, détecte et enregistre les interactions composites découvertes et calcule les prédictions de succès ou d'échec des interactions composites. Il est possible de sauvegarder les interactions composites et leur signatures dans un fichier texte.

Paramètres :

- int timeSize : définit la longueur maximale des ineractions composites (idéalement, 2 à 4).
- boolean load : si à 'true', charge la liste des interactions composites et leurs signatures à partir d'un fichier texte.

Fonctions et méthodes principales :

- `public void nextStep(PrimitiveInteraction interaction)` : met à jour la mémoire spatiale et calcule les prédictions de succès ou d'échec des interactions composites.
- `public Composite getMissingSaccade(float[][] missing)` : détermine une interaction épistémique permettant d'obtenir l'information manquante indiquée par le vecteur *missing*.
- `public void save()` : enregistre les interactions composites et leur signatures dans un fichier texte.
- `public void load()` : charge les interactions composites et leur signatures depuis un fichier texte.

EnvironementMemory

Module qui construit le contexte environnemental E_t sous forme d'un vecteur (on utilise une matrice 3x2 pour des raisons de simplification algorithmique), ainsi que les contextes augmentés E_t' à E_t'"

Fonctions et méthodes principales :

- `public void updateEnvironment(PrimitiveInteraction inter)` : construit le contexte interactionnel et les contextes augmentés à partir de l'interaction énactée *inter* et de la mémoire spatiale.
- `public float[][] getMissingEnv(Composite p)` : détermine les informations du contexte interactionnel manquantes pour déterminer si l'interaction *p* est énactable ou non.

MemoryElement

Structure utilisée par la mémoire spatiale, contenant une interaction composite mise à jour à chaque cycle de décision, et l'interaction composite initialement détectée quand l'élément a été ajouté en mémoire. Cette structure permet notamment, lorsqu'une incohérence est détectée en mémoire, de savoir quelle interaction a pu être à l'origine de la présence de l'interaction composite impliquée dans l'incohérence.

Fonctions et méthodes principales :

- `public MemoryElement nextStep(PrimitiveInteraction inter, ArrayList<Composite> patternList)` : mise à jour de l'élément.

SequenceMemory

Mécanisme qui gère et met à jour la liste des interactions composites considérées comme possible (True) et non possible (False). La mémoire contient plusieurs listes True et False, définissant plusieurs niveaux de fiabilité.

Fonctions et méthodes principales :

- `public void updateSequences(ArrayList<Composite> compositeList,`

`ArrayList<Float[]> predictions)` : met à jour les listes True et False en ajoutant les interactions composites considérées comme possible ou impossible.

- `public void updateMemory(PrimitiveInteraction inter)` : met à jour les séquences stockées en mémoire.
- `public ArrayList<Composite> getCandidates1()` : fournit, lorsque c'est possible, une interaction considérée comme non fiable pouvant être testée dans le contexte courant. Si une interaction épistémique est nécessaire, elle sera définie et envoyée au module de décision.
- `public ArrayList<Composite> getCandidates2()` : détermine la liste des interactions considérées comme énactable.

Version 7.2

Cette version est destinée d'une part à tester le principe des signatures d'interaction dans des environnements plus complexes, et d'autre part, de valider le principe de la mémoire spatiale à l'aide d'une mémoire spatiale codée en dur. Dans cette version, les signatures sont dissociées des interactions composites.

Package spaceMemory

Composite

Description de l'interaction composite. Une interaction composite est constituée d'une séquence d'interactions primitives, d'une signature d'interaction et d'une valeur de satisfaction. Une *signature floue* est ajoutée pour permettre une simplification algorithmique.

Fonctions et méthodes principales :

- `public int checkSequence(PrimitiveInteraction[] seq)` : définit le résultat d'une interaction composite (1 : succès, -1 : échec, 0 : non utilisée).
- `public void setPrediction(float[] img)` : calcule la prédiction de succès de l'interaction composite dans le contexte *img*.
- `public void setPrediction1(int nbInteraction, float[][][] img)` : calcule la proximité globale de l'interaction composite dans le contexte courant *img*.
- `public void setPrediction2(ArrayList<Float[][][]> predictionMap)` : calcule la proximité globale de l'interaction composite dans l'ensemble des images prédites de la mémoire spatiale (liste *predictionMap*).

Decision

Il s'agit du mécanisme de décision qui détermine l'interaction primitive à énacter au cycle suivant. Le mécanisme vérifit dans un premier temps le résultat du cycle de décision suivant, puis, si l'énaction de l'interaction composite précédente s'est terminé ou a été interrompu, sélectionne l'interaction composite suivante. Le module teste tout d'abord si une interaction peut être testée dans le contexte courant. Sinon, le module récupère la liste des interactions énactables et sélectionne celle dont la valeur de satisfaction est la plus élevée. Les interactions primitives qui composent l'interaction composite sont ensuite énactées une à une.

Paramètres :

- float incertitude : seuil d'incertitude dans la prédiction de succès ou d'échec à partir duquel le mécanisme d'apprentissage est utilisé.

Fonctions et méthodes principales :

- `public PrimitiveInteraction decision()` : sélectionne les interactions composites et fourni l'interaction primitive intention à énacter au cycle de décision suivant.
- `public void check(PrimitiveInteraction enacted)` : Si l'interaction énactée n'est pas l'interaction intention, l'énaction de l'interaction est interrompue.
- `public void setLearning()` : active ou désactive le mécanisme d'apprentissage.

SpaceMemory

Module principal de la mémoire spatiale. Ce module met à jour les autres composants, détecte et enregistre les interactions composites découvertes, calcule les prédictions de succès ou d'échec des interactions composites et les variations de proximités globales que produiaient les interactions candidates. Il est possible de sauvegarder les interactions composites et leur signatures dans un fichier texte.

Paramètres :

- int timeSize : définit la longueur maximale des ineractions composites (par défaut, 1).
- float memoryCoef : coefficient d'influence de la mémoire spatiale, donnant l'influence de la mémoire spatiale par rapport aux valeurs de satisfaction des interactions.
- Int objectCoef : coefficient d'influence des objets, caractérisant l'affaiblissement de l'influence d'un objet lorsque sa distance augmente.
- boolean load : si 'true', les signatures d'interactions sont chargées depuis un fichier.

Fonctions et méthodes principales :

- `public void updateMemory(PrimitiveInteraction inter, float[] enacted)` : met à jour la mémoire spatiale, calcule les prédictions de succès ou d'échec des interactions et les variations de proximité globale produites par les interactions cndidates.

Signature

Description de la signature d'interaction. La signature est constituée d'un vecteur définissant les poids d'un neurone formel. On définit également les valeurs maximales de l'ensemble des poids, de l'ensemble des poids associées aux interactions primaires, et l'ensemble associé aux interactions secondaires, pour pouvoir normaliser les poids dans les systèmes d'affichage (l'agent n'utilise pas ces valeurs)

Paramètres :

- float learning_rate : coefficient d'apprentissage du neurone formel permettant l'apprentissage de la signature.

Fonctions et méthodes principales :

- `public float prediction(float[] img)` : calcule la prédiction de succès et d'échec de l'interaction dans le contexte *img*.
- `public void learn(float[] img, float output)` : intègre le résultat *output* de l'interaction composite dans le contexte du cycle de décision t-1 *img*.

BlurPattern

Une simplification algorithmique permettant d'obtenir la proximité globale d'un objet dans la mémoire spatiale. Cette simplification consiste à calculer à intervalle régulier les images des signatures d'interaction, plutôt que de les calculer à chaque cycle de décision.

Paramètres :

- int cycle : intervalle de temps entre deux constructions de la signature floue (par défaut, 10 cycles de décicion).

Fonctions et méthodes principales :

- `public void generateBlur(Signature s)` : calcule, tous les *cycles* cycles de décision, la projection de la signature *s* dans l'espace.

Version 7.2.2.2

Cette version est destinée à tester le principe des signatures d'interaction avec un système interactionnel complet, c'est à dire sans simplification sur les interactions visuelles. Le but est d'observer l'émergence de relations entre les signatures d'interactions, qui seront utilisés dans les versions suivantes pour construire une mémoire de l'espace. Cette version ne propose pas de mécanisme d'exploitation de la mémoire spatiale et ne permet donc pas l'émergence de comportments.

Package agnosticMemory

Decision

Il s'agit du mécanisme de décision qui détermine l'interaction primitive à énacter au cycle suivant. Le mécanisme de sélection détermine l'interaction pour laquelle la certitude de succès et d'échec est la plus faible (en valeur absolue). Dans le cas d'une interaction secondaire, l'interaction primaire associée doit être considérée comme énactable, afin de s'assurer que l'interaction secondaire puisse être testée.

Fonctions et méthodes principales :

- `public PrimitiveInteraction decision()` : sélectionne l'interaction à tester au cycle de décision suivant.

EnvironmentContext

Ce module construit le contexte interactionnel sous forme d'un vecteur à partir de la liste des interactions énactées. Il est également possible de construire le contexte augmenté (obtenu en ajoutant les signatures des interactions considérées comme enactables) et le contexte prédit (formé par les interactions prédites comme un succès).

Paramètres :

- boolean predict : si 'true', le contexte prédit est construit à chaque cycle de décision.
- Boolean improve : si 'true', le contexte augmenté est construit à chaque cycle de décision.

Fonctions et méthodes principales :

- `public void updateContext(PrimitiveInteraction inter, float[] enacted)` : construit le contexte interactionnel sous la forme d'un vecteur à partir des interactions énactées, et renforce les signatures des interactions énactées. En fonction des paramètres, les contextes augmenté et prédit sont également construits.

Signature

Description de la signature d'interaction. La signature est constituée d'un vecteur définissant les poids d'un neurone formel. On définit également les valeurs maximales de l'ensemble des poids associées aux interactions primaires, et l'ensemble associé aux interactions secondaires, pour pouvoir normaliser les poids dans les systèmes d'affichage (l'agent n'utilise pas ces valeurs). Une signature est considérée comme fiable si elle a permis 5 prédicitons correctes consécutives et si sa dernière prédiction est supperieure à un certain seuil.

Fonctions et méthodes principales :

- `public float prediction(float[] img)` : calcule la prédiction de succès et d'échec de l'interaction dans le contexte *img*.
- `public void learn(float[] img, float output)` : intègre le résultat *output* de l'interaction composite dans le contexte du cycle de décision t-1 *img*.
- `public boolean isAccurate()` : détermine si la signature est fiable.
- `public void setPrediction(float[] img)` : calcule et enregistre la certitude de succès et d'échec dans le contexte *img*.

SignatureList

Structure qui gère l'ensemble des interactions. Il est possible d'enregistrer et charger les signatures d'interactions dans un fichier.

Paramètres :

- boolean load : si 'true', les signatures d'interactions sont chargées depuis un fichier.

Fonctions et méthodes principales :

- `public void learn(float[] input, float[] output, PrimitiveInteraction enacted)` : permet l'apprentissage et le renforcement des signatures des interactions énactées (*output*) dans le contexte *input*.
- `public void setPredictions(float[] img)` : calcule et enregistre les prédictions de succès et d'échec de chaque interaction.
- `public boolean isValid(int id)` : détermine si l'interaction désignée par *id* est valide. Une interaction primaire est toujours valide. Une interaction secondaire est valide si sa signature est fiable et si son interaction associée est prédite comme un succès.
- `public PrimitiveInteraction getInteractionOfIndex(int id)` : retourne l'interaction désignée par l'identifiant *id*.
- `public void save()` : sauvegarde les interactions dans un fichier
- `public void load()` : charge les signatures depuis un fichier.

Version 7.2.8

Cette version est destinée à tester les mécanismes de construction de la mémoire spatiale. Elle se base sur l'utilisation des signatures d'interactions obtenues avec la version 7.2., c'est pourquoi une partie des mécanismes utilisés dans la version 7.2 est utilisé ici (package *spaceMemory*) afin de faire la transition avec les mécanismes de la mémoire agnostique (package *agnosticmemory*). L'apprentissage des signatures de lieu et de présence s'effectue en plusieurs étapes. Il est donc nécessaire d'enregistrer les signatures, puis de changer les paramêtres pour passer d'une étape à l'autre.

Il est nécessaire de respecter l'ordre suivant :

- Apprentissage des signatures de lieu (Decision.learning=true, AgnosticMemory.learn_place=true). Il est nécessaire d'utiliser le partage des signatures de façon régulière (tous les 10 000 pas environ).
- Apprentissage des signatures de présence (Decision.learning=true, AgnosticMemory.learn_place=false)
- Exploitation de la mémoire spatiale (Decision.learning=false)

Package agnosticMemory

AgnosticMemory

Module principal de la mémoire spatiale agnostique. Ce module permet la détection des instances d'objet et la mise à jour des signatures de lieu et de présence. Il est possible de sauvegarder les interactions composites et leur signatures dans un fichier texte.

Paramètres :

- int range : longueur maximale des lieux composites (par défaut à 3).
- boolean load : si 'true', les signatures d'interactions sont chargées depuis un fichier.
- int share : permet d'activer les possibilitée de partage des signatures de lieu (1 -> détection des incohérences, 2-> copie des signatures de lieu)
- boolean learn_place : si 'true', le mécanisme d'apprentissage sera optimisé pour l'apprentissage des signatures de lieu (et l'apprentissage des signatures de présence sera désactivé), et pour les signatures de présence sinon.
- float memoryCoef : coefficient d'influence de la mémoire spatiale, donnant l'influence de la mémoire spatiale par rapport aux valeurs de satisfaction des interactions.
- int objectCoef : coefficient d'influence des objets, caractérisant l'affaiblissement de l'influence d'un objet lorsque sa distance augmente.

Fonctions et méthodes principales :

- `public void updateMemory(PrimitiveInteraction inter, float[] enacted)` : met à jour la mémoire spatiale,détecte les instances d'objet et permet l'apprentissage des signatures.

Interaction

Description d'une interaction utilisée par la mémoire agnostique. Ces interactions diffèrent de celles utilisées par la mémoire codée en dur. L'interaction comporte une signature, une signature floue, ainsi que des informations sur le rapprochement qu'elle produit.

Fonctions et méthodes principales :

- `public void learnMovement(Agent agent)` : mesure les variations de mouvement produites par l'énaction de l'interaction.
- `public void setPatterns(Signature s,Agent agent)` : copie la signature de l'interaction correspondante de la mémoire codée en dur, et génère la signature floue.

Object

Description d'une instance d'objet utilisé par la mémoire spatiale. Une instance est caractérisé par l'intéraction affordé par le type d'objet, un identifiant, la transformation (position) où elle a été détectée, ainsi que la liste des lieux qui caractérisent sa position. Nous utilisons ici deux listes de lieux, la première contient les lieux dotés d'une forte certitude, et est mise à jour uniquement en mettant à jour les lieux. La seconde réduit le seuil de fiabilité, et permet l'ajout de lieux évoqués.

Fonctions et méthodes principales :

- `public void update(PrimitiveInteraction inter, ArrayList<Place> list)` : met à jour la liste des lieux.
- `public void evoke(ArrayList<Place> list)` : détecte les lieux évoqués.
- `public void defineMap()` : construit une carte de certitude à partir des signatures de lieu des lieux stockés en mémoire.
- `public void computeMovements(ArrayList<Interaction> list)` : détermine le rapprochement que provoqueraient les interactions sur cette instance.

Place

Desription d'un lieu. Un lieu est caractérisé par l'interaction et la distance qui définissent son lieu final, l'interaction affordée par le type d'objet que peut suivre ce lieu, un path, une signature de lieu et une signature de présence.

Fonctions et méthodes principales :

- `public void learnPlaceSignature(PrimitiveInteraction[] timeline , boolean[][][] map, float[][][][] context)` : met à jour la signature de lieu si le lieu a été énacté (i.e. Il s'inscrit dans la timeline), à partir du résultat *map* et du contexte *context*.
- `public void learnPresenceSignature(PrimitiveInteraction[] timeline , boolean[][][] map, float[][][][] context)` : met à jour la signature de présence si le lieu a été énacté (i.e. Il s'inscrit dans la timeline), à partir du résultat *map* et du contexte *context*.

PlaceSignature

Description de la signature de lieu.

Fonctions et méthodes principales :

- `public float prediction(float[][] img, boolean positive)` : calcule la prédiction de présence ou d'absence d'un objet dans le lieu dans le contexte de position *img*. Si *positive* est à 'true', on ne tiendra compte que des poids positifs.
- `public void learn(float[][] img, float output)` : intègre le résultat *output* de l'interaction composite dans le contexte du cycle de décision t-1 *img*.

PlaceContext

Ensemble des contextes de lieux formés par les instances d'objet stockées en mémoire. Un contexte de lieux est constitué d'un vecteur donnant la présence ou l'absence des lieux dans la description d'une instance d'objet.

Fonctions et méthodes principales :

- `public void updateContext(ArrayList<Object> objectList)` : met à jour les contextes de lieux d'après une liste *objectList* d'instances d'objet.

PresenceSignature

Description de la signature de présence.

Fonctions et méthodes principales :

- `public float prediction(float[][] img, boolean positive)` : calcule la prédiction de présence ou d'absence d'un objet dans le lieu dans le contexte de lieux *img*. Si *positive* est à 'true', on ne tiendra compte que des poids positifs.
- `public void learn(float[][][] img, float output, int firstIndex)` : intègre le résultat *output* de l'interaction composite dans le contexte du cycle de décision t-1 *img*. *firstIndex* est l'index du premier élément du path.

Package spaceMemory

Composite

Description de l'interaction composite. Une interaction composite est constituée d'une séquence d'interactions primitives, d'une signature d'interaction et d'une valeur de satisfaction.

Fonctions et méthodes principales :

- `public int checkSequence(PrimitiveInteraction[] seq)` : définit le résultat d'une interaction composite (1 : succès, -1 : échec, 0 : non utilisée).
- `public void setPrediction(float[] img)` : calcule la prédiction de succès de l'interaction composite dans le contexte *img*.

Decision

Il s'agit du mécanisme de décision qui détermine l'interaction primitive à énacter au cycle suivant. Dans le cas d'un apprentissage, le mécanisme sélectionne une interaction permettant de tester le lieu pour lequel la certitude de présence (de position ou de lieu, en fonction des paramêtres de la mémoire spatiale agnostique), est la plus faible. Dans le cas d'une exploitation de la mémoire spatiale, le mécanisme cherchera à maximiser la valeur de satisfaction globale.

Paramètres :

- boolean learning : si 'true', le système utilise le mécanisme d'apprentissage, sinon, il utilise le mécanisme d'exploitation de la mémoire spatiale.

Fonctions et méthodes principales :

- `public PrimitiveInteraction decision()` : sélectionne l'interaction à tester au cycle de décision suivant.
- `public void check(PrimitiveInteraction enacted)` : Si l'interaction énactée n'est pas l'interaction intention, l'énaction de l'interaction est interrompue.

EnvironmentMemory

Ce module construit le contexte interactionnel sous forme d'un vecteur à partir de la liste des interactions énactées. Il contient également la liste des interactions composites apprises par l'agent.

Paramètres :

- int timeSize : définit la longueur maximale des ineractions composites (par défaut à 1).

Fonctions et méthodes principales :

- `public void updateContext(PrimitiveInteraction inter, float[] enacted)` : construit le contexte interactionnel sous la forme d'un vecteur à partir des interactions énactées, et renforce les signatures des interactions énactées. En fonction des paramètres, les contextes augmenté et prédit sont également construits.
- `public void save()` : sauvegarde les interactions dans un fichier
- `public void loadFile()` : charge les signatures depuis un fichier.

Signature

Description de la signature d'interaction. La signature est constituée d'un vecteur définissant les poids d'un neurone formel. On définit également les valeurs maximales de l'ensemble des poids associées aux interactions primaires, et l'ensemble associé aux interactions secondaires, pour pouvoir normaliser les poids dans les systèmes d'affichage (l'agent n'utilise pas ces valeurs). Une signature est considérée comme fiable si elle a permis 5 prédicitons correctes consécutives et si sa dernière prédiction est supperieure à un certain seuil.

Paramètres :
- float learning_rate : coefficient d'apprentissage du neurone formel permettant l'apprentissage de la signature.

Fonctions et méthodes principales :

- `public float prediction(float[] img)` : calcule la prédiction de succès et d'échec de l'interaction dans le contexte *img*.
- `public void learn(float[] img, float output)` : intègre le résultat *output* de l'interaction composite dans le contexte du cycle de décision t-1 *img*.

Version 7.3

Cette version implémente les mécanismes de la version 7.2 sur un robot physique, dans le but de tester ces mécanismes dans un environnement bruité.

Package spaceMemory

Composite

Description de l'interaction composite. Une interaction composite est constituée d'une séquence d'interactions primitives, d'une signature d'interaction et d'une valeur de satisfaction. Une *signature floue* est ajoutée pour permettre une simplification algorithmique.

Fonctions et méthodes principales :

- `public int checkSequence(PrimitiveInteraction[] seq)` : définit le résultat d'une interaction composite (1 : succès, -1 : échec, 0 : non utilisée).
- `public void setPrediction(float[] img)` : calcule la prédiction de succès de l'interaction composite dans le contexte *img*.
- `public void setPrediction1(int nbInteraction, float[][][] img)` : calcule la proximité globale de l'interaction composite dans le contexte courant *img*.
- `public void setPrediction2(ArrayList<Float[][][]> predictionMap)` : calcule la proximité globale de l'interaction composite dans l'ensemble des images prédites de la mémoire spatiale (liste *predictionMap*).

Decision

Il s'agit du mécanisme de décision qui détermine l'interaction primitive à énacter au cycle suivant. Le mécanisme vérifit dans un premier temps le résultat du cycle de décision suivant, puis, si l'énaction de l'interaction composite précédente s'est terminé ou a été interrompu, sélectionne l'interaction composite suivante. Le module teste tout d'abord si une interaction peut être testée dans le contexte courant. Sinon, le module récupère la liste des interactions énactables et sélectionne celle dont la valeur de satisfaction est la plus élevée. Les interactions primitives qui composent l'interaction composite sont ensuite énactées une à une.

Paramètres :

- float incertitude : seuil d'incertitude dans la prédiction de succès ou d'échec à partir duquel le mécanisme d'apprentissage est utilisé.

Fonctions et méthodes principales :

- `public PrimitiveInteraction decision()` : sélectionne les interactions composites et fourni l'interaction primitive intention à énacter au cycle de décision suivant.
- `public void check(PrimitiveInteraction enacted)` : Si l'interaction énactée n'est pas l'interaction intention, l'énaction de l'interaction est interrompue.
- `public void setLearning()` : active ou désactive le mécanisme d'apprentissage.

SpaceMemory

Module principal de la mémoire spatiale. Ce module met à jour les autres composants, détecte et enregistre les interactions composites découvertes, calcule les prédictions de succès ou d'échec des interactions composites et les variations de proximités globales que produiaient les interactions candidates. Il est possible de sauvegarder les interactions composites et leur signatures dans un fichier texte.

Paramètres :

- int timeSize : définit la longueur maximale des ineractions composites (par défaut, 1).
- float memoryCoef : coefficient d'influence de la mémoire spatiale, donnant l'influence de la mémoire spatiale par rapport aux valeurs de satisfaction des interactions.
- Int objectCoef : coefficient d'influence des objets, caractérisant l'affaiblissement de l'influence d'un objet lorsque sa distance augmente.
- boolean load : si 'true', les signatures d'interactions sont chargées depuis un fichier.

Fonctions et méthodes principales :

- `public void updateMemory(PrimitiveInteraction inter, float[] enacted)` : met à jour la mémoire spatiale, calcule les prédictions de succès ou d'échec des interactions et les variations de proximité globale produites par les interactions cndidates.

Signature

Description de la signature d'interaction. La signature est constituée d'un vecteur définissant les poids d'un neurone formel. On définit également les valeurs maximales de l'ensemble des poids, de l'ensemble des poids associées aux interactions primaires, et l'ensemble associé aux interactions secondaires, pour pouvoir normaliser les poids dans les systèmes d'affichage (l'agent n'utilise pas ces valeurs)

Paramètres :

- float learning_rate : coefficient d'apprentissage du neurone formel permettant l'apprentissage de la signature.

Fonctions et méthodes principales :

- `public float prediction(float[] img)` : calcule la prédiction de succès et d'échec de l'interaction dans le contexte *img*.
- `public void learn(float[] img, float output)` : intègre le résultat *output* de l'interaction composite dans le contexte du cycle de décision t-1 *img*.

BlurPattern

Une simplification algorithmique permettant d'obtenir la proximité globale d'un objet dans la mémoire spatiale. Cette simplification consiste à calculer à intervalle régulier les images des signatures d'interaction, plutôt que de les calculer à chaque cycle de décision.

Paramètres :

- int cycle : intervalle de temps entre deux constructions de la signature floue (par défaut, 10 cycles de décicion).

Fonctions et méthodes principales :

- `public void generateBlur(Signature s)` : calcule, tous les *cycles* cycles de décision, la projection de la signature *s* dans l'espace.

VII Afficheurs

Versions	2.8.5	2.8.6	2.8.6.1	7.2	7.2.2.1	7.2.8	7.3
EfficiencyPanel	X	X	X	X		X	X

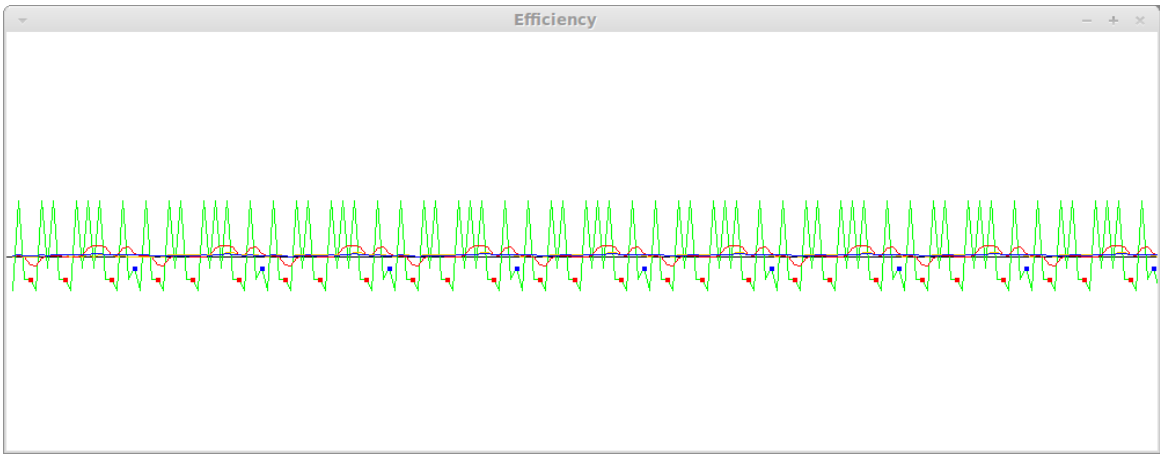

Figure 4 : affichage des valeurs de satisfaction au cours du temps (versions 2.8.). La courbe verte indique la valeur de satisfaction à chaque cycle de décision. La courbe rouge la moyenne des 10 derniers cycles, la courbe jaune la moyenne des 50 derniers cycles et la courbe bleu la moyenne des 100 derniers cycles. Dans les versions 2.8.*, les points bleux indiquent des interactions "toucher un espace vide" et les points rouge des interactions "toucher un mur".*

SpaceFrame	X	X	X	X		X	X

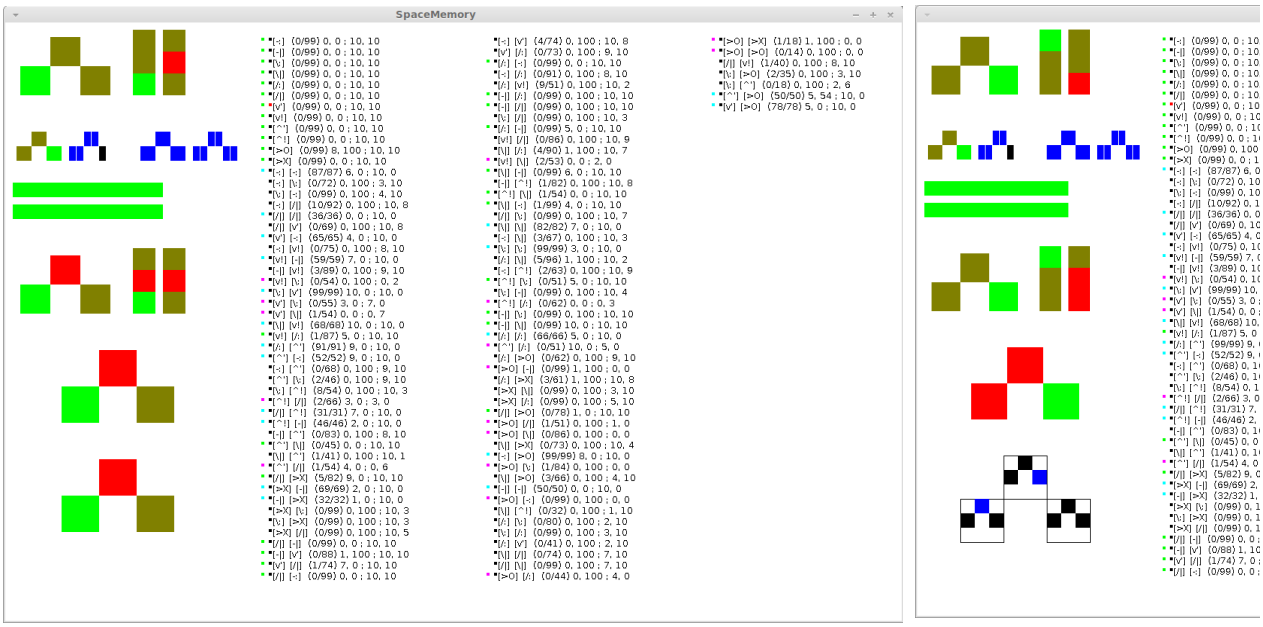

Figure 5 : à gauche : affichage de la mémoire spatiale des versions 2.8.. Dans la partie gauche, de haut en bas sont affichés le contexte interactionnel courant E_t et les contextes au pas t-1 à t-n (sous forme de vecteurs), la signature l'interaction composite sélectionnée et les entrées déconnectées (en bleu), le nombre de prédictions correctes consécutives de succès et d'échec, le premier contexte augmenté E'_t et E'_{t-1} à E'_{t-n}, et les contextes augmentés E''_t et E'''_t. Dans la partie droite sont listées les interactions composites, que l'on peut sélectionner (l'interaction sélectionnée est marquée par un carré rouge). Dans les versions 2.8.6 et 2.8.6.1, les interactions fiables sont précédées d'un carré vert, les interactions toujours vraies, d'un carré cyan et les interactions décorrélées, d'un carré rouge. À droite : dans la version 2.8.6, il est possible de calculer et afficher les signatures de mouvmenent à la place du contexte E'''. Cette signature donne le déplacement d'une interaction vers une autre.*

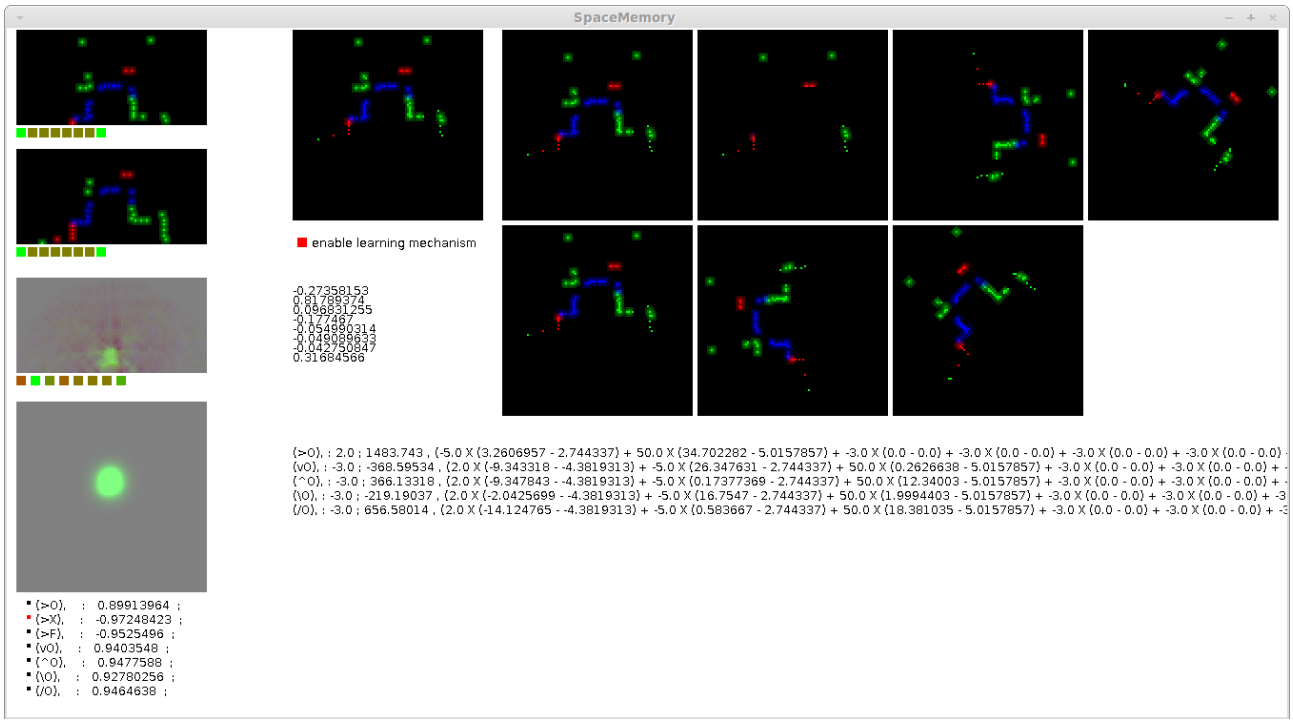

Figure 6 : affichage de la mémoire spatiale des versions 7.2 et 7.3. Dans la partie gauche, de haut en bas : le contexte courant E_t et le contexte précédent E_{t-1} mis sous une forme lisible, la signature de l'interaction sélectionnée et sa signature floue, et la liste des interactions, suivies de leur certitude de succès. L'interaction sélectionnée est désignée par un carré rouge. À droite : la mémoire spatiale codée en dur, les sept mémoires prédites, un interrupteur permettant d'activer et désactiver le mécanisme d'apprentissage, la valeur des poids associés aux interactions primaires et au biais de l'interaction sélectionnée, et enfin le détail des calculs des variations de proximité globale.

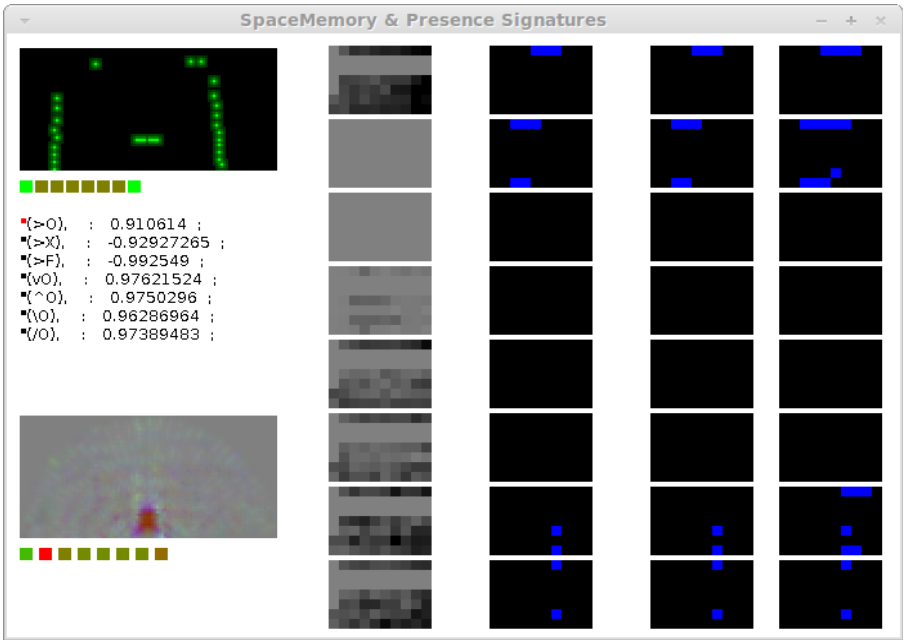

Figure 7 : affichage de la mémoire spatiale de la version 7.2.8. L'affichage se résume au contexte interactionnel E_t, à la liste des interactons, suivies de la certitude de succès, et la sigature de l'interaction sélectionnée. À droite, la signature de présence du lieu sélectionné dans l'affichage des lieux, le contexte de lieu est les contextes de lieu augmentés de l'instance d'objet sélectionnée dans l'affichage des lieux.

ColliculusPanel	**X**	**X**	**X**

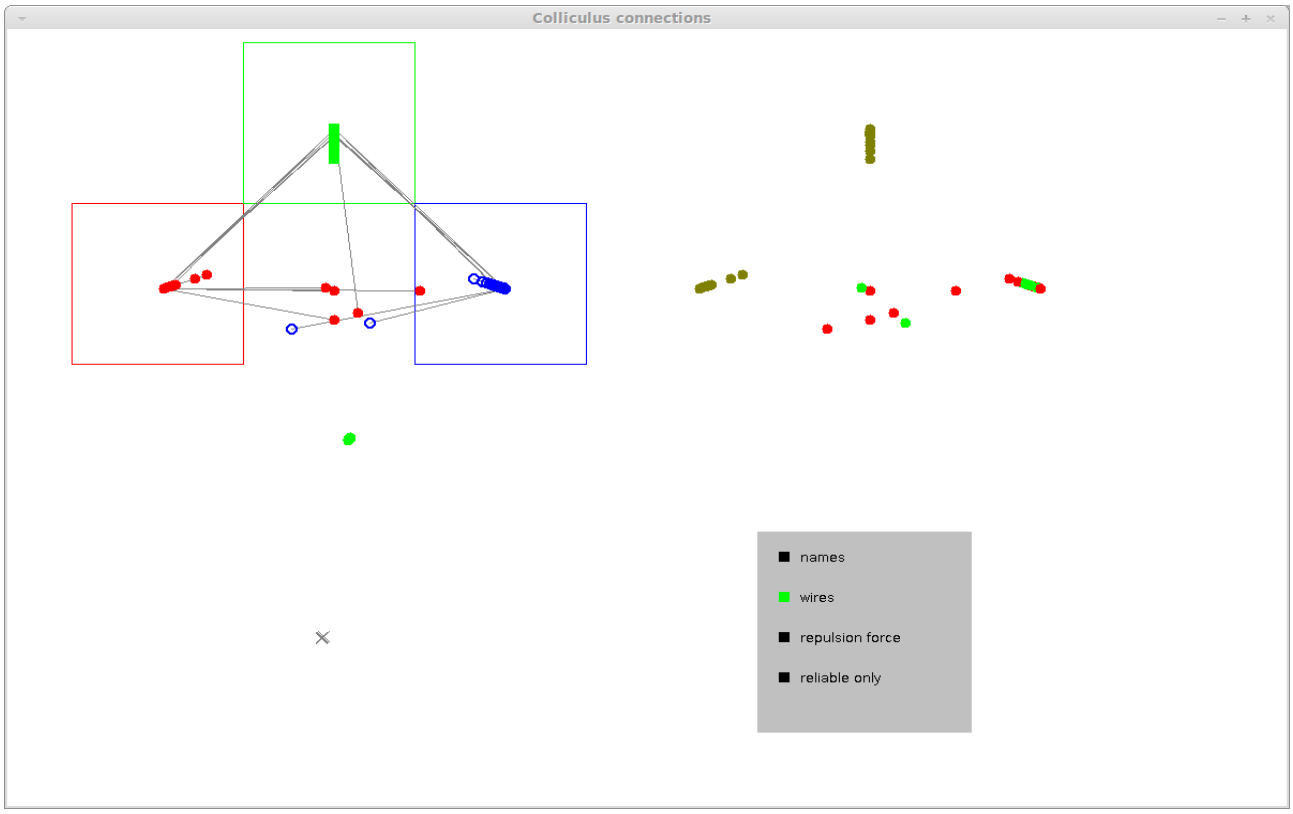

Figure 8 : affichage des bundles (versions 2.8.). En haut à gauche, les bundles de position : les interactions, représentées par des points (rouge pour ceux associés à un objet à gauche, vert pour devant et bleu pour la droite), sont positionné en fonction des poids de leur signatures. Un lien relie les interactions composites conposées d'une même interaction finale. En bas à gauche, les bundles d'objets. Les interactions associés aux murs sont représentés par un point vert et celles associées aux espaces vides par une croix grise. Une force d'attraction importante est appliquée entre les interactions composées d'une même interaction finale, puisque caractérisant un même objet par définition. Une force d'attraction positive ou négative est ensuite appliquée entre chaque paire d'interaction, en fonction de la ressemblance ou de l'incompatibilité de leurs signatures. En haut à droite, l'activité des interactions : vert pour un succès, rouge pour un échec. En bas à droite, l'interface, permettant d'afficher ou nom les noms des interactions et les liens, de générer une force de répulsion pour l'affichage des bundles d'objet, de façon à séparer les interactions, et, dans le cas des versions 2.8.6 et 2.8.6.1, de masquer les interactions non fiables.*

PatternDataPanel X X

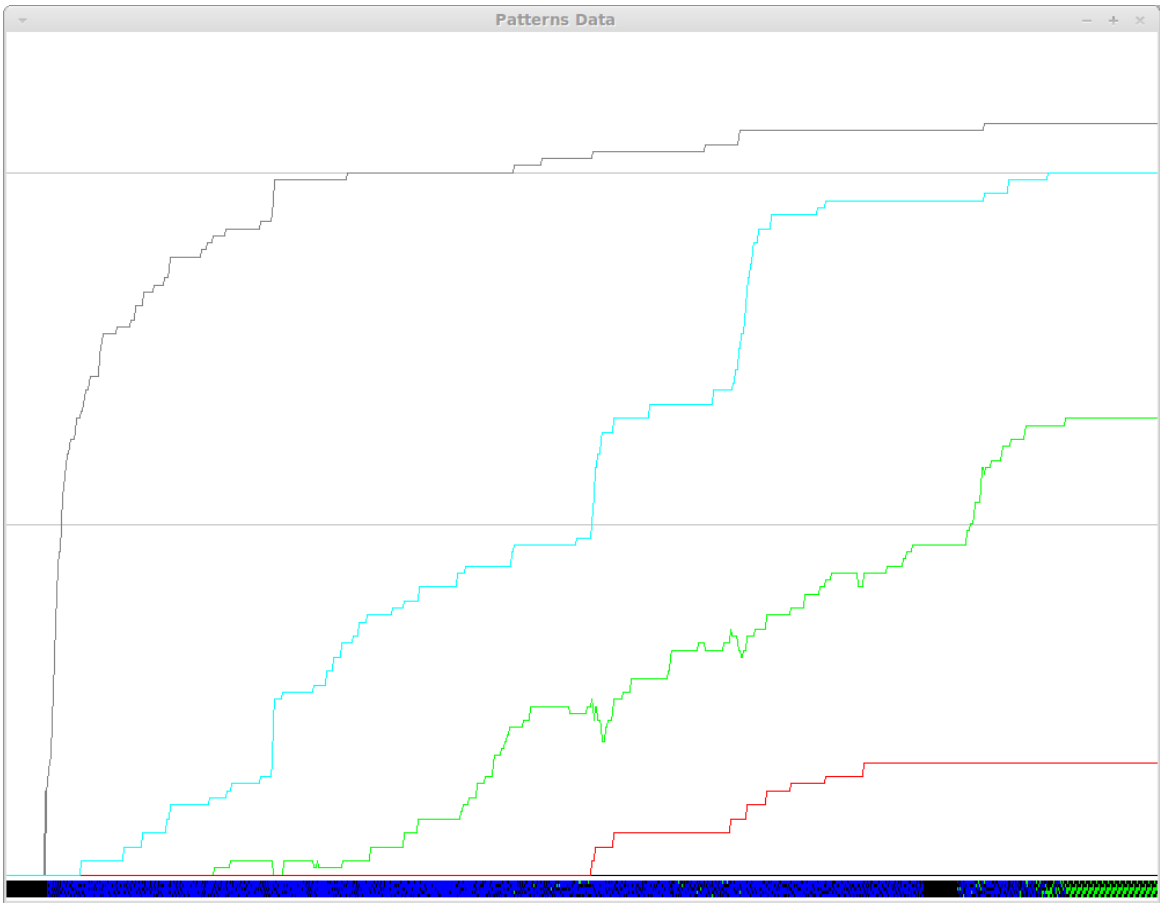

Figure 9 : évolution du nombre d'interactions composites au cours du temps (versions 2.8.6 et 2.8.6.1). Chaque unitée de largeur (pixel) représente 5 cycles de décision. Des graduations sont marquées toutes les 50 interactions. La courbe noie indique le nombre d'interactions composites apprises par l'agent. La courbe vert montre l'e nombre d'interactions considérées comme fiables et la courbe rouge le nombre d'interactions dé-corrélées (signature "vide"). La courbe cyan affiche le nombre d'interactions dont la signature est correcte (d'un point de vue extérieur). En bas, chaque colonne indique le mécanisme de décision utilisé, pour les 5 cycles correspondants : bleu : mécanisme d'apprentissage, vert : mécanisme d'exploitation, noir : interaction épistémique.

ProbePanel X X X

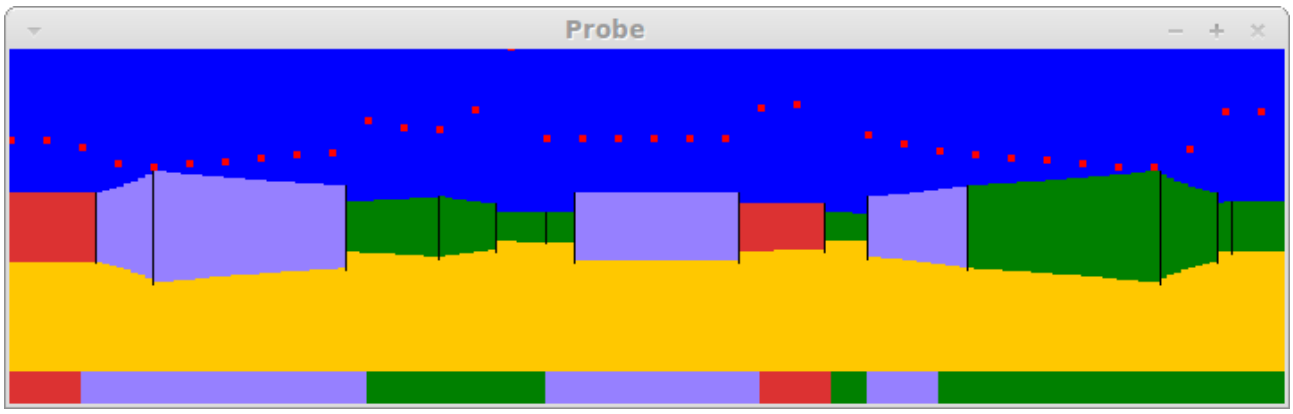

Figure 10 : rendu du système visuel de l'agent (versions 7.2.). En haut, le rendu 3d de l'environnement. Les points rouges indiquent la valeur du flux optique simulé en chaque pixel de la rétine, qui dépend de la distance. En bas, la couleur détectée par chaque pixel. La rétine utilisée par l'agent comporte 36 pixels.*

PredictionPanel	X	X	X

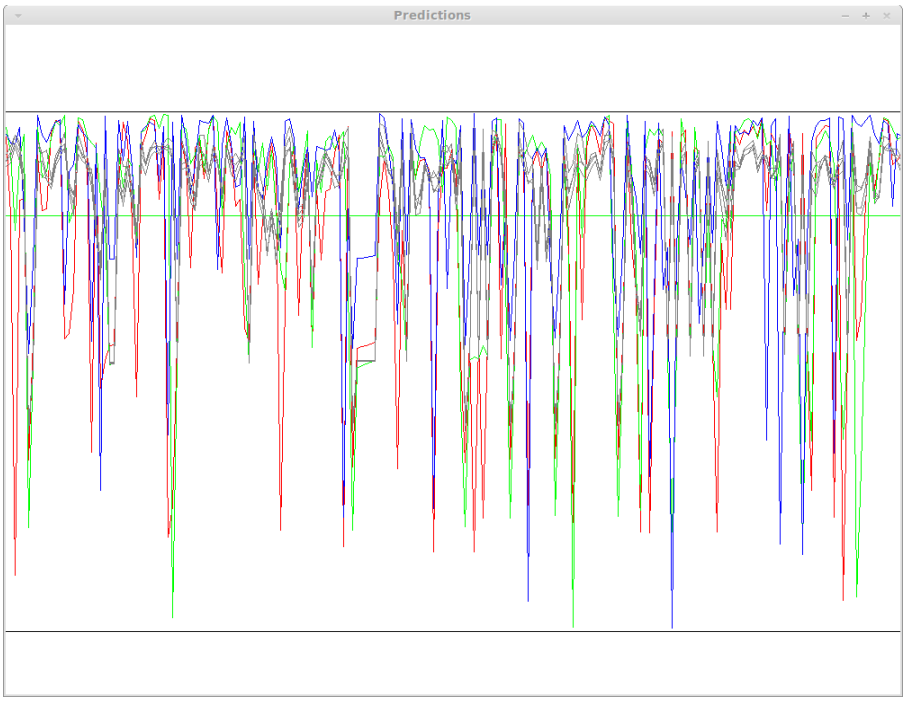

Figure 11 : prédictions des interactions à chaque cycle de décision (versions 7.2, 7.2.8 et 7.3). Chaque courbe indique la valeur absolue des prédictions de chaque interaction (rouge : avancer, vert : se cogner, bleu : manger. Les quatre interactions tourner sont représentées par les courbes grises). La graduation verte indique le seuil de certitude en dessous duquel le mécanisme d'apprentissage est utilisé.

SignaturePanel	X

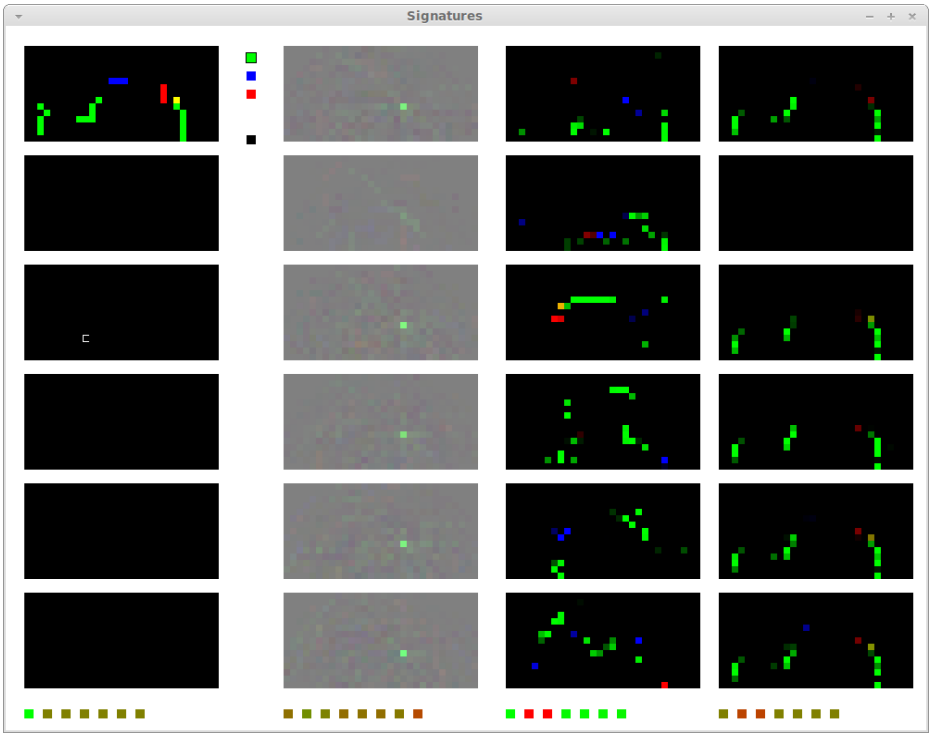

Figure 12 : affichage des signatures d'interaction et des contextes complété et prédit (version 7.2.2.1). De gauche à droite, le contexte interactionnel, permettant également de sélectionner une interaction à afficher, la signature de l'interaction sélectionnée, le contexte prédit et le contexte complété. Si une interaction visuelle est sélectionnée, elle est marquée par un carré blanc (visible dans le troisième groupe en partant du haut). Les contextes et la signature sont affichés sous une forme facilitant la lecture.

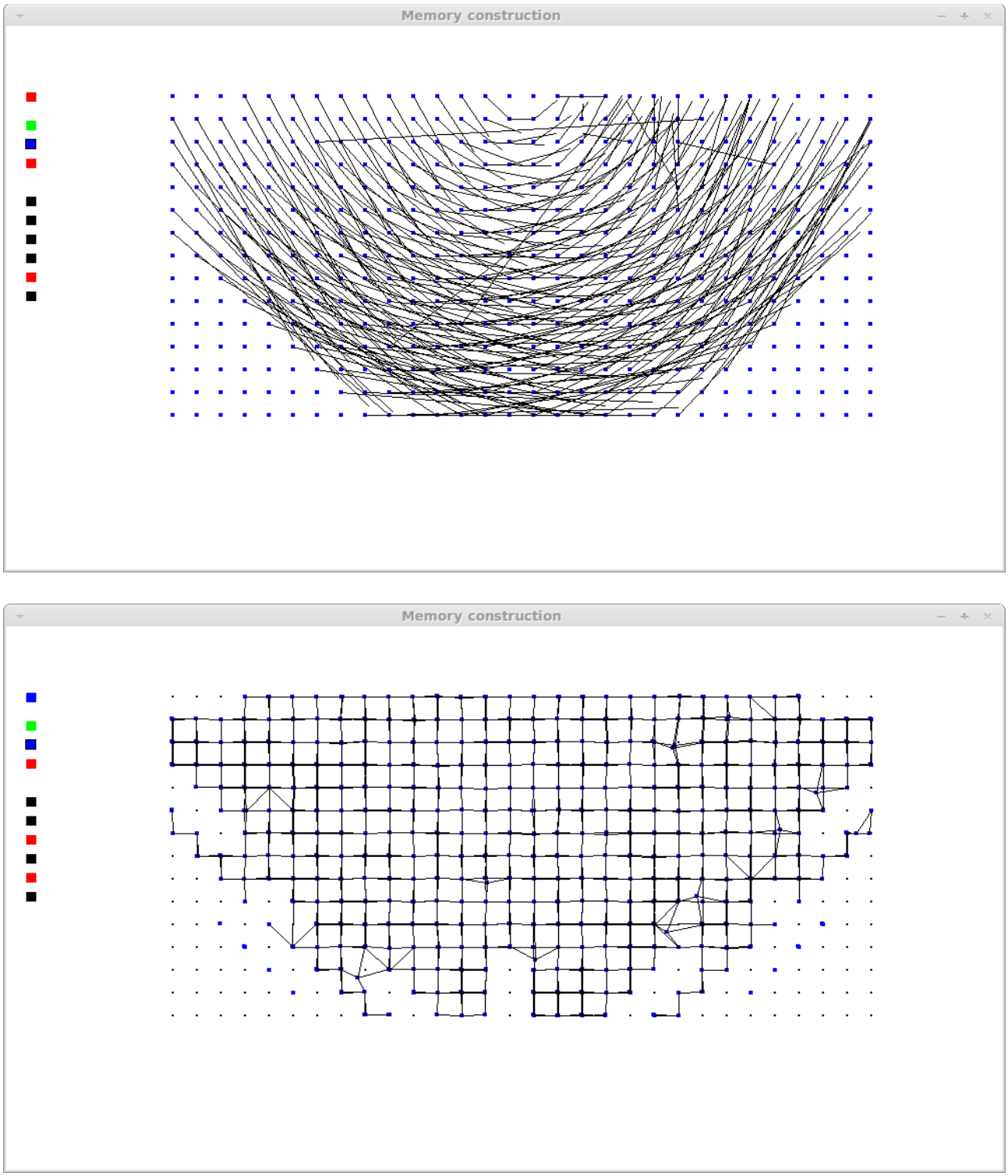

Figure 13 : Affichage de la structure géométrique émergente de l'espace observable (version 7.2.2.1). En haut, l'affichage montre la différence spatiale entre une interaction et le barycentre de sa signature. En bas, les interactions désignées par une même signature se rapprochent. On place les interactions associées à avancer pour définir une grille de référence sur laquelle les autres interactions vont se positionner. À gauche, l'interface. Le premier bouton permet de passer d'un mode d'affichage à l'autre. Il est également possible de choisir les interactions secondaires à afficher en sélectionnant leur couleur (triplet de boutons) et l'interaction primaire associée (plusieurs groupes peuvent être affichés simultanément).

MovementPanel **X**

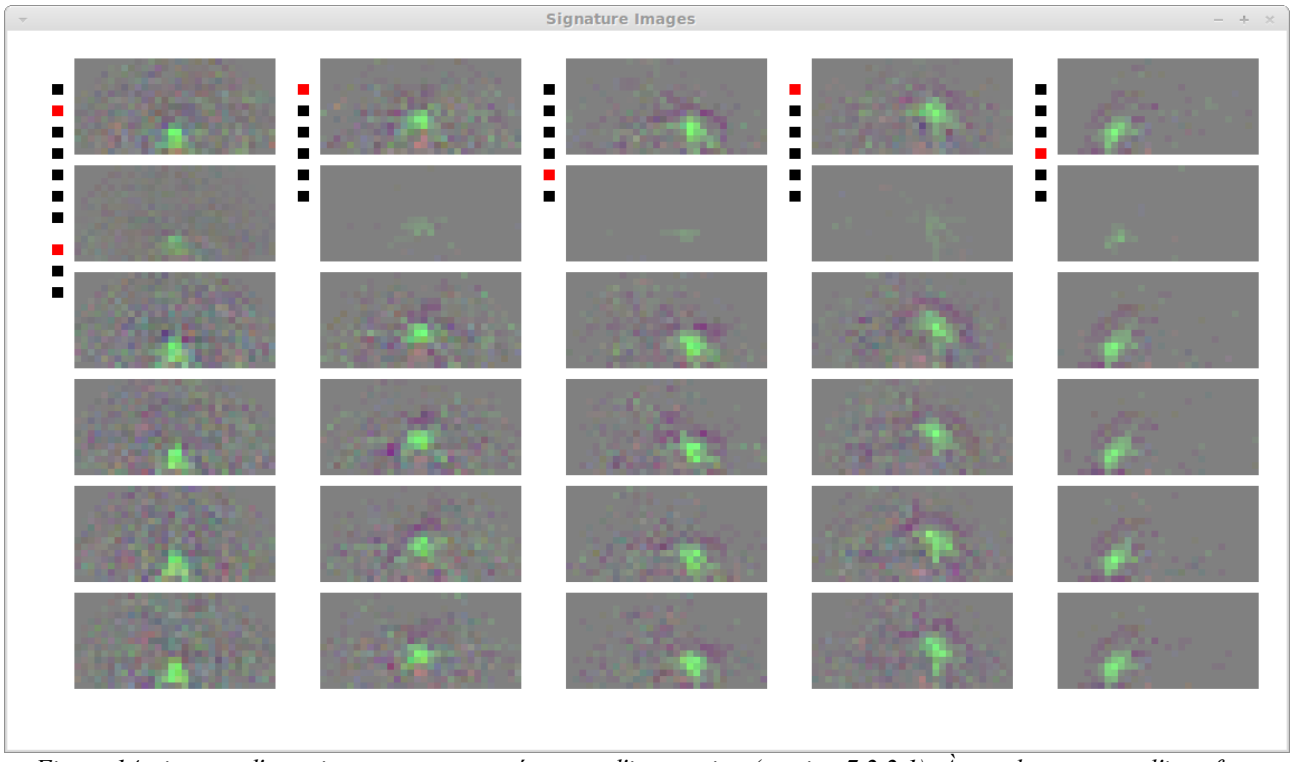

Figure 14 : images d'une signature par une séquence d'interaction (version 7.2.2.1). À gauche se trouve l'interface permettant de sélectionner une interaction primitive (les septs boutons), ou une interaction visuelle (en sélectionnant une couleur et une position dans le premier groupe d'interaction en haut à gauche. La première colonne affiche la signature de l'interaction sélectionnée. Les colonnes suivantes affichent les images successives de cette signature par la transformation sélectionnée (par l'un des six boutons). Dans cet exemple, les transformations successives sont : avancer, tourner à droite de 45°, avancer et tourner à gauche de 90°.

PlacePanel **X**

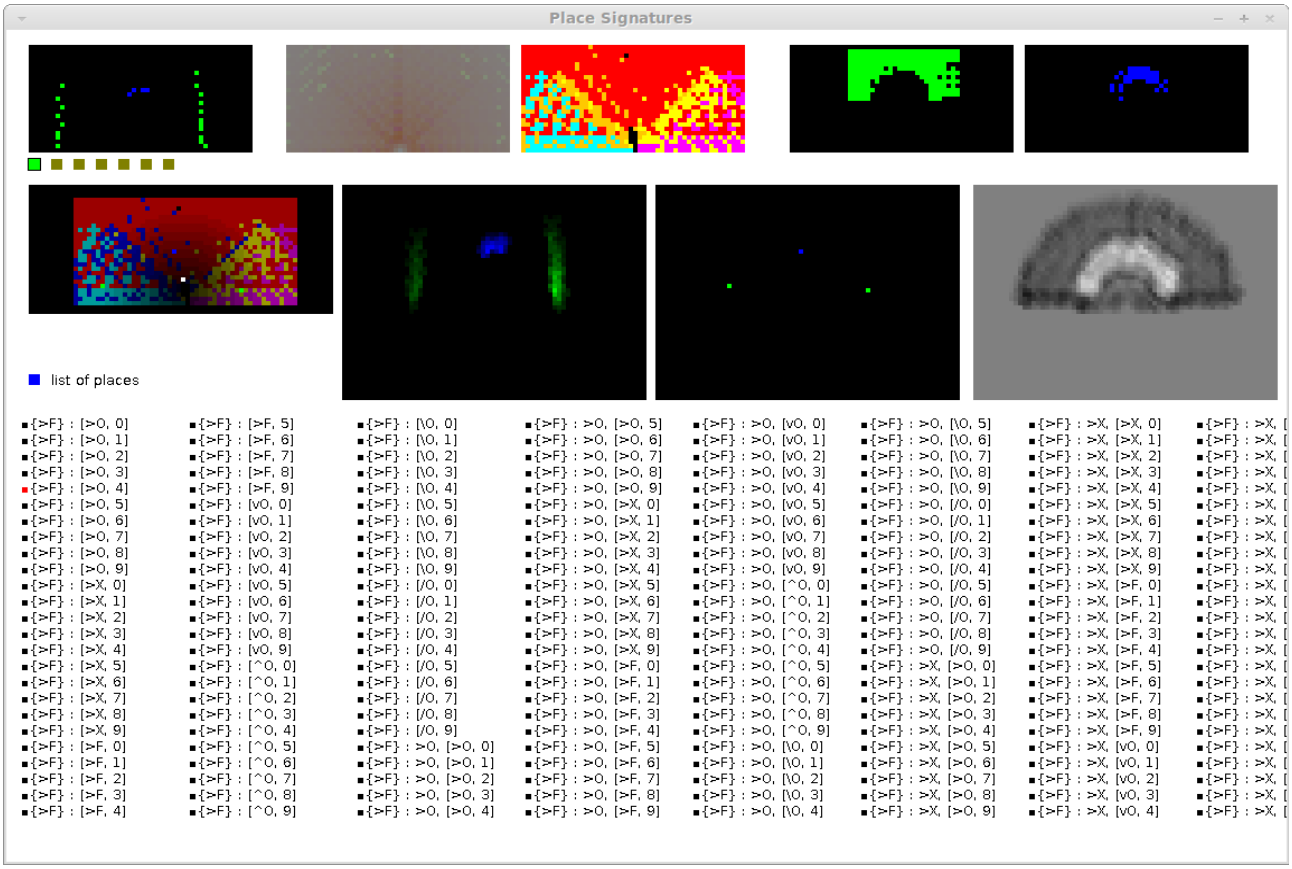

Figure 15 : affichage des lieux, signatures de lieu et contenu de la mémoire spatiale agnostique (mode affichage des lieux) (version 7.2.8). Sur la première ligne : le contexte interactionnel, la signature floue de l'interaction primaire sélectionnée, la carte donnant à chaque position l'interaction permettant un rapprochement maximal, les lieux dans lesquels ont été détecté des objets affordant se cogner et manger. Seconde ligne : découpage des lieux primitifs, bouton permettant d'accéder à l'affichage des objets, carte des instances d'objets, carte indiquant les instances d'objet retenues et signature de lieu du lieu sélectionné. En bas : liste des lieux utilisés. Il est possible de sélectionner un lieu pour afficher sa signature.

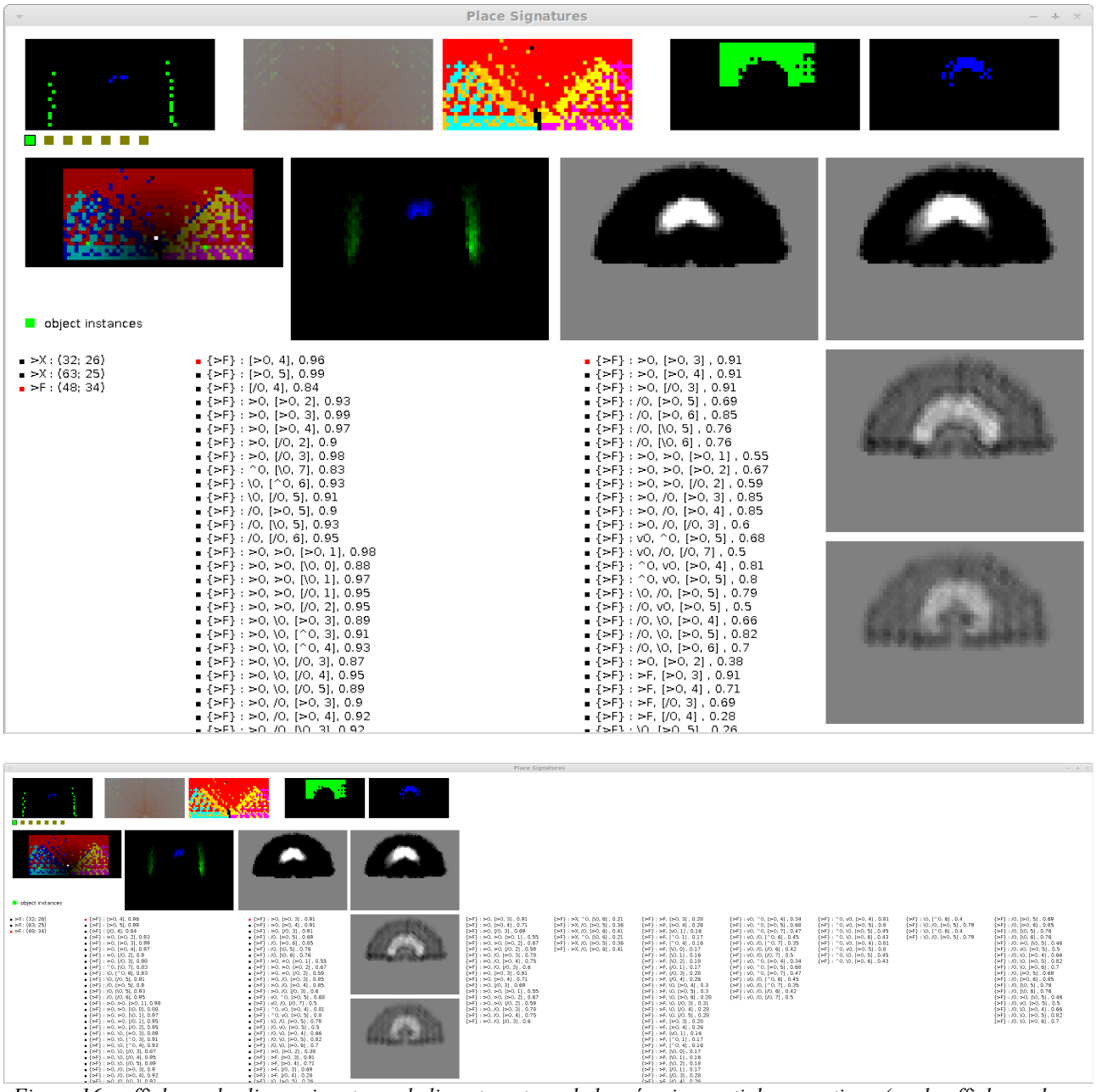

Figure 16 : affichage des lieux, signatures de lieu et contenu de la mémoire spatiale agnostique (mode affichage des objets) (version 7.2.8). Sur la première ligne : le contexte interactionnel, la signature floue de l'interaction primaire sélectionnée, la carte donnant à chaque position l'interaction permettant un rapprochement maximal, les lieux dans lesquels ont été détecté des objets affordant se cogner et manger. Seconde ligne : découpage des lieux primitifs, bouton permettant d'accéder à l'affichage des lieux, carte des instances d'objets, position estimées de l'objet sélectionné, d'après les lieux de la première et de la seconde liste. En bas, liste des objets stockés en mémoire. Le nom de l'objet est constitué de la position où il à été détecté initialement, et des interactions qui ont été énactées depuis. Si un objet est sélectionné, les deux listes de lieu caractérisant sa position est affichée. Il est possible de sélectionner un lieu dans chaques listes pour afficher leurs signatures (en bas à droite). En agrandissant la fenêtre, (image du bas), on permet l'affichage des lieux évoqués et succeptibles d'être ajouté à la seconde liste au cycle de décision suivant.

CamPanel	**X**
MapPanel	**X**