Programmation Orientée Émotion (*EOP*)

K. Darty [a] N. Sabouret [a]
dartykevin@gmail.com Nicolas.Sabouret@lip6.fr

[a]Université Pierre et Marie CURIE
4 place Jussieu, 75005 Paris, France

Résumé :
Certains problèmes algorithmiques actuels se heurtent à une multitude d'états possibles, souvent dus à un environnement dynamique. L'être humain a comme solution l'utilisation d'émotions qui augmente sa réactivité et l'aide dans ses décisions. Nous proposons une méthode de résolution de problèmes algorithmiques par une modélisation du système émotionnel humain visant à améliorer la réactivité et les performances de la résolution.

Mots-clés : Programmation Orientée Émotion, émotion, résolution de problème

Abstract:
Some current algorithmic problems are facing a multitude of states, often due to a dynamic environment. The human being's approach to such problems consists of using emotions which increase his reactivity and help him in his decisions. We put forward an algorithmic problem solving method using a modelling of the human emotional system aiming at improving the resolution's reactivity and performance.

Keywords: Emotion Oriented Programming, emotion, problem solving

1 Introduction

Du point de vue des psychologues-cogniticiens, les êtres humains sont constamment en train de résoudre des problèmes à partir des informations fournies par leur environnement (via leurs perceptions) [11]. Les recherches ont montré que ces informations à analyser sont bien trop volumineuses pour que les problèmes soient résolus uniquement de façon rationnelle. Les émotions ressenties par l'être humain et d'autres animaux sont apparues au cours de l'évolution par la sélection naturelle et ont pour but d'améliorer la réactivité et ainsi les décisions [1]. Par la suite, Scherer [14] définit formellement l'émotion comme réponse d'un stimulus intervenant à des niveaux organiques notamment cognitif et physique. Sa théorie permet après évaluation d'un événement d'en déduire la nature et la force de l'émotion.

Les émotions permettent d'une part des réponses comportementales plus rapides et mieux adaptées et d'autre part une meilleure distribution de l'attention sur les informations les plus récentes ou les plus importantes dans l'environnement. Ainsi, des décisions rapides, incertaines sont prises et c'est ce mécanisme qui permet des changements de comportement adaptés (dans une majorité de cas).

Lorsqu'on s'intéresse à la résolution de problème en informatique dans des environnements réels, comme c'est le cas en robotique ou dans le domaine des *Agents Conversationnels Animés* en interaction avec un être humain et dans des environnements virtuels de plus en plus riches, on se retrouve confronté aux mêmes difficultés. En effet, parce qu'il peut être très dynamique et contenir beaucoup d'informations, l'environnement conduit à une explosion combinatoire de l'espace d'états, ce qui rend difficile l'utilisation des algorithmes classiques issus de la recherche en Intelligence Artificielle (*IA*).

Dans ce contexte, l'utilisation de mécanismes similaires aux émotions humaines semble constituer une heuristique intéressante pour la résolution de problèmes complexes. Dans cet article, nous présentons un modèle informatique, inspiré des émotions, pour définir des programmes informatiques capables de résoudre des problèmes d'*IA* difficiles dans des environnements dynamiques et incertains.

Dans la section 2, nous présentons les travaux dans le domaine de l'informatique affective et des agents conversationnels animés qui ont servi de support à notre étude. Nous définissons ensuite dans la section 3 notre modèle de *Programmation Orientée Émotion* (*EOP*). Nous illustrons son fonctionnement sur un exemple dans la section 4 et nous discutons des perspectives de recherche dans la section 5.

2 Étude de la problématique

Pour pouvoir aborder la question de la construction d'un modèle d'*EOP*, il faut étudier d'un coté les modèles d'émotions développés en psychologie et de l'autre les techniques classiques de résolution de problème en *IA*.

Notre objectif est de définir un outil de résolution de problèmes qui mette en œuvre de manière automatique ou semi-automatique des heuristiques inspirées du mécanisme "émotionnel" des être vivants. C'est pourquoi nous devons étudier à la fois les modèles issus de l'informatique affective et les modèles de résolution de problème en *IA*.

2.1 Informatique affective

Pour Lazarus[7], lorsqu'un événement survient, le sujet fait des évaluations cognitives qui entraînent la création d'une émotion. Ces émotions sont à l'origine d'efforts servant à garder le contrôle sur la situation. Finalement, le "*coping*" fait converger l'individu vers une solution qui se traduit par des comportements ou bien des décisions réponses au problème non trivial initial.

En informatique, les modèles d'évaluation cognitive (*appraisal*) des émotions, inspirés des travaux de Roseman [12] et Scherer [14], sont fréquemment utilisés. On distingue deux approches pour représenter et manipuler l'état émotionnel. L'approche dimensionnelle [9] considère l'émotion comme un vecteur dans un espace à n dimensions, l'ensemble des émotions constituant le tempérament. Par exemple, le *framework* de Mehrabian considère que les axes *Pleasure*, *Arousal* et *Dominance* (*PAD*) forment une base suffisante pour décrire une émotion. L'approche catégorielle considère au contraire qu'il est possible d'énumérer l'ensemble des émotions. Le modèle d'*appraisal* d'Ortony, Clore et Collins (*OCC*) [10] se situe dans cette approche et définit chaque catégorie d'émotion comme une conséquence de multiples cognitions et interprétations de l'environnement. Plus récemment, la thèse de Gebhard [4] regroupe ces deux approches dans un modèle computationnel d'évaluation cognitive : la partie évaluation se fait grâce à *OCC* pour classifier en type d'émotion et la partie évolution des humeurs qui se fait via des états émotionnels *PAD* simulés continuellement.

Nous proposons ici de nous inspirer de ce modèle pour définir notre solveur de problème : les événements de l'environnement induisent des évaluations cognitives qui débouchent sur des émotions. Ainsi, les émotions "synthétisent" l'état de l'environnement et permettent de réduire l'ensemble des règles comportementales à décrire pour la résolution du problème.

De plus, les émotions interviennent directement dans la perception de l'environnement en modifiant le mécanisme d'attention [16]. Dans notre modèle, nous utiliserons les émotions et l'humeur pour sélectionner les perceptions dans l'environnement et réduire ainsi l'espace de recherche. Ainsi, nous proposons de transformer le principe classique de perception-action des agents [13] en influant sur l'équilibre entre ces deux dernières.

2.2 Résolution de problème en *IA*

Du point de vue de l'*IA*, un problème peut être considéré comme un ensemble d'objectifs et de données [8]. La résolution d'un problème consiste alors à passer d'un état initial vers un état solution tout en minimisant le temps de calcul et en respectant les contraintes de l'environnement (qui définissent les transitions possibles entre les états). Dès 1969, certains théoriciens pensent approcher une solveur de problème général par décomposition en sous-problèmes [2] mais la définition du problème reste une étape complexe et cette méthode se voit limitée aux problèmes bien définis.

Par la suite, les systèmes de production puis à base de règles [17] font leur apparition avec pour formalisme $Condition \Rightarrow Action$ et comme base le *modus ponens*. Ils sont capables de modéliser de nombreux phénomènes expérimentaux comme *SOAR* avec son modèle computationnel de l'espace-problème [6]. Leur succès provient de la modélisation simple des connaissances disponibles et des procédures existantes. Cependant, ces solveurs déterministes s'adaptent mal à un environnement dynamique : c'est toujours le même cheminement vers la solution qui sera suivi.

Les approches à base de planification, comme le modèle *STRIPS* [3], s'appuient sur une analyse des buts et des actions de transition pour construire un parcours de l'espace d'état avec des heuristiques. Cette opération de planification est confrontée à plusieurs difficultés : pour qu'elle aboutisse, il est nécessaire de connaître parfaitement l'ensemble des règles de transition qui régissent l'environnement, ce qui est impraticable dès que la perception est limitée ou que l'environnement est trop riche ou trop dynamique.

Plus récemment, l'apprentissage par renforcement [15] a montré qu'il était possible pour les

agents d'explorer leur environnement pour dégager, de manière non-supervisée, un chemin vers la solution. Mais une fois encore, ces algorithmes ne peuvent pas faire face à un trop grand nombre d'actions ou de perceptions, ni à un environnement dynamique.

C'est pourquoi nous proposons, tout en nous appuyant sur les mécanismes de perception-action qui sont ceux de l'*IA* et des approches de résolution de problèmes à base de règle, d'utiliser les émotions comme une heuristique pour le parcours de l'espace d'états. Nous ne prétendons pas ici que les émotions sont nécessairement une bonne approche mais nous pensons qu'il existe une classe de problèmes, caractérisée par le grand nombre de perceptions et d'actions possibles, ainsi que par des changements trop fréquents dans l'environnement, pour lesquels les émotions peuvent permettre aux algorithmes classiques de converger.

3 Présentation du modèle *EOP*

Nous présentons dans cet article un modèle de programmation orientée émotion (*EOP*) qui a pour objectif de décrire et de résoudre des problèmes en utilisant un mécanisme inspiré des émotions. Notre modèle est donc composé de deux parties : la description du problème (langage d'*EOP*) et le moteur de résolution.

Le modèle *EOP* n'est donc pas un solveur général de problème, comme *STRIPS* [3]. Il demande un travail au programmeur : celui de choisir comment son agent interprète émotionnellement les variables de l'environnement pour générer ses émotions et de déterminer la réaction (*coping* [7]) à ces émotions pour sélectionner les actions.

3.1 Architecture générale

Le fonctionnement général du modèle est résumé dans la figure 1. L'environnement (1) envoie des perceptions au moteur *EOP* (2). Les perceptions sont traduites en émotions par les *Threads de perception* (2b). Ces émotions sont utilisées en entrée par les *Threads de décision* (2b) pour sélectionner les actions qui modifient l'environnement. L'équilibre entre perception et décision est géré par le contrôleur (2a) en fonction de l'humeur (qui elle même dépend des émotions générées).

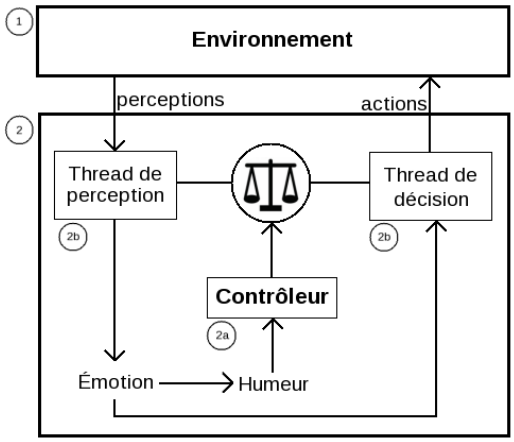

FIG. 1 – Blocs fonctionnels du modèle *EOP*

Modèle *EOP*. Le moteur *EOP* met en jeu des émotions et des humeurs. Elles sont représentées dans l'espace dimensionel *PAD* de Mehrabian [9] dont la sémantique dans notre modèle *EOP* est la suivante :

- *Pleasure* représente le sentiment de proximité du but. Un fort plaisir équivaut à un état "proche" de l'état but (la mesure de proximité devant être définie par le programmeur).
- *Arousal* exprime le sentiment de nouveauté dans les perceptions. Un faible *arousal* correspond à un parcours dans lequel les états restent similaires.
- *Dominance* est une indication du sentiment de contrôle de l'agent sur la situation. Une forte dominance équivaut à une plus grande capacité d'action sur l'environnement.

Comme dans les travaux de Gebhard [4], les émotions peuvent être catégorisées (chaque catégorie d'émotion pouvant être calculée par une *PerceptionThread* différente). Le nombre et la nature de ces catégories n'est pas prédéfini (c'est au programmeur de les choisir).

L'humeur de l'agent-solveur est aussi définie en utilisant le modèle *PAD* et évolue en fonction des émotions déclenchées au cours de l'éxécution (*cf.* section 3.4). De plus, le solveur sera paramétré par une humeur initiale $\overrightarrow{h_{init}}$ et des paramètres de sensibilité $\overrightarrow{h_{sens}}$ pour chacune des trois composantes.

Définition du problème. Le problème à résoudre est modélisé dans l'environnement : celui-ci contient l'ensemble des variables du problème et des actions par lesquelles les threads de décision peuvent agir dessus. De plus, l'environnement pouvant être dynamique, il dispose d'une

fonction de "mise à jour" qui définit son changement d'état en dehors de toute action externe.

Pour résoudre ce problème, le programmeur définit un ensemble de *threads* : les threads de perception (*PerceptionThreads*) et les threads de décision (*DecisionThreads*). Les premières servent de créateur d'émotions à partir des variables l'environnement. Les secondes utilisent les émotions pour déclencher une action de l'environnement.

Toutes les threads sont caractérisées par :
– Des conditions d'exécution qui définissent dans quels états la thread (perception ou action) peut être appliquée ;
– Une priorité calculée en fonction de l'humeur.

Fonctionnement par cycle. Le moteur de résolution fonctionne de manière cyclique. À chaque cycle, il choisit un thread parmi les thread disponibles (c'est-à-dire dont les conditions d'exécution sont réunies) en fonction de leur priorité, puis il l'exécute. Cela peut conduire à l'apparition d'émotions (*perceptionThread*) ou à des modifications de l'environnement (*decisionThread*). Le moteur exécute ensuite la fonction de mise à jour de l'environnement puis réévalue la disponibilité de toutes les threads en fonction du nouvel état. Il met à jour l'humeur en fonction de l'humeur actuelle et des émotions éventuellement produites et calcule la nouvelle valeur de priorité pour chaque thread.

Dans les sections suivantes, nous présentons formellement le modèle *EOP*. Nous présentons tout d'abord la description de l'environnement (section 3.2). Nous définissons ensuite (section 3.3) le modèle de programmation des threads de perceptions et de décisions. Nous décrivons enfin dans la section 3.4 le moteur de résolution qui contrôle l'exécution de notre solveur d'*EOP* : l'interprète *EOP*.

3.2 Description de l'environnement

L'environnement rassemble les données du problème ainsi que le problème en lui-même. Il contient les actions et les variables ainsi que des informations sur l'état du problème telles que sa solvabilité. L'environnement E est définit par un ensemble de variables V, un ensemble d'opérations O sur ces variables, un processus P_{env} et un ensemble F de fonctions d'évaluation du problème :

$$E = \langle V, O, P_{env}, F \rangle$$

Nous notons $V_e \subset V$ l'ensemble des variables externes, utilisables par le programeur dans la description des threads. Pour chaque $v \in V$, nous notons D_v le domaine de valeurs de la variable v.

Chaque opération $o \in O$ est caractérisée par ses préconditions et ses effets :

$$o = \langle Prec_o, Eff_o \rangle$$

où $Prec_o$ est une fonction booléenne sur l'ensemble des variables : $Prec_o : V \rightarrow \mathbb{B}$. Eff_o est un ensemble ordonné de couples (var, val), $var \in V$ et $val : V \rightarrow D_{var}$ est une fonction de l'ensemble des variables qui construit la nouvelle valeur de var.

Par exemple, si on souhaite faire un système *EOP* de conduite automobile assistée, on peut avoir une opération de bas niveau comme "changer de file", "freiner", *etc*. Les préconditions portent alors sur la présence d'autres véhicules, leurs vitesses, de panneaux de signalisation...

Les fonctions d'évaluation $f \in F$ du problème travaillent sur l'ensemble V et servent principalement au programmeur dans la définition des threads émotionnels, pour définir ses heuristiques.

Par exemple, dans le problème du labyrinthe, on peut souhaiter utiliser une fonction de distance qui donne la distance à vol d'oiseau au but (sans tenir compte des murs) :

$$d : (x_{agent}, y_{agent}, x_{but}, y_{but}) \rightarrow \mathbb{R}$$

Mais il existe quatre fonctions qui sont utilisées directement par le moteur de résolution pour évaluer l'état de la résolution du problème :
– la fonction $solved : V \rightarrow \mathbb{B}$ qui vaut $true$ si et seulement si l'environnement est dans l'état but du problème.
– la fonction $solvable : V \rightarrow \mathbb{B}$ qui vaut $false$ si et seulement si l'environnement est dans un état qui rend impossible la résolution du problème (par exemple, si l'explorateur a été mangé par un monstre).
– la fonction $distance : V_e^2 \rightarrow \mathbb{R}^+$ qui définit la dissimilarité entre deux états :

$$distance(v, v') = 0 \Leftrightarrow v = v'$$

$distance(v, v')$ croit en fonction des différences de valeurs des variables.
– la fonction $goal : V_e \rightarrow \mathbb{R}^+$ qui définit la distance au but. Cette distance peut intégrer des

optimums locaux (par exemple, l'explorateur est à côté d'un trésor, d'un point de ravitaillement alors qu'il a soif, *etc*) et vaut 0 pour tout état but :

$$goal(v) = 0 \Leftrightarrow solved(v)$$

Ces fonctions doivent être définies lors de la description du problème dans l'environnement.

Processus. Un processus p est un ensemble d'opérations $O' \subset O$, munies de préconditions et d'effets, travaillant sur un sous-ensemble $V_p \in V$ et fonctionnant de manière cyclique.

Les préconditions des opérations doivent être mutuellement exclusives (au plus une seule action est éligible à chaque instant) [1] :

$$\forall (o_i, o_j) \in p^2, Prec_{o_i} \wedge Prec_{o_j} = \bot$$

À chaque cycle, le processus exécute l'action éligible (s'il y en a une).[2]

L'environnement a un processus p_{env} mais on aura aussi des processus dans les threads de perception et de décision.

Exemple. Le problème classique du Wumpus [13] permet d'illustrer de façon concrète le fonctionnement de l'environnement : lorsqu'un monstre aperçoit le joueur dans un couloir, le processus permet de changer la position du monstre vers le joueur pour qu'il lui court après.

3.3 Programmation Orientée Émotion

Soit $\mathcal{T}$ l'ensemble des threads, décomposé en perceptions $\mathcal{P}$ et décisions $\mathcal{D}$. Pour chaque thread $t \in \mathcal{T}$, on note $p_t \in \mathbb{R}$ sa priorité et $c_t \in \mathbb{B}$ son éligibilité (déterminée par ses conditions d'exécution).

Les priorités des threads éligibles sont normalisés pour en faire des probabilités :

$$\sum_{\forall t \in \mathcal{T}.c_t = \top} norm(p_t) = 1$$

Puis un thread est sélectionné aléatoirement en respectant leur probabilité respective :

$$\forall t \in \mathcal{T}.c_t = \top, p(select = t) = norm(p_t)$$

[1] Cela permet de ne pas avoir à choisir d'ordre dans l'exécution des actions, qui est un problème courant en systèmes multi-agents ou dans les algèbres de Gurevich [5].

[2] Nous verrons dans la suite de cet article comment les cycles sont contrôlés au niveau du moteur de résolution (section 3.4).

Comme dans le modèle de Gebhard [4], chaque émotion $e \in \mathcal{E}$ est caractérisée par un vecteur dans l'espace *PAD* :

$$\overrightarrow{PAD}^{(e)} = (P^{(e)}, A^{(e)}, D^{(e)}) \in [-1, 1]^3$$

Soit v l'état courant et v' l'état précédent :

- $P^{(e)} = f(goal(v) - goal(v'))$ est la valeur sur l'axe *pleasure*.
  La fonction f est une exponentielle inverse définie de la manière suivante :

$$f(x) = \begin{cases} 1 - e^{-x} & \text{si } x \geq 0 \\ e^{x} - 1 & \text{sinon} \end{cases}$$

  Ainsi, un plaisir proche de 1 correspond à un réel progrès en direction du but. Le plaisir diminue de manière exponentielle dès que cet écart décroit. Symétriquement, le déplaisir est d'autant plus grand que le but s'éloigne rapidement.
- $A^{(e)} = h(g(distance(v, v')))$ est la valeur sur l'axe *arousal*
  La fonction $g : \mathbb{R}^+ \rightarrow [0, 1[$ est définie de la manière suivante : $g(x) = 1 - e^{-x}$. Cela permet d'accentuer aux bords l'impact de la distance sur l'*arousal*.
  La fonction $h : [0, 1] \rightarrow [-1, 1]$ est, dans notre implémentation actuelle, définie par $h(x) = 2x - 1$ (elle transpose l'ensemble linéairement). Elle devrait en fait dépendre de l'écart-type entre deux états v et v' voisins dans l'espace d'état.[3]
- $D^{(e)} = h(\frac{card(t \in \mathcal{D}.c_t = \top)}{card(\mathcal{D})})$ est la valeur sur l'axe *dominance*. Elle utilise la même fonction h que dans la formule de $A^{(e)}$ pour transposer linéairement l'ensemble de valeurs. Elle devrait en fait dépendre du nombre d'actions valides (*i.e.* $c_t = \top$) en moyenne dans chaque état.

3.4 Émotions et Humeur

Les threads de perception permettent de générer des émotions. $\forall t \in \mathcal{P}$, nous notons $E(t) \subset \mathcal{E}$ l'ensemble des émotions produites par t.

Émotions. Chaque émotion $e \in \mathcal{E}$ est caractérisée non seulement par sa valeur dans le vecteur $\overrightarrow{PAD}^{(e)}$, comme énoncé précédemment, mais aussi par une date d'apparition $c_d^{(e)}$, une date de

[3] Nous pensons qu'en réalité, l'*arousal* devrait aussi dépendre du nombre de thread de perception actives par rapport au nombre de thread de perception possibles.

fin de vie $c_f^{(e)}$, définies par le programmeur, ainsi qu'un ensemble de variables sources $vs^{(e)} \subset V_e$ qui interviennent dans la construction de l'émotion (par exemple, une émotion de peur peut être provoquée par l'apparition d'un danger, la perte d'une resource, etc...) :

$$e = \langle \overrightarrow{PAD}^{(e)}, c_d^{(e)}, c_f^{(e)}, vs^{(e)} \rangle$$

De plus, $\forall e \in \mathcal{E}$, nous notons $t_p(e) \in \mathcal{T}$ le thread créateur de l'émotion e.

Deux émotions e et e' sont considérées comme similaires quand leur valeurs dans l'espace *PAD*, leur ensemble de variables sources et le thread créateur sont identiques :

$$e \sim e' \Leftrightarrow \begin{cases} \overrightarrow{PAD}^{(e)} = \overrightarrow{PAD}^{(e')} \\ vs^{(e)} = vs^{(e')} \\ t_p(e) = t_p(e') \end{cases}$$

Au cycle τ, s'il existe une émotion e "vivante" ($\tau \in [c_d^{(e)}, c_f^{(e)}]$) et si un thread crée une émotion $e' \sim e$, alors e' est fusionnée à e pour créer une nouvelle émotion :

$$\langle \overrightarrow{PAD}^{(e)}, \min(c_d^{(e)}, c_d^{(e')}), \max(c_f^{(e)}, c_f^{(e')})), vs^{(e)} \rangle$$

Lorsque le cycle de fin de vie c_f est atteint, l'émotion meure : elle disparaît des données du moteur de résolution.[4]

Humeur. Le moteur de résolution maintient à jour une "humeur" de l'agent qui permet au programmeur de gérer les priorités des threads de perception et de décision.[5] L'humeur $\overrightarrow{h}$ est caractérisée par un vecteur dans l'espace *PAD* initialisé à la valeur $\overrightarrow{h_{init}}$.

À chaque cycle τ, l'humeur est mise à jour en deux étapes. Premièrement, elle se rapproche des émotions vivantes :

$$\overrightarrow{h_{new}} = \overrightarrow{h} + \frac{1}{evol_h} \times \left(\overrightarrow{h_{sens}}.\Sigma_E - \overrightarrow{h} \right)$$

où $evol_h \in \mathbb{R}^+$ représente le pas de progression de l'humeur en direction des émotions vivantes

[4]En fait, l'émotion est seulement supprimée de la liste des émotions vivantes. Cependant les émotions les plus récentes sont conservées dans une file appelée "mémoire émotionnelle" afin que le programmeur puisse les utiliser dans la définition des threads de décision. Ainsi, la décision de l'agent peut dépendre d'émotions ressenties par le passé.

[5]À terme, notre objectif est que le programmeur ne définisse pas directement la priorité p_t des threads mais que celle-ci soit calculée automatiquement par le moteur à partir de fonctions caractéristiques.

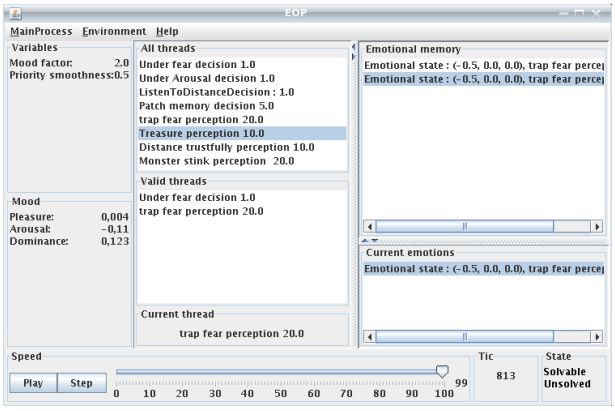

FIG. 2 – Moteur de résolution

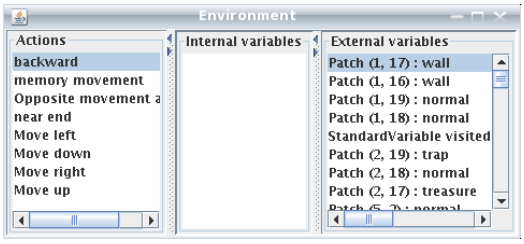

FIG. 3 – Environnement

et Σ_E est la somme des émotions vivantes :

$$\Sigma_E = \sum_{\forall e \in \mathcal{E}, \tau \in [c_d^{(e)}, c_f^{(e)}]} \overrightarrow{PAD}^{(e)}$$

Ensuite, elle se rapproche de $\overrightarrow{h_{init}}$ suivant un facteur de convergence $conv_h$:

$$\frac{(\overrightarrow{h_{new}} - \overrightarrow{h_{init}})}{conv_h} + \overrightarrow{h_{init}}$$

4 Implémentation et évaluation

La structure du modèle *EOP* a été implémentée en *Java 1.5*. Une interface simple, illustrée sur la figure 2, permet de visualiser l'humeur (à gauche), les threads disponibles, éligibles et leurs priorités (au centre) ainsi que les émotions déclenchées (à droite). Une barre de vitesse permet de contrôler l'exécution de moteur *EOP*. Enfin, il est possible de visualiser l'état de l'environnement dans une autre fenêtre (figure 3).

Exemple. Nous avons décidé de tester notre modèle *EOP* sur un problème de type *Wumpus* (labyrinthe avec des obstacles, des pièges et un ou plusieurs buts). La figure 4 montre l'*IHM* de visualisation de cet environnement.

FIG. 4 – Labyrinthe

Notre agent doit aller de la case de départ jusqu'à la case d'arrivée (drapeau) tout en récoltant le plus possible de trésors (en rouge sur l'*IHM*), en évitant des pièges statiques (en noir) et des monstres (Wumpus, représentés par un carré jaune sur l'*IHM*) qui le poursuivent dès qu'il s'approche.

Notre environnement contient donc trois ensembles de variables internes T (liste des positions des trésors), P (liste des positions des pièges) et W (liste des positions des Wumpus) ainsi que les variables externes M (liste des positions des murs), (x, y) (la position courante de l'agent), (x_{but}, y_{but}) (la position but), t le nombre de trésors ramassés, ainsi que des variables pour la perception : *piege visible* (pv), *monstre visible* (mv) et *trésor visible* (tv).

Le processus de l'environnement met à jour les variables de perception : lorsque l'agent se situe à moins d'une case d'un trésor ou d'un piège. Pour les monstres, la distance est de 3 cases. De plus, il déplace les monstres d'une case en direction de l'agent si celui-ci est situé sur la même ligne ou sur la même colonne, quelque soit la distance (sans obstacle).

Les opérations possibles sur l'environnement sont les déplacements de l'agent d'une case vers une case voisine accessible et l'action de ramasser un trésor. Chaque déplacement a pour précondition $solvable$ et l'absence de mur dans la direction du déplacement. Ramasser un trésor a pour précondition la présence d'un trésor (c'est-à-dire que $(x, y) \in T$).

Les fonctions $solved$, $solvable$, $distance$ et $goal$ sont définies de la manière suivante :
- $solved$: $x = x_{but} \wedge y = y_{but}$ (l'agent est arrivé sur la case but) ;
- $solvable$: $(x, y) \notin P \cup W$ (l'agent n'est pas sur un piège ou sur un monstre) ;
- $distance$: $\delta.\sqrt{(x-x')^2 + (y-y')^2} + \alpha.|pv - pv'| + \beta.|mv - mv'| + \gamma.|tv - tv'|$ où α, β, γ et δ représentent le facteur d'importance pour chaque critère (piège, monstre, trésor, distance à l'arrivée). Dans notre implémentation, nous avons pris $\alpha = \beta = \gamma = 2$ et $\delta = 1$.
- $goal(v) = distance(v, v_{but})$ avec $\alpha = \beta = 2$ $\gamma = 1$ $\delta = max^{-1}_{distance}$ où $max_{distance}$ est la longeur de la diagonale du labyrinthe.

Solution *EOP*. Pour tester la résolution de ce problème en utilisant une approche *EOP*, nous avons choisi des threads de perception et de décision simples : générer de la peur, fuir, s'approcher du but, etc.

Par exemple, nous avons une thread de perception t_{peur} qui génère une émotion lorsqu'on perçoit un piège ou un monstre. Cette perception produit une émotion de *pleasure* négatif, d'*arousal* positif et de *dominance* neutre. Sa priorité dépend de la composante *dominance* de l'humeur : moins l'agent se sent dominant, plus il va être attentif aux dangers potentiels. Les données mémorisées dans l'émotion sont les coordonnées de la menace, ce qui permet de les utiliser dans les threads de décision pour fuir.

Évaluation en cours. L'exemple décrit a été implanté dans une version intermédiaire de notre moteur *EOP*, qui n'intègre pas toutes les composantes du modèle décrit ici. Nous sommes actuellement en train de ré-implanter ce modèle pour pouvoir faire des comparaisons précises.

Nous avons implanté un algorithme de parcours de chemin en largeur (algorithme de Floyd) pour comparer cet algorithme classique avec notre modèle *EOP*. Nous avons mené une première expérience avec 100 labyrinthes de 10×10 cases générés aléatoirement avec deux Wumpus présents et les probabilités de piège, trésor et mur sur chaque case fixées à 5%, 5% et 40% respectivement. En utilisant le modèle *EOP*, l'agent atteint la case d'arrivée dans 27% des cas. Le nombre de trésors trouvés par parcours (même en cas d'échec) est en moyenne de 23% du nombre total de trésors.

De manière assez peu surprenante, notre modèle converge dans un nombre raisonnable de situations là où les algorithmes classiques sont voués à l'échec (Floyd ne peut pas gérer la dynamicité des Wumpus dans l'environnement). Évidemment, il faudrait comparer à d'autres méthodes de recherches heuristiques (telles que A^*) et discuter l'expressivité de notre modèle pour pouvoir évaluer correctement l'approche

EOP. Nous n'en sommes qu'au début de nos recherches.

5 Conclusion et perspectives

Le modèle de Programmation Orientée Émotion présenté est encore en stade de développement. Les premières expérimentations montrent que cette méthode est une technique de résolution de problèmes qui donne des résultats intéressants : son fonctionnement lui permet de s'adapter en environnement dynamique et complexe. Une phase d'évaluation plus poussée (en cours) doit nous permettre de définir précisément la classe de problèmes qu'il peut traiter.

Pour généraliser notre modèle et ainsi le compléter, nous voudrions réduire la partie à réaliser par le programmeur au strict minimum. Nous pourrions, par exemple, changer la méthode de définition des priorités - pour l'instant laissée à la charge du programmeur - en demandant uniquement à quelles caractéristiques de l'émotion le thread doit réagir.

À terme, nous pensons que l'*EOP* peut définir une nouvelle approche pour la résolution de problèmes en environnement dynamique et en temps contraint, tirant parti à la fois des méthodes heuristiques en résolution de problèmes en *IA* et de la souplesse et de la réactivité qu'offrent les émotions chez les être vivants.

6 Remerciements

Merci à Cindy Mason du *MIT* pour nous avoir autorisé à reprendre l'idée originale de la programmation orientée émotions qu'elle avait initié. Merci à Guillaume Carré qui a travaillé sur la première version de notre moteur *EOP*.

Références

[1] C. Darwin, P. Ekman, and P. Prodger. *The expression of the emotions in man and animals*. Oxford University Press, USA, 2002.

[2] G. Ernst and A. Newell. *GPS : A case study in generality and problem solving*. Academic Pr, 1969.

[3] R. Fikes and N. Nilsson. STRIPS : A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4) :189–208, 1971.

[4] P. Gebhard. Alma : A layered model of affect. *International Conference On Autonomous Agent*, pages 29–39, 1996.

[5] Y. Gurevich. Sequential abstract-state machines capture sequential algorithms. *ACM Transactions on Computational Logic (TOCL)*, 1(1) :111, 2000.

[6] J. Laird, A. Newell, P. Rosenbloom, C.-M. U. P. P. A. INTELLIGENCE, and P. PROJECT. Soar : An architecture for general intelligence. 1987.

[7] R. Lazarus and S. Folkman. *Stress, appraisal, and coping*. Springer, 1996.

[8] R. Mayer. Thinking, problem solving, cognition. 1983.

[9] A. Mehrabian. Pleasure-arousal-dominance : A general framework for describing and measuring individual differences in temperament. *Current Psychology*, 14(4) :261–292, 1996.

[10] A. Ortony, G. Clore, and A. Collins. *The cognitive structure of emotions*. Cambridge Univ Pr, 1990.

[11] J.-F. Richard, S. Poitrenaud, and C. Tijus. Problem-Solving Restructuration : Elimination of Implicit Constraints. *Cognitive Science*, 17 :497–529, 1993.

[12] I. Roseman. Appraisal determinants of emotions : Constructing a more accurate and comprehensive theory. *Cognition & Emotion*, 10(3) :241–278, 1996.

[13] S. Russell and P. Norvig. *Artificial intelligence : a modern approach*. Prentice hall, 2009.

[14] K. Scherer, H. Wallbott, and A. Summerfield. *Experiencing emotion : A cross-cultural study*. Cambridge University Press ; Paris : Maison des Sciences de l'Homme, 1986.

[15] R. Sutton and A. Barto. *Reinforcement learning : An introduction*. The MIT press, 1998.

[16] J. Taylor and N. Fragopanagos. Modelling the interaction of attention and emotion. In *2005 IEEE International Joint Conference on Neural Networks, 2005. IJCNN'05. Proceedings*, pages 1663–1668.

[17] D. Waterman. Generalization learning techniques for automating the learning of heuristics. *Artificial Intelligence*, 1(1-2) :121–170, 1970.