Approche de présélection multicritère à base de traces pour la prise de décision dans les applications interactives de type jeux

Hoang Nam Ho [1], **Mourad Rabah** [1], **Samuel Nowakowski** [2], **Pascal Estraillier** [1]

1. *Laboratoire L3i, Université de La Rochelle*
   Avenue Michel Crépeau - 17042 La Rochelle Cedex 1 - France
   {hoang_nam.ho,mourad.rabah,pascal.estraillier}@univ-lr.fr

2. *LORIA, UMR 7503, Université de Lorraine*
   Campus Scientifique, BP 239, Vandoeuvre-lès-Nancy, France
   samuel.nowakowski@loria.fr

RÉSUMÉ. *La prise de décision dans les jeux est une fonctionnalité indispensable pour automatiser les jeux et les rendre plus autonomes et plus intelligents. Un algorithme de décision effectue les calculs d'optimisation sur l'ensemble des solutions possibles. Cela fait augmenter le temps de calcul et pose un problème d'explosion combinatoire lorsque l'espace de solutions est grand. Afin de surmonter ce problème, nous présentons une approche de présélection des solutions pertinentes avant d'effectuer une décision. Notre approche comporte deux étapes : i) utilisation des traces (exécutions antérieures des utilisateurs) pour identifier toutes les solutions potentielles ; ii) estimation de la pertinence, appelée utilité, de chacune de ces solutions potentielles. Nous obtenons un ensemble de solutions candidates présélectionnées qui sera utilisé comme entrée à la prise de décision. Nous expérimentons notre approche sur un prototype du jeu Tamagotchi.*

ABSTRACT. *The decision making in games is essential to make them more automated and smart. A decision algorithm performs its calculations on the set of all the possible solutions. This increases the computation time and may become a combinatorial explosion problem if we have a huge solution space. To overcome this problem, we present our work on relevant solutions preselection before making a decision. We propose a two-steps strategy: i) the first step analyzes system's traces (users past executions) to identify all the potential solutions; ii) the second step aims to estimate the relevance, called utility, of each of these potential solutions. We get a set of alternative solutions that can be used as an input to any decision algorithm. We illustrate our approach on the Tamagotchi game.*

MOTS-CLÉS : *système interactif adaptatif, traces, prédiction, utilité, décision multicritère*

KEYWORDS: *interactive adaptive system, traces, prediction, utility, multi-criteria decision making*

DOI:10.3166/RIA.31.311-335

1. Introduction

Les jeux informatisés appartiennent naturellement à la famille des applications interactives. Ces dernières sont vues comme une suite d'activités successives et d'événements réalisés entre l'utilisateur et le système (Brun, Beaudouin-Lafon, 1995). Ces activités sont les principaux composants qu'il faut mettre en œuvre en vue de disposer d'une organisation ainsi que d'une structuration des interactions, ou scénario, dans une application interactive. Nos travaux s'appuient sur l'hypothèse qu'une application interactive est contextualisée au moyen des *situations*. Une *situation* est ici vue comme un composant dans lequel les acteurs, humains ou automatisés, interagissent en utilisant des ressources locales associées à un contexte commun en vue d'atteindre un ou plusieurs objectifs identifiés (Pham *et al.*, 2015). Ainsi, l'utilisateur d'une application interactive exécute et participe à des *situations* successives jusqu'à atteindre un objectif prédéfini par le concepteur et l'exécution de l'application repose sur l'enchaînement des différentes *situations*. Nous souhaitons optimiser le processus d'enchaînement des situations composant une application interactive de type jeu par l'utilisation de l'intelligence artificielle (IA). L'IA dans les jeux comprend plusieurs domaines à traiter tels que : le dialogue automatique entre l'ordinateur et le joueur, l'aide aux diagnostics, la résolution de problèmes complexes, l'aide à la décision, etc. Dans le présent article, nous nous intéressons à l'aide à la décision dans des applications, et plus particulièrement les jeux, structurées en *situations*.

Un jeu est un terrain d'expérimentation idéal pour les aspects d'IA et en particulier du domaine d'aide à la décision. Les règles sont généralement bien définies et déterministes, ce qui permet de découper l'exécution d'un jeu en un enchaînement de situations. Avec une telle structuration, à chaque sortie de situation, nous disposons d'un vecteur d'état global représentant les attributs du système à la fin de l'exécution de la situation actuelle. En fonction de ce vecteur d'état, la logique d'enchaînement de situations va déterminer la meilleure situation à suivre parmi celles disponibles.

Le problème réside dans le choix de la situation suivante à l'issue de la situation courante tout en respectant les objectifs fixés par le concepteur du scénario et en s'adaptant au comportement de l'utilisateur. Il s'agit de concevoir des méthodes et des outils permettant d'effectuer et/ou d'accompagner le choix des situations parmi un ensemble de situations disponibles. Pour y parvenir, nous nous intéressons aux systèmes d'aide à la décision. Ces systèmes vont nous permettre d'optimiser les actions : identifier la situation qui permet de poursuivre l'exécution de l'application et respecter un ou plusieurs objectifs de l'application. La situation ainsi proposée par le système de décision est obtenue en respectant un critère déterminé. Cependant, pour une application donnée, le choix ne se base pas sur un critère unique et fait, en général, intervenir une multitude de critères. Dans ce cas, l'aide à la décision doit prendre en compte tous ces critères pour choisir une solution optimale parmi l'ensemble des solutions possibles. Ceci nous a amené à envisager des systèmes d'aide à la décision multicritère (Köksalan *et al.*, 2011). Un tel système comporte généralement cinq étapes dont les 4 dernières se répètent tout au long de l'exécution :

i) étape initiale : définition de l'objectif de l'application que l'utilisateur doit atteindre ;

ii) étape 1 : identification des critères qui nous guident dans l'accomplissement et l'évaluation de l'objectif ;

iii) étape 2 : pondération des critères pour quantifier l'importance relative entre les critères ;

iv) étape 3 : analyse des solutions possibles pour identifier les candidates et préparer la prise de décision ;

v) étape 4 : traitement des solutions candidates par les algorithmes de décision pour identifier la meilleure.

Par ailleurs, l'observation et l'évaluation de l'exécution d'une application interactive sont souvent basées sur l'analyse de grands volumes de données porteuses d'informations contextualisées, collectées pendant l'exécution, appelées *traces* (Laflaquière *et al*., 2006). Ainsi, au cours de l'exécution de l'application, la génération des traces permet de renforcer le système d'information avec des éléments qui peuvent permettre de faciliter la décision pour l'analyse et l'adaptation de la logique d'exécution. Nous proposons d'améliorer le processus de décision par l'utilisation des traces des exécutions précédentes. Un système de traces collecte les traces générées par l'utilisateur au cours de l'interaction avec le système. Puis, ces traces servent à analyser les usages dans la phase de décision (Doumat *et al*., 2010).

En combinant les aspects de décision et de traces, nous avons proposé dans (Ho *et al*., 2014) un processus général de décision multicritère à base de traces (illustré en figure 1). Nous avons repris le modèle du système d'aide à la décision classique en ajoutant les traces dans les trois dernières étapes du processus général : celle de la pondération des critères (Étape 2) (Ho *et al*., 2014), celle de la prise de décision (Étape 4) (Ho *et al*., 2015 ; 2016) et celle d'identification des solutions candidates (Étape 3), objet du présent article.

Il existe différents mécanismes de décision multicritère (Köksalan *et al*., 2011 ; Russell, Norvig, 2010). La plupart des mécanismes considèrent toutes les situations possibles pour prendre la décision, à chaque étape de décision. Le calcul s'effectue alors sur un grand ensemble de solutions. Pour surmonter ce problème, nous proposons de réduire l'espace des solutions possibles (nombre de situations) avant de lancer l'algorithme de décision (l'étape 3 dans la figure 1) en exploitant les traces d'exécution. Le présent article décrit en détail l'approche de réduction de l'espace de solutions pour la prise de décision dans un contexte à base de situations.

Pour illustrer et valider notre approche, nous avons considéré le jeu Tamagotchi comme cas d'étude. Notre prototype du jeu Tamagotchi est un jeu interactif dans lequel le joueur doit s'occuper d'un Tamagotchi, un animal virtuel que le joueur doit soigner, nourrir, faire dormir, faire jouer... En nous positionnant dans les applications interactives structurées en situations, nous contextualisons les tranches de la vie du Tamagotchi en différentes situations : *Nourrir*, *Jouer*, *Dormir*... Ainsi l'exécution de

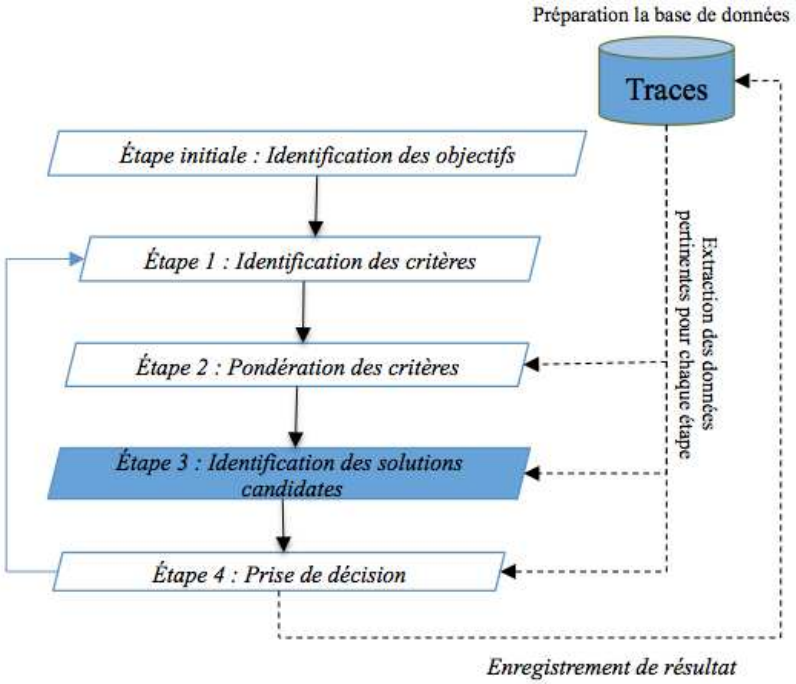

Figure 1. Processus de décision multicritère à base de traces

l'application consiste en un enchaînement de ces situations et la décision porte sur la sélection de la situation la plus appropriée à un instant donné.

Notre article est organisé comme suit. La section 2 est dédiée au positionnement de notre approche par rapport aux travaux existants autour de la réduction de l'espace de solutions possibles. Nous y abordons également les différentes techniques qui pourraient être utilisées pour résoudre notre problématique. La section 3 décrit les concepts et principes d'un système de gestion de traces. Ces traces seront utilisées dans notre approche présentée dans la section 4. La validation et l'évaluation de notre approche sur l'étude de cas Tamagotchi seront présentées dans la section 5. Enfin, nous conclurons et décrirons les perspectives dans la section 6.

2. Positionnement des travaux

La prise de décision multicritère vise à trouver une solution à partir d'un ensemble d'alternatives en synthétisant les données de différents points de vue et selon des paramètres différents provenant de différentes sources. Il existe plusieurs méthodes de décision telles que : les méthodes de pondération telles que WSW (*Weighting Sum Model*) (Triantaphyllou *et al*., 1998) ou WPM (*Weighting Product Model*) (Triantaphyllou *et al*., 1998), la méthode de l'utilité de multi-attributs MAUT (*Multi-Attributes Uti-*

lity Theory) (Russell, Norvig, 2010) et les méthodes de sur-classement telles que PROMETHEE (Taillandier, Stinckwich, 2011 ; Behzadian *et al*., 2010) ou ELECTRE (Corrente *et al*., 2013 ; Hatami-Marbini, Tavana, 2011). Cependant, toutes ces méthodes ne se concentrent que sur la recherche de la solution finale avec deux hypothèses qui sont : i) tous les critères ont été pondérés, ii) l'ensemble des alternatives est disponible. En ce qui concerne ces deux hypothèses, notre article vise principalement à surmonter le point ii) et de fournir une approche pour déterminer les alternatives pertinentes pour la prise de décision. Notre méthode est générale et peut être intégrée à n'importe quel algorithme de décision.

En ce qui concerne la présélection des alternatives en utilisant les traces du système, il y a quelques d'études qui tiennent compte de cet aspect. Le raisonnement à base de cas (CBR) (Riesbeck, Schank, 2013) est une méthode qui explore les traces obtenues lors des exécutions précédentes pour faire de la présélection. CBR imite le raisonnement humain et tente de résoudre de nouveaux problèmes en réutilisant des expériences antérieures. Pour chaque problème rencontré, le système a tracé les informations associées à ce problème ainsi que la solution pour le résoudre, appelé *cas*. L'idée principale de CBR est basée sur l'hypothèse que des problèmes similaires auront des solutions similaires. CBR mesure la similarité entre le problème actuel et les derniers enregistrés pour tenter d'appliquer la solution choisie à ce moment-là, qui a également été enregistrée. Si plusieurs solutions ont été envisagées dans le passé pour le même problème, les techniques d'exploration de données sont utilisées pour évaluer celle qui est la plus appropriée (Guo *et al*., 2011). Cette approche à base de cas est utilisée dans plusieurs types de jeux vidéo. Par exemple, un système CBR, intégré dans le jeu Civilisation C-evo (Sánchez-Pelegrín *et al*., 2005), est utilisé pour aider le joueur à sélectionner une action à faire quand il rencontre un problème n'ayant jamais apparu. Le système filtre les cas les plus proches en utilisant une fonction de calcul de similarité pour calculer le niveau de gain de chaque cas. Le cas ayant le gain le plus haut sera proposé aux joueurs. Dans le jeu RoboCup (championnat de football pour les robots), le CBR est appliqué pour construire des stratégies planifiées pour les robots en se basant sur les expériences passées (Karol *et al*., 2004 ; Marling *et al*., 2003). CBR a également montré ses performances dans la conception des comportements des robots dans des environnements incertains, dynamiques et en temps réel (Ros *et al*., 2009). Enfin, dans (Ontañón, Ram, 2011), les auteurs focalisent sur les techniques permettant aux utilisateurs qui n'ont pas de connaissances techniques de créer les jeux vidéo par l'utilisation CBR.

Dans (Burke, 2007 ; Cheetham, 2003), les auteurs calculent la distance pour déterminer quel élément peut être recommandé à l'utilisateur. La distance augmente lorsque la similitude est faible. Cependant, l'inconvénient majeur de cette approche est le temps de calcul. Nous devons considérer toutes les données existantes lorsque nous voulons calculer la distance pour un état. La méthode ne fournit pas un modèle général permettant d'éviter de recalculer les distances quand les nouvelles données arrivent. Dans (Dang *et al*., 2013), les auteurs utilisent la logique linéaire pour identifier toutes les alternatives possibles dans les applications à base de situations en analysant les entrées de la situation actuelle, appelées pré-conditions. Cette méthode est

très intuitive car elle effectue simplement la vérification entre les préconditions, qui font partie de la structure de la situation, et l'état actuel du système. Cependant, cette méthode nécessite une structuration en situations avec des conditions préalables et la logique linéaire ne fournit pas une mesure permettant de quantifier la pertinence des alternatives comme dans l'approche de distance ci-dessus. Cette mesure est nécessaire afin de comparer et de trier les situations disponibles.

Dans notre travail, nous voulons fournir un ensemble de situations présélectionnées pour les algorithmes de décision selon l'état actuel du système. Comme dans l'approche CBR, nous considérons l'état courant du système, nous prenons ensuite une décision en utilisant des informations enregistrées concernant les exécutions précédentes, appelé *traces*. Ces traces permettent de déterminer quelles sont les situations qui ont été choisies dans le passé dans un contexte similaire. Pour appliquer ce principe, nous intégrons un système de gestion de traces. Grâce aux traces collectées, nous calculons la probabilité d'être exécutable pour chaque situation parmi celles disponibles. Pour y parvenir, nous nous rattachons à l'apprentissage supervisé (Cornuéjols, Miclet, 2011) qui est une technique d'apprentissage automatique où l'on cherche à produire des règles et des modèles à partir d'une base de données observées et validées. Dans notre contexte, nous disposons d'une base de traces contenant les exécutions antérieures, nous appliquons dessus les techniques existantes pour construire un modèle d'apprentissage permettant de prédire les situations pertinentes. Parmi les méthodes de l'apprentissage supervisé, nous avons considéré les quatre méthodes les plus utilisées (Tan *et al*., 2006) : Bayésienne Naïve, Réseaux de neurones, k-plus proches voisins, Support Vector Machine (SVM) pour prédire quelles sont les situations pertinentes. Parmi elles, nous avons retenu la méthode Bayésienne Naïve (Ho, 2015) comme décrit dans la section 4.1.

De plus, il nous faut prendre en compte l'évaluation des situations selon plusieurs critères pour choisir celle qui peut apporter la meilleure *utilité* (Podinovski, 2014) : meilleur impact de la situation sur la progression de la valeur des critères considérés d'après les exécutions précédentes. La notion d'*utilité* a été souvent utilisée dans les systèmes d'IA et fait partie des méthodes d'apprentissage par renforcement (Sutton, Barto, 2012). Ce type d'apprentissage permet d'apprendre à partir d'expériences afin d'obtenir une récompense quantitative au cours du temps. Un système à base de l'utilité permet d'estimer le gain qu'apporte une action si elle est choisie mais ne tient pas compte du moment de l'exécution ou de l'action exécutée. Les actions potentielles sont considérées en fonction d'une variété de facteurs (Mark, 2016). Un des exemples les plus populaires d'un système d'IA à base de l'utilité est le jeu The Sims (Jeuxvidéo.com, 2008 ; Evans, 2009). Chaque action potentielle dans le jeu est évaluée en fonction d'une combinaison des besoins actuels pour estimer la capacité de satisfaction de ces besoins par l'action considérée. Chaque action est associée à une somme pondérée permettant de déterminer quelle action est « la meilleure » à l'instant donné. L'action avec le score le plus élevé sera celle choisie. Par ailleurs, dans (Dill, Mark, 2010 ; 2012) les auteurs ont montré comment utiliser la théorie de l'utilité pour modéliser la décision et l'appliquer dans les jeux vidéo. La méthode Q-learning (Watkins, Dayan, 1992) est une autre méthode typique pour l'apprentissage par renforcement.

Cependant, le temps de calcul de Q-learning augmente avec le nombre d'exemples dans la base de données. Ainsi, elle ne convient pas pour le traitement en temps réel nécessaire dans les jeux vidéos. Nous avons choisi de transposer la notion de *l'utilité* dans notre contexte à base de situations afin d'estimer l'apport que chaque situation aura si elle est choisie pour être exécutée.

En résumé, pour résoudre nos deux problèmes : i) *la construction d'un modèle de prédiction de la probabilité d'être exécutable des situations à un moment donné* et ii) *l'estimation de l'impact de chaque situation pré-sélectionnée*, nous proposons de considérer une approche hybride intégrant l'apprentissage supervisé (méthode Bayésienne) et l'apprentissage par renforcement (notion d'utilité).

Notre approche est basée sur les traces générées lors de l'interaction entre l'utilisateur et système. Avant de décrire en détail notre approche de présélection des situations candidates, nous présentons dans la section suivante notre système à base de traces.

3. Système à base de traces

Nous commençons par définir les concepts de traces et de modèle de traces (Settouti *et al*., 2009) :

– *Observé* : les observés sont des données relatives à une observation faite d'une activité ;

– *Trace* : une trace est une information ou une séquence d'informations générée par toute action relative à un objet ou un élément. Dans les systèmes informatiques, la trace est considérée comme la suite des observés ;

– *Modèle de trace* : un modèle de trace est un modèle représentant la trace d'une manière formelle. Il contient notamment les propriétés et attributs de la trace : date et heure, identifiant d'utilisateur, action réalisée, etc ;

– *Trace modélisée* : une trace modélisée ou m-trace est une trace qui est associée avec son modèle de trace ;

– *Système à base de traces* : un système à base de traces (SBT) est un système informatique permettant et facilitant l'exploitation des traces.

Lors de l'exécution, le système à base de traces fonctionne en respectant les trois sous-systèmes suivants : collecte de traces, transformation de traces et exploitation de traces. Nous décrivons sommairement chacun d'eux ci-après.

3.1. Collecte de traces

La collecte des traces permet l'observation et la récupération de plusieurs sources de données générées lors de l'exécution d'une application interactive. Elle sert à convertir ou à transformer des informations générées par l'interaction utilisateur/système pendant l'activité en une trace initiale, dite *trace première*. Dans notre contexte, nous avons identifié trois observés qui sont trois sources de traçage.

L'exécution de l'application est un enchaînement des situations. L'utilisateur participe à une situation à la fois. Pour savoir ce qui s'est passé au cours du déroulement, nous avons besoin d'observer les situations exécutées à chaque instant en analysant les interactions effectuées. C'est la première source de traçage. À chaque instant de l'exécution, nous disposons du vecteur d'état représentant l'état courant du système. Ce vecteur d'état, deuxième source de traçage, contient tous les attributs et paramètres définis par le concepteur et contribuant au fonctionnement de l'application. Quand l'utilisateur utilise l'application, il veut atteindre un certain objectif. Cet objectif est évalué par plusieurs critères. Nous ajoutons les critères en tant que troisième source de traçage. Nous cherchons à observer l'état ainsi que les valeurs associées à chaque critère, identifié au préalable, pendant l'exécution de l'application.

En nous basant sur ces trois sources de traçage, nous définissons trois types d'observés composant les traces : l'observé du système, l'observé de situation et l'observé de critère. Notre SBT collecte les données issues des interactions. Les traces collectées représentent les traces premières, notées T_P. Chaque trace première est associée à un modèle de traces permettant de recueillir les informations selon les trois observés définis. Ce modèle se compose de 3 éléments $\{V,S,C\}_t$ qui décrivent l'exécution à un moment *t* dans laquelle :

– *V* est le vecteur d'état du système, il rassemble tous les états contribuant à l'exécution de l'application ;

– *S* est la situation exécutée ;

– *C* est l'ensemble des valeurs des critères définis.

Lors de l'exécution de l'application, nous récupérons les informations selon ce modèle, nous obtenons une base des traces premières T_P. Ces traces sont conservées pour être utilisées dans les sous-systèmes suivants.

3.2. Transformation de traces

Les traces récupérées précédemment, T_P, sont des informations brutes. Nous avons besoin de les transformer pour extraire les données exploitables. Le résultat de la transformation est une base des traces transformées notée T_T. Dans notre contexte, nous avons besoin d'une base de traces transformées dans laquelle chaque enregistrement se rapporte au contexte suivant : Quel est l'état du système ? Quelle situation a été choisie pour être exécutée ? Quel est le changement dans l'évaluation des critères ?

Depuis la base de traces premières, nous extrayons des traces transformées dont chaque enregistrement contient les composants suivants :

– Ω = *{*$V \times S$*}* : est l'ensemble du vecteur d'état *V* associé à la situation exécutée *S*. Un enregistrement dans Ω a le format suivant :

$$< V = (att_1,\ att_2,...,att_m),\ S = sit_i >$$

– Δ = *{*$C_{avant} \times S \times C_{après}$*}* : l'ensemble des critères et leurs valeurs d'évaluation avant et après l'exécution de la situation *S*. Un enregistrement dans Δ est

représenté par :

$$< \{\gamma_{avant}(h)\}, S = sit_i, \{\gamma_{après}(h)\} >$$

3.3. Exploitation de traces

Dans notre SBT, la prise de décision sera abordée dans le sous-système d'exploitation de traces. Selon la figure 1, nous avons besoin d'analyser les traces pour la pondération de critères (étape 2), l'identification des solutions candidates (étape 3) et la prise de décision (étape 4). Dans la prochaine section, nous allons présenter comment nous explorons les traces pour effectuer la présélection des candidats pour la prise de décision (étape 3).

4. Approche de présélection des candidats multicritère à base de traces

Nous présentons dans cette section notre approche de présélection des situations candidates permettant de construire l'espace des solutions pour la prise de décision. Pour ce faire, nous utilisons la base de traces transformées obtenue dans la section 3.2. Notre stratégie comporte deux étapes :

1. la prédiction des situations potentielles ;
2. l'estimation de l'utilité de ces situations potentielles.

La première étape consiste à évaluer la capacité de chaque situation disponible à être potentiellement exécutable à la prochaine exécution en fonction de l'état courant du système. Nous obtenons ainsi un ensemble de probabilités associées à toutes les situations. Une *situation potentielle* est celle qui a une probabilité calculée supérieure à un seuil prédéfini et fixé par le concepteur ou l'utilisateur. Si une situation est considérée comme potentielle, nous l'ajouterons à l'ensemble des situations potentielles.

La seconde étape calcule pour chaque situation potentielle sa pertinence selon les critères définis. Cette mesure, appelée *utilité*, permet aux utilisateurs de comparer la pertinence des situations obtenues dans la première étape. La combinaison de ces deux étapes nous aide à déterminer quelles sont les situations candidates pour le processus décisionnel et de quantifier leur impact potentiel sur l'évolution des critères de l'application.

4.1. Prédiction des situations potentielles

Nous allons nous intéresser à notre stratégie de prédiction des situations potentielles à base de traces. Nous avons utilisé la partie Ω de la base de traces transformées extraite dans la section 3.2 pour construire le modèle de prédiction. Les traces obtenues sont analysées pour prédire quelles sont les situations potentielles en fonction de l'état actuel du système et d'un seuil d'acceptation défini par l'utilisateur. Pour construire le modèle de prédiction, nous avons testé les méthodes existantes de la

fouille de données comme indiqué dans la section 2 : Bayésienne Naïve (Domingos, Pazzani, 1997 ; J.Hand, Yu, 2001), Réseaux de neurones (Tan *et al*., 2006), k plus proches voisins (Wu *et al*., 2007) et SVM (Vapnik, 2000). Nous avons retenu l'algorithme Bayésienne Naïve parce qu'il est le plus adapté à notre problème. La comparaison détaillée de ces quatre méthodes est décrite dans (Ho, 2015). En effet, les données que nous manipulons sont de différents types (continues et discrètes). Ceci est la raison pour laquelle nous n'utilisons pas SVM bien que sa performance de prédiction soit très élevée. En raison de la petite taille de notre base de données, la méthode des k plus proches voisins ne nous permet pas d'obtenir une performance suffisante. Entre les deux méthodes restantes, Bayésienne Naïve et Réseaux de neurones, nous nous rendons compte que Réseaux de neurones doit choisir un nombre de neurones et le temps de construction du modèle de prédiction est très élevé. Cela ne peut convenir à une application interactive en temps réel de type jeu informatisé. En outre, le procédé de Bayésienne est simple et facile à mettre en œuvre. Pour conclure, nous considérons que la méthode Bayésienne Naïve est la plus appropriée pour notre cas.

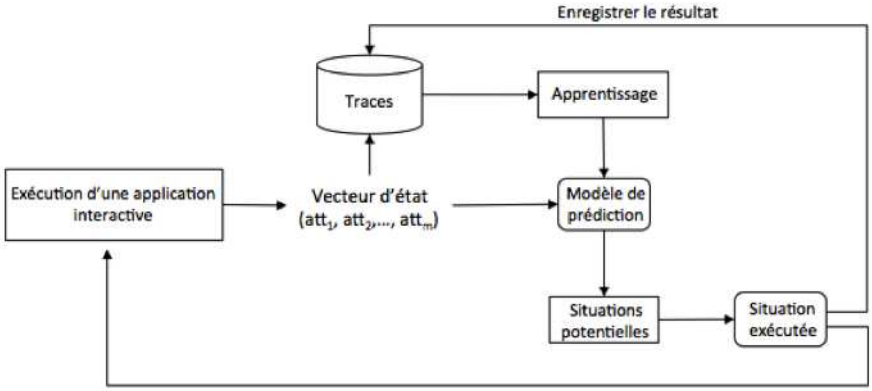

Figure 2. Processus de prédiction à base de traces des situations potentielles

Nous construisons maintenant le modèle de prédiction en appliquant la méthode Bayésienne Naïve. Le processus détaillé est présenté dans l'algorithme suivant.

Algorithme 1 – Prédiction des situations potentielles

Entrées: le vecteur d'état V, l'ensemble S de n situations et le seuil s

initialiser $Pot = \varnothing$ ▷ l'ensemble des situations potentielles
pour tout $sit_i \in S$ **faire** ▷ i de 1 à n
  calculer $Prédiction(sit_i)$
  si $Prédiction(sit_i) \geq s$ **alors**
    $Pot = Pot \cup \{sit_i\}$
  fin si
fin pour

Sortie: l'ensemble des situations potentielles Pot

La probabilité d'une situation *i* d'être potentiellement exécutable à partir d'un état *V = (*att_1*,* att_2*,...,*att_m*)*, noté $Prédiction(sit_i)$ est calculée par :

$$Prédiction(sit_i) = \frac{P(sit_i/V)}{\sum_{i=1}^{n} P(sit_i/V)} \quad (1)$$

Avec $P(sit_i/V)$ est la probabilité a posteriori de la situation i sachant le vecteur d'état V, elle est calculée par :

$$P(sit_i/V) = P(sit_i) \times \prod_{j=1}^{m} P(att_j/sit_i) \quad (2)$$

Dans (2), il nous faut calculer $P(att_j/sit_i)$. Elle dépend du type de donnée de l'attribut att_j. Si elle est numérique, nous supposons que ses valeurs respectent la distribution de Gauss et la probabilité de att_j sachant sit_i est alors donnée par :

$$P(att_j/sit_i) = \frac{1}{\sqrt{2\pi} \times \sigma_j^i} \times e^{\frac{-(att_j - \mu_j^i)^2}{2 \times (\sigma_j^i)^2}} \quad (3)$$

Avec μ_j^i et σ_j^i respectivement la moyenne et l'écart type de l'attribut j pour la situation i. Si la valeur de att_j n'est pas numérique, nous devrons calculer la probabilité ou la fréquence d'occurrence de att_j sachant la situation i dans la base de données pour obtenir la probabilité $P(att_j/sit_i)$.

Après avoir calculé pour chaque situation, la probabilité d'être potentiellement exécutable, nous obtenons la liste des résultats. Nous devons ensuite effectuer la vérification par la comparaison avec le seuil s. La valeur du seuil dépend du choix de l'utilisateur ou de concepteur. Le choix du seuil sera présenté dans notre cas d'étude dans la section 5. Une situation est considérée comme *situation potentielle* si sa probabilité est supérieure ou égale à s.

4.2. Estimation de l'utilité des situations potentielles

Cette étape vise à estimer, pour chaque situation potentielle, son utilité. Cette dernière représente l'impact de la situation sur la progression de la valeur des critères considérés d'après les exécutions précédentes. Pour évaluer l'utilité de chaque situation potentielle, nous avons besoin d'analyser les traces laissées dans les exécutions antérieures. Depuis la base de traces transformées, nous extrayons l'ensemble Δ qui contient des enregistrements dans lesquels sont décrits : la situation exécutée et le changement des valeurs des critères avant et après l'exécution de cette situation.

Nous commençons par le calcul de la déviation d de chaque critère. Cette valeur est la différence entre l'évaluation du critère h après ($\gamma_{après}$) et avant (γ_{avant}) avoir

exécuté une situation. Supposant que nous considérons l'enregistrement j dans la base de traces, la déviation d'un critère h est obtenue par :

$$d_h^j = \gamma_{après}^j(h) - \gamma_{avant}^j(h) \tag{4}$$

Si nous avons une base contenant q enregistrements, la déviation globale du critère h est calculée par :

$$d_h = \frac{\sum_{j=1}^{q}(\gamma_{après}^j(h) - \gamma_{avant}^j(h))}{q} = \frac{\sum_{j=1}^{q} d_h^j}{q} \tag{5}$$

Nous utilisons la déviation obtenue pour définir notre fonction d'utilité $u(d_h)$ où p_h représente le seuil d'acceptation d'une déviation d'un critère h. Ce seuil est défini expérimentalement par le concepteur de l'application. Chaque critère a un seuil différent. La valeur d'utilité d'un critère h est calculée en fonction de la déviation obtenue dans (5). Nous voulons avoir une fonction qui détermine si la valeur des critères apporte une utilité positive ou négative. Cette fonction doit se baser sur la déviation entre les valeurs pour identifier l'utilité des critères. Par conséquent, nous fixons un seuil p_h pour bien distinguer le niveau d'utilité comme montré dans (6).

$$u(d_h) = \begin{cases} -1 & \text{si} \quad d_h < 0 \\ 0 & \text{si} \quad d_h = 0 \\ \dfrac{d_h}{p_h} & \text{si} \quad 0 < d_h < p_h \\ 1 & \text{si} \quad d_h \geq p_h \end{cases} \tag{6}$$

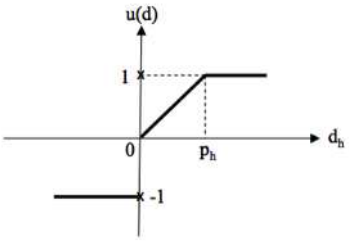

Figure 3. Fonction d'utilité

Comme illustré par figure 3, si la déviation est négative, la situation exécutée a diminué la valeur d'évaluation du critère, la valeur d'utilité correspondante sera négative. Si la déviation est supérieure à un seuil p_h, la valeur d'évaluation du critère a augmenté après l'exécution de la situation, l'utilité sera égale à 1. L'utilité sera égale à 0 si la déviation est égale à 0. Autrement, l'utilité associée vaudra $\frac{d_h}{p_h}$: plus la déviation approche du seuil, plus l'utilité approche linéairement de 1.

Algorithme 2 – Estimation de l'utilité des situations potentielles

Entrées: l'ensemble des situations potentielles Pot, l'ensemble de m critères

initialiser $Cand = \varnothing$ ▷ l'ensemble des situations candidates
pour tout $sit_i \in Pot$ **faire**
  initialiser $U = \varnothing$ ▷ l'ensemble des critères qui ont des utilités positives
  pour tout h **faire** ▷ h de 1 à m
    calculer $u(d_h)$
    si $u(d_h) \geq 0$ **alors**
      $U = U \cup \{h\}$
    fin si
  fin pour
  si $size(U) \geq K$ **alors**
    $Cand = Cand \cup \{sit_i\}$
  fin si
fin pour

Sortie: l'ensemble des situations candidates $Cand$

Nous appliquons cette approche pour estimer l'utilité des m critères de l'application pour chaque situation potentielle obtenue ci-dessus. Une situation potentielle est considérée comme *candidate* s'il existe au moins K critères sur m dont l'utilité est supérieure à 0. Dans le cas contraire, elle est *non-candidate*. Toutes les situations obtenues au bout de cette phase constituent un ensemble de candidates pour l'algorithme de décision multicritère choisi. L'algorithme 2 résume notre méthodologie.

5. Cas d'étude : jeu Tamagotchi

5.1. Description du jeu Tamagotchi

Le principe général est bien connu. Il s'agit de s'occuper d'un animal virtuel appelé Tamagotchi [1]. Nous considérons la vie du Tamagotchi à partir du début : Tamagotchi est un œuf au départ et l'utilisateur doit le faire couver afin qu'il éclose. À partir de ce moment, lorsque le Tamagotchi a besoin de quelque chose, un petit signal va apparaître pour avertir l'utilisateur afin qu'il intervienne. L'utilisateur a accès à l'état complet du Tamagotchi à travers divers indicateurs afin d'agir de façon appropriée, par exemple : donner à manger, jouer, entretenir ou éduquer... Nous avons identifié 7 situations possibles tout au long de l'exécution du jeu Tamagotchi. Elles sont :

– *Nourir* : donner à manger ou à boire au Tamagotchi ;

– *Entretenir* : nettoyer l'environnement où le Tamagotchi vit ou entretenir le Tamagotchi ;

– *Jouer* : permettre au Tamagotchi de s'amuser pour distraire le Tamagotchi ;

1. https://fr.wikipedia.org/wiki/Tamagotchi

– *Soigner* : si le Tamagotchi est malade, il a besoin d'être soigné ;

– *Dormir* : il faut faire dormir le Tamagotchi quand il est fatigué ;

– *Socialiser* : aider le Tamagotchi à avoir des amis ; on le met en contact avec d'autres Tamagotchi (gérés par l'application) ;

– *Éduquer* : apprendre les bonnes manières au Tamagotchi.

Nous avons défini le vecteur d'état du Tamagotchi qui est constitué de plusieurs attributs. Chaque attribut est une propriété décrivant une partie de son état global. Les différents attributs sont présentés dans le tableau 1.

Tableau 1. Les attributs du Tamagotchi

Attribut	**Valeur**	**Description**
satiété	[0,1]	La satiété du Tamagotchi admet une valeur de 0 (il a faim) jusqu'à 1 (il n'a pas faim)
fatigue	[0,1]	Très fatigué (0) → Pas fatigué (1)
ennui	[0,1]	Ennui maximum (0) → Ennui minimum (1)
soin	[0,1]	Cette valeur décrit le soin à accorder au Tamagotchi; plus cette valeur augmente, moins il a besoin de soin
amis	Valeur entière	Le nombre total des amis du Tamagotchi
politesse	[0,1]	Impoli (0) → Poli (1)

Figure 4. Interface du jeu Tamagotchi

De plus, nous avons défini 3 critères visant à évaluer l'objectif du jeu. Le premier critère vise à maintenir le Tamagotchi en vie jusqu'à la fin du jeu. Ce critère concerne sa *santé*. Le deuxième critère concerne la *socialisation* du Tamagotchi. Ce critère vise à faire évoluer son niveau social. Le dernier critère vise à éduquer le Tamagotchi et permet d'évaluer sa *maturité*. Les formules de calcul des valeurs des critères en fonction les valeurs des attributs sont décrites dans (Ho, 2015). Ces fonctions sont définies par le concepteur. L'interface du jeu Tamagotchi est montrée dans la figure 4.

Au début du jeu, l'utilisateur arrive à l'étape initiale, Naître. Ensuite, il enchaîne les différentes situations parmi les 7 disponibles. L'utilisateur devra notamment veiller à maintenir en vie le Tamagotchi. En effet, certaines actions peuvent mener à sa mort (s'il n'est pas bien soigné, ou s'il s'ennuie trop). Nous n'abordons pas l'ensemble des règles du fonctionnement du jeu ici, car ce n'est pas l'objet du présent l'article. Le comportement complet du jeu peut être trouvé dans (Ho, 2015). Nous décrivons dans la suite comment appliquer notre approche de présélection des situations candidates quand l'utilisateur finit une situation.

5.2. Application de l'approche de présélection des situations candidates

Notre approche de présélection est divisée en deux phases : celle d'identification des situations potentielles et celle d'estimation de l'utilité des situations potentielles. Nous disposons d'une base de traces pour effectuer le calcul (voir ci-après). Nous extrayons depuis cette base de traces premières deux bases de traces transformées (Ω et Δ comme décrit dans la section 3.2). Chacune correspond à la base de données utilisée pour chaque phase ci-dessus. La figure 5 et la figure 6 donnent un aperçu du contenu de ces deux bases.

Phase d'identification des situations potentielles

Chaque enregistrement comporte sept champs dont les six attributs du Tamagotchi (le vecteur d'état) et un dernier élément déterminant la situation exécutée : *Jouer*, *Dormir*, etc. Grâce à notre prototype, nous avons recueilli les données réelles lors de l'exécution du jeu par les utilisateurs. Notre base de traces [2] est disponible pour tester notre méthode. Quantitativement, nous avons 9315 traces qui se composent de 2261 situations *Nourrir*, de 188 situations *Entretenir*, de 559 situations *Jouer*, de 1935 situations *Soigner*, de 2217 situations *Dormir*, de 1548 situations *Socialiser* et de 607 situations *Éduquer*. Nous avons utilisé ces informations pour construire notre modèle de prédiction des situations potentielles.

Comme décrit dans la section 4.1, nous avons besoin d'un modèle de prédiction des situations potentielles parmi les situations disponibles. Pour construire ce modèle, il nous faut considérer tous les enregistrements dans la base extraite dans la figure 5 et appliquer le principe Bayésienne Naïve. Puisque les attributs dans cette base sont des valeurs numériques, il faut calculer la moyenne et l'écart type de chaque attribut.

2. Tamagotchi Traces: https://app.box.com/s/nw3jty6sfiohk9rg75uztn7yabvepkt2

L'ensemble des moyennes et des écarts types constitue notre modèle de prédiction des situations potentielles.

Relation: Tamagotchi

No.	satiété Numeric	fatigue Numeric	ennui Numeric	soin Numeric	amis Numeric	politesse Numeric	situation Nominal
1	0.97	0.42	0.93	0.31	2.0	0.1	Soigner
2	0.47	0.7	0.06	0.68	6.0	0.47	Socialiser
3	0.48	0.29	0.21	0.02	6.0	0.67	Soigner
4	0.62	0.59	0.34	0.27	7.0	0.32	Jouer
5	0.03	0.51	0.35	0.46	8.0	0.33	Nourrir
6	0.39	0.19	0.53	0.85	0.0	0.71	Dormir
7	0.46	0.34	0.2	0.35	4.0	0.49	Socialiser
8	0.56	0.83	0.42	0.8	5.0	0.95	Entretenir
9	0.24	0.76	0.57	0.78	1.0	0.24	Nourrir
10	0.38	0.3	0.29	0.78	1.0	0.65	Jouer
11	0.56	0.45	0.55	0.74	0.0	0.42	Dormir
12	0.46	0.63	0.09	0.59	0.0	0.91	Jouer
13	0.48	0.14	0.14	0.31	4.0	0.36	Dormir
14	0.97	0.52	0.47	0.8	3.0	0.61	Socialiser
15	0.07	0.22	0.66	0.32	0.0	0.39	Nourrir

Figure 5. Base de données pour la prédiction des situations potentielles

Afin d'illustrer comment identifier les situations potentielles selon le modèle de prédiction construit, nous prenons un petit exemple comme suit : le contexte du système à un instant donné est défini par le vecteur $V = (satiété = 0{,}03\,; fatigue = 0{,}26\,; ennui = 0{,}04\,; soin = 0{,}09\,; amis = 2\,; politesse = 0{,}7)$. Selon V, nous vérifions quelles sont les situations potentiellement exécutables. Nous décrivons en détail le calcul de prédiction de la situation *Nourrir*.

Deux valeurs $\mu_{satiété}^{Nourrir} = 0{,}45$ et $\sigma_{satiété}^{Nourrir} = 0{,}22$ sont respectivement la moyenne et l'écart type de l'attribut $satiété$ pour la situation *Nourrir*. Pour calculer la probabilité a posteriori de la situation *Nourrir* sachant le vecteur d'état V, noté $P(Nourrir/V)$, nous devons calculer la vraisemblance de chaque attribut du V sachant *Nourrir*, par exemple $P(satiété = 0{,}03/Nourrir)$, etc.

$$P(satiété = 0{,}03/Nourrir) = \frac{1}{\sqrt{2\pi} \times 0{,}22} \times e^{\frac{-(0{,}03+0{,}45)^2}{2\times(0{,}05)^2}} \approx 0{,}19$$

Nous procédons de la même façon pour les autres attributs et nous obtenons : $P(fatigue = 0{,}26/Nourrir)$, $P(ennui = 0{,}04/Nourrir)$, $P(soin = 0{,}09/Nourrir)$, $P(amis = 2/Nourrir)$ et $P(politesse = 0{,}7/Nourrir)$. La probabilité de la situation *Nourrir* d'après les traces est $P(Nourrir) = 2261/9315$.

$$\begin{aligned} P(Nourrir/V) = & P(Nourrir) \times P(satiété = 0{,}03/Nourrir) \times \\ & P(fatigue = 0{,}26/Nourrir) \times P(ennui = 0{,}04/Nourrir) \times \\ & P(soin = 0{,}09/Nourrir) \times P(amis = 2/Nourrir) \times \\ & P(politesse = 0{,}7/Nourrir) \approx 0{,}0012 \end{aligned}$$

Nous calculons ensuite la prédiction pour chaque situation comme décrit dans 4.1. En donnant la valeur du seuil $s = 10\%$ (ce paramètre dépend de l'application considérée). Nous obtenons l'ensemble des situations potentielles dans le tableau 2.

Tableau 2. Résultat de la prédiction des situations potentielles

Situation	Probabilié de prédiction	Résultat
Nourrir	**21,662 %**	**potentielle**
Entretenir	0,0003 %	non potentielle
Jouer	**11,128 %**	**potentielle**
Soigner	**20,3 %**	**potentielle**
Dormir	2,338 %	non potentielle
Socialiser	**16,863 %**	**potentielle**
Éduquer	**27,127 %**	**potentielle**

Selon le tableau 2, l'ensemble des situations potentielles se compose des situations suivantes : *Nourrir*, *Jouer*, *Soigner*, *Socialiser* et *Éduquer*. Pourtant, pour le moment, ce n'est que de la prédiction statistique. Cet ensemble ne prend pas en compte l'aspect multicritère. En effet, il y a des situations avec des probabilités de prédiction très élevées mais qui ne satisferont pas forcement les critères définis. Par conséquent, nous passons à la deuxième phase qui consiste à estimer l'utilité des situations potentielles identifiées.

Phase d'estimation de l'utilité des situations potentielles

Nous allons maintenant considérer la phase de calcul de l'utilité par l'analyse des traces dans la figure 6. Chaque enregistrement dans la figure 6 représente le changement des valeurs de trois critères. En détail, il contient les trois valeurs de l'accomplissement des critères avant d'exécuter la situation choisie et les trois valeurs après l'exécution de la situation choisie.

Relation: Tamagotchi

No.	santéAvant Numeric	socialisationAvant Numeric	maturitéAvant Numeric	situation Nominal	santéAprès Numeric	socialisationAprès Numeric	**maturitéAprès** Numeric
1	0.77	2.47	0.44	Socialiser	1.79	6.03	2.96
2	1.79	6.03	2.96	Soigner	0.58	6.11	4.36
3	0.58	6.11	4.36	Jouer	1.14	7.17	2.41
4	1.14	7.17	2.41	Nourrir	0.65	8.18	2.84
5	0.65	8.18	2.84	Dormir	0.9	0.27	0.53
6	0.9	0.27	0.53	Socialiser	0.95	4.1	2.25
7	0.95	4.1	2.25	Entretenir	1.77	5.21	4.87
8	1.77	5.21	4.87	Nourrir	1.21	1.29	0.37
9	1.21	1.29	0.37	Jouer	1.17	1.15	0.98
10	1.17	1.15	0.98	Dormir	1.2	0.28	0.22
11	1.2	0.28	0.22	Jouer	1.59	0.05	0.16
12	1.59	0.05	0.16	Dormir	0.79	4.07	2.15
13	0.79	4.07	2.15	Socialiser	1.82	3.24	2.02
14	1.82	3.24	2.02	Nourrir	-0.05	0.33	0.45

Figure 6. Base de données pour l'estimation de l'utilité des situations potentielles

Il faut d'abord définir les seuils pour les trois critères. Dans notre contexte, ses valeurs sont : $p_{santé} = 0{,}4$; $p_{socialisation} = 1$; $p_{maturité} = 1$. Ces seuils doivent

être définis par le concepteur ou l'utilisateur. Normalement, ces seuils devraient être définis de manière expérimentale, nous donnons un exemple qui a été expérimenté dans (Ho, 2015). Cependant, dans le cas présent, nous les avons fixé arbitrairement à titre d'exemple. Une fois les seuils définis, nous appliquons (5) et (6) pour calculer l'utilité de chaque situation potentielle. Le résultat de cette phase est résumé dans le tableau 3.

Tableau 3. Résultat d'estimation de l'utilité des situations potentielles

Situation	**Utilité du critère**			**Résultat**
	Santé	*Socialisation*	*Maturité*	
Nourrir	**1**	**0,2**	0	**alternative**
Jouer	-1	1	-1	non alternative
Soigner	**1**	0	**1**	**alternative**
Socialiser	-1	**1**	**0,6**	**alternative**
Éduquer	0	**1**	**1**	**alternative**

Nous choisissons de fixer la valeur de K à 2, c'est-à-dire qu'une situation doit avoir au moins deux valeurs d'utilité positives sur les trois critères pour être considérée comme utile. Selon l'estimation de l'utilité présentée dans le tableau 3, il y a seulement quatre situations (*Nourrir, Soigner, Socialiser* et *Éduquer*) qui font augmenter l'utilité d'au moins 2 critères en se basant sur l'analyse des traces. Si on suit l'une de ces situations, nous prédisons de globalement mieux remplir les critères de l'application. Ainsi, le résultat de notre processus de présélection des alternatives est l'ensemble des quatre situations ci-dessus.

Nous pouvons maintenant appliquer tout algorithme de décision pour calculer le choix final ou laisser à l'utilisateur le choix de sélectionner la situation à exécuter. Pour tout moment lors de l'exécution du jeu Tamagotchi, notre approche ne présélectionne que les situations candidates pour la décision, l'utilisateur a le droit de choisir n'importe quelle situation même si elle est ni utile ni potentielle.

5.3. Évaluation et Discussion

Nous avons effectué des tests de performance pour valider la qualité et la pertinence de notre approche.

Comparaison des méthodes pour l'identification des situations potentielles

Tout d'abord, nous évaluons la performance de la méthode Bayésienne Naïve pour l'identification des situations potentielles en utilisant le logiciel Weka (Hall *et al.*, 2009). Dans le tableau 4, nous avons résumé le taux de prédiction correcte (mesuré avec Weka) et le temps nécessaire (en unités de temps) pour construire le modèle de prédiction en utilisant les quatre méthodes de fouille de données mentionnées à la section 2. Nous utilisons la base de traces obtenue (décrite dans la figure 5) pour effectuer l'évaluation.

Nous voyons que le taux de prédiction correct de la technique k-NN est le plus faible. Pour les 3 autres techniques, la différence du taux de prédiction n'est pas significative. Cependant, même si le taux de prédiction de Bayésienne Naïve n'est pas le meilleur comparé aux deux méthodes (Réseaux de neurones et SVM), cette différence est minime et, rapportée au temps de construction des modèles de prédiction pour ces deux méthodes, on constate qu'on obtient des résultats comparables en un temps plus court.

Tableau 4. Comparaison de performance entre quatre méthodes de prédiction : Bayésienne Naïve, k-NN, Réseaux de neurones, SVM

Méthode	**Taux de prédiction correct**	**Temps pour construire le modèle de prédiction**
Bayésienne Naïve	83,42 %	1 unité
k-NN	78,2 %	Pas besoin de modèle
Réseaux de neurones	83.8 %	108,5 unités
SVM	85.92 %	6,8 unités

Apport de notre approche de présélection dans la prise de décision

Nous avons testé l'intégration de notre approche dans les méthodes de prise de décision multicritère WSM, MAUT et PROMETHEE II pour évaluer l'apport de performance de notre approche dans une application réelle. Nous avons choisi les trois algorithmes représentant les trois familles de décision mentionnées dans la section 2 : celle de pondération (WSM), celle de l'utilité (MAUT) et celle de sur-classement (PROMETHEE II). Pour chaque algorithme, nous avons réalisé deux tests : l'un avec la décision utilisant notre approche de présélection des situations candidates et l'autre sans notre approche. Nous avons observé le temps de calcul de 20 décisions et nous avons obtenu les résultats représentés dans les figures 7, 8 et 9.

Nous observons que le temps de calcul de prise de décision avec la présélection est souvent inférieur à celui sans l'intégration de notre approche. Si nous appliquons la présélection des situations candidates avant de prendre la décision, nous pouvons réduire le nombre d'alternatives et la prise de décision n'a pas besoin de prendre en compte toutes les situations disponibles mais seulement celles qui sont pertinentes.

Toutefois, il existe certains cas où le temps de calcul avec l'intégration de notre approche est plus élevé que la décision sans notre approche, par exemple dans la figure 7 (décisions 9, 14, 17), figure 8 (décisions 4, 9, 14) et figure 9 (décisions 6, 9, 12). La raison est que, dans ces cas, toutes les situations disponibles sont considérées comme des alternatives. Par conséquent, le temps de calcul est plus élevé en raison de la charge de l'exécution de notre approche de présélection des situations candidates. Néanmoins, dans le cas général, nous pouvons remarquer de meilleurs résultats de performance de notre approche. Bien que la différence de temps de calcul pour chaque décision ne soit pas significative entre les deux stratégies, cette différence sera importante si nous considérons le temps cumulé tout au long des décisions au cours de

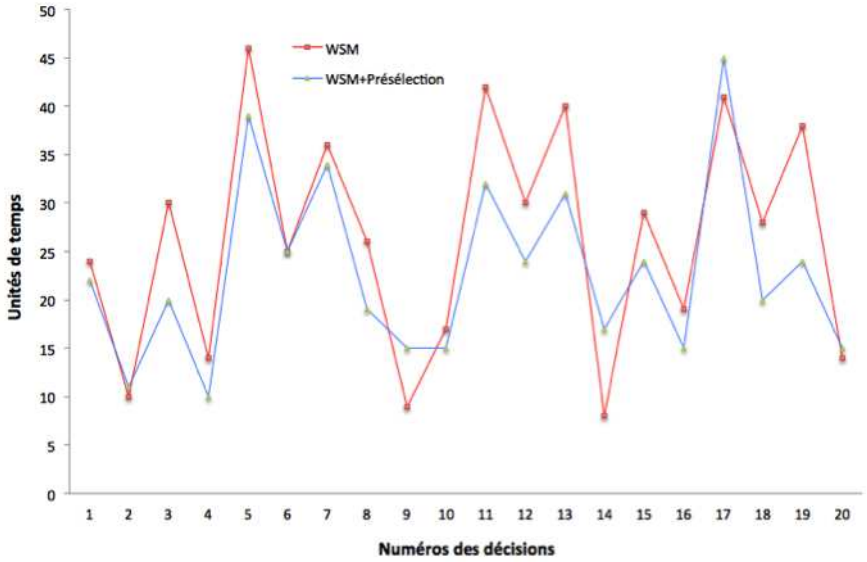

Figure 7. Comparaison du temps d'exécution de l'algorithme de décision WSM avec et sans l'approche de présélection des situations candidates

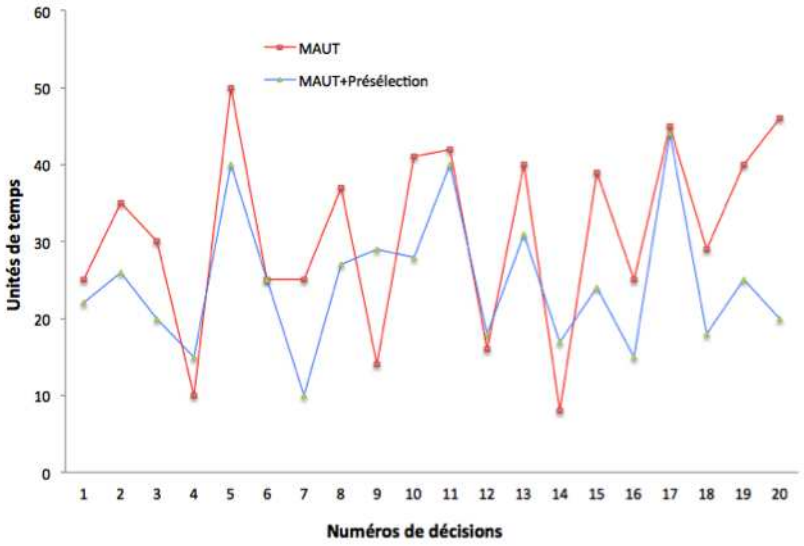

Figure 8. Comparaison du temps d'exécution de l'algorithme de décision MAUT avec et sans l'approche de présélection des situations candidates

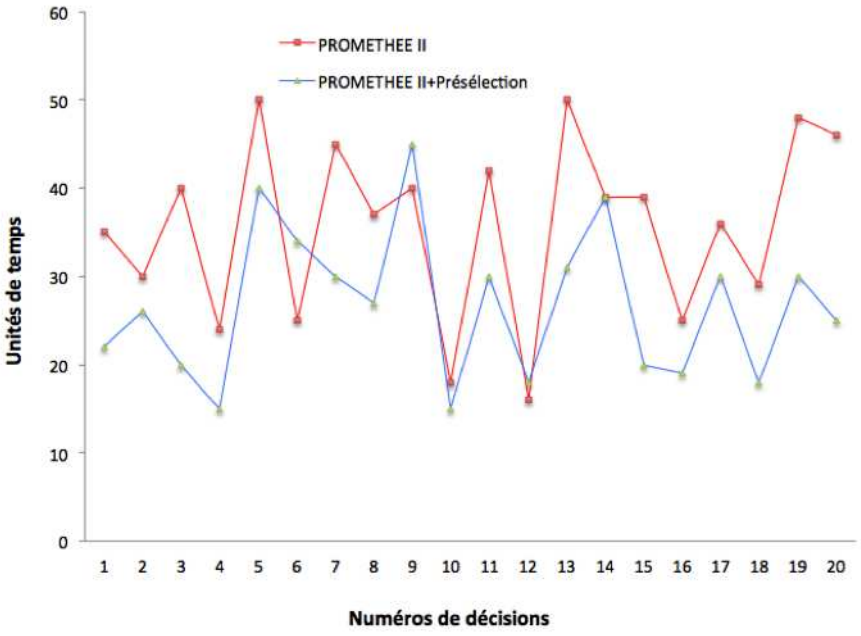

Figure 9. Comparaison du temps d'exécution de l'algorithme de décision PROMETHEE II avec et sans l'approche de présélection des situations candidates

l'exécution de l'application. En outre, cette différence augmente lorsque l'ensemble des situations disponibles de l'application augmente.

5.4. Limites

Nous avons identifié plusieurs limites à notre approche. Elle est efficace si et seulement si nous avons suffisamment de traces dans la base de données. Puisque les calculs sont basés sur l'analyse de traces générées lors des exécutions passées, la taille de la base de traces obtenue est une donnée très sensible. C'est le problème du démarrage à froid. La quantité de traces disponibles influence directement la mise en œuvre notre approche et par conséquent la pertinence des situations candidates obtenues. Pendant les premières exécutions, nous ne disposons pas suffisamment de traces pour construire le modèle de prédiction fiable. Dans ce cas, les utilisateurs doivent décider par eux-mêmes.

Une autre limite de notre méthode est le réglage des seuils s et p. Nous devons éviter de choisir des valeurs élevées, car ils représentent la limite de pertinence pour vérifier la probabilité et l'utilité. Expérimentalement, la valeur de s dans la première phase, l'identification des situations potentielles, doit être de $5\ \% \leq s \leq 10\%$ et la valeur de p dans la seconde phase, l'estimation de l'utilité des situations potentielles,

doit être choisie selon le type et la nature des critères pris en compte. Les choix ci-dessus s'appliquent au jeu Tamagotchi. Pour d'autres applications, nous devons effectuer plusieurs tests afin d'obtenir une valeur de seuil optimale. Le nombre de situations candidates dépend de ces valeurs.

6. Conclusion

Dans cet article, nous avons présenté une stratégie de présélection des situations candidates dans les systèmes interactifs structurées en situations. Notre approche s'appuie sur l'analyse des traces générées au cours de l'exécution. Nous avons construit un système à base de traces adapté à notre contexte et nous avons appliqué la technique bayésienne naïve afin de construire un modèle de prédiction qui nous permet d'identifier les situations potentielles en fonction de l'état actuel. Nous avons estimé ensuite l'utilité de chaque situation potentielle identifiée pour tous les critères définis. En nous basant sur ces valeurs d'utilité, nous identifions quelles sont les situations candidates pour la prise de décision. Notre approche ne modifie pas la structure des situations. Nous utilisons uniquement les exécutions passées du système, enregistrées sous forme de traces.

La contribution principale de cet article est la présélection des alternatives pour l'algorithme de décision en utilisant les traces de système afin de réduire le temps de prise de décision. Nous l'avons appliquée au cas du jeu Tamagotchi et nous avons effectué plusieurs tests pour montrer son efficacité.

Notre approche de présélection est destinée à être intégrée dans un système de décision multicritère qui pourra être appliquée pour différents types d'applications interactives telles que : les jeux sérieux, les dispositifs d'e-éducation, avec l'objectif d'optimiser le fonctionnent des algorithmes de décision. Il est à noter que notre approche pourrait être généralisée à d'autres structurations que celle des situations. En effet, la situation est un bloc de base contextualisant un certain nombre d'actions, interactions ou activités. Elle représente la granularité du découpage de l'exécution de l'application, et par conséquent pourrait être adaptée selon les besoins.

Bibliographie

Behzadian M., Kazemzadeh R. B., Albadvi A., Aghdasi M. (2010). PROMETHEE: A comprehensive literature review on methodologies and applications. *European Journal of Operational Research*, vol. 200, n° 1, p. 198–215.

Brun P., Beaudouin-Lafon M. (1995). A taxonomy and evaluation of formalisms for the specification of interactive systems. In *Proceeding of 6th International Conference on Human-Computer Interaction*, p. 197–212. Huddersfield, United Kingdom.

Burke R. (2007). Hybrid Web Recommender Systems. In Brusilovsky, Peter and Kobsa, Alfred and Nejdl, Wolfgang (Ed.), *The Adaptive Web: Methods and Strategies of Web Personalization*, vol. 4321, p. 377–408. Berlin, Heidelberg, Springer-Verlag.

Cheetham W. (2003). Global Grade Selector: A Recommender System for Supporting the Sale of Plastic Resin. In *5th International Conference on Case-Based Reasoning: Research and Development*, p. 96–106. Norway, Springer-Verlag.

Cornuéjols A., Miclet L. (2011). *Apprentissage artificiel: concepts et algorithmes.* Editions Eyrolles.

Corrente S., Greco S., Słowiński R. (2013, octobre). Multiple Criteria Hierarchy Process with ELECTRE and PROMETHEE. *Omega*, vol. 41, n° 5, p. 820–846.

Dang K. D., Pham P. T., Champagnat R., Rabah M. (2013). Linear logic validation and hierarchical modeling for interactive storytelling control. In D. Reidsma, H. Katayose, A. Nijholt (Eds.), *Advances in Computer Entertainment: 10th International Conference, ACE 2013*, vol. 8253, p. 524–527. Boekelo, The Netherlands,, Springer International Publishing.

Dill K., Mark D. (2010). Improving AI Decision Modeling Through Utility Theory (video content). In *Game Developers Conference 2010.* San Francisco, CA. http://www.gdcvault .com/play/1012410/Improving-AI-Decision-Modeling-Through

Dill K., Mark D. (2012). Embracing the Dark Art of Mathematical Modeling in AI (video content). In *Game Developers Conference 2012.* San Francisco, CA. http://www.gdcvault .com/play/1015683/Embracing-the-Dark-Art-of

Domingos P., Pazzani M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, vol. 29, n° 2-3, p. 103–130.

Doumat R., Egyed-Zsigmond E., Pinon J.-M. (2010). User Trace-Based Recommendation System for a Digital Archive. In Bichindaritz, Isabelle and Montani, Stefania (Ed.), *International Conference on Case-Based Reasoning 2010*, vol. 6176, p. 360–374. Alessandria, Italy, Springer-Verlag.

Evans R. (2009). *AI Challenges in Sims 3 - Invited talk in The Fifth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment Conference.* Palo Alto, California. http://intrinsicalgorithm.com/IAonAI/2009/10/aiide-2009-ai-challenges-in-sims-3 -richard-evans/

Guo Y., Hu J., Peng Y. (2011). Research on CBR system based on data mining. *Applied Soft Computing*, vol. 11, n° 8, p. 5006–5014.

Hall M., Frank E., Holmes G., Pfahringer B., Reutemann P., Witten I. H. (2009). The WEKA Data Mining Software: An Update. *ACM SIGKDD Explorations Newsletter*, vol. 11, n° 1, p. 10–18.

Hatami-Marbini A., Tavana M. (2011). An extension of the Electre I method for group decision-making under a fuzzy environment. *Omega*, vol. 39, n° 4, p. 373–386.

Ho H. N. (2015). *Décision multicritère à base de traces pour les applications interactives à exécution adaptative.* Thèse de doctorat - Université de La Rochelle, France.

Ho H. N., Rabah M., Nowakowski S., Estraillier P. (2014, août). Trace-Based Weighting Approach for Multiple Criteria Decision Making. *Journal of Software*, vol. 9, n° 8, p. 2180–2187.

Ho H. N., Rabah M., Nowakowski S., Estraillier P. (2015). Application of trace-based subjective logic to user preferences modeling. In *20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning (short papers)*, vol. 35, p. 94–105. EPiC Series in Computing.

Ho H. N., Rabah M., Nowakowski S., Estraillier P. (2016). Toward a Trace-Based PROMETHEE II Method to answer " What can teachers do? " in Online Distance Learning Applications. In *13th International Conference on Intelligent Tutoring Systems*, p. 480–484. Zagreb, Croatia.

Jeuxvidéo.com. (2008). *Les Sims passent les 100 millions.* http://www.jeuxvideo.com/news/2008/00025410-les-sims-passent-les-100-millions.htm

J.Hand D., Yu K. (2001). Idiot's Bayes - not so stupid after all? *International Statistical Review*, vol. 69, n° 3, p. 385–398.

Karol A., Nebel B., Stanton C., Williams M.-A. (2004). Case based game play in the robocup four-legged league part i the theoretical model. In D. Polani, B. Browning, A. Bonarini, K. Yoshida (Eds.), *RoboCup 2003: Robot Soccer World Cup VII*, vol. 3020, p. 739–747. Berlin, Heidelberg, Springer Berlin Heidelberg.

Köksalan M., Wallenius J., Zionts S. (2011). *Multiple criteria decision making: From early history to the 21st century*. World Scientific.

Laflaquière J., Settouti L. S., Prié Y., Mille A. (2006). Trace-Based Framework for Experience Management and Engineering. In *Proceedings of the 10th International Conference on Knowledge-Based Intelligent Information and Engineering Systems - Volume Part I*, p. 1171–1178. Bournemouth, UK, Springer-Verlag.

Mark D. (2016). *Intrisic Algorithm - IA on AI.* http://intrinsicalgorithm.com/IAonAI/

Marling C., Tomko M., Gillen M., Alex D., Chelberg D. (2003). Case-based reasoning for planning and world modeling in the robocup small sized league. In *IJCAI Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World Modeling, Planning, Learning, and Communicating*, p. 29–36. Acapulco, Mexico.

Ontañón S., Ram A. (2011). Case-based reasoning and user-generated artificial intelligence for real-time strategy games. In P. A. González-Calero, M. A. Gómez-Martín (Eds.), *Artificial Intelligence for Computer Games*, p. 103–124. New York, NY, Springer New York.

Pham P. T., Rabah M., Estraillier P. (2015). A situation-based multi-agent architecture for handling misunderstandings in interactions. *Applied Mathematics and Computer Science*, vol. 25, n° 3, p. 439–454.

Podinovski V. V. (2014). Decision making under uncertainty with unknown utility function and rank-ordered probabilities. *European Journal of Operational Research*, vol. 239, n° 2, p. 537–541.

Riesbeck C., Schank R. (2013). *Inside case-based reasoning*. Taylor & Francis.

Ros R., Arcos J. L., Mantaras R. Lopez de, Veloso M. (2009). A Case-based Approach for Coordinated Action Selection in Robot Soccer. *Artificial Intelligence.*, vol. 173, n° 9-10, p. 1014–1039.

Russell S. J., Norvig P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Settouti L. S., Prié Y., Cram D., Champin P.-A., Mille A. (2009). A Trace-Based Framework for supporting Digital Object Memories. In *1st International Workshop on Digital Object Memories (DOMe'09) in the 5th International Conference on Intelligent Environments (IE 09)*, p. 39–44. Barcelona, Spain.

Sutton R. S., Barto A. G. (2012). *Introduction to reinforcement learning* (2e éd.). MIT Press.

Sánchez-Pelegrín R., Gómez-Martín M. A., Díaz-Agudo B. (2005). A CBR module for a strategy videogame. In *1st Workshop on Computer Gaming and Simulation Environments, at 6th International Conference on Case-Based Reasoning (ICCBR)*, p. 217–226. Chicago, USA.

Taillandier P., Stinckwich S. (2011). Using the PROMETHEE multi-criteria decision making method to define new exploration strategies for rescue robots. In *IEEE International Workshop on Safety, Security, and Rescue Robotics*, p. 193–202. Kyoto, Japan.

Tan P.-N., Steinbach M., Kumar V. (2006). *Introduction to data mining*. Wesley, Pearson Addison.

Triantaphyllou E., Shu B., Sanchez S. N., Ray T. (1998). Multi-Criteria Decision Making : An Operations Research Approach. *Encyclopedia of Electrical and Electronics Engineering*, vol. 15, p. 175–186.

Vapnik V. (2000). *The Nature of Statistical Learning Theory.* New York, Springer-Verlag New York.

Watkins C. J. C. H., Dayan P. (1992). Q-learning. *Machine Learning*, vol. 8, n° 3-4, p. 279–292.

Wu X., Kumar V., Ross Quinlan J., Ghosh J., Yang Q., Motoda H. *et al*. (2007). Top 10 Algorithms in Data Mining. *Knowledge and Information Systems*, vol. 14, n° 1, p. 1–37.