Rapport de projet

IHM et Machine Learning

Dans le cadre du
MASTER de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité
Infomatique

Parcours
ANDROIDE

Module
Projet ANDROIDE

Avril 2017

Villerabel Mathias	Étudiant
Bailly Gilles	Encadrant
Bredeche Nicolas	Enseignant
Maudet Nicolas	Enseignant

Table des matières

1 Introduction

Le but de cette application est de pouvoir modéliser les stratégies utilisées par des utilisateurs pour choisir dans un menu un item en particulier. Cette application repose sur les principes détaillés dans cette publication[1]. Que j'aurais l'occasion de citer de nombreuses fois en précisant les pages et chapitres concernés.

On peut facilement remarquer que nos menus sont souvent classés de tel sorte que les éléments aux champs lexicaux proches , ou aux fonctionnalités identiques sont proches les uns des autres. Bien que nous ne sachons pas vraiment où est situé l'élément que nous cherchons. Nous avons une idée de où il se trouve. C'est sur ce principe que repose cette application. Modéliser cette recherche que nous faisons naturellement, sauter plusieurs éléments si celui que l'on regarde n'est pas intéressant et au contraire regarder ceux proches quand ils semblent pertinent.

Nous allons aborder les points clefs qui permettent d'aborder les problématiques définies dans le cahier des charges

Menus et Items : Un menu est un ensemble d'Items. Un Item est par exemple celui de Enregistrer sous , zoom ou tout autre fonctionnalité proposé dans un menu. Ces items sont triés ou non dans un menu selon une pertinence ou relevance.

Pertinence : On entend par pertinence la façon dont un item est comparé à un autre item. Il y a plusieurs types de pertinences , la pertinence sémantique , la pertinence alphabétique ou encore la pertinence sur la taille d'un item.

Politiques/Stratégies : On parle de politique ou encore de stratégie quand on décrit les actions que prennent notre modèle en fonction de l'environnement où il est.

2 Architecture générale du code

L'application propose les cas d'utilisations suivants :

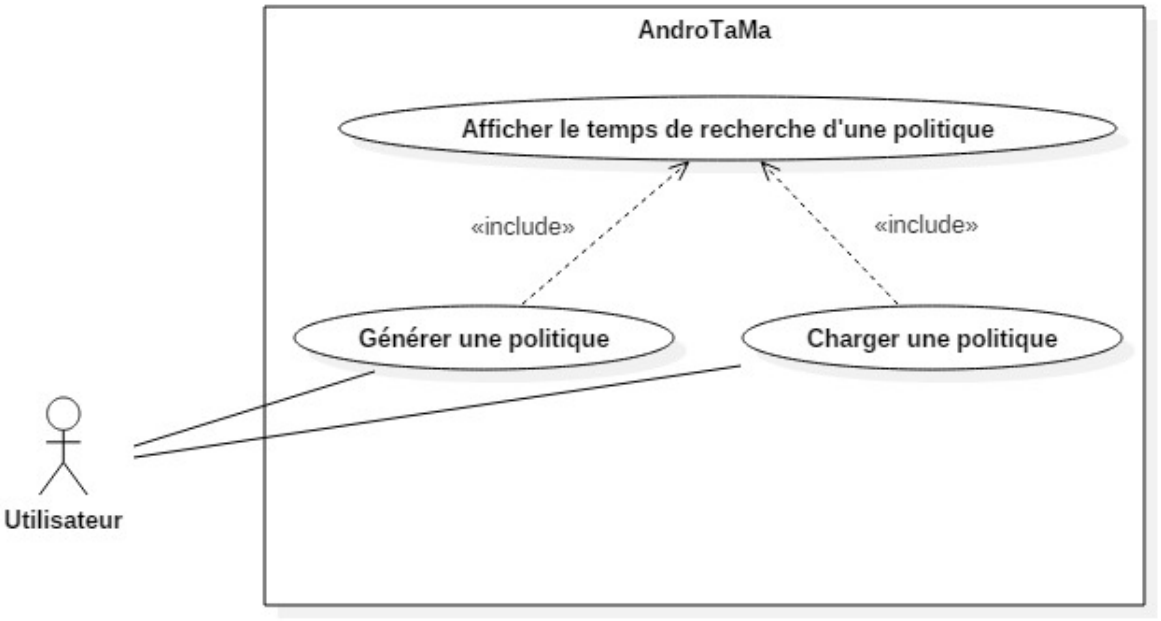

FIGURE 1 – Diagramme de package de src

L'utilisateur peut générer une politique à partir des bases de données ou en choisir une déjà générée. Une fois une des deux étapes choisies on affiche l'efficacité de cette politique dans différents types de Menu.

Les classes codées sont regroupées dans 3 packages.

2.1 Le package src

Il contient trois classes. main, projet et gestFile. Ainsi que 2 packages Ihm et model.

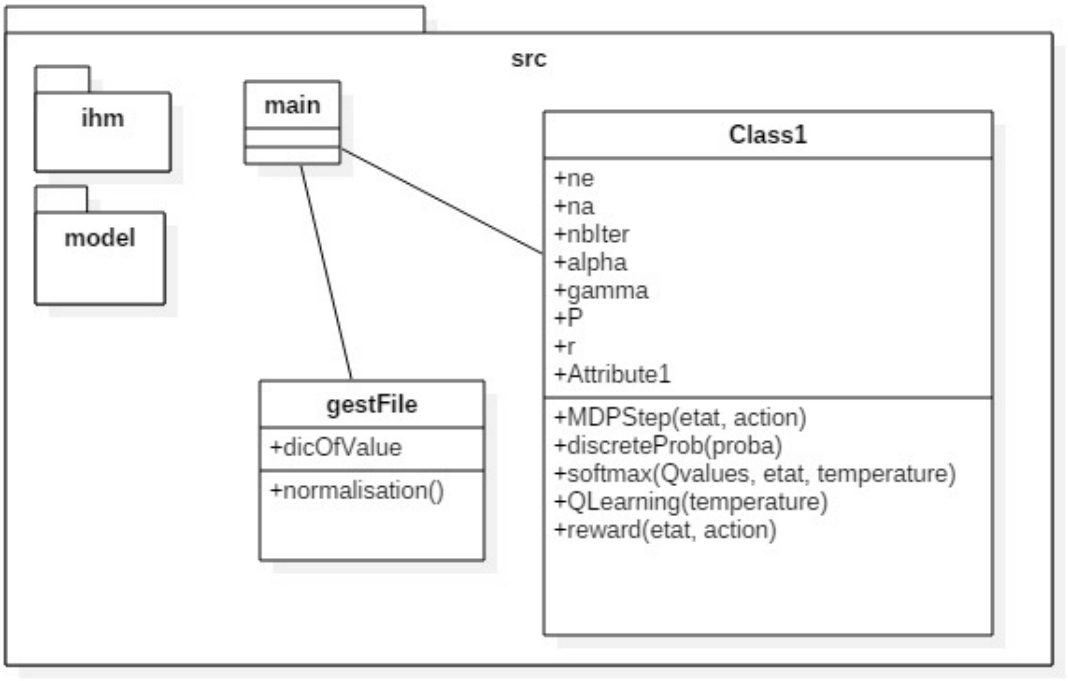

FIGURE 2 – Diagramme de package de src

main : Cette classe lance l'ensemble des éléments de l'application : be
- L'invité de commande de la Classe frame
- le gestionnaire de fichier de la classe gestFile
- l'algorithme de Q learning de la classe projet
- la comparaison du modèle de la classe plot

projet : Cette classe implémente l'algorithme de QLearning. Elle a pour attributs :

— ne : nombre d'états.
— na : nombre d'actions.
— gf : gestionnaire de fichier.
— nbIter : nombre d'itération.
— alpha : le coefficient d'apprentissage.
— gama : le coefficient de réduction.
— P : un dictionnaire contenant les probabilités de passer de l'état s à s' avec l'action a.
— r : contient les valeurs d'évaluations de la reward.

Ainsi que les opérations suivantes :

— MDPStep qui renvoie un état s' issu de l'état s et de l'action a.
— discreteProb qui permet de retourner un nombre en utilisant les probabilités de P.
— softmax qui renvoie une action en fonction des probabilités des actions de la QTable pour un état donné.
— QLearning implémente la boucle principale de l'algorithme de Qlearning.
— reward renvoie le score d'un état en fonction de son action

gestFile : Cette classe contient un dictionnaire contenant pour clefs le nom d'un type de pertinence et comme valeur des pertinences correspondantes aux items. L'opération normalisation permet de lisser ces valeurs de pertinences en 4 valeurs : 0.0 0.33 0.66 1.0

2.2 Le package src.model

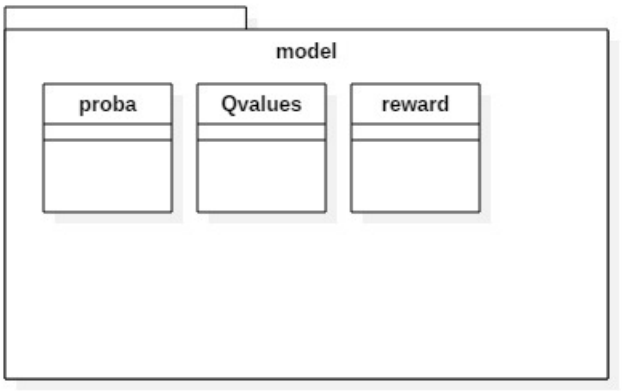

FIGURE 3 – Diagramme de package de src.model

Ce package contient des classes qui override les fonctions classiques des dictionnaires de python. Elles permettent de gérer les cas de Key error afin d'initialiser uniquement quand des clefs sont appelées. Les clefs des dictionnaires sont des tuples (état,action) pour reward et Qvalues et un triplet (etat1,action,etat2) pour Proba.

2.3 Le package src.Ihm

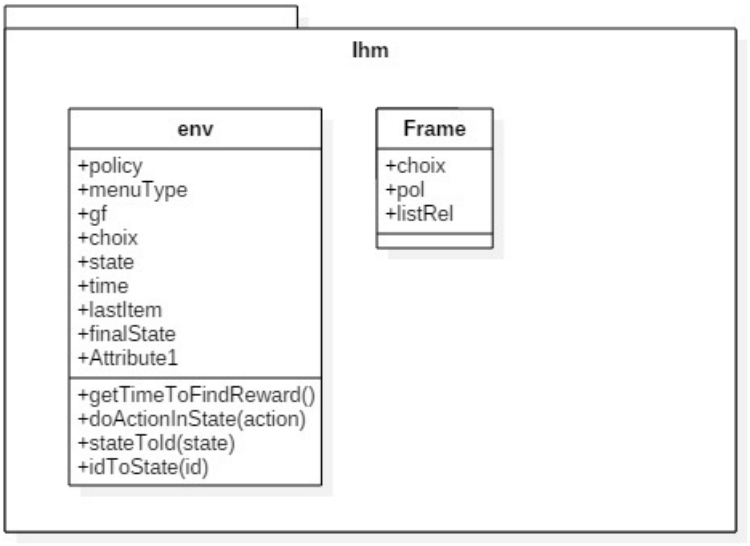

FIGURE 4 – Diagramme de package de src.Ihm

Frame : Permet de choisir de charger une politique ou en générer une via un invité de commande.

env : Cette classe permet de modéliser le comportement appris par une politique ainsi que le temps pris pour sélectionné un item but. Elle comprend :

- policy : la politique considérée.
- menuType : le type du menu sur le quel la politique a été apprise.
- state : l'état courant après avoir pris une action donnée par la politique.
- getTimeToFindReward() : renvoie le temps pris pour trouver l'item But en fonction de la politique.
- doActionInState(action) : met à jour l'état courant en fonction d'une action.
- stateToId(state) , idToState(id) : facilite l'accès aux états.

2.4 Les Dossiers BD et Polities

Dans BD/ sont stockés les bases de données en fichier .mat. Dans Polities sont stockées les politiques générées par la classe projet.

3 Gestion de la Pertinence

La pertinence ou relevance est la valeur donnée à la ressemblance entre un item du menu et l'item but

3.1 Bases de données :

Les pertinences sont stockées dans BD/ sous forme de fichier d'extension .mat et sont lues par la classe gestFile qui peut contenir plusieurs types de pertinences sous forme d'un dictionnaire. Un type de pertinence contient 10000 vecteur de taille 8. C'est pourquoi nous considérons nos menus comme ayant 8 items avec l'indice 0 de ce vecteur considéré comme l'item 1.

3.2 Valeur et normalisation

Valeur : Un vecteur contient des valeurs allant de 0.0 à 1.0. Plus la valeur est proche de 0.0 plus l'item considéré est peut pertinent. A l'inverse à une valeur de 1.0 l'item est celui que l'on recherche. Il y a 10000 vecteurs où les valeurs sont différentes et représentent des recherches d'item différents.

normalisation : les valeurs obtenues qui jugent de la pertinence d'items entre eux peuvent êtres infiniment différentes. Afin de réduire le nombre d'état à considérer (cf section 4.1) nous normalisons les valeurs possibles afin de les réduire à 0.0 0.33 0.66 et 1.0. [2]

4 Modélisation des états et des actions

Pour modéliser notre problème il nous faut définir un état qui permet de représenter notre environnement ainsi que les actions effectuables depuis cet état.

4.1 Définition d'un état :

Un état est définit sous cette forme : [val , val , val , val] pour 4 items. Où val peut prendre comme valeur Null si l'item n'a jamais été fixé. Ou être la valeur de pertinence stockée à l'indice item -1 si l'item à déjà été fixé.

Exemple Ainsi pour un menu à 4items on a un état initial :
- etat 0 [Null, Null, Null, Null] si on effectue l'action fixé l'item 1 on obtient comme nouvel état issu de cette action :
- etat 1 [0.0 , Null, Null, Null] avec 0.0 la valeur de pertinence entre l'item 1 et l'item target

4.2 Définition d'une action :

Ici on considère un ensemble de 10 actions allant de 0 à 9. L'action 0 consiste à fixer l'item 1. l'action 8 à sélectionner l'item courant et 9 à quitter le menu courant.

4.3 Processus de décision Markovien :

Pour modéliser l'état courant du système nous utilisons les processus de décision Markovien [5] [7]. L'avantage de cette méthode repose sur la représentation d'un état et des actions qui lui sont associés. Cela permet notamment de ne considérer que l'état courant qui est un problème à un moment donné plutôt que l'ensemble des solutions possibles.

5 Apprentissage par renforcement : QLearning

L'apprentissage par renforcement se base sur le fait qu'un état ou une action peuvent être évalués avec une fonction de score. Contrairement à l'apprentissage supervisé qui lui donne une réponse attendue. Le QLearning repose sur ce feed back sauf qu'il met un score non pas sur un état ou sur une action mais sur l'arc emmenant d'une action s à s' par l'action a. La politique optimale est alors pour une état s de prendre l'action avec le meilleur score. Cet algorithme converge il permet donc d'obtenir une politique optimale.[4]

5.1 Définition des paramètres de l'algorithme

L'algorithme prend en compte 3 paramètres déjà abordés en partie 2.1 :

— le nombre d'itération de l'algorithme , plus il est élevé plus nous parcourons différents états ou plus souvent des états donc notre politique est plus fiable.

— alpha est le taux d'apprentissage si il est trop élevé on risque de manquer des états qui ne seront jamais explorés

— gamma , plus il est proche de 0 plus on s'intéresse a des solutions immédiates. Proche de 1 les scores futurs sont tout aussi important que les scores immédiats

5.2 Le choix d'un état

On commence tout d'abord par choisir un vecteur de pertinence de façon aléatoire Puis tant que qu'on ne sélectionne pas une action qui est de sortir ou de sélection un item on tire aléatoire un état dans l'ensemble des états

Ici self.choix contient l'indice du vecteur dans la liste des vecteurs de pertinence. x est l'état choisi.

```
for i in range(self.nbIter):
    u=0
    self.choix = np.random.randint(len(self.gf.
    dicOfValue[list(self.gf.dicOfValue.keys())[0]])-1)

    while(u<8):
        print('iteration ',i)
        x = np.floor(self.ne*np.random.random())
```

5.3 Le choix de l'action

Quand on choisi une action il faut choisir entre explorer ou exploiter. c'est à dire essayer une action sur la quelle on n'as pas de retour ou prendre au contraire une action dont on connaît le score. Quitte a prendre le risque que l'action qu'on ne connaît pas soit meilleure que celle connue. Ici on utilise la politique du soft-max c'est à dire qu'on va faire gonfler les probabilités des actions avec un meilleur score sans pour autant ignorer les autres. Tau ici compris entre 0 et 1.0 plus il est proche de 0 plus on se rapproche de l'approche où on priorise le choix d'exploiter.

```
u = self.discreteProb(self.softmax(Q,x,tau))
```

5.4 le choix de l'état suivant

On appel la fonction MDPStep(x,u) ici on choisi un état parmis les états probables du à l'action u. les valeurs sont aléatoire mais sont égales à 1.0

```
[y,r] = self.MDPStep(x,u)
```

```
def MDPStep(self,x,u):
        #choix d'un etat par rapport a une
           probabilit  sur une action
        temp = []
        for i in np.arange(0.0,self.ne,1.0):
            temp.append(self.P[x,u,i])
        tab = np.array(temp)
        y = self.discreteProb(tab.reshape(self.ne,1))
        r = self.reward(x,u) + 0.1*np.random.randn()
```

5.5 Évaluation de l'action

Appelé dans MDPStep notre fonction d'évaluation se presente de cette facon : time = 437 + 2.7 * A [3] Score = 10000 × correct 10000 ×error time, corect si on sélectionne l'item et error pour les autres actions

5.6 Politique Optimale

notre politique optimale correspond pour chaques états à l'action avec la valeur de Qvalue la plus élevé à la fin de l'itération.

```
pol =  np.argmax(Q,axis=1)
```

6 Comparaison du modèle

Une fois une politique générée ou chargée nous allons simuler un environnement afin de tester la politique optimale et de la comparer dans différents types de menus l'opération getTimeToFindReward de la classe env simule un environnement où on choisit à chaque itérations une action à prendre

A chaque actions le coût en temps augmente. si une action nous mène sur l'item but alors on est dans un état finit et on retire au sort un vecteur dans la liste des peritnances. on Réitère 10 000 fois. Et on sauvegarde le temps mis par cette politique sur chaque type de menu.

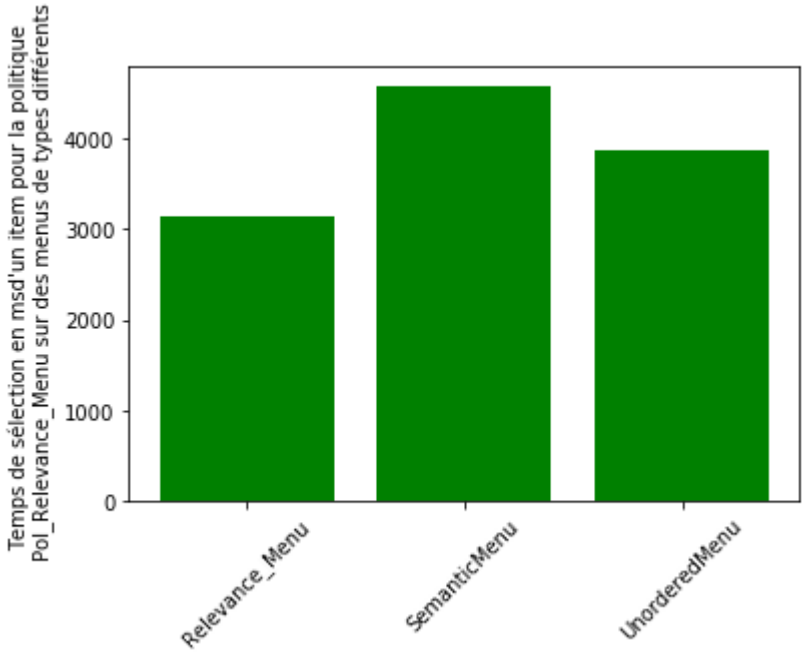

FIGURE 5 – Histogramme de l'application de la politique optimale apprise sur un menu de type Shape Relevance sur d'autres types de menu

Ici on voit bien que la politique apprise sur Relevance Menu prend moins de temps à sélectionner le bon item comparé à d'autres types de menu.

De même la politique apprise est plus efficace sur le type de menu. Confirmant bien que les politiques apprises sur un type de menu permettent de modéliser un comportement plus rapide et à la fois plus pertinent.

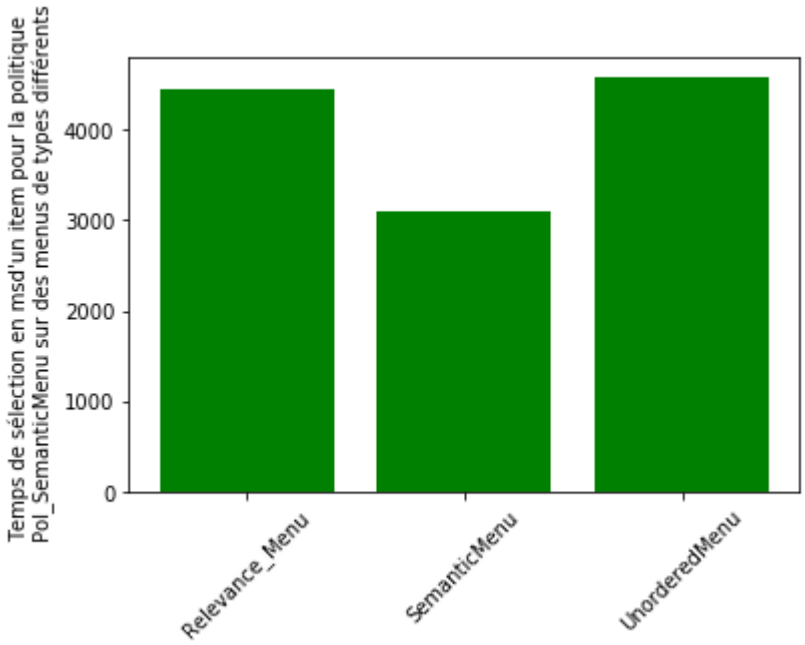

FIGURE 6 – Histogramme de l'application de la politique optimale apprise sur un menu de type Sémantique sur d'autres types de menu

7 Conclusion

7.1 Synthèse :

Notre application est capable de générer des politiques optimales. Elle est aussi capable de charger des politiques déjà générées , pratique pour des très longs apprentissages. Elle offre un affichage clair sous forme d'un histogramme permettant de montré leurs efficacités dans les menu du même type de pertinence qu'elles. Nous répondons donc à nos problématiques soulevées lors du cahier des charges : Il y ici deux problématiques, l'une est de modéliser efficacement le comportement d'un utilisateur face à un menu , et l'autre de comparer ce comportement face à un autre menu.

7.2 Améliorations :

L'approche actuelle des états est très simpliste, étant parti d'une gestion de donnée avec des arrays cela ne convient pas du tout à des grosses instances. De plus je devrais considérer 5 puissance 8 états. Je n'en considère que 2 à la puissance 8. c'est à dire la présence ou non d'item but. J'ai du faire ce choix afin d'avoir au moins une base d'algorithme et de politique à tester. J'ai commencé à implémenter des classes de dictionnaires plus adaptés à ce genre de manipulation un peux comme la classe Proba. Ainsi que des lectures en fichier si la tables des Q values se trouvait trop grande pour la Ram. Cependant la meilleur solution serai d'implémenter les méthodes que propose dans son livre Olivier Sigaud sur les factorisation des MDP [6]

Une autre approche aurait été aussi d'implémenter les MDRL , Model Bases Reinforcement Learning.

La majeurs partie des classes de lectures et d'écriture de politiques prennent en compte le fait que plusieurs vecteur de pertinence peuvent former un état. En effet la shape Relevance d'un objet et la relevance sémantique sont tout à fait compatible dans la sélection d'un item but et même complémentaire.

7.3 Le mot de la fin :

je voudrais tout d'abord remercier mon encadrant gilles Bailly pour m'avoir accompagné lors de ce projet. J'ai depuis ma deuxième année de Licence été toujours attiré par les méthodes d'apprentissages. J'ai tout de suite pris à coeur ce projet. Un peux trop sûrement car j'ai perdu beaucoup de temps à réfléchir à plein de méthodes d'implémentation, quitte à louper quelques points importants dans la tournure des algorithmes. Néanmoins ce fut une expérience enrichissante, notamment sur les documents à fournir dont nous n'avons pas eu l'habitude à faire jusqu'à présent (cahier des charges, bibliographie en norme ieee). Mon seul regret restera cependant d'avoir compris trop tard que mon binôme ne serait pas là pour faire le projet avec moi.

Références

[1] Xiuli Chen, Gilles Bailly, Duncan P. Brumby, Antti Oulasvirta, and Andrew Howes. The emergence of interactive behavior : A model of rational menu search. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 4217–4226, New York, NY, USA, 2015. ACM.

[2] Xiuli Chen, Gilles Bailly, Duncan P. Brumby, Antti Oulasvirta, and Andrew Howes. *The Emergence of Interactive Behaviour : A Model of Rational Menu Search*, chapter Alphabetics and Shape relevance, page 3. april 2015. http ://www.gillesbailly.fr/publis/BAILLY$_IAModel.pdf$.

[3] Xiuli Chen, Gilles Bailly, Duncan P. Brumby, Antti Oulasvirta, and Andrew Howes. *The Emergence of Interactive Behaviour : A Model of Rational Menu Search*, chapter Saccade duration, Fixation duration, page 4. april 2015. http ://www.gillesbailly.fr/publis/BAILLY$_IAModel.pdf$.

[4] Jing Peng and Ronald J Williams. Efficient learning and planning within the dyna framework. *Adaptive Behavior*, 1(4) :437–454, 1993.

[5] Olivier Sigaud. Reinforcement learning : the basics, September 2015. http ://-pages.isir.upmc.fr/ sigaud/teach/MSR/basic$_rl.pdf$.

[6] Olivier Sigaud and Olivier Buffet. *Processus décisionnels de Markov en intelligence artificielle volume 2*, chapter 2.2.1 Représentatiion de l'espace d'état, pages 52 – 54. LAVOISIER, 2008.

[7] Olivier Sigaud and Olivier Buffet. *Markov Decision Processes in Artificial Intelligence*. Wiley-IEEE Press, 2010.