BENABOU Nathan
VUILLEMIN Pierre

ENSAE deuxième année
mathématiques appliquées

Projet de C++

Le bandit multibras

le 21 février 2014

Encadré par Matthieu DURUT

Sommaire

Introduction

Le bandit-multibras est un problème d'apprentissage qui peut être abordé de façon simple. C'est pour cela que nous l'avons choisi. L'idée de pouvoir optimiser la prise de décisions via des lagorithmes nous a séduite. Le fait de développer une interface graphique en plus des algorithmes de regrets que nous avons pu trouver, nous ouvrait les perspectives de travailler sur des problématiques d'informatique "pure" au sens ou cela ne se base pas sur un problème mathématique ou financier d'une part. Et de travailler avec une librarie externe d'autre part, ce qui semble être une pratique courante. Ainsi au travers de ce projet, nous avons développé plusieurs algorithmes de regrets ainsi qu'une interface graphique. Ce sont ces deux parties que nous présentons dans ce rapport.

1 La Structure du Bandit

Le bandit-multibras est une sorte de machine à sous à plusieurs bras. Chaque bras génère, lorsqu'il est actionné, une récompense qui n'est autre qu'une réalisation d'une variable aléatoire selon une distribution qui est spécifique au bras. Le bandit est ainsi représenté par une classe "Bandit" et un bras est également représenté par une classe "Arm".

1.1 La classe Arm

La classe Arm permet de créer un bras, la principale difficulté de cette classe est qu'elle comprend une méthode Generate qui permet de générer une récompense selon la distribution du bras et qui dépend ainsi du type du bras. J'ai choisi de créer plusieurs classes filles à la classe Arm, chaque classe fille correspondant à un type de distribution (Exponentielle, Uniforme, Log Normale, ..). La méthode Generate de la classe mère Arm est par conséquent une méthode virtuelle et la classe Arm une classe abstraite : On ne peut pas créer d'objet de type Arm.

1.2 La classe Bandit

La classe bandit permet de créer cette fameuse machine à sous à plusieurs bras. Parmi ses attributs se trouvent par conséquents les bras, mais ce n'est pas si simple car comme vu au paragraphe précédent on ne peut pas créer d'objet de type Arm, par conséquent on trouvera dans les attributs du bandit des pointeurs vers des bras et non des bras. Cette nuance a occasionné certaines difficultés dans la rédaction du constructeur du bandit. En effet le constructeur prend en argument le nombre de bras, le type de bras souhaités ainsi que leurs paramètres pour ensuite créer les bras souhaités et faire pointer le bandit sur ces derniers. Dans un premier temps, si par exemple il s'agissait de créer un bras exponentiel, j'avais choisi de créer directement le bras dans le constructeur pour ensuite faire pointer le bandit sur ce bras. Néanmoins j'ai pu me rendre compte que lorsque je compilais le programme, le bras était certes créé, l'adresse du bras bien spécifiée dans le pointeur mais une fois la méthode de construction du bandit exécutée, le bras était supprimé et le bandit ne pointait plus que sur une case vide. J'ai ainsi opté pour une allocation dynamique de la mémoire en utilisant la fonction "new" pour allouer moi même la mémoire de l'ordinateur et ne pas supprimer les bras créés.

2 Le joueur

Le joueur est la personne qui joue sur la machine à sous. Il est représenté par la classe "Player"et il est ainsi naturel de faire pointer cette classe sur un bandit. Il peut décider de jouer naïvement en actionnant lui même les bras qu'il veut ou suivant une certaine stratégie. Les stratégies sont basées sur les moyennes empiriques des tirages sur chaque bras. Il est donc nécessaire de toujours initialiser le bandit en tirant plusieurs fois (5 par exemple) sur chacun des bras avant de lancer la stratégie.

L'indicateur le plus souvent utiliser pour évaluer la qualité de la stratégie d'un joueur est le regret cumulé. En notant à l'instant T R_T le regret cumulé, $j(t)$ le numéro du bras choisi, et μ^* la moyenne du meilleur bras, le regret cumulé s'écrit :

$$R_T = T\mu^* - \sum_{t=1}^{T} \mu_{j(t)}$$

Considérons un joueur jouant sur un bandit de n bras. Après avoir initialisé la partie en tirant quelques coup sur chaque bras, on a, pour chaque bras i la moyenne empirique $\hat{\mu}_i(t)$ au coup numéro t et $n_i(t)$ le nombre de fois que le bras a été tiré au coup numéro t.

2.1 La stratégie UCB

Après le coup $t-1$ on choisit le bras $i(t)$ à actionner au coup suivant de la manière suivante :

$$i(t) = \arg\max_{j=1..n} (\hat{\mu}_j(t) + \sqrt{\frac{2ln(t)}{n_j}})$$

Dans l'implémentation de la méthode, on crée le vecteur des $\hat{\mu}_j + \sqrt{\frac{2ln(t)}{n_j}}$ pour $j = 1..n$ et on cherche l'indice du maximum pour ensuite activer le bras correspondant.

2.2 La stratégie ϵ-greedy

Au premier coup on crée le vecteur P tel que $P_1[i] = \frac{1}{n}$ est la probabilité de tirer le bras i au coup 1. A chaque coup on réalise l'opération suivante, en spécifiant le paramètre ϵ souhaité :

$$P_{t+1}[i] = \begin{cases} 1\text{-}\epsilon + \frac{\epsilon}{n} & \text{si } i = \arg\max_{j=1..n} (\hat{\mu}_j(t)) \\ \frac{\epsilon}{n} & \text{sinon} \end{cases}$$

Etant donné qu'il est complexe d'implémenter sa propre distribution et de tirer aléatoirement un nombre suivant cette dernière, on contourne cette difficulté en découpant l'intervalle $[0, 1]$ en segments de tailles égales aux probabilités du vecteur P, puis en tirant un nombre suivant une loi uniforme sur cet intervalle, et enfin de choisir le bras à tirer suivant la position du nombre généré par rapport aux composantes du vecteur P.

2.3 La stratégie Boltzmann Exploration

On crée le vecteur P_t tel que $P_t[i] = \frac{1}{n}$ est la probabilité de tirer le bras i au coup t de la manière suivante, en spécifiant le paramètre τ souhaité :

$$P_{t+1}[i] = \frac{e^{\frac{\hat{\mu}_i(t)}{\tau}}}{\sum_{j=1}^{n} \hat{\mu}_j(t)}$$

2.4 La stratégie Pursuit Algorithms

Au premier coup on crée le vecteur P tel que $P_1[i] = \frac{1}{n}$ est la probabilité de tirer le bras i au coup 1. A chaque coup on réalise l'opération suivante, en spécifiant le paramètre β souhaité :

$$P_{t+1}[i] = \begin{cases} P_t[i] + \beta(1\text{-}\ P_t[i]) & \text{si } i = \arg\max_{j=1..n} (\hat{\mu}_j(t)) \\ P_t[i] + \beta(0\text{-}\ P_t[i]) & \text{sinon} \end{cases}$$

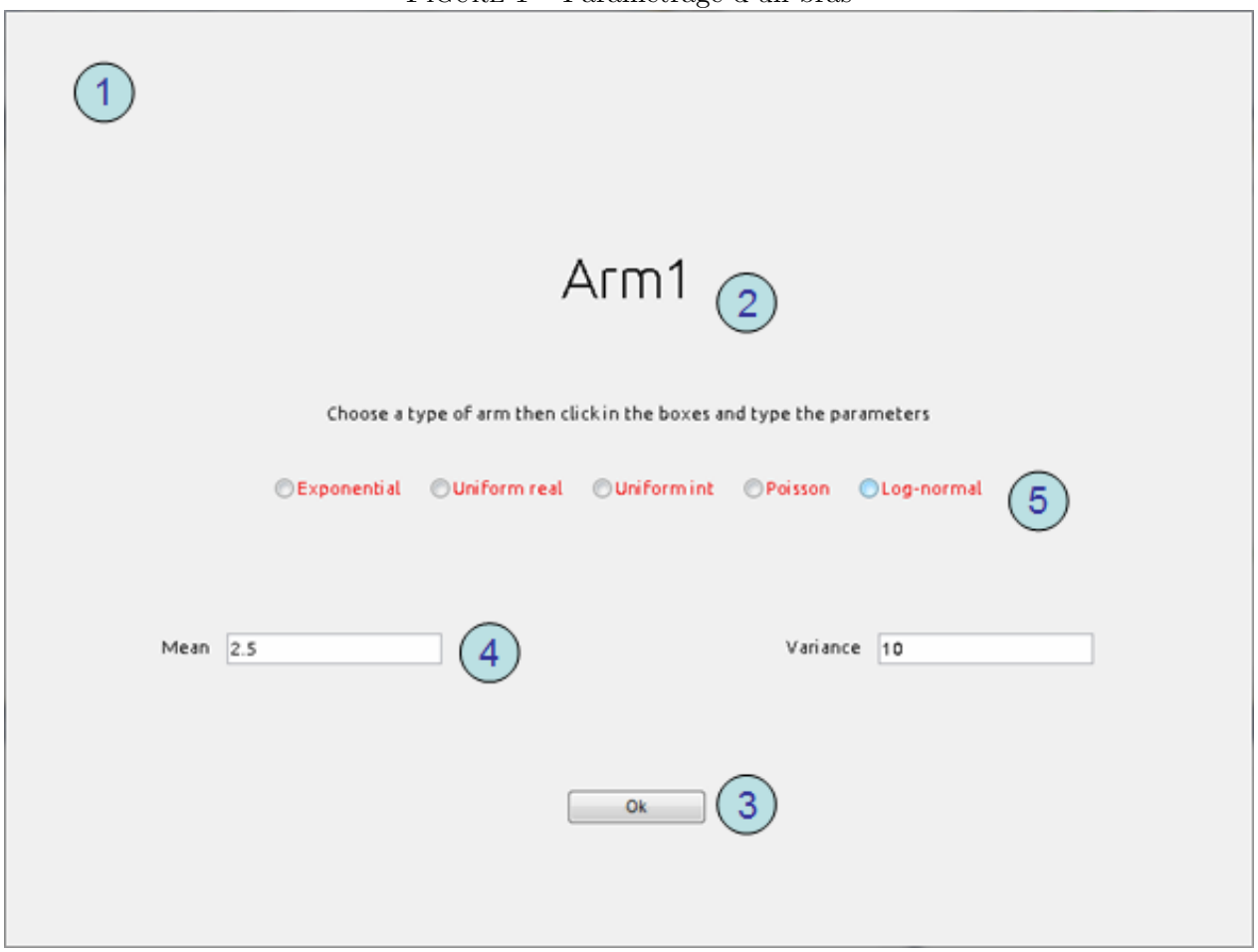
3 L'interface graphique

L'interface graphique a été réalisée en utilisant la bibliothèque SDL ainsi que ses extensions SDL_image et SDL_ttf. Cette bibliothèque étant issue du C, elle n'est pas constituée de classes par défaut. Nous avons donc créée nos propres classes. nous les présentons ici de façon succinte. On trouvera le détail des attributs et méthodes en commentaire dans le code.

3.1 L'usage de la bibliothèque SDL dans les classes

Dans un premier temps, des fonctions utiles ont été codés à partir des fonctions de la SDL et regroupées dans un fichier "functions.cpp". Ceci a été fait conformément à l'excellent tutoriel du site lazy foo présenté en TD. Elles permettent de regrouper des fonctions complémentaires de la SDL qui sont ainsi plus robustes.

FIGURE 1 – Paramétrage d'un bras



De manière générale, la SDL propose l'outils SDL_Rect qui permet d'avoir accès facilement à la position et aux dimensions d'un objet (x, y pour la position, h,w pour la hauteur et la largeur). Dans la suite, cet outil est systématiquement utilisé et est nommé _box dans les attributs des classes. De nombreux "getters" et "setters" s'y rapportent. De même, les destructeurs ont été modifiés de sorte à utiliser la fonction SDL_FreeSurface qui est conçu pour effacer convenablement les objets affichés à l'écran. Ceci permet peut être d'éviter des "memory leaks". En effet on ne sait pas vraiment ce qu'il se passe lors de la création d'une SDL_Surface qui est gérée par la bibliothèque SDL. Cette pratique a été renouvelée dans toutes les classes suivantes à chaque fois que celles-ci utilisaient une SDL_Surface. Il en va de même pour les fonts issues de l'extension SDL_TTF.

3.2 Screens

Screens est la classe des fenêtres. La fenêtre (cf. 1 sur la figure 1), est l'objet sur lequel tous les autres objets de l'interface sont affichés. Cette classe possède des méthodes simples permettant de rafraîchir et d'afficher la fenêtre ainsi que de la nettoyer avant d'en changer et de passer à l'étape suivante.

3.3 Texts

Texts est la classe des textes que l'on affiche à l'écran (cf 2 sur le figure 1). Là encore, c'est une classe sur laquelle aucune intéraction n'est possible. Ainsi les seules méthodes créées correspondent

à l'affichage de ces textes, et au repositionnement de ceux-ci. Ces dernières sont utiles lors de la création des fenêtres de l'interface.

3.4 Buttons

Buttons est la classe des boutons classiques tel que 3 sur la figure 1. Il s'agit de la classe mère de tous les autres boutons. Un bouton est caractérisé par sa position et trois images : lorsque l'utilisateur pointe un pixel en dehors du bouton (_outImage), sur le bouton (_inImage) et une lorsque l'utilisateur clique sur le bouton (_pushImage). Il s'agit ainsi de la première classe qui intéragit avec l'utilisateur. Cette intéraction (HandleEvents dans le code), se matérialise par un changement de couleur selon la position du curseur et par le changement de la valeur d'un booléen après un clique. Ces boutons sont utilisés au sein de boucle while qui maintiennent la fenêtre ouverte jusqu'à ce que la valeur du booléen change (après un clique ou l'appui sur la touche entrée). Les autres méthodes sont des "setters" et "getters" classiques ainsi qu'une méthode d'affichage.

3.5 TypeZone

Les TypeZones (4 sur la figure 1) sont les zones dans lesquelles il est possible de taper des nombres (double ou int selon les situations). L'intéraction est de deux types : il faut d'abord cliquer dessus pour l'activer puis taper les chiffres successifs. Le point (clavier azerty) est utilisé pour les nombres décimaux. Outre la méthode d'intéraction, des méthodes d'affichages ont été implémentées. Une TypeZone est toujours accompagnée d'un Texts (en attribut).

3.6 RadioButtons

Les radioButtons (5 sur la figure 1), héritent de la classe Buttons. Ils permettent d'effectuer un choix unique parmi différentes propositions. Ainsi cliquer sur un bouton, en désactive un autre. Un RadioButtons a six positions : selon s'il est actif ou non, les trois de la classe buttons. Ainsi dans leur conception, deux Buttons ont été utilisés. Comme pour les TypeZone, un Texts est toujours associé (en attribut).

FIGURE 2 – Up&Down Button

3.7 Up&Down Button

Les UpButton et DownButton (cf figure 2), sont deux classes héritant de Buttons qui combinées permettent de faire varier un entier en cliquant sur les flèches. Le UpButton correspond à la flèche supérieure ainsi qu'au Texts associé et à la boite blanche. Le DownButton ne correspond qu'à la flèche inférieure. L'intéraction avec l'utilisateur fait intervenir le nombre affiché qui est passé par référence. En tant que Buttons, chacune des flèches possède trois position (out,in et push).

3.8 Gui

La classe Gui est la finalité de toutes les classes précédentes. Ses méthodes correspondent aux différentes fenêtres que l'utilisateur rencontre et sont appelées dans la main. La fenêtre de bienvenue, les fenêtre permttant de choisir le nombre de bras et de les paramètrer (figure 1 et 2), etc... Les captures d'écrans de toutes les fenêtres sont présentes en annexe. Les paramètres obtenus grace à l'interface graphique sont ensuite réutilisés pour paramétrer les instances des classes permettant

de simuler un bandit multibras. Il est à noter que l'interface graphique est quasi indépendante des classes formant le bandit, ce qui assure une bonne portabilité, et permet de décorréler le travail des mambres du groupe. De plus les classes présentées ci-dessus sont réutilisable en l'état pour un éventuel autre projet.

Conclusion

En conclusion, le problème du bandit multibras est extrêmement instructif tant d'un point de vue informatique que mathématique. La création des différentes classes nécessaires à la création du bandit et du joueur permettent de parcourir les difficultés classiques que l'on peut rencontrer en C++. D'autre part la création d'une interface graphique aussi simple soit-elle donne un aperçu intéressant de ce à quoi l'on peut aboutir avec C++. De plus, ce projet a été l'occasion de mettre en pratique certaines de nos connaissances en statistiques et en probabilités.