Compte rendu du projet informatique S2

Bot B.E.A.R.L.

Joueur de Poker

Baptiste DELORME, Etienne NAVARRE, Arthur FINDELAIR, Raphaël LEMAS et Louis BOSSU
Groupe 7b

1er Juillet 2019

Sommaire

Première partie

Bot amélioré

1 Introduction

Ce projet est la suite logique du travail effectué lors du premier semestre. Nous allons utiliser le code sources d'un des groupes composant notre nouvelle équipe afin de le faire évoluer vers un projet le plus robuste possible. De la sorte il sera alors possible de travailler en parallèle sereinement sur différentes nouvelles fonctionnalités du Bot, humblement nommé BEARL : *Bot Extremely Aggressive Ready to Lead* (ou les initiales de nos prénoms). La première étape est donc tout naturellement de définir le code de base et d'établir les évolutions à effectuer. Par la suite nous modifierons et créerons les packages nécessaires à l'implémentation de ces nouvelles fonctionnalités pour finalement réaliser le Bot.

2 Etat des lieux et Amélioration du bot initial

2.1 Choix du bot source

Notre équipe avait à disposition trois BOT, celui de Raphael et Baptiste, d'Etienne et Tanguy (qui s'est envolé vers de nouveaux horizons ...) et de Louis et Arthur. Le premier critère était l'avancement des Bot respectifs. Les deux premiers étaient globalement plus avancés avec les mêmes fonctionnalités tels qu'une évaluation efficace des mains, une gestion aboutie des messages, une stratégie efficace ... On notera que la qualité d'écriture était équivalente. Le choix entre ces deux dernier Bot s'est fait par un duel à mort, le gagnant sur 100 parties sera sélectionné. Le BOT retenu est donc celui d'Arthur et Louis, qui héritent alors de la tâche d'encapsulation initiale.

2.2 Description du Bot initial :

Le compte rendu du S1 du Bot correspondant est donné en annexe pour des informations complémentaires. Le bot est décomposé en trois principales parties : la modélisation du jeu, la lecture et mise a jours de informations de jeu, la capacité de faire des actions cohérentes.

Modélisation du jeu :

La structure de donnée repose sur 2 types articles principaux : T_Jeu et T_Joueur. Ils permettent le stockage respectif des informations de la partie en cours tel que le pot et les blinds, et le stockage des informations relatives aux joueurs tel que ses cartes et son stack (cf. 2.2 Modélisation d'une partie, de la table, des joueurs).

Lectures des messages :

La lecture est assez basique, après avoir décomposé le message en mots clefs, les informations sont stockées dans la variable adéquate d'après un organigramme défini en 3.1 Représentation des messages. La lecture des messages n'est pas exhaustive mais la modélisation des messages peut être facilement complété lors de l'encapsulation.

Stratégie :

Le Bot présente la capacité d'identifier la combinaison de carte d'un joueur et de l'évaluer. Cette évaluation est effectuée par la génération exhaustive des potentielles cartes de l'adversaire afin de comparer les combinaisons et de déterminer un potentiel gagnant. Sur cette génération, on peut établir la puissance de la main (cf. 4.3 Stratégie). Le choix de l'action du Bot repose sur ce pourcentage de gagne d'après le graphique Figure 6 – Schéma des règles pour la stratégie.

2.3 Évolutions envisagées

Nous souhaitons améliorer l'évaluation de la force de la main d'un joueur par une méthode aléatoire et itérative puis évaluer le caractère de l'adversaire afin d'adapter notre stratégie. Pour cela, nous allons effectuer d'importantes modifications. Afin de simplifier le travail sur chaque packages, ll va être nécessaire d'encapsuler notre projet. Ainsi chaque groupe de travail n'a pas à se soucier du travail des autres étant donné que les packages nouvellement créés seront indépendants. Chacun de ces nouveaux packages va néanmoins reposer sur la structure de données de stockage de la partie et des joueurs présentent dans le code source. La première étape du projet est donc de simplifier l'utilisation grâce à des Getter et Setter. L'objectif est de limiter au maximum le travail nécessaire à la réalisation et au debogage du BOT à proprement dis, lors de la mise en commun de toutes les parties.

Graphe dépendance des packages

2.4 Encapsulation et debogage du bot source (Arthur et Louis)

Le bot initial présentait les 3 packages décris en *2.2 Description du Bot initial*. On en récupère le code source pour réaliser les packages encapsulés suivant :

- Modelisation_poker :

  On definie ici les types modélisant les informations recues par le Launcher et la structure de données du Bot. Les types mis à disposition sont les suivants :

```
Type T_Couleur is (pique,carreau,coeur,trefle);
Type T_Motif is (deux,trois,quatre,cinq,six,sept,huit,neuf,dix,valet,reine,roi,as);
Type T_moves is (fold, check, call, bet);
Type T_Carte is private;
Type T_liste_cartes is private;
Type T_Jeu is private;
Type T_joueur is private;
```

Listing 1 – Types de *Modelisation_poker*

  Les types *T_Motif*, *T_Couleur* et *T_moves* sont publics afin de pouvoir etre utilisé dans la création et le stockage des cartes lues par le Launcher et les mouvements de chaques joueurs. Les autres types etant plus complexes et ouvert aux modifications ils sont encapsulés. Dans la suite du projet nous ajouterons des articles dans ces types afin d'ameliorer la structure de données. Des accesseurs de la forme suivantes sont mis à disposition pour les types *T_Jeu* et *T_joueur*, tout deux des types articles.

```
Procedure Set_Article(Var_stockage : OUT T_articles;
                      valeurs : IN T_adequat);
Function Get_Article(Var_stockage : IN T_articles) return T_adequat;

-- Par exemple :
Procedure Set_joueur_main(joueur : OUT T_joueur ;
                          main : IN T_liste_cartes);
Function Get_joueur_main(joueur : IN T_joueur) return T_liste_cartes;
```

Listing 2 – Accesseurs des types *T_Jeu* et *T_joueur*

La manipulations des cartes se fait par le biais des accesseurs ci-dessous. Le nombre de carte dans une liste est mis à jours par les Setters. De plus, nous implémentons une fonction vérifiant l'unicité des cartes d'une liste et nous surchargeons les operateurs de comparaison pour faciliter l'étude des mains

```
-- Unicite_cartes
-- E/ Liste de cartes : Liste_carte
-- Necessite : Null
-- S/ Boolean
-- Entraine : True si chaque carte est unique dans la liste, False sinon
Function Unicite_cartes(Liste_carte : IN T_liste_cartes) return Boolean;

Procedure Set_carte(carte : OUT T_Carte; motif : IN T_Motif; couleur : IN T_Couleur);
Procedure Add_liste_carte(carte : IN T_Carte; liste_carte : IN OUT T_liste_cartes);
Procedure Set_liste_carte(carte : IN T_Carte; liste_carte : IN OUT T_liste_cartes; Index : IN
    Natural);

Function Init_liste_carte return T_liste_cartes; -- Permet de reinitialiser la table de jeu
Function Get_nbr_liste_carte(liste_carte : IN T_liste_cartes) return Natural;
Function Get_liste_carte(liste_carte : IN T_liste_cartes; index : IN Natural) return T_Carte;
Function Get_carte_motif(Carte : IN T_Carte) return T_Motif;
Function Get_carte_couleur(Carte : IN T_Carte) return T_Couleur;

Function ">" (Carte_1 : IN T_Carte ; Carte_2 : IN T_Carte) return Boolean; --Uniquement base
    sur le motif (neglige la couleur)
Function "<" (Carte_1 : IN T_Carte ; Carte_2 : IN T_Carte) return Boolean;
Function "=" (Carte_1 : IN T_Carte ; Carte_2 : IN T_Carte) return Boolean;
```

Listing 3 – Accesseurs des types *T_carte* et *T_liste_cartes*

Finalement, quelques fonctions d'affichage en sortie d'erreur standard (afin d'etre utilisé dans le Bot) sont présentes pour faciliter la rédactions de fichiers tests.

```
procedure Affiche_carte(Carte : IN T_Carte);
procedure Affiche_liste_carte(liste_carte : IN T_liste_cartes);
procedure Affiche_jeu(jeu : IN T_Jeu);
procedure Affiche_joueur(joueur : IN T_joueur);
```

Listing 4 – Procedures d'affichages

- Lecture_messages :

A ce niveau du projet, très peu de modifications sont apportée au package homonyme du Bot source mis à part l'encapsulation. Le type *T_message* stockant les messages est assez complexe et avait mené à quelques confusions d'utilisation lors du premier semestre *(cf. 8.1 Représentation des messages)*. A present, il n'y a que deux procedure relative au traitement des messages :

```
Type T_chaine is private;
-- stock le message sous forme de chaine et comme liste de mots clefs
Type T_message is private;

-- Get
-- E/ Lecture console d'un message du Launcher
-- S/ Message : T_message
-- Entraine : Message lue et decompose, pret pour Update_data
-- Verification : test_maj_bot.adb
procedure Get(Message : OUT T_message);

-- Update_data
-- E/ Message : T_message
-- Necessite : Le message est conforme au Launcher et decompose
```

```
-- S/ Table : T_liste_cartes ; Infos de la partie : T_jeu ; 2 joueures : T_joueur
-- Entraine : Mise a jour de l'etat de la partie en fonction du message
-- Verification : test_maj_bot.adb
procedure Update_data(message : IN T_message ; Table : OUT T_liste_cartes ; info_partie : Out
    T_jeu ; J_Self : Out T_joueur ; J_Other : Out T_joueur);
```

Listing 5 – Manipulation des messages

Les accesseurs sont definies pour l'utilisation de *T_message*. Ces fonctions sont cependant inutile au Bot, elles ne seront utilisé que lors de certains tests. On notera que les accesseurs prennent en compte la décomposition du message afin de ne rencontrer aucun problème de lecture.

```
-- Manipulation de T_chaine

Procedure Set_chaine(Chaine : OUT T_chaine ; line : IN String ; line_length : IN Integer);
Procedure get_line(Chaine : OUT T_chaine);

Function Get_chaine_line(Chaine : IN T_chaine) return String;
Function Get_chaine_len(Chaine : IN T_chaine) return Natural;

-- Manipulation de T_message

procedure Set_message(Message : OUT T_message; Chaine : IN T_chaine);

Function Get_message_entier(Message : IN T_message) return T_chaine;
Function Get_message_mot(Message : IN T_message ; Index : IN Natural) Return T_chaine;
Function Get_message_nbr_mot(Message : IN T_message) return Natural;
```

Listing 6 – Accesseurs de *T_chaine* et *T_message*

Lorsque la structure de donnée du Bot évolue en ajoutant des articles, la lectures de messages doit permettre le stockage des ces données à la lecture du Launcher. La structure du programme du Bot initial permet facilement d'implémenter ces ajouts. Il suffit d'ajouter les cas adéquat dans la structure à condition multiple *(cf. 8.2 Lecture et interprétation des messages)*.

Au cours du projet, nous avons ajouter à *T_Jeu* les articles *Action_needed* et *time_action*. Le premier est un boolean indiquant que le Launcher attend une action, le second stock le temps alloué par le Launcher pour répondre. Nous ajoutons de plus les articles *move* et *amount_move* à *T_joueur* permettant de connaitre les actions du joueurs adverse et le montant de jetons reellement mis au pot. Cette information n'est pas toujours directement donné par le Launcher, en particulié dans le cas d'un *Call* ou *Bet* de surenchère. La connaissance de ces informations permet alors aisément de mettre en place un suivie en "temps reel" du stack des joueurs sans devoir attendre cette information du Launcher. Nous avons pris la peine d'implémenter ces nouvelle fonctionalités car Raphael et Baptiste en ont besoin pour l'élaboration de leur package *Strategie* (cf. 4.1 Etablissement du profil de l'adversaire).

Afin de facilité davantage l'appropriation du package par les autres membres du groupe, nous avons réalisé un fichier tests couplé à un script Python pour simulé le Bot jouant avec le Launcher. L'utilisateur peut alors envoyé autant de messages qu'il veut puis voir l'etat du Bot. Le lien entre le script Python et l'executable ADA est fait par l'ouverture d'un PIPE sur les flux standard. La fonction *Get* associé à *T_Message* peut alors etre utilisé de la meme facon que dans le Bot.

```
procedure test_MaJ_Bot is
   i : Integer := 0;
   message_tmp : T_message;
   -- Simulation partie
   Table : T_liste_cartes;
   info_partie : T_Jeu;
   J_Self,J_Other : T_joueur;
```

```
begin
-- Serie de commandes passees par le script python :
   Get(i);
   Skip_Line;
   for j in 1..i LOOP
      get(message_tmp);
      Update_data(message      => message_tmp,
                  Table        => Table,
                  info_partie  => info_partie,
                  J_Self       => J_Self,
                  J_Other      => J_Other);
   end loop;
-- Affichage de l'etat en fin de MaJ :
   Put_Line(" - ETAT DE LA PARTIE :");  Affiche_jeu(info_partie);
   Put_Line(" - ETAT DE LA TABLE :");   Affiche_liste_carte(Table);
   Put_Line(" - ETAT DE SELF :");       Affiche_joueur(J_Self);
   Put_Line(" - ETAT DE OTHER :");      Affiche_joueur(J_Other);
end test_MaJ_Bot;
```

Listing 7 – Test du package *Lecture_message*

```
import subprocess
# Creation d'un processus test_maj avec ouverture d'un PIPE en entree/sortie avec le script Python
process = subprocess.Popen('test_maj_bot.exe', shell=True,
                           stdout=subprocess.PIPE,
                           stderr=subprocess.STDOUT,
                           stdin = subprocess.PIPE)

# Commandes qui vont etre envoyees :
liste_commandes = []
liste_commandes.append("update_game big_blind 42000")
liste_commandes.append("update_game stack self 2000")
liste_commandes.append("update_game big_blind 60")
liste_commandes.append("update_game hand 1")
liste_commandes.append("update_game stack other 20")
liste_commandes.append("update_game stack self 2000")
liste_commandes.append("update_hand min_bet 60")
liste_commandes.append("update_hand hand other 2d,7c")
liste_commandes.append("update_hand hand self Td,Kc")
liste_commandes.append("update_hand pot 90")
liste_commandes.append("update_hand amount_to_call 30")
liste_commandes.append("update_hand table Ks,Kd,Ts")

process.stdin.write(str.encode("{}".format(len(liste_commandes))))
for commande in liste_commandes :
    process.stdin.write(str.encode(commande+"\n")) #Ecriture des commandes sur l'entree standard de test_maj

result = process.communicate()[0] # On lit que STDOUT, STDERR est redirige vers STDOUT
print(result.decode())

process.stdin.close() # Fermeture du process (meme comportement que la methode communicate)
process.wait()
```

```
 - ETAT DE LA PARTIE :
Blinds :  60; 15
Num de main :  1
Pot :  90
Amount_to_call :  30
Min_bet :  60
 - ETAT DE LA TABLE :
ROI de PIQUE
ROI de CARREAU
DIX de PIQUE
 - ETAT DE SELF :
Stack :  2000
Button : FALSE
Cartes du joueur :
DIX de CARREAU
ROI de TREFLE
 - ETAT DE OTHER :
Stack :  20
Button : FALSE
Cartes du joueur :
DEUX de CARREAU
SEPT de TREFLE

```

Listing 8 – Test du package *Lecture_message*

Nous nous assurons que tous les potentiels messages lues soit testés au sein de ce programme pour éviter tous

problème de stockage de données.

- Analyse_Cartes :

  La dernière partie exploitable du BOT d'origine est l'identification de la combinaison de carte d'un joueur avec la table. Comme précédemment, la première étape est d'encapsuler les fonctions associées à l'analyse des cartes se trouvant initialement dans le package *Intelligence_ artificielle*. Le principe de détection des combinaison est expliqué dans le compte rendu du S1 *(cf. 9.2 Analyse d'une combinaison)*. Le package en est extrêmement simplifié, sur la dizaine de procedures définies dans le package, seul deux sont présentes dans la partie public.

```
Type T_combinaison_nom is (hauteur, paire, double_paire, brelan, quinte, flush, full, carre,
    quinte_flush);
Type T_combinaison is private;

-- concatenation_cartes
-- E/ Deux listes de cartes : T_liste_cartes
-- Necessite : Len(liste_1)+Len(liste_2) < MAX_LISTE_CARTES
-- S/ Une liste de cartes : T_liste_cartes
-- Entraine : Une liste de cartes constituees des cartes des deux listes initiales
-- Verification : test_analyse_main.adb
function concatenation_cartes(Liste_1 : IN T_liste_cartes ; Liste_2 : IN T_liste_cartes)
    return T_liste_cartes;

-- detecte_meilleure_combinaison
-- E/ Liste de cartes : T_liste_cartes
-- Necessite : Null
-- S/ Combinaison : T_combinaison
-- Entraine : La meilleure combinaison de cartes possible au sein de la liste de cartes
-- Verification : test_analyse_main.adb
function detecte_meilleure_combinaison(cartes_in : IN T_liste_cartes) return T_combinaison;
```

Listing 9 – Partie public du package *Analyse_ main*

Les accesseurs et outils de comparaison du types privée *T_ combinaison* sont bien évidement présents.

Les fichiers de test implémenté au S1 on été modifié afin d'être utilisé par ce nouveau package. Lors de la phase de test du package par les autres membre de l'équipe, il est apparue que les quintes-flush n'étaient pas détectées. Ce problème s'est avéré relativement gênant à déboguer due à l'algorithme de détection de la combinaison utilisé au S1. Nous pouvions facilement détecter une suite ou une couleur au sein d'une liste de carte, cependant nous n'avions pas pris en compte le cas de détection d'une couleur au sein d'un sous-groupe de carte (la suite).

3 Évaluation de la force d'une main

3.1 Main de départ : Preflop (Etienne)

La première étape est la lecture efficace des cartes en pré-flop. La probabilité de gagner, en fonction de 2 cartes tirées, est pré-calculée. L'excel fourni "heads-up-preflop-win" indique le nombre de parties gagnées sur 2097572400 parties jouées soit toutes les parties possibles à partir de ces 2 cartes. Le cas d'égalité ne nous intéresse pas, ce choix est arbitraire et est pris en compte dans l'évaluation complète de force de main. Le travail en amont concerne ce tableau excel. Nous avons dû le transformer pour avoir un pourcentage lisible. Nous avons voulu l'importer dans ADA pour que le traitement soit plus rapide, nous savons que les valeurs ne peuvent pas changer, le jeu et ses règles étant bien établies.

,A,K,Q,J,T,9,8,7,6,5,4,3,2
A,11402312,34610976,37553414,41744140,46711746,53342951,60242341,67008466,72447218,77979612,79533094,79089680,78559848
K,35687839,11676960,41615142,45759944,50404375,56648553,63847580,70957375,77025547,82185635,83726409,83267308,82718864
Q,38723369,42939290,12299076,49852123,54407652,60474078,67152284,74622259,81105669,86250529,87773537,87294132,86722846
J,43129563,47298240,51535593,13277356,57602534,65045137,71484894,78462767,85231180,90862846,92368088,91868379,91274251
T,48393949,52214074,56357281,59642287,14746566,69247375,76570746,83391254,89796157,95533249,97596865,97076852,96459882
9,55504495,58911625,62861351,67618021,71987711,16428186,81573810,89249246,95528642,101068049,102984434,103094363,102454551
8,62862129,66641512,70028300,74526982,79840679,85106929,18695790,94484411,101722112,107166353,109025471,108693752,108755450
7,70129398,74254989,78084287,82048768,87190712,93373294,98901298,21423310,106652749,113123404,114963889,114627274,113942467
6,76022178,80805137,85058721,89387379,94133959,100178053,106737798,111965689,24530044,116843119,119659864,119483186,118782017
5,81983213,86422307,90660555,95491577,100409701,106238788,112692288,119028199,122982645,28731922,122535513,123097534,122489533
4,83768953,88195273,92415629,97228759,102727325,108455313,114847103,121159938,126168959,129230044,32147698,122271282,122123616
3,83453609,87864179,92064087,96856769,102334887,108721395,114693723,121002505,126183313,130040672,129193193,35819814,121342843
2,83116441,87508267,91685171,96454849,101909963,108273467,114982393,120541920,125706786,129665137,129308773,128529945,39805340

FIGURE 1 – Tableau d'évaluation de la main pre-flop (sous forme de liste)

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	0,84931916	0,66219608	0,65313699	0,64397621	0,63488915	0,61509674	0,60507809	0,59386672	0,58178898	0,58064119	0,57137799	0,56335054	0,55506261
K	0,64469378	0,82117335	0,6240842	0,61476557	0,60587065	0,5863814	0,5679041	0,55846321	0,54804671	0,53833852	0,52888844	0,52070127	0,51239952
Q	0,63508787	0,60432253	0,7963199	0,59070876	0,5817064	0,56222796	0,54417019	0,52523487	0,51679244	0,5071345	0,49763037	0,48938409	0,48101816
J	0,62535175	0,59441233	0,56906232	0,7715298	0,56154781	0,54111985	0,52311653	0,50454458	0,48574245	0,47820945	0,46868668	0,4604175	0,45202441
T	0,61568084	0,58494284	0,5594739	0,53826005	0,74660265	0,52376874	0,50509148	0,48651234	0,46800198	0,44939025	0,44204069	0,43378483	0,42540161
9	0,59449741	0,56407644	0,53862004	0,51639381	0,49815696	0,71665652	0,48856274	0,4699029	0,45151167	0,43312707	0,41407123	0,40806789	0,39972955
8	0,58374156	0,54431634	0,51930522	0,49713657	0,47818105	0,46068332	0,68717382	0,45684108	0,43818511	0,4199046	0,40102779	0,38282563	0,37679222
7	0,57169515	0,53417328	0,49904365	0,47726131	0,45829764	0,4407206	0,42693294	0,65725354	0,42829477	0,40979005	0,39108909	0,37303556	0,35439838
6	0,55870306	0,52297121	0,48996501	0,45713539	0,43848125	0,42103393	0,40696585	0,39653812	0,62700024	0,40348188	0,38480977	0,3668543	0,34837548
5	0,55742294	0,51253917	0,47959	0,44904648	0,41857476	0,4013672	0,38741271	0,37674684	0,37012755	0,59640035	0,38532532	0,36758677	0,34929535
4	0,5473287	0,50225154	0,46924765	0,43868735	0,41055389	0,38085796	0,36709172	0,35661729	0,35002989	0,3507482	0,56256514	0,35727358	0,33917944
3	0,53855312	0,49331264	0,46024897	0,42966762	0,40155186	0,37427914	0,34749849	0,33717911	0,33069789	0,33164981	0,32066305	0,52839237	0,33091972
2	0,52947306	0,48422782	0,4510993	0,42049266	0,39239115	0,36517013	0,3408683	0,3171028	0,31078656	0,31193807	0,30117407	0,2923945	0,49385176

FIGURE 2 – Tableau d'évaluation de la main pre-flop

On sépare les cases au niveau de chaque virgule puis on divise toutes les parties jouées par le nombre de parties total.

Ces valeurs sont ensuite copiées dans un tableau constant (*Data_winconstant*) sur ADA, chaque case est séparée une virgule, chaque ligne est dans une parenthèse. Nous avons décidé que la fonction *puispreflop* renverrait la puissance de la main du bot en pourcentage flottant, en accord avec les valeurs pré-calculées. Cette fonction consiste en la lecture du tableau. Le tableau est séparé en 2 : la diagonale supérieure concerne les cartes de même couleur la diagonale traite des paires la diagonale inférieure indique le pourcentage de gagner avec deux cartes de couleurs différentes. La couleur des cartes est donc analysée afin que nous piochons le bon pourcentage. La généricité améliore grandement la lisibilité de cet algorithme et l'implémentation est plus facile. Si les 2 cartes ont une même couleur, le motif le plus grand est lu sur les lignes et le plus petit sur les colonnes. Puisque les colonnes du tableau vont de l'AS au 2 et que le type *T_motif* est défini dans l'autre sens, une conversion est réalisée lors de la lecture du tableau : " pourc :=(tab(12-motif1,12-motif2))*100.0 ;". Nous avons enfin testé cet algorithme avec 2 cartes de couleurs différentes et 2 cartes de même couleurs.

```
1  package Force_mains is
2
3     Type T_winpreflop is array (0..12,0..12) of float;
4
5     -- puispreflop
6     -- E/ Joueur : T_joueur
7     -- Necessite : la table doit etre vide
8     -- S/ Float
9     -- Entraine : renvoie le pourcentage de gagner la partie
10
11    Function puispreflop(joueur: IN T_joueur) return float;
12
13 end Force_mains;
```

Listing 10 – Package Force_mains

```
_____________Couleur differente_____________
Stack :  0
Button : FALSE
Cartes du joueur :
NEUF de CARREAU
DIX de COEUR
La probabilite de gagner au preflop est de  4.98157E+01 %

_____________Meme couleur_____________
Stack :  0
Button : FALSE
Cartes du joueur :
NEUF de COEUR
DIX de COEUR
La probabilite de gagner au preflop est de  5.23769E+01 %
```

Listing 11 – Test_tableau

3.2 Génération aléatoire (PostFlop) (Louis et Arthur)

Ici, on se place dans le cas où le bot connaît ses cartes et au moins 3 cartes sur la table (3 à 5). Nous connaissons au moins 5 cartes (celles du joueur : 2 ; et celles de la table : 3) sur 9 cartes mises en jeu à la fin de la partie. Il manque la connaissance des cartes du joueurs adverses (2 cartes) et les 2 autres cartes qui vont sortir pendant la partie sur la table.

Dans le bot source, nous nous étions limités à la génération exhaustive des cartes inconnues de l'adversaire *cf. 9.3 Stratégie*. La génération de 4 cartes et le traitement est un processus extrêmement lourd en calcul ($47*46*45*44 = 4280760$ combinaisons de cartes à traiter). Nous avons réalisé quelques tests à l'aide d'un script Python afin d'établir le temps de détermination de la puissance d'une main avec les 5 cartes sur la table, 4 cartes puis seulement 3 cartes. Nous implémentons un fichier test Ada prenant en argument d'exécution le message de mis à jour des cartes du joueurs et de la table (0 à 5 cartes possible). L'exécutable renvoie en fin d'exécution sur la sortie standard le résultat obtenue pour le calcul exhaustif et le temps de calcul. On exploite cet exécutable grâce au script suivant.

```python
import subprocess
import random
import csv
from tqdm import tqdm

Nombre_test = 400

motif = ["2", "3", "4", "5", "6", "7", "8", "9", "T", "J", "Q", "K", "A"]
couleur = ["s", "d", "h", "c"]
liste_cartes = []
resultats = []

for i in tqdm(range(Nombre_test)):

    # Creer une liste aleatoire de cartes afin de les utiliser en commandes
    liste_cartes = []
    while len(liste_cartes)<7:
        carte = "{}{}".format(random.choice(motif), random.choice(couleur))
        if not carte in liste_cartes:
            liste_cartes.append(carte)

```

```
    # Envoie des arguments (messages)
    Args = [
        "update_hand;table;{},{},{},{}format(*liste_cartes[0:5]),
        "update_hand;hand;self;{},{}".format(*liste_cartes[5:7])]
    SubP_S2 = subprocess.Popen(["test_comparaison_algo.exe", Args[0], Args[1]], stdout=subprocess.
    PIPE)
    result = SubP_S2.communicate()

    # Reception des resultats
    data_row = []
    data_row.append(int(result[0].decode(encoding='UTF-8', errors='strict').split("; ")[0]))
    data_row.append(float(result[0].decode(encoding='UTF-8', errors='strict').split("; ")[1]))
    data_row.append("Table : {},{},{},{}".format(*liste_cartes[0:5]))
    data_row.append("Main self : {},{}".format(*liste_cartes[5:7]))

    resultats.append(data_row)

print(resultats)

# Ecriture d'un fichier pour exploiter les resultats
with open('Comparaison_algo.csv', mode='w') as comparaison_file:
    comparaison_writer = csv.writer(comparaison_file, delimiter=';', quotechar='"', quoting=csv.
    QUOTE_MINIMAL)
    for row in resultats:
        comparaison_writer.writerow(row)
```

Listing 12 – Test temporel de la fonction *Evaluation_main*

Voici un extrait des resultats.

Puissance	Durée	Cartes	Cartes
5	0.003108281	Table : 4s,5d,Ts,9d,Jc	Main self : 2d,3h
6	0.002893926	Table : 6s,Kd,6d,Ah,Td	Main self : 9c,5s
6	0.003032656	Table : Kh,8s,Ac,2c,Jh	Main self : 9h,6s
42	0.00306868	Table : 7s,4d,4c,8c,8d	Main self : 2d,Qh
45	0.003006341	Table : 6c,4h,2s,Ks,9h	Main self : Qh,4s
67	0.00300915	Table : 3s,Td,Ad,Tc,2c	Main self : Ac,7h
84	0.002908743	Table : 2c,2d,Tc,5c,Qc	Main self : Qs,Ah
96	0.002967251	Table : 7d,7s,Ks,9c,8h	Main self : 6s,8s
99	0.003031378	Table : 3s,9h,9s,7h,Kd	Main self : Ks,Kc
100	0.002960607	Table : 5d,9h,Kc,5c,Qs	Main self : 5h,5s

FIGURE 3 – 5 Cartes sur la table

Puissance	Durée	Cartes	Cartes
5	0.146205693	Table : 2s,Td,7s,3h	Main self : 9d,Qd
28	0.141862377	Table : 2c,8s,7d,8d	Main self : 4d,Kh
20	0.144058563	Table : Qs,Qd,Jh,Tc	Main self : 2c,7d
15	0.139708093	Table : Ad,8d,7s,Kh	Main self : 4h,Th
5	0.141398665	Table : Js,Ad,Qh,7s	Main self : 9h,4d
0	0.149894189	Table : Qc,2d,Ah,6h	Main self : 8d,7h
0	0.135043373	Table : 8d,Qc,7d,Kd	Main self : 5c,3c
14	0.140005993	Table : 5d,8c,Qh,4d	Main self : Ts,As
81	0.138844285	Table : 4h,Qd,8h,6s	Main self : 7d,5s
31	0.143226179	Table : 2s,9s,6c,9c	Main self : 5c,2h

FIGURE 4 – 4 Cartes sur la table

Puissance	Durée	Cartes	Cartes
53	6.35591131	Table : Ah,7d,4s	Main self : Kc,Ad
3	6.430591725	Table : 8d,Jh,Ks	Main self : Th,2s
70	6.170228906	Table : 7s,Js,6s	Main self : Ks,Td
30	6.362509564	Table : Kc,4d,Jh	Main self : 5h,5s
0	6.465917147	Table : Ad,9h,Kc	Main self : 5s,4c
3	6.377528733	Table : 5d,8s,Qh	Main self : 7c,Tc
12	6.341022951	Table : 4d,3h,Kc	Main self : 8c,Ah
1	6.40925023	Table : 2c,4s,4h	Main self : 8s,7h
7	6.264315998	Table : 7d,8s,Ts	Main self : Kd,Ac
1	6.253909682	Table : Ks,7c,Qs	Main self : 9h,6c

FIGURE 5 – 3 Cartes sur la table

La durée de traitement est problématique dans le cas où le nombre d'inconnues est le plus grand : 4 cartes inconnues i.e. 45.5^4 combinaisons avec la méthode exhaustive. Ainsi, il va être intéressant de cloisonner l'algorithme d'évaluation en fonction du contexte. Pour 2 à 3 inconnues, la génération exhaustive donne un résultat "exact" rapidement.

Il nous reste à établir une méthode pour traiter le cas de 4 inconnues permettant d'obtenir un résultat le plus proche possible d'une approche exhaustive en faisant moins de calculs. Nous allons limiter le temps ! Seulement en limitant le temps sur notre génération exhaustive, nous ne traiterons que certaines combinaisons trop proches les unes des autres et ne permettant pas d'avoir une réelle idée globale sur la force de notre main. Ainsi, pour obtenir un échantillon de cartes homogène au jeu de carte entier, il est essentiel de réaliser une génération de cartes aléatoires (Algorithme de Monte-Carlo)..

Pour 4 inconnues, nous allons utiliser la génération aléatoire. En effet, la génération exhaustive met 6 secondes à se réaliser, ce qui est beaucoup trop pour le maximum de 2 secondes mis à disposition.

Pour générer aléatoirement ces combinaisons de 4 cartes inconnues, nous allons tout d'abord générer 4 listes de 52 cartes mélangées. Le parcours de cette liste, au premier abord, pourrait être simple, c'est-à-dire 4 boucles imbriquées l'une dans l'autre représentant les 4 listes parcourues.

```
for i in 1..52 LOOP
    for j in 1..52 LOOP
        for k in 1..52 LOOP
            for l in 1..52 LOOP
            [...]
```

Listing 13 – Parcours des 4 listes

Seulement, avec cette méthode, la première liste est bien moins parcourue que la dernière, on ne balaye pas toutes les valeurs. D'autant plus, qu'un temps externe à ces boucles va couper le parcours des boucles une fois le temps de x secondes atteint (partie suivante). Ainsi pour améliorer ce parcours, nous avons décidé de parcourir la liste de façon alternée entre parcours normal (comme décrit précédemment) et parcours inverse. Dans cet autre parcours on part de la fin de la liste (dernière ligne dernière colonne et on effectue le parcours inverse), voici un schéma explicatif (parce qu'une image vaut mieux que des mots ici) :

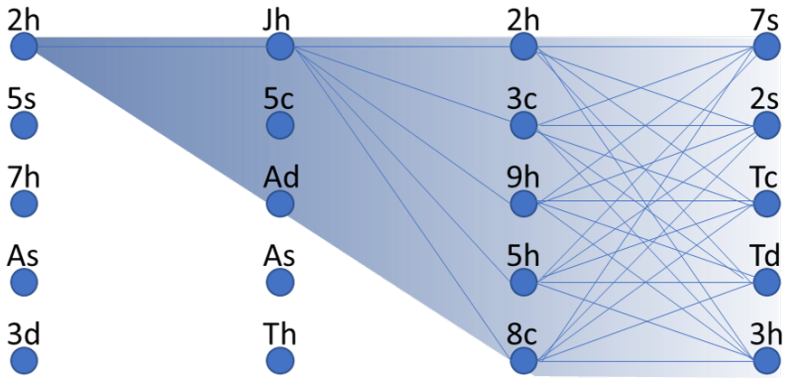


FIGURE 6 – Schéma version normale sur 4 listes de 5 cartes

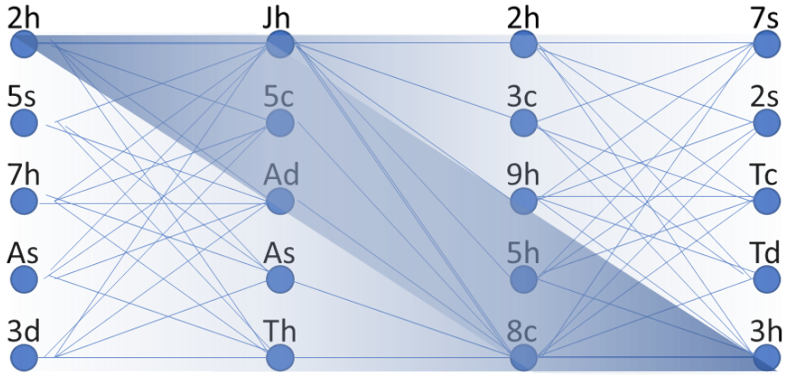


FIGURE 7 – Schéma amélioré sur 4 listes de 5 cartes

Remarque : On peut constater que la même carte peut se retrouver plusieurs fois dans la même combinaison, ceci est aberrant physiquement (carte unique dans un paquet) mais l'erreur sur le calcul de la force de la main sera négligeable et donc ce cas ne sera pas pris en compte.

Ce parcours nous permet d'avoir une plus grande homogénéisation et de ne pas laisser un grand paquet de valeur non exploité. D'autres techniques auraient pu être imaginées. En effet, enfin de compte plus l'on homogénéise le tracé sur le schéma précédent plus, la précision du résultat sera grande (écart avec le résultat exact). Pour gérer l'alternance des deux parcours, nous avons mis en place 1 booléen qui change de valeur à chaque passage et donc permet de changer de condition et donc de parcours.

```
1 IF Flag_inverse THEN
2     Set_liste_carte(Liste_alea_1(i), main_tmp,1);
3     Set_liste_carte(Liste_alea_2(j), main_tmp,2);
4     Set_liste_carte(Liste_alea_3(k), Table_tmp,4);
```

```
    Set_liste_carte(Liste_alea_4(l), Table_tmp,5);
ELSE
    Set_liste_carte(Liste_alea_1(53-l), main_tmp,1);
    Set_liste_carte(Liste_alea_2(53-k), main_tmp,2);
    Set_liste_carte(Liste_alea_3(53-j), Table_tmp,4);
    Set_liste_carte(Liste_alea_4(53-i), Table_tmp,5);
END IF;
Flag_inverse := NOT Flag_inverse;
```

Listing 14 – Booléen *Flag_ inverse* dans les boucles imbriquées

Finalement nous obtenons des résultats assez proche du calcul exhaustif. La différence entre le pourcentage calculé en 1.5s et le pourcentage calculé exhaustivement est en moyenne de 1.39% sur 400 essaies.

Puissance 1	Temps 1	Puissance 2	Temps 2	Cartes	Cartes	Différence
20	1.500004471	22	6.519167158	Table : Qd,Ad,9s	Main self : 9c,2h	2
23	1.500004727	20	6.41720232	Table : 3s,Kd,Qs	Main self : Ts,3h	3
0	1.500004215	0	6.453564262	Table : Js,9h,Qd	Main self : 5s,2c	0
1	1.500004982	1	7.141768979	Table : Jh,7d,Ac	Main self : 6c,3c	0
5	1.50000396	6	6.326799121	Table : 6s,2h,Ks	Main self : 7h,Ts	1
5	1.500004982	6	6.511925325	Table : 2s,8h,3d	Main self : Ts,Kd	1
76	1.500004216	77	6.490152565	Table : 3c,5c,9h	Main self : 7c,4c	1
0	1.500004471	0	6.372966469	Table : 7d,Ts,3s	Main self : Js,9d	0
19	1.500006259	17	6.428990577	Table : Ks,9c,Th	Main self : 3d,3s	2
3	1.500004471	4	6.297972568	Table : Qd,3c,Td	Main self : 7d,Jh	1
9	1.50000396	11	6.336778005	Table : 3d,Kh,Jh	Main self : 6h,Ad	2

FIGURE 8 – Extrait de l'évaluation temporelle du calcul de puissance

3.3 Gestion du temps (PostFlop) (Louis et Arthur)

Pour gérer le temps nous avons mis en place une simple boucle qui englobe le processus de génération aléatoire et qui vérifie que le temps consacré à l'évaluation de la main n'a pas été dépassé. Si, c'est le cas, on sort de la boucle : l'évaluation s'arrête et on obtient le pourcentage de la force de la main évalué jusqu'à ce moment.

La condition de cette boucle est que l'écart entre le temps initial obtenu par une commande clock() en début d'algorithme et le temps calculé à chaque passage dans la dernière boucle ne doit pas dépasser un pourcentage de la valeur donnée par *Action*. Le reste du temps étant consacré à la prise en compte du profil de l'adversaire. Ce pourcentage peut être modifié directement en changeant l'argument de la fonction qui le demande. Ici, la condition est t $<$ 0.8*Valeur de *Action* :

```
Cycle_calcul :
    for i in 1..52 LOOP
        for j in 1..52 LOOP
            for k in 1..52 LOOP
                for l in 1..52 LOOP
                    exit Cycle_calcul when (Float(Clock-Temps_init)>Limite_duree);
                End LOOP;
            End LOOP;
        End LOOP;
    End LOOP Cycle_calcul;
```

Listing 15 – Structure de la boucle limité en temps

Grâce a cette méthode, la génération aléatoire se fait sans soucis de temps et s'arrête au moment ou elle dépasse le temps demandé.

4 Estimation de la stratégie de l'adversaire (Raphaël et Baptiste)

Un bon joueur de poker ne prend pas seulement en compte la puissance de sa main, mais essaye de lire le jeu de son adversaire et de déceler sa stratégie afin de pouvoir anticiper ses mouvements et augmenter ainsi ses chances de gagner. C'est ce que nous avons tenté d'implémenter dans notre bot : un algorithme de détection de la stratégie de l'adversaire.

Voici comment notre algorithme est construit :

Idée : A chaque fin de main, nous enregistrons dans un tableau les cartes qu'a eu l'adversaire à chaque main (lorsque que l'on accède à cette information), et les actions qu'il a effectué au cours de chaque main. Pour être plus précis voici la liste complète des informations que nous avons stockés dans notre tableau :

- ☐ Les deux joueurs self et other
- ☐ Notre stack et celui de l'adversaire
- ☐ Le couillu (L'adversaire bluff souvent)
- ☐ Notre main ainsi que la puissance calculée (celle de l'adversaire aussi quand on accède à cette information)
- ☐ La table
- ☐ Le montant misé lors de la dernière action du joueur other

```
1  ------------------------------------------
2  ----------TYPES PRIVES ----------
3  ------------------------------------------
4
5     type T_tour is record
6        Stack_self : Natural :=0;
7        Main_self : T_liste_cartes;
8        Stack_other : Natural :=0;
9        Main_other : T_liste_cartes;
10       Table : T_liste_cartes;
11       puissance_main_self : Natural := 0;
12       puissance_main_other : Natural := 0;
13       self : T_joueur;
14       other : T_joueur;
15       Amount_move_other : Natural := 0;
16    end record;
17    type stockage_tours_tab is array (1..N_max_tour) of T_tour;
18    type stockage_tours is record
19       tours : stockage_tours_tab;
20       nb_tours : Natural := 0;
21    end record;
```

Listing 16 – types privés permettant le stockage des informations

Ces informations sont la matière première que nous allons exploiter pour déceler la stratégie de notre adversaire. C'est pourquoi nous les sauvegardons à chaque fois qu'une action est demandée grâce à la procédure Stocke_tour (procédure privée) :

```
procedure stocke_tour(jeu : in T_Jeu; self,other : in T_joueur; table : in T_liste_cartes;
   stockage : in out stockage_tours; puissance_self : in natural) is
      i : Natural:=1;
   begin
      i := stockage.nb_tours+1;
      stockage.nb_tours := i;
      stockage.tours(i).self := self;
      stockage.tours(i).other := other;
      stockage.tours(i).Stack_self := Get_Stack(self);
      stockage.tours(i).Stack_other := Get_Stack(other);
      stockage.tours(i).Main_self := Get_joueur_main(self);
      stockage.tours(i).Main_other := Get_joueur_main(other);
      stockage.tours(i).Table := table;
      stockage.tours(i).puissance_main_self := puissance_self;
      stockage.tours(i).Amount_move_other := Get_joueur_amount_move(other);
   end stocke_tour;
```

Listing 17 – implémentation de stocke$_t$*our*

On notera que celle ci est simple grâce aux getteurs que nous avons demandé au groupe d'Arthur et Louis lors de l'encapsulation de leur bot de départ.

Afin de déceler la stratégie de notre adversaire, nous allons construire 4 profils type de joueurs. L'objectif de notre algorithme sera donc d'associer notre adversaire au profil auquel il correspond. Ainsi la stratégie adoptée par notre bot ne sera pas la même en fonction du profil de joueur qu'il aura en face.

Nb : Le profil du joueur se construit au cours de la partie, c'est à dire qu'un même adversaire peut passer d'un profil à l'autre en fonction de l'évolution de son style de jeu.

4.1 Établissement du profil de l'adversaire (Raphaël et Baptiste)

Voici les 4 profils de joueurs que nous avons construits :

- ☐ Le frileux (dès qu'on mise une somme importante l'adversaire se couche)
- ☐ Le con (l'adversaire ne se couche quasiment jamais)
- ☐ Le couillu (L'adversaire bluff souvent)
- ☐ Le Suisse (l'adversaire est neutre, pas de stratégie particulière)

Nb : Vous nous excuserez pour ces termes quelques peu bourrus, mais nous les trouvions plus parlant.

Le Frileux : Nous regardons toutes les sommes misées par notre adversaire. Nous créons un critère, le critère F. A chaque fois que l'adversaire a misé,nous faisons le rapport de la somme misée par l'adversaire sur la valeur de son stack (nombre compris entre 0 et 1), que j'additionne à F, lui-même initialisé à 0. A la fin de l'analyse, on normalise F en le divisant par le nombre de fois où l'adversaire a misé. Plus F est faible, plus l'adversaire est frileux.

Le Con : Nous utilisons le critère F, plus celui-ci est grand, plus l'adversaire est con ! Nous introduisons donc le critère Cn = 1 - F. Plus celui-ci est petit, plus l'adversaire est con !

Le Couillu : Nous mettons en place le critère Cu. Nous procédons en 2 étapes :

1ère étape : Nous décortiquons tous les cas où nous avons eu accès aux cartes de l'adversaire (les 2 joueurs sont donc allés jusqu'au bout). Nous faisons le rapport de la puissance de ses cartes sur la puissance de nos cartes, et ce quel que soit le vainqueur de la manche (plus ce rapport est faible, plus le joueur a eu du cran de nous suivre). Nous additionnons ce rapport au critère Cu, puis nous le normalisons en divisant par le nombre de fois où on a eu accès aux cartes de l'adversaire. Plus Cu est faible, plus l'adversaire est couillu !

2nd étape : Nous décortiquons toutes les fois où nous avons dû nous coucher. En effet si nous nous sommes couchés alors que notre main avait une grande puissance, il y a plus de chance que l'adversaire bluff plutôt qu'il ait une bonne main. A nous donc de coefficienter le critère Cu en fonction de ce critère (nous rectifierons ce coefficient Coeff_CU en fonction de nos simulations) Cela permet donc d'affiner notre critère Cu.

```
1 -- determination_profil_adversaire
2 -- E/ jeu : T_Jeu ; self,other : T_joueur ; Table : T_liste_cartes; puissance_self : Natural
3 -- E/S/ stockage : stockage _tours
4 -- S/ Profil : T_profil_adversaire
5 -- Entraine : Determine le profil de l'adversaire
6 procedure determination_profil_adversaire(jeu : in T_Jeu; self,other : in T_joueur; table : in
      T_liste_cartes; stockage : in out stockage_tours;puissance_self : in Natural; profil : out
      T_profil_adversaire;F,Cu,Cn : out Float);
```

Listing 18 – procédure Détermination_profil_adversaire

Une fois les critères F, Cn et Cu élaborés, nous les comparons à des valeurs calibrées expérimentalement sur un grand nombre de parties avec d'autres bots plus ou moins agressif (on pourrait faire cela avec du machine learning, peut être dans la partie 3 du projet si partie 3 il y a). Pour pouvoir calibrer ces coefficients F,Cn,Cu, on a sauvegardé chaque tour d'une partie dans un fichier texte grâce au fonctions open(file_type) de ADA et aux fonction init_sauvegarde_tour et sauvegarde_tour de notre package On obtient des feuilles excel de ce type :

Stack self	Stack other	Puissance Main Self	move_self	Puissance main adv	move other	jetons depenses other	F	CU	CN	Profil_other
1000	1000	35	FOLD	0	FOLD	20	NaN********	1.00000E+00	NaN********	SUISSE
980	925	35	CALL	0	BET	75	8.10811E-02	1.00000E+00	9.18919E-01	FRILEUX
980	924	42	FOLD	0	BET	96	9.24886E-02	1.00000E+00	9.07511E-01	FRILEUX
960	1040	64	FOLD	0	BET	20	6.80693E-02	7.00000E-01	9.31931E-01	FRILEUX
940	926	64	CALL	0	BET	114	8.18295E-02	7.00000E-01	9.18170E-01	FRILEUX
940	992	35	FOLD	0	BET	68	7.91733E-02	7.00000E-01	9.20827E-01	FRILEUX
920	1080	42	FOLD	0	BET	20	6.90642E-02	7.00000E-01	9.30936E-01	FRILEUX
910	1009	49	FOLD	0	BET	81	7.06661E-02	7.00000E-01	9.29334E-01	FRILEUX
890	1110	44	FOLD	0	BET	20	6.40851E-02	7.00000E-01	9.35915E-01	FRILEUX
870	1016	44	CALL	0	BET	94	6.72445E-02	7.00000E-01	9.32756E-01	FRILEUX
870	1034	52	FOLD	0	BET	96	6.98044E-02	7.00000E-01	9.30196E-01	FRILEUX
850	1150	36	FOLD	0	BET	20	6.50395E-02	7.00000E-01	9.34960E-01	FRILEUX
840	1073	42	FOLD	0	BET	87	6.63763E-02	7.00000E-01	9.33624E-01	FRILEUX
820	1180	56	FOLD	0	BET	30	6.32261E-02	7.00000E-01	9.36774E-01	FRILEUX
790	1110	56	CALL	0	BET	70	6.32145E-02	7.00000E-01	9.36786E-01	FRILEUX
790	1108	44	FOLD	0	BET	102	6.51374E-02	7.00000E-01	9.34863E-01	FRILEUX
760	1240	36	FOLD	0	BET	30	6.25784E-02	7.00000E-01	9.37422E-01	FRILEUX
745	1143	48	FOLD	0	BET	112	6.46613E-02	7.00000E-01	9.35339E-01	FRILEUX
715	1285	63	FOLD	0	BET	30	6.23660E-02	4.90000E-01	9.37634E-01	COUILLU
685	1112	63	CALL	0	BET	173	6.72718E-02	4.90000E-01	9.32728E-01	COUILLU
685	1250	64	FOLD	0	BET	65	6.65082E-02	3.43000E-01	9.33492E-01	COUILLU
620	1250	10	CALL	0	BET	65	6.65082E-02	3.43000E-01	9.33492E-01	COUILLU
620	1219	9	CHECK	0	BET	31	6.45521E-02	3.43000E-01	9.35448E-01	COUILLU
589	1219	15	CALL	0	BET	31	6.45521E-02	3.43000E-01	9.35448E-01	COUILLU
589	1219	100	CHECK	0	CHECK	0	6.45521E-02	3.43000E-01	9.35448E-01	COUILLU
1047	953	48	BET	0	CALL	30	6.30488E-02	3.43000E-01	9.36951E-01	COUILLU
1017	915	48	CALL	0	BET	38	6.21132E-02	3.43000E-01	9.37887E-01	COUILLU
979	877	65	CALL	0	BET	38	6.13306E-02	3.43000E-01	9.38669E-01	COUILLU
941	877	63	CALL	0	CHECK	0	6.13306E-02	3.43000E-01	9.38669E-01	COUILLU
941	877	50	CHECK	0	CHECK	0	6.13306E-02	3.43000E-01	9.38669E-01	COUILLU
1153	788	34	CHECK	0	BET	59	6.18723E-02	3.43000E-01	9.38128E-01	COUILLU
1094	788	0	CALL	0	BET	59	6.18723E-02	3.43000E-01	9.38128E-01	COUILLU
1094	788	1	CHECK	0	CHECK	0	6.18723E-02	3.43000E-01	9.38128E-01	COUILLU
1094	788	36	CHECK	0	CHECK	0	6.18723E-02	3.43000E-01	9.38128E-01	COUILLU

FIGURE 9 – Outil de calibrage

```
-- init_sauvegarde_tour
-- E/ fichier : File_Type
-- Entraine : Initialise les colonnes du fichier texte pour la sauvegarde des donn es
procedure init_sauvegarde_tour(fichier: in File_Type);

-- sauvegarde_tour
-- E/ jeu : fichier : File_Type; stockage : stockage_tour; F,CU,CN : Float; Profil :
    T_profil_adversaire
-- Entraine : sauvegarde les donn es d'un tour dans le fichier texte fichier
procedure sauvegarde_tour(fichier : in File_Type; stockage : in stockage_tours; F,CU,CN : in Float
    ; Profil : in T_profil_adversaire);
```

Listing 19 – procédures utiles aux tests et calibrage des paramètres

Finalement on obtient :

☐ F <= 0.1 et Cu > 0.5 l'adversaire est Frileux

☐ Cn <= 0.8 et Cu > 0.5 l'adversaire est Con

☐ Cu <= 0.5 l'adversaire est Couillu

☐ Sinon l'adversaire a un profil Suisse

Donc on peut implémenter ceci dans la procédure détermination_profil_adversaire :

```
procedure determination_profil_adversaire(jeu : in T_Jeu; self,other : in T_joueur; table : in
    T_liste_cartes; stockage : in out stockage_tours;puissance_self : in Natural; profil : out
    T_profil_adversaire ;F,Cu,Cn : out Float) is
   begin
```

```
      stocke_tour(jeu            => jeu,
                  self           => self,
                  other          => other,
                  table          => table,
                  stockage       => stockage,
                  puissance_self => puissance_self);
      Profil_adversaire(stockage => stockage,
                        profil   => profil,
                        F        => F,
                        Cu       => Cu,
                        Cn       => Cn);
   end determination_profil_adversaire;
```

Listing 20 – implémentation de détermination_profil_adversaire

Avec la procédure Profil_adversaire :

```
procedure Profil_adversaire(stockage : in stockage_tours; profil : out T_profil_adversaire  ;F,Cu
   ,Cn : out Float) is
      i : Natural := stockage.nb_tours;
      nb_pari, Coeff_Cu: Float;
      tour, tour_prec : T_tour;
      n_acces_cartes_adv : natural := 0;
      fichier : File_Type;
   begin
      Coeff_Cu := 0.7;
      nb_pari := 1.0;
      F := 0.0;
      Cn := 0.0;
      Cu :=  1.0;
      For t in 1..i loop
         tour := stockage.tours(t);
         if Get_nbr_liste_carte(Get_joueur_main(tour.other)) > 1 then
            n_acces_cartes_adv := n_acces_cartes_adv + 1;
            Coeff_Cu := ((float(Get_puissance_main(Table=> tour.Table,Self=> tour.other,
    Limite_duree => 0.1))/float(tour.puissance_main_self))+Coeff_Cu)/float(n_acces_cartes_adv);
         end if;
         if Get_joueur_move(tour.self) = Fold then

            If tour.puissance_main_self >= 60 then
               Cu := Cu * Coeff_Cu;
            end if;
         end if;
         if Get_joueur_move(tour.other) = call or Get_joueur_move(tour.other) = bet then
            tour_prec := stockage.tours(t-1);
            if tour.Stack_other /= tour_prec.Stack_other then --eviter les redondances
               F := F+Float(tour.Amount_move_other)/Float(tour.Stack_other);
               nb_pari := nb_pari +1.0;
            end if;
         end if;
      end loop;
      F:=F/(nb_pari-1.0);
      Cn := 1.0-F;
      if F <= 0.1 and Cu > 0.5 then             --Determination du profil en fonction
         profil := Frileux;                     --des parametres
      elsif Cn <= 0.8 and Cu > 0.5 then
         profil := con;
      elsif Cu <= 0.5 then
         profil := Couillu;
      else profil := Suisse;
      end if;
```

```
43    end Profil_adversaire;
```

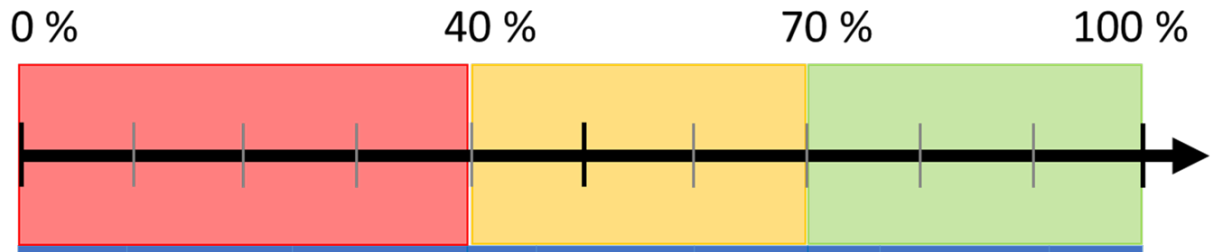
Listing 21 – implémentation de profil_adversaire

4.2 Réaction du Bot BEARL (Raphael, Baptiste et Louis)

La fonction Think_Then_Play :

Cette fonction est finalement celle qui va permettre à notre bot de réfléchir et d'agir en conséquence à l'aide de tous les paramètres déterminés par les packages des autres sous groupes de notre groupe de projet. En effet, cette procédure est appelée à chaque fois qu'une action est demandée par le launcher.

Ainsi, selon le profil de l'adversaire obtenu et le pourcentage évaluant la force de la main, nous avons défini différentes stratégies à adopter comme nous avions fait pour le bot source (*cf 9.3 Stratégie*). Les voici résumées dans un tableau :



Profil	Amount_to_call	Action	Profil	Amount_to_call	Action	Profil	Amount_to_call	Action
Frileux	0	CHECK	Frileux	0	BET (pourcentage-40)*stack%	Frileux	0	BET TAPIS
	Autre	FOLD		<0.5*(pourcentage-40)*stack%	CALL		<(pourcentage-60)*stack%	CALL
				Autre	FOLD		Autre	FOLD
Con	0	CHECK	Con	0	CHECK	Con	<MP/2	BET MP
	< 10% du stack	CALL		<1.5*(pourcentage-40)*stack%	CALL		<(pourcentage-30)*stack%	CALL
	Autre	FOLD		Autre	FOLD		Autre	FOLD
Couillu	0	CHECK	Couillu	0	CHECK	Couillu	0	BET 40*stack %
	<10% du stack	CALL		<(pourcentage-40)*stack%	CALL		Autre	CALL
	Autre	FOLD		Autre	FOLD			
Suisse	0	CHECK	Suisse	0	CHECK	Suisse	<MP*3/2	BET MP
	<10% du stack	CALL		<(pourcentage-40)*stack%	CALL		<(pourcentage-30)*stack%	CALL
	Autre	FOLD		Autre	FOLD		Autre	FOLD

Mise potentielle (MP)
= (pourcentage*2-110)*stack %

FIGURE 10 – Stratégie en fonction du profil de l'adversaire et du pourcentage de gagne

Nous les avons mis en place directement dans le *bot.adb* comme au S1 en formulant plusieurs *case*. Sauf qu'ici, il y a beaucoup plus de cas puisque pour les 3 catégories de pourcentage, on prend maintenant en compte le profil de l'adversaire.

Remarque : La mise potentielle MP ainsi que certaines stratégies n'ont pas changé par rapport au S1.

Voici donc cette procédure Think_then_play ainsi que sa spécification :

```
1 -- Think_Then_Play
2 -- E/ Force_main : Natural ; profil_adv : T_profil_adversaire ; jeu : T_jeu
3 -- E/S/ self,other : T_joueur
4 -- Entraine : Determine l'action    jouer dans le launcher et la joue
```

```
procedure Think_Then_Play(force_main : in Natural; profil_adv : in T_profil_adversaire; jeu : in
    T_Jeu; self,other : in out T_joueur);
```

Listing 22 – Spécification de Think_Then_Play

```
procedure Think_Then_Play(force_main : in Natural; profil_adv : in T_profil_adversaire; jeu : in
    T_Jeu; self,other : in out T_joueur) is
      Amount_to_call : Natural;
      Stack : Natural;
      mise_potentielle : Natural := 0 ;
   begin
      Amount_to_call := Get_Amount_to_call(jeu);
      Stack := Get_Stack(self);
      if force_main >=70 then
         mise_potentielle :=(force_main*2-110);
      end if;

      case profil_adv is
      when frileux =>                                          -- FRILEUX
         if force_main <= 40 then
            if Amount_to_call = 0 then
               Jouer(move        => check,
                     Amount_move => 0,
                     Self        => self,
                     Other       => other);
            else
               Jouer(move        => fold,
                     Amount_move => 0,
                     Self        => self,
                     Other       => other);
            end if;
         elsif force_main > 40 and force_main <= 70 then
            if Amount_to_call = 0 then
               Jouer(move        => bet,
                     Amount_move => integer(float((force_main-40))*0.01*float(Stack)),
                     Self        => self,
                     Other       => other);
            elsif float(Amount_to_call) <  (0.5*(float(force_main)-40.0)*float(Stack))*0.01 then
               Jouer(move        => call,
                     Amount_move => 0,
                     Self        => self,
                     Other       => other);
            else
               Jouer(move        => fold,
                     Amount_move => 0,
                     Self        => self,
                     Other       => other);
            end if;
         else
            if Amount_to_call = 0 then
               Jouer(move        => bet,
                     Amount_move => Integer(0.8*float(Stack)),
                     Self        => self,
                     Other       => other);
            elsif float(Amount_to_call) < float(force_main-60)*0.01*Float(Stack) then
               jouer(move        => call,
                     Amount_move => 0,
                     Self        => self,
                     Other       => other);
            else
               Jouer(move        => fold,
                     Amount_move => 0,
                     Self        => self,
```

```
                    Other       => other);

            end if;
        end if;

    when con =>                                         -- CON
        if force_main <= 40 then
            if Amount_to_call = 0 then
                Jouer(move        => check,
                      Amount_move => 0,
                      Self        => self,
                      Other       => other);
            elsif float(Amount_to_call) < 0.1 * Float(Stack) then
                Jouer(move        => call,
                      Amount_move => 0,
                      Self        => self,
                      Other       => other);
            else
                Jouer(move        => fold,
                      Amount_move => 0,
                      Self        => self,
                      Other       => other);
            end if;
        elsif force_main > 40 and force_main <= 70 then
            if Amount_to_call = 0 then
                Jouer(move        => check,
                      Amount_move => 0,
                      Self        => self,
                      Other       => other);
            elsif float(Amount_to_call) < (1.5*(float(force_main)-40.0)*float(Stack))*0.01 then
                Jouer(move        => call,
                      Amount_move => 0,
                      Self        => self,
                      Other       => other);
            else
                Jouer(move        => fold,
                      Amount_move => 0,
                      Self        => self,
                      Other       => other);
            end if;
        else
            if Amount_to_call < mise_potentielle/2 then
                Jouer(move        => bet,
                      Amount_move => mise_potentielle,
                      Self        => self,
                      Other       => other);
            elsif float(Amount_to_call) < float((force_main-30)*Stack)*0.01 then
                jouer(move        => call,
                      Amount_move => 0,
                      Self        => self,
                      Other       => other);
            else
                Jouer(move        => fold,
                      Amount_move => 0,
                      Self        => self,
                      Other       => other);
            end if;
        end if;


        when couillu =>                                 -- COUILLU
            if force_main <= 40 then
```

```
120                if Amount_to_call = 0 then
121                   Jouer(move        => check,
122                         Amount_move => 0,
123                         Self        => self,
124                         Other       => other);
125                elsif float(Amount_to_call) < 0.1 * Float(Stack) then
126                   Jouer(move        => call,
127                         Amount_move => 0,
128                         Self        => self,
129                         Other       => other);
130                else
131                   Jouer(move        => fold,
132                         Amount_move => 0,
133                         Self        => self,
134                         Other       => other);
135                end if;
136             elsif force_main > 40 and force_main <= 70 then
137                if Amount_to_call = 0 then
138                   Jouer(move        => check,
139                         Amount_move => 0,
140                         Self        => self,
141                         Other       => other);
142                elsif float(Amount_to_call) <  (float(force_main)-40.0)*float(Stack)*0.01 then
143                   Jouer(move        => call,
144                         Amount_move => 0,
145                         Self        => self,
146                         Other       => other);
147                else
148                   Jouer(move        => fold,
149                         Amount_move => 0,
150                         Self        => self,
151                         Other       => other);
152                end if;
153             else
154                if Amount_to_call = 0 then
155                   Jouer(move        => bet,
156                         Amount_move => Integer(0.4*float(Stack)),
157                         Self        => self,
158                         Other       => other);
159                else
160                   Jouer(move        => call,
161                         Amount_move => 0,
162                         Self        => self,
163                         Other       => other);
164                end if;
165             end if;
166
167
168          when suisse =>                                        -- SUISSE
169             if force_main <= 40 then
170                if Amount_to_call = 0 then
171                   Jouer(move        => check,
172                         Amount_move => 0,
173                         Self        => self,
174                         Other       => other);
175                elsif float(Amount_to_call) < 0.1 * Float(Stack) then
176                   Jouer(move        => call,
177                         Amount_move => 0,
178                         Self        => self,
179                         Other       => other);
180                else
181                   Jouer(move        => fold,
```

```
182                         Amount_move => 0,
183                         Self        => self,
184                         Other       => other);
185                end if;
186             elsif force_main > 40 and force_main <= 70 then
187                if Amount_to_call = 0 then
188                   Jouer(move        => check,
189                         Amount_move => 0,
190                         Self        => self,
191                         Other       => other);
192                elsif float(Amount_to_call) <  (float(force_main)-40.0)*float(Stack)*0.01 then
193                   Jouer(move        => call,
194                         Amount_move => 0,
195                         Self        => self,
196                         Other       => other);
197                else
198                   Jouer(move        => fold,
199                         Amount_move => 0,
200                         Self        => self,
201                         Other       => other);
202                end if;
203             else
204                if float(Amount_to_call) < 3.0/2.0*Float(mise_potentielle) then
205                   Jouer(move        => bet,
206                         Amount_move => mise_potentielle,
207                         Self        => self,
208                         Other       => other);
209                elsif float(Amount_to_call) < float((force_main-30)*Stack)*0.01 then
210                   jouer(move        => call,
211                         Amount_move => 0,
212                         Self        => self,
213                         Other       => other);
214
215                else
216                   Jouer(move        => fold,
217                         Amount_move => 0,
218                         Self        => self,
219                         Other       => other);
220                end if;
221             end if;
222       end case;
223    end Think_Then_Play;
```

Listing 23 – Implémentation de Think_Then_Play

5 Conclusion

Bien que le Bot ne soit pas au meilleur de ses performances (la stratégie pourra être repensée plus en détail avec des fonctions plus précises), celui-ci est implémenté d'une plus "propre" façon qu'au S1 grâce à une encapsulation bien établie pour chaque package ce qui permet de tout faire fonctionner ensemble alors que les divers parties ont été réalisées par différentes personnes. Le code est plus lisible et robuste. De plus, il prend en compte plus de paramètres, le rendant plus proche de la réalité.

Voici le fichier *bot.adb* final (qui illustre bien cette meilleure lisibilité) :

```
1  WITH ada.text_io, Modelisation_poker, lecture_messages, Analyse_cartes, strategie_adversaire,
       Evaluation_main;
2  USE  ada.text_io, Modelisation_poker, lecture_messages, Analyse_cartes, strategie_adversaire,
       Evaluation_main;
3
4  procedure bot is
5     message : T_message;
6     info_partie : T_jeu;
7     Self : T_joueur;
8     Other : T_joueur;
9     Table : T_liste_cartes;
10    Profil_adversaire : T_profil_adversaire := Suisse;
11    Puissance_main_self : Natural :=70;
12    stockage : stockage_tours;
13 begin
14    while not End_Of_File loop   -- Loop infini de jeu
15
16       Get(message);
17       Update_data(message     => message,
18                   Table       => Table,
19                   info_partie => info_partie,
20                   J_Self      => Self,
21                   J_Other     => Other);
22
23       if Get_Action_needed(info_partie) THEN -- Si le launcher attend une reponse
24
25          Puissance_main_self := Get_puissance_main(Table         => Table,
26                                                    Self          => Self,
27                                                       Limite_duree => (Get_time_to_play(info_partie)
       /1000.0)*0.8);
28
29          determination_profil_adversaire(jeu            => info_partie,
30                                          self           => Self,
31                                          other          => Other,
32                                          table          => Table,
33                                          stockage       => stockage,
34                                          puissance_self => Puissance_main_self,
35                                          profil         => Profil_adversaire);
36
37          Think_Then_Play(force_main => Puissance_main_self,
38                          profil_adv => Profil_adversaire,
39                          jeu        => info_partie,
40                          self       => Self,
41                          other      => Other);
42
43          Set_Action_needed(info_partie, False);
44       end if;
45    end loop;
46 end;
```

Listing 24 – Bot.adb

Deuxième partie

ANNEXE : Bot de Base

6 Introduction

Dans ce projet nous implémenterons un *Bot* capable de jouer au *Poker Head's up No limit Texas Hold'em* en interaction avec un moteur de jeu (*i.e.* le *Launcher*). Toutes les informations nécessaires au bout déroulement de la partie (actions de l'adversaire, cartes sur la table, quantité de jetons dans le pot ...) sont fournies par le Launcher au fur et à mesure de la partie. Le Bot devra donc être capable de lire ces informations, les stocker en interne dans un format adapté pour une utilisation optimale et finalement jouer correctement face au bot adverse. On identifie alors 3 principaux objectifs dans la construction de ce programme : la représentation des éléments de jeu, l'interaction avec le launcher et l'intelligence artificielle. Un projet de cet envergure nécessitera un test continu des primitives implémentées pour ce prévenir de bugs lors de l'utilisation finale du *Bot*.

Notre programme repose sur trois packages chacun dédié à un des trois principaux objectifs du projet définis précédemment.

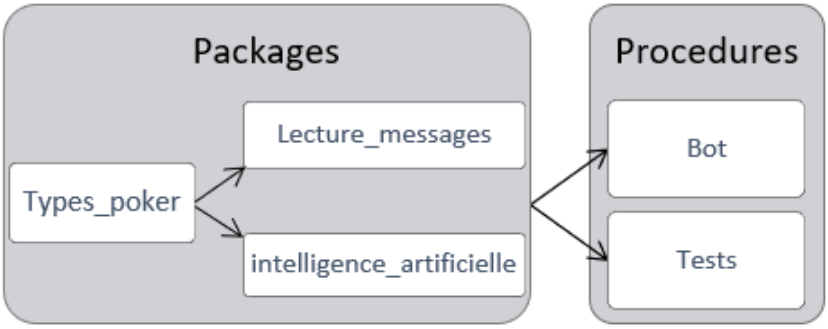


FIGURE 11 – Structure du projet

7 Définition de la représentation des éléments de jeu

Après la prise en main des différents éléments composant le dossier initial du projet nous identifions deux principaux types à définir :

- Le jeu et la table. Le type *T_jeu* comporte toutes les informations évoluant au cours de chaque main. Le type *T_table* est une simple liste de cartes.
- Les joueurs. Le type *T_joueur* comporte la main du bot, le nombre de jetons dont il dispose et si il est ou non *Button*.

Avant de pouvoir implémenter ces types, il nous faut une modélisation des cartes.

7.1 Modélisation des cartes

Il est essentiel que le type que nous allons définir ici soit le plus explicite possible pour faciliter sa manipulation par la suite. De plus il est intéressant de pouvoir classer les cartes par leur « puissance » pour définir la qualité d'une main par la suite. Nous optons alors pour un type article composé de deux types énumérés : la *couleur* et le *motif* de la carte.

Le *Launcher* représente les cartes par un couple de caractère en utilisant la représentation ci-dessous.

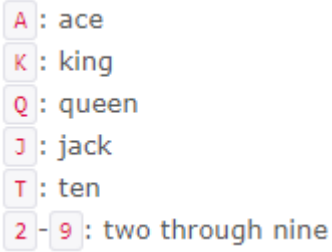


FIGURE 12 – Motifs à modéliser

FIGURE 13 – Couleurs à modéliser

Un roi de carreaux est alors lu (après décomposition de la ligne d'entrée) par le bot comme une string de 2 éléments « Ks ». Pour une conversion simple de ce format vers le type défini pour notre Bot, le plus simple est de définir des types coïncidant parfaitement à la représentation du *Launcher* :

```
1 Type T_Motif is (2,3,4,5,6,7,8,9,T,J,Q,K,A);
```

Cependant les entiers ne peuvent être utilisés tels quels dans l'énumération car d'un type différent des éléments suivants dans l'énumération. De plus, il est impossible de stocker les chiffres en type string car ce n'est pas un type discret (les seuls supportés dans l'énumération). Il ne reste qu'une représentation possible : en *Caractère*. Cependant cette représentation compliquera la suite du programme, nous définissons alors les types *T_Motif* et *T_Couleur* comme tels :

```
1 Type T_Couleur is (pique,carreau,coeur,trefle); --s,d,h,c
2 Type T_Motif is (deux,trois,quatre,cinque,six,sept,huit,neuf,dix,valet,reine,roi,as); --
    2,3,4,5,6,7,8,9,T,J,Q,K,A (par ordre de "puissance")
```

On notera bien la position relative de chaque motif au sein de l'énumération, cela nous permettra de comparer la puissance des motifs par la méthode *T_Motif'Pos*. Le type *T_Carte* est composé de deux variables motif et couleur du type adéquat. Une fonction de conversion est implémentée dans le package *Type_poker* afin de passer de la représentation en string du *Launcher* vers notre représentation en *T_Carte* :

```
Function Carte_Str2Carte (Carte_str : IN String) return T_Carte;
```

Pour vérifier la modélisation des cartes nous réalisons une procédure de test.

```
Donner une carte (ex:Ks pour le roi de pique) : Kd
Votre carte (image de T_Carte) est le ROI de CARREAU
Donner une deuxi me carte (ex : Ks pour le roi de pique) pour comparer : 8s
Le ROI de CARREAU est plus puissant que le HUIT de PIQUE
```

Listing 25 – Test du type *T_carte* et de la fonction *Carte_Str2carte*

7.2 Modélisation d'une partie, de la table et des joueurs

A présent que les cartes sont modélisées, nous créons une liste d'un maximum de 7 cartes. Le type *T_liste_cartes* est un *Type article* stockant à la fois une liste de 7 cartes et un entier représentatif du nombre de cartes actives (*i.e.* le nombre de cartes qui ont un sens). On place le maximum à 7 cartes en prévision du stockage des cartes d'un joueur et de la table dans une même liste. Cela facilitera la recherche de la meilleure combinaison de cartes d'un joueur. Ce type de liste de cartes sera utilisé pour modéliser la *table* et *la main d'un joueur*.

Afin de modéliser la partie et ses paramètres courants, nous créons le type *T_jeu*. C'est un *Type article* comportant les différentes variables d'une partie fournie par le *Launcher* : les valeurs de blind, la taille du pot et les valeurs de suivi et de pari minimales en *Natural* puis le numéro de la partie courante en *Positive*.

```
Type T_Jeu is record
   big_blind : Natural := 30;
   small_blind : Natural := 15;
   N_hand : Positive := 1;
   Pot : Natural := 0;
   Amount_to_call : Natural := 0;
   Min_bet : Natural := 0;
end record;

```

Nous modélisons les joueurs de la même façon que la partie, par le biais d'un *Type article*. Le type *T_joueur* comporte son nombre de jetons en *Natural*, sa main en *T_liste_cartes* et sa position dans le tour, si il est *Button* ou non sous forme d'un *Boolean*.

```
Type T_joueur is record
   Stack : Natural;
   Main : T_liste_cartes;
   Button : Boolean := False;
end record;

```

```
Quantite de jeton initial pour chaque joueur puis dans le pot :
1000 200
Quantite de jeton apres placement des blind ; Moi :  970 ; autre :  985
Quantite de jeton place dans le pot par moi (positif ou negatif i.e. gain ou perte):
42
Quantite final de jeton au tour  1 ;  Pot :  242 ; Moi :  928 ; Autre :  985
```

Listing 26 – Sortie console du test des types *T_joueur* et *T_jeu*

8 Interaction avec le Launcher

Maintenant que nous avons une structure de stockage des informations, il faut remplir cette base de données. Nous allons réaliser le package *lecture_messages* utile à la lecture et la mise à jour des informations fournies par le *Launcher*.

8.1 Représentation des messages

Dans un premier temps il faut une structure de stockage du message lu par le *Bot*. Il est nécessaire de le stocker sous forme d'une *String* capable de stocker tous les messages (en taille) à laquelle on ajoute la taille du message au sein d'un *type article* nommé *T_chaine*.

Par la suite nous modélisons les messages par une liste de mots. Il est alors bien plus aisé d'accéder directement au n^{ème} mot du message qui comporte toujours moins de 20 mots.

```
Type T_tab_mot is Array (1..10) of T_chaine(20);
Type T_message is record
  total : T_chaine(100);
  decompose : T_tab_mot;
  nbr_mot : Integer := 0;
end record;
```

Listing 27 – Définition de *T_message*

En particulier l'identification des mots clefs d'un message doit être aisée. La définition de *types énumérés* s'impose.

```
Type T_mots_clefs_1 is (action, update_game, update_hand);
Type T_mots_clefs_2 is (button, stack, small_blind, big_blind, hand, table, pot, amount_to_call,
    min_bet, move, win);
Type T_mots_clefs_3 is (self, other);
Type T_moves is (fold, check, call, bet);
```

Listing 28 – Définitions des mots clefs

On notera l'absence du mot clef *Settings* que nous ignorons pour le moment. En effet, les informations qu'il apporte sont pour le moment superflus dans l'élaboration de notre *Bot*.

Les différents mots clefs sont organisés selon les arborescences de construction de messages suivantes :

Bien que les messages auraient pu être stockés comme décomposition de ces types énumérés, nous préférons les garder en liste de *T_chaine*. Cela permet de garder un type homogène pour chaque mot d'un message tout en faisant la distinction du rang des mots lors de leur analyses par la suite.

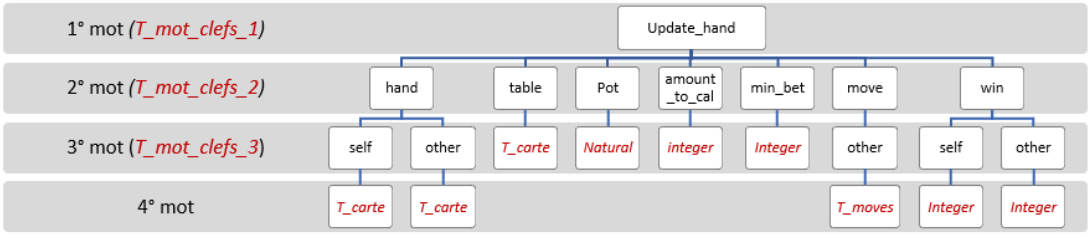


FIGURE 14 – Structure de la décomposition des messages de main

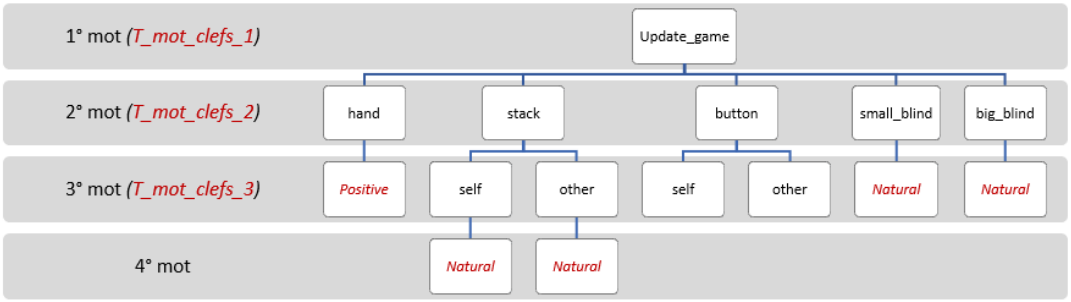


FIGURE 15 – Structure de la décomposition des messages de jeu

8.2 Lecture et interprétation des messages

Le package *lecture_ messages* est complété par les *fonctions* et *procedures* utiles à la décomposition du message et la mise à jour des variables. Nous réalisons la procédure *decompose_ message*. Elle permet de compléter la liste de mots clefs d'une variable du type *T_ message* dont le message entier y est déjà stocké (accéder par *message.total*).

```
1 procedure decompose_message(message : IN OUT T_message);
```

A la lecture des figures 4 et 5 nous remarquons que le dernier mot d'un message correspond toujours à l'information à stocker. Dans la plupart des cas, cette information peut être facilement convertie dans le type adéquat par la méthode *'value*. Cependant dans le cas d'une énumération de cartes par le *Launcher* il nous faut créer une fonction de conversion *lecture_ cartes* qui prend en argument une liste de cartes de taille indéterminée au format du *Launcher* : "Ks,Ad,4h". Elle fournit la liste de cartes correspondante au format de notre *Bot*, *T_ liste_ cartes*.

```
1 function lecture_cartes(cartes_str : IN T_chaine) return T_liste_cartes;
```

A présent il ne reste plus qu'à réaliser les procédures *lecture_ update_ game* et *lecture_ update_ hand* qui stockent les informations fournies par le *Launcher* à la bonne place de notre structure de données. Elle est composée dans notre *Bot* des variables ci-contre.

```
1 info_partie : T_jeu;
2 Self : T_joueur;
3 Other : T_joueur;
4 Table : T_liste_cartes;
5
```

Listing 29 – Initialisation de la structure de données du Bot

Les procédures de lecture prennent donc ces variables en argument en plus, évidemment, du message lu et décomposé. L'implémentation des procédures est facilitée par la lecture des figures 4 et 5. En effet il suffit de suivre

une des lignes conditionnelles qui descend jusqu'à la valeur à stocker.

```
case T_mots_clefs_2'value(message.decompose(2).line(1..message.decompose(2).line_length)) is -- Si
     le deuxieme mot du message est ...
    when T_mots_clefs_2'Val(4) => -- ...hand, alors ...
        case T_mots_clefs_3'value(message.decompose(3).line(1..message.decompose(3).line_length))
    is -- ... remplir les cartes du joueur ...
            when T_mots_clefs_3'Val(0) => -- ...self
                J_Self.Main := lecture_cartes(message.decompose(4));
            ...
        end case;
    ...
end case;
```

Listing 30 – Exemple d'implémentation de la première ligne de la Figure 4

```
procedure lecture_update_game(
            message : IN T_message;
            Table : OUT T_liste_cartes;
            info_partie : Out T_jeu;
            J_Self : Out T_joueur;
            J_Other : Out T_joueur);
```

```
procedure lecture_update_hand(
            message : IN T_message;
            Table : OUT T_liste_cartes;
            info_partie : Out T_jeu;
            J_Self : Out T_joueur;
            J_Other : Out T_joueur);
```

Lecture_update_game et *lecture_update_hand* analysent les messages recus seulement à partir du second mot. En effet, il est nécessaire d'analyser le premier mot du message dans l'implémentation du *Bot* afin de vérifier si nous sommes dans le cas d'une *Action*. De ce fait il est inutile d'inclure la lecture du premier mot dans une unique procédure regroupant *Lecture_update_game* et *lecture_update_hand*. Nous préférons garder ces deux *procédures* indépendantes. La structure du *Bot* en sera simplifiée, en mettant clairement en évidence sa réaction au différents mots clefs *T_mots_clefs_1*.

Pour exclure les cas non pris en compte dans la lecture des messages, tels que les *Settings*, nous réalisons la fonction *est_mots_clefs_1*. Cette *fonction* renvoie un boolean nous affirmons si nous sommes dans le cas de la modélisation des messages afin d'éviter des problèmes lors de l'interprétation du 1er mot lu.

```
function est_mots_clefs_1(mot_clef : IN T_chaine) return Boolean;
```

8.3 Vérification du package

Afin d'assurer le bon fonctionnement du *Bot* nous testons toutes les fonctions récemment définies dans le package *lecture_message*. Pour se faire, nous réalisons une première *procédure* testant *decompose_message*, *est_mots_clefs_1* et *lecture_cartes*.

```
Inserer un message du type du Launcher (mots separes par espace) :
Blabla bla blablabla blabla
4 mots :
Blabla
bla
blablabla
blabla

Un mot qui appartient ou non au type enumere T_mots_clefs_1 :
Action
C'est bien un mot clef.

```

```
Inserer une liste de carte du type Launcher (Ks,Qd,4s,...) :
4d,Kh,Qs,Ah

Carte 1 : QUATRE de CARREAU
Carte 2 : ROI de COEUR
Carte 3 : REINE de PIQUE
Carte 4 : AS de COEUR
```

Listing 31 – Sortie console des 1er tests de *lecture_messages*

Apres avoir réalisé plusieurs tests à l'aide de cette *procedure*, nous vérifions *Lecture_update_game* et *lecture_update_hand*. Au sein d'une première *procedure* de test de *Lecture_update_game* nous simulons la lecture de messages du type du *Launcher*. Le test consiste en la comparaison de l'état de la base de données avant et après la lecture des messages ci-dessous. Cette liste de messages est exhaustive dans le champ d'application de la *procédure*. En effet les mots clefs *Win* et *Move* ne sont pas traités. A l'instar de *Settings*, les informations suivants ces mots ne nous paraissent pas intéressantes pour le moment. Cependant la structure du programme permet d'inclure leur traitements si besoin.

La même procédure de test est appliquée à *Lecture_update_hand*. On obtient de même des résultats cohérents à nos attentes.

```
"update_game button self"
"update_game stack self 4000"
"update_game stack other 10"
"update_game hand 2"
"update_game small_blind 300"
"update_game big_blind 600"

```

Listing 32 – Messages de test de *Lecture_update_game*

```
"update_hand hand self Qd,Tc"
"update_hand hand other 2d,7c"
"update_hand table 4s,Td,Ts"
"update_hand table 5s,Td,Ts,3c"
"update_hand pot 120"
"update_hand amount_to_call 10"
"update_hand min_bet 60"

```

Listing 33 – Messages de test de *Lecture_update_hand*

```
Etats joueurs actuels :
Self :  0 ; FALSE
Other :  0 ; FALSE
Etats joueurs apres lecture :
Self :  4000 ; TRUE
Other :  10 ; FALSE

Etat partie actuelles :
Big blind : 30 ; Small blind : 15 ;
Numero de main : 1
Etat partie apres lecture :
Big blind : 600 ; Small blind : 300 ;
Numero de main : 2


```

Listing 34 – Procedure de test de *Lecture_update_game*

```
Cartes des joueurs :
Self : REINE de CARREAU et DIX de TREFLE
Other : DEUX de CARREAU et SEPT de TREFLE

Cartes sur la table :
QUATRE de PIQUE
DIX de CARREAU
DIX de PIQUE

Cartes sur la table (ajout carte) :
CINQUE de PIQUE
DIX de CARREAU
DIX de PIQUE
TROIS de TREFLE

Le pot est de 120
La valeur pour suivre est de 10
La valeur minimale pour remiser est de 60

```

Listing 35 – Procedure de test de *Lecture_update_hand*

9 L'intelligence artificielle

A ce niveau du projet, notre *Bot* est capable de lire les messages reçus par le *Launcher* et de mettre à jour les informations de la partie. Il ne lui manque plus que l'"intelligence" nécessaire pour agir. Dans ce troisième package *Intelligence_ Artificielle* nous établissons la stratégie de notre *Bot*. Pour être performant, notre *Bot* doit pouvoir détecter la combinaison de cartes qu'il peut former à partir de sa main et de la table. En fonction de la puissance de sa combinaison, il réagira aux actions du *Bot adverse* tout en respectant les règles !

9.1 Modélisation des combinaisons

Nous avons plusieurs combinaisons à modéliser : hauteur, paire, double paire, brelan, quinte, flush, full, carre, quinte flush, quinte flush royale (combinaisons possibles extraites des règles du No limit Texas Hold'em). Chacune de ces combinaisons est du type énuméré *T_ combinaison_ nom*.

```
1 Type T_combinaison_nom is (hauteur, paire, double_paire, brelan, quinte, flush, full, carre,
      quinte_flush);
```

Listing 36 – Noms des *combinaisons*

Ensuite nous devons modéliser la « valeur » de ces combinaisons (prise en compte des motifs qui composent ces combinaisons si cela est nécessaire). Nous avons donc décidé de créer un type article *T_ combinaison* dont les composantes sont le *nom de cette combinaison* (composante de type T_combinaison_nom) et l'unique *motif* associé à la combinaison dans les cas suivants : *hauteur*, *paire*, *brelan*, *carré*, *flush* et *quinte*. Pour la *flush* et la *quinte* la valeur de ce motif est le motif le plus fort de la combinaison de cartes. On s'attachera à détécter ces motifs dans les procédures de détection de *flush* et de *quinte*.

Cependant, dans certain cas, nous avons besoin de deux motifs pour définir la « valeur » d'une combinaison. Nous utilisons alors la dernière composante de ce type article (*motif_ 2*) qui définit le deuxième motif de notre combinaison. Cette composante s'avère utile dans les cas suivants : *double paire, full*.

Bien que cette solution soit peu élégante, elle simplifie grandement la modélisation des combinaisons. En effet, il est possible de réaliser un *Type polymorphe* distinguant les combinaisons nécessitant un ou deux motifs. Cependant cette méthode complexifie l'initialisation des combinaisons et leurs comparaisons.

```
1 Type T_combinaison is record
2    nom : T_combinaison_nom;
3    motif_1 : T_Motif := T_Motif'Val(0);
4    motif_2 : T_Motif := T_Motif'Val(0); -- Facultatif
5 end record;
```

Listing 37 – Type *T_ combinaison*

Remarque : la quinte flush royale est un cas particulier de quinte flush ou le motif le plus important serait l'As. Si on détecte une suite et une flush dont la carte la plus haute est l'AS : il s'agit d'une quinte flush royale !

9.2 Analyse d'une combinaisons

Maintenant que l'on a défini chaque combinaison possible, il faut analyser l'ensemble des cartes sur la table et dans notre main. L'objectif étant de déterminer la meilleure combinaison que l'on peut former à partir de cette liste de carte. Cette analyse se fait à chaque ajout de carte sur la table au cours de la partie. Ainsi l'analyse doit pouvoir s'effectuer sur les 2 cartes de la main puis sur 5,6 ou 7 cartes après sortie des cartes sur la table. La fonction

réalisant cet objectif se nomme *detecte_meilleure_combinaison*. Elle prend en argument la liste de cartes que l'on possède (main et table) et retourne la meilleure combinaison.

```
function detecte_meilleure_combinaison(cartes_in : IN T_liste_cartes) return T_combinaison;
```

Cette fonction utilise d'autres primitives :

- *compte_cartes* donne le nombre d'occurrences de chaque motif au sein d'une liste de cartes. La position dans la liste correspondant à la position dans le type énuméré

  T_combinaison_nom

- *detecte_meilleure_motifs* permet de connaître le motif le plus présent dans la liste de carte, s'il y a égalité, c'est le motif le plus fort.

- *detecte_couleur* détermine s'il y a 5 cartes de la même couleur dans une liste de carte et précise la meilleure carte.

- *detecte_suite* détermine s'il y a une suite dans une liste de carte et précise la meilleure carte.

De plus, l'utilisation de *detecte_meilleure_combinaison* nécessite la création d'une unique liste de carte composée de la *Table* et du *Bot*. On implémente alors une fonction de concaténation des cartes d'un joueur et de la table.

```
function compte_cartes(cartes_in: IN T_liste_cartes) return T_liste_nombre_motif;
procedure detecte_meilleure_motifs(liste_motifs : IN T_liste_nombre_motif ; nbr_motif_max : OUT
    T_nbr_motif_max ; motif_max : OUT T_Motif);
procedure detecte_couleur(cartes_in : IN T_liste_cartes ; motif_max : OUT T_Motif; bool_flush :
    OUT Boolean);
procedure detecte_suite(cartes_in : IN T_liste_cartes ; motif_max : OUT T_Motif; bool_quinte : OUT
     Boolean);

function concatenation_cartes(joueur : IN T_joueur ; table : IN T_liste_cartes) return
    T_liste_cartes;
function detecte_meilleure_combinaison(cartes_in : IN T_liste_cartes) return T_combinaison;
```

Listing 38 – Primitives de *Intelligence_Artificielle*

La recherche de la meilleure combinaison débute par l'obtention de la liste retournée par la fonction *compte_cartes*. Grâce à cette liste, on peut déterminer combien de fois apparaît le meilleur motif. Plusieurs cas sont alors à considérer selon la valeur de ce *nbr_motif_max*.

```
Determiner le motif_max le plus represente dans la liste (nbr_motif_max)
Dans le cas ou nbr_motif_max vaut
   1 Alors
      Combinaison_max est Hauteur de motif_max
   2 Alors
      Combinaison_max est Paire de motif_max
      Eliminer la paire de la liste de carte
      Determiner le motif_max le plus represente dans la liste (nbr_motif_max)
      Si nbr_motif_max > 1 Alors
         Combinaison_max est Double Paire des motif_max precedents
   3 Alors
      Combinaison_max est Brelan de motif_max
      Eliminer le Brelan de la liste de carte
```

```
14      Determiner le motif_max le plus represente dans la liste (nbr_motif_max)
15      Si nbr_motif_max > 1 Alors
16         Combinaison_max est Full des motif_max precedents
17   4 Alors
18      Combinaison_max est Carre de motif_max
```

Listing 39 – Algorithme de recherche des combinaisons independamment de la couleur et des suites possibles

Nous devons à présent considérer la flush et la quinte. Pour chacune de ces combinaisons, une procédure permet de la détecter.

On commence par la détection de la *Flush*, cette combinaison étant plus importante que les précédentes, si elle existe elle prendra la place de la combinaison obtenue précédemment. La procédure *detecte_couleur* (flush) compare la couleur de chacune des 3 premières cartes de la liste à toutes les suivantes. Si on détecte 4 autres cartes de la couleur d'une des trois premières alors il y a bien un *Flush*. De plus on attribue à motif_max le motif maximal dans la flush.

De la même façon, si la *Quinte* est détecter par la suite, elle remplace la combinaison obtenue précédemment. On fera attention au cas de la *quinte flush* si on détecte à la fois une *quinte* et une *flush*. La procédure *detecte_suite* repose sur l'analyse de la non nullité d'éléments se suivant dans la liste d'occurrences de motifs donnés par *compte_cartes*.

Afin de vérifier notre *Procedure* de détection de combinaisons, nous réalisons un test mettant en avant chaque étape de *detecte_meilleure_combinaison*.

```
1  Nombre de motifs : 5
2  DEUX : 0
3  TROIS : 0
4  QUATRE : 0
5  CINQUE : 0
6  SIX : 0
7  SEPT : 0
8  HUIT : 0
9  NEUF : 0
10 DIX : 1
11 VALET : 1
12 REINE : 1
13 ROI : 1
14 AS : 1
15
16 Le motifs present le plus de fois, etant le plus haut est : AS
17 Il est present 1 fois.
18 On detecte un flush de carte la plus haute : AS
19 On detecte une suite de carte la plus haute : AS
20
21 La meilleure combinaison est donc :
22 FLUSH de AS et (si combinaiason a deux motifs) DEUX
```

Listing 40 – Résultat du test *Analyse_main* pour une Quinte flush royale

On réalise ce test pour différentes combinaisons de cartes pour s'assurer que toutes les combinaisons sont détectées. En particulier, nous avons remarqué qu'une première version de *Detecte_suite* ne détectait pas la suite dont la première carte est l'As.

9.3 Stratégie

Cette partie traite de la stratégie de notre bot pour gagner face au *JavaBot* inclus dans le projet. Pour avoir un maximum de chance de gagner, notre bot va déterminer à chaque ajout de carte sur la table la meilleure combinaison

qu'il possède puis il va la comparer à toutes les combinaisons possibles que peut avoir le bot adverse. On réalise cette comparaison en générant l'ensemble des doublets de cartes qui peuvent constitués la main du *JavaBot*. On élimine évidemment de ces possibilités toutes les cartes présentes dans la main de notre Bot et sur la table. Cette méthode d'évaluation de main nécessite d'appliquer notre fonction *Detecte_meilleure_combinaison* et comparer entre 2450 et 1980 mains de cartes adverse. La limite temporelle d'action devrait être suffisante pour exécuter ces calculs.

```
1 function pourcentage_gagne(Table : IN T_liste_cartes ; Self : IN T_joueur) return Natural;
```

Pourcentage_gagne donne un résultat explicite (pourcentage entier) de la chance du bot de gagner avec sa combinaison. En effet, le résultat sera le quotient du nombre de fois où le bot gagne avec sa meilleure combinaison face au bot adverse sur le nombre de combinaisons possibles (multiplié par cent pour avoir un pourcentage). En fonction de ce pourcentage nous allons définir le comportement du *Bot*. Les règles peuvent se comprendre facilement sur cet axe de valeur de pourcentage_gagne :

Remarque : ces règles ont été définies de manière arbitraire (simple bon sens) et pourront être ajustées par la suite.

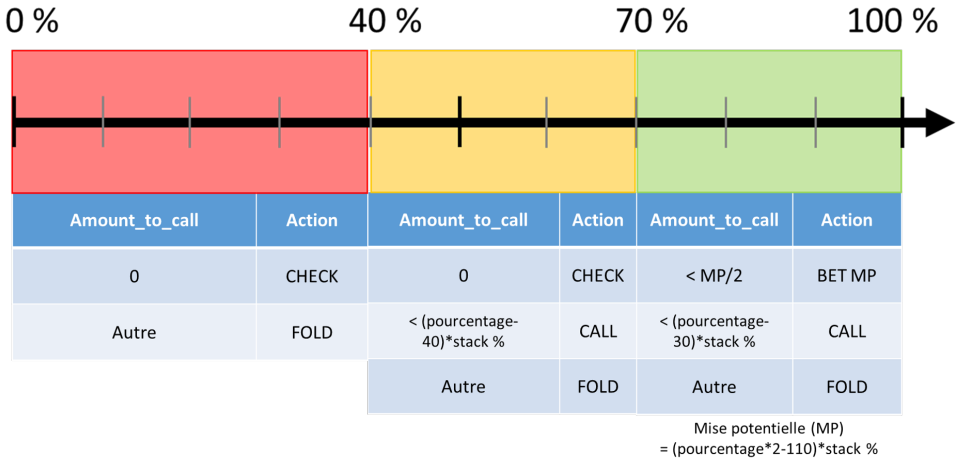


FIGURE 16 – Schéma des règles pour la stratégie

Ainsi on comprend bien ici que les règles à appliquer sont différentes selon la valeur de pourcentage_gagne et la valeur du *amount_ to_ call*. Les différents cas étant présentés dans le tableau en dessous de l'axe.

- Dans la zone allant de 0 à 40 % (faible chance de gagner) :

  Si Amount_to_call vaut 0 alors on check sinon on se couche.

- Dans la zone allant de 40 à 70 % (chance de gagner moyenne) :

  On effectue les mêmes actions sauf qu'en plus on rajoute le cas où l'Amount_to_call serait inférieur à la valeur ((pourcentage_gagne-40)/100)*stack_de_son_bot. Dans ce cas (amount_to_call pas trop important par rapport au stack et fonction du pourcentage de gagne), notre bot suit.

- Dans la zone allant de 70 à 100 % (forte chance de gagner) :

  On suit pour une valeur du amount_to_call plus grande : ici la valeur à ne pas dépasser est ((pourcentage_gagne-30)/100)*stack_de_son_bot. On peut aussi relancer. On relance d'une mise qui vaut ((pourcentage_gagne*2-110)/100)*stack_de_son_bot que l'on nommera mise_potentielle (MP). On relance évidemment que si cela est possible et intéressant : nous avons choisi comme condition d'avoir une mise potentielle deux fois supérieure à amount_to_call.

Toutes ces valeurs, choisi arbitrairement respecte une loi linéaire fonction du pourcentage. Par exemple, la mise potentielle a été initialement définie par des bornes et une pente (cf. schéma) d'où son expression. Nous voulions utiliser entre 30 % et 90 % de notre stack en fonction du pourcentage de gagne que l'on avait pour remiser.

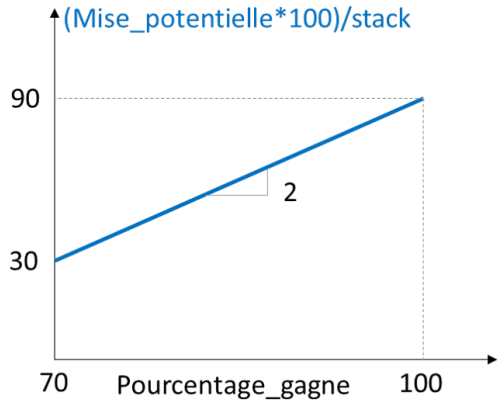


FIGURE 17 – Schéma de définition de la mise potentielle

10 Conclusion

Notre *Bot* répond à présent aux exigences du projet. Il est capable de jouer selon les règles établies par le *Launcher*. On finalise le projet en vérifiant l'efficacité du *Bot* et réfléchir à de potentielles améliorations.

10.1 Validation du Bot et évolutions possibles

Bien que le *Launcher* ne relève aucune erreur de jeu, on remarque des incohérences dans la stratégie du *Bot* : miser 98% de la quantité de jetons avec une paire de 3 ... Ces incohérences se sont avérées être liées à l'oubli de la réinitialisation des cartes sur la table de jeu en fin de partie. Pour résoudre ce problème nous ajoutons la réinitialisation de la table (*Table.nbr=0*) au sein de *lecture_update_game* lors de la lecture du second mot clef *"hand"* qui correspond à la distribution de nouvelles cartes.

A présent, nous sommes satisfait du comportement de notre *Bot*. On détecte cependant quelques dépassements de la durée allouée à la réponse de *Action*. Afin d'évaluer l'efficacité du programme nous réalisons un script implémenté en *Python* qui détermine le pourcentage d'erreur et de gagne sur 500 parties.

```
import os

nombre_parties = 500
i=0
count_win  = 0
count_fail = 0
for i in range (nombre_parties) :
    i+=1
    os.system("launcher.exe > log.txt")
    f = open("log.txt")
    Error_line = f.readlines()[-3]
    Win_line = f.readlines()[4]
    if Error_line[0:23] == "raised CONSTRAINT_ERROR" :
        count_fail += 1
    if Win_line.strip() == "winner: Adabot" :
        count_win += 1
    f.close()
print("Pourcentage de victoire final : "+str(100*count_win/nombre_parties)+" %")
print("Pourcentage d'erreur final : "+str(100*count_fail/nombre_parties)+" %")
```

Listing 41 – Script de test du Bot

Notre *Bot* est vainqueur dans 73% des cas. Cependant on observe un taux d'erreur de 8%. Ce taux d'erreur est relativement faible et permet tout de même d'atteindre un bon taux de réussite, nous gardons le programme dans cet état. De plus Brice CHARDIN nous a confirmé que le temps de réponse du *Bot* ne sera pas évalué pour le moment et que le paramètre *Time_bank* sera augmenté pour les tests.

Le principal défaut de notre programme est évidemment son temps de réponse. Pour l'optimiser nous pourrons utiliser la fonction *Clock* du package *Ada.calendar*. Celle-ci nous permettra d'adapter la quantité de simulation de cartes en fonction du temps restant pour répondre. De plus, une amélioration efficace serait de prendre en compte les potentielles cartes prêtes à sortir sur la table en plus de la main de l'adversaire pour évaluer justement notre main.

Pour le moment nous n'avons pas étudié le comportement du bot adverse. Il serait intéressant de créer un package capable d'évaluer la stratégie adverse. En particulier, on peut déterminer son seuil de mise, sa capacité à se coucher, son taux de suivi de mise ...

10.2 Acquis pédagogiques

Ce projet nous a prouvé la nécessité de structurer correctement le programme en cohérence avec les différents objectifs dégagés du problème initial. Cette décomposition en sous-problème nous a permis de travailler efficacement à deux, chacun s'attaquant à une partie différente du programme.

Ce travail en parallèle nous a imposé de structurer correctement les fichiers de *spécifications* et d'ajouter une notice aux éléments le nécessitant. En particulier nous précisions au cours de l'implémentation des packages lorsque

les fonctions et procédures sont testées pour éviter d'en construire d'autres reposant sur une base bancale.

L'implémentation de *fichiers tests* a finalement engendré un gain de temps. La phase de débugage du Bot en a été grandement facilitée. En effet, il est rapidement possible de déceler une éventuelle erreur.

Finalement, il est intéressant d'auto-évaluer la pertinance de nos choix de programation lors de la rédaction de ce compte rendu. En particulier nous avons implémenté une fonction de tri des cartes au sein d'une liste que nous n'avons finalement pas utilisé.