Compte rendu Projet IFB belote coinché

##### Introduction :

La belote coinché constitue une version stratégique de la belote : ce jeu de cartes populaire comprend de nombreuses stratégies ce qui donne un grand nombre de possibilités du mode de jeu. Sa principale différence avec la belote se situe au niveau des contrats où toutes les cartes sont disponibles d’un coup.

Le but de notre projet était de rendre un jeu où l’on puisse jouer contre des ordinateurs, qui soit réaliste en termes de jeu, mais aussi et surtout joli et agréable à utiliser. Pour ce faire on a séquencé notre projet en 3 parties. Dans un premier temps, on a mis en place la base du jeu, c’est-à-dire un jeu avec peu de fonctionnalités, peu évolué mais fonctionnel et complètement débogué. Ensuite, nous avons fait évoluer notre jeu en rajoutant tous les éléments qui font de ce jeu une vraie coinche, mais aussi une « intelligence » plus avancée des ordinateurs lors des phases de jeu et de contrats, afin de rendre le jeu plus réaliste et amusant. Tout à la fin, dans notre souhait de rendre le jeu à la fois joli et ludique, on a mis en place une interface graphique.

Table des matières

[Compte rendu Projet IFB belote coinché 1](#_Toc43200645)

[I. Fonctionnement basique du jeu 3](#_Toc43200646)

[1. Choix des variables et du jeu. 3](#_Toc43200647)

[2. Fonctionnement général du jeu 3](#_Toc43200648)

[a. Mélanger le paquet de cartes, distribution et rangement par ordre croissant d’un jeu 3](#_Toc43200649)

[b. Les contrats 4](#_Toc43200650)

[c. Lancement d’une manche 4](#_Toc43200651)

[d. Lancement d’un pli 4](#_Toc43200652)

[e. Poser les cartes pour chaque joueur 4](#_Toc43200653)

[f. Affichage des possibilités des cartes pour chaque joueur (ordinateur ou réel) 5](#_Toc43200654)

[g. Choix des humains 6](#_Toc43200655)

[h. Choix des ordinateurs 6](#_Toc43200656)

[i. Actualisation de la carte maitresse dans le pli 7](#_Toc43200657)

[j. Calcul des points et attribution des points 7](#_Toc43200658)

[k. La vérification des contrats 8](#_Toc43200659)

[II. Amélioration du mode de jeu 9](#_Toc43200660)

[1. Calcul des annonces 9](#_Toc43200661)

[a. Calcul des annonces de types suites par couleurs 9](#_Toc43200662)

[b. Calcul des annonces de type carrés 9](#_Toc43200663)

[c. Calcul de l’annonce maximum du joueur 9](#_Toc43200664)

[2. Amélioration et perfectionnement des contrats 10](#_Toc43200665)

[3. Amélioration et perfectionnement de l’intelligence des ordinateurs 11](#_Toc43200666)

[4. Mise en place de la belote-rebelote et du dix de der 12](#_Toc43200667)

[5. Enregistrement des scores 13](#_Toc43200668)

[III. Interface graphique 13](#_Toc43200669)

[1. Fonctionnement de la partie affichage de la SDL 14](#_Toc43200670)

[2. Fonctionnement des événements en SDL 14](#_Toc43200671)

[3. Mise en place de ce système dans la coinche 14](#_Toc43200672)

[4. Les fonctionnalités permises par cette bibliothèque 15](#_Toc43200673)

[IV. Conclusion : 16](#_Toc43200674)

# Fonctionnement basique du jeu

## Choix des variables et du jeu.

* + Concernant la programmation des cartes, nous avons choisi dès le début du projet d’utiliser des chaines de caractères pour plusieurs raisons :
    - * Pouvoir comparer facilement les cartes de chacun des joueurs avec des fonctions de comparaison (strcmp,…).
      * Permet le codage de la carte sur une seule chaine de caractères (à la fois la valeur de la carte mais aussi la couleur).
  + Ensuite, nous avons décidé d’utiliser 3 structures pour le fonctionnement du jeu :
    - * La structure Joueur contient les informations liées à chaque joueurs (N,S,E,W) telles que ses cartes, son contrat …
      * La structure Jeu, stocke les informations liées au déroulement de la partie telles que le type de contrat en cours, la carte maitresse du pli…, mais aussi des informations liées à l’affichage : detenteur\_belote\_rebelote …
      * Une structure Equipe va stocker sur différentes informations sur inhérentes aux deux équipes (score de la manche de l’équipe, score total de l’équipe…)
  + Cela nous permet lors d’un appel de fonction de simplifier l’envoi et la réception de variables. En effet, il y a eu tellement de variables que si on devait les exploiter une par une, le codage aurait été infaisable. Cependant cela nous a pris pas mal de temps au début pour mettre en place les différentes variables à mettre dans chaque structure, et tout au long de l’avancement du projet, il y a eu énormément de modifications.

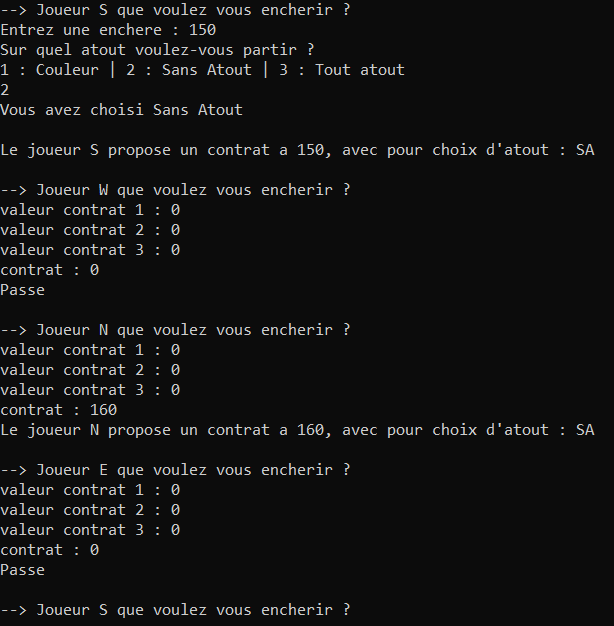
## Fonctionnement général du jeu

### Mélanger le paquet de cartes, distribution et rangement par ordre croissant d’un jeu

* + Cette partie est fondamentale pour le fonctionnement de base de notre jeu. Nous avons donc fait trois fonctions pour répondre à ces 3 besoins :
    - Une fonction qui va mélanger le paquet de cartes par interversion de cartes de manière aléatoire.
    - Une fonction de rangement par ordre croissant des cartes et par couleur : d’abord les trèfles rangés par ordre croissant, puis les cœurs, pics, carreaux. Cet ordre va nous faciliter de nombreuses choses dans beaucoup de fonctions. Par exemple, pour trouver le maximum on regarde la carte la plus à droite.
    - Une fonction qui va distribuer au 4 joueurs 8 cartes au début de la manche.
  + On a fait de nombreux ajustement notamment pour la fonction qui mélange le paquet pour à la fois avoir un jeu bien mélangé mais aussi amusant, car plus on mélangeait, moins il y avait d’annonce et plus les jeux devenait homogènes, on s’est donc fixé à 15 interversions.

### Les contrats

* + Une fois la distribution finie, c’est la phase de contrat qui s’entame, pour commencer on a décidé de coder la partie contrat de manière simple mais fonctionnelle, on a posé les choix pour le joueur réel, et pour les ordinateurs on est parti sur un programme très basique.

****

* + On est parti du principe que l’ordinateur n’enchérirait qu’à la couleur dans un premier temps, et que pour son contrat à la couleur, il le déterminerait grâce au nombre de carte qu’il possède à l’atout. Pour cela on compte le nombre de cartes dans chaque couleur, puis on retient la couleur ayant le plus de cartes, puis en fonction du nombre de cartes, il va faire une enchère.
  + C’est un système basique, qui fonctionnait bien, mais qui dans la réalité est complètement dépassé si on veut faire jouer les ordinateurs de manière réaliste.

Exemple de la phase de contrat de notre jeu classique, chaque joueur propose un contrat ou passe (on calcul les contrats de types TA/SA/couleur puis on garde le meilleur).

### Lancement d’une manche

* On utilisera la fonction lancement\_manche qui va lancer la manche et gérer son déroulement.
* Cette fonction se décompose en deux parties : tout d’abord, elle permet l’initialisation de la manche, et ensuite gère l’exécution des 8 plis.
  + Pour cela elle appelle lancement\_pli, c’est la fonction qui va gérer l’exécution complète d’un pli**.**

### Lancement d’un pli

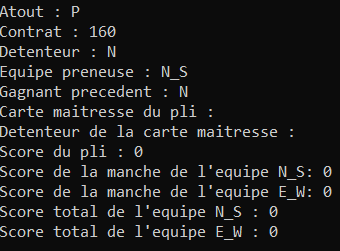
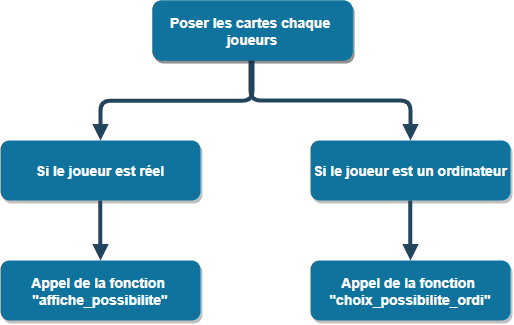
* + ****Elle est subdivisée en plusieurs parties à savoir : gérer l’ordre de jeu, faire jouer les joueurs à l’aide de la fonction « pose », centraliser les différents événements liés au fonctionnement de la coinche (annonces, belote-rebelote…) et gérer l’attribution des points en fin de pli.

Tableau récapitulatif de la partie pour chaque pli

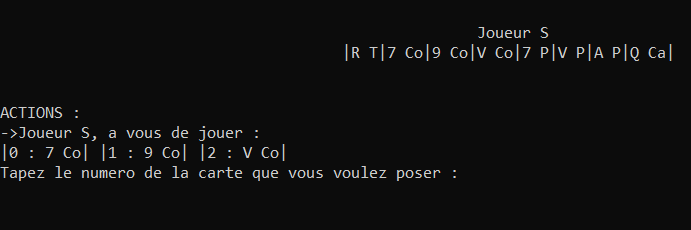
### Poser les cartes pour chaque joueur

* + Cette fonction est là pour gérer globalement la pose d’une carte. Cela va comprendre l’affichage des possibilités, la récupération du choix, la mise à jour du paquet du joueur, la mise de la carte sur la table puis l’actualisation de la carte maitresse du pli.

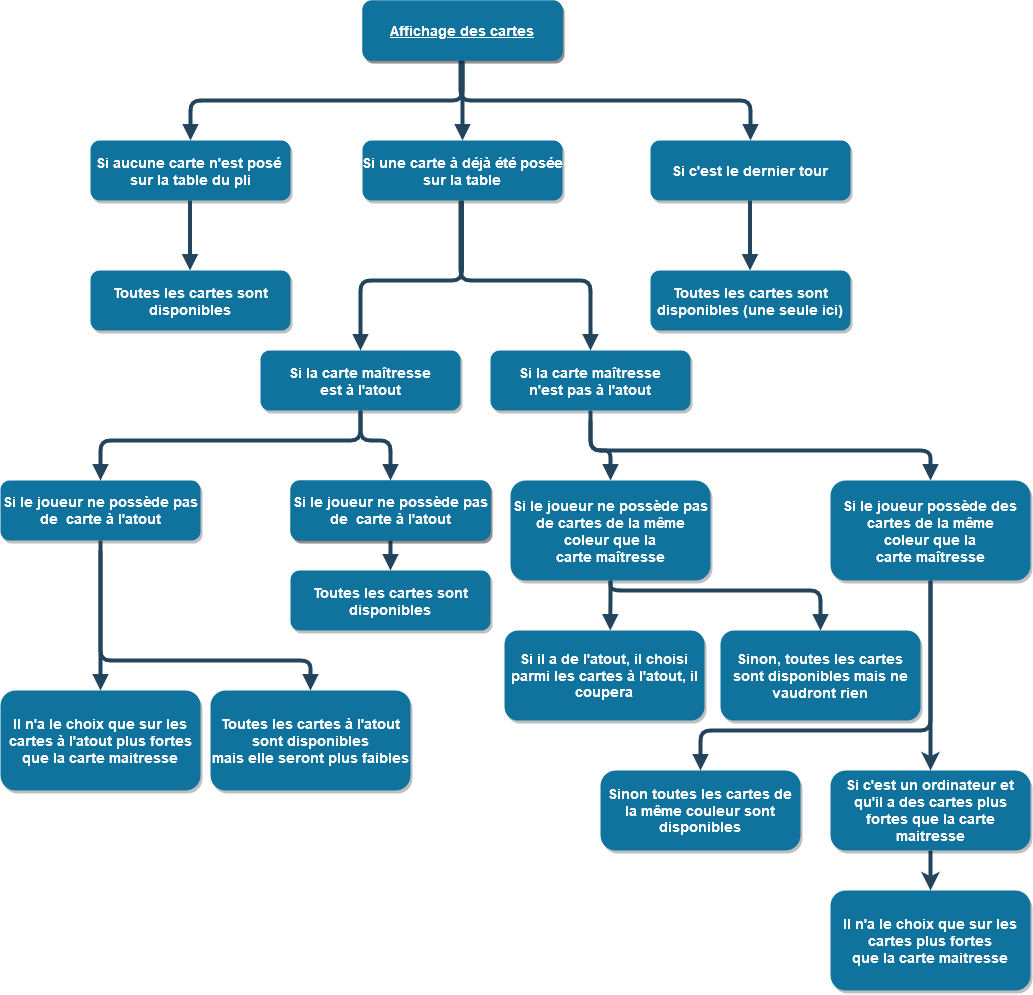
****

* + De plus si c’est le premier tour de la partie, elle va appeler la fonction qui va permettre le calcul des annonces pour chaque joueur. Et dans la fonction lancement manche, une fois le premier pli fini, on va pouvoir décider qui possède la plus grosse annonce en fonction des points de celles-ci.

### Affichage des possibilités des cartes pour chaque joueur (ordinateur ou réel)

* + On a décidé de séparer en deux les fonctions d’affichage des possibilités au lieu de faire une boucle if, pour éviter d’avoir à gérer une trop grosse fonction et pour avoir un code plus clair.
  + ****Leurs fonctionnements sont presque les mêmes, cependant pour la version ordinateur, il y aura une différence dans le cas où la carte maitresse est à la couleur, au lieu d’afficher toutes les cartes à la couleur, on va tester si le joueur possède des cartes de la même couleur mais plus fortes et les lui afficher.

Exemple d’affichage des possibilités des cartes pour le joueurs S

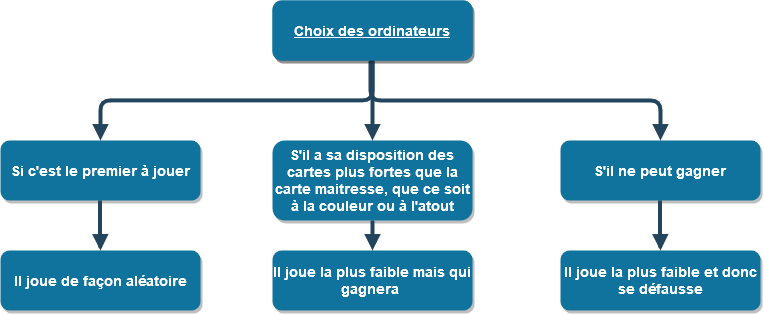
* + La partie commune est la mise en place d’un tableau contenant seulement les cartes que le joueur peut jouer, pour cela nous avons procédé à une dissociation des cas possible en fonction des jeux de chacun, de l’atout de la partie et aussi de la carte maitresse de la table.
  + Une fois le tableau contenant toutes les cartes disponibles rempli, il va falloir laisser le choix des cartes

### Choix des humains

* Lors de l’affichage, on a affiché un nombre à côté des cartes, et l’on récupère le choix de la carte grâce à ce nombre compris entre 0 et 7.

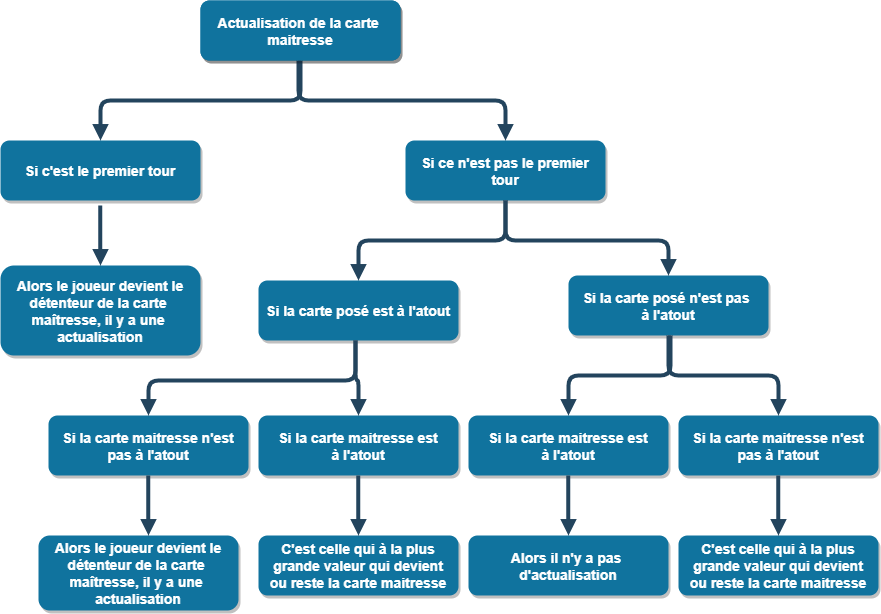
### Choix des ordinateurs

* + C’est une étape plus compliquée, on a dû utiliser une fonction qui devait choisir toute seule : la fonction jeu\_ordi**.** Cependant, comme l’ordinateur n’a à sa disposition que les cartes qu’il peut jouer, le choix est plus simple.



* + Nous avons discerné 3 cas possibles :
* Cependant nous avons vite remarqué que ce système n’était pas efficace, mais dans cette première partie nous voulions prioriser un jeu complètement fonctionnel même s’il n’est pas optimal.

### Actualisation de la carte maitresse dans le pli

* + La carte maitresse du pli désigne la carte la plus forte du pli, c’est par rapport à elle que l’on va venir déterminer ce que peuvent poser les autres joueurs et qui gagne le pli.
  + L’actualisation de celle-ci est réalisée par actualisation\_couleur\_maitresse. Elle est réalisée à chaque pose de carte, on a donc à comparer la nouvelle carte avec la carte maitresse actuelle.

### Calcul des points et attribution des points

* + Une fois le pli fini, on doit récupérer les informations de celui-ci, on appellera alors la fonction calcul\_des\_points qui permet de calculer les points des cartes présents sur la table.
  + Cette fonction va compter les cartes présentes sur la table, grâce à un tableau de valeurs contenu dans la fonction valeur, qui retourne la valeur de chaque carte.
  + Il ne reste plus qu’à attribuer les points à l’équipe qui a gagné mais aussi au joueur concerné s’il a demandé un contrat général. C’est réalisé par la fonction attribution\_des\_points.
  + On distingue aussi le cas « dix de der », lorsqu’une équipe gagne le dernier pli, alors elle gagne dix points supplémentaires.

### La vérification des contrats

* + Une fois la manche finie et donc les 8 plis terminés, il va falloir vérifier si le contrat est rempli et attribuer les points à chaque équipe.
  + La fonction verification\_contrat s’en occupe. Encore une fois on réalise une dissociation des cas :

**Une image contenant signe, rue, trafic

Description générée automatiquementUne image contenant signe, rue, trafic

Description générée automatiquementUne image contenant signe, texte, rue, autoroute

Description générée automatiquement**

# Amélioration du mode de jeu

## Calcul des annonces

* + Dans la belote coinché, on peut trouver plusieurs types d’annonces telles que les annonces de carrés (carrés de valets, de 9, rois, dames, as ou 10), et les annonces de suites par couleur (suite de 5, 4, 3). De plus les annonces vont varier selon si c’est un mode couleur ou SA.
  + On va avoir besoin de calculer les annonces à plusieurs reprises. Premièrement dans les contrats où l’ordinateur va tenir compte des annonces de son jeu pour faire une enchère. Puis une fois que le jeu a commencé pour calculer les annonces de chacun en fonction du mode de jeu.
  + Cependant, on ne pouvait pas afficher les annonces n’importe comment et n’importe quand. Ainsi lors du premier tour on va tester pour chaque joueur, dans leur ordre de jeu, s’il a une annonce et si son annonce est plus forte qu’une qui aurait été précédemment déclaré lors de ce tour. Si tel est le cas alors son annonce est retenue pour la partie comme la plus forte.
  + Pour arriver à nos fins, nous avons séparé le travail à effectuer en 3 parties.

### Calcul des annonces de types suites par couleurs

* + Nous devions pour cette partie trier les cartes par couleurs, déterminer pour chaque couleur l’annonce maximale que le joueur peut avoir, et enfin retourner l’annonce maximale parmi toutes les couleurs.
  + Une fonction principale va d’abord trier le paquet de cartes du joueur en couleurs, puis va appeler une autre fonction plusieurs fois pour calculer les annonces. Fonction qui pour chaque famille, vérifie si le joueur possède une suite dans celle-ci, et lui attribuer (ou non) les points de l’annonce. On gardera uniquement l’annonce maximale parmi les 4 couleurs présentes dans le paquet de jeu.
  + On va rechercher les suites en vérifiant toutes les possibilités de suite possibles pour une famille (suite de 3, 4 ou 5 cartes), après avoir préalablement vérifié que la famille contenait bien au minimum trois cartes.

### Calcul des annonces de type carrés

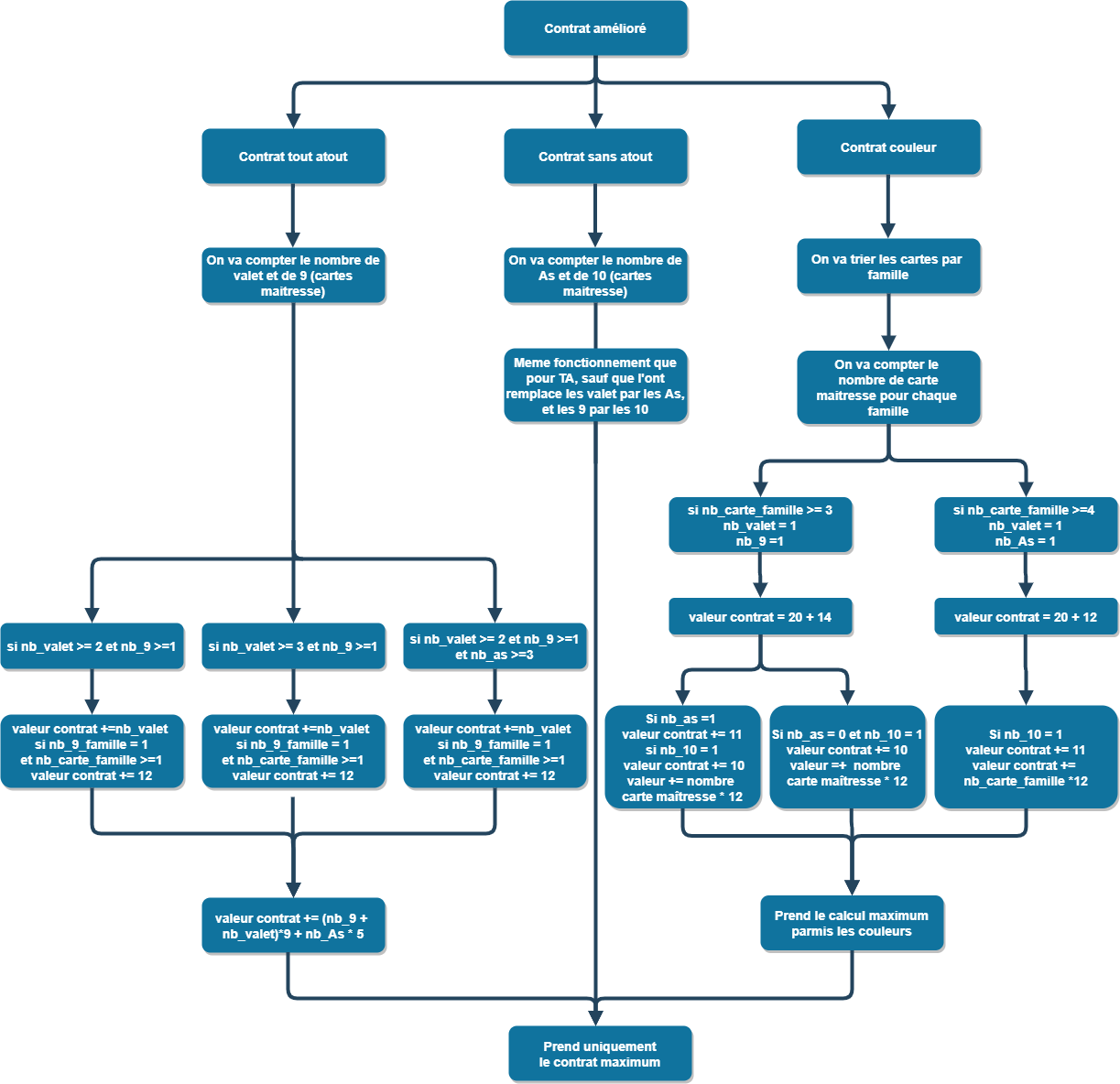
* + Pour cette partie, il a fallu prendre en paramètre l’atout de la partie. En effet, selon que l’on est à SA ou le reste des modes, la valeur des carrés diffère du fait que les cartes les plus fortes changent (Valet au lieu de l’As …).
  + Dans cette partie on va venir compter le nombre de cartes qui importent pour les carrés dans le paquet du joueur, on va ensuite venir tester les différentes possibilités et retourner la plus grande.

### Calcul de l’annonce maximum du joueur

* + Finalement, une fois toutes les possibilités d’annonce épluchées, on va venir retenir l’annonce la plus forte pour le joueur et la lui attribuer.

## Amélioration et perfectionnement des contrats

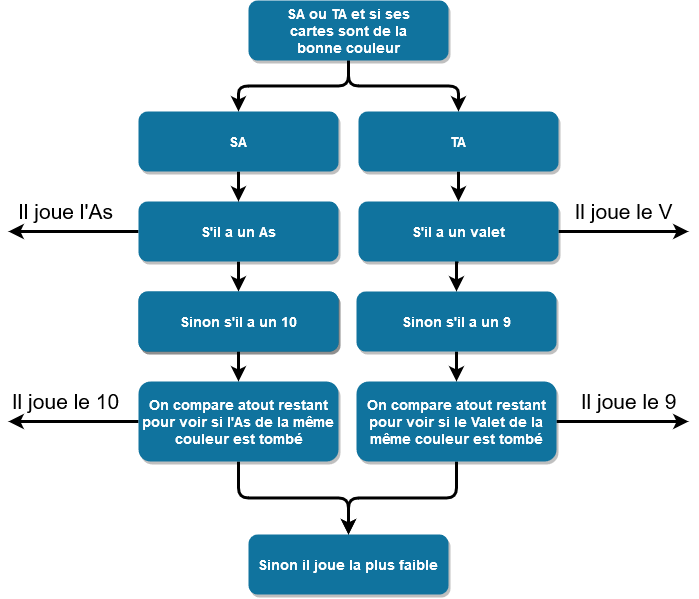
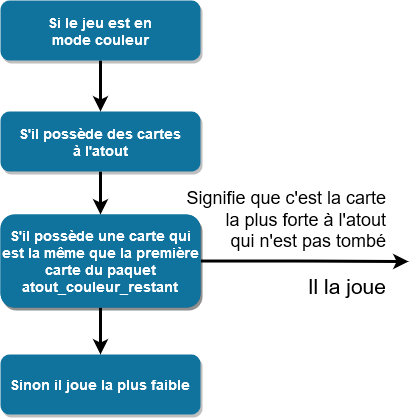
* + On a trouvé notre ancien système, bien que fonctionnel, désuet et non représentatif d’un joueur réel de coinche.
  + On a donc décidé d’étoffer les possibilités pour permettre à l’ordinateur de faire des contrats en fonction de leur jeu, bien plus proche d’un joueur réel. Nous avons créé plusieurs fonctions qui va évaluer les paquets de chaque joueur, et proposer un contrat pour les ordinateurs.
  + Pour cela, on a mis en place un système de comparaison des différentes possibilités offertes pour un jeu. On va ainsi tester ce que pourrait valoir son jeu en TA, en SA puis s’il devait choisir une couleur. On va s’appuyer sur une variable qui va simuler les points que pourraient remporter le joueur grâce à ses cartes.
  + De manière générale, pour ce qui est SA et TA, on va venir attribuer un nombre de points élevé à la carte la plus forte du mode (la carte maitresse) car avec elle on est sûr de remporter le pli, puis d’autres points sur la carte juste inférieure à celle-ci. Cependant sur cette dernière pour ajouter du réalisme au contrat, il faut qu’il ait d’autres cartes de la même couleur pour éviter de se faire prendre sa carte au cas où un autre joueur aurait une carte supérieure à la sienne.
  + En couleur notre système est assez différent puisqu’il se rapproche du fonctionnement des annonces, où le système va venir faire des estimations de point sur chaque famille de couleur. Il va commencer par vérifier qu’il ait les cartes requises pour qu’il parte sur cette couleur, tel que le valet puis un 9 ou un As et un nombre de carte suffisant dans cette couleur.
  + On a tout d’abord pensé que pour faire ces estimations, on pouvait se baser sur la valeur réelle des cartes, mais on s’est rendu compte que l’on devrait ajuster ces valeurs pour que le jeu soit réaliste et que l’ordinateur décide d’enchérir. Ainsi on a fait de nombreux essais afin que l’ordinateur enchérisse de manière assez variée et précise par rapport à son jeu.
  + Dans notre volonté de rendre nos ordinateurs plus « réalistes et intelligents », on a mis en place un système de surenchère, son principe est simple mais son exécution a été laborieuse. Lorsqu’une enchère a déjà eu lieu, le coéquipier de celui qui a posé l’enchère va vérifier s’il a dans son jeu des cartes qui pourraient compléter le contrat de son allier, si c’est le cas, alors leurs jeux cumulés vont permettre de partir sur un contrat plus risqué et donc rajouter plus 10 aux contrats. On a dû mettre en place un système qui va vérifier si une surenchère a déjà eu lieu pour éviter que les ordinateurs ne surenchérissent entre eux sans fin. De plus, il a fallu réinitialiser cette surenchère au cas où entre temps une nouvelle enchère qui ne soit pas de la surenchère ait eu lieu.
  + Nous sommes arrivés à nos fins en balayant un grand nombre de possibilités, mais cette méthode reste très « approximative ». Nous avons dû en effet faire des choix pour simplifier nos algorithmes qui ne reflètent parfois pas totalement la réalité. D’autant plus que cette façon de prédire les plis est assez risquée car elle va dépendre de la partie.



## Amélioration et perfectionnement de l’intelligence des ordinateurs

* L’objectif était de rendre le jeu des ordinateurs bien plus naturel mais aussi beaucoup plus logique et performant.
* On est parti du constat que notre ancien mode de fonctionnement avec 3 possibilités : aléatoire/plus faible/plus forte était en réalité inefficace. On a donc choisi une toute nouvelle logique bien plus performante, basée sur le principe de comptage des cartes à l’atout.
* Pour cela, on a d’abord mis en place deux variables
* atout\_restant qui va venir stocker les couleurs des cartes maîtresse, donc le Valet en tout-atout ou As en sans-atout.
* atout\_couleur\_restant qui va elle stocker le nom des cartes de la famille à l’atout. Elle a la particularité d’être classée par ordre décroissante, de telle sorte que la carte la plus forte se trouve toujours en première position.

Ces variables ont pour objectif de stocker les cartes maitresses qui sont tombées. Ce qui va permettre aux ordinateurs de jouer en fonction des cartes maitresses tombées.

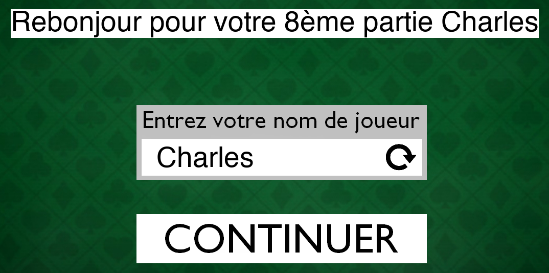
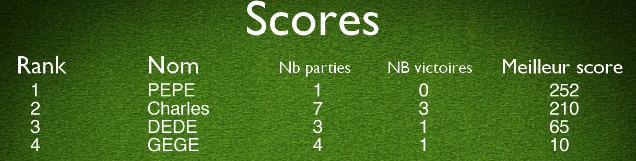
1. On va tester si le joueur possède des cartes maitresses du pli selon les modes, s’il n’a rien il joue la plus faible
2. A la fin du pli, on va devoir actualiser ces paquets de cartes qui comptent les atouts. C’est la fonction actualisation\_atout qui gère cela, elle est appelée à la fin du pli.

* Sur le paquet atout\_restant, qui compte les cartes en TA et SA, elle va éliminer la couleur correspondante de l’As ou du Valet selon les modes.
* Sur le paquet atout\_couleur\_restant qui compte les cartes à l’atout, si une carte à l’atout a été posée, elle est enlevée du paquet et toutes les cartes sont décalées pour ne pas laisser de trou et avoir la carte la plus forte en première position.

## Mise en place de la belote-rebelote et du dix de der

* La belote-rebelote correspond à un bonus de dix points attribués à l’équipe qui possède le roi et la dame à l’atout, c’est un bonus qui ne fonctionne que lors d’un jeu à la couleur. Pour faire cela, on va calculer dès le début de la manche la belote-rebelote, on va donc vérifier si dans les paquets des joueurs d’une même équipe on trouve un roi et une dame à l’atout, si oui on va attribuer dès à présent les points à l’équipe.
* Il y a en revanche un problème, il n’est pas logique d’afficher la belote-rebelotte dès le début de la partie, il a donc fallu trouver un moyen de l’afficher au bon moment. Ainsi on a mis en place un comptage du roi et de la dame à l’atout lors de la vérification des cartes maitresses, une fois que ces deux cartes sont tombées, on vérifie si une équipe possède ce bonus, si c’est le cas on l’affiche alors à ce moment-là.
* Le dix-de-der constitue le bonus octroyé à l’équipe qui remporte le dernier pli. Ainsi lors de l’attribution des points de la manche, si c’est le dernier tour, on retient l’équipe détentrice du dix de der, pour pouvoir ajouter les 10 points supplémentaires lors du comptage final des points.

## Enregistrement des scores

* Il était intéressant dans notre jeu de pouvoir à la fois enregistrer les scores mais aussi de pouvoir les afficher dans le menu principal. Pour cela on a mis en place un tableau de structures qui va stocker pour chaque joueur enregistrés leurs informations. C’est à dire leur nom, leur nombre de parties jouées et leur meilleur score qui représente la plus grosse différence de score que l’équipe du joueur à pu marquer face à l’équipe adverse.
* Après avoir récupérer le nom du joueur au lancement du jeu, on va remplir ce tableau. Pour cela on récupère ligne par ligne les informations du fichier que l’on remet dans le tableau. Ensuite on va rechercher le nom du joueur nom dans le tableau pour savoir s’il a déjà joué. Si c’est le cas, on va retenir sa position dans le tableau. Sinon on va venir lui créer une nouvelle ligne juste après les autres joueurs déjà enregistrés.
* A la fin du jeu, on va venir compléter ce tableau de structure qui retient les scores avec les informations de la partie. Pour cela on va retourner à la ligne qui correspond au joueur que l’on a déterminer au début du jeu. On lui actualise son nombre de parties, de parties gagnées et puis si son score est meilleur que son précédent meilleur score enregistré, alors il est aussi changé.
* L’étape suivante est de reclasser le tableau par ordre croissant, car le score du joueur a pu changer. Pour cela on va effectuer un tri à bulles, qui est parfait dans ce cas. En effet seul le score du joueur concerné a changé et il n’est pas nécessaire de reclasser tous les éléments du tableau.
* Une fois l’actualisation et le reclassement terminé, on va venir actualiser le fichier en écrasant les anciennes données par les nouvelles contenues dans le tableau de structure de la partie.

# Interface graphique

* Notre objectif était de réaliser une interface jolie et ludique pour notre jeu de coinche. Chose que l’on ne parvenait pas à réaliser avec la console, pour cela on a donc dû utiliser des bibliothèques additionnelles en C.
* La bibliothèque de gestion d’affichage SDL en version 1.2, c’est la bibliothèque principale, c’est elle qui gérait toute l’interface graphique
* La bibliothèque complémentaire à la SDL, nommé SDL Image qui venait rajouter le support d’image png, jpeg, tiff … à la SDL, elle nous a été utile pour insérer des images transparentes.
* La bibliothèque SDL ttf, comme la SDL ne gère pas de base l’incrustation de texte, j’ai donc eu besoin de cette bibliothèque.

## Fonctionnement de la partie affichage de la SDL

* La partie affichage, s’articule autour de variables de types :
* SDL\_Surfaces, qui vont être des pointeurs liés à des surfaces rectangulaires que je vais pouvoir manier
* SDL\_Rect, qui contient les éléments de positionnement
* TTF\_Font qui va stocker les informations sur la police à utiliser
* Pour gérer ces bibliothèques, on devait utiliser de nombreuses nouvelles fonctions. Pour citer les plus utiles, il fallait une fonction pour mettre une image dans une surface (IMG\_Load), coller cette surface sur la fenêtre (SDL\_Blit\_Surface) et actualiser la fenêtre car tout ce que l’on à pu faire est stocké en mémoire mais ne s’affiche pas automatiquement (SDL\_Flip). De plus vu que toutes ces variables ont été créés par allocution dynamique, il fallait les vider (SDL\_Free\_Surface) .
* Une image contenant capture d’écran, signe

  Description générée automatiquementVoici le schéma de fonctionnement global de l’affichage que l’on a utilisé

## Fonctionnement des événements en SDL

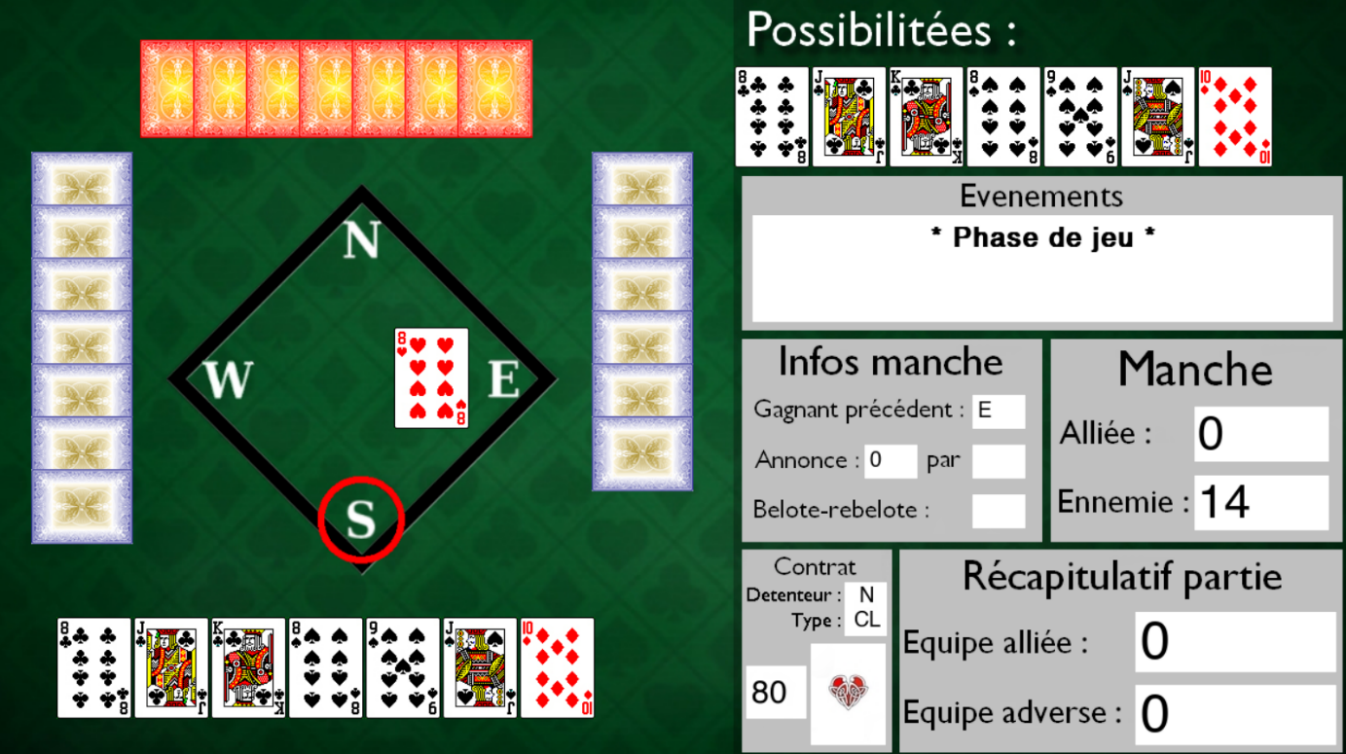
* Contrairement à la console où tout s’enchaine assez simplement, pour gérer les différents événements où l’utilisateur doit interagir, il faut mettre en place un schéma à chaque fois.
* Pour cela on utilise, la fonction de gestion des évènements de la SDL.

1. On créer un pointeur SDL\_Event qui va servir à stocker les événements qui se passent
2. On affecte l’évènement produit à la variable SDL\_Event à l’aide de la fonction SDL\_WaitEvent, cette fonction va attendre qu’il se passe quelque chose pour y affecter ce qu’il s’est passé.
3. Mise en place d’une boucle while qui ne s’arrêtera uniquement que quand le bon évènement aura été enregistré.
4. A l’intérieur de cette boucle while, pour identifier le type d’évènement, on utilise un switch, car chaque type d’évènement est codé et peut être retrouvé.
5. Si le bon événement a été enregistré et que toutes les conditions liées à l’évènement sont respectées alors on quitte la boucle et on continue la suite du code.

## Mise en place de ce système dans la coinche

* Comme il n’est plus possible d’interagir avec la console, dès le lancement de cette bibliothèque d’images, toutes les interactions doivent se faire dès à présent selon le schéma d’évènements présentés au-dessus.
* Ensuite, parce que l’écran ne change pas en temps réel, il a fallu mettre en place un système qui allait rafraichir l’écran, cependant il aurait été trop compliqué de rafraichir uniquement ce dont on a besoin alors il a fallu mettre en place une fonction qui à chaque fois remet le fond vide, puis rappelle toutes les sous-fonctions d’affichage pour ainsi reconstituer toute l’interface.
* Il y aura une exception pour l’affichage des évènements qui ne requièrent pas d’être présents de manière permanentes, c’est une fonction qui prend en paramètre notre texte, et va l’afficher au bon endroit.

## Les fonctionnalités permises par cette bibliothèque

* Une interface de jeu jolie, ludique et contenant toutes les informations nécessaires.
* Un vrai menu, contenant le lancement du jeu, les règles et les scores de façon élégante.
* La temporisation du jeu à l’aide de boutons à cliquer qui permettent de ne pas passer certaines informations trop rapidement telles qu’en fin de manche.
* De nombreuses informations affichées de manière organisées et ludiques à comprendre
* Pour savoir quel joueur doit jouer, on a pu faire apparaitre un rond rouge qui permet de rendre l’interface plus interactive.
* Lors des contrats, il y a une fonction dédiée qui fait apparaitre le choix des joueurs quant à leur décision d’enchérir ou de passer, cette fonction a été difficile à coder car il fallait retenir les noms de ceux qui étaient passés précédemment pour pouvoir placer les informations au bon endroit.
* L’affichage et la récupération du choix du joueur de manière interactive sous formes de cartes. Pour récupérer le choix du joueur, on récupère la position relative de sa souris dans le cadre des possibilités, que l’on divise par la taille des cartes, ce qui nous ramène à un nombre compris entre 0 et 7, ce qui est parfait car cela nous a permis de réutiliser tout notre ancien système de possibilités où on récupérait le choix avec un getch.
* L’affichage des cartes et de la table, cela se base sur une fonction qui va en fonction du nom de la carte à afficher du paquet, retourner le chemin de la carte à afficher.

Interface graphique de notre belote coinchée lors d’une partie

# Conclusion :

Dans ce jeu de belote coinchée, nous avons réussi à répondre à quasiment l’intégralité de nos objectifs annoncés à savoir : créer une belote coinchée basique pour une personne, jouant avec 3 ordinateurs codés par une intelligence artificielle assez réaliste pour que le jeu soit intéressant, et une interface à la fois ludique, simple à utiliser et agréable.

Cependant, il restera encore certaines améliorations pour notre jeu. Premièrement nos intelligences sont encore assez primaires et n’évaluent pas entièrement leur jeu. Dans la partie des contrats nous ne prenons pas en compte l’intégralité du paquet de cartes du joueur, mais seulement les cartes maitresses, de plus notre système de prédiction des points pour calculer les contrats à ses limites car il reste de nombreux paramètres à prendre en compte qui peuvent retourner la partie. Dans la phase de jeu, hormis le fait que le joueur joue ses cartes si elles sont maitresses, l’ordinateur ne va poser que la plus faible ce qui peut parfois poser un problème car il peut être judicieux de garder certaines cartes d’une famille même si elle est faible. Il nous manquera aussi toute la partie des « appels » que les ordinateurs ne gèrent pas, c’est indispensable si on veut rendre les ordinateurs plus performants. Concernant la partie graphique, certains choix que l’on a fait ne sont peut-être pas les plus optimaux en termes d’organisation de l’interface, il y a aussi quelques problèmes ergonomiques, comme le fait que l’on ne peut pas annuler notre enchère si on l’a commencé, ou encore une description de certains boutons quand on passe la souris dessus qui pourrait rendre l’interface bien plus agréable à utiliser.

Malgré tout c’est avec rigueur, investissement et organisation que nous sommes parvenus à crée un jeu de belote assez esthétique, ludique et attractif. Nous sommes donc en mesure de nous demander quelle autre stratégie pourrions-nous adopter afin de traiter l’intégralité des stratégies pour ce jeu de belote coinchée pour rendre nos ordinateurs au niveau voir supérieur aux humains ?