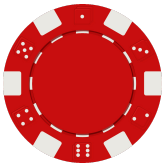
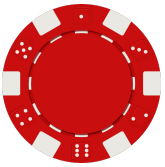


# Le blackjack



Plan:

- Présentation du projet
- Analyse du besoin
- Répartitions des tâches et démarches collaboratives
- Architecture
- Réalisation
- Bilan & perspectives

Membres:

- Benjamin CUNY - TS1
- **Lilian GALLON - TS1**
- Hugo FEYRIT - TS1

**Dossier de Lilian GALLON TS1**

# Présentation du projet

---

## 1 - Pourquoi ce projet?

Le blackjack est un jeu assez complet qui demande un aspect graphique intéressant qui, de ce fait apporte une complexité au code. De plus, nous recherchons un jeu qui incite le joueur à recommencer des parties. Pour se faire, nous allons créer des comptes bancaires, avec des paris lors du jeu.

## 2 - Règles du jeu - Contraintes

Le but, au blackjack est d'arriver le plus proche possible de 21, sans le dépasser. Le joueur commence avec deux cartes, et à n'importe quel moment il peut décider de garder ses cartes, ou d'en reprendre, avec pour limite 5 cartes au total. De plus, la valeur d'un valet, d'une dame ou d'un roi est de 10. Ce qui rend plus probable d'obtenir un 10. L'as aussi a une contrainte. Lorsque le joueur obtient un as, il choisit soit de prendre pour valeur 1, soit de prendre pour valeur 11. Le joueur pioche dans un jeu de 52 cartes, sans joker, de l'as au roi. Lorsque le joueur a finit, il compare ses cartes au croupier, dans le programme, c'est un iA, et le joueur le plus proche de 21 sans le dépasser remporte la partie, et de ce fait gagne sa mise.

## 3 - Visuel

Nous avions réfléchit au visuel que nous voulions avoir avant, mais celui-ci n'a que peu changé au cours du développement:

# Analyse du besoin

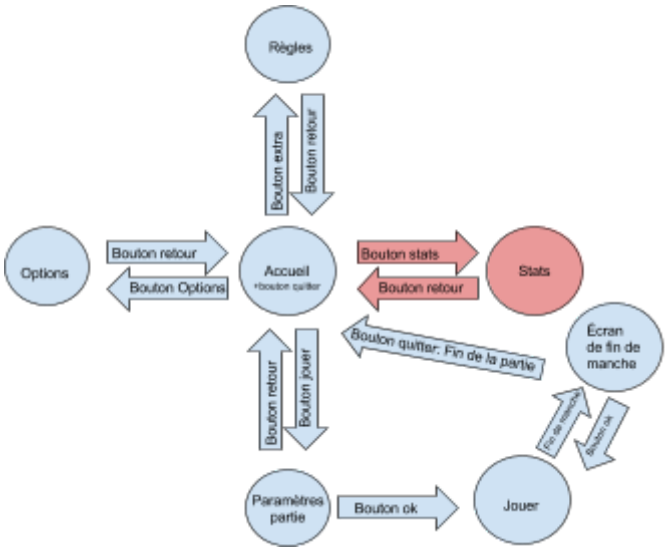
---

## 1 - Le cahier des charges

Nous avions pour but de:

- Faire le **système principal** : Intelligence artificielle primitive, sélection des cartes, possibilité de jouer;
- Ajouter un l'**aspect graphique;**
- Ajouter différents **niveaux à l'IA** (du facile à expert, tout en respectant le fait que choisir une carte est aléatoire);
- Ajouter un **système de pari**, qui prendra en compte le niveau de l'IA pour les cotes (ia facile = petite cote ; ia expert = cote énorme);
- Ajouter un système pour **sauvegarder une partie**, **plusieurs comptes**, des **options** pour choisir son compte en banque.
- Peu importe la **résolution**, le jeu doit fonctionner

Graph d'état:




En rouge: Non réalisé

## 2 - L'environnement de travail

Pour se faire, nous avons travaillé sous java, un langage étudié depuis le début de l'année. Pour débuter le développement, nous avons travaillé sous "java's cool", afin de réaliser rapidement le programme et ainsi cibler certains problèmes. Le programme final a été réalisé avec "processing", qui nous permettait d'avoir un aspect graphique intéressant, et d'autres éléments essentiels pour un jeu vidéo.

# Répartition des tâches & Démarches collaboratives

---

## 1 - Répartition des tâches

Au début, nous avions décidé de diviser nos tâches afin d'être plus efficaces, entre l'aspect visuel, et certaines fonctionnalités du programme. Mais il s'est avéré que cette méthode ne nous correspondait pas. Nous avons donc décider de travailler autrement: On se met en objectif d'ajouter une fonctionnalité au programme, et on divise de toutes petites parties de cette fonctionnalité, pour que notre travail en groupe soit lié, et pour éviter les décalages qu'il pourrait y avoir entre les membres du groupe.

## 2 - Notre démarche collaborative

La première étape, lorsque nous avions divisé nos tâches, j'ai [Lilian] étudié la manière dont le programme serait fait, l'aspect visuel, les points sur lesquels nous allions avoir à travailler indépendemment :

- Écriture / Lecture sur un fichier externe
- Boutons dynamiques
- Cartes
- Affichage / Redimensionnement d'images

En parallèle, j'ai étudié l'iA, comment nous pourrions faire pour avoir différents niveaux. J'en ai conclut que l'idée d'un iA était beaucoup trop complexe, et les différences entre les niveaux n'allait pas être très grandes, car cela ce serait joué sur les probabilités. En revanche, l'idée d'un iA basique et d'un expert est restée, car il n'y a pas d'éléments très complexes.

- Un iA basique prend une carte si sa valeur totale est inférieure à un nombre X, et s'il a un as, il vérifie s'il peut prendre 11, sinon il prend 1.
- Un iA expert connaît la carte qui va sortir et la prend si c'est inférieur ou égal à 21.

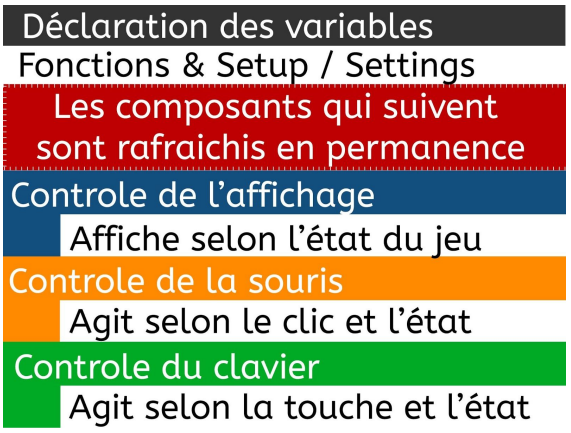
Pendant ce temps, Benjamin et Hugo ont travaillé sur le blakjack sur java's cool. C'est à ce moment là qu'on s'est aperçu que diviser nos tâches ne servait à rien: pendant que nous travaillons sur notre domaine, les autres travaillent sur un autre, et on ne prend pas connaissance du code de chacun, de plus, il nous fait les mettre en commun après, donc il faut les mêmes variables, et encore il pourrait y avoir des décalages; certains membres en avance sur d'autres..

Nous avons donc réfléchit et nous avons décidé de travailler ensemble sur le jeu. Certaines tâches étaient divisées, mais elles étaient étroitement liées, de ce fait, nous devions travailler à deux voir trois pour se compléter.

## Architecture

---

Le jeu est architecturé en trois composants essentiels; le contrôle de l'affichage, le contrôle de la souris, et le contrôle du clavier. L'affichage est rafraîchi 60 fois par seconde, quant au contrôle du clavier et de la souris, ils sont activés uniquement lors d'une interaction avec une souris ou un clavier. Le jeu agira en fonction de l'interaction du joueur avec l'écran, et de l'état du jeu.




Pour faire une partie, le jeu effectue ces tâches:

- L'initialisation du paquet, en créant un tableau aléatoirement rempli avec des valeurs de 101 à 113, 201 à 213, 301 à 3013 et 401 à 413. Avec la centaine qui correspond à un attribut, et les dizaines aux cartes (1: as, 2: 2, ..)
- Avant de commencer à jouer, il y a un écran de mise. Dans celui-ci, on peut incrémenter notre mise, de 5 en 5, avec une cote de 1, pour l'iA classique, et une cote de 10 pour l'iA extreme.
- Une fois notre mise choisie, le jeu commence. Un tableau prend aléatoirement 40 cartes. Les cartes attribuées pour le joueur sont dans un tableau, et correspondent à la 1ere à 5eme carte, tandis que l'iA prend les cartes dans ce paquet à partir de la 20eme à la 25eme carte.
- Si le joueur décide de prendre une carte, il appuie sur "prendre", ce qui lui ajoute une carte dans sa main.
- Lorsque le joueur a fini de jouer, il clique sur "garder". A partir de là, le croupier joue (sans avoir connaissance des cartes de l'humain). Pour l'iA classique, il prend tant que sa valeur totale de cartes est inférieur à 16. Et l'iA extrême connaît les cartes qui vont sortir, et les prend si c'est inférieur ou égal à 21.
- Une fois que le croupier a joué, on compare la valeur totale des cartes des deux joueurs, et on donne les gains aux différents joueurs. Si le joueur a perdu, il perd sa mise, et si le croupier a perdu, le joueur gagne sa mise fois la cote.
- On recommence, et on incrémente le compte du joueur avec l'argent gagnée, voire perdue.

# Réalisation

---

## 1 - La fonction croupier

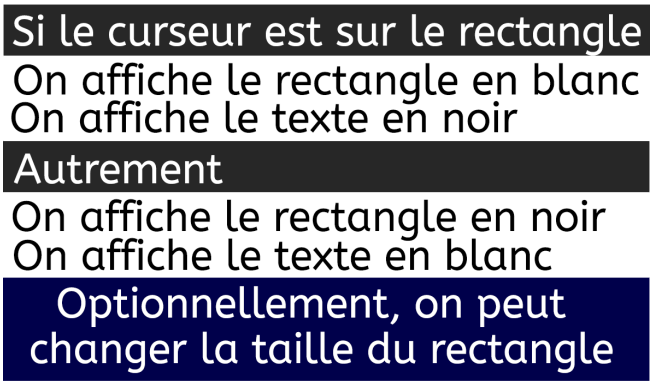
```
173 int valeurCarteC(int cartesTirees, int cartesC) {
174   if (cartesTirees%100 == 11 || cartesTirees%100 == 12 ||cartesTirees%100 == 13) { // Si la carte est un valet, une dame, ou un roi
175     return 10;                                                                     // Cela envoie 10 en valeur de carte
176   } else if (cartesTirees%100 == 1) {                                              // Sinon, si c'est un as
177     if (11+cartesC <=21) {                                                         // Si 11+ la valeur de ces cartes <= 21
178       return 11;                                                                   // On prend 11 comme valeur pour l'as
179     } else {                                                                       // Autrement
180       return 1;                                                                    // On prend 1 comme valeur de l'as
181     }
182   } else {                                                                         // Si ce n'est pas: valet, dame, roi ou as
183     return cartesTirees%100;                                                       // Alors, la valeur de la carte correspond à son numero
184   }
185 }
```

Cette fonction donne la valeur d'une carte choisie par le croupier, peu importe son niveau. Si c'est 11, 12, ou 13, cela prend pour valeur 10, si c'est un as , cela vérifie si on dépasse 21 en prenant 11, et autrement, cela renvoie la valeur de la carte (2: 2, 3: 3,..)

## 2 - Boutons dynamiques

Pour faire un bouton dynamique, nous avions besoin d'une fonction overRect(x,y,largeur,hauteur), qui envoie true si la souris est sur un rectangle.

Pour prendre les positions en temps réel de la souris, nous avions besoin de "pmouseX" et "pmouseY", car "mouseX" et "mouseY" prennent les position de la souris uniquement lorsque l'on clique.




*Code en annexe (trop long)*

## 3 - La possibilité de faire plusieurs parties

Pour faire plusieurs parties, il a fallu réinitialiser toutes les variables nécessaire à une partie. Pour se faire, le programme appelle la fonction reinitalisation(). Cette fonction remet par défaut les variables concernant le choix de l'as de l'utilisateur, la valeur totale des cartes des deux joueurs, incrémente le nombre de parties, mélange le paquet et indique que la partie peut commencer.

4 - Le gros problème avec le choix de l'as pour l'utilisateur

L'intégration du choix de l'as dans le logiciel a été un majeur problème lors du développement.

La première idée était de faire une boucle "tant que" dans le programme, avec pour condition "tant que le joueur n'a pas choisi la valeur de l'as", "on affiche les boutons, et s'il clique il prend 1, ou il prend 11". Mais, en réalité il est impossible de faire une boucle while dans un élément rafraîchi en permanence. Simplement, car le système va 60 fois par seconde dans la boucle while, et cela provoque un crash car le programme n'arrive plus à calculer.

La seconde alternative, une idée de Benjamin, a été de mettre une pause dans le programme, et tant que le choix de l'as n'est pas fait, la pause est en "true". Or, cela reste très difficile, mais cela nous a mis vers la 3eme voie, celle de créer un etatjeu pour le choix de l'as.

La solution a été de changer l'état du jeu lorsqu'un as est pioché, et de mémoriser le choix de l'as dans un tableau choixas[rangdelas], pour la réutiliser plus tard.

5 - Le problème de l'optimisation

Lors de l'importation des images de hugo sous processing, ma fonction:

```
1126 PImage convert(int numimage){
1127   PImage converti;                              // Déclare une variable image
1128   converti = loadImage(str(numimage)+".png"); // Cette variable image prend pour image "numimage.png"
1129   return converti;                              // Renvoie la variable image liée à l'image
1130 }
```

Il faut savoir que le nom des images sont 101 à 113, 201 à 213, 301 à 313, et 401 à 413

A posé de gros problèmes de fluidité. On s'en est aperçu plus tard, car les images sont chargées en permanence dans le draw(). De ce fait, benjamin a résolu le problème en initialisant les images dans un tableau lors du démarrage du programme. De plus, j'ai trouvé des boucles if() qui servaient à rien, ce qui a rendu complètement fluide le jeu.

```
1320 for(int u = 100; u < 500;u=u+100){
1321   for (int i = 1; i < 14; i++){
1322     imgs[u+i] = loadImage(u+i+".png");
1323   }
1324 }
```

L'initialisation des images, ci-dessus est faite dans setup().

## 6 - Écriture / Lecture

Le système d'écriture lecture a lui aussi posé de nombreux problèmes. Le problème venait du fait que lorsque on charge les données d'un fichier .txt, il faut les récupérer avec une variable String[]. Pour se faire j'ai réalisé deux fonction : Une qui remet à zero le fichier texte, et l'autre qui permet d'écrire.

```
1330 // Reset un fichier .txt
1331 void resetTXT(String txt) {
1332   String words = "   ";                // Initilise "words" qui sera que des espaces
1333   String[] empty = split(words, " ");  // Met "words" dans une variable String[] grâce au split
1334   saveStrings("/data/"+txt, empty);    // Sauvegarde "empty" (vide), dans /data/document.txt
1335 }
1336
1337 // Fonction pour écrire
1338 void ecrire(String ecrire, String txt) {
1339   String[] write = split(ecrire, " ");  // Met "ecrire" dans une variable String[]
1340   saveStrings("/data/"+txt, write);     // Sauvegarde "ecrire" dans /data/document.txt
1341 }
```

Ces fonctions permettent d'écrire l'argent des comptes, le compte sélectionné, l'option du son (activé ou non), l'option de la résolution (fenêtré ou plein écran). Pour charger, il suffit d'un loadStrings(fichier.txt). Exemple pour compte:

```
1229   String[] compteSTRING = loadStrings("/data/compte.txt");
1230   compte = int(compteSTRING[0]);
```

De plus, j'ai voulu que le programme vérifie s'il est à jour, et dans le cas contraire, il indique qu'une mise à jour est nécessaire et nous envoie vers le lien si on clique dessus. Pour se faire, j'ai du créer un fichier .txt disponible en ligne. Dans celui-ci, j'ai écrit 4 lignes : 1ere: "Dernière actuelle: v.x.x", 2eme: "Non à jour, cliquez ici pour mettre à jour", 3eme : [version], 4eme: "Jeu à jour! Amusez vous!". Lors du démarrage du programme, il faut récupérer les information du fichier en ligne. Il a fallu, premièrement, vérifier si le fichier est disponible:

```
1290 if (loadStrings("http://dl.dropboxusercontent.com/u/104417097/version.txt")==null) {
1291   println("Lien online innacessible");
1292 }else{
1293   println("Lien online accessible");
1294  saveStrings("/data/versionOnline.txt", loadStrings("http://dl.dropboxusercontent.com/u/104417097/version.txt"));
1295 }
```

Dans le cas où il ne l'est pas, cela pouvait provoquer un crash du jeu. Grâce à cette vérification, si le document n'est pas accessible, le système prend par défaut la dernière sauvegarde locale du fichier en ligne. Dans le cas contraire, le fichier en ligne est sauvegardé localement, sous forme de document .txt : "versionOnline.txt".

Pour l'affichage, j'ai fait:

```
306   String versionLocale[] = loadStrings("versionLocale.txt");  // Initialise versionLocale[]
307   String versionOnline[] = loadStrings("versionOnline.txt");  // Initialise versionOnline[]
308   textFont(classic);
309   textSize(width/50);
310   text(versionOnline[0],width/50,height/30);          // Affiche la ligne 1 du fichier en ligne
311   if(int(versionLocale[0])!=int(versionOnline[2])){   // Si la ligne 1 du fichier local n'est pas égal à la ligne 3 du fichier en ligne
312   text(versionOnline[1],width/50,height/15);          // On affiche la ligne 2 fichier en ligne
313   }else{                                              // Sinon
314   text(versionOnline[3],width/50,height/15);          // On affiche la ligne 4 du fichier en ligne
315   }
```

Le test if(), fait correspondre la valeur entière, sous forme de nombre de la ligne 1 de la versionLocale (où on a écrit le numéro de version), et la ligne 3 de la versionOnline (où on a écrit le numéro de version). Dans le cas où c'est identique, cela affiche la ligne 4, donc "à jour" du fichier en ligne, sinon, "non à jour".

## 7 - Les différents niveaux d'iA

Le code est disponible en annexe

L'iA normal prend une carte tant que sa valeur totale de cartes est inférieure à 16. Tandis que l'iA extrême connaît la carte qui va sortir, et la prend donc, si cela ne dépasse pas 21. La cote change en fonction de l'iA utilisé.

## 8 - Le travail sur l'aspect visuel

Le travail sur l'aspect visuel n'a pas été difficile, mais il a fallu de la patiente, pour réussir à afficher des rectangles, du texte où l'on voulait en utilisant les proportions de l'écran (width & height). Ce qui nous permet de jouer au blackjack peu importe la résolution. Quant aux cartes et aux différents fonds, il ont été faits avec des logiciels externes.

## 9 - Correspondance selon "etatjeu" de l'affichage et des interactions

Prenons l'exemple du bouton pour choisir 11 comme valeur de l'as. Nous allons étudier la relation qu'il existe entre l'affichage et les interactions.

| **Affichage** | **Interaction** | **Conséquence** |
|---|---|---|
| On affiche le bouton "prendre 11" | Le joueur appuie sur le rect. correspondant au bouton prendre 11 | On prend 11 comme choix de l'as |
| `fill(0, 50, 0);`<br>`rect(width/3.05, height/2.55, width/2.9, height/5.9, 7);`<br>`fill(200);`<br>`textSize(width/20);`<br>`text("Prendre 11", width/2.8, height/2.1);` | if(overRect(position rectangle == true) | `choixas[mainJ-1] = 11;`<br>`println("[AS] As selectionné en tant que 11");`<br>`etatJeu=EN_COURS;` |

C'est sur cette structure que toutes les interactions opérateur sont construites

## 10 - Le son

Le son, non disponible par défaut avec processing, nous avons téléchargé une bibliothèque, autrement dit, une extension, afin d'intégrer quelques sons. Le jeu a 3 sons : lorsque l'on clique, lorsque le joueur fait 21 (applaudissements), et un son de fond de casino. Nous allons nous intéresser au son du casino. Le son, pour avoir un programme léger, fait 10 secondes, et il a fallu le redémarrer donc toutes les 10 secondes pour que ce soit continu. Pour ce faire, j'ai paramétré frameRate à 60, c'est à dire le nombre de fois que l'affichage est rafraîchi par secondes. Et on incrémente une variable. Lorsqu'elle est égale à 600, cela fera 60*10 soit 10 secondes que le programme a fonctionné.

```
1254    x ++;                    // On incrémente x
1255  if(x==60*10 && son==1){    // Si x==600 & le son est activé
1256    ambiance.play(0);        // On joue le son à partir de t=0
1257    x=0;                     // On reset x
1258  }
```

# Bilan et perspectives

---

## 1 - Ce que l'on pourrait encore ajouter

Nous aurions pu ajouter des statistiques au programme, avec le plus gros pari, le record d'argent, le plus de victoire d'affilé, le ratio victoire/défaites,.. Ce qui aurait été, de plus facile à réaliser puisque nous avions les fonctions pour écrire et lire très facilement. Au niveau de l'affichage, cela aurait pu être fait dans le menu "extra". Nous aurions pu ajouter aussi des succès: faire 3 blackjack d'affilé, parier plus de 1000$,.. qui auraient pu débloquer des cartes avec des design plus intéressant par exemple.

## 2 - Ce que cela m'a apporté

Ce projet m'a apprit énormément de choses, et cela m'a confirmé ma motivation dans ce domaine. Je ne vois pas l'ISN, le développement comme une contrainte mais une passion, ce qui est une chance. Dès que j'ai su faire plein de choses sous java, j'ai directement voulu améliorer le programme, sans m'arrêter. Le blackjack à première vue était un projet qui paraissait facile, mais en ajoutant tout un tas de fonctionnalités, cela s'est complexifié. A quelques moments, c'était difficile, il a fallu chercher sur des forums, comprendre comment cela marchait pour enfin y arriver (choix de l'as et écriture / lecture). La prochaine étape est le jeu en réseau qui parait intéressant, à travers un site web par exemple.

D'ailleurs, j'ai pour but de l'améliorer encore le blackjack, en voyant tout ce qu'on peut faire, indépendamment de l'ISN pour le faire partager à d'autres personnes. D'où l'ajout de la vérification de la version.

# Annexe

## L'iA:

```
int Croupier() {
  finc=false;                                                    // Indique que le croupier n'a pas encore fini de jouer
  int cartesC = 0;                                               // La valeur totale des cartes de l'iA = 0
  cartesC = valeurCarteC(cartesTirees[19], cartesC);             // Ajoute la valeur de la premiere cartes au total
  cartesC = valeurCarteC(cartesTirees[20], cartesC) + cartesC;   // Ajoute la valeur de la seconde cartes au total
  int mainC = 2;                                                 // Indique que le croupier a 2 cartes au total
  if(iA==1){                                                       // Si l'iA est au niveau normal
  while (cartesC <16) {                                              // Tant que la valeur de ses cartes est inférieur à 16
    mainC ++;                                                          // On ajoute une carte dans sa main
    cartesC = cartesC + valeurCarteC(cartesTirees[18 + mainC], cartesC);   // On ajoute la valeur de cette carte au total
  }                                                                  // Fin du tant que
  return  cartesC;                                                 // Renvoie la valeur totale de ses cartes
  }else{                                                         // Si l'iA est au niveau extreme
    while(finc==false){                                            // Tant que l'iA n'a pas fini de jouer
    mainC ++;                                                      // On ajoute une carte à sa main
    if (cartesC + valeurCarteC(cartesTirees[18 + mainC], cartesC)<=21){   // Si la valeur totale + la carte suivante est inférieure à 21
    cartesC = cartesC + valeurCarteC(cartesTirees[18 + mainC], cartesC);     // Il prend une carte dans ce cas
    }else{finc=true;}                                              // Autrement il finit de jouer
    }                                                            // Fin du tant que
    return cartesC;                                              // Renvoie la valeur totale
  }
}
```

## Fonction overRect:

```
boolean overRect(double x, double y, double largeur, double hauteur) {
  if (pmouseX >= x && pmouseX <= x+largeur &&
    pmouseY >= y && pmouseY <= y+hauteur) {
    return true;
  } else {
    return false;
  }
}
```

## Exemple d'un bouton dynamique:

```
    if (overRect(width/2.5, height/3, width/5, height/6)==true) {  // Si le joueur a sur curseur sur le rectangle
      fill(255, 0, 0);                                               // On prend pour couleur rouge
      rect(width/2.5, height/3, width/5, height/5.9, 7);             // On dessine un rectangle de cette couleur
      fill(255);                                                     // On prend pour couleur: blanc
      textFont(menu, width/20);                                      // On prend comme police de caractère celle du menu
      text(text, width/2.4, height/2.23);                            // On affiche "text"
    } else {                                                       // Sinon
      fill(255);                                                     // On prend pour couleur: blanc
      rect(width/2.5, height/3, width/5, height/6, 7);               // On dessine un rectangle de cette couleur
      fill(255, 0, 0);                                               // On prend pour couleur: rouge
      textFont(menu, width/20);                                      // On prend comme police de caractère celle du menu
      text(text, width/2.4, height/2.23);                            // On affiche "text"
    }
```

Toutes les versions (*=Résolution de bug ; + = Ajout ; - = Suppresion):

Alpha 1.0: ?
Fonctionnel, avec iA classique

---

Alpha 2.0: ?

+ Nombre de partie à choisir
+ Scores
+ Réinitialisation

---

**Alpha 2.1:** *26/03/16*

+ Reconnaissance de l'as, dame, roi et valet pour le croupier aussi)
+ Affichage clear

---

Alpha 2.2: *29/03/16*

* Problème résolu sur les défaites / victoires / égalités

---

Beta1dev: *01/04/16*

! Début de l'export vers processing (la structure du programme est faite)

---

Beta2 - Beta 2.1: *06/04/16 - 07/04/16*
Première intégration du jeu dans processing : qqes fonctionnalités supprimées, des bugs, besoin de l'aspect graphique vu que les "read…" ne marchent pas sous Processing

---

Beta3: *08/04/16*
Programme fonctionnel, avec AUCUN affichage

---

Beta3_1: *12/04/16*
Ajout d'un premier affichage acceuil (Benjamin) et en jeu (Lilian)

---

**Beta 3.2:** *24/04/16*
* Réglage du bug lorsque l'on cliquait sur "quitter", maintenant, il faut finir la partie après avoir cliqué sur quitter pour quitter.

+ Initialisation permettant de recommencer plusieurs parties, tant que certaines conditions sont remplies
+ Score est affiché
+ Emplacement des cartes avec, en attendant d'afficher les cartes, leur valeur qui s'affiche
+ Le total est affiché à gauche

Bilan 3.2: Programme fonctionnel, comme sous java's cool, mais sans intégrer l'écran d'affichage de Benjamin. Mais l'ajout du choix de l'AS reste difficile et sera mis plus tard, ne dépendant d'aucune fonctionnalité du programme, le mettre maintenant ou plus tard revient au même.

---

**Beta 3.3:** *26/04/16*

+ Menu de benjamin (reste à bien intégrer les boutons)
+ Choix de l'as
+ Ajout surprise ( ͡° ل͜ ͡°)

* Les valeurs des cartes ne sont plus cachées quand on doit choisir l'as

Bug à noter:

- Quand un spam (plus de 50 fois par sec), il est possible de faire buguer le programme avec un tableau qui demande une colonne de plus.
- Quelques problèmes d'affichage quand on choisi l'AS

---

**Beta 3.4:** *27/04/16*

* Bug avec choix de l'as en deuxième carte
* Bug de proportion au niveau de certaines écritures
* Refonte du menu avec les boutons
+ Ajout de polices d'écriture
+ Ajout d'un bouton "options"
+ Ajout d'un bouton "extra"
+ Compatibilité possible sur android (un test a été fait, cela fonctionne)

Dans la fenêtre options:

+ Résolution : Fenêtré, ou plein écran (avec un bouton appliquer pour restart)
+ Choix du compte (3 possibles)
+ Une fois un élément sélectionné, il est mis en couleur, pour montrer qu'il a été choisit

Problèmes:

- Il reste à mettre un bouton pour quitter le menu options et revenir à l'accueil
- Il manque le programme d'écriture / lecture pour sauvegarder et charger les valeurs de l'option (le programme est tout prêt pour ajouter cette fonctionnalité)

---

**Beta 3.5:** *25/05/16*

* Menu options fonctionnel
+ Écriture / Lecture
+ Paris ajouté par Benjamin

---

**Beta 3.6:** *06/05/16*

* Améliorations graphiques
+ Image en fond écran du menu
+ Dans le menu MISE, ajout du bouton retour et affichage de la mise et de l'argent
+ Problème pour le bouton MISE +, la police de caractère ne prend pas en compte les

* Les paris
+ Quand on ouvre le programme, le jeu va chercher combien d'argent on a dans le fichier externe
+ A la fin de chaque manche il sauvegarde dans le fichier externe notre argent

**Beta 3.7:** *07/05/16*

+ Écran de fin de manche
+ Écran de fin de partie
+ Ajout de mise sur le panel en jeu

* Élargissement du panel
* Le texte s'affiche sur le panel et non plus le panel sur le texte

---

**Version 1.0 :** *08/05/16*

+ Fond
+ Fondu stylay en fin de manche
+ Chargement des comptes dans le menu options
+ Argent sauvegardé automatiquement
+ Menu de pause
+ Option activer / désactiver le son
+ Cartes de hugo

* Menu pause rectifié
* Carré noir pourri

Note: il y a un problème de refresh de l'écran (aspect visuel)
Problème plus important: Vérifier où est sauvegardé l'argent, si aucun fichier n'est créé au mauvais endroit, après il y a confusion

---

**Version 1.1 :** *10/05/16*

+ Ajout d'un bouton "quitter" dans le menu de fin de manche

* Résolution du problème de la 5eme carte non prise en compte
* Amélioration de l'aspect graphique lors du choix de l'as avec un dégradé et la suppression d'un carré noir
* Résolution du problème de scintillement de l'affichage

Note: Il reste un problème de délai lorsque l'on met son curseur sur un bouton (lié aux performances du logiciel, donc indépendant de nous)

* Fond plus clair en jeu et amélioré
* Résolution d'un crash lorsque l'on clique en haut à gauche (par Benjamin)
* Résolution d'un problème d'affichage dans les options (compte #3)
* Résolution d'un problème de rafraîchissement dans le menu options (argent pas actualisé)
* Résolution de la sauvegarde des fichiers txt n'importe où

---

**Version 1.2 :** *11/05/16*

+ Ajout de l'affichage de la carte du croupier

* Ajout du chargement de l'argent depuis le fichier externe au lancement du programme en fonction du numéro du compte (aussi lu dans un fichier externe). Avant l'argent était 5cr par défaut avant qu'on sélectionne un compte dans les options

---

**Version 1.3 :** *11/05/16*

* “cr” remplacé dans le menu miser par $
* Résolution du bug qui nous fait quitter quand on appuie sur OK après avoir quitté une fois
* La valeur de la carte du croupier n’est plus affichée
* Suppression du point d’interrogation au lieu de l’as lors du choix de celui-ci
* Si quelqu’un pari tout son argent et il perd, son compte en banque revient à 5 (défaut), et sa mise se met automatiquement à 5
* Optimisation en cours de jeu (plus fluide)

---

**Version 1.4 :** *12/05/16*

* Grosse optimisation au niveau du framerate (remplacement du stockage des cartes)

---

**Version 1.5 :** *12/05/16* - **VERSION STABLE**

* Programme entièrement optimisé (fluidité parfaite en jeu comme dans le menu)
* Résolution de tous les problèmes liés aux caractères non-reconnus ($ remplacé par CR)

---

**Version 2.0 :** *19/05/16*

+ Ajout d’un test qui vérifie qu’on utilise bien la dernière version du jeu et qui demande de la télécharger dans le cas contraire
+ Ajout de règles dans le menu EXTRA
+ Ajout du nom des créateurs dans le menu EXTRA
+ Ajout de l’iA casi-Imbatable
+ Ajout d’une cote lors d’une victoire face à l’iA imbatable de X10
+ Ajout d’un son de fond au menu qui se relance automatiquement grâce à frameRate
+ Ajout d’un son d’applaudissement pour 21
- Son troll pour un 21

* Problème de $ et CR
* L’iA classque prenait deux fois la deuxième carte piochée
* Résolution d’un moyen pour tricher sur les paris avec un compte bien garni

---

**Version 2.1 :** *20/05/16*

+ Ajout d’un système qui vérifie si le lien enligne est accessible, autrement, il prend la dernière sauvegarde du fichier en ligne - Cela règle un crash au démarrage (rare)

---

**Version 2.2 :** *20/05/16*

* Lorsqu’on perd, avec l’ia extreme, on perd la mise et non pas la mise*cote
* Différence cote / mise dans l’affichage
* Le son de victoire pour avoir fait un blackjack se joue directement

+ Texte de chargement (fonctionne 1/10)

---

**Version 2.3 :** *21-22/05/16*

+ Ajout graphique : le jeton du menu tourne

* Résolution du bug avec l’argent et la mise par défaut de 5 sans prendre en compte l’iA
* Résolution du bug où le jeton n’était pas dynamique lorsque le curseur est dessus
* Résolution du problème avec les hitbox extra et options

- Suppression de la fonction valeurcarteJdemande, qui ne servait plus