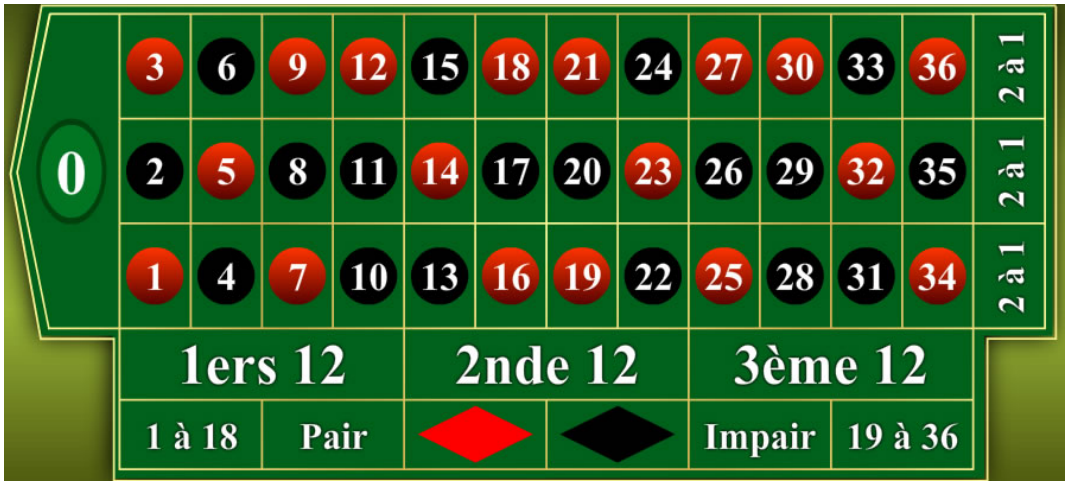
PROJET ROULETTE



Etudiant :	Gilles LORIQUER
Etablissement :	CNAM (Pays de Loire)
Unité d'enseignement :	NFA031 (Programmation Java – notions de base)
Enseignant :	Patrick DOUGUET
Promotion :	2017 – 2018

Table des matières

Introduction

Dans le cadre du suivi de l'unité d'enseignement NFA031 (Programmation Java – notions de base) au sein du Conservatoire des Arts et Métier de Nantes, un projet de fin de semestre est demandé aux étudiants afin de valider les compétences acquises.

Ce projet a pour objectif la réalisation du jeu de la roulette de casino. Vous trouverez le détail des règles du jeu en annexe.

L'objectif de ce projet est de réaliser un jeu de roulette en langage Java et en respectant certaines exigences. Ce programme doit être capable :

1. De tirer un numéro aléatoire entre 0 et 36
2. De définir les caractéristiques de ce numéro (couleur, parité, appartenance aux colonnes etc.)
3. De permettre à un utilisateur de choisir un type de pari et de miser sur celui-ci (paris intérieurs et extérieurs)
4. De définir si le pari réalisé est gagnant ou perdant et de ventiler les gains ou les pertes selon les rapports de gain propres à chaque pari
5. Enfin, ce programme devra permettre jusqu'à trois utilisateurs de prendre part à une partie

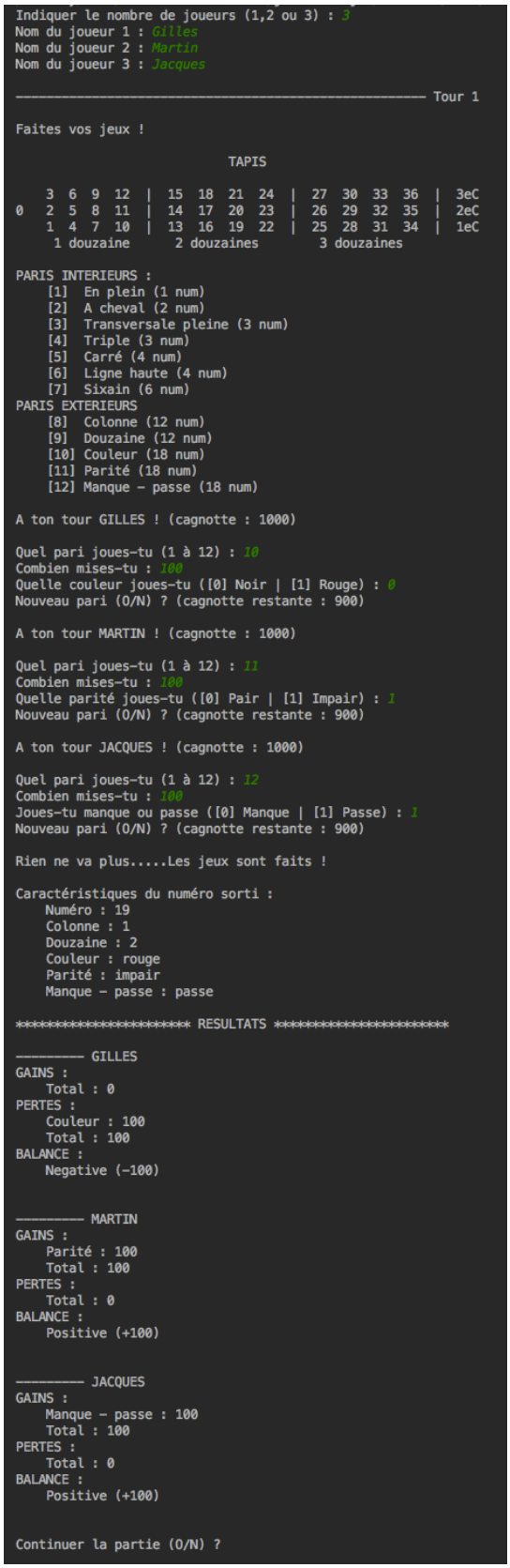
L'objectif de ce document est de présenter l'analyse qui a pu conduire à la réalisation de ce programme. Vous trouverez dans les lignes qui suivent les explications nécessaires à la bonne compréhension du programme Java fourni en annexe.

Présentation du programme

Cette section décrit le déroulement d'une partie à travers la réalisation du programme Java.

Déroulement d'une partie

L'image ci-dessous montre le déroulement complète d'une partie. Les étapes détaillées se trouvent à la suite.



Phase 1 : récupération du nombre de joueurs

Cette phase permet de récupérer le nombre de joueur de la partie (de 1 à 3).

```
Indiquer le nombre de joueurs (1,2 ou 3) : 3
```

Phase 2 : récupération des noms des joueurs

Cette phase récupère les noms de chaque joueur de la partie en fonction du nombre de joueurs indiqué en phase 1.

```
Nom du joueur 1 : Gilles
Nom du joueur 2 : Martin
Nom du joueur 3 : Jacques
```

Phase 3 : affichage du tapis de roulette

Cette phase affiche le tapis de la roulette.

```
                              TAPIS

     3  6  9  12  |  15  18  21  24  |  27  30  33  36  |  3eC
0    2  5  8  11  |  14  17  20  23  |  26  29  32  35  |  2eC
     1  4  7  10  |  13  16  19  22  |  25  28  31  34  |  1eC
       1 douzaine      2 douzaines       3 douzaines
```

Phase 4 : affichage des types de pari

Cette phase affiche les types de pari disponibles pour les joueurs.

```
PARIS INTERIEURS :
    [1]  En plein (1 num)
    [2]  A cheval (2 num)
    [3]  Transversale pleine (3 num)
    [4]  Triple (3 num)
    [5]  Carré (4 num)
    [6]  Ligne haute (4 num)
    [7]  Sixain (6 num)
PARIS EXTERIEURS
    [8]  Colonne (12 num)
    [9]  Douzaine (12 num)
    [10] Couleur (18 num)
    [11] Parité (18 num)
    [12] Manque – passe (18 num)
```

Phase 5 : récupération des paris pour chaque joueur

Pour chaque joueur présent dans la partie, on récupère ses paris avec une cagnotte par défaut initialisée à 1000. Si le joueur effectue plusieurs paris, sa cagnotte est provisoirement amputée des paris qu'il a déjà effectué, de sorte qu'il ne puisse miser plus que le contenu de sa cagnotte.

```
A ton tour GILLES ! (cagnotte : 1000)

Quel pari joues-tu (1 à 12) : 10
Combien mises-tu : 100
Quelle couleur joues-tu ([0] Noir | [1] Rouge) : 0
Nouveau pari (O/N) ? (cagnotte restante : 900)

A ton tour MARTIN ! (cagnotte : 1000)

Quel pari joues-tu (1 à 12) : 11
Combien mises-tu : 100
Quelle parité joues-tu ([0] Pair | [1] Impair) : 1
Nouveau pari (O/N) ? (cagnotte restante : 900)

A ton tour JACQUES ! (cagnotte : 1000)

Quel pari joues-tu (1 à 12) : 12
Combien mises-tu : 100
Joues-tu manque ou passe ([0] Manque | [1] Passe) : 1
Nouveau pari (O/N) ? (cagnotte restante : 900)
```

Phase 6 : simulation du lancement de la bille et annonce des résultats

Cette phase simule le lancement de la bille et les annonces faites par le croupier et affiche les caractéristiques du numéro sorti.

```
Rien ne va plus.....Les jeux sont faits !

Caractéristiques du numéro sorti :
    Numéro : 19
    Colonne : 1
    Douzaine : 2
    Couleur : rouge
    Parité : impair
    Manque – passe : passe
```

Phase 7 : affichage des résultats par joueur (gains, pertes et balance)

Cette phase affiche les résultats pour chaque joueur en termes de gains, pertes et de balance globale.

```
********** RESULTATS **********

--------- GILLES
GAINS :
    Total : 0
PERTES :
    Couleur : 100
    Total : 100
BALANCE :
    Negative (-100)


--------- MARTIN
GAINS :
    Parité : 100
    Total : 100
PERTES :
    Total : 0
BALANCE :
    Positive (+100)


--------- JACQUES
GAINS :
    Manque – passe : 100
    Total : 100
PERTES :
    Total : 0
BALANCE :
    Positive (+100)
```

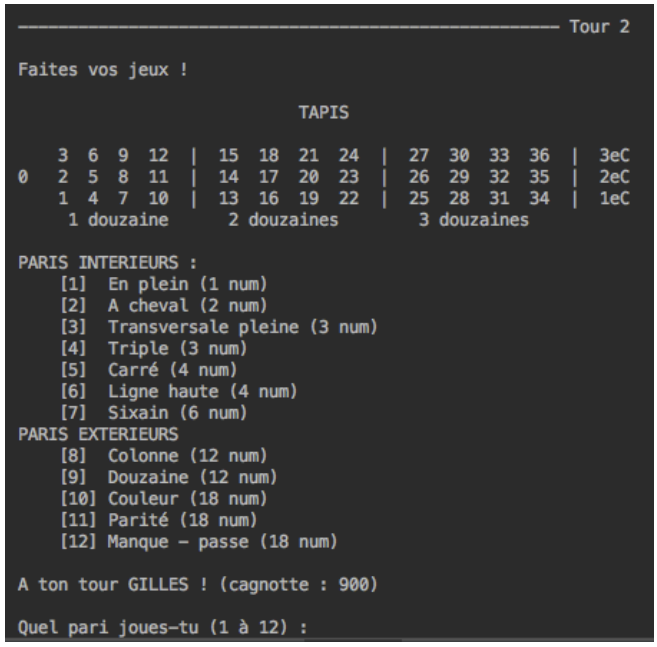
Phase 8 : demande de poursuite de la partie

Cette phase permet de proposer aux joueurs s'ils souhaitent continuer la partie.

```
Continuer la partie (O/N) ?
```

Phase 9 : poursuite de la partie avec mise à jour des cagnottes

En cas de poursuite de la partie, cette phase propose de nouveau aux utilisateurs d'effectuer des paris et leur cagnotte est mise à jour en fonction des pertes et des gains de la partie précédente. Le nombre de tour est également mis à jour pour connaître le nombre de parties qui se sont déroulées. Si un utilisateur n'a plus d'argent dans sa cagnotte, celui-ci est annoncé comme ruiné et on passe au joueur suivant.


---------- Tour 2
Faites vos jeux !
TAPIS
3 6 9 12 | 15 18 21 24 | 27 30 33 36 | 3eC
0 2 5 8 11 | 14 17 20 23 | 26 29 32 35 | 2eC
1 4 7 10 | 13 16 19 22 | 25 28 31 34 | 1eC
1 douzaine 2 douzaines 3 douzaines
PARIS INTERIEURS :
[1] En plein (1 num)
[2] A cheval (2 num)
[3] Transversale pleine (3 num)
[4] Triple (3 num)
[5] Carré (4 num)
[6] Ligne haute (4 num)
[7] Sixain (6 num)
PARIS EXTERIEURS
[8] Colonne (12 num)
[9] Douzaine (12 num)
[10] Couleur (18 num)
[11] Parité (18 num)
[12] Manque - passe (18 num)
A ton tour GILLES ! (cagnotte : 900)
Quel pari joues-tu (1 à 12) :

Analyse et conception

Cette section présente l'analyse et la conception du programme Java déroulé précédemment. Est abordé ici le découpage du programme et, pour chaque classe, le descriptif de son contenu :

- Variables
- Fonctions
- Procédure
- Organigrammes (global et par fonctions complexes si nécessaire)

Découpage du programme

A la vue du déroulement d'une partie de roulette j'ai décidé de découper le programme en 7 classes :

- Classe Main
- Classe Partie
- Classe Joueur
- Classe Pari
- Classe Jeu
- Classe Util
- Classe Saisie

Pour chaque classe une description ainsi que l'explication de son contenu sera réalisée (variables, fonctions, procédures etc.).

L'organigramme global correspond à la classe Main. Des organigrammes seront également présentés pour les fonctions complexes de la classe Jeu.

Classe Main

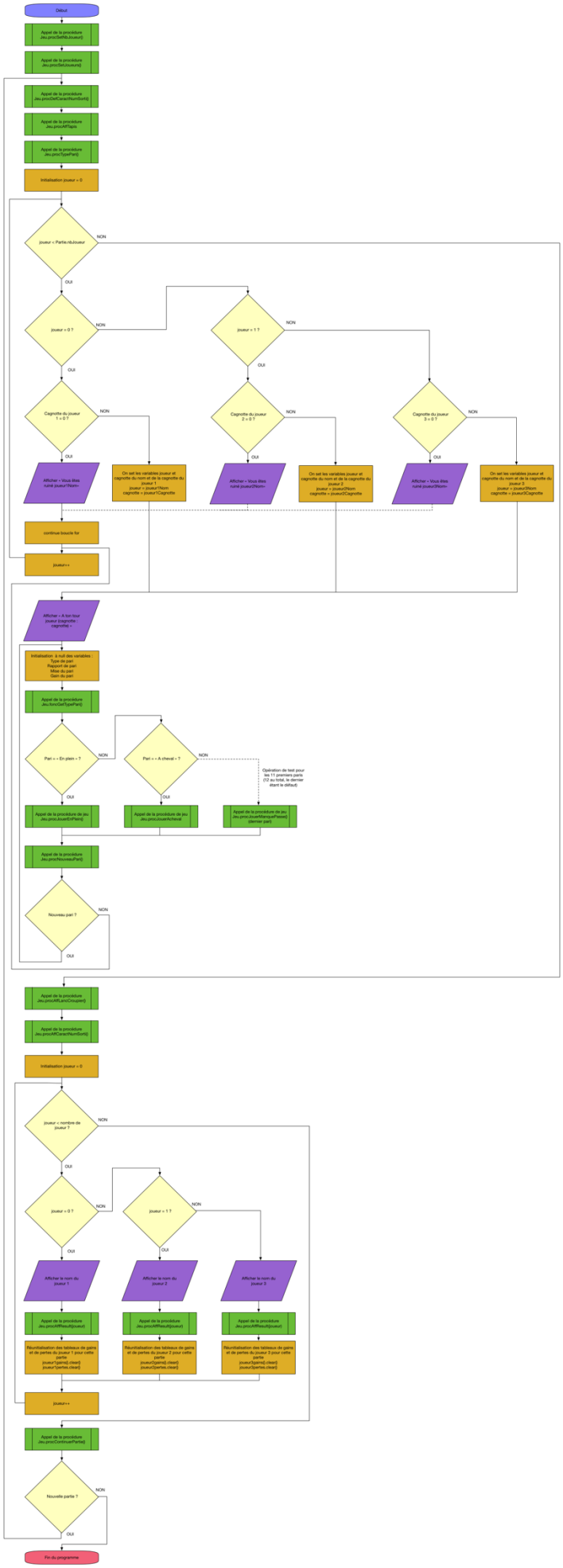
Description

La classe Main est la classe maître de l'ensemble du programme. C'est elle qui est exécutée et permet le déroulement d'une partie. Elle est découpée en 17 étapes qui permettent chacune la réalisation d'une action. L'indentation des étapes signifie l'imbrication dans des boucles. Voici ces étapes :

- 1. Paramétrer le nombre de joueurs dans la partie
- 2. Récupérer le nom de chaque joueur et set des cagnottes à 1000
- 3. Boucler sur la partie (pour rappel il est possible de jouer plusieurs parties)
    - 4. Tirer le numéro de sortie (aléatoire entre 0 et 36) et calculer ses caractéristiques (parité, couleur etc.)
    - 5. Afficher le tapis de roulette
    - 6. Afficher les types de paris
    - 7. Boucler sur le nombre d'utilisateurs
        - 8. Vérifier si la cagnotte du joueur est supérieure à 0. Si oui on continue (si non retour 7)
        - 9. Boucler sur le souhait de l'utilisateur d'effectuer plusieurs paris
            - 10. Récupérer le nom du pari souhaité par le joueur
            - 11. Jouer le pari sélectionné par le joueur
            - 12. Calcul des pertes et des gains du pari
        - 13. Demander si le joueur souhaite un nouveau pari (si oui, retour 10)
    - 14. Afficher le lancement de la bille et les annonces du croupier
    - 15. Afficher les caractéristiques du numéro sorti (suite étape 5)
    - 16. Afficher les résultats de la partie
- 17. Demander aux joueurs s'ils souhaitent une nouvelle partie (si oui, retour 4, sinon exit)

Organigramme

Pour plus de lisibilité l'organigramme est fourni en fichier PNG (voir sources).


Début
Appel de la procédure Jeu.procSetNbJoueur()
Appel de la procédure Jeu.procSetJoueurs()
Appel de la procédure Jeu.procGetCaractNumSorti()
Appel de la procédure Jeu.procAffTapis
Appel de la procédure Jeu.procTypePari()
Initialisation joueur = 0
joueur < Partie.nbJoueur
NON
OUI
joueur = 0 ?
NON
OUI
joueur = 1 ?
NON
OUI
Cagnotte du joueur 1 = 0 ?
NON
OUI
Cagnotte du joueur 2 = 0 ?
NON
OUI
Cagnotte du joueur 3 = 0 ?
NON
OUI
Afficher « Vous êtes ruiné joueur1Nom»
On set les variables joueur et cagnotte du nom et de la cagnotte du joueur 1
joueur = joueur1Nom
cagnotte = joueur1Cagnotte
Afficher « Vous êtes ruiné joueur2Nom»
On set les variables joueur et cagnotte du nom et de la cagnotte du joueur 2
joueur = joueur2Nom
cagnotte = joueur2Cagnotte
Afficher « Vous êtes ruiné joueur3Nom»
On set les variables joueur et cagnotte du nom et de la cagnotte du joueur 3
joueur = joueur3Nom
cagnotte = joueur3Cagnotte
continue boucle for
joueur++
Afficher « A ton tour joueur (cagnotte : cagnotte) »
Initialisation à null des variables : Type de pari Rapport de pari Mise du pari Gain du pari
Appel de la procédure Jeu.fonctGetTypePari()
Pari = « En plein » ?
NON
OUI
Pari = « A cheval » ?
NON
OUI
Opération de test pour les 11 premiers parts (12 au cinéma le dernier étant le défaut)
Appel de la procédure de jeu Jeu.proc.JouerEnPlein()
Appel de la procédure de jeu Jeu.proc.JouerACheval
Appel de la procédure de jeu Jeu.proc.JouerManquePasse() (dernier cas)
Appel de la procédure Jeu.procNouveauPari()
Nouveau pari ?
NON
OUI
Appel de la procédure Jeu.procAffLancCroupier()
Appel de la procédure Jeu.procAffCaractNumSorti()
Initialisation joueur = 0
joueur < nombre de joueur ?
NON
OUI
joueur = 0 ?
NON
OUI
joueur = 1 ?
NON
OUI
Afficher le nom du joueur 1
Afficher le nom du joueur 2
Afficher le nom du joueur 3
Appel de la procédure Jeu.procAffResultJoueur()
Appel de la procédure Jeu.procAffResultJoueur()
Appel de la procédure Jeu.procAffResultJoueur()
Réunitialisation des tableaux de gains et de pertes du joueur 1 pour cette partie
joueur1game[].clear()
joueur1perte.clear()
Réunitialisation des tableaux de gains et de pertes du joueur 2 pour cette partie
joueur2game[].clear()
joueur2perte.clear()
Réunitialisation des tableaux de gains et de pertes du joueur 3 pour cette partie
joueur3game[].clear()
joueur3perte.clear()
joueur++
Appel de la procédure Jeu.procContinuerPartie()
Nouvelle partie ?
NON
OUI
Fin du programme

Classe Partie

Description

La classe Partie contient les variables relatives à la définition d'une partie. Ces variables privées sont au nombre de 4.

Variables

- **nbJoueur** : entier contenant le nombre de joueurs de la partie
- **nbPartie** : entier jouant le rôle de compteur de tours d'une partie. Celui-ci est incrémenté à chaque fois que le joueur indique vouloir continuer la partie
- **caractNumSorti[]** : tableau de 6 entiers contenant les éléments suivant :
    1. index 0 : numéro de sortie aléatoire entre 0 et 36
    2. index 1 : numéro de colonne d'appartenance du numéro sorti (de 1 à 3)
    3. index 2 : numéro de douzaine d'appartenance du numéro sorti (de 1 à 3)
    4. index 3 : numéro de couleur (0 – noir, 1 – rouge)
    5. index 4 : numéro de parité (0 – pair, 1 – impair)
    6. index 5 : numéro manque / passe (0 – manque, 1 – passe)
- **continuerPartie** : booléen permettant de déterminer si le joueur décide de continuer la partie (True si on continue la partie, False sinon)

Accesseurs et mutateurs

Ces variables sont déclarées de manière privée et sont donc accessibles uniquement dans la classe Partie. Afin de pouvoir accéder à ces variables en lecture et en écriture depuis l'extérieur, des accesseurs et mutateurs ont été définis au travers de méthodes qui sont les suivantes :

- **getNbJoueur()** : accesseur qui retourne le nombre de joueurs contenu dans la variable nbJoueur
- **setNbJoueur(int nbJoueur)** : mutateur qui paramètre la variable nbJoueur avec l'entier passé en paramètre
- **getNbPartie()** : accesseur qui retourne la valeur contenue dans la variable nbPartie
- **setNbPartie(int nbPartie)** : mutateur qui paramètre la variable nbPartie avec l'entier passé en paramètre
- **getCaractNumSorti()** : accesseur qui retourne le tableau complet caractNumSorti[]
- **getCaractNumSorti(int index)** : accesseur qui retourne la valeur de l'élément contenu dans le tableau caractNumSorti[] à l'index fourni en paramètre
- **setCaractNumSorti(int index, int num)** : mutateur qui paramètre num à l'index « index » dans le tableau caractNumSorti[]
- **isContinuerParti()** : accesseur qui retourne la valeur du booléen continuerPartie
- **setContinuerParti(boolean continuerParti)** : mutateur qui paramètre la variable continuerPartie avec la valeur booléenne fournie en paramètre (True ou False)

Nous n'avons pas abordé la notion d'accesseur et de mutateur (getter and setter) dans le cadre de NFA031. Lors de mes recherches j'ai pu prendre connaissance d'un certain nombre de bonnes pratiques. Il est en effet déconseillé de rendre public les variables de classe définies plus haut afin de garder un contrôle sur la manipulation de celles-ci. C'est dans un but purement formateur que j'ai décidé d'utiliser ces notions.

Classe Joueur

Description

Cette classe est à l'image de la classe Partie. Elle ne contient que les variables relatives aux joueurs ainsi que leurs accesseurs et mutateurs correspondants.

Variables

- **joueur1Nom** : chaine de caractère contenant le nom du joueur 1
- **joueur1Cagnotte** : entier contenant la cagnotte du joueur 1
- **joueur1gain** = HashTable contenant les gains du joueur 1. Ce HashTable est construit sous la forme d'une clef de chaine de caractères contenant le nom du pari à laquelle est associé son gain (mise * rapport de gain du pari).
- **joueur1perte** = HashTable contenant les pertes du joueur1. Ce HashTable est construit sous la forme d'une clef de chaine de caractères contenant le nom du pari à laquelle est associée sa perte (égale à la mise effectuée sur le pari)

Accesseurs et mutateurs

- **getJoueur1Nom()** : retourne le nom du joueur 1
- **setJoueur1Nom(String joueur1Nom)** : paramètre le nom du joueur 1 avec la chaine de caractères passée en paramètre
- **getJoueur1Cagnotte()** : retourne la valeur de la cagnotte du joueur 1
- **setJoueur1Cagnotte(int joueur1Cagnotte)** : paramètre la cagnotte du joueur 1 avec l'entier passé en paramètre
- **Hashtable<String, Integer> getJoueur1gain()** : retourne le HashTable contenant les gains par type de pari
- **setJoueur1gain(String clef, int valeur)** : permet d'ajouter une entrée dans le HashTable des gains avec le nom de pari et la valeur des gains passée en paramètre. Si le pari n'existe pas, on ajoute une entrée. S'il existe, on ajoute à la valeur du gain existant la nouvelle valeur de gain passée en paramètre
- **Hashtable<String, Integer> getJoueur1perte()** : retourne le HashTable contenant les pertes par type de pari
- **getJoueur1perte(String clef)** : retourne la valeur de perte du pari passé en paramètre
- **setJoueur1perte(String clef, int valeur)** : permet d'ajouter une entrée dans le HashTable des pertes avec le nom de pari et la valeur des pertes passée en paramètre. Si le pari n'existe pas, on ajoute une entrée. S'il existe, on ajoute à la valeur de la perte existante la nouvelle valeur de mise passée en paramètre

Ces variables, accesseurs et mutateurs sont dupliqués pour pouvoir gérer 3 joueurs.

Classe Pari

Description

Cette classe est la dernière classe construite sur le modèle des deux précédentes. Elle ne contient que les variables relatives aux paris ainsi que leurs accesseurs et mutateurs correspondants.

Variables

- **typePari** : chaine de caractères contenant le type de pari (« En plein », « A cheval » etc.)
- **joueur** : chaine de caractères contenant le nom du joueur en cours
- **rapport** : entier du rapport de gain (ex : 35 pour un pari en plein)
- **mise** : entier contenant la mise du joueur en cours
- **gain** : entier contenant le gain du joueur en cours
- **cagnotte** : entier contenant le montant de la cagnotte du joueur en cours
- **nouveauPari** : booléen à True si le joueur souhaite réaliser un nouveau pari, False s'il ne souhaite plus parier.

Accesseurs et mutateurs

- **getTypePari()** : retourne le type de pari en cours
- **setTypePari(String typePari)** : paramètre la variable typePari avec la chaine de caractères passée en paramètre
- **getJoueur()** : retourne le nom du joueur courant
- **setJoueur()** : paramètre la variable joueur avec la chaine de caractères passée en paramètre
- **getRapport()** : retourne la valeur de la variable rapport
- **setRapport(int rapport)** : paramètre la variable rapport avec l'entier passé en paramètre
- **getMise()** : retourne la valeur de la variable mise
- **setMise(int mise)** : paramètre la variable mise avec l'entier passé en paramètre
- **getGain()** : retourne la valeur de la variable gain
- **setGain(int gain)** : parametre la variable gain avec l'entier passé en paramètre
- **getCagnotte()** : retourne la valeur de la variable cagnotte
- **setCagnotte(int cagnote)** : paramètre la valeur de la cagnotte avec l'entier passé en paramètre
- **isNouveauPari()** : retourne la valeur du booléen nouveauPari
- **setNouveauPari(boolean nouveauPari)** : paramètre la variable nouveauPari avec la valeur True ou False passée en paramètre

Précisions importantes

A ce stade il est possible de se demander pourquoi créer des variables de gain, de type de pari, ou même de cagnotte étant donné que chaque joueur possède ses propres variables. Il est à noter que dans la conception du programme, la partie qui gère la prise de pari est générique, c'est à dire qu'elle ne concerne aucun utilisateur en particulier.

Il existe avant la prise de pari une condition définissant le joueur actuellement en train de jouer.

Cette condition permet de paramétrer le montant de la variable cagnotte et la variable joueur avec respectivement la valeur de la cagnotte du joueur en cours et le nom du joueur en cours.

Pour terminer, ceci explique pourquoi ces valeurs sont réinitialisées à chaque nouveau pari, exceptée la variable cagnotte qui elle doit toujours servir d'indication au joueur. En effet la variable cagnotte est initialisée avant le premier pari du joueur avec la valeur de sa cagnotte personnelle. Dans le cas où le

joueur effectue plusieurs paris, cette cagnotte doit toujours comptabiliser le total de la mise de tous les paris du joueur, c'est pourquoi elle ne peut être réinitialisée tant que le joueur ne termine pas ses paris.

Classe Util

Description

Cette classe contient des fonctions et une procédure qui permettent d'effectuer diverses opérations.

Fonctions et procédure

- **foncIsInt(String chaine, int base)** : cette fonction prend en paramètre une chaine de caractères et un entier. Cette fonction retourne un booléen à True si la chaine représente un entier, False sinon. L'entier base représente la base sur laquelle doit être testée chaine (dans le cadre de ce programme, entier en base 10 uniquement). La vérification se fait à l'aide de la méthode « digit » de la classe « Character ».
- **foncAreInts(String stringTab[], int base)** : cette fonction utilise la fonction précédente mais le premier paramètre contient un tableau de chaine de caractères. La fonction renvoie True si tous les éléments du tableau sont des entiers.
- **foncIsInRange(int num, int x, int y)** : cette fonction prend en paramètre trois entiers. Elle a pour objectif de retourner True si la valeur num est comprise entre x et y. Sinon la valeur de retour est False.
- **foncAreInRange(int intTab[], int x, int y)** : cette fonction utilise la fonction précédente mais le premier paramètre contient un tableau d'entier. La fonction renvoie True si tous les éléments du tableau sont compris dans l'intervalle x – y, False dans le cas contraire.
- **foncStringTabToIntTab(String stringTab[])** : cette fonction prend en paramètre un tableau de chaine de caractères et retourne un tableau avec ces caractères convertis en entiers.
- **procWait(int temps)** : cette procédure prend en paramètre un entier qui correspond à un nombre de secondes. La procédure a pour objectif d'observer un temps de pause de la durée passée en paramètre (en seconde) et d'afficher un point toutes les secondes observées.

Classe Saisie

Description

Cette classe est la classe fournie dans le cadre du cours pour récupérer une entrée clavier utilisateur en fonction du type de valeur attendue.

Classe Jeu

Description

La classe Jeu est une classe importante car elle a pour but de gérer tous les événements relatifs au déroulement du jeu (affichages, prises de paris, calculs des pertes et des gains etc.). Nous allons ici aborder son contenu qui est découpé en types d'opération :

- Variables (avec accesseurs)
- Affichages
- Gestion du numéro de sortie (numéro aléatoire)
- Gestion de la partie
- Gestion des paris

Pour les fonctions complexes un organigramme sera présenté afin d'expliquer leur logique d'exécution.

Variables

- **tableJeu[][]** : cette variable contient un tableau à deux dimensions qui représente le tapis de jeu de la roulette. Un accesseur getTableJeu() permet de récupérer le contenu du tableau.
- **nbRouges[]** : cette variable contient la liste des numéros qui possèdent la couleur rouge sur le tapis de jeu. Un accesseur getNbRouges() permet de récupérer le contenu du tableau.

Affichages

- **procAffTapis()** : quand cette procédure est appelée celle-ci affiche une représentation du tapis de roulette

```
                          TAPIS

    3  6  9  12  |  15  18  21  24  |  27  30  33  36  |  3eC
0   2  5  8  11  |  14  17  20  23  |  26  29  32  35  |  2eC
    1  4  7  10  |  13  16  19  22  |  25  28  31  34  |  1eC
      1 douzaine      2 douzaines       3 douzaines
```

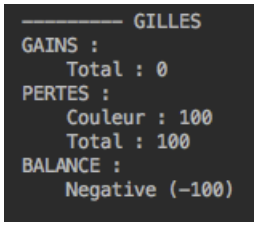
- **procTypePari()** : quand cette procédure est appelée celle-ci affiche la liste des paris qu'il est possible de réaliser

```
PARIS INTERIEURS :
    [1]  En plein (1 num)
    [2]  A cheval (2 num)
    [3]  Transversale pleine (3 num)
    [4]  Triple (3 num)
    [5]  Carré (4 num)
    [6]  Ligne haute (4 num)
    [7]  Sixain (6 num)
PARIS EXTERIEURS
    [8]  Colonne (12 num)
    [9]  Douzaine (12 num)
    [10] Couleur (18 num)
    [11] Parité (18 num)
    [12] Manque - passe (18 num)
```

- **procAffCaractNumSorti()** : cette procédure utilise la variable caractNumSorti de la classe Partie pour afficher les caractéristiques du numéro de sortie aléatoire

```
Caractéristiques du numéro sorti :
    Numéro : 19
    Colonne : 1
    Douzaine : 2
    Couleur : rouge
    Parité : impair
    Manque - passe : passe
```

- **procAffResult(int joueur)** : cette procédure affiche les résultats du numéro de joueur passé en paramètre (gains, pertes et balance)



- **procAffLancCroupier()** : cette procédure simule les annonces et le lancement de la bille par le croupier. Elle fait appel à la procédure procWait() de la classe Util pour une attente de 5 secondes.

Gestion du numéro de sortie

- **procDefCaractNumSorti()** : cette procédure effectue deux opérations principales :
    1. Tirer un numéro aléatoire entre 0 et 36
    2. Définir les caractéristiques de ce numéro et enregistrer ces caractéristiques dans la variable caractNumSorti[] de la classe Partie (voir détail de caractNumSorti[] pour signification des valeurs)

  Cette procédure utilise les cinq fonctions décrites ci-dessous pour définir ces caractéristiques.

  Cas particulier : si le numéro aléatoire est 0, toutes les valeurs de caractNumSorti[] sont initialisées à 0 également.
- **foncQuelleColonne(int num)** : cette fonction retourne 1, 2 ou 3 selon que le numéro aléatoire sorti (paramètre num) se trouve dans la colonne 1, 2 ou 3 de la table de jeu.
- **foncQuelleDouzaine(int num)** : cette fonction retourn 1, 2 ou 3 selon que le numéro aléatoire sorti passé en paramètre appartient à la première, seconde ou troisième douzaine.
- **foncIsRouge(int num)** : cette fonction retourne True si le numéro aléatoire sorti passé en paramètre se trouve dans la variable tableau nbRouge[].
- **foncIsPair(int num)** : cette fonction retourne True selon que le numéro aléatoire sorti passé en paramètre est pair, False si le numéro est impair.
- **foncIsManque(int num)** : cette fonction retourne True selon que le numéro aléatoire sorti passé en paramètre est inférieur à 18, False si le numéro est supérieur à 18.

Gestion de la partie

- **procSetNbJoueur()** : cette procédure demande à l'utilisateur le nombre de joueurs de la partie et paramètre la variable nbJoueur de la classe partie de la valeur saisie. Un contrôle de saisie est présent et valide que l'utilisateur fournit un nombre entre 1 et 3 à l'aide de la fonction foncIsInRange de la classe Util.
- **procSetJoueurs()** : cette procédure demande à l'utilisateur de saisir tour à tour les noms des joueurs et paramètre, pour chaque joueur, son nom et sa cagnotte. Ci-dessous un organigramme décrivant le processus :


Début procSetJoueurs()
Initialisation joueur = 0
joueur < Nombre de joueurs dans la partie ?
NON
Afficher « Nom du joueur (joueur+1) : »
OUI
joueur = 0 ?
NON
joueur = 1 ?
NON
OUI
OUI
Définition du nom du joueur joueur1Nom = saisie utilisateur
Définition du nom du joueur joueur2Nom = saisie utilisateur
Définition du nom du joueur 3 joueur3Nom = saisie utilisateur
Définition du montant de la cagnotte du joueur 1 joueur1Cagnotte = 1000
Définition du montant de la cagnotte du joueur 2 joueur2Cagnotte = 1000
Définition du montant de la cagnotte du joueur 3 joueur3Cagnotte = 1000
joueur++
Fin procSetJoueurs()

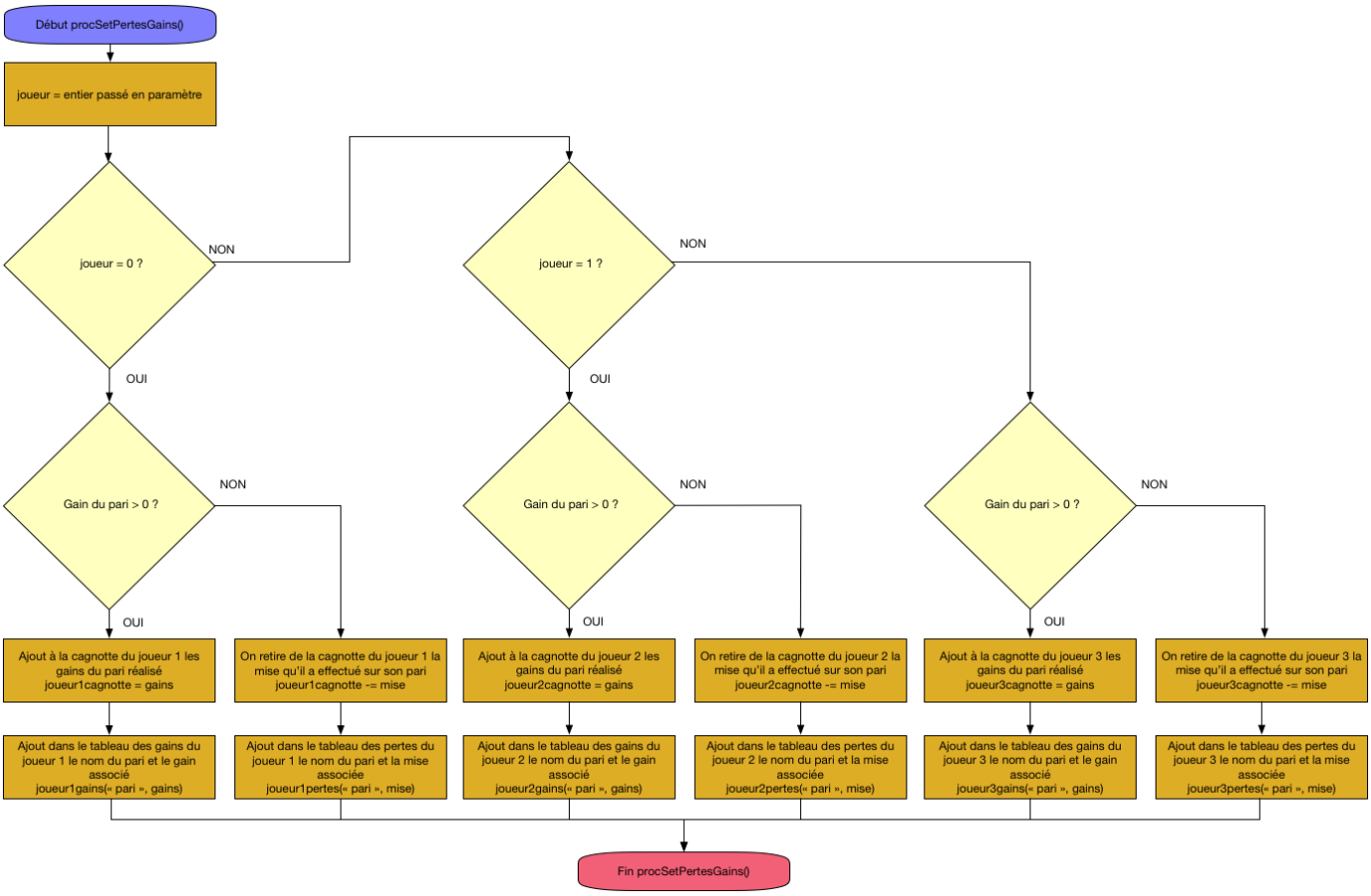
- **procContinuerPartie()** : cette procédure demande à l'utilisateur s'il souhaite continuer la partie. Si sa réponse est positive la variable continuerPartie de la classe Partie est paramétrée à True, False dans le cas contraire.

Gestion des paris

La gestion des paris est la partie qui contient toute la logique de prise de pari, de calcul des valeurs etc. En ce sens et puisque les paris suivent un même schéma de logique je présenterai l'organigramme des deux fonctions liées à la prise de pari des sixains (la plus compliquée). Une fois cette logique assimilée celle-ci est reportable sur tous les autres paris à quelques différences près.

- **procNouveauPari()** : cette procédure demande au joueur en cours si celui-ci désire réaliser un nouveau pari. Si sa réponse est positive la variable nouveauPari de la classe Pari est paramétrée à True et à False dans le cas contraire. Cette fonction procède également à la vérification du montant de la cagnotte en cours, de sorte que si cette cagnotte est égale à 0 le joueur ne puisse réaliser de nouveau pari et que la mention « Cagnotte vide, fin des paris ! » soit affichée.
- **procGetPari()** : cette procédure demande à l'utilisateur de choisir un pari sur la base d'un numéro de 1 à 12 (voir l'affichage de la fonction procAffTypePari). Ce numéro est ensuite analysé et la procédure valide qu'il est bien compris entre 1 et 12, dans le cas contraire on demande à l'utilisateur de ressaisir un numéro. Ce numéro correspond en réalité à un nom de pari et cette correspondance permet de paramétrer la variable typePari de la classe Pari de la valeur correspondante au numéro (ex : « En plein »).
- **procSetMise()** : cette procédure est appelée lors de chaque prise de pari. Elle demande à l'utilisateur combien il souhaite miser : si le montant de cette mise est inférieur à la variable cagnotte de la classe Pari (donc la cagnotte du joueur en cours) le message « Mise trop haute ! Entrez un nouveau montant » est affiché, ainsi que le montant restant de cette cagnotte. Le joueur est à nouveau invité à indiquer une mise. Lorsque cette mise est correcte la variable mise de la classe Pari est affectée de la valeur saisie par le joueur et la valeur de cette mise est soustraite à la variable cagnotte (de sorte d'afficher au joueur une cagnotte à jour s'il réalise un nouveau pari).
- **procSetPertesGains(int joueur)** : cette procédure prend en paramètre un numéro de joueur. Selon le numéro du joueur les actions s'effectueront sur le joueur 1, le joueur 2 ou le joueur 3. Les actions sont les suivantes :
    - Si la variable gain de la classe Pari est supérieure à 0, cela signifie que le pari est gagnant. On ajoute donc à sa cagnotte personnelle la valeur de ce gain (JoueurXcagnotte). La seconde opération consiste à mettre à jour le HashTable joueurXgains du joueur et d'y ajouter le nom du pari auquel on associe le gain (ex : « En plein » -> 100).
    - Si la variable gain de la classe Pari n'est pas supérieure à 0, cela signifie que le pari est perdant. Dans ce cas on soustrait à la cagnotte personnelle de l'utilisateur (joueurXcagnotte) la valeur de la variable mise de la classe Pari. La seconde opération consiste à mettre à jour le HashTable joueurXpertes du joueur et d'y ajouter le nom du pari auquel on associe la mise (ex : « En plein » -> 50).
    - **Info** : les variables utilisées ici (gain et mise) ne sont valables que pour la durée d'un pari, c'est la raison pour laquelle elles sont réinitialisées avant chaque nouveau pari (voir organigramme global)
    - **Info 2** : il est possible qu'un joueur souhaite réaliser deux fois le même pari et qu'il choisisse de miser sur les mêmes valeurs… Dans ce cas les mutateurs des HashTable JoueurXgains et JoueurXpertes mettent à jour la valeur de gain ou de perte (selon le cas) pour ne pas écraser les valeurs existantes.

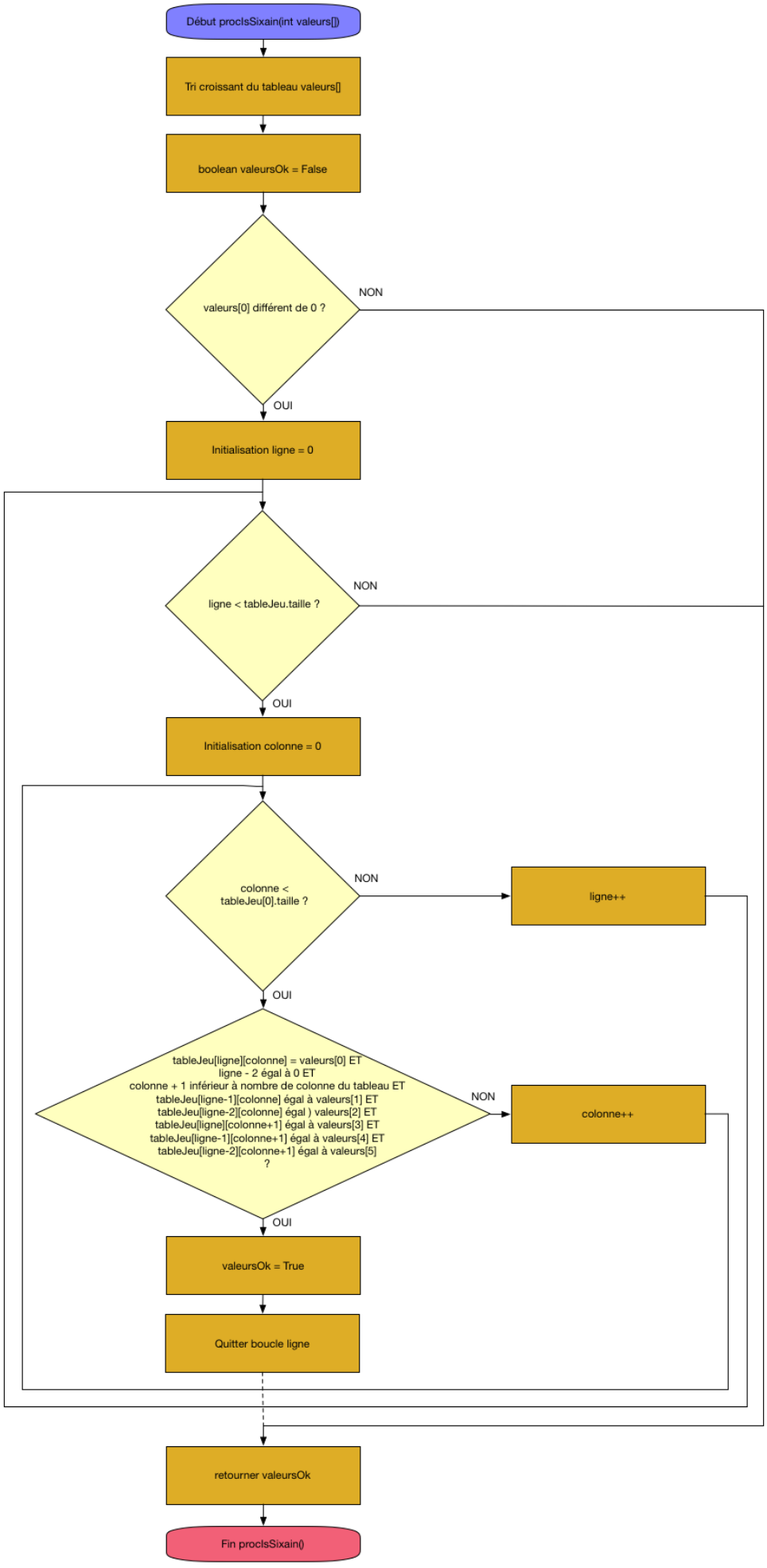
  Voici un organigramme précisant le fonctionnement de cette procédure :



- **procJouerEnPlein()** : cette procédure effectue les actions suivantes :
  - Elle paramètre la valeur de la variable mise de la classe Pari à 35
  - Elle demande à l'utilisateur de réaliser une mise (à travers l'appel de la procédure procSetMise()
  - Elle demande à l'utilisateur d'indiquer le numéro qu'il souhaite jouer
  - Si la valeur indiquée n'est pas comprise entre 0 et 36, on demande à nouveau à l'utilisateur de saisir un numéro
  - Si la valeur indiquée est comprise entre 0 et 36, on la compare avec la valeur de la variable tableau caractNumSorti[] à l'index 0 (pour rappel il s'agit du numéro aléatoire préalablement tiré).
  - Si cette valeur est égale on affecte à la variable gain de la classe Pari la valeur du rapport multiplié par la mise.
  - Si la valeur n'est pas égale, on ne fait rien (rappel : avant chaque pari la variable gain est initialisée à 0 et le calcul des pertes et des gains se fait sur cette valeur : > 0 le pari est gagnant, < 0 le pari est perdant)
- **procJouerAcheval()** : voir procJouerSixain() (rapport : 17, nbNumbers : 2)
- **foncIsAcheval(int valeurs[])** : voir foncIsSixain(int valeurs[])
- **procJouerTransversale()** : voir procJouerSixain() (rapport : 11, nbNumbers : 3)
- **foncIsTransversale(int valeurs[])** : voir foncIsSixain(int valeurs[])
- **procJouerCarre()** : voir procJouerSixain() (rapport : 8, nbNumbers : 4)
- **foncIsCarre(int valeurs[])** : voir foncIsSixain(int valeurs[])
- **procJouerSixain()** : les procédures procJouerAcheval(), procJouerTransversarle(), procJouerCarre() et procJouerSixain() possèdent le même schéma de fonctionnement. Je présente ici le fonctionnement des deux fonctions du pari en sixain.
  - **procJouerSixain()** :
    - Paramètre la variable rapport de la classe Pari à 5

- Initialisation d'un tableau de chaine de caractères valIstrings[]
- Initialisation d'un tableau d'entier vallints[]
- Initialisation de nbNumbers à 6 (nombre de valeurs devant être fournies par le joueur)
- Appel de la procédure procSetMise()
- Affichage de la phrase « Quels numéros de sixain joues-tu (ex : 1,2,3,4,5,6) : »
- Vérification que l'entrée clavier du joueur comporte bien 6 valeurs
- Vérification que l'utilisateur a respecté le formalisme (1,2,3,4,5,6)
- Vérification que ces valeurs sont toutes comprises entre 0 et 36
- Appel de la procédure procIsSixain() avec passage du tableau vallints[] qui va s'assurer que les numéros entrés sont des numéros valables (c.a.d qu'il s'agit bien d'un sixain)
- Vérification si le numéro aléatoire sorti se trouve bien dans le sixain indiqué par le joueur

- **foncIsSixain()** : cette fonction s'assure que les numéros entrés par l'utilisateur sont des numéros valides en termes de positionnement sur la table de jeu. Elle retourne True si le sixain est valide, False si non. Voici un organigramme expliquant son fonctionnement :


Début procIsSixain(int valeurs[])
Tri croissant du tableau valeurs[]
boolean valeursOk = False
valeurs[0] différent de 0 ?
NON
OUI
Initialisation ligne = 0
ligne < tableJeu.taille ?
NON
OUI
Initialisation colonne = 0
colonne < tableJeu[0].taille ?
NON
ligne++
OUI
tableJeu[ligne][colonne] = valeurs[0] ET
ligne - 2 égal à 0 ET
colonne + 1 inférieur à nombre de colonne du tableau ET
tableJeu[ligne-1][colonne] égal à valeurs[1] ET
tableJeu[ligne-2][colonne] égal) valeurs[2] ET
tableJeu[ligne][colonne+1] égal à valeurs[3] ET
tableJeu[ligne-1][colonne+1] égal à valeurs[4] ET
tableJeu[ligne-2][colonne+1] égal à valeurs[5]
?
NON
colonne++
OUI
valeursOk = True
Quitter boucle ligne
retourner valeursOk
Fin procIsSixain()

- **procJouerTriple()** : cette procédure réalise les actions suivantes :
    - Paramétrage de la variable rapport de la classe Pari à la valeur de 11
    - Appel de la procédure procSetMise()
    - Affichage de la mention « Quels numéros triples joues-tu ([0] 0,1,2 | [1] 0,2,3) : »
    - Vérification de la valeur entrée (0 ou 1). Si différent de 0 ou 1 on demande à l'utilisateur de renseigner à nouveau une valeur
    - Si la valeur est 0 ou 1, on teste comme suit :
        - Cas 0 : le numéro aléatoire sorti est-il égal à 0,1 ou 2 ? Si oui on affecte à la variable gain de la classe Pari la valeur du rapport multiplié par la mise
        - Cas 1 : le numéro aléatoire sorti est-il égal à 0,2 ou 3 ? Si oui on affecte à la variable gain de la classe Pari la valeur du rapport multiplié par la mise.
- **procJouerLigne()** : cette procédure réalise les actions suivantes :
    - Paramétrage de la variable rapport de la classe Pari à la valeur de 8
    - Affichage de la mention « Les numéros de la ligne du haut sont les suivants : 0,1,2,3 »
    - Appel de la procédure procSetMise()
    - Si le numéro aléatoire sorti est compris entre 0 et 3 alors on affecte à la variable gain de la classe Pari la valeur du rapport multiplié par la mise.
- **procJouerColonne()** : cette procédure réalise les actions suivantes :
    - Paramétrage de la variable rapport de la classe Pari à la valeur de 2
    - Appel de la procédure procSetMise()
    - Affichage de la mention « Quel numéro de colonne joues-tu : »
    - Si la valeur entrée par l'utilisateur n'est pas comprise entre 1 et 3 on demande à l'utilisateur de ressaisir une valeur
    - Quand la valeur saisie est correcte et si cette valeur est égale à la valeur de l'indice 1 du tableau caractNumSorti[] (numéro de colonne du numéro aléatoire sorti) alors on affecte à la variable gain de la classe Pari la valeur du rapport multiplié par la mise.
- **procJouerDouzaine()** : cette procédure fonctionne à l'identique de la précédente. Seul l'affichage et l'indice sont différents (indice 2).
- **procJouerCouleur()** : cette procédure réalise les actions suivantes :
    - Paramétrage de la variable rapport de la classe Pari à la valeur de 1
    - Appel de la procédure procSetMise()
    - Affichage de la mention « Quel numéro de colonne joues-tu : »
    - Si la valeur entrée par l'utilisateur n'est pas 0 ou 1 on demande à l'utilisateur de ressaisir une valeur
    - Quand la valeur saisie est correcte on vérifie les conditions suivantes :
        - 1. Que la valeur aléatoire sortie n'est pas égale à 0 (dans ce cas, les paris de couleur, de parité et de manque / passe sont nuls)
        - 2. Que la valeur saisie par l'utilisateur est égale à la valeur de l'indice 3 du tableau caractNumSorti[] (numéro de couleur du numéro aléatoire sorti)
    - Si ces deux conditions sont remplies alors on affecte à la variable gain de la classe Pari la valeur du rapport multiplié par la mise.
- **procJouerParite()** : cette procédure fonctionne à l'identique de la précédent hormis ce qui concerne l'affichage et l'indice de tableau (4).
- **procJouerManquePasse ()** : cette procédure fonctionne à l'identique des deux précédentes hormis ce qui concerne l'affichage et l'indice de tableau (5).

Conclusion

Difficultés

Les difficultés rencontrées tiennent principalement à la conception logique du programme. En effet j'ai tenté la réalisation des organigrammes en amont de la réalisation du code, ce qui m'a paru relativement accessible. En revanche je me suis aperçu lors du développement qu'il y avait certains éléments auxquels je n'avais pas pensé. Au final le résultat de mon programme n'avait plus grand chose à voir avec ma conception initiale.
Je pense que la conception logique au travers d'organigrammes est une bonne pratique qui devient profitable avec une certaine expérience du développement. Pour des personnes fraichement initiées je trouve difficile d'obtenir un résultat convainquant et synonyme de gain de temps.

Satisfactions

Je suis très content d'avoir réalisé ce projet. Je ne saurais comparer mon niveau en Java (et même en logique) avant la réalisation de celui-ci et aujourd'hui. J'ai trouvé cet exercice réellement formateur. Je n'ai pas rencontré de difficulté particulière lors de la programmation et j'ai apprécié Java (j'ai quelques points de comparaison avec d'autres langages, ruby par exemple). Bien sûr j'ai aussi pu me rendre compte qu'il existe des langages ou tout est plus facile (notamment au niveau des calculs !) mais cela reste anecdotique. J'ai également pu intégrer des éléments que nous n'avions pas abordé ce qui pourra me servir par la suite (HashTable et leur itération, accesseurs et mutateurs etc.).
Je suis globalement satisfait du travail réalisé et que je vous présente dans ce document.

En somme, une très bonne expérience !

Points d'amélioration

J'avoue ressentir une certaine frustration au niveau des choix faits pour la réalisation de ce programme mais je n'ai pas trouvé de réponses me permettant d'envisager d'autres solutions. Par exemple, le mode multijoueur implique pour moi la création de variables dupliquées et nominatives pour chaque joueur car il n'est pas possible de créer de variable dynamiquement (basé sur le nombre de joueurs de la partie par exemple). Ce mode multijoueur ne concerne que trois joueurs mais je me demande quelles pourraient être les solutions pour créer un programme avec X joueurs.

Annexes

Règles du jeu de la roulette française (ou européenne)

Le jeu de la Roulette Française, également appelée Roulette Européenne, se déroule sur une table de jeu. Celle-ci est composée à une extrémité d'une roulette, et à l'autre d'un tapis de jeu. La roue est un plateau tournant (le plus souvent en laiton massif, finement usiné), inséré dans une cuvette en bois marqueté. La cuvette porte sur sa face interne des chicanes en laiton, destinées à rendre plus imprévisible la chute de la bille.
La cuvette de la Roulette Française est divisée en 37 cases, numérotées de 0 à 36. Chaque numéro possède une couleur, noire ou rouge, excepté pour le zéro qui est de couleur verte. La bille, à l'origine en ivoire, aujourd'hui en résine ou en téflon, est lancée par le croupier dans le sens inverse de la rotation de la roulette. Le sens du lancer de la boule - c'est ainsi qu'on l'appelle - et de la rotation du plateau, alternent entre chaque tour (chaque numéro sorti).

Ce jeu est fascinant par l'excitation suscitée à suivre la boule tourner dans le cylindre, pour finalement s'immobiliser dans une des cases de la roulette. L'attrait de ce jeu réside également dans la diversité des paris pouvant être lancés.

But du jeu :

Le but du jeu est d'arriver à anticiper le numéro qui va sortir en y positionnant des mises. Pour cela, il est possible d'optimiser son jeu, afin d'augmenter les probabilités de gagner, et d'avoir parié sur le bon numéro. De nombreuses techniques de jeu ont été mises en place, pour permettre au joueur de tenter sa chance et décrocher un jackpot en minimisant ses pertes. Ces techniques s'appellent les martingales.

Déroulement d'un tour de jeu :

"Faites vos jeux"

A l'annonce de cette phrase par le croupier, tous les joueurs peuvent poser leurs mises sur le tapis. Cette phase est terminée lorsque tous les joueurs ont déposé leurs mises. Pour miser, le joueur peut déposer ses jetons soit dans une seule case, soit à cheval sur deux, trois ou quatre en fonction des types de paris (voir ci-dessous).

"Rien ne va plus"

Lorsque le croupier annonce cette phrase, il lance la roue dans le sens inverse du tour précèdent, et envoie la bille à contre-courant dans le cylindre. A ce moment, sur certaines tables, les retardataires peuvent encore jouer très rapidement car cela reste souvent tolèré, mais il faut savoir que cela peut être également refusé par le chef de table.

"Les jeux sont faits"

A cette annonce, il est impératif de ne plus miser. C'est le moment où la bille entre en contact avec les cases de la roue, et va commencer à rebondir de case en case. Une fois la bille immobilisée dans l'une des 37 cases, le croupier annonce le numéro sorti ainsi que les trois groupes d'égalité auquel il appartient (voir ci-dessous). C'est à ce moment que les paiements commencent, en fonction des mises de chacun.

Les différents types de paris :

Les types de paris possibles à la Roulette Française sont très variés, et c'est ce qui fait le charme de ce jeu de casino.
Il y a deux grands types de paris à la roulette : les paris dits "intérieurs", qui sont les mises placées à l'intérieur de la grille de numéros imprimée, et les paris dits "extérieurs", qui sont les mises placées à l'extérieur de cette grille.

Les paris intérieurs

En plein : la mise est positionnée sur un numéro individuel
A cheval : la mise est placée sur la ligne entre deux numéros, permettant de jouer deux numéros en même temps
Transversale pleine : la mise est placée sur le bord gauche d'une ligne de numéros, et permet de jouer toute la ligne
Triple : la mise est placée à l'intersection de trois numéros qui se touchent. Il est possible de faire des paris Triple uniquement pour les numéros 0, 1 et 2, ou bien pour 2, 3, 0
En carré : la mise est positionnée sur l'intersection de quatre numéros
Ligne du haut : la mise est placée à gauche, à l'intersection de la première ligne verticale de la grille de numéros et de la ligne horizontale séparant le 0 des 1, 2, 3, pour jouer ces quatre premiers numéros en haut de la grille
Sixain : la mise est positionnée à cheval entre deux lignes de numéros, sur la ligne de gauche, et permet de jouer avec une seule mise deux Transversales pleines.

Les paris extérieurs

Colonnes : la mise est placée en bas de la colonne et à l'extérieur de la grille pour miser une colonne verticale entière dans la grille. Attention, une colonne n'inclut pas le 0
Douzaines : la mise est placée sur une des cases des douzaines situées à l'extérieur de la grille, pour miser sur les nombres allant de 1 à 12, de 13 à 24 ou bien de 25 à 36, et ne comprend pas le zéro. Il est possible de placer la mise à cheval sur deux douzaines
Rouge / Noir : la mise est placée sur une des deux cases de couleur à l'extérieur de la grille, pour miser sur la couleur du numéro qui sortira
Pair / Impair : la mise est placée sur une des deux cases "Pair" ou "Impair", pour miser sur le fait que le numéro qui sortira soit pair ou impair
Manque / Passe (ou Low Half / High Half) : la mise est placée sur une des deux cases "Manque" ou "Passe", pour miser sur la moitié inferieure (Low Half / Manque, de 1 à 18) ou sur la moitié supérieure (High Half / Passe, de 19 à 36).

Paiement des gains

Paris Intérieurs	**Rapports**
En Plein (nombre simple)	35 pour 1
A Cheval (deux numéros)	17 pour 1
Transversale Pleine (trois numéros)	11 pour 1
Triple (trois nombres)	11 pour 1
En Carré (quatre numéros)	8 pour 1
Ligne du haut (5 premiers nombres : 0, 1, 2, 3)	8 pour 1
Sixain (six numéros en deux colonnes transversales)	5 pour 1

Paris Extérieurs	**Rapports**
1ère, 2ème ou 3ème Colonne (12 numéros)	2 pour 1
1ère, 2ème ou 3ème Douzaine (1st, 2nd or 3rd Dozen)	2 pour 1
Rouge ou Noir (Red or Black)	1 pour 1
Even ou Odd (Paire ou Impaire)	1 pour 1
Manque / Low Half (de 1 à 18) ou Passe / High Half (de 19 à 36)	1 pour 1

Code source

Classe Main

```
public class Main {
  public static void main(String args[]) {
    /****************************** 1. PARAMETRER LE NOMBRE DE JOUEURS DANS LA PARTIE */
    Jeu.procSetNbJoueur();

    /****************************** 2. RECUPERER LE NOM DE CHAQUE JOUEUR ET SET DES CAGNOTTES A 1000 */
    Jeu.procSetJoueurs();

    /****************************** 3. BOUCLER SUR VALIDITE DU BOOLEAN continuerPartie (CLASSE PARTIE) */
    do {
      /****************************** 4. TIRER LE NUMERO DE SORTIE ET CALCULER SES CARACTERISTIQUES */
      Jeu.procDefCaractNumSorti();

      /****************************** 5. AFFICHER LE TAPIS DE ROULETTE */
      Jeu.procAffTapis();

      /****************************** 6. AFFICHER LES TYPES DE PARI */
      Jeu.procTypePari();

      /****************************** 7. BOUCLER SUR LE NOMBRE D'UTILISATEURS DANS LA PARTIE */
      loopJoueur : for (int joueur = 0; joueur < Partie.getNbJoueur(); joueur++) {

        /****************************** 8. VERIFIER SI CAGNOTTE > 0. SET DE JOUEUR ET CAGNOTTE AVEC JOUEUR COURANT */
        switch (joueur) {
          case 0: {
            if (Joueur.getJoueur1Cagnotte() == 0) {
              System.out.println("\nVous êtes ruiné "+Joueur.getJoueur1Nom()+" ! Joueur suivant...\n");
              continue loopJoueur;
            } else {
              Pari.setJoueur(Joueur.getJoueur1Nom());
              Pari.setCagnotte(Joueur.getJoueur1Cagnotte());
            }
            break;
          }
          case 1: {
            if (Joueur.getJoueur2Cagnotte() == 0) {
              System.out.println("\nVous êtes ruiné "+Joueur.getJoueur2Nom()+" ! Joueur suivant...\n");
              continue loopJoueur;
            } else {
              Pari.setJoueur(Joueur.getJoueur2Nom());
              Pari.setCagnotte(Joueur.getJoueur2Cagnotte());
            }
            break;
          }
          case 2: {
            if (Joueur.getJoueur3Cagnotte() == 0) {
              System.out.println("\nVous êtes ruiné "+Joueur.getJoueur3Nom()+" ! Joueur suivant...\n");
              continue loopJoueur;
            } else {
              Pari.setJoueur(Joueur.getJoueur3Nom());
              Pari.setCagnotte(Joueur.getJoueur3Cagnotte());
            }
            break;
          }
        }

        // On affiche "A ton tour -nom du joueur courant- ! cagnotte (-cagnotte du joueur courant-)"
        System.out.println("\nA ton tour "+Pari.getJoueur()+" ! (cagnotte : "+Pari.getCagnotte()+")\n");

        /****************************** 9. TANT QUE LE JOUEUR SOUHAITE FAIRE UN NOUVEAU PARI, ON BOUCLE */
        do {
          Pari.setTypePari(null);                          // Reinit du nom de pari
          Pari.setRapport(0);                              // Reinit du rapport de pari
          Pari.setMise(0);                                 // Reinit de la mise de pari
          Pari.setGain(0);                                 // Reinit des gains du pari

          /****************************** 10. RECUPERER LE NOM DU PARI */
          Jeu.foncGetPari();

          /****************************** 11. JOUER LES PARIS */
          // Jouer les paris
          switch (Pari.getTypePari()) {
            case "En plein": Jeu.procJouerEnPlein(); break;
            case "A cheval": Jeu.procJouerAcheval(); break;
            case "Transversale pleine": Jeu.procJouerTransversale(); break;
            case "Triple": Jeu.procJouerTriple(); break;
            case "Carré": Jeu.procJouerCarre(); break;
            case "Ligne haute": Jeu.procJouerLigne(); break;
            case "Sixain": Jeu.procJouerSixain(); break;
            case "Colonne": Jeu.procJouerColonne(); break;
            case "Douzaine": Jeu.procJouerDouzaine(); break;
            case "Couleur": Jeu.procJouerCouleur(); break;
            case "Parité": Jeu.procJouerParite(); break;
            case "Manque - passe": Jeu.procJouerManquePasse(); break;
          }

          /****************************** 12. CALCUL DES PERTES ET DES GAINS */
          Jeu.procSetPertesGains(joueur);
```

```
        /************************ 13. DEMANDER SI LE JOUEUR SOUHAITE UN NOUVEAU PARI */
        Jeu.procNouvauPari();
      } // Fin récupération des paris du joueur en cours
      while (Pari.isNouveauPari());
    } // Fin loopJoueur

    /*************************** 14. AFFICHER LE LANCEMENT DE LA BILLE */
    Jeu.procAffLancCroupier();

    /*************************** 15. AFFICHER LES CARACTERISTIQUES DU NUMERO SORTI */
    Jeu.procAffCaractNumSorti();

    /*************************** 16. AFFICHER LES RESULTATS DE LA PARTIE */
    System.out.println("\n************************ RESULTATS ************************\n");

    for (int joueur = 0; joueur < Partie.getNbJoueur(); joueur++) {
      switch (joueur) {
        case 0: {
          System.out.println("--------- " + Joueur.getJoueur1Nom());

          Jeu.procAffResult(joueur);

          Joueur.getJoueur1gain().clear();
          Joueur.getJoueur1perte().clear();
          break;
        }
        case 1: {
          System.out.println("--------- " + Joueur.getJoueur2Nom());

          Jeu.procAffResult(joueur);

          Joueur.getJoueur2gain().clear();
          Joueur.getJoueur2perte().clear();
          break;
        }
        case 2: {
          System.out.println("--------- " + Joueur.getJoueur3Nom());

          Jeu.procAffResult(joueur);

          Joueur.getJoueur3gain().clear();
          Joueur.getJoueur3perte().clear();
          break;
        }
      }
    }

    /*************************** 17. DEMANDER AU JOUEUR S'IL SOUHAITE CONTINUER LA PARTIE */
    Jeu.procContinuerPartie();
  } // Fin de la partie
  while (Partie.isContinuerParti());
 }
}
```

Classe Partie

```
public class Partie {
  private static int nbJoueur;                          // Nombre de joueurs de la partie
  private static int nbPartie = 1;                      // Compteur du nombre de parties
  private static int caractNumSorti[] = new int[6];     // Tableau contenant le numéro sorti et ses caractéristiques
  private static boolean continuerPartie;               // True si on continue la partie, false sinon

  public static int getNbJoueur() {
    return nbJoueur;
  }

  public static void setNbJoueur(int nbJoueur) {
    Partie.nbJoueur = nbJoueur;
  }

  public static int getNbPartie() {
    return nbPartie;
  }

  public static void setNbPartie(int nbPartie) {
    Partie.nbPartie = nbPartie;
  }

  public static int[] getCaractNumSorti() {
    return caractNumSorti;
  }

  public static int getCaractNumSorti(int index) { return caractNumSorti[index]; }

  public static void setCaractNumSorti(int index, int num) {
    Partie.caractNumSorti[index] = num;
  }

  public static boolean isContinuerParti() {
    return continuerPartie;
  }

  public static void setContinuerParti(boolean continuerParti) {
    Partie.continuerPartie = continuerParti;
  }
}
```

Classe Joueur

```
import java.util.Hashtable;

public class Joueur {
  // Variables relatives à chaque joueur (de 1 à 3 joueurs)
  private static String joueur1Nom = "";                                    // Nom du joueur 1
  private static int joueur1Cagnotte = 0;                                   // Cagnotte du joueur 1
  private static Hashtable<String, Integer> joueur1gain = new Hashtable<>(); // Contient gains ("Nom pari" > gains)
  private static Hashtable<String, Integer> joueur1perte = new Hashtable<>();// Contient pertes ("Nom pari" > pertes)

  private static String joueur2Nom = "";                                    // Idem pour le joueur 2
  private static int joueur2Cagnotte = 0;                                   // Idem pour le joueur 2
  private static Hashtable<String, Integer> joueur2gain = new Hashtable<>(); // Idem pour le joueur 2
  private static Hashtable<String, Integer> joueur2perte = new Hashtable<>();// Idem pour le joueur 2

  private static String joueur3Nom = "";                                    // Idem pour le joueur 3
  private static int joueur3Cagnotte = 0;                                   // Idem pour le joueur 3
  private static Hashtable<String, Integer> joueur3gain = new Hashtable<>(); // Idem pour le joueur 3
  private static Hashtable<String, Integer> joueur3perte = new Hashtable<>();// Idem pour le joueur 3

  /****************************** JOUEUR 1 */
  public static String getJoueur1Nom() {
    return joueur1Nom;
  }

  public static void setJoueur1Nom(String joueur1Nom) { Joueur.joueur1Nom = joueur1Nom; }

  public static int getJoueur1Cagnotte() {
    return joueur1Cagnotte;
  }

  public static void setJoueur1Cagnotte(int joueur1Cagnotte) {
    Joueur.joueur1Cagnotte = joueur1Cagnotte;
  }

  public static Hashtable<String, Integer> getJoueur1gain() {
    return joueur1gain;
  }

  public static void setJoueur1gain(String clef, int valeur) {
    if (!(Joueur.getJoueur1gain().containsKey(clef))) Joueur.joueur1gain.put(clef, valeur);
    else Joueur.joueur1gain.put(clef, Joueur.getJoueur1gain().get(clef) + valeur);
  }

  public static Hashtable<String, Integer> getJoueur1perte() {
    return joueur1perte;
  }

  public static int getJoueur1perte(String clef) { return joueur1perte.get(clef); }

  public static void setJoueur1perte(String clef, int valeur) {
    if (!(Joueur.getJoueur1perte().containsKey(clef))) Joueur.joueur1perte.put(clef, valeur);
    else Joueur.joueur1perte.put(clef, Joueur.getJoueur1perte(clef) + valeur);
  }

  /****************************** JOUEUR 2 */
  public static String getJoueur2Nom() {
    return joueur2Nom;
  }

  public static void setJoueur2Nom(String joueur2Nom) {
    Joueur.joueur2Nom = joueur2Nom;
  }

  public static int getJoueur2Cagnotte() {
    return joueur2Cagnotte;
  }

  public static void setJoueur2Cagnotte(int joueur2Cagnotte) {
    Joueur.joueur2Cagnotte = joueur2Cagnotte;
  }

  public static Hashtable<String, Integer> getJoueur2gain() {
    return joueur2gain;
  }

  public static void setJoueur2gain(String clef, int valeur) {
    if (!(Joueur.getJoueur2gain().containsKey(clef))) Joueur.joueur2gain.put(clef, valeur);
    else Joueur.joueur2gain.put(clef, Joueur.getJoueur2gain().get(clef) + valeur);
  }

  public static Hashtable<String, Integer> getJoueur2perte() {
    return joueur2perte;
  }

  public static int getJoueur2perte(String clef) { return joueur2perte.get(clef); }

  public static void setJoueur2perte(String clef, int valeur) {
    if (!(Joueur.getJoueur2perte().containsKey(clef))) Joueur.joueur2perte.put(clef, valeur);
    else Joueur.joueur2perte.put(clef, Joueur.getJoueur2perte(clef) + valeur);
  }
```

```
  /***************************** JOUEUR 3 */
  public static String getJoueur3Nom() {
    return joueur3Nom;
  }

  public static void setJoueur3Nom(String joueur3Nom) {
    Joueur.joueur3Nom = joueur3Nom;
  }

  public static int getJoueur3Cagnotte() {
    return joueur3Cagnotte;
  }

  public static void setJoueur3Cagnotte(int joueur3Cagnotte) {
    Joueur.joueur3Cagnotte = joueur3Cagnotte;
  }

  public static Hashtable<String, Integer> getJoueur3gain() {
    return joueur3gain;
  }

  public static void setJoueur3gain(String clef, int valeur) {
    if (!(Joueur.getJoueur3gain().containsKey(clef))) Joueur.joueur3gain.put(clef, valeur);
    else Joueur.joueur3gain.put(clef, Joueur.getJoueur3gain().get(clef) + valeur);
  }

  public static Hashtable<String, Integer> getJoueur3perte() {
    return joueur3perte;
  }

  public static int getJoueur3perte(String clef) { return joueur3perte.get(clef); }

  public static void setJoueur3perte(String clef, int valeur) {
    if (!(Joueur.getJoueur3perte().containsKey(clef))) Joueur.joueur3perte.put(clef, valeur);
    else Joueur.joueur3perte.put(clef, Joueur.getJoueur3perte(clef) + valeur);
  }
}
```

Classe Pari

```
public class Pari {
  private static String typePari;                  // Nom du pari
  private static String joueur;                    // Nom du joueur courant
  private static int rapport;                      // Rapport du gain (35 contre 1 etc.)
  private static int mise;                         // Mise réalisée par le joueur
  private static int gain;                         // Gain réalisé par le joueur
  private static int cagnotte;                     // Cagnotte du joueur courant
  private static boolean nouveauPari;              // True si le joueur souhaite faire un nouveau pari

  public static String getTypePari() {
    return typePari;
  }

  public static void setTypePari(String typePari) {
    Pari.typePari = typePari;
  }

  public static String getJoueur() { return joueur; }

  public static void setJoueur(String joueur) { Pari.joueur = joueur; }

  public static int getRapport() {
    return rapport;
  }

  public static void setRapport(int rapport) {
    Pari.rapport = rapport;
  }

  public static int getMise() {
    return mise;
  }

  public static void setMise(int mise) { Pari.mise = mise; }

  public static int getGain() {
    return gain;
  }

  public static void setGain(int gain) { Pari.gain = gain; }

  public static int getCagnotte() {
    return cagnotte;
  }

  public static void setCagnotte(int cagnotte) {
    Pari.cagnotte = cagnotte;
  }

  public static boolean isNouveauPari() {
    return nouveauPari;
  }

  public static void setNouveauPari(boolean nouveauPari) {
    Pari.nouveauPari = nouveauPari;
  }
}
```

Classe Util

```
public class Util {
  /****************************** VERIFIER SI LA VALEUR FOURNIE EST BIEN UN INT */
  public static boolean foncIsInt(String chaine, int base) {
    if (chaine.isEmpty()) return false;

    for (int i = 0; i < chaine.length(); i++) {
      if (i == 0 && chaine.charAt(i) == '-') {
        if (chaine.length() == 1) return false;
        else continue;
      }
      if (Character.digit(chaine.charAt(i), base) < 0) return false;
    }
    return true;
  }

  /****************************** VERIFIER SI TOUTES LES VALEURS DU TABLEAU FOURNI SONT DES INT */
  public static boolean foncAreInts(String stringTab[], int base) {
    boolean isInt = false;

    for (String chaine : stringTab) {
      if (foncIsInt(chaine, base)) isInt = true;
      else isInt = false;
    }
    return isInt;
  }

  /****************************** VERIFIER QUE LE NUMERO FOURNI SE TROUVE ENTRE X ET Y */
  public static boolean foncIsInRange(int num, int x, int y) {
    if (num  >= x && num <= y) return true;
    else return false;
  }

  /****************************** VERIFIER QUE LES VALEURS DU TABLEAU FOURNI SONT ENTRE X ET Y */
  public static boolean foncAreInRange(int intTab[], int x, int y) {
    boolean isInRange = false;

    for (int val : intTab) {
      if (foncIsInRange(val, x, y)) isInRange = true;
      else isInRange = false;
    }
    return isInRange;
  }

  /****************************** TRANSFORMER UN TABLEAU DE STRING EN TABLEAU DE INT */
  public static int[] foncStringTabToIntTab(String stringTab[]) {
    int intTab[] = new int[stringTab.length];

    for (int i = 0; i < intTab.length; i++) intTab[i] = Integer.parseInt(stringTab[i]);
    return intTab;
  }

  /****************************** ATTENDRE LE TEMPS FOURNI EN PARAMETRE (EN SECONDE) */
  public static void procWait(int i_temps) {
    for (int i = 0; i < i_temps; i++) {
      System.out.print(".");
      try { Thread.sleep(1000); }
      catch (InterruptedException ie) { }
    }
  }
}
```

Classe Saisie

```java
import java.util.Scanner;

public class Saisie {
  // STRING
  public static String foncLireString() {
    String chaine = null;
    try {
      Scanner input = new Scanner(System.in);
      chaine = input.nextLine();
    }
    catch (NumberFormatException e) {System.err.println(e);}
    return chaine;
  }

  // CHAR
  public static char foncLireChar() {
    String chaine = foncLireString();
    return chaine.charAt(0);
  }

  // BYTE
  public static byte lire_byte() {
    return Byte.parseByte(foncLireString());
  }

  // SHORT
  public static short lire_short() { return Short.parseShort(foncLireString()); }

  // INT
  public static int foncLireInt(int base) {
    Scanner input = new Scanner(System.in);
    String chaine = input.nextLine();
    boolean isInt = Util.foncIsInt(chaine, base);

    while (!(isInt)) {
      System.out.print("Saisie incorrect, caractères non conformes : ");
      chaine = input.nextLine();
      isInt = Util.foncIsInt(chaine, base);
    }
    return Integer.parseInt(chaine);
  }

  // SHORT
  public static long foncLireLong() { return Long.parseLong(foncLireString()); }

  // FLOAT
  public static float foncLireFloat() {
    return Float.parseFloat(foncLireString());
  }

  // DOUBLE
  public static double foncLireDouble() {
    return Double.parseDouble(foncLireString());
  }
}
```

Classe Jeu

```
import java.util.Hashtable;
import java.util.stream.IntStream;
import java.util.Set;
import java.util.Arrays;

public class Jeu {
  /*****************************************************************************************************************
   *                                                VARIABLES
   *****************************************************************************************************************/

  /****************************** TABLEAU 2 DIMENSIONS REPRESENTANT LA TABLE DE JEU */
  private static int tableJeu[][] = {
          {3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36},
          {2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35},
          {1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34}
  };
  // GETTER TABLE DE JEU
  public static int[][] getTableJeu() {
    return tableJeu;
  }

  /****************************** TABLEAU CONTENANT LA LISTE DES VALEURS POSSEDANT LA COULEUR ROUGE */
  private static int nbRouges[] = {1, 3, 5, 7, 9, 12, 14, 16, 18, 19, 21, 23, 25, 27, 30, 32, 34, 36};
  // GETTER NOMBRES ROUGES
  public static int[] getNbRouges() {
    return nbRouges;
  }

  /*****************************************************************************************************************
   *                                                AFFICHAGES
   *****************************************************************************************************************/

  /****************************** AFFICHER LE TAPIS DE JEU */
  public static void procAffTapis() {
    System.out.println("\n\t\t\t\t\t\t\tTAPIS\n");
    System.out.println("\t3  6  9  12  |  15  18  21  24  |  27  30  33  36  |  3eC");
    System.out.println("0   2  5  8  11  |  14  17  20  23  |  26  29  32  35  |  2eC");
    System.out.println("\t1  4  7  10  |  13  16  19  22  |  25  28  31  34  |  1eC");
    System.out.println("\t 1 douzaine      2 douzaines        3 douzaines\n");
  }

  /****************************** AFFICHER LES TYPES DE PARI */
  public static void procTypePari() {
    String[] tab = {
            "PARIS INTERIEURS :",
            "\t[1]  En plein (1 num)",
            "\t[2]  A cheval (2 num)",
            "\t[3]  Transversale pleine (3 num)",
            "\t[4]  Triple (3 num)",
            "\t[5]  Carré (4 num)",
            "\t[6]  Ligne haute (4 num)",
            "\t[7]  Sixain (6 num)",
            "PARIS EXTERIEURS",
            "\t[8]  Colonne (12 num)",
            "\t[9]  Douzaine (12 num)",
            "\t[10] Couleur (18 num)",
            "\t[11] Parité (18 num)",
            "\t[12] Manque - passe (18 num)",
    };

    for (String el : tab) {
      System.out.println(el);
    }
  }

  /****************************** AFFICHER LES CARACTERISTIQUES DU NUMERO SORTI */
  public static void procAffCaractNumSorti() {
    String a_trame[] = {"Numéro : ", "Colonne : ", "Douzaine : ", "Couleur : ", "Parité : ", "Manque - passe : "};

    System.out.println("Caractéristiques du numéro sorti : ");

    System.out.println("\t" + a_trame[0] + "" + Partie.getCaractNumSorti(0));

    if (Partie.getCaractNumSorti(0) == 0) {
      for (int i = 1; i < Partie.getCaractNumSorti().length; i++) {
        System.out.println("\t" + a_trame[i] + "N/A");
      }
      return;
    }
    else {
      System.out.println("\t" + a_trame[1] + "" + Partie.getCaractNumSorti(1));

      System.out.println("\t" + a_trame[2] + "" + Partie.getCaractNumSorti(2));

      if (Partie.getCaractNumSorti(3) == 1) System.out.println("\t" + a_trame[3] + "" + "rouge");
      else System.out.println("\t" + a_trame[3] + "" + "noir");

      if (Partie.getCaractNumSorti(4) == 0) System.out.println("\t" + a_trame[4] + "" + "pair");
      else System.out.println("\t" + a_trame[4] + "" + "impair");
```

```
      if (Partie.getCaractNumSorti(5) == 0) System.out.println("\t" + a_trame[5] + "" + "manque");
      else System.out.println("\t" + a_trame[5] + "" + "passe");
    }
  }

  /****************************** AFFICHER LES RESULTATS DE LA PARTIE */
  public static void procAffResult(int joueur) {
    Hashtable<String, Integer> gain = new Hashtable<>();
    Hashtable<String, Integer> perte = new Hashtable<>();
    int gainTotal = 0;
    int perteTotal = 0;

    switch (joueur) {
      case 0: {
        gain = Joueur.getJoueur1gain();
        perte = Joueur.getJoueur1perte();
        break;
      }
      case 1: {
        gain = Joueur.getJoueur2gain();
        perte = Joueur.getJoueur2perte();
        break;
      }
      case 2: {
        gain = Joueur.getJoueur3gain();
        perte = Joueur.getJoueur3perte();
        break;
      }
    }

    // Afficher les gains
    System.out.println("GAINS : ");

    Set<String> gainClefs = gain.keySet();
    for (String s_key : gainClefs) {
      gainTotal += gain.get(s_key);
      System.out.println("\t" + s_key + " : " + gain.get(s_key));
    }
    System.out.println("\tTotal : " + gainTotal);

    // Afficher les pertes
    System.out.println("PERTES : ");

    Set<String> perteClefs = perte.keySet();
    for (String s_key : perteClefs) {
      perteTotal += perte.get(s_key);
      System.out.println("\t" + s_key + " : " + perte.get(s_key));
    }
    System.out.println("\tTotal : " + perteTotal);

    // Afficher la balance
    System.out.println("BALANCE : ");

    if (gainTotal > perteTotal) System.out.println("\tPositive (+" + (gainTotal - perteTotal) + ")");
    else if (gainTotal < perteTotal) System.out.println("\tNegative (" + (gainTotal - perteTotal) + ")");
    else System.out.println("\tNull (à l'équilibre)");

    System.out.println("\n");
  }

  /****************************** AFFICHER LE LANCEMENT DE LA BILLE */
  public static void procAffLancCroupier() {
    System.out.print("\nRien ne va plus");
    Util.procWait(5);
    System.out.println("Les jeux sont faits !\n");
  }

  /*************************************************************************************************************
   *                                        GESTION DU NUMERO DE SORTI
   *************************************************************************************************************/

  /****************************** TIRER UN NUMERO ENTRE 0 ET 36 ET DEFINIR SES CARACTERISTIQUES */
  public static void procDefCaractNumSorti() {
    System.out.println("\n---------------------------------------------------- Tour " + Partie.getNbPartie() + "\n");
    System.out.println("Faites vos jeux !");

    // Tirer le numéro de sorti
    int num = (int)Math.round((Math.random() * 36) * 100) / 100;
    //int num = 3;

    // Définir les caractéristiques du numéro
    switch (num) {
      case 0: {
        for (int i = 0; i < Partie.getCaractNumSorti().length; i++) Partie.setCaractNumSorti(i, 0);
        break;
      }
      default: {
        // Insérer le numéro sorti comme premier élément du tableau
        Partie.setCaractNumSorti(0, num);

        // Définir le numéro de colonne
        Partie.setCaractNumSorti(1, foncQuelleColonne(num));
```

```
      // Définir le numéro de douzaine
      Partie.setCaractNumSorti(2, foncQuelleDouzaine(num));

      // Définir la couleur (0 - noir / 1 - rouge)
      if (foncIsRouge(num)) Partie.setCaractNumSorti(3, 1);
      else Partie.setCaractNumSorti(3, 0);

      // Définir la parité (0 - pair / 1 - impair)
      if (foncIsPair(num)) Partie.setCaractNumSorti(4, 0);
      else Partie.setCaractNumSorti(4, 1);

      // Définir manque ou passe (0 - manque - 1 passe)
      if (foncIsManque(num)) Partie.setCaractNumSorti(5, 0);
      else Partie.setCaractNumSorti(5, 1);
      break;
    }
  }
}

/****************************** RETOURNER LE NUMERO DE COLONNE DU NUMERO SORTI */
public static int foncQuelleColonne(int num) {
  if (IntStream.of(tableJeu[0]).anyMatch(x -> x == num)) return 3;
  else if (IntStream.of(tableJeu[1]).anyMatch(x -> x == num)) return 2;
  else return 1;
}

/****************************** RETOURNER LE NUMERO DE DOUZAINE DU NUMERO SORTI */
public static int foncQuelleDouzaine(int num) {
  if (num <= 12) return 1;
  else if (num <= 24) return 2;
  else return 3;
}

/****************************** RETOURNER TRUE SI LE NUMERO DE SORTI EST ROUGE */
public static boolean foncIsRouge(int num) {
  if (IntStream.of(nbRouges).anyMatch(x -> x == num)) return true;
  else return false;
}

/****************************** RETOURNER TRUE SI LE NUMERO SORTI EST PAIR */
public static boolean foncIsPair(int num) {
  if (num % 2 == 0) return true;
  else return false;
}

/****************************** RETOURNER TRUE SI LE NUMERO SORTI MANQUE */
public static boolean foncIsManque(int num) {
  if (num <= 18) return true;
  else return false;
}

/*****************************************************************************************************************
 *                                          GESTION DE LA PARTIE
 *****************************************************************************************************************/

/****************************** PARAMETRER LE NOMBRE DE JOUEURS DANS LA PARTIE */
public static void procSetNbJoueur() {
  boolean nbJoueurInRange;
  do {
    System.out.print("Indiquer le nombre de joueurs (1,2 ou 3) : ");
    Partie.setNbJoueur(Saisie.foncLireInt(10));
    nbJoueurInRange = Util.foncIsInRange(Partie.getNbJoueur(), 1, 3);

    if (!(nbJoueurInRange)) System.out.println("ERR : le nombre de joueurs doit être compris entre 1 et 3 !");
  }
  while (!(nbJoueurInRange));
}

/****************************** RECUPERER LE NOM DE CHAQUE JOUEUR ET SET DE LA CAGNOTTE A 1000 */
public static void procSetJoueurs() {
  for (int joueur = 0; joueur < Partie.getNbJoueur(); joueur++) {
    System.out.print("Nom du joueur " + (joueur + 1) + " : ");

    switch (joueur) {
      case 0: {
        Joueur.setJoueur1Nom(Saisie.foncLireString().toUpperCase());
        Joueur.setJoueur1Cagnotte(1000);
        break;
      }
      case 1: {
        Joueur.setJoueur2Nom(Saisie.foncLireString().toUpperCase());
        Joueur.setJoueur2Cagnotte(1000);
        break;
      }
      case 2: {
        Joueur.setJoueur3Nom(Saisie.foncLireString().toUpperCase());
        Joueur.setJoueur3Cagnotte(1000);
        break;
      }
    }
  }
```

```
}

/****************************** RENVOYER TRUE SI LE JOUEUR CONTINU LA PARTIE */
public static void procContinuerPartie() {
  System.out.print("Continuer la partie (O/N) ? ");

  if (Saisie.foncLireString().toUpperCase().equals("O")) {
    Partie.setContinuerParti(true);
    Partie.setNbPartie(Partie.getNbPartie()+1);
  } else {
    Partie.setContinuerParti(false);
  }
}

/*****************************************************************************************************************
 *                                          GESTION DES PARIS
 *****************************************************************************************************************/

/****************************** RENVOYER TRUE SI LE JOUEUR SOUHAITE UN NOUVEAU PARI */
public static void procNouvauPari() {
  if (Pari.getCagnotte() == 0) {
    System.out.println("Cagnotte vide, fin des paris !");
    Pari.setNouveauPari(false);
  } else {
    System.out.print("Nouveau pari (O/N) ? (cagnotte restante : " + Pari.getCagnotte() + ") ");
    Pari.setNouveauPari(Saisie.foncLireString().toUpperCase().equals("O"));
  }
}

/****************************** RECUPERER LES PARIS */
public static void foncGetPari() {
  int numPari;
  boolean pariInRange;
  String nomPari = "";

  System.out.print("Quel pari joues-tu (1 à 12) : ");

  do {
    numPari = Saisie.foncLireInt(10);
    pariInRange = Util.foncIsInRange(numPari, 1, 12);

    if (!(pariInRange)) System.out.println("ERR : le nombre de paris doit être compris entre 1 et 12 : ");
  }
  while (!(pariInRange));

  switch (numPari) {
    case 1: nomPari = "En plein"; break;
    case 2: nomPari = "A cheval"; break;
    case 3: nomPari = "Transversale pleine"; break;
    case 4: nomPari = "Triple"; break;
    case 5: nomPari = "Carré"; break;
    case 6: nomPari = "Ligne haute"; break;
    case 7: nomPari = "Sixain"; break;
    case 8: nomPari = "Colonne"; break;
    case 9: nomPari = "Douzaine"; break;
    case 10: nomPari = "Couleur"; break;
    case 11: nomPari = "Parité"; break;
    case 12: nomPari = "Manque - passe"; break;
  }

  Pari.setTypePari(nomPari);
}

/****************************** RECUPERER LA MISE */
public static void procSetfMise() {
  // Vérifier que le joueur mise moins que ce qu'il lui reste dans sa cagnotte
  System.out.print("Combien mises-tu : ");
  Pari.setMise(Saisie.foncLireInt(10));

  while (!(Pari.getCagnotte() - Pari.getMise() >= 0)) {
    System.out.print("Mise trop haute ! Entrez un nouveau montant (cagnotte restante : " + Pari.getCagnotte() + ") : ");
    Pari.setMise(Saisie.foncLireInt(10));
  }
  Pari.setCagnotte(Pari.getCagnotte() - Pari.getMise());
}

/****************************** AFFECTER LES PERTES ET GAINS A CHAQUE JOUEUR */
public static void procSetPertesGains(int joueur) {
  switch (joueur) {
    case 0: {
      if (Pari.getGain() > 0) {
        Joueur.setJoueur1Cagnotte(Joueur.getJoueur1Cagnotte() + Pari.getGain());
        Joueur.setJoueur1gain(Pari.getTypePari(), Pari.getGain());
      } else {
        Joueur.setJoueur1Cagnotte(Joueur.getJoueur1Cagnotte() - Pari.getMise());
        Joueur.setJoueur1perte(Pari.getTypePari(), Pari.getMise());
      }
      break;
    }
    case 1: {
      if (Pari.getGain() > 0) {
        Joueur.setJoueur2Cagnotte(Joueur.getJoueur2Cagnotte() + Pari.getGain());
```

```
        Joueur.setJoueur2gain(Pari.getTypePari(), Pari.getGain());
      } else {
        Joueur.setJoueur2Cagnotte(Joueur.getJoueur2Cagnotte() - Pari.getMise());
        Joueur.setJoueur2perte(Pari.getTypePari(), Pari.getMise());
      }
      break;
    }
    case 2: {
      if (Pari.getGain() > 0) {
        Joueur.setJoueur3Cagnotte(Joueur.getJoueur3Cagnotte() + Pari.getGain());
        Joueur.setJoueur3gain(Pari.getTypePari(), Pari.getGain());
      } else {
        Joueur.setJoueur3Cagnotte(Joueur.getJoueur3Cagnotte() - Pari.getMise());
        Joueur.setJoueur3perte(Pari.getTypePari(), Pari.getMise());
      }
      break;
    }
  }
}

/****************************** PARI EN PLEIN */
public static void procJouerEnPlein() {
  Pari.setRapport(35);                          // Rapport du gain (35 contre 1 etc.)
  int valI;                                     // Valeur jouée par le joueur
  boolean isInRange;                            // True si la valeur testée est entre x et y

  // Récupérer la mise du joueur et valider que sa mise en inférieur à sa cagnotte
  procSetfMise();

  System.out.print("Quel numéro joues-tu : ");
  do {
    valI = Saisie.foncLireInt(10);
    isInRange = Util.foncIsInRange(valI, 0, 36);

    if (!(isInRange)) System.out.print("ERR : le nombre de joueurs doit être compris entre 0 et 36 : ");
    else if (valI == Partie.getCaractNumSorti(0)) Pari.setGain(Pari.getRapport() * Pari.getMise());
  }
  while (!(isInRange));
}

/****************************** PARI A CHEVAL */
public static void procJouerAcheval() {
  Pari.setRapport(17);                          // Rapport du gain (35 contre 1 etc.)
  String valIstrings[];                         // Valeurs jouées par le joueur (ex : 1,2 pour un pari à cheval)
  int valIints[];                               // Valeurs jouées par le joueur (ex : 1,2 pour un pari à cheval)
  int nbNumbers = 2;                            // Nombre de numéros à jouer (ex : carré => 4)
  boolean areNumOk = false;                     // True les valeurs entrées sont valables

  // Récupérer la mise du joueur et valider que sa mise en inférieur à sa cagnotte
  procSetfMise();

  System.out.print("Quels numéros à cheval joues-tu (ex : 3,6) : ");

  do {
    valIstrings = Saisie.foncLireString().trim().split(",");
    if (valIstrings.length == nbNumbers) {
      if (Util.foncAreInts(valIstrings, 10)) {
        valIints = Util.foncStringTabToIntTab(valIstrings);
        if (Util.foncAreInRange(valIints, 0, 36)) {
          areNumOk = foncIsAcheval(valIints);
          if (areNumOk) {
            if (IntStream.of(valIints).anyMatch(x -> x == Partie.getCaractNumSorti(0))) {
              Pari.setGain(Pari.getRapport() * Pari.getMise());
            }
          } else System.out.print("ERR : les valeurs selectionnées ne sont pas valables : ");
        } else System.out.print("ERR : les nombres doivent être compris entre 0 et 36 : ");
      } else System.out.print("ERR : vous devez entrer des numéros séparés par une virgule : ");
    } else System.out.print("ERR : vous devez entrez " + nbNumbers + " numéros : ");
  }
  while (!(areNumOk));
}

public static boolean foncIsAcheval(int valeurs[]) {
  Arrays.sort(valeurs);
  boolean valeursOk = false;

  if (valeurs[0] != 0) {
    loopLigne : for (int ligne = 0; ligne < getTableJeu().length; ligne++) {
      for (int colonne = 0; colonne < getTableJeu()[0].length; colonne++) {
        if (Jeu.getTableJeu()[ligne][colonne] == valeurs[0]
                && (ligne-1 >= 0
                    && Jeu.getTableJeu()[ligne-1][colonne] == valeurs[1])
                  || (ligne+1 < getTableJeu().length
                    && Jeu.getTableJeu()[ligne+1][colonne] == valeurs[1])
                  || (colonne-1 >= 0
                    && Jeu.getTableJeu()[ligne][colonne-1] == valeurs[1])
                  || (colonne+1 < getTableJeu()[0].length
                    && Jeu.getTableJeu()[ligne][colonne+1] == valeurs[1])) {
          valeursOk = true;
          break loopLigne;
        }
      }
```

```
    }
  }
  return valeursOk;
}

/****************************** PARI TRANSVERSALE PLEINE */
public static void procJouerTransversale() {
  Pari.setRapport(11);                          // Rapport du gain (35 contre 1 etc.)
  String valIstrings[];                         // Valeurs jouées par le joueur (ex : 1,2 pour un pari à cheval)
  int valIints[];                               // Valeurs jouées par le joueur (ex : 1,2 pour un pari à cheval)
  int nbNumbers = 3;                            // Nombre de numéros à jouer (ex : carré => 4)
  boolean areNumOk = false;                     // True les valeurs entrées sont valables

  // Récupérer la mise du joueur et valider que sa mise en inférieur à sa cagnotte
  procSetfMise();

  System.out.print("Quels numéros transvesales joues-tu (ex : 1,2,3) : ");

  do {
    valIstrings = Saisie.foncLireString().trim().split(",");
    if (valIstrings.length == nbNumbers) {
      if (Util.foncAreInts(valIstrings, 10)) {
        valIints = Util.foncStringTabToIntTab(valIstrings);
        if (Util.foncAreInRange(valIints, 0, 36)) {
          areNumOk = foncIsTransversal(valIints);
          if (areNumOk) {
            if (IntStream.of(valIints).anyMatch(x -> x == Partie.getCaractNumSorti(0))) {
              Pari.setGain(Pari.getRapport() * Pari.getMise());
            }
          } else System.out.print("ERR : les valeurs selectionnées ne sont pas valables : ");
        } else System.out.print("ERR : les nombres doivent être compris entre 0 et 36 : ");
      } else System.out.print("ERR : vous devez entrer des numéros séparés par une virgule : ");
    } else System.out.print("ERR : vous devez entrez " + nbNumbers + " numéros : ");
  }
  while (!(areNumOk));
}

public static boolean foncIsTransversal(int valeurs[]) {
  Arrays.sort(valeurs);
  boolean valeursOk = false;

  if (valeurs[0] != 0) {
    loopLigne : for (int ligne = 0; ligne < Jeu.getTableJeu().length; ligne++) {
      for (int colonne = 0; colonne < Jeu.getTableJeu()[0].length; colonne++) {
        if (Jeu.getTableJeu()[ligne][colonne] == valeurs[0]
                && ligne == 2
                && Jeu.getTableJeu()[ligne-1][colonne] == valeurs[1]
                && Jeu.getTableJeu()[ligne-2][colonne] == valeurs[2]) {
          valeursOk = true;
          break loopLigne;
        }
      }
    }
  }
  return valeursOk;
}

/****************************** PARI TRIPLE */
public static void procJouerTriple() {
  Pari.setRapport(11);                          // Rapport du gain (35 contre 1 etc.)
  int valI;                                     // Valeur jouée par le joueur
  boolean isInRange;                            // True si la valeur testée est entre x et y

  // Récupérer la mise du joueur et valider que sa mise en inférieur à sa cagnotte
  procSetfMise();

  System.out.print("Quels numéros triple joues-tu ([0] 0,1,2 | [1] 0,2,3) : ");

  do {
    valI = Saisie.foncLireInt(10);
    isInRange = Util.foncIsInRange(valI, 0, 1);

    if (isInRange) {
      switch (valI) {
        case 0: {
          if (Partie.getCaractNumSorti(0) == 0
                  || Partie.getCaractNumSorti(0) == 1
                  || Partie.getCaractNumSorti(0) == 2) {
            Pari.setGain(Pari.getRapport() * Pari.getMise());
          }
        }
        case 1: {
          if (Partie.getCaractNumSorti(0) == 0
                  || Partie.getCaractNumSorti(0) == 2
                  || Partie.getCaractNumSorti(0) == 3) {
            Pari.setGain(Pari.getRapport() * Pari.getMise());
          }
        }
      }
    }
    else System.out.print("ERR : vous devez entrer 0 ou 1 : ");
  }
```

```
    while (!(isInRange));
  }

  /****************************** PARI CARRE */
  public static void procJouerCarre() {
    Pari.setRapport(8);                          // Rapport du gain (35 contre 1 etc.)
    String valIstrings[];                        // Valeurs jouées par le joueur (ex : 1,2 pour un pari à cheval)
    int valIints[];                              // Valeurs jouées par le joueur (ex : 1,2 pour un pari à cheval)
    int nbNumbers = 4;                           // Nombre de numéros à jouer (ex : carré => 4)
    boolean areNumOk = false;                    // True les valeurs entrées sont valables

    // Récupérer la mise du joueur et valider que sa mise en inférieur à sa cagnotte
    procSetfMise();

    System.out.print("Quels numéros carre joues-tu (ex : 2,3,5,6) : ");

    do {
      valIstrings = Saisie.foncLireString().trim().split(",");
      if (valIstrings.length == nbNumbers) {
        if (Util.foncAreInts(valIstrings, 10)) {
          valIints = Util.foncStringTabToIntTab(valIstrings);
          if (Util.foncAreInRange(valIints, 0, 36)) {
            areNumOk = foncIsCarre(valIints);
            if (areNumOk) {
              if (IntStream.of(valIints).anyMatch(x -> x == Partie.getCaractNumSorti(0))) {
                Pari.setGain(Pari.getRapport() * Pari.getMise());
              }
            } else System.out.print("ERR : les valeurs selectionnées ne sont pas valables : ");
          } else System.out.print("ERR : les nombres doivent être compris entre 0 et 36 : ");
        } else System.out.print("ERR : vous devez entrer des numéros séparés par une virgule : ");
      } else System.out.print("ERR : vous devez entrez " + nbNumbers + " numéros : ");
    }
    while (!(areNumOk));
  }

  public static boolean foncIsCarre(int valeurs[]) {
    Arrays.sort(valeurs);
    boolean valeursOk = false;

    if (valeurs[0] != 0) {
      loopLigne : for (int ligne = 0; ligne < Jeu.getTableJeu().length; ligne++) {
        for (int colonne = 0; colonne < Jeu.getTableJeu()[0].length; colonne++) {
          if (Jeu.getTableJeu()[ligne][colonne] == valeurs[0]
                  && ligne-1 >= 0
                  && colonne+1 < Jeu.getTableJeu()[0].length
                  && Jeu.getTableJeu()[ligne-1][colonne] == valeurs[1]
                  && Jeu.getTableJeu()[ligne][colonne+1] == valeurs[2]
                  && Jeu.getTableJeu()[ligne-1][colonne+1] == valeurs[3]) {
            valeursOk = true;
            break loopLigne;
          }
        }
      }
    }
    return valeursOk;
  }

  /****************************** PARI LIGNE HAUTE */
  public static void procJouerLigne() {
    Pari.setRapport(8);                          // Rapport du gain (35 contre 1 etc.)

    System.out.println("Les numéros de la ligne du haut sont les suivants : 0,1,2,3");

    // Récupérer la mise du joueur et valider que sa mise en inférieur à sa cagnotte
    procSetfMise();

    if (Util.foncIsInRange(Partie.getCaractNumSorti(0), 0, 3)) {
      Pari.setGain(Pari.getRapport() * Pari.getMise());
    }
  }

  /****************************** PARI SIXAIN */
  public static void procJouerSixain() {
    Pari.setRapport(5);                          // Rapport du gain (35 contre 1 etc.)
    String valIstrings[];                        // Valeurs jouées par le joueur (ex : 1,2 pour un pari à cheval)
    int valIints[];                              // Valeurs jouées par le joueur (ex : 1,2 pour un pari à cheval)
    int nbNumbers = 6;                           // Nombre de numéros à jouer (ex : carré => 4)
    boolean areNumOk = false;                    // True les valeurs entrées sont valables

    // Récupérer la mise du joueur et valider que sa mise en inférieur à sa cagnotte
    procSetfMise();

    System.out.print("Quels numéros de sixain joues-tu (ex : 1,2,3,4,5,6) : ");

    do {
      valIstrings = Saisie.foncLireString().trim().split(",");
      if (valIstrings.length == nbNumbers) {
        if (Util.foncAreInts(valIstrings, 10)) {
          valIints = Util.foncStringTabToIntTab(valIstrings);
          if (Util.foncAreInRange(valIints, 0, 36)) {
            areNumOk = foncIsSixain(valIints);
            if (areNumOk) {
```

```
            if (IntStream.of(valIints).anyMatch(x -> x == Partie.getCaractNumSorti(0))) {
              Pari.setGain(Pari.getRapport() * Pari.getMise());
            }
          } else System.out.print("ERR : les valeurs selectionnées ne sont pas valables : ");
        } else System.out.print("ERR : les nombres doivent être compris entre 0 et 36 : ");
      } else System.out.print("ERR : vous devez entrer des numéros séparés par une virgule : ");
    } else System.out.print("ERR : vous devez entrez " + nbNumbers + " numéros : ");
  }
  while (!(areNumOk));
}

public static boolean foncIsSixain(int valeurs[]) {
  Arrays.sort(valeurs);
  boolean valeursOk = false;

  if (valeurs[0] != 0) {
    loopLigne : for (int ligne = 0; ligne < Jeu.getTableJeu().length; ligne++) {
      for (int colonne = 0; colonne < Jeu.getTableJeu()[0].length; colonne++) {
        if (Jeu.getTableJeu()[ligne][colonne] == valeurs[0]
                && ligne-2 == 0
                && colonne+1 < Jeu.getTableJeu()[0].length
                && Jeu.getTableJeu()[ligne-1][colonne] == valeurs[1]
                && Jeu.getTableJeu()[ligne-2][colonne] == valeurs[2]
                && Jeu.getTableJeu()[ligne][colonne+1] == valeurs[3]
                && Jeu.getTableJeu()[ligne-1][colonne+1] == valeurs[4]
                && Jeu.getTableJeu()[ligne-2][colonne+1] == valeurs[5]) {
          valeursOk = true;
          break loopLigne;
        }
      }
    }
  }
  return valeursOk;
}

/****************************** PARI COLONNE */
public static void procJouerColonne() {
  Pari.setRapport(2);                       // Rapport du gain (35 contre 1 etc.)
  int indice = 1;                           // Indice du tableau caractNumSorti à vérifier
  int valI;                                 // Valeur jouée par le joueur (ex : numéro 10)
  boolean isInRange;                        // True si la valeur testée est entre x et y

  // Récupérer la mise du joueur et valider que sa mise en inférieur à sa cagnotte
  procSetfMise();

  System.out.print("Quel numéro de colonne joues-tu : ");

  do {
    valI = Saisie.foncLireInt(10);
    isInRange = Util.foncIsInRange(valI, 1, 3);

    if (isInRange) {
      if (valI == Partie.getCaractNumSorti(indice)) Pari.setGain(Pari.getRapport() * Pari.getMise());
    }
    else System.out.print("ERR : le numéro doit être compris entre 1 et 3 : ");
  }
  while (!(isInRange));
}

/****************************** PARI DOUZAINE */
public static void procJouerDouzaine() {
  Pari.setRapport(2);                       // Rapport du gain (35 contre 1 etc.)
  int indice = 2;                           // Indice du tableau caractNumSorti à vérifier
  int valI;                                 // Valeur jouée par le joueur (ex : numéro 10)
  boolean isInRange;                        // True si la valeur testée est entre x et y

  // Récupérer la mise du joueur et valider que sa mise en inférieur à sa cagnotte
  procSetfMise();

  System.out.print("Quelle numéro de colonne joues-tu : ");

  do {
    valI = Saisie.foncLireInt(10);
    isInRange = Util.foncIsInRange(valI, 1, 3);

    if (isInRange) {
      if (valI == Partie.getCaractNumSorti(indice)) Pari.setGain(Pari.getRapport() * Pari.getMise());
    }
    else System.out.print("ERR : le numéro doit être compris entre 1 et 3 : ");
  }
  while (!(isInRange));
}

/****************************** PARI COULEUR */
public static void procJouerCouleur() {
  Pari.setRapport(1);                       // Rapport du gain (35 contre 1 etc.)
  int indice = 3;                           // Indice du tableau caractNumSorti à vérifier
  int valI;                                 // Valeur jouée par le joueur (ex : numéro 10)
  boolean isInRange;                        // True si la valeur testée est entre x et y

  // Récupérer la mise du joueur et valider que sa mise en inférieur à sa cagnotte
  procSetfMise();
```

```
    System.out.print("Quelle couleur joues-tu ([0] Noir | [1] Rouge) : ");

    do {
      valI = Saisie.foncLireInt(10);
      isInRange = Util.foncIsInRange(valI, 0, 1);

      if (isInRange) {
        if (Partie.getCaractNumSorti(0) != 0
                && valI == Partie.getCaractNumSorti(indice)) {
          Pari.setGain(Pari.getRapport() * Pari.getMise());
        }
      } else System.out.print("ERR : vous devez entrer 0 ou 1 : ");
    }
    while (!(isInRange));
  }

  /****************************** PARI PARITE */
  public static void procJouerParite() {
    Pari.setRapport(1);                       // Rapport du gain (35 contre 1 etc.)
    int indice = 4;                           // Indice du tableau caractNumSorti à vérifier
    int valI;                                 // Valeur jouée par le joueur (ex : numéro 10)
    boolean isInRange;                        // True si la valeur testée est entre x et y

    // Récupérer la mise du joueur et valider que sa mise en inférieur à sa cagnotte
    procSetfMise();

    System.out.print("Quelle parité joues-tu ([0] Pair | [1] Impair) : ");

    do {
      valI = Saisie.foncLireInt(10);
      isInRange = Util.foncIsInRange(valI, 0, 1);

      if (isInRange) {
        if (Partie.getCaractNumSorti(0) != 0
                && valI == Partie.getCaractNumSorti(indice)) {
          Pari.setGain(Pari.getRapport() * Pari.getMise());
        }
      }
      else System.out.print("ERR : vous devez entrer 0 ou 1 : ");
    }
    while (!(isInRange));
  }

  /****************************** PARI MANQUE PASSE */
  public static void procJouerManquePasse() {
    Pari.setRapport(1);                       // Rapport du gain (35 contre 1 etc.)
    int indice = 5;                           // Indice du tableau caractNumSorti à vérifier
    int valI;                                 // Valeur jouée par le joueur (ex : numéro 10)
    boolean isInRange;                        // True si la valeur testée est entre x et y

    // Récupérer la mise du joueur et valider que sa mise en inférieur à sa cagnotte
    procSetfMise();

    System.out.print("Joues-tu manque ou passe ([0] Manque | [1] Passe) : ");

    do {
      valI = Saisie.foncLireInt(10);
      isInRange = Util.foncIsInRange(valI, 0, 1);

      if (isInRange) {
        if (Partie.getCaractNumSorti(0) != 0
                && valI == Partie.getCaractNumSorti(indice)) {
          Pari.setGain(Pari.getRapport() * Pari.getMise());
        }
      }
      else System.out.print("ERR : vous devez entrer 0 ou 1 : ");
    }
    while (!(isInRange));
  }
}
```