UNIVERSITÉ
PARIS DESCARTES
UFR DE MATHÉMATIQUES ET INFORMATIQUE

Conception Détaillée

DAVID JANISZEK

Projet L2AN1

UNIVERSITÉ PARIS DESCARTES
UFR DE MATHÉMATIQUES ET INFORMATIQUE

Conception Détaillée

Blackjack (L2AN1)

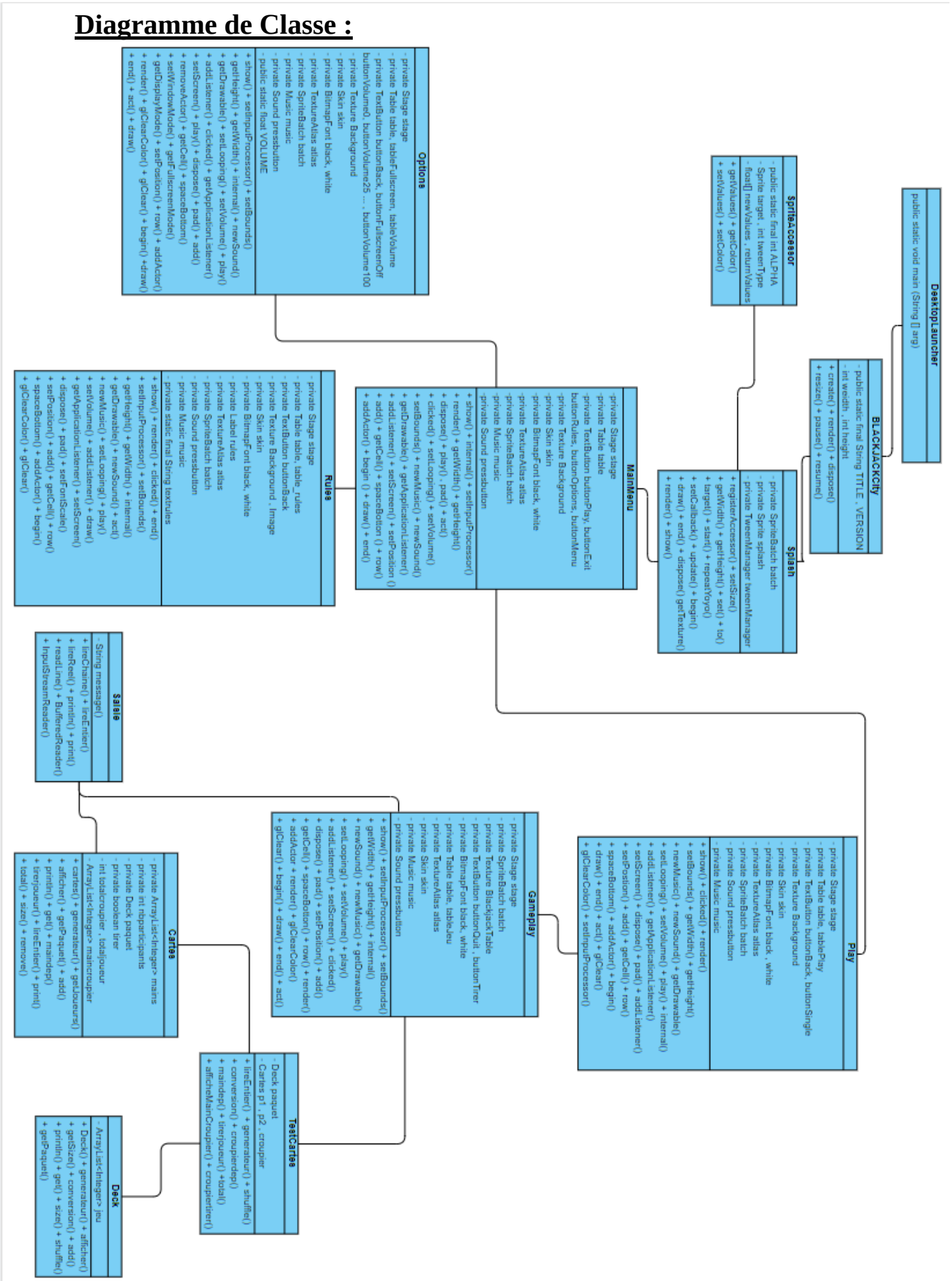
Les informations d'identification du document

Référence du document :	D3
Version du document :	1.01
Date du document :	21 mars 2021
Auteur(s) :	David Janiszek

Les éléments de vérification du document

Validé par :	David Janiszek
Validé le :	21/03/2021
Soumis le :	21/03/2021
Type de diffusion :	Document électronique (.odt)
Confidentialité :	Standard / Étudiants UFR Maths-Info de l'université Paris Descartes

Diagramme de Classe :



Description des Classes :

Le projet est composé de 2 fichiers principaux : Core et Desktop.

Desktop contient le fichier src avec un seul package : fr.mygdx.game.desktop qui ne contient que la class permettant de lancer l'application nommée DesktopLauncher.java. La class DesktopLauncher est composée du main de l'application. Dans le main, il y a l'objet qu'on a appelé config, par convention, qui provient de la class LwglApplicationConfiguration. Cette dernière permet de définir les configurations de lancement de l'application. Il est nécessaire de la relier au Core afin de lier le main au reste du code : on lie l'objet config à la class ''passerelle'' entre le Core et le Desktop qu'on a nommé BLACKJACKCity.java.

On arrive donc au fichier Core qui contient 2 fichiers principaux :

- assets : contient les différentes images, les sons et musiques utilisés dans le jeu.
- src : contenant 3 packages :
    - fr.mygdx.game :
        - BLACKJACKcity.java:
            - Attributs :
                - public static final String TITLE : définit le titre et la version du jeu
            - Méthodes :
                - public void create() : appelle l'écran de chargement
                - public void render() : Permet d'organiser l'affichage de l'écran
                - public void resize() : Permet de changer la taille d'un objet à l'écran
                - public void pause() : Permet de gérer la pause
                - public void resume() : Permet de gérer la remise en jeu après une pause
                - public void hide() : Permet de cacher un objet affiché pendant un temps déterminé
                - public void dispose() : Permet de fermer les différents objets utilisés lors d'un changement d'écran afin d'optimiser l'application et de ne pas surcharger l'ordinateur
        - Options.java :
            - Attributs :
                - private Stage stage : Permet de créer une organisation de la totalité de l'écran en fonction de la résolution de l'écran.
                - private Table table, tableFullscreen, tableVolume : Permet de créer un petit tableau permettant de lier certains boutons par exemple et de les placer ensemble.
                - private TextButton buttonBack, buttonFullscreenOn, buttonFullscreenOff, buttonVolume25, buttonVolume50, buttonVolume75, buttonVolume100, buttonVolume0 : Permet de créer les boutons associé à

cette classe, ici les boutons volumes qui sont dans les options.
        - private Texture Background :Permet de créer l'image qui sera en arrière-plan du menu des options.
        - private Skin skin : Permet de créer la texture des boutons
        - private BitmapFont black, white : Permet de créer les polices d'écritures utilisées
        - private TextureAtlas atlas : Permet de donner une texture au bouton.
        - private SpriteBatch batch : Permet de créer un support pour générer des textures
        - private Music music : Permet de créer la musique du jeu
        - private Sound pressbutton : Permet de créer le son qu'on entend lorsque l'on clique sur un bouton
    - Méthodes :
        - public void show() : Permet de définir les objets à afficher
        - setInputProcessor() : Permet de faire fonctionner les animations des boutons
        - setBounds() : Définit les limites de la table.
        - newMusic() : Permet d'importer une nouvelle musique.
        - newSound() : Permet d'importer un nouveau son (pour un bouton par exemple).
        - getDrawable() : Dessine le la texture.
        - pressedOffsetX() :
        - pressedOffsetY() :
        - font() : Définit la couleur de la police.
        - public void clicked(InputEvent event, float x, float y) : Permet de récupérer l'information qu'un bouton a été cliqué.

- Play.java :
    - Attributs :
        - private Stage stage : Permet de créer une organisation de la totalité de l'écran en fonction de la résolution de l'écran.
        - private Table table : Permet de créer un petit tableau permettant de lier certains boutons par exemple et de les placer ensemble.
        - private TextButton buttonSingle, buttonBack: Permet de créer les boutons de cette class
        - private Texture Background : Permet de créer l'image qui sera en arrière-plan du menu
        - private Skin skin : Permet de créer la texture des boutons
        - private BitmapFont black, white : Permet de créer les polices d'écritures utilisées

        - private SpriteBatch batch : Permet de créer un support pour générer des textures
        - private Music music : Permet de créer la musique du jeu
        - private Sound pressbutton : Permet de créer le son qu'on entend lorsque l'on clique sur un bouton
        - private TextureAtlas atlas:Permet de donner une texture au bouton.
    - Méthodes :
        - public void show() : Permet de définir les objets à afficher
        - public void render() : Permet d'organiser l'affichage de l'écran
        - public void resize() : Permet de changer la taille d'un objet à l'écran
        - public void pause() : Permet de gérer la pause
        - public void resume() : Permet de gérer la remise en jeu après une pause
        - public void hide() : Permet de cacher un objet affiché pendant un temps déterminé
        - public void dispose() : Permet de fermer les différents objets utilisés lors d'un changement d'écran afin d'optimiser l'application et de ne pas surcharger l'ordinateur
        - Gdx.input.setInputProcessor : Permet de faire fonctionner les animations des boutons
        - table.setBounds() :
        - skin.getDrawable : Dessine le la texture
        - textButtonStyle.up : Position du bouton non enfoncé
        - textButtonStyle.down : Position du bouton enfoncé
        - textButtonStyle.pressedOffsetX :
        - textButtonStyle.pressedOffsetY :
        - textButtonStyle.font : ajoute la police
        - music.setLooping() : Permet de faire tourner la musique en boucle.
        - music.setVolume() : Permet de régler le volume de la musique.
        - music.play() : Commence la musique
        - pressbutton.play(): Appui sur le bouton play
        - button.addListener() : Permet de donner une fonction à un bouton
        - button.pad() : Permet de donner les dimensions d'un bouton
        - button.setPosition() : Permet de donner la position du tableau sur l'écran
        - button.add() : Ajoute un élément dans un tableau
        - button.getCell().spaceBottom() : Permet de donner de l'espace sous une cellule du tableau que l'on référence à l'aide de .getCell()
        - button.row() : Permet de sauter une ligne sous une cellule

            - Gdx.gl.glClearColor() : Permet de donner une couleur de base à l'arrière plan
            - Gdx.gl.glClear() : Permet de vider l'arrière plan de l'écran
            - batch.begin() : début de la création de texture
            - batch.draw() : dessine un rectangle selon des dimensions que l'on précise.
            - batch.end() : Fin de création de textures.
            - object.dispose() : Fin d'une musique.
            - getWidth() : retourne la largeur
            - getHeight() : retourne la hauteur
            - internals() : cherche un fichier dans le projet automatiquement sans besoin de renseigner le chemin
            - void clicked(InputEvent event, float x, float y) : Permet de récupérer l'information qu'un bouton a été cliqué.
- fr.mygdx.game.splashscreen : permet à l'application de se lancer sur un splashscreen puis d'accéder au menu principal à l'aide de 2 class :
    - Splash.java :
        - Cette classe permet de générer l'écran de chargement au lancement du jeu
        - Attributs :
            - private SpriteBatch batch : Permet de créer un support pour générer des textures
            - private Sprite splash : Désigne l'écran de chargement private TweenManager : Permet de gérer l'écran de chargement
            - Méthodes :
            - registerAccessor(Sprite.class, new SpriteAccessor())
            - set() : définit des valeurs pour ce qui est passé en paramètres
            - to() : Fais la transition dans le fondu de l'écran de chargement
            - target() : règle l'opacité
            - start() : commence en appellent tweenManager
            - repeatYoyo() : défini le temps de l'écran de chargement
            - setCallback() : appelle le menu
            - public void pause() :
            - public void resume() :
            - public void hide() :
            - public void dispose() :
            - getTexture() : retourne la texture
            - onEvent(int type, BaseTween<?> source):permet d'accéder au menu après l'écran de chargement
            - setScreen() : affiche le menu
            - batch.begin() : début de la création de texture
            - batch.draw() : dessine une texture
            - batch.end() : fin de la création de texture
            - splash.setSize() : définit la taille
            - Gdx.gl.glClearColor() : définit la couleur de l'arrière plan

        - Gdx.gl.glClear() : efface tout l'arrière plan
        - public void show() : Permet de définir les objets à afficher
        - public void render() : Permet d'organiser l'affichage de l'écran
        - public void resize() : Permet de changer la taille d'un objet à l'écran
        - public void pause() : Permet de gérer la pause
        - public void resume() : Permet de gérer la remise en jeu après une pause
        - public void hide() : Permet de cacher un objet affiché pendant un temps déterminé
        - public void dispose() : Permet de fermer les différents objets utilisés lors d'un changement d'écran afin d'optimiser l'application et de ne pas surcharger l'ordinateur
    - MainMenu.java :
        - Cette classe permet de gérer le menu principal
        - Attributs :
            - private Stage stage : Permet de créer une organisation de la totalité de l'écran en fonction de la résolution de l'écran.
            - private Table table : Permet de créer un petit tableau permettant de lier certains boutons par exemple et de les placer ensemble.
            - private TextButton buttonPlay, buttonExit, buttonRules, buttonOptions, buttonMenu : Permet de créer les boutons de cette class
            - private Texture Background : Permet de créer l'image qui sera en arrière-plan du menu
            - private Skin skin : Permet de créer la texture des boutons
            - private BitmapFont black, white : Permet de créer les polices d'écritures utilisées
            - private SpriteBatch batch : Permet de créer un support pour générer des textures
            - private Music music : Permet de créer la musique du jeu
            - private Sound pressbutton : Permet de créer le son qu'on entend lorsque l'on clique sur un bouton
        - Méthodes :
            - public void show() : Permet de définir les objets à afficher
            - public void render() : Permet d'organiser l'affichage de l'écran
            - public void resize() : Permet de changer la taille d'un objet à l'écran
            - public void pause() : Permet de gérer la pause
            - public void resume() : Permet de gérer la remise en jeu après une pause

            - public void hide() : Permet de cacher un objet affiché pendant un temps déterminé
            - public void dispose() : Permet de fermer les différents objets utilisés lors d'un changement d'écran afin d'optimiser l'application et de ne pas surcharger l'ordinateur
            - Gdx.input.setInputProcessor : Permet de faire fonctionner les animations des boutons
            - table.setBounds() :
            - skin.getDrawable : Dessine le la texture
            - textButtonStyle.up : Position du bouton non enfoncé
            - textButtonStyle.down : Position du bouton enfoncé
            - textButtonStyle.pressedOffsetX :
            - textButtonStyle.pressedOffsetY :
            - textButtonStyle.font : ajoute la police
            - music.setLooping() : Permet de faire tourner la musique en boucle.
            - music.setVolume() : Permet de régler le volume de la musique.
            - music.play() : Commence la musique
            - pressbutton.play(): Appui sur le bouton play
            - button.addListener() : Permet de donner une fonction à un bouton
            - button.pad() : Permet de donner les dimensions d'un bouton
            - button.setPosition() : Permet de donner la position du tableau sur l'écran
            - button.add() : Ajoute un élément dans un tableau
            - button.getCell().spaceBottom() : Permet de donner de l'espace sous une cellule du tableau que l'on référence à l'aide de .getCell()
            - button.row() : Permet de sauter une ligne sous une cellule
            - Gdx.gl.glClearColor() : Permet de donner une couleur de base à l'arrière plan
            - Gdx.gl.glClear() : Permet de vider l'arrière plan de l'écran
            - batch.begin() : début de la création de texture
            - batch.draw() : dessine un rectangle selon des dimensions que l'on précise.
            - batch.end() : Fin de création de textures.
            - object.dispose() : Fin d'une musique.
- fr.mygdx.game.splashengine : insère la librairie Tween Engine, permettant la disposition du splashscreen, dans la class : SpriteAccessor.java
    - SpriteAccessor.java :
        - class implement TweenAccessor<Sprite> : permettant d'avoir les fonctions getValues et setValues
        - Attributs :

        - public static final int Alpha = 0 : c'est le cas principal du switch de la méthode getValues et de celui de la méthode setValues
    - Méthodes :
        - getValues(Sprite target, int tweenType, float[] returnValues) : composé d'un switch, cette fonction permet de récupérer les valeurs des couleurs affichées lors du Splashscreen
        - setValues(Sprite target, int tweenType, float[] newValues) : composé d'un switch, cette fonction permet de donner les valeurs des couleurs à afficher lors du Splashscreen

L'ensemble des classes ci-dessous permettent le fonctionnement de la classe TestCartes.java qui permet de jouer au blackjack sans interface graphique.

La classe Deck.java :

Un seul attribut : Un ArrayList nommé "jeu" qui va permettre de stocker le jeu de cartes.
Lorsque l'on joue au blackjack, on prend six paquets de 52 cartes que l'on mélange, et on joue avec.
L'attribut jeu est donc un ArrayList de taille 312, car c'est la taille d'un jeu composé de 6 paquets.
Voici les méthodes de la classe Deck :

void generateur() : Permet de générer un jeu de 312 cartes, numérotées de 1 (As) à 13 (Roi), rangées dans l'ordre, c'est à dire qu'après l'appel de cette méthode jeu est l'ArrayList suivant :
[1,1,1,…,13,13,13]
void shuffle() : Permet de mélanger le paquet, grâce à la méthode shuffle(jeu).
void conversion() : Permet d'affecter à tous les éléments supérieurs à 10 de l'attribut jeu la valeur 10.
Au blackjack, le but est de faire 21 en sommant la valeur de toutes nos cartes, la carte 2 ayant la valeur 2, la 5 la valeur 5,et ainsi de suite,les cartes Valet,Dame,Roi ayant aussi une valeur de 10 au blackjack il faut donc les transformer en 10. La carte As est un cas particulier qui possède à la fois la valeur 1 et la valeur 11.
ArrayList<Integer> getPaquet() : Permet de retourner l'attribut "jeu".

La classe Saisie.java :

Elle permet de récupérer des entrées claviers.
Pas d'attributs.
Les méthodes :

public static String lireChaine(String message)
public static int lireEntier(String message) : Permet de lire un entier.
public static double lireReel(String message) : Permet de lire un réel.
La classe Cartes.java :

Les attributs :
-L'attribut "mains" qui désigne la main du joueur, de type ArrayList<Integer>.
-Lattribut "nbparticipants" qui désigne le nombre de participants sur la table, de type int.
-L'attribut "paquet" qui désigne le paquet avec lequel on joue, de type Deck vu au-dessus.
-L'attribut "tirer" qui permet de savoir si le joueur veut tirer une carte supplémentaire ou rester avec son jeu, de type boolean.

Avant de présenter les méthodes, rappelons brièvement le but du blackjack.
Le but du blackjack est d'obtenir un meilleur score que le croupier. Pour cela, on peut tirer autant de cartes qu'on le souhaite, mais il est interdit de dépasser le score de 21. Notre score s'obtient en faisant l'addition de nos cartes. Toutes les cartes valent le numéro qui leur est associé (2 vaut 2 points par exemple), le valet, la dame et le roi valent 10 points et l'As vaut 11 ou 1 (ce qui est le plus avantageux). On commence avec deux cartes,qui correspondent à notre main de départ, le croupier commence également avec deux cartes, mais une seule est révélée. Ensuite il y a deux phases. D'abord on peut donc choisir de tirer une carte ou de rester avec notre score. Si on dépasse 21 en tirant des cartes, on perd automatiquement. Une fois qu'on a choisi d'arrêter de tirer, vient la deuxième phase, qui est le tour du croupier. Le croupier suit un algorithme très simple : tant qu'il a un score inférieur à 17, il continue de tirer. Si jamais il passe au-dessus de 21 et que le joueur a un score encore réglementaire, alors le joueur gagne. Si le croupier possède un score réglementaire supérieur au joueur, le croupier gagne, et si jamais ils ont le même score il y a égalité. Maintenant, voici les méthodes :

int getJoueurs() : Cette méthode permet de retourner le nombre de joueurs présents à la table, on se sert de la classe saisie.java pour avoir cette information.
void maindep() : Cette méthode permet de construire la main de départ d'un joueur, composé de deux cartes. Elle est composée des deux premiers éléments du paquet, qu'elle remet ensuite au fond du paquet.
void tirerjoueur(ArrayList<Integer> maincroupier, int totalcroupier) : Cette méthode permet de demander au joueur s'il souhaite tirer une carte supplémentaire.
ArrayList<Integer> croupierdep() : Cette méthode permet de construire la main de depart du croupier et de la retourner, le fait de retourner la main du croupier permet ensuite de l'afficher dans une prochaine méthode.
ArrayList<Integer> afficheMainCroupier() : Affiche la main du croupier.
void croupiertirer(int totaljoueur) : Cette méthode applique l'algorithme décrit plus haut concernant le croupier. Elle affiche aussi si le joueur à gagné, perdu, ou fait égalité. En résumé, elle fait tirer des cartes au croupier s'il a moins de 17.
int total() : Cette méthode retourne le total de la main d'un joueur.

Enfin, la classe Testcartes contient le main et se sert des objets des différentes classes construites auparavant.
Ces classes ne sont pas définitives et vont être modifiées et améliorées au cours du projet.