REMISE TP1

Code TP1

Mathieu Gravel GRAM02099206[??*??] and Nicolas Reybaud REYN23119308[??,??]

[*]Correspondence:
gravel.mathieu.3@courrier.uqam.ca
[??]Department d'informatique, UQAM, UQAM des Sciences, Montreal, Quebec

Résumé

Resume: Ce document detient le code source associe a l'implementation de notre TP1.

Keywords: TP1; Code Source; API

Table des matières

Code source

1 CarteJeux

1.1 Carte.Java

```
package cardgame.JeuxCartes;

import javax.json.JsonObject;

/**
 * Classe abstraite, Carte sert d'interface commun pour
    tout les types de cartes
 * du jeu. (La raison derrière le choix de classe
    abstraite et non d'interface
 * réside dans l'identifiant unique. Celle-ci nous
    permet de lier une carte du
 * modèle aux demandes du controlleurs si nécessaire.)
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public abstract class Carte {

    //Int statique utilisé poour s'assurer que chaque
       carte ait un Id unique.
    static int SSID = 0;
    private final int cardID;

    /**
     * Description des fonctions représentant le JSon
        des cartes, non définie
```

```
     * ici
     *
     * @return null, fonction non définie ici
     */
    public abstract JsonObject toJSON();

    /**
     * Constructeur par défaut. Initialise l'identifiant
        de la carte.
     */
    public Carte() {
        cardID = SSID;
        ++Carte.SSID;
    }

    /**
     * Permet d'avoir l'id de la carte.
     *
     * @return l'identifiant unique associé à la carte
     */
    public int getCardID() {
        return cardID;
    }
}
```

1.2 Perso.Java

```
package cardgame.JeuxCartes;

import cardgame.Regles.TypeArme;
import cardgame.ResultUtils.AttaquePersoResult;
import java.util.ArrayList;
import java.util.List;
import javax.json.*;

/**
 * Classe représentant les cartes de type personnages du
    jeu. La carte peut être
 * extend directement pour créer un perso non-combattant.
 *
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale. 12-Fév-2016 :
    1.1 - Modification du
```

```
 * code pour marcher avec Cible. Guerrier,Pretre et
    Paladin. 14-Fév-2016 : 1.2 -
 * Modification du code pour marcher avec Combattant.
 */
public class Perso extends Carte implements Cible {

    private int hp;
    private final int maxHp;
    private int mp;
    private final int maxMp;
    private Arme armePerso;
    private final List<TypeArme> armesUtilisables;

    public Perso(int _hp, int _mp, List<TypeArme> armes)
       {
        super();
        hp = _hp;
        mp = _mp;
        maxHp = hp;
        maxMp = _mp;
        armePerso = null;
        armesUtilisables = armes;
    }

    /**
     * @return L'arme du personnage
     */
    public Arme getArme() {
        return armePerso;
    }

    public int getMp() {
        return mp;
    }

    public List<TypeArme> getArmesUtilisables() {
        return armesUtilisables;
    }

    /**
     * Utilise un point de magie.
     */
    protected void utiliserMagie() {
        Math.max(mp--, 0);
    }
```

```java
/**
 * Permet d'obtenir la liste des cartes qui était
    associée au personnage.
 * Ceci nous permet de les ajouter au cimetière à la
    mort du perso.
 *
 * @return Liste des cartes présente sur le perso
 */
protected List<Carte> libererCartes() {
    List<Carte> cartesMortes = new ArrayList<>();
    cartesMortes.addAll(armePerso.listEnchant);
    cartesMortes.addAll(armePerso.listEnchantStase);
    cartesMortes.add(armePerso);
    cartesMortes.add(this);
    return cartesMortes;
}

/**
 * Permet de vérifier si le perso peut utiliser une
    arme et si oui, la lui
 * place.
 *
 * @param arme arme à donner au perso
 * @return true si l'arme est placée, false sinon
 */
protected boolean equiperArme(Arme arme) {
    boolean armeLibre = false;
    if (this.armePerso == null &&
        arme.peutUtiliserArme(this)) {
        this.armePerso = arme;
        armeLibre = true;
    }
    return armeLibre;
}

/**
 * Permet au personnage de recevoir le soin
    (Autrement dit, réinit ses
 * points de vie).
 */
protected void recevoirSoins() {
    this.hp = this.maxHp;
}

/**
 * Permet d'obtenir le type d'arme utilisée par le
    perso
```

```
     *
     * @return le type de l'arme si une est équipée,
        null sinon
     */
    public TypeArme getTypeArme() {
        return armePerso != null ? armePerso.type : null;
    }

    /**
     * Permet de savoir si le personnage est mort.
     *
     * @return true si le personnage est mort, false
        sinon
     */
    @Override
    public boolean estMort() {
        return this.hp <= 0;
    }

    /**
     * Permet d'obtenir le Json associé au personnage.
     *
     * @return le JSon représentant le perso
     */
    @Override
    public JsonObject toJSON() {
        JsonObjectBuilder obj =
            Json.createObjectBuilder();
        obj.add("Id", this.getCardID());
        //obj.add("Type Personnage",
            typeperso.toString());
        obj.add("hp", hp);
        obj.add("mp", mp);
        if (armePerso != null) {
            obj.add("Arme personnage",
                armePerso.toJSON());
        }

        return obj.build();
    }

    @Override
    public boolean peutEtreAttaque() {
        return !estMort();
    }
```

```
    @Override
    public AttaquePersoResult recoitAttaque(Combattant
        attaqueur) {
        AttaquePersoResult res;
        assert (this.armePerso != null);
        int degat =
            attaqueur.forceAttaque(this.armePerso.getTypeArme());
        this.hp -= degat;
        res = new AttaquePersoResult(degat,
            this.getCardID(), attaqueur.getCardID(),
            estMort());
        return res;
    }

}
```

1.3 Carte.Java

```
package cardgame.JeuxCartes;

import cardgame.Regles.TypeArme;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import javax.json.*;

/**
 * Classe représentant chacunes des cartes d'armes du
    jeu. Initialisé par
 * ArmeFactory (Afin d'attribuer les bons attributs pour
    chaque type d'armes),
 * Arme permet de traiter la logique de : - Force
    d'attaque - ÃĽquipage
 * d'enchantements - Vérification de l'arme pour un
    déploiement.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.2
 *
 * Historique : 08-Fév-2016 : 1.0 - Version initiale.
 * 11-Fév-2016 : 1.1 - Découpage de ListUtilisateurs
    pour le placer
 *                       dans Perso.
 *               1.2 - Ajout de fonctions pour vérifier
    si l'amr
```

```
 *                      est déployé.
 */
public class Arme extends Carte {

    protected TypeArme type;
    /**
     * Boolean notant explicitement si l'arme est stased
        pour fins
     * d'éfficiences.
     */
    private boolean estStase;
    private boolean armeUtilise;
    protected int degat;
    protected List<Enchant> listEnchant;
    protected boolean estFacile;

    /**
     * Liste utilisé pour noter les enchantements qui
        ont été Stased. Cette
     * liste sert seulement pour fins d'affichages.
     */
    protected List<Enchant> listEnchantStase;

    /*Variables notant les valeurs initiales de l'arme.
     Ceci nous permet de remettre l'arme à zéro si
        nécessaire.*/
    private final int degatOrg;
    private final TypeArme typeOrg;

    public Arme(TypeArme _type, int dmg) {
        super();
        type = _type;
        typeOrg = _type;
        degat = dmg;
        degatOrg = dmg;
        armeUtilise = false;
        estFacile = false;
        listEnchant = new ArrayList<>();
        listEnchantStase = new ArrayList<>();
        estStase = false;
    }

    /**
     * Permet de savoir la force d'attaque de l'arme en
        appliquant le triangle
     * des degats
```

```
     *
     * @param arme Type d'arme sur laquel faire le
        triangle de modificateur de
     * dégats
     * @return la force d'attaque de l'arme
     */
    public int forceAttaque(TypeArme arme) {
        return this.degat +
            this.type.calculModificateur(arme);
    }

    /**
     * Permet de savoir si un perso peut utiliser ou non
        une arme.
     *
     * @param p personnage à verifier
     * @return true si le perso peut porter l'arme false
        sinon
     */
    public boolean peutUtiliserArme(Perso p) {
        return (estFacile ||
            p.getArmesUtilisables().contains(this.type));
    }

    /**
     * Permet d'ajouter un enchantement à l'arme courante
     *
     * @param ench Enchant à appliquer à l'arme
     */
    protected void ajouterEnchant(Enchant ench) {
        if (!this.estStase) {
            listEnchant.add(ench);
            ench.placerEnchant(this);
        }
    }

    /**
     * Permet de réinitaliser l'arme à son état
        d'origine.
     */
    protected void reset() {
        this.listEnchantStase = new
            ArrayList<>(this.listEnchant);
        this.listEnchant = new ArrayList<>();
        this.degat = this.degatOrg;
        this.type = this.typeOrg;
```

```
}

/**
 * Permet d'avoir la représentation JSon d'une arme.
 *
 * @return le jSon associé à une arme
 */
@Override
public JsonObject toJSON() {
    JsonObjectBuilder obj =
       Json.createObjectBuilder();
    obj.add("Id", this.getCardID());
    obj.add("Type d'arme", type.name());
    obj.add("Degats", degat);
    Iterator<Enchant> it = listEnchant.iterator();
    int enchNum = 1;
    while (it.hasNext()) {
        obj.add("Enchantement actif #" + enchNum,
           it.next().toJSON());
        ++enchNum;
    }
    enchNum = 1;
    it = listEnchantStase.iterator();
    while (it.hasNext()) {
        obj.add("Enchantement inactif #" + enchNum,
           it.next().toJSON());
        ++enchNum;
    }

    return obj.build();
}

/**
 * Getter
 * @return Type d'arme
 */
public TypeArme getTypeArme(){
    return type;
}

/**
 * Permet de staser une Arme
 */
protected void staserArme() {
    this.estStase = true;
}
```

```
    /**
     * Setter notant que l'arme a été déployé.
     */
    protected void deployerArme() {
        armeUtilise = true;
    }

    /**
     * Permet de savoir si une arme est Stase
     *
     * @return true si l'arme est stase false sinon
     */
    public boolean peutAjouterEnchantement() {
        return !estStase;
    }

    /**
     * Getter
     * @return Bool dictant si l'arme est déployé sur un
        perso.
     */
    public boolean armeEstDeploye() {
        return armeUtilise;
    }
}
```

1.4 Enchant.Java

```
package cardgame.JeuxCartes;

import javax.json.*;
/**
 * Classe abstraite, Enchant sert d'interface commun
    pour tout les types
 * d'enchantements du jeu.
 *
 * Les implémentations de cette classes sont basés sur
    le patron Visiteur,
 * afin de pouvoir ajouter dynamiquement une nouvelle
    opération à arme
 * (ajout l'enchantement) sans toutefois modifier sa
    classe.
 * Ceci nous permet d'assurer que tout ajouts
    d'enchantements basés
 * sur des valeurs d'armes existantes n'auront pas
    besoin de modifier Arme.
```

```
 * (https://sourcemaking.com/design_patterns/visitor)
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public abstract class Enchant extends Carte {

    private final String description;

    public Enchant(String desc) {
        super();
        description = desc;
    }

     /**
     * Déclaration de la fonction placerEnchant, celle
        ci n'a aucun effet
     *
     * @param arme Arme sur laquel placer enchant
     */
    protected abstract void placerEnchant(Arme arme);


    /**
     * Permet d'avoir la représentation de la carte
        d'enchantement.
     *
     * @return le JSon représentant l'enchant de la carte
     */
    @Override
    public JsonObject toJSON() {
        JsonObjectBuilder obj =
            Json.createObjectBuilder();
        obj.add("Id", this.getCardID());
        obj.add("Nom",
            this.getClass().getCanonicalName());
        obj.add("Description", description);

        return obj.build();
    }
```

```
}
```

1.5 EnchantStase.Java

```
package cardgame.JeuxCartes;

/**
 * Implémentation de la classe abstraite Enchant.
 *
 * EnchantStase permet de placer l'effet de stase sur
    une arme.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class EnchantStase extends Enchant {

    public EnchantStase() {
        super("Cette carte applique un effet de Stase
           sur l'arme choisi.");
    }

    /**
     * Modifie l'arme et la met en stase.
     *
     * @param arme arme qui va être mise sous stase.
     */
    @Override
    protected void placerEnchant(Arme arme) {
        arme.staserArme();
        arme.reset();
    }
}
```

1.6 EnchantNeutre.Java

```
package cardgame.JeuxCartes;

import cardgame.Regles.TypeArme;

/**
 * Implémentation de la classe abstraite Enchant.
```

```
 *
 * EnchantNeutre permet de changer le type de l'arme.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class EnchantNeutre extends Enchant {

    public EnchantNeutre() {
        super("Cette carte rend cette arme neutre.");
    }

    /**
     * Modifie l'arme sur lequel le triangle des dégats
        ne sera plus appliqué.
     *
     * @param arme arme dont le triangle de dégat va
        etre retiré
     */
    @Override
    protected void placerEnchant(Arme arme) {
        arme.type = TypeArme.Neutre;
    }
}
```

1.7 EnchantFacile.Java

```
package cardgame.JeuxCartes;


/**
 * Implémentation de la classe abstraite Enchant.
 *
 * EnchantFacile permet de changer les utilisateurs
    possibles de l'arme.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class EnchantFacile extends Enchant {
```

```
    public EnchantFacile() {
        super("Cette carte rend cette arme utilisable
            par tout le monde.");
    }

    /**
     * Applique l'enchantement sur l'arme passé en
        paramètre.
     *
     * @param arme arme qui pourra être équipée par tout
        le monde
     */
    @Override
    protected void placerEnchant(Arme arme) {
        if (!arme.armeEstDeploye()) {
            arme.estFacile = true;
        }
    }
}
```

1.8 EnchantDegatPlus.Java

```
package cardgame.JeuxCartes;

/**
 * Implémentation de la classe abstraite Enchant.
 *
 * EnchantDegatPlus permet d'augmenter la force d'une
    arme.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class EnchantDegatPlus extends Enchant {

    public EnchantDegatPlus() {
        super("Cette carte augmente les degats de l'arme
            choisi par un.");
    }

    /**
     * Applique l'enchantement sur l'arme passé en
        paramètre.
```

```
     *
     * @param arme arme dont les degats vont etre
        augmenté
     */
    @Override
    protected void placerEnchant(Arme arme) {
        arme.degat++;
    }
}
```

1.9 EnchantDegatMoins.Java

```
package cardgame.JeuxCartes;

/**
 * Implémentation de la classe abstraite Enchant.
 *
 * EnchantDegatMoins permet d'abaisser la force d'une
    arme.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class EnchantDegatMoins extends Enchant {

    public EnchantDegatMoins() {
        super("Cette carte abaisse les dommages de
           l'arme choisi par 1.");
    }

    /**
     * Applique l'enchantement sur l'arme passé en
        paramètre.
     *
     * @param arme arme dont les degats vont etre diminué
     */
    @Override
    protected void placerEnchant(Arme arme) {
        arme.degat--;
    }
}
```

1.10 Cible.Java

```java
package cardgame.JeuxCartes;

import cardgame.ResultUtils.Resultat;

/**
 * Interface utilisé pour les fonctions reliés au recoit
    de coups.
 * Cette interface nous permet de généraliser les appels
    d'attaques au xperso et Joueurs.
 * (L'idée provient de l'équipe Philippe Pépos
    PetitClerc et Mehdi Ait Younes)
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 * 12-Fév-2016 : 1.0 - Version initiale.
 */
public interface Cible {

    /**
     * @return True si la cible peut actuellement être
        attaqué.
     * (Ex un Joueur doit avoir un jeu vide.)
     */
    public abstract boolean peutEtreAttaque();
    /**
     * recoitAttaque applique le dommage reÃğu, vérifie
        si le coup était
     * fatal et retourne le résultat du coup.
     * @param attaqueur Le combattant qui attaque la
        cible.
     * @return Le résultat du coup reÃğu (Soit
        AttackPerso ou AttackJoueur)
     */
    public abstract Resultat recoitAttaque(Combattant
       attaqueur);

    /**
     * @return True si la cible est morte.
     */
    public abstract boolean estMort();

}
```

1.11 Soigneur.Java

```java
package cardgame.JeuxCartes;
```

```java
import cardgame.ResultUtils.SoinsResult;

/**
 * Interface utilisé pour les fonctions reliés aux soins.
 * Cette interface nous permet de découpler la logique
    des sortilèges de soins
 * des Personnages, ce qui nous permettrai d'ajouter de
    nouveaux métiés axés sur
 * le support et les effets de status (Haste,Vitality
    etc...)
 * (L'idée provient de l'équipe Philippe Pépos
    PetitClerc et Mehdi Ait Younes)
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 * 12-Fév-2016 : 1.0 - Version initiale.
 */
public interface Soigneur {

     /**
      * Permet au perso de soigner un allié.
      * Pour soigner, le perso a besoin d'avoir encore
         des points de magie.
      *
      * @param p Personnage allié à soigner.
      * @return un SoinsResult si le soin à réussi,
         RefuseResult sinon.
      */
    public abstract SoinsResult soigner (Perso p);

    /**
     * Vérifie si le soigneur est capable de faire le
        sort de soins.
     * @param p Le perso à soigner.
     * @return True si le sort de soins peut être é
        ffectué.
     */
    public abstract boolean peutSoigner(Perso p);

}
```

1.12 Combattant.Java

```java
package cardgame.JeuxCartes;
```

```java
import cardgame.Regles.TypeArme;
import cardgame.ResultUtils.Resultat;
import java.util.List;

/**
 * Classe abstraite qui extend Perso, Combattant nous
    permet de
 * d.coupler la logique d'attaque hors des perso. Ceci
    nous permettrait
 * alors dans le futur d'ajouter des classes qui ne
    peuvent attaquer,
 * tel des troubadour ou Bardes.
 *
 * (Inspiré par les travaux de Phillipe Pépos PetitClerc
    et Zerrouk Rahdia.)
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.2
 *
 * Historique :
 * 14-Fév-2016 : 1.0 - Version initiale
 */
public abstract class Combattant extends Perso {

    public Combattant(int _hp, int _mp, List<TypeArme>
       armes) {
        super(_hp, _mp, armes);
    }

    /**
     * Fonction qui calcule le nombre de dégats fait à
        l'opposant armé.
     * @param ta Type d'arme de l'opposant.
     * @return Le nombre de dégats.
     */
    public int forceAttaque(TypeArme ta) {
        return this.getArme().forceAttaque(ta);
    }

    /**
     * Cette fonction effectue l'attaque d'une cible.
     * @param c Instance de la cible attaquée.
     * @return Soit unAttaquePersoResult
     * ou AttaquePlayerResult décrivant le coup.
```

```
     */
    public Resultat Attaque(Cible c) {
        return c.recoitAttaque(this);
    }

}
```

1.13 Guerrier.Java

```
package cardgame.JeuxCartes;

import cardgame.Regles.*;
import java.util.Arrays;
import javax.json.*;

/**
 * Classe représentant la classe Guerrier du jeu. La
    classe est une extension de
 * Combattant, ce qui lui permet d'attaquer.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 * 12-Fév-2016 : 1.0 - Version initiale.
 */
public class Guerrier extends Combattant {

    public Guerrier() {
        super(Regle.GUERRIERHP, Regle.GUERRIERMP,
            Arrays.asList(TypeArme.values()));
    }

    @Override
    public JsonObject toJSON() {
        JsonObject json = super.toJSON();
        JsonObjectBuilder addition =
            Json.createObjectBuilder();
        addition.add("Type Personnage", "Guerrier");
        addition.add("General Info", json);
        return addition.build();
    }
}
```

1.14 Pretre.Java

```
package cardgame.JeuxCartes;
```

```
import cardgame.Regles.*;
import cardgame.ResultUtils.SoinsResult;
import java.util.Arrays;
import javax.json.*;

/**
 * Classe représentant la classe Pretre du jeu. La
    classe est une extension de
 * Combattant et implémente soigneur, ce qui lui permet
    d'attaquer et soigner.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0 12-Fév-2016 : 1.0 - Version initiale.
 */
public class Pretre extends Combattant implements
   Soigneur {

    public Pretre() {
        super(Regle.PRETREHP, Regle.PRETREMP,
            Arrays.asList(TypeArme.Contondant,
            TypeArme.Neutre));
    }

    @Override
    public SoinsResult soigner(Perso allie) {
        SoinsResult resultat;
        this.utiliserMagie();
        allie.recevoirSoins();
        resultat = new SoinsResult(true,
            this.getCardID(), allie.getCardID());
        return resultat;
    }

    @Override
    public boolean peutSoigner(Perso p) {
        return this.getMp() > 0 && (this.getCardID() !=
            p.getCardID());
    }

    @Override
    public JsonObject toJSON() {
        JsonObject json = super.toJSON();
        JsonObjectBuilder addition =
            Json.createObjectBuilder();
```

```
            addition.add("Type Personnage", "Pretre");
            addition.add("General Info", json);

            return addition.build();
        }
    }
```

1.15 Paladin.Java

```
package cardgame.JeuxCartes;

import cardgame.Regles.*;
import cardgame.ResultUtils.SoinsResult;
import java.util.Arrays;
import javax.json.*;

/**
 * Classe représentant la classe Paladin du jeu. La
    classe est une extension de
 * Combattant et implémente soigneur, ce qui lui permet
    d'attaquer et soigner.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 * 12-Fév-2016 : 1.0 - Version initiale.
 */
public class Paladin extends Combattant implements
   Soigneur {

    public Paladin() {
        super(Regle.PALADINHP, Regle.PALADINMP,
            Arrays.asList(TypeArme.values()));
    }

    @Override
    public SoinsResult soigner(Perso allie) {
        SoinsResult resultat;
        this.utiliserMagie();
        allie.recevoirSoins();
        resultat = new SoinsResult(true,
            this.getCardID(), allie.getCardID());
        return resultat;
    }

    @Override
```

```
    public boolean peutSoigner(Perso p) {
        return this.getMp() > 0 && (this.getCardID() !=
            p.getCardID());
    }

    @Override
    public JsonObject toJSON() {
        JsonObject json = super.toJSON();
        JsonObjectBuilder addition =
            Json.createObjectBuilder();
        addition.add("Type Personnage", "Paladin");
        addition.add("General Info", json);
        return addition.build();
    }

}
```

1.16 Deck.Java

```
package cardgame.JeuxCartes;

import cardgame.Regles.Regle;
import cardgame.Init.*;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;
import javax.json.*;

/**
 * Classe représentant le paquet de cartes non-pigés
    d'un joueur. La classe
 * permet d'initialiser le deck et de traiter la logique
    de pioche et de vie
 * (puisque les points de vie sont == au nombre de
    cartes restantes.) sans
 * donner accès à cette logique au joueur.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0 08-Fév-2016 : 1.0 - Version initiale.
 *          1.1 10-Fév-2016 : 1.1 - Modification de
    InitialiserDeck
 *                                pour utiliser
    PersoFactory.
 */
```

```java
public class Deck {

    /**
     * Structure représentant le deck lui-même.
     */
    private final List<Carte> cartespioches;

    public Deck(){
        cartespioches = new ArrayList<>();
        initialiserDeck();
    }

    /**
     * Permet de créer le deck et d'initialiser son
        contenu.
     * La classe initialise le nombre de cartes
        nécessaire de chaque type
     * selon les règles du jeu et ensuite mélange le
        deck.
     */
    private void initialiserDeck() {
        ArmeFactory createurArmes = new ArmeFactory();
        EnchantFactory createurEnchants = new
            EnchantFactory();
        PersoFactory createurPersos = new PersoFactory();

        cartespioches.addAll(createurPersos.creerSetGuerrier(Regle.CARTEGUERRIE
        cartespioches.addAll(createurPersos.creerSetPretre(Regle.CARTEPRETRE));
        cartespioches.addAll(createurPersos.creerSetPaladin(Regle.CARTEPALADIN)
        cartespioches.addAll(createurArmes.creerSetArmes(Regle.CARTEARMEUN,
            1));
        cartespioches.addAll(createurArmes.creerSetArmes(Regle.CARTEARMEDEUX,
            2));
        cartespioches.addAll(createurEnchants.creerSetEnchants(Regle.CARTEENCHA

        Collections.shuffle(cartespioches, new
            Random(System.nanoTime()));
    }

    /**
     * Fonction qui permet de vider le deck.
     */
    public void viderDeck(){
        this.cartespioches.clear();
    }
```

```java
/**
 * Permet de piocher une liste de carte.
 * @param nbCartes nombre de carte à piocher
 * @return Liste des cartes piochées
 */
public List<Carte> piocherCarte(int nbCartes) {

    /*On s'assure de piocher le min entre le nombre
       de cartes restantes,
    le nombre demandé ou le nombre maximal dans une
       main.*/
    int nbAPiocher = Math.min(nbCartes,
       this.carteRestantes());
    nbAPiocher = Math.min(nbAPiocher,
       Regle.CARTEMAIN);

    List<Carte> nouvCartes = new ArrayList<>();

    while ( nbAPiocher != 0) {
        nouvCartes.add(cartespioches.remove(0));
        --nbAPiocher;
    }

    return nouvCartes;
}

/**
 * Permet d'appliquer les dégats recu par un joueur
    sur son Deck.
 * @param nbDegatCarte Le nombre de dégat pris
 * @return la Liste des cartes perdues
 */
public List<Carte> dommageJoueur(int nbDegatCarte) {
    return piocherCarte(nbDegatCarte);
}

/**
 * Permet de savoir le nombre de carte réstante dans
    la pioche.
 * @return le nombre de cartes encore présentes dans
    la pioche.
 */
public int carteRestantes() {
    return cartespioches.size();
}
```

```
    /**
     * @return True si le deck est vide.
     */
    public boolean deckEstVide(){
        return cartespioches.isEmpty();
    }

    /**
     * Pemet d'avoir la représentation en JSon du Deck
     * A noter : Le contenu du deck n'est pas communiqué
        pour éviter la triche.
     * @return la représentation du Deck en JSon
     */
    public JsonObject toJSon() {
        JsonObjectBuilder obj =
           Json.createObjectBuilder();
        obj.add("Nombre de cartes restantes a piger",
           cartespioches.size());
        return obj.build();
    }
}
```

1.17 Joueur.Java

```
package cardgame.JeuxCartes;

import cardgame.ResultUtils.Resultat;
import cardgame.Regles.*;
import cardgame.ResultUtils.*;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.CopyOnWriteArrayList;
import javax.json.*;

/**
 * Classe utilisé par l'API pour placé le coup du joueur
    reÃğu du controlleur.
 * Comparé à Jeux qui sert de facade pour le
    controlleur, Joueur traite les
 * appels des joueurs dans le modèle et retourne ces
    conséquences.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
```

```
 * @version 1.1
 * Historique : 08-Fév-2016 : 1.0 - Version initiale.
 *              13-Fév-2016 : 1.1 - Réécriture des
    fonctions pour
 *                                  fonctionner avec
    cible.
 */
public class Joueur implements Cible {

    private final int idJoueur;
    private final Deck carteDeck;
    private final Map<Integer, Carte> main;
    private final Map<Integer, Perso> carteEnJeu;
    private final List<Carte> cimetiere;

    public Joueur(int i) {
        idJoueur = i;
        carteDeck = new Deck();
        main = new ConcurrentHashMap<>();
        cimetiere = new CopyOnWriteArrayList<>();
        carteEnJeu = new ConcurrentHashMap<>();
        List<Carte> mainDeb =
           carteDeck.piocherCarte(Regle.CARTEMAIN);
        for (Carte c : mainDeb) {
            main.put(c.getCardID(), c);
        }
    }

    /**
     * @return L'identifiant unique du joueur.
     */
    public int getIdjoueur() {
        return idJoueur;
    }

    /**
     * Permet d'obtenir le deck du joueur
     *
     * @return le deck du joueur
     */
    public Deck getCarteDeck() {
        return carteDeck;
    }

    /**
     * Permet d'avoir la main du joueur
```

```
     *
     * @return la liste de Carte contenu dans la main du
        joueur
     */
    public Map<Integer, Carte> getMain() {
        return main;
    }

    /**
     * Pemet d'avoir les cartes en jeu du joueur
     *
     * @return la liste de carte présente sur le jeu,
        cartes associées au joueur
     */
    public Map<Integer, Perso> getCarteEnJeu() {
        return carteEnJeu;
    }

    /**
     * Permet d'avoir le cimetiere du joueur
     *
     * @return la liste de carte présente dans le
        cimetiere du joueur
     */
    public List<Carte> getCimetiere() {
        return cimetiere;
    }

    /**
     * Permet de savoir si le joueur à perdu
     *
     * @return true si le joueur à perdu (autrement dit
        si il n'a plus de carte
     * null part ) [Cimetiere non compris] false sinon
     */
    public boolean aPerdu() {
        return (main.isEmpty() && carteEnJeu.isEmpty()
            && carteDeck.deckEstVide());
    }

    /**
     * Permet au joueur de defausser une liste de Carte
     *
     * @param defausse liste des cartes à défausser
     * @return Un DefausseResult si la defausse s'est
        bien passé Un
```

```
     * RefusedResult sinon
     */
    public Resultat defausserCartes(List<Carte>
        defausse) {
        Resultat res;

        for (Carte c : defausse) {
            cimetiere.add(main.remove(c.getCardID()));
        }

        res = new DefausseResult(this.getIdjoueur(),
            true, defausse);
        return res;
    }

    /**
     * Permet au joueur de piocher des cartes
     *
     * @return un PiocheResult si tout s'est bien passé
        un RefusedResult sinon
     */
    public Resultat piocher() {
        int nbAPiocher = Regle.CARTEMAIN - main.size();
        nbAPiocher = Math.max(nbAPiocher, 0);
        List<Carte> lc =
            carteDeck.piocherCarte(nbAPiocher);

        for (Carte c : lc) {
            main.put(c.getCardID(), c);
        }

        return new PiocheResult(this.getIdjoueur(),
            true, lc);
    }

    /**
     * Permet à un joueur d'attaquer un joueur à l'aide
        d'une de ses cartes
     *
     * @param attaqueur position de la carte attaquant
        sur le jeu
     * @param attaque Joueur à attaquer
     * @return un AttackResult si tout c'est bien passé
        un RefusedResult sinon
     */
    public Resultat attaque(Combattant attaqueur, Cible
        attaque) {
```

```
        Resultat res;

        res = attaqueur.Attaque(attaque);

        return res;
    }

    /**
     * Fonction auxiliaire appelé après chaque attaque
        reÃğu, MAJCartesPlancher
     * enlève les perso mort de la partie.
     */
    public void MAJCartesPlancher() {
        for (Perso pers : this.carteEnJeu.values()) {
            if (pers.estMort()) {
                cimetiere.add(carteEnJeu.remove(pers.getCardID()));
            }
        }
    }

    /**
     * Permet d'ajouter une liste d'enchant à un joueur
     *
     * @param enchs liste des positions des enchants
        dans la main
     * @param carteTouchee carte étant affectée par
        l'enchant
     * @return une Liste de Result, chaqu'un étant soit
        un EnchantResult si tout
     * c'est bien passé sinon un RefusedResult
     */
    public Resultat ajouterEnchants(List<Enchant> enchs,
       Carte carteTouchee) {
        Resultat res;
        Arme arm;
        if (carteTouchee instanceof Perso) {
            arm = ((Perso) carteTouchee).getArme();
        } else if (carteTouchee instanceof Arme) {
            arm = (Arme) carteTouchee;
        } else {
            return new RefuseResult("Erreur interne.");
        }

        for (Enchant ench : enchs) {
            arm.ajouterEnchant(ench);
            cimetiere.add(ench);
```

```
            main.remove(ench.getCardID());
        }
        res = new EnchantResult(true, carteTouchee,
            enchs);
        return res;
    }

    /**
     * Permet au joueur de placer un personnage en jeu
     *
     * @param personnage position dans la main du
        personnage à jouer
     * @param arm arme à équiper au perso
     * @param ench Liste des enchants à ajouter à l'arme
     * @return un PersoDeploieResult si tout c'est bien
        passé un Refusedresult
     * sinon
     */
    public Resultat placerPerso(Perso personnage, Arme
        arm, List<Carte> ench) {
        Resultat res;

        for (Carte c : ench) {
            Enchant en = (Enchant) c;
            arm.ajouterEnchant(en);
            cimetiere.add(main.remove(en.getCardID()));
        }

        personnage.equiperArme(arm);
        main.remove(personnage.getCardID());
        main.remove(arm.getCardID());
        carteEnJeu.put(personnage.getCardID(),
            personnage);

        res = new PersoDeploieResult(this.getIdjoueur(),
            true, personnage);
        return res;
    }

    /**
     * @param car La carte qu'on veut vérifier
        l'emplacement
     * @return True si la carte est dans la main du
        joueur.
     */
    public boolean carteDansMain(Carte car) {
```

```
        return main.containsKey(car.getCardID());
    }

    /**
     * @param car La carte qu'on veut vérifier
        l'emplacement
     * @return True si la carte est dans la jeu du
        joueur.
     */
    public boolean carteDansJeu(Carte car) {
        return carteEnJeu.containsKey(car.getCardID());
    }

    /**
     * @return True si le joueur a un deck vide
     */
    public boolean destVide() {
        return carteDeck.deckEstVide();
    }

    /**
     * Permet au joueur de declarerForfait Autrement
        dit, de passer toutes ces
     * cartes dans le cimetiere.
     *
     * @return RefusedResult si le joueur à déjà perdu
        FinDePartieResult si le
     * joueur à déclarer forfait
     */
    public Resultat declarerForfait() {
        Resultat res;
        main.clear();
        carteDeck.viderDeck();
        carteEnJeu.clear();
        res = new FinDePartieResult(this.getIdjoueur(),
            true, -1);
        return res;
    }

    /**
     * Permet d'effectué l'action de soin sur un
        personnage présent sur le jeu.
     *
     * @param soins Carte effectuant le soin
     * @param soignee Position dans la liste des
        carteEnJeu du soignee
```

```
     * @return RefusedResult si le joueur ne peux pas
         soigner le personnage
     * soignee SoinsResult si le joueur peu être soigné
     */
    public Resultat soignerPerso(Soigneur soins, Perso
        soignee) {

        if (!soins.peutSoigner(soignee)) {
            return new RefuseResult("Le soins ne peut
                pas être éffectué.");
        }

        return soins.soigner(soignee);
    }

    /**
     * Permet d'avoir le JSon associé à un joueur
     *
     * @return le JSon objet représentant le joueur
     */
    public JsonObject toJSON() {
        JsonObjectBuilder obj =
            Json.createObjectBuilder();
        obj.add("main", this.mainToJSon());
        obj.add("cimetiere", this.cimetiereToJSon());
        obj.add("deck", this.deckToJSon());

        return obj.build();
    }

    /**
     * Permet d'avoir le JSon associé au contenu de la
         main du joueur
     *
     * @return le JSon associé à la main
     */
    private JsonObject mainToJSon() {
        JsonObjectBuilder obj =
            Json.createObjectBuilder();

        Iterator<Carte> cd = main.values().iterator();
        int numCarte = 1;
        while (cd.hasNext()) {
            obj.add("carte #" + numCarte,
                cd.next().toJSON());
            ++numCarte;
```

```
        }

        return obj.build();
    }

    /**
     * Permet d'avoir le JSon associé au contenu du
        cimetiere du joueur
     *
     * @return le JSon associé au cimetiere du joueur
     */
    private JsonObject cimetiereToJSon() {
        JsonObjectBuilder obj =
            Json.createObjectBuilder();

        Iterator<Carte> cd = cimetiere.iterator();
        int numCarte = 1;
        while (cd.hasNext()) {
            obj.add("carte #" + numCarte,
                cd.next().toJSON());
            ++numCarte;
        }

        return obj.build();
    }

    /**
     * Permet d'avoir le contenu du Deck en format jSon
     *
     * @return le JSon associé au contenu du Deck du
        joueur
     */
    private JsonObject deckToJSon() {
        return carteDeck.toJSon();
    }

    /**
     * @return True si le joueur a le droit de piocher.
     */
    public boolean peutPiocher() {
        return ((main.size() < Regle.CARTEMAIN) &&
            (!carteDeck.deckEstVide()));
    }

    @Override
    public boolean peutEtreAttaque() {
```

```
        return carteEnJeu.isEmpty();
    }

    @Override
    public AttaquePlayerResult recoitAttaque(Combattant
       attaqueur) {
        int degat =
            attaqueur.forceAttaque(TypeArme.Neutre);
        List<Carte> cartePerdus =
            this.carteDeck.dommageJoueur(degat);
        for (Carte c : cartePerdus) {
            cimetiere.add(c);
        }
        AttaquePlayerResult res = new
            AttaquePlayerResult(degat, idJoueur,
            attaqueur.getCardID(), idJoueur, estMort());
        return res;
    }

    @Override
    public boolean estMort() {
        return ((carteDeck.deckEstVide() &&
            main.isEmpty()) && (carteEnJeu.isEmpty()));
    }

}
```

2 Init

2.1 ArmeFactory.Java

```
package cardgame.Init;

import cardgame.JeuxCartes.Arme;
import cardgame.Regles.TypeArme;
import java.util.ArrayList;
import java.util.List;

/**
 * Classe utilisé pour instancier correctement les
    cartes d'armes dans une
 * partie. ArmeFactory, classe basé sur le patron
    Factory, nous permet de
 * découpler et cacher la logique de création des cartes
    des classes reliés à
 * l'API.
 *
```

```
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class ArmeFactory {

    /**
     * Permet de créer une liste d'arme
     *
     * @param nbCopies Nombre d ecopies de chacunes des
        types d'armes à
     * instancier.
     * @param degats Nombre de dégats fait par ces armes.
     * @return Liste de nbCopie élements contenant les
        armes demandées.
     */
    public List<Arme> creerSetArmes(int nbCopies, int
       degats) {
        List<Arme> armes = new ArrayList<>();

        for (int copieAct = 0; copieAct < nbCopies;
           copieAct++) {
            armes.add(new Arme(TypeArme.Contondant,
               degats));
            armes.add(new Arme(TypeArme.Perforant,
               degats));
            armes.add(new Arme(TypeArme.Tranchant,
               degats));
        }

        return armes;
    }
}
```

2.2 EnchantFactory.Java

```
package cardgame.Init;

import cardgame.JeuxCartes.*;
import java.util.ArrayList;
import java.util.List;

/**
 * Classe utilisé pour instancier correctement les
    cartes d'enchantements dans
```

```
 * une partie. EnchantFactory, classe basé sur le patron
    Factory, nous permet de
 * découpler et cacher la logique de création des cartes
    de l'API.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class EnchantFactory {

    /**
     * Permet de créer un ensemble de tout les types
        d'enchantements du jeu.
     *
     * @param nbCopies nombre de copie des cartes
        Enchants à instancier.
     * @return Une liste de nbCopie élements qui
        contient tout les enchants.
     */
    public List<Enchant> creerSetEnchants(int nbCopies) {
        List<Enchant> enchantements = new ArrayList<>();

        for (int copieAct = 0; copieAct < nbCopies;
            copieAct++) {
            enchantements.add(new EnchantNeutre());
            enchantements.add(new EnchantStase());
            enchantements.add(new EnchantDegatPlus());
            enchantements.add(new EnchantDegatMoins());
            enchantements.add(new EnchantFacile());
        }

        return enchantements;
    }
}
```

2.3 PersoFactory.Java

```
package cardgame.Init;

import cardgame.JeuxCartes.Guerrier;
import cardgame.JeuxCartes.Paladin;
import cardgame.JeuxCartes.Perso;
import cardgame.JeuxCartes.Pretre;
```

```java
import java.util.ArrayList;
import java.util.List;

/**
 * Classe basé sur le patron Factory utilisé pour
    instancier des cartes
 * personnages. La classe est écrite de telle manière
    que tout ajout de nouveaux
 * personnages dans le jeu nécessite seulement d'être
    ajouté comme fonction dans
 * cette classe ainsi que d'inscrire ces règles dans la
    classe Règle.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.1
 *
 * Historique : 8 Fév-2016 : 1.0 Version initiale.
 * 13Fév-2016 : 1.1 Modification du code pour marcher
    avec les
 * nouvelles classes Guerrier, Pretre et Paladin.
 */
public class PersoFactory {

    /**
     * Retourne une liste de cartes Guerriers
     *
     * @param nbCopies nombre de copie de cartes
        guerriers
     * @return liste de nbCopies d'instance de Guerriers.
     */
    public List<Perso> creerSetGuerrier(int nbCopies) {
        List<Perso> guerriers = new ArrayList<>();

        for (int copieAct = 0; copieAct < nbCopies;
            copieAct++) {
            guerriers.add(new Guerrier());
        }

        return guerriers;
    }

    /**
     * Retourne une liste de cartes Prêtres.
     *
     * @param nbCopies nombre de copie de cartes prêtres.
```

```
     * @return liste de nbCopies d'instance de Prêtres.
     */
    public List<Perso> creerSetPretre(int nbCopies) {
        List<Perso> pretres = new ArrayList<>();

        for (int copieAct = 0; copieAct < nbCopies;
            copieAct++) {
            pretres.add(new Pretre());
        }

        return pretres;
    }

    /**
     * Retourne une liste de cartes Paladins.
     *
     * @param nbCopies nombre de copie de cartes
        paladins.
     * @return liste de nbCopies d'instance de Paladins.
     */
    public List<Perso> creerSetPaladin(int nbCopies) {
        List<Perso> paladins = new ArrayList<>();

        for (int copieAct = 0; copieAct < nbCopies;
            copieAct++) {
            paladins.add(new Paladin());
        }

        return paladins;
    }
}
```

3 Regles

3.1 Regle.Java

```
package cardgame.Regles;

/**
 * Classe non instantiable, Regle sert de conteneur pour
    les valeurs
 * paramétrisables du jeu.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
```

```
 * Historique :
 *
 * -8 Fév-2016 : 1.0 Version initiale.
 */
public class Regle {

    private Regle() { }

    /**
     * Liste de toutes les règles de jeu.
     */
    public static final int GUERRIERHP = 5;
    public static final int GUERRIERMP = 0;
    public static final int PRETREHP = 3;
    public static final int PRETREMP = 3;
    public static final int PALADINHP = 4;
    public static final int PALADINMP = 1;
    public static final int CARTEGUERRIER = 4;
    public static final int CARTEPRETRE = 4;
    public static final int CARTEPALADIN = 2;
    public static final int CARTEARMEUN = 2;
    public static final int CARTEARMEDEUX = 2;
    public static final int CARTEENCHANTEMENT = 2;
    public static final int CARTEMAIN = 5;
}
```

3.2 TypeArme.Java

```
package cardgame.Regles;

/**
 * Enum, le type de données TypeArme contient chaque
    type d'armes possibles dans
 * le jeu, ainsi que leurs logiques personnelles, tel
    que leurs
 * forces/faiblesses. La classe détient aussi la
 * fonction de calcul du triangle d'attaque pour fins
    d'évolutions faÃğiles.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * Historique :
 *
 * -8 Fév-2016 : 1.0 Version initiale.
```

```
 */
public enum TypeArme {

    /**
     * Déclaration de l'enum des types d'armes possibles
        dans le jeu. On décrit
     * en même temps leur forces,faiblesses et leurs
        utilisables possibles. Les
     * forces/faiblesses sont décrits en String pour
        fins de visibilité humaine.
     */
    Contondant("Contondant", "Perforant", "Tranchant"),
    Perforant("Perforant", "Tranchant", "Contondant"),
    Tranchant("Tranchant", "Contondant", "Perforant"),
    Neutre("Neutre", "", "");

    private final String nom;
    private final String force;
    private final String faiblesse;

    TypeArme(String nom, String force, String faiblesse)
       {
        this.nom = nom;
        this.force = force;
        this.faiblesse = faiblesse;
    }

    /**
     * Définit le calcul de modificateur du triangle
        d'armes.
     *
     * @param armeEnnemi Type d'arme de l'ennemi
     * @return 1 si l'arme actuelle est la faiblesse de
        l'arme ennemi, -1 si
     * l'arme ennemi est la faiblesse de l'arme
        actuelle, Sinon 0.
     */
    public int calculModificateur(TypeArme armeEnnemi) {
        if (this.force.equals(armeEnnemi.nom)) {
            return 1;
        } else if
           (this.faiblesse.equals(armeEnnemi.nom)) {
            return -1;
        }

        return 0;
```

```
    }
}
```

4 API

4.1 Jeux.Java

```
package cardgame.API;

import cardgame.JeuxCartes.Arme;
import cardgame.JeuxCartes.Combattant;
import cardgame.JeuxCartes.Carte;
import cardgame.JeuxCartes.Cible;
import cardgame.JeuxCartes.Enchant;
import cardgame.JeuxCartes.EnchantFacile;
import cardgame.JeuxCartes.Joueur;
import cardgame.JeuxCartes.Perso;
import cardgame.JeuxCartes.Soigneur;
import cardgame.ResultUtils.Resultat;
import cardgame.ResultUtils.RefuseResult;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;
import javax.json.Json;
import javax.json.JsonObject;
import javax.json.JsonObjectBuilder;

/**
 * Classe API utilisé pour communiquer entre le modèle
    et le controlleur. Toutes
 * les actions possibles par un joueur doit passer par
    une des fonctions de
 * cette classe. Afin d'éviter toutes triches, chaque
    action demandé par le
 * joueur est vérifié avant de s'effectuer. (Ex : Le
    joueur 2 essaie de jouer
 * pour le joueur . Le joueur 1 essaie de fair eune
    action impossible tel que
 * déployer des cartes de l'adversaire.) ) Si l'action
    est mauvaise, Jeux
 * retourne un RefuseResult. Sinon, l'action est é
    ffectué et Jeux retourne un
 * Resultat décrivant les conséquences de l'acte.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
```

```
 * @version 1.2
 * 08-Fév-2016 : 1.0 - Version initiale.
 * 10-Fév-2016 : 1.1 - Réécriture des fonctions afin de
    pouvoir retourner
 * les cartes au controlleur si nécessaire.
 * 13-Fév-2016 : 1.2 - Ajout de fonctions de validation
    de coup pour les contrÃ¶leurs
 *                      (Remerciments : L'idée est
    inspiré du travail de RUBIN Jehan et HAAS Ellen)
 */
public class Jeux {

    private final List<Joueur> joueurList;
    private boolean partieEnMarche;
    private int joueurTour;

    public Jeux() {
        joueurList = new ArrayList<>();
        partieEnMarche = false;
        joueurTour = -1;
    }

    public int getnbCartesDeck(int idJoueur) {
        assert (idJoueur < joueurList.size() - 1);
        return
            joueurList.get(idJoueur).getCarteDeck().carteRestantes();

    }

    /**
     * Permet de passer au joueur suivant
     */
    private void prochainJoueur() {
        joueurTour++;
        joueurTour = joueurTour % joueurList.size();
    }

    /**
     * Permet de savoir à qui est le tour ( id du joueur
        )
     *
     * @return l'id du joueur à qui c'est le tour de
        jouer
     */
    public int aQuiLeTour() {
        return joueurTour;
```

```
}

/**
 * Permet de démarrer une partie à nbJoueur
 *
 * @param nbJoueur nombre de joueur voulant jouer
 */
public void demarrerPartie(int nbJoueur) {
    assert (nbJoueur >= 2);
    if (!partieEnMarche) {
        finPartie();
        joueurTour = 0;
        for (int i = 0; i < nbJoueur; i++) {
            joueurList.add(new Joueur(i));
        }
    }

    partieEnMarche = true;
}

/**
 * Permet au joueur idJoueur de déclarer forfait
 *
 * @param idJoueur id du joueur déclarant forfait
 * @return
 */
public Resultat declarerForfait(int idJoueur) {
    if (joueurTour == idJoueur) {
        this.prochainJoueur();
        return
            joueurList.get(idJoueur).declarerForfait();
    } else {
        return new RefuseResult("Tu peux pas
            déclarer forfait pour quelqu'un
            d'autre.");
    }
}

/**
 * Permet de savoir l'état du jeu à tout moment (
    sous un format JSon)
 *
 * @return le contenu de chaque joueur
 */
public JsonObject getEtatJeu() {
    JsonObjectBuilder obj =
        Json.createObjectBuilder();
```

```
        obj.add("aQuiLeTour", this.joueurTour);
        obj.add("partieEnCours", this.partieEnMarche);

        Iterator<Joueur> j = joueurList.iterator();
        int numJoueur = 1;
        while (j.hasNext()) {
            obj.add("Joueur #" + numJoueur,
                j.next().toJSON());
            ++numJoueur;
        }

        return obj.build();
    }

    /**
     * Permet de savoir si la partie est finie
     *
     * @return true si la partie est fini, autrement dit
        si un joueur à gagné
     * false sinon
     */
    public boolean partieFini() {
        return getJoueurGagnant() != -1; // tout les
            joueurs ont perdu sauf 1
    }

    /**
     * Permet d'avoir l'id du joueur ayant gagné
     *
     * @return l'id du joueur gagnant -1 si aucun joueur
        n'a gagné
     */
    public int getJoueurGagnant() {
        int joueursPerdants = 0;
        int joueurGagnant = 0;
        for (int i = 0; i < joueurList.size(); i++) {
            if (!joueurList.get(i).aPerdu()) {
                joueurGagnant = i;
            } else {
                joueursPerdants++;
            }
        }

        return joueursPerdants == joueurList.size() -1?
            joueurGagnant : -1;
    }
```

```
/**
 * Permet au joueur idJoueur de piocher
 *
 * @param idJoueur id du joueur voulant piocher
 * @return une liste de Result, PiocheResult en cas
    de succes de la pioche
 * un RefusedResult sinon
 */
public Resultat piocherCartes(int idJoueur) {
    Resultat res;
    if (idJoueur != aQuiLeTour()) {
        res = new RefuseResult("Ce n'est pas le tour
           de ce joueur.");
        return res;
    }

    res = joueurList.get(idJoueur).piocher();
    return res;
}

public boolean peutPiocherCartes(int idJoueur) {
    return idJoueur == joueurTour &&
       joueurList.get(idJoueur).peutPiocher();
}

/**
 * Permet d'attaquer un perso présent sur le deck
 *
 * @param idJoueur id du joueur effectuant l'action
 * @param idAdversaire position relative sur le
    terrain de jeu de la carte
 * attaquante
 * @param attaqueur carte attaquant la seconde carte
 * @param receveur carte recevant le coupÅİ
 * @return un AttackResult si tout c'est bien passé
    un refusedResult sinon
 */
public Resultat attaquePerso(int idJoueur, int
   idAdversaire, Carte attaqueur, Carte receveur) {
    Resultat res;

    if (attaquePersoValide(idJoueur, idAdversaire,
       attaqueur, receveur)) {
        Combattant att = (Combattant) attaqueur;
        Cible attaquee = (Cible) receveur;
```

```
            res = joueurList.get(idJoueur).attaque(att,
                attaquee);
            joueurList.get(idAdversaire).MAJCartesPlancher();
            this.prochainJoueur();
        } else {
            res = new RefuseResult("L'attaque n'est pas
                possible");
        }

        return res;
    }

    public boolean attaquePersoValide(int idJoueur, int
        idAdversaire, Carte attaqueur, Carte receveur) {

        boolean coupP = ((this.aQuiLeTour() == idJoueur)
            && (idJoueur != idAdversaire));
        coupP = coupP && (idAdversaire >= 0) &&
            (joueurList.size() >= idAdversaire);
        coupP = coupP && (attaqueur instanceof
            Combattant) && (receveur instanceof Cible);
        if (coupP) {
            coupP = coupP &&
                joueurList.get(idJoueur).carteDansJeu(attaqueur);
            coupP = coupP &&
                joueurList.get(idAdversaire).carteDansJeu(receveur);
            coupP = coupP && ((Cible)
                receveur).peutEtreAttaque();
        }
        return coupP;

    }

    /**
     * Permet d'attaquer un perso présent sur le deck
     *
     * @param idJoueur id du joueur effectuant l'action
     * @param idAdversaire position relative sur le
        terrain de jeu de la carte
     * attaquante
     * @param attaqueur Carte effectuant l'attaque sur
        le joueur
     * @return un AttackResult si tout c'est bien passé
        un refusedResult sinon
     */
    public Resultat attaqueJoueur(int idJoueur, int
        idAdversaire, Carte attaqueur) {
```

```
        Resultat res;

        if (attaqueJoueurValide(idJoueur, idAdversaire,
            attaqueur)) {
            Combattant att = (Combattant) attaqueur;
            res = joueurList.get(idJoueur).attaque(att,
                joueurList.get(idAdversaire));
            this.prochainJoueur();
        } else {
            res = new RefuseResult("L'attaque n'est pas
                possible");
        }

        return res;
    }

    public boolean attaqueJoueurValide(int idJoueur, int
        idAdversaire, Carte attaqueur) {

        boolean verifValide = idJoueur == aQuiLeTour()
            && idAdversaire >= 0 && idAdversaire <=
            joueurList.size() && idJoueur != idAdversaire;
        if (verifValide) {
            verifValide =
                joueurList.get(idJoueur).getCarteEnJeu().containsKey(attaqueur.g
            verifValide = verifValide &&
                joueurList.get(idAdversaire).peutEtreAttaque();
            verifValide = verifValide && (attaqueur
                instanceof Combattant);
        }

        return verifValide;
    }

    /**
     * Permet d'ajouter une liste d'enchant sur un joueur
     *
     * @param idJoueur id du joueur qui va recevoir les
        enchants
     * @param carteTouche La carte qui va recevoir les
        enchants
     * @param enchant listes des positions relatives
        dans le deck des cartes
     * d'enchantements à appliquer
     * @return un list Result, chaqu'un contenant un
        EnchantResult si l'enchant
```

```
     * fonctionne un refused Result sinon
     */
    public Resultat ajouterEnchantements(int idJoueur,
        Carte carteTouche, List<Carte> enchant) {
         Resultat res;

         if (peutAjouterEnchantements(idJoueur,
             carteTouche, enchant)) {
              List<Enchant> enchs = new ArrayList<>();
              for (Carte c : enchant) {
                   enchs.add((Enchant) c);
              }
              res =
                  joueurList.get(idJoueur).ajouterEnchants(enchs,
                  carteTouche);
              this.prochainJoueur();
         } else {
              res = new RefuseResult("Vous ne pouvez pas
                  enchanter cette cartes avec votre
                  sélection.");
         }
         return res;
    }

    public boolean peutAjouterEnchantements(int
        idJoueur, Carte carteTouche, List<Carte>
        enchants) {
         boolean verifValide = idJoueur == aQuiLeTour();
         Joueur j = joueurList.get(idJoueur);
         boolean carteDeploye = false;
         for (Carte c : enchants) {
              verifValide = verifValide &&
                  j.carteDansMain(c);
         }
         for (Joueur joueurAct : joueurList) {
              carteDeploye = carteDeploye ||
                  joueurAct.carteDansJeu(carteTouche);
         }
         carteDeploye = carteDeploye && carteTouche
             instanceof Perso;
         if (carteDeploye) {
              Perso p = (Perso) carteTouche;
              carteDeploye =
                  p.getArme().peutAjouterEnchantement();
         }
```

```
        return verifValide && carteDeploye;
    }

    public Collection<Carte> getCartesMainJoueur(int
        idJoueur) {
        return
            joueurList.get(idJoueur).getMain().values();
    }

    public Collection<Perso> getCartesJeuJoueur(int
        idJoueur) {
        Perso p;
        return
            joueurList.get(idJoueur).getCarteEnJeu().values();
    }

    public Collection<Carte> getCimetiereJoueur(int
        idJoueur) {
        return joueurList.get(idJoueur).getCimetiere();
    }

    /**
     * Permet de placer un personnage
     *
     * @param idJoueur id relatif du joueur dans la
        liste Joueur List
     * @param personnage position relative dans la main
        du joueur à placer
     * @param arme position relative dans la main de
        l'arme a placer
     * @param enchants liste des cartes d'enchants à
        placer sur le perso
     * @return un PersoDeploieResult si tout ce passe
        bien un refusedResult
     * sinon
     */
    public Resultat placerPerso(int idJoueur, Carte
        personnage, Carte arme, List<Carte> enchants) {
        Resultat res;
        if (this.peutDeployerPerso(idJoueur, personnage,
            arme, enchants)) {
            res =
                joueurList.get(idJoueur).placerPerso((Perso)
                personnage, (Arme) arme, enchants);
            this.prochainJoueur();
```

```
        } else {
            res = new RefuseResult("Vous n'avez pas le
                droit d'utiliser une des cartes choisi.");
        }
        return res;

    }

    public boolean peutDeployerPerso(int idJoueur, Carte
        perso, Carte arme, List<Carte> enchants) {
        boolean verif = idJoueur == aQuiLeTour();
        boolean enchFac = false;
        Joueur j = joueurList.get(idJoueur);
        verif = verif && j.carteDansMain(perso) &&
            j.carteDansMain(arme);
        verif = verif && (perso instanceof Perso) &&
            (arme instanceof Arme);
        for (Carte c : enchants) {
            verif = verif && j.carteDansMain(c);
            verif = verif && c instanceof Enchant;
            enchFac = enchFac || c instanceof
                EnchantFacile;
        }
        if (verif) {
            Perso p = (Perso) perso;
            Arme a = (Arme) arme;
            verif = !a.armeEstDeploye();
            verif = verif && (a.peutUtiliserArme(p) ||
                enchFac);
        }

        return verif;

    }

    /**
     * Permet de défausser n Carte à partie de l'api
     *
     * @param idJoueur id du joueur souhaitant se
        défausser
     * @param defausse Liste des positions relatives
        dans la main des cartes à
     * defausser
     * @return un defausseResult si tout ce passe bien
        un refusedResult en cas
     * d'erreur
```

```
     */
    public Resultat defausserCartes(int idJoueur,
        List<Carte> defausse) {
         Resultat res;
         if (peutDefausserCartes(idJoueur, defausse)) {
             res =
                 joueurList.get(idJoueur).defausserCartes(defausse);
             this.prochainJoueur();
         } else {
             res = new RefuseResult("La liste est soit
                 vide, soit rempli de cartes pas situés
                 dans la main.");
         }

         return res;

    }

    public boolean peutDefausserCartes(int idJoueur,
        List<Carte> defausse) {

         boolean coupValide = idJoueur == aQuiLeTour();
         coupValide = coupValide && defausse != null;
         coupValide = coupValide && !defausse.isEmpty();
         Joueur joueurAct = joueurList.get(idJoueur);
         for (Carte c : defausse) {
             coupValide = coupValide &&
                 joueurAct.carteDansMain(c);
             if (!coupValide) {
                  break;
             }
         }

         return coupValide;

    }

    /**
     * Permet de soigner un personnage du jeu
     *
     * @param idJoueur id du joueur qui effectue l'action
     * @param soigneur id relatif de la position de la
         carte sur le terrain de
     * jeu
     * @param soignee id relatif de la position de la
         carte sur le terrain de
```

```
     * jeu ( carte à soigner)
     * @return un SoinsResult si tout c'est bien passé
        un RefusedResult sinon
     */
    public Resultat soignerPerso(int idJoueur, Carte
        soigneur, Carte soignee) {
        Resultat res;

        if (peutSoignerPerso(idJoueur, soigneur,
            soignee)) {
            Soigneur s = (Soigneur) soigneur;
            Perso p = (Perso) soignee;
            res =
                joueurList.get(idJoueur).soignerPerso(s,
                p);
            this.prochainJoueur();

        } else {
            res = new RefuseResult("Vous n'avez pas le
                droit de faire ce soin.");
        }

        return res;

    }

    public boolean peutSoignerPerso(int idJoueur, Carte
        soigneur, Carte soignee) {
        boolean coupValide = idJoueur == aQuiLeTour();
        coupValide = coupValide && soigneur != soignee;
        Joueur joueurAct = joueurList.get(idJoueur);
        coupValide = coupValide &&
            joueurAct.carteDansJeu(soigneur)
                 && joueurAct.carteDansJeu(soignee);
        coupValide = coupValide && soigneur instanceof
            Soigneur && soignee instanceof Perso;
        if (coupValide) {
            Soigneur s = (Soigneur) soigneur;
            Perso p = (Perso) soignee;
            coupValide = s.peutSoigner(p);
        }
        return coupValide;
    }

    /**
     * Permet de liberer les informations de jeu et
        reinialisé les valeurs de
```

```
     * l'api
     */
    public void finPartie() {
        joueurList.clear();
        joueurTour = 0;
        partieEnMarche = false;
    }
}
```

5 ResultUtils

5.1 Resultat.Java

```
package cardgame.ResultUtils;

/**
 * Interface utilisé pour communiquer aux joueurs les
    conséquences des coups joués/refusés.
 * Resultat est implémenté par une sous-classe pour
    chaque type de coup possible, afin de
 * pouvoir transmettre la totalité des informations
    pertinentes.
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0 08-Fév-2016 : 1.0 - Version initiale.
 */
public interface Resultat {

    /**
     * Déclaration de coupAMarcher, Interface Result
     *
     * @return rien, non déclarée ici
     */
    public boolean coupAMarcher();

    /**
     * Déclaration de getDescription, Interface Result
     *
     * @return rien, non déclarée ici
     */
    public String getDescription();

    /**
     * Déclaration de coupJouerPar, Interface Result
     *
     * @return rien, non déclarée ici
     */
```

```
    public int coupJouerPar();

    /**
     * Déclaration de setJoueur, Interface Result
     * Ce setter existe seulement afin d'éviter
        d'envoyer aux classes
     * Cartes le Id du joueur.
     * @param idJoueur non définie ici
     */
    public void setJoueur(int idJoueur);
}
```

5.2 AttaquePersoResult.Java

```
package cardgame.ResultUtils;

/**
 * Implémentation de Resultat pour décrire les
    conséquences d'une attaque
 * perso-joueur.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class AttaquePersoResult implements Resultat {

    private final int dommageRecu;
    private int attaqueur;
    private final int idCarte;
    private final int idCarteAttack;
    private final boolean attaqueTuer;
    private String desc;

    public AttaquePersoResult(int dmg, int joueurId, int
        carteId, int persoCoupId, boolean attaqueTue) {
        dommageRecu = dmg;
        attaqueur = joueurId;
        idCarte = carteId;
        attaqueTuer = attaqueTue;
        idCarteAttack = persoCoupId;
        desc = "L'attaque de la carte " + carteId + "sur
            " + persoCoupId + "a causé " + dmg +
            "dégats.\n";
```

```
        if (attaqueTue) {
            desc = desc + "Le perso à été tué.";
        }
    }

    public AttaquePersoResult(int dmg, int carteId, int
        persoCoupId, boolean attaqueTue) {
        dommageRecu = dmg;
        idCarte = carteId;
        attaqueTuer = attaqueTue;
        idCarteAttack = persoCoupId;
        desc = "L'attaque de la carte " + carteId + "sur
            " + persoCoupId + "a causé " + dmg +
            "dégats.\n";
        if (attaqueTue) {
            desc = desc + "Le perso à été tué.";
        }
    }

    /**
     * Getter
     *
     * @return le dommage reÃğu par l'attaque.
     */
    public int getDmgEffectue() {
        return dommageRecu;
    }

    /**
     * @return l'attaque a-t-elle fait des dégats réels?
     */
    @Override
    public boolean coupAMarcher() {
        return dommageRecu > 0;
    }

    /**
     * Getter
     *
     * @return Description de ce type de coup.
     */
    @Override
    public String getDescription() {
        return desc;
    }
```

```
/**
 * Getter
 *
 * @return l'identifiant du joueur qui a joué de
    coup.
 */
@Override
public int coupJouerPar() {
    return attaqueur;
}

/**
 * Getter
 *
 * @return l'identifiant de la carte qui a attaqué.
 */
public int getAttaqueurPerso() {
    return idCarte;
}

/**
 * Getter
 *
 * @return l'identifiant de la carte qui a reÃğu
    l'attaque ou -1 si le joeur
 * a pris le coup.
 *
 */
public int getPersonneAttaque() {
    return idCarteAttack;
}

/**
 * Getter
 *
 * @return True si le perso est mort par cette
    attaque,false sinon.
 */
public boolean attaqueATuer() {
    return attaqueTuer;
}

/**
 * Setter
 *
 * @param joueurId l'identifiant du joueur qui a
    fait le coup.
```

```
     */
    @Override
    public void setJoueur(int joueurId) {
        attaqueur = joueurId;
    }
}
```

5.3 AttaquePlayerResult.Java

```
package cardgame.ResultUtils;

/**
 * Implémentation de Resultat pour décrire les
    conséquences d'une attaque
 * perso-joueur.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class AttaquePlayerResult implements Resultat {

    private final int dommageRecu;
    private int attaqueur;
    private final int idCarte;
    private final int idAdversaire;
    private final boolean attaqueTuer;
    private  String desc;

    public AttaquePlayerResult(int dmg, int joueurId,
       int carteId, int adversaireId, boolean
       attaqueTue) {
        dommageRecu = dmg;
        attaqueur = joueurId;
        idCarte = carteId;
        attaqueTuer = attaqueTue;
        idAdversaire = adversaireId;
        desc = "L'attaque de la carte " + carteId + "sur
           le joueur" + idAdversaire + "a causé " + dmg
           + "dégats.\n";
        if (attaqueTue)
            desc = desc + "Le joueur à été tué.";
    }
```

```
public AttaquePlayerResult(int dmg, int carteId, int
    adversaireId, boolean attaqueTue) {
     dommageRecu = dmg;
     idCarte = carteId;
     attaqueTuer = attaqueTue;
     idAdversaire = adversaireId;
     desc = "L'attaque de la carte " + carteId + "sur
        le joueur" + idAdversaire + "a causé " + dmg
        + "dégats.\n";
     if (attaqueTue)
          desc = desc + "Le joueur à été tué.";
}

/**
 * Getter
 *
 * @return le dommage reÃğu par l'attaque.
 */
public int getDmgEffectue() {
    return dommageRecu;
}

/**
 * @return l'attaque a-t-elle fait des dégats réels?
 */
@Override
public boolean coupAMarcher() {
    return dommageRecu > 0;
}

/**
 * Getter
 *
 * @return Description de ce type de coup.
 */
@Override
public String getDescription() {
    return desc;
}

/**
 * Getter
 *
 * @return l'identifiant du joueur qui a joué de
     coup.
 */
```

```
    @Override
    public int coupJouerPar() {
        return attaqueur;
    }

    /**
     * Getter
     *
     * @return l'identifiant de la carte qui a attaqué.
     */
    public int getAttaqueurPerso() {
        return idCarte;
    }

    /**
     * Getter
     * @return True si le perso est mort par cette
        attaque,false sinon.
     */
    public boolean attaqueATuer() {
        return attaqueTuer;
    }

    /**
     * Setter
     * @param joueurId l'identifiant du joueur qui a
        fait le coup.
     */
    @Override
    public void setJoueur(int joueurId) {
        attaqueur = joueurId;
    }
}
```

5.4 DefausseResult.Java

```
package cardgame.ResultUtils;

import cardgame.JeuxCartes.Carte;
import java.util.ArrayList;
import java.util.List;
import javax.json.JsonObject;

/**
 * Implémentation de Resultat pour décrire les
    conséquences d'une action de
```

```
 * défaussage.
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class DefausseResult implements Resultat {

    private final List<Integer> cartesId;
    private final List<JsonObject> cartesJSON;
    private int joueurId;
    private final String description;
    private final boolean coupAFonctionne;

    public DefausseResult(int idJoueur, boolean
       coupCorrect, List<Carte> cartes) {
        cartesId = new ArrayList<>();
        cartesJSON = new ArrayList<>();
        String cartesStr = "";
        for (Carte cartePioche : cartes) {
            cartesId.add(cartePioche.getCardID());
            cartesJSON.add(cartePioche.toJSON());
            cartesStr = cartesStr +
               cartePioche.toJSON().toString() + "\n";
        }
        joueurId = idJoueur;
        description = "Les cartes suivantes ont été
           défaussés :" + cartesStr;
        coupAFonctionne = coupCorrect;
    }

    /**
     * @return True si l'action a fonctionné,false sinon.
     */
    @Override
    public boolean coupAMarcher() {
        return coupAFonctionne;
    }

    /**
     * Getter
     *
     * @return Description de ce type de coup.
     */
```

```
@Override
public String getDescription() {
    return description;
}

/**
 * Getter
 *
 * @return l'identifiant du joueur qui a joué de
    coup.
 */
@Override
public int coupJouerPar() {
    return joueurId;
}

/**
 * Getter
 *
 * @return les identifiant des cartes défaussés.
    (Utile pour une vue qui
 * l'utilise en tandem avec le Json de la partie.)
 */
public List<Integer> getCartesID() {
    return cartesId;
}

/**
 * Getter
 *
 * @return La représentation JSON des cartes
    défaussés. (Utile pour une vue
 * qui veut faire des animations.)
 */
public List<JsonObject> getCartesJSON() {
    return cartesJSON;
}

/**
 * Setter
 *
 * @param idJoueur l'identifiant du joueur qui a
    fait le coup.
 */
@Override
public void setJoueur(int idJoueur) {
```

```
        this.joueurId = idJoueur;
    }
}
```

5.5 EnchantResult.Java

```
package cardgame.ResultUtils;

import cardgame.JeuxCartes.Carte;
import cardgame.JeuxCartes.Enchant;
import java.util.List;
import javax.json.JsonObject;

/**
 * Implémentation de Resultat pour décrire les
    conséquences d'une action
 * d'enchantement.
 *
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class EnchantResult implements Resultat {

    private final String description;
    private int joueurId;
    private final boolean coupAFonctionne;
    private final Carte carteEnchant;
    private final List<Enchant> enchant;

    public EnchantResult(boolean coupCorrect, Carte
        carteEnch,List<Enchant> enchants) {
        coupAFonctionne = coupCorrect;
        carteEnchant = carteEnch;
        enchant = enchants;
        String enchStr = "";
        for (Enchant ench : enchants)
            enchStr = enchStr + ench.toJSON().toString();
        description = "La carte " +
            carteEnch.toJSON().toString() + "a recu les
            enchantements suivants :" + enchStr;
    }
```

```
public EnchantResult(int jId, boolean coupCorrect,
    Carte carteEnch,List<Enchant> enchants) {
     joueurId = jId;
     coupAFonctionne = coupCorrect;
     carteEnchant = carteEnch;
     enchant = enchants;
     description = "Enchantement d'une carte";
}

/**
 * Getter
 *
 * @return identifiant de la carte enchanté.
 */
public JsonObject getCarteEnchante() {
    return carteEnchant.toJSON();
}

/**
 * @return True si l'action a fonctionné,false sinon.
 */
@Override
public boolean coupAMarcher() {
    return coupAFonctionne;
}

/**
 * Getter
 *
 * @return Description de ce type de coup.
 */
@Override
public String getDescription() {
    return description;
}

/**
 * Getter
 *
 * @return l'identifiant du joueur qui a joué de
     coup.
 */
@Override
public int coupJouerPar() {
    return joueurId;
}
```

```
    /**
     * Setter
     *
     * @param idJoueur l'identifiant du joueur qui a
        fait le coup.
     */
    @Override
    public void setJoueur(int idJoueur) {
        this.joueurId = idJoueur;
    }
}
```

5.6 ForfaitResult.Java

```
package cardgame.ResultUtils;

/**
 * Implémentation de Resultat pour décrire la
    conséquence d'un forfait.
 *
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class ForfaitResult implements Resultat {

    private final String description;
    private int joueurId;
    private final boolean coupAFonctionne;
    private final int joueurPerdu;

    public ForfaitResult(int jId, boolean coupCorrect,
       int idJoueurPerdu) {
        joueurId = jId;
        coupAFonctionne = coupCorrect;
        joueurPerdu = idJoueurPerdu;
        description = "Le joueur" + idJoueurPerdu +
           "vient de perdre la partie";
    }

    /**
     * @return True si l'action a fonctionné,false sinon.
```

```
     */
    @Override
    public boolean coupAMarcher() {
        return  coupAFonctionne;
    }

    /**
     * Getter
     *
     * @return Description de ce type de coup.
     */
    @Override
    public String getDescription() {
        return description;
    }

    /**
     * Getter
     *
     * @return l'identifiant du joueur qui a joué de
         coup.
     */
    @Override
    public int coupJouerPar() {
        return joueurId;
    }

    /**
     * Getter
     * @return l'identifiant du joueur  qui a gagné.
     */
    public int getJoueurQuiAPerdu() {
        return joueurPerdu;
    }

    /**
     * Setter
     * @param idJoueur l'identifiant du joueur qui a
         fait le coup.
     */
    @Override
    public void setJoueur(int idJoueur) {
        this.joueurId = idJoueur;
    }
}
```

5.7 FinDePartieResult.Java

```java
package cardgame.ResultUtils;

/**
 * Implémentation de Resultat pour décrire la fin d'une
    partie.
 *
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class FinDePartieResult implements Resultat {

    private final String description;
    private int joueurId;
    private final boolean coupAFonctionne;
    private final int joueurGagne;

    public FinDePartieResult(int jId, boolean
       coupCorrect, int idJoueurGagne) {
        joueurId = jId;
        coupAFonctionne = coupCorrect;
        joueurGagne = idJoueurGagne;
        description = "Le joueur" + idJoueurGagne +
           "vient de gagner la partie";
    }

    /**
     * @return True si l'action a fonctionné,false sinon.
     */
    @Override
    public boolean coupAMarcher() {
        return  coupAFonctionne;
    }

    /**
     * Getter
     *
     * @return Description de ce type de coup.
     */
    @Override
    public String getDescription() {
        return description;
    }
```

```
    /**
     * Getter
     *
     * @return l'identifiant du joueur qui a joué de
         coup.
     */
    @Override
    public int coupJouerPar() {
        return joueurId;
    }

    /**
     * Getter
     * @return l'identifiant du joueur  qui a gagné.
     */
    public int getJoueurQuiAGagne() {
        return joueurGagne;
    }

    /**
     * Setter
     * @param idJoueur l'identifiant du joueur qui a
         fait le coup.
     */
    @Override
    public void setJoueur(int idJoueur) {
        this.joueurId = idJoueur;
    }
}
```

5.8 PersoDeploieResult.Java

```
package cardgame.ResultUtils;

import cardgame.JeuxCartes.Carte;

/**
 * Implémentation de Resultat pour décrire la
    conséquence d'un déploiment d'un
 * perso et de son arme sur le jeu.
 *
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
```

```
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class PersoDeploieResult implements Resultat {

    private final String description;
    private int joueurId;
    private final boolean coupAFonctionne;

    public PersoDeploieResult(int jId, boolean
       coupCorrect, Carte perso) {
        joueurId = jId;
        coupAFonctionne = coupCorrect;
        description = "Le joueur " + jId + "vient de
           déployer sur le jeu : " + perso.toJSON();
    }

    /**
     * @return True si l'action a fonctionné,false sinon.
     */
    @Override
    public boolean coupAMarcher() {
        return coupAFonctionne;
    }

    /**
     * Getter
     *
     * @return Description de ce type de coup.
     */
    @Override
    public String getDescription() {
        return description;
    }

    /**
     * Getter
     *
     * @return l'identifiant du joueur qui a joué de
        coup.
     */
    @Override
    public int coupJouerPar() {
        return joueurId;
    }
```

```
    /**
     * Setter
     *
     * @param idJoueur l'identifiant du joueur qui a
        fait le coup.
     */
    @Override
    public void setJoueur(int idJoueur) {
        this.joueurId = idJoueur;
    }
}
```

5.9 PiocheResult.Java

```
package cardgame.ResultUtils;

import cardgame.JeuxCartes.Carte;
import java.util.ArrayList;
import java.util.List;
import javax.json.JsonObject;

/**
 * Implémentation de Resultat pour décrire la
    conséquence d'une pioche.
 *
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class PiocheResult implements Resultat {

    private final String description;
    private int joueurId;
    private final boolean coupAFonctionne;
    private final List<Integer> cartesId;
    private final List<JsonObject> cartesJSON;

    public PiocheResult(int jId, boolean coupCorrect,
       List<Carte> cartes) {
        joueurId = jId;
        coupAFonctionne = coupCorrect;
        cartesId = new ArrayList<>();
        cartesJSON = new ArrayList<>();
```

```
        String cStr = "";
        for (Carte cartePioche : cartes) {
            cartesId.add(cartePioche.getCardID());
            cartesJSON.add(cartePioche.toJSON());
            cStr = cStr +
                cartePioche.toJSON().toString();
        }
        description = "Le joueur " + jId + "vient de
            piocher les cartes suivantes : \n" + cStr;
    }

    /**
     * Getter
     *
     * @return les identifiant des cartes pigés. (Utile
         pour une vue qui
     * l'utilise en tandem avec le Json de la partie.)
     */
    public List<Integer> getCartesID() {
        return cartesId;
    }

    /**
     * Getter
     *
     * @return La représentation JSON des cartes pigés.
         (Utile pour une vue qui
     * veut faire des animations.)
     */
    public List<JsonObject> getCartesJSON() {
        return cartesJSON;
    }

    /**
     * @return True si l'action a fonctionné,false sinon.
     */
    @Override
    public boolean coupAMarcher() {
        return coupAFonctionne;
    }

    /**
     * Getter
     *
     * @return Description de ce type de coup.
     */
```

```
    @Override
    public String getDescription() {
        return description;
    }

    /**
     * Getter
     *
     * @return l'identifiant du joueur qui a joué de
        coup.
     */
    @Override
    public int coupJouerPar() {
        return joueurId;
    }

    /**
     * Setter
     *
     * @param idJoueur l'identifiant du joueur qui a
        fait le coup.
     */
    @Override
    public void setJoueur(int idJoueur) {
        this.joueurId = idJoueur;
    }
}
```

5.10 RefuseResult.Java

```
package cardgame.ResultUtils;

/**
 * Implémentation de Resultat pour décrire la
    conséquence d'un coup refusé
 * puisqu'il était impossible.
 *
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class RefuseResult implements Resultat {
```

```
private final String description;
private int joueurId;

public RefuseResult(int idJoueur, String coupRefuse)
   {
    description = coupRefuse;
    joueurId = idJoueur;
}

public RefuseResult(String coupRefuse) {
    description = coupRefuse;
}

/**
 * @return True si l'action a fonctionné,false sinon.
 */
@Override
public boolean coupAMarcher() {
    return false;
}

/**
 * Getter
 *
 * @return Description de ce type de coup.
 */
@Override
public String getDescription() {
    return description;
}

/**
 * Getter
 *
 * @return l'identifiant du joueur qui a joué de
    coup.
 */
@Override
public int coupJouerPar() {
    return joueurId;
}

/**
 * Setter
 *
 * @param idJoueur l'identifiant du joueur qui a
    fait le coup.
```

```
     */
    @Override
    public void setJoueur(int idJoueur) {
        this.joueurId = idJoueur;
    }
}
```

5.11 SoinsResult.Java

```
package cardgame.ResultUtils;

/**
 * Implémentation de Resultat pour décrire la
    conséquence d'un sortilège de
 * soins.
 *
 *
 * @author Mathieu Gravel GRAM02099206
 * @author Nicolas Reynaud REYN23119308
 * @version 1.0
 *
 * 08-Fév-2016 : 1.0 - Version initiale.
 */
public class SoinsResult implements Resultat {

    private final String description;
    private int joueurId;
    private final boolean coupAFonctionne;
    private final int healerId;
    private final int persoSoigneeId;

    public SoinsResult(int jId, boolean coupCorrect, int
       hId, int cId) {
        description = "Le personnage " + hId + " vient
           de soigner " + cId + ".";
        joueurId = jId;
        coupAFonctionne = coupCorrect;
        healerId = hId;
        persoSoigneeId = cId;
    }

    public SoinsResult(boolean coupCorrect, int hId, int
       cId) {
        description = "Le personnage " + hId + " vient
           de soigner " + cId + ".";
        coupAFonctionne = coupCorrect;
```

```
        healerId = hId;
        persoSoigneeId = cId;
    }

    /**
     * Getter
     * @return L'identifiant du perso qui a fait le sort
        de soins.
     */
    public int getHealerId() {
        return healerId;
    }

    /**
     * Getter
     * @return L'identifiant du perso soignée.
     */
    public int getCarteSoigneeId() {
        return persoSoigneeId;
    }

    /**
     * @return True si l'action a fonctionné,false sinon.
     */
    @Override
    public boolean coupAMarcher() {
        return coupAFonctionne;
    }

    /**
     * Getter
     *
     * @return Description de ce type de coup.
     */
    @Override
    public String getDescription() {
        return description;
    }

    /**
     * Getter
     *
     * @return l'identifiant du joueur qui a joué de
        coup.
     */
    @Override
```

```
    public int coupJouerPar() {
        return joueurId;
    }

    /**
     * Setter
     *
     * @param idJoueur l'identifiant du joueur qui a
        fait le coup.
     */
    @Override
    public void setJoueur(int idJoueur) {
        this.joueurId = idJoueur;
    }

}
```