REMISE TP1

Code TP1

Mathieu Gravel GRAM02099206[??*] and Nicolas Reynaud 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 associé a l'implémentation de notre TP1.

Keywords: TP1; Code Source; API

Table des matières

Code source

1 CarteJeux

1.1 Carte.Java

```
1  package cardgame.JeuxCartes;
2
3  import javax.json.JsonObject;
4
5  /**
6   * Classe abstraite, Carte sert d'interface commun pour tout les
        types de cartes
7   * du jeu. (La raison derrière le choix de classe abstraite et non
        d'interface
8   * réside dans l'identifiant unique. Celle-ci nous permet de lier une
        carte du
9   * modèle aux demandes du controlleurs si nécessaire.)
10  *
11  * @author Mathieu Gravel GRAM02099206
12  * @author Nicolas Reynaud REYN23119308
13  * @version 1.0
14  *
15  * 08-Fév-2016 : 1.0 - Version initiale.
16  */
17 public abstract class Carte {
18
19     //Int statique utilisé poour s'assurer que chaque carte ait un Id
           unique.
20     static int SSID = 0;
21     private final int cardID;
22
23     /**
24      * Description des fonctions représentant le JSon des cartes, non
           définie
25      * ici
26      *
27      * @return null, fonction non définie ici
28      */
29     public abstract JsonObject toJSON();
30
31     /**
32      * Constructeur par défaut. Initialise l'identifiant de la carte.
33      */
34     public Carte() {
35         cardID = SSID;
36         ++Carte.SSID;
37     }
38
39     /**
40      * Permet d'avoir l'id de la carte.
41      *
```

```
42       * @return l'identifiant unique associé à la carte
43       */
44      public int getCardID() {
45          return cardID;
46      }
47  }
```

1.2 Perso.Java

```
1   package cardgame.JeuxCartes;
2
3   import cardgame.Regles.TypeArme;
4   import cardgame.ResultUtils.AttaquePersoResult;
5   import java.util.ArrayList;
6   import java.util.List;
7   import javax.json.*;
8
9   /**
10   * Classe représentant les cartes de type personnages du jeu. La
         carte peut être
11   * extend directement pour créer un perso non-combattant.
12   *
13   *
14   * @author Mathieu Gravel GRAM02099206
15   * @author Nicolas Reynaud REYN23119308
16   * @version 1.0
17   *
18   * 08-Fév-2016 : 1.0 - Version initiale. 12-Fév-2016 : 1.1 -
         Modification du
19   * code pour marcher avec Cible. Guerrier,Pretre et Paladin.
         14-Fév-2016 : 1.2 -
20   * Modification du code pour marcher avec Combattant.
21   */
22  public abstract class Perso extends Carte implements Cible {
23
24      private int hp;
25      private final int maxHp;
26      private int mp;
27      private final int maxMp;
28      private Arme armePerso;
29      private final List<TypeArme> armesUtilisables;
30
31      public Perso(int _hp, int _mp, List<TypeArme> armes) {
32          super();
33          hp = _hp;
34          mp = _mp;
35          maxHp = hp;
36          maxMp = _mp;
37          armePerso = null;
38          armesUtilisables = armes;
39      }
40
41      /**
42       * @return L'arme du personnage
43       */
44      public Arme getArme() {
45          return armePerso;
46      }
47
48      public int getMp() {
49          return mp;
50      }
51
52      public List<TypeArme> getArmesUtilisables() {
53          return armesUtilisables;
54      }
55
56      /**
57       * Utilise un point de magie.
58       */
```

```
59      protected void utiliserMagie() {
60          Math.max(mp--, 0);
61      }
62
63      /**
64       * Permet d'obtenir la liste des cartes qui était associée au
             personnage.
65       * Ceci nous permet de les ajouter au cimetière à la mort du
             perso.
66       *
67       * @return Liste des cartes présente sur le perso
68       */
69      protected List<Carte> libererCartes() {
70          List<Carte> cartesMortes = new ArrayList<>();
71          cartesMortes.addAll(armePerso.listEnchant);
72          cartesMortes.addAll(armePerso.listEnchantStase);
73          cartesMortes.add(armePerso);
74          cartesMortes.add(this);
75          return cartesMortes;
76      }
77
78      /**
79       * Permet de vérifier si le perso peut utiliser une arme et si
             oui, la lui
80       * place.
81       *
82       * @param arme arme à donner au perso
83       * @return true si l'arme est placée, false sinon
84       */
85      protected boolean equiperArme(Arme arme) {
86          boolean armeLibre = false;
87          if (this.armePerso == null && arme.peutUtiliserArme(this)) {
88              this.armePerso = arme;
89              armeLibre = true;
90          }
91          return armeLibre;
92      }
93
94      /**
95       * Permet au personnage de recevoir le soin (Autrement dit,
             réinit ses
96       * points de vie).
97       */
98      protected void recevoirSoins() {
99          this.hp = this.maxHp;
100     }
101
102     /**
103      * Permet d'obtenir le type d'arme utilisée par le perso
104      *
105      * @return le type de l'arme si une est équipée, null sinon
106      */
107     public TypeArme getTypeArme() {
108         return armePerso != null ? armePerso.type : null;
109     }
110
111     /**
112      * Permet de savoir si le personnage est mort.
113      *
114      * @return true si le personnage est mort, false sinon
115      */
116     @Override
117     public boolean estMort() {
118         return this.hp <= 0;
119     }
120
121     /**
122      * Permet d'obtenir le Json associé au personnage.
123      *
124      * @return le JSon représentant le perso
125      */
126     @Override
```

```
127     public JsonObject toJSON() {
128         JsonObjectBuilder obj = Json.createObjectBuilder();
129         obj.add("Id", this.getCardID());
130         //obj.add("Type Personnage", typeperso.toString());
131         obj.add("hp", hp);
132         obj.add("mp", mp);
133         if (armePerso != null) {
134             obj.add("Arme personnage", armePerso.toJSON());
135         }
136 
137         return obj.build();
138     }
139 
140     @Override
141     public boolean peutEtreAttaque() {
142         return !estMort();
143     }
144 
145     @Override
146     public AttaquePersoResult recoitAttaque(Combattant attaqueur) {
147         AttaquePersoResult res;
148         assert (this.armePerso != null);
149         int degat =
                attaqueur.forceAttaque(this.armePerso.getTypeArme());
150         this.hp -= degat;
151         res = new AttaquePersoResult(degat, this.getCardID(),
                attaqueur.getCardID(), estMort());
152         return res;
153     }
154 
155 }
```

1.3 Carte.Java

```
1  package cardgame.JeuxCartes;
2  
3  import cardgame.Regles.TypeArme;
4  import java.util.ArrayList;
5  import java.util.Iterator;
6  import java.util.List;
7  import javax.json.*;
8  
9  /**
10  * Classe représentant chacunes des cartes d'armes du jeu. Initialisé
        par
11  * ArmeFactory (Afin d'attribuer les bons attributs pour chaque type
        d'armes),
12  * Arme permet de traiter la logique de : - Force d'attaque -
        ÃĽquipage
13  * d'enchantements - Vérification de l'arme pour un déploiement.
14  *
15  * @author Mathieu Gravel GRAM02099206
16  * @author Nicolas Reynaud REYN23119308
17  * @version 1.2
18  *
19  * Historique : 08-Fév-2016 : 1.0 - Version initiale.
20  * 11-Fév-2016 : 1.1 - Découpage de ListUtilisateurs pour le placer
21  *                     dans Perso.
22  *               1.2 - Ajout de fonctions pour vérifier si l'amr
23  *                     est déployé.
24  */
25 public class Arme extends Carte {
26 
27     protected TypeArme type;
28     /**
29      * Boolean notant explicitement si l'arme est stased pour fins
30      * d'éfficiences.
31      */
32     private boolean estStase;
33     private boolean armeUtilise;
```

```
34      protected int degat;
35      protected List<Enchant> listEnchant;
36      protected boolean estFacile;
37
38      /**
39       * Liste utilisé pour noter les enchantements qui ont été Stased.
             Cette
40       * liste sert seulement pour fins d'affichages.
41       */
42      protected List<Enchant> listEnchantStase;
43
44      /*Variables notant les valeurs initiales de l'arme.
45       Ceci nous permet de remettre l'arme à zéro si nécessaire.*/
46      private final int degatOrg;
47      private final TypeArme typeOrg;
48
49      public Arme(TypeArme _type, int dmg) {
50          super();
51          type = _type;
52          typeOrg = _type;
53          degat = dmg;
54          degatOrg = dmg;
55          armeUtilise = false;
56          estFacile = false;
57          listEnchant = new ArrayList<>();
58          listEnchantStase = new ArrayList<>();
59          estStase = false;
60      }
61
62      /**
63       * Permet de savoir la force d'attaque de l'arme en appliquant le
             triangle
64       * des degats
65       *
66       * @param arme Type d'arme sur laquel faire le triangle de
             modificateur de
67       * dégats
68       * @return la force d'attaque de l'arme
69       */
70      public int forceAttaque(TypeArme arme) {
71          return this.degat + this.type.calculModificateur(arme);
72      }
73
74      /**
75       * Permet de savoir si un perso peut utiliser ou non une arme.
76       *
77       * @param p personnage à verifier
78       * @return true si le perso peut porter l'arme false sinon
79       */
80      public boolean peutUtiliserArme(Perso p) {
81          return (estFacile ||
                p.getArmesUtilisables().contains(this.type));
82      }
83
84      /**
85       * Permet d'ajouter un enchantement à l'arme courante
86       *
87       * @param ench Enchant à appliquer à l'arme
88       */
89      protected void ajouterEnchant(Enchant ench) {
90          if (!this.estStase) {
91              listEnchant.add(ench);
92              ench.placerEnchant(this);
93          }
94      }
95
96      /**
97       * Permet de réinitaliser l'arme à son état d'origine.
98       */
99      protected void reset() {
100         this.listEnchantStase = new ArrayList<>(this.listEnchant);
101         this.listEnchant = new ArrayList<>();
```

```
102             this.degat = this.degatOrg;
103             this.type = this.typeOrg;
104         }
105 
106         /**
107          * Permet d'avoir la représentation JSon d'une arme.
108          *
109          * @return le jSon associé à une arme
110          */
111         @Override
112         public JsonObject toJSON() {
113             JsonObjectBuilder obj = Json.createObjectBuilder();
114             obj.add("Id", this.getCardID());
115             obj.add("Type d'arme", type.name());
116             obj.add("Degats", degat);
117             Iterator<Enchant> it = listEnchant.iterator();
118             int enchNum = 1;
119             while (it.hasNext()) {
120                 obj.add("Enchantement actif #" + enchNum,
                        it.next().toJSON());
121                 ++enchNum;
122             }
123             enchNum = 1;
124             it = listEnchantStase.iterator();
125             while (it.hasNext()) {
126                 obj.add("Enchantement inactif #" + enchNum,
                        it.next().toJSON());
127                 ++enchNum;
128             }
129 
130             return obj.build();
131         }
132 
133         /**
134          * Getter
135          * @return Type d'arme
136          */
137         public TypeArme getTypeArme(){
138             return type;
139         }
140 
141         /**
142          * Permet de staser une Arme
143          */
144         protected void staserArme() {
145             this.estStase = true;
146         }
147 
148         /**
149          * Setter notant que l'arme a été déployé.
150          */
151         protected void deployerArme() {
152             armeUtilise = true;
153         }
154 
155         /**
156          * Permet de savoir si une arme est Stase
157          *
158          * @return true si l'arme est stase false sinon
159          */
160         public boolean peutAjouterEnchantement() {
161             return !estStase;
162         }
163 
164         /**
165          * Getter
166          * @return Bool dictant si l'arme est déployé sur un perso.
167          */
168         public boolean armeEstDeploye() {
169             return armeUtilise;
170         }
171     }
```

1.4 Enchant.Java

```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

```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.
```

```
6   *
7   * EnchantFacile permet de changer les utilisateurs possibles de
        l'arme.
8   *
9   * @author Mathieu Gravel GRAM02099206
10  * @author Nicolas Reynaud REYN23119308
11  * @version 1.0
12  *
13  * 08-Fév-2016 : 1.0 - Version initiale.
14  */
15 public class EnchantFacile extends Enchant {
16
17      public EnchantFacile() {
18          super("Cette carte rend cette arme utilisable par tout le
                monde.");
19      }
20
21      /**
22       * Applique l'enchantement sur l'arme passé en paramètre.
23       *
24       * @param arme arme qui pourra être équipée par tout le monde
25       */
26      @Override
27      protected void placerEnchant(Arme arme) {
28          if (!arme.armeEstDeploye()) {
29              arme.estFacile = true;
30          }
31      }
32  }
```

1.8 EnchantDegatPlus.Java

```
1   package cardgame.JeuxCartes;
2
3   /**
4    * Implémentation de la classe abstraite Enchant.
5    *
6    * EnchantDegatPlus permet d'augmenter la force d'une arme.
7    *
8    * @author Mathieu Gravel GRAM02099206
9    * @author Nicolas Reynaud REYN23119308
10   * @version 1.0
11   *
12   * 08-Fév-2016 : 1.0 - Version initiale.
13   */
14  public class EnchantDegatPlus extends Enchant {
15
16      public EnchantDegatPlus() {
17          super("Cette carte augmente les degats de l'arme choisi par
                un.");
18      }
19
20      /**
21       * Applique l'enchantement sur l'arme passé en paramètre.
22       *
23       * @param arme arme dont les degats vont etre augmenté
24       */
25      @Override
26      protected void placerEnchant(Arme arme) {
27          arme.degat++;
28      }
29  }
```

1.9 EnchantDegatMoins.Java

```
1   package cardgame.JeuxCartes;
2
3   /**
4    * Implémentation de la classe abstraite Enchant.
```

```
5   *
6   * EnchantDegatMoins permet d'abaisser la force d'une arme.
7   *
8   * @author Mathieu Gravel GRAM02099206
9   * @author Nicolas Reynaud REYN23119308
10  * @version 1.0
11  *
12  * 08-Fév-2016 : 1.0 - Version initiale.
13  */
14 public class EnchantDegatMoins extends Enchant {
15
16      public EnchantDegatMoins() {
17          super("Cette carte abaisse les dommages de l'arme choisi par
                1.");
18      }
19
20      /**
21       * Applique l'enchantement sur l'arme passé en paramètre.
22       *
23       * @param arme arme dont les degats vont etre diminué
24       */
25      @Override
26      protected void placerEnchant(Arme arme) {
27          arme.degat--;
28      }
29 }
```

1.10 Cible.Java

```
1  package cardgame.JeuxCartes;
2
3  import cardgame.ResultUtils.Resultat;
4
5  /**
6   * Interface utilisé pour les fonctions reliés au recoit de coups.
7   * Cette interface nous permet de généraliser les appels d'attaques
        au xperso et Joueurs.
8   * (L'idée provient de l'équipe Philippe Pépos PetitClerc et Mehdi
        Ait Younes)
9   * @author Mathieu Gravel GRAM02099206
10  * @author Nicolas Reynaud REYN23119308
11  * @version 1.0
12  * 12-Fév-2016 : 1.0 - Version initiale.
13  */
14 public interface Cible {
15
16      /**
17       * @return True si la cible peut actuellement être attaqué.
18       * (Ex un Joueur doit avoir un jeu vide.)
19       */
20      public abstract boolean peutEtreAttaque();
21      /**
22       * recoitAttaque applique le dommage reÃğu, vérifie si le coup é
            tait
23       * fatal et retourne le résultat du coup.
24       * @param attaqueur Le combattant qui attaque la cible.
25       * @return Le résultat du coup reÃğu (Soit AttackPerso ou
            AttackJoueur)
26       */
27      public abstract Resultat recoitAttaque(Combattant attaqueur);
28
29      /**
30       * @return True si la cible est morte.
31       */
32      public abstract boolean estMort();
33
34 }
```

1.11 Soigneur.Java

```java
package cardgame.JeuxCartes;

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 éffectué.
     */
    public abstract boolean peutSoigner(Perso p);

}
```

1.12 Combattant.Java

```java
package cardgame.JeuxCartes;

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);
     }
```

```
27
28      /**
29       * Fonction qui calcule le nombre de dégats fait à l'opposant
             armé.
30       * @param ta Type d'arme de l'opposant.
31       * @return Le nombre de dégats.
32       */
33      public int forceAttaque(TypeArme ta) {
34          return this.getArme().forceAttaque(ta);
35      }
36
37      /**
38       * Cette fonction effectue l'attaque d'une cible.
39       * @param c Instance de la cible attaquée.
40       * @return Soit unAttaquePersoResult
41       * ou AttaquePlayerResult décrivant le coup.
42       */
43      public Resultat Attaque(Cible c) {
44          return c.recoitAttaque(this);
45      }
46
47  }
```

1.13 Guerrier.Java

```
1   package cardgame.JeuxCartes;
2
3   import cardgame.Regles.*;
4   import java.util.Arrays;
5   import javax.json.*;
6
7   /**
8    * Classe représentant la classe Guerrier du jeu. La classe est une
         extension de
9    * Combattant, ce qui lui permet d'attaquer.
10   *
11   * @author Mathieu Gravel GRAM02099206
12   * @author Nicolas Reynaud REYN23119308
13   * @version 1.0
14   * 12-Fév-2016 : 1.0 - Version initiale.
15   */
16  public class Guerrier extends Combattant {
17
18      public Guerrier() {
19          super(Regle.GUERRIERHP, Regle.GUERRIERMP,
                Arrays.asList(TypeArme.values()));
20      }
21
22      @Override
23      public JsonObject toJSON() {
24          JsonObject json = super.toJSON();
25          JsonObjectBuilder addition = Json.createObjectBuilder();
26          addition.add("Type Personnage", "Guerrier");
27          addition.add("General Info", json);
28          return addition.build();
29      }
30  }
```

1.14 Pretre.Java

```
1   package cardgame.JeuxCartes;
2
3   import cardgame.Regles.*;
4   import cardgame.ResultUtils.SoinsResult;
5   import java.util.Arrays;
6   import javax.json.*;
7
8   /**
```

```
 9   * Classe représentant la classe Pretre du jeu. La classe est une
         extension de
10   * Combattant et implémente soigneur, ce qui lui permet d'attaquer et
         soigner.
11   *
12   * @author Mathieu Gravel GRAM02099206
13   * @author Nicolas Reynaud REYN23119308
14   * @version 1.0 12-Fév-2016 : 1.0 - Version initiale.
15   */
16  public class Pretre extends Combattant implements Soigneur {
17
18      public Pretre() {
19          super(Regle.PRETREHP, Regle.PRETREMP,
                Arrays.asList(TypeArme.Contondant, TypeArme.Neutre));
20      }
21
22      @Override
23      public SoinsResult soigner(Perso allie) {
24          SoinsResult resultat;
25          this.utiliserMagie();
26          allie.recevoirSoins();
27          resultat = new SoinsResult(true, this.getCardID(),
                allie.getCardID());
28          return resultat;
29      }
30
31      @Override
32      public boolean peutSoigner(Perso p) {
33          return this.getMp() > 0 && (this.getCardID() !=
                p.getCardID());
34      }
35
36      @Override
37      public JsonObject toJSON() {
38          JsonObject json = super.toJSON();
39          JsonObjectBuilder addition = Json.createObjectBuilder();
40          addition.add("Type Personnage", "Pretre");
41          addition.add("General Info", json);
42
43          return addition.build();
44      }
45  }
```

1.15 Paladin.Java

```
 1  package cardgame.JeuxCartes;
 2
 3  import cardgame.Regles.*;
 4  import cardgame.ResultUtils.SoinsResult;
 5  import java.util.Arrays;
 6  import javax.json.*;
 7
 8  /**
 9   * Classe représentant la classe Paladin du jeu. La classe est une
         extension de
10   * Combattant et implémente soigneur, ce qui lui permet d'attaquer et
         soigner.
11   *
12   * @author Mathieu Gravel GRAM02099206
13   * @author Nicolas Reynaud REYN23119308
14   * @version 1.0
15   * 12-Fév-2016 : 1.0 - Version initiale.
16   */
17  public class Paladin extends Combattant implements Soigneur {
18
19      public Paladin() {
20          super(Regle.PALADINHP, Regle.PALADINMP,
                Arrays.asList(TypeArme.values()));
21      }
22
```

```
23      @Override
24      public SoinsResult soigner(Perso allie) {
25          SoinsResult resultat;
26          this.utiliserMagie();
27          allie.recevoirSoins();
28          resultat = new SoinsResult(true, this.getCardID(),
                allie.getCardID());
29          return resultat;
30      }
31
32      @Override
33      public boolean peutSoigner(Perso p) {
34          return this.getMp() > 0 && (this.getCardID() !=
                p.getCardID());
35      }
36
37      @Override
38      public JsonObject toJSON() {
39          JsonObject json = super.toJSON();
40          JsonObjectBuilder addition = Json.createObjectBuilder();
41          addition.add("Type Personnage", "Paladin");
42          addition.add("General Info", json);
43          return addition.build();
44      }
45
46  }
```

1.16 Deck.Java

```
1   package cardgame.JeuxCartes;
2
3   import cardgame.Regles.Regle;
4   import cardgame.Init.*;
5   import java.util.ArrayList;
6   import java.util.Collections;
7   import java.util.List;
8   import java.util.Random;
9   import javax.json.*;
10
11  /**
12   * Classe représentant le paquet de cartes non-pigés d'un joueur. La
         classe
13   * permet d'initialiser le deck et de traiter la logique de pioche et
         de vie
14   * (puisque les points de vie sont == au nombre de cartes restantes.)
         sans
15   * donner accès à cette logique au joueur.
16   *
17   * @author Mathieu Gravel GRAM02099206
18   * @author Nicolas Reynaud REYN23119308
19   * @version 1.0 08-Fév-2016 : 1.0 - Version initiale.
20   *          1.1 10-Fév-2016 : 1.1 - Modification de InitialiserDeck
21   *                                  pour utiliser PersoFactory.
22   */
23  public class Deck {
24
25      /**
26       * Structure représentant le deck lui-même.
27       */
28      private final List<Carte> cartespioches;
29
30      public Deck(){
31          cartespioches = new ArrayList<>();
32          initialiserDeck();
33      }
34
35      /**
36       * Permet de créer le deck et d'initialiser son contenu.
37       * La classe initialise le nombre de cartes nécessaire de chaque
             type
```

```
38       * selon les règles du jeu et ensuite mélange le deck.
39       */
40      private void initialiserDeck() {
41          ArmeFactory createurArmes = new ArmeFactory();
42          EnchantFactory createurEnchants = new EnchantFactory();
43          PersoFactory createurPersos = new PersoFactory();
44
45          cartespioches.addAll(createurPersos.creerSetGuerrier(Regle.CARTEGUERRIER));
46          cartespioches.addAll(createurPersos.creerSetPretre(Regle.CARTEPRETRE));
47          cartespioches.addAll(createurPersos.creerSetPaladin(Regle.CARTEPALADIN));
48          cartespioches.addAll(createurArmes.creerSetArmes(Regle.CARTEARMEUN,
                1));
49          cartespioches.addAll(createurArmes.creerSetArmes(Regle.CARTEARMEDEUX,
                2));
50          cartespioches.addAll(createurEnchants.creerSetEnchants(Regle.CARTEENCHANTEMENT));
51
52          Collections.shuffle(cartespioches, new
                Random(System.nanoTime()));
53      }
54
55      /**
56       * Fonction qui permet de vider le deck.
57       */
58      public void viderDeck(){
59          this.cartespioches.clear();
60      }
61
62      /**
63       * Permet de piocher une liste de carte.
64       * @param nbCartes nombre de carte à piocher
65       * @return Liste des cartes piochées
66       */
67      public List<Carte> piocherCarte(int nbCartes) {
68
69          /*On s'assure de piocher le min entre le nombre de cartes
                restantes,
70          le nombre demandé ou le nombre maximal dans une main.*/
71          int nbAPiocher = Math.min(nbCartes, this.carteRestantes());
72          nbAPiocher = Math.min(nbAPiocher, Regle.CARTEMAIN);
73
74          List<Carte> nouvCartes = new ArrayList<>();
75
76          while ( nbAPiocher != 0) {
77              nouvCartes.add(cartespioches.remove(0));
78              --nbAPiocher;
79          }
80
81          return nouvCartes;
82      }
83
84      /**
85       * Permet d'appliquer les dégats recu par un joueur sur son Deck.
86       * @param nbDegatCarte Le nombre de dégat pris
87       * @return la Liste des cartes perdues
88       */
89      public List<Carte> dommageJoueur(int nbDegatCarte) {
90          return piocherCarte(nbDegatCarte);
91      }
92
93      /**
94       * Permet de savoir le nombre de carte réstante dans la pioche.
95       * @return le nombre de cartes encore présentes dans la pioche.
96       */
97      public int carteRestantes() {
98          return cartespioches.size();
99      }
100
101     /**
102      * @return True si le deck est vide.
103      */
104     public boolean deckEstVide(){
105         return cartespioches.isEmpty();
```

```
106      }
107
108      /**
109       * Pemet d'avoir la représentation en JSon du Deck
110       * A noter : Le contenu du deck n'est pas communiqué pour éviter
              la triche.
111       * @return la représentation du Deck en JSon
112       */
113      public JsonObject toJSon() {
114          JsonObjectBuilder obj = Json.createObjectBuilder();
115          obj.add("Nombre de cartes restantes a piger",
                 cartespioches.size());
116          return obj.build();
117      }
118  }
```

1.17 Joueur.Java

```
1   package cardgame.JeuxCartes;
2
3   import cardgame.ResultUtils.Resultat;
4   import cardgame.Regles.*;
5   import cardgame.ResultUtils.*;
6   import java.util.Iterator;
7   import java.util.List;
8   import java.util.Map;
9   import java.util.concurrent.ConcurrentHashMap;
10  import java.util.concurrent.CopyOnWriteArrayList;
11  import javax.json.*;
12
13  /**
14   * Classe utilisé par l'API pour placé le coup du joueur reÃğu du
         controlleur.
15   * Comparé à Jeux qui sert de facade pour le controlleur, Joueur
         traite les
16   * appels des joueurs dans le modèle et retourne ces conséquences.
17   *
18   * @author Mathieu Gravel GRAM02099206
19   * @author Nicolas Reynaud REYN23119308
20   * @version 1.1
21   * Historique : 08-Fév-2016 : 1.0 - Version initiale.
22   *              13-Fév-2016 : 1.1 - Réécriture des fonctions pour
23   *                                  fonctionner avec cible.
24   */
25  public class Joueur implements Cible {
26
27      private final int idJoueur;
28      private final Deck carteDeck;
29      private final Map<Integer, Carte> main;
30      private final Map<Integer, Perso> carteEnJeu;
31      private final List<Carte> cimetiere;
32
33      public Joueur(int i) {
34          idJoueur = i;
35          carteDeck = new Deck();
36          main = new ConcurrentHashMap<>();
37          cimetiere = new CopyOnWriteArrayList<>();
38          carteEnJeu = new ConcurrentHashMap<>();
39          List<Carte> mainDeb = carteDeck.piocherCarte(Regle.CARTEMAIN);
40          for (Carte c : mainDeb) {
41              main.put(c.getCardID(), c);
42          }
43      }
44
45      /**
46       * @return L'identifiant unique du joueur.
47       */
48      public int getIdjoueur() {
49          return idJoueur;
50      }
```

```
51
52      /**
53       * Permet d'obtenir le deck du joueur
54       *
55       * @return le deck du joueur
56       */
57      public Deck getCarteDeck() {
58          return carteDeck;
59      }
60
61      /**
62       * Permet d'avoir la main du joueur
63       *
64       * @return la liste de Carte contenu dans la main du joueur
65       */
66      public Map<Integer, Carte> getMain() {
67          return main;
68      }
69
70      /**
71       * Pemet d'avoir les cartes en jeu du joueur
72       *
73       * @return la liste de carte présente sur le jeu, cartes
             associées au joueur
74       */
75      public Map<Integer, Perso> getCarteEnJeu() {
76          return carteEnJeu;
77      }
78
79      /**
80       * Permet d'avoir le cimetiere du joueur
81       *
82       * @return la liste de carte présente dans le cimetiere du joueur
83       */
84      public List<Carte> getCimetiere() {
85          return cimetiere;
86      }
87
88      /**
89       * Permet de savoir si le joueur à perdu
90       *
91       * @return true si le joueur à perdu (autrement dit si il n'a
             plus de carte
92       * null part ) [Cimetiere non compris] false sinon
93       */
94      public boolean aPerdu() {
95          return (main.isEmpty() && carteEnJeu.isEmpty() &&
                 carteDeck.deckEstVide());
96      }
97
98      /**
99       * Permet au joueur de defausser une liste de Carte
100      *
101      * @param defausse liste des cartes à défausser
102      * @return Un DefausseResult si la defausse s'est bien passé Un
103      * RefusedResult sinon
104      */
105     public Resultat defausserCartes(List<Carte> defausse) {
106         Resultat res;
107
108         for (Carte c : defausse) {
109             cimetiere.add(main.remove(c.getCardID()));
110         }
111
112         res = new DefausseResult(this.getIdjoueur(), true, defausse);
113         return res;
114     }
115
116     /**
117      * Permet au joueur de piocher des cartes
118      *
```

```
119         * @return un PiocheResult si tout s'est bien passé un
                RefusedResult sinon
120         */
121        public Resultat piocher() {
122            int nbAPiocher = Regle.CARTEMAIN - main.size();
123            nbAPiocher = Math.max(nbAPiocher, 0);
124            List<Carte> lc = carteDeck.piocherCarte(nbAPiocher);
125
126            for (Carte c : lc) {
127                main.put(c.getCardID(), c);
128            }
129
130            return new PiocheResult(this.getIdjoueur(), true, lc);
131        }
132
133        /**
134         * Permet à un joueur d'attaquer un joueur à l'aide d'une de ses
                cartes
135         *
136         * @param attaqueur position de la carte attaquant sur le jeu
137         * @param attaque Joueur à attaquer
138         * @return un AttackResult si tout c'est bien passé un
                RefusedResult sinon
139         */
140        public Resultat attaque(Combattant attaqueur, Cible attaque) {
141            Resultat res;
142
143            res = attaqueur.Attaque(attaque);
144
145            return res;
146        }
147
148        /**
149         * Fonction auxiliaire appelé après chaque attaque reÃğu,
                MAJCartesPlancher
150         * enlève les perso mort de la partie.
151         */
152        public void MAJCartesPlancher() {
153            for (Perso pers : this.carteEnJeu.values()) {
154                if (pers.estMort()) {
155                    cimetiere.add(carteEnJeu.remove(pers.getCardID()));
156                }
157            }
158        }
159
160        /**
161         * Permet d'ajouter une liste d'enchant à un joueur
162         *
163         * @param enchs liste des positions des enchants dans la main
164         * @param carteTouchee carte étant affectée par l'enchant
165         * @return une Liste de Result, chaqu'un étant soit un
                EnchantResult si tout
166         * c'est bien passé sinon un RefusedResult
167         */
168        public Resultat ajouterEnchants(List<Enchant> enchs, Carte
            carteTouchee) {
169            Resultat res;
170            Arme arm;
171            if (carteTouchee instanceof Perso) {
172                arm = ((Perso) carteTouchee).getArme();
173            } else if (carteTouchee instanceof Arme) {
174                arm = (Arme) carteTouchee;
175            } else {
176                return new RefuseResult("Erreur interne.");
177            }
178
179            for (Enchant ench : enchs) {
180                arm.ajouterEnchant(ench);
181                cimetiere.add(ench);
182                main.remove(ench.getCardID());
183            }
184            res = new EnchantResult(true, carteTouchee, enchs);
```

```
185         return res;
186     }
187 
188     /**
189      * Permet au joueur de placer un personnage en jeu
190      *
191      * @param personnage position dans la main du personnage à jouer
192      * @param arm arme à équiper au perso
193      * @param ench Liste des enchants à ajouter à l'arme
194      * @return un PersoDeploieResult si tout c'est bien passé un
           Refusedresult
195      * sinon
196      */
197     public Resultat placerPerso(Perso personnage, Arme arm,
         List<Carte> ench) {
198         Resultat res;
199 
200         for (Carte c : ench) {
201             Enchant en = (Enchant) c;
202             arm.ajouterEnchant(en);
203             cimetiere.add(main.remove(en.getCardID()));
204         }
205 
206         personnage.equiperArme(arm);
207         main.remove(personnage.getCardID());
208         main.remove(arm.getCardID());
209         carteEnJeu.put(personnage.getCardID(), personnage);
210 
211         res = new PersoDeploieResult(this.getIdjoueur(), true,
              personnage);
212         return res;
213     }
214 
215     /**
216      * @param car La carte qu'on veut vérifier l'emplacement
217      * @return True si la carte est dans la main du joueur.
218      */
219     public boolean carteDansMain(Carte car) {
220         return main.containsKey(car.getCardID());
221     }
222 
223     /**
224      * @param car La carte qu'on veut vérifier l'emplacement
225      * @return True si la carte est dans la jeu du joueur.
226      */
227     public boolean carteDansJeu(Carte car) {
228         return carteEnJeu.containsKey(car.getCardID());
229     }
230 
231     /**
232      * @return True si le joueur a un deck vide
233      */
234     public boolean destVide() {
235         return carteDeck.deckEstVide();
236     }
237 
238     /**
239      * Permet au joueur de declarerForfait Autrement dit, de passer
           toutes ces
240      * cartes dans le cimetiere.
241      *
242      * @return RefusedResult si le joueur à déjà perdu
           FinDePartieResult si le
243      * joueur à déclarer forfait
244      */
245     public Resultat declarerForfait() {
246         Resultat res;
247         main.clear();
248         carteDeck.viderDeck();
249         carteEnJeu.clear();
250         res = new FinDePartieResult(this.getIdjoueur(), true, -1);
251         return res;
```

```
252     }
253
254     /**
255      * Permet d'effectué l'action de soin sur un personnage présent
             sur le jeu.
256      *
257      * @param soins Carte effectuant le soin
258      * @param soignee Position dans la liste des carteEnJeu du soignee
259      * @return RefusedResult si le joueur ne peux pas soigner le
             personnage
260      * soignee SoinsResult si le joueur peu être soigné
261      */
262     public Resultat soignerPerso(Soigneur soins, Perso soignee) {
263
264         if (!soins.peutSoigner(soignee)) {
265             return new RefuseResult("Le soins ne peut pas être é
                 ffectué.");
266         }
267
268         return soins.soigner(soignee);
269     }
270
271     /**
272      * Permet d'avoir le JSon associé à un joueur
273      *
274      * @return le JSon objet représentant le joueur
275      */
276     public JsonObject toJSON() {
277         JsonObjectBuilder obj = Json.createObjectBuilder();
278         obj.add("main", this.mainToJSon());
279         obj.add("cimetiere", this.cimetiereToJSon());
280         obj.add("deck", this.deckToJSon());
281
282         return obj.build();
283     }
284
285     /**
286      * Permet d'avoir le JSon associé au contenu de la main du joueur
287      *
288      * @return le JSon associé à la main
289      */
290     private JsonObject mainToJSon() {
291         JsonObjectBuilder obj = Json.createObjectBuilder();
292
293         Iterator<Carte> cd = main.values().iterator();
294         int numCarte = 1;
295         while (cd.hasNext()) {
296             obj.add("carte #" + numCarte, cd.next().toJSON());
297             ++numCarte;
298         }
299
300         return obj.build();
301     }
302
303     /**
304      * Permet d'avoir le JSon associé au contenu du cimetiere du
             joueur
305      *
306      * @return le JSon associé au cimetiere du joueur
307      */
308     private JsonObject cimetiereToJSon() {
309         JsonObjectBuilder obj = Json.createObjectBuilder();
310
311         Iterator<Carte> cd = cimetiere.iterator();
312         int numCarte = 1;
313         while (cd.hasNext()) {
314             obj.add("carte #" + numCarte, cd.next().toJSON());
315             ++numCarte;
316         }
317
318         return obj.build();
319     }
```

```
320
321     /**
322      * Permet d'avoir le contenu du Deck en format jSon
323      *
324      * @return le JSon associé au contenu du Deck du joueur
325      */
326     private JsonObject deckToJSon() {
327         return carteDeck.toJSon();
328     }
329
330     /**
331      * @return True si le joueur a le droit de piocher.
332      */
333     public boolean peutPiocher() {
334         return ((main.size() < Regle.CARTEMAIN) &&
                (!carteDeck.deckEstVide()));
335     }
336
337     @Override
338     public boolean peutEtreAttaque() {
339         return carteEnJeu.isEmpty();
340     }
341
342     @Override
343     public AttaquePlayerResult recoitAttaque(Combattant attaqueur) {
344         int degat = attaqueur.forceAttaque(TypeArme.Neutre);
345         List<Carte> cartePerdus = this.carteDeck.dommageJoueur(degat);
346         for (Carte c : cartePerdus) {
347             cimetiere.add(c);
348         }
349         AttaquePlayerResult res = new AttaquePlayerResult(degat,
                idJoueur, attaqueur.getCardID(), idJoueur, estMort());
350         return res;
351     }
352
353     @Override
354     public boolean estMort() {
355         return ((carteDeck.deckEstVide() && main.isEmpty()) &&
                (carteEnJeu.isEmpty()));
356     }
357
358 }
```

2 Init

2.1 ArmeFactory.Java

```
1  package cardgame.Init;
2
3  import cardgame.JeuxCartes.Arme;
4  import cardgame.Regles.TypeArme;
5  import java.util.ArrayList;
6  import java.util.List;
7
8  /**
9   * Classe utilisé pour instancier correctement les cartes d'armes
        dans une
10  * partie. ArmeFactory, classe basé sur le patron Factory, nous
        permet de
11  * découpler et cacher la logique de création des cartes des classes
        reliés à
12  * l'API.
13  *
14  * @author Mathieu Gravel GRAM02099206
15  * @author Nicolas Reynaud REYN23119308
16  * @version 1.0
17  *
18  * 08-Fév-2016 : 1.0 - Version initiale.
19  */
20 public class ArmeFactory {
21
```

```
22      /**
23       * Permet de créer une liste d'arme
24       *
25       * @param nbCopies Nombre d ecopies de chacunes des types d'armes
             à
26       * instancier.
27       * @param degats Nombre de dégats fait par ces armes.
28       * @return Liste de nbCopie élements contenant les armes
             demandées.
29       */
30      public List<Arme> creerSetArmes(int nbCopies, int degats) {
31          List<Arme> armes = new ArrayList<>();
32
33          for (int copieAct = 0; copieAct < nbCopies; copieAct++) {
34              armes.add(new Arme(TypeArme.Contondant, degats));
35              armes.add(new Arme(TypeArme.Perforant, degats));
36              armes.add(new Arme(TypeArme.Tranchant, degats));
37          }
38
39          return armes;
40      }
41  }
```

2.2 EnchantFactory.Java

```
1   package cardgame.Init;
2
3   import cardgame.JeuxCartes.*;
4   import java.util.ArrayList;
5   import java.util.List;
6
7   /**
8    * Classe utilisé pour instancier correctement les cartes
         d'enchantements dans
9    * une partie. EnchantFactory, classe basé sur le patron Factory,
         nous permet de
10   * découpler et cacher la logique de création des cartes de l'API.
11   *
12   * @author Mathieu Gravel GRAM02099206
13   * @author Nicolas Reynaud REYN23119308
14   * @version 1.0
15   *
16   * 08-Fév-2016 : 1.0 - Version initiale.
17   */
18  public class EnchantFactory {
19
20      /**
21       * Permet de créer un ensemble de tout les types d'enchantements
             du jeu.
22       *
23       * @param nbCopies nombre de copie des cartes Enchants à
             instancier.
24       * @return Une liste de nbCopie élements qui contient tout les
             enchants.
25       */
26      public List<Enchant> creerSetEnchants(int nbCopies) {
27          List<Enchant> enchantements = new ArrayList<>();
28
29          for (int copieAct = 0; copieAct < nbCopies; copieAct++) {
30              enchantements.add(new EnchantNeutre());
31              enchantements.add(new EnchantStase());
32              enchantements.add(new EnchantDegatPlus());
33              enchantements.add(new EnchantDegatMoins());
34              enchantements.add(new EnchantFacile());
35          }
36
37          return enchantements;
38      }
39  }
```

2.3 PersoFactory.Java

```
1  package cardgame.Init;
2
3  import cardgame.JeuxCartes.Guerrier;
4  import cardgame.JeuxCartes.Paladin;
5  import cardgame.JeuxCartes.Perso;
6  import cardgame.JeuxCartes.Pretre;
7  import java.util.ArrayList;
8  import java.util.List;
9
10 /**
11  * Classe basé sur le patron Factory utilisé pour instancier des
        cartes
12  * personnages. La classe est écrite de telle manière que tout ajout
        de nouveaux
13  * personnages dans le jeu nécessite seulement d'être ajouté comme
        fonction dans
14  * cette classe ainsi que d'inscrire ces règles dans la classe Règle.
15  *
16  * @author Mathieu Gravel GRAM02099206
17  * @author Nicolas Reynaud REYN23119308
18  * @version 1.1
19  *
20  * Historique : 8 Fév-2016 : 1.0 Version initiale.
21  * 13Fév-2016 : 1.1 Modification du code pour marcher avec les
22  * nouvelles classes Guerrier, Pretre et Paladin.
23  */
24 public class PersoFactory {
25
26     /**
27      * Retourne une liste de cartes Guerriers
28      *
29      * @param nbCopies nombre de copie de cartes guerriers
30      * @return liste de nbCopies d'instance de Guerriers.
31      */
32     public List<Perso> creerSetGuerrier(int nbCopies) {
33         List<Perso> guerriers = new ArrayList<>();
34
35         for (int copieAct = 0; copieAct < nbCopies; copieAct++) {
36             guerriers.add(new Guerrier());
37         }
38
39         return guerriers;
40     }
41
42     /**
43      * Retourne une liste de cartes Prêtres.
44      *
45      * @param nbCopies nombre de copie de cartes prêtres.
46      * @return liste de nbCopies d'instance de Prêtres.
47      */
48     public List<Perso> creerSetPretre(int nbCopies) {
49         List<Perso> pretres = new ArrayList<>();
50
51         for (int copieAct = 0; copieAct < nbCopies; copieAct++) {
52             pretres.add(new Pretre());
53         }
54
55         return pretres;
56     }
57
58     /**
59      * Retourne une liste de cartes Paladins.
60      *
61      * @param nbCopies nombre de copie de cartes paladins.
62      * @return liste de nbCopies d'instance de Paladins.
63      */
64     public List<Perso> creerSetPaladin(int nbCopies) {
65         List<Perso> paladins = new ArrayList<>();
66
```

```
67          for (int copieAct = 0; copieAct < nbCopies; copieAct++) {
68              paladins.add(new Paladin());
69          }
70
71          return paladins;
72      }
73  }
```

3 Regles

3.1 Regle.Java

```
1   package cardgame.Regles;
2
3   /**
4    * Classe non instantiable, Regle sert de conteneur pour les valeurs
5    * paramétrisables du jeu.
6    *
7    * @author Mathieu Gravel GRAM02099206
8    * @author Nicolas Reynaud REYN23119308
9    * @version 1.0
10   *
11   * Historique :
12   *
13   * -8 Fév-2016 : 1.0 Version initiale.
14   */
15  public class Regle {
16
17      private Regle() { }
18
19      /**
20       * Liste de toutes les règles de jeu.
21       */
22      public static final int GUERRIERHP = 5;
23      public static final int GUERRIERMP = 0;
24      public static final int PRETREHP = 3;
25      public static final int PRETREMP = 3;
26      public static final int PALADINHP = 4;
27      public static final int PALADINMP = 1;
28      public static final int CARTEGUERRIER = 4;
29      public static final int CARTEPRETRE = 4;
30      public static final int CARTEPALADIN = 2;
31      public static final int CARTEARMEUN = 2;
32      public static final int CARTEARMEDEUX = 2;
33      public static final int CARTEENCHANTEMENT = 2;
34      public static final int CARTEMAIN = 5;
35  }
```

3.2 TypeArme.Java

```
1   package cardgame.Regles;
2
3   /**
4    * Enum, le type de données TypeArme contient chaque type d'armes
       possibles dans
5    * le jeu, ainsi que leurs logiques personnelles, tel que leurs
6    * forces/faiblesses. La classe détient aussi la
7    * fonction de calcul du triangle d'attaque pour fins d'évolutions
       faÃğiles.
8    *
9    * @author Mathieu Gravel GRAM02099206
10   * @author Nicolas Reynaud REYN23119308
11   * @version 1.0
12   *
13   * Historique :
14   *
15   * -8 Fév-2016 : 1.0 Version initiale.
16   */
17  public enum TypeArme {
```

```
18
19      /**
20       * Déclaration de l'enum des types d'armes possibles dans le jeu.
             On décrit
21       * en même temps leur forces,faiblesses et leurs utilisables
             possibles. Les
22       * forces/faiblesses sont décrits en String pour fins de
             visibilité humaine.
23       */
24      Contondant("Contondant", "Perforant", "Tranchant"),
25      Perforant("Perforant", "Tranchant", "Contondant"),
26      Tranchant("Tranchant", "Contondant", "Perforant"),
27      Neutre("Neutre", "", "");
28
29      private final String nom;
30      private final String force;
31      private final String faiblesse;
32
33      TypeArme(String nom, String force, String faiblesse) {
34          this.nom = nom;
35          this.force = force;
36          this.faiblesse = faiblesse;
37      }
38
39      /**
40       * Définit le calcul de modificateur du triangle d'armes.
41       *
42       * @param armeEnnemi Type d'arme de l'ennemi
43       * @return 1 si l'arme actuelle est la faiblesse de l'arme
             ennemi, -1 si
44       * l'arme ennemi est la faiblesse de l'arme actuelle, Sinon 0.
45       */
46      public int calculModificateur(TypeArme armeEnnemi) {
47          if (this.force.equals(armeEnnemi.nom)) {
48              return 1;
49          } else if (this.faiblesse.equals(armeEnnemi.nom)) {
50              return -1;
51          }
52
53          return 0;
54      }
55  }
```

4 API

4.1 Jeux.Java

```
1   package cardgame.API;
2
3   import cardgame.JeuxCartes.Arme;
4   import cardgame.JeuxCartes.Combattant;
5   import cardgame.JeuxCartes.Carte;
6   import cardgame.JeuxCartes.Cible;
7   import cardgame.JeuxCartes.Enchant;
8   import cardgame.JeuxCartes.EnchantFacile;
9   import cardgame.JeuxCartes.Joueur;
10  import cardgame.JeuxCartes.Perso;
11  import cardgame.JeuxCartes.Soigneur;
12  import cardgame.ResultUtils.Resultat;
13  import cardgame.ResultUtils.RefuseResult;
14  import java.util.ArrayList;
15  import java.util.Collection;
16  import java.util.Iterator;
17  import java.util.List;
18  import javax.json.Json;
19  import javax.json.JsonObject;
20  import javax.json.JsonObjectBuilder;
21
22  /**
23   * Classe API utilisé pour communiquer entre le modèle et le
         controlleur. Toutes
```

```
24   * les actions possibles par un joueur doit passer par une des
       fonctions de
25   * cette classe. Afin d'éviter toutes triches, chaque action demandé
       par le
26   * joueur est vérifié avant de s'effectuer. (Ex : Le joueur 2 essaie
       de jouer
27   * pour le joueur . Le joueur 1 essaie de fair eune action impossible
       tel que
28   * déployer des cartes de l'adversaire.) ) Si l'action est mauvaise,
       Jeux
29   * retourne un RefuseResult. Sinon, l'action est éffectué et Jeux
       retourne un
30   * Resultat décrivant les conséquences de l'acte.
31   *
32   * @author Mathieu Gravel GRAM02099206
33   * @author Nicolas Reynaud REYN23119308
34   * @version 1.2
35   * 08-Fév-2016 : 1.0 - Version initiale.
36   * 10-Fév-2016 : 1.1 - Réécriture des fonctions afin de pouvoir
       retourner
37   * les cartes au controlleur si nécessaire.
38   * 13-Fév-2016 : 1.2 - Ajout de fonctions de validation de coup pour
       les contrÃ¢leurs
39   *                       (Remerciments : L'idée est inspiré du travail
       de RUBIN Jehan et HAAS Ellen)
40   */
41  public class Jeux {
42
43      private final List<Joueur> joueurList;
44      private boolean partieEnMarche;
45      private int joueurTour;
46
47      public Jeux() {
48          joueurList = new ArrayList<>();
49          partieEnMarche = false;
50          joueurTour = -1;
51      }
52
53      public int getnbCartesDeck(int idJoueur) {
54          assert (idJoueur < joueurList.size() - 1);
55          return
                joueurList.get(idJoueur).getCarteDeck().carteRestantes();
56
57      }
58
59      /**
60       * Permet de passer au joueur suivant
61       */
62      private void prochainJoueur() {
63          joueurTour++;
64          joueurTour = joueurTour % joueurList.size();
65      }
66
67      /**
68       * Permet de savoir à qui est le tour ( id du joueur )
69       *
70       * @return l'id du joueur à qui c'est le tour de jouer
71       */
72      public int aQuiLeTour() {
73          return joueurTour;
74      }
75
76      /**
77       * Permet de démarrer une partie à nbJoueur
78       *
79       * @param nbJoueur nombre de joueur voulant jouer
80       */
81      public void demarrerPartie(int nbJoueur) {
82          assert (nbJoueur >= 2);
83          if (!partieEnMarche) {
84              finPartie();
85              joueurTour = 0;
```

```
86              for (int i = 0; i < nbJoueur; i++) {
87                  joueurList.add(new Joueur(i));
88              }
89          }
90
91          partieEnMarche = true;
92      }
93
94      /**
95       * Permet au joueur idJoueur de déclarer forfait
96       *
97       * @param idJoueur id du joueur déclarant forfait
98       * @return
99       */
100     public Resultat declarerForfait(int idJoueur) {
101         if (joueurTour == idJoueur) {
102             this.prochainJoueur();
103             return joueurList.get(idJoueur).declarerForfait();
104         } else {
105             return new RefuseResult("Tu peux pas déclarer forfait
                    pour quelqu'un d'autre.");
106         }
107     }
108
109     /**
110      * Permet de savoir l'état du jeu à tout moment ( sous un format
             JSon)
111      *
112      * @return le contenu de chaque joueur
113      */
114     public JsonObject getEtatJeu() {
115         JsonObjectBuilder obj = Json.createObjectBuilder();
116         obj.add("aQuiLeTour", this.joueurTour);
117         obj.add("partieEnCours", this.partieEnMarche);
118
119         Iterator<Joueur> j = joueurList.iterator();
120         int numJoueur = 1;
121         while (j.hasNext()) {
122             obj.add("Joueur #" + numJoueur, j.next().toJSON());
123             ++numJoueur;
124         }
125
126         return obj.build();
127     }
128
129     /**
130      * Permet de savoir si la partie est finie
131      *
132      * @return true si la partie est fini, autrement dit si un joueur
             à gagné
133      * false sinon
134      */
135     public boolean partieFini() {
136         return getJoueurGagnant() != -1; // tout les joueurs ont
                perdu sauf 1
137     }
138
139     /**
140      * Permet d'avoir l'id du joueur ayant gagné
141      *
142      * @return l'id du joueur gagnant -1 si aucun joueur n'a gagné
143      */
144     public int getJoueurGagnant() {
145         int joueursPerdants = 0;
146         int joueurGagnant = 0;
147         for (int i = 0; i < joueurList.size(); i++) {
148             if (!joueurList.get(i).aPerdu()) {
149                 joueurGagnant = i;
150             } else {
151                 joueursPerdants++;
152             }
153         }
```

```
154
155         return joueursPerdants == joueurList.size() -1? joueurGagnant
                : -1;
156     }
157
158     /**
159      * Permet au joueur idJoueur de piocher
160      *
161      * @param idJoueur id du joueur voulant piocher
162      * @return une liste de Result, PiocheResult en cas de succes de
           la pioche
163      * un RefusedResult sinon
164      */
165     public Resultat piocherCartes(int idJoueur) {
166         Resultat res;
167         if (idJoueur != aQuiLeTour()) {
168             res = new RefuseResult("Ce n'est pas le tour de ce
                    joueur.");
169             return res;
170         }
171
172         res = joueurList.get(idJoueur).piocher();
173         return res;
174     }
175
176     public boolean peutPiocherCartes(int idJoueur) {
177         return idJoueur == joueurTour &&
                joueurList.get(idJoueur).peutPiocher();
178     }
179
180     /**
181      * Permet d'attaquer un perso présent sur le deck
182      *
183      * @param idJoueur id du joueur effectuant l'action
184      * @param idAdversaire position relative sur le terrain de jeu de
           la carte
185      * attaquante
186      * @param attaqueur carte attaquant la seconde carte
187      * @param receveur carte recevant le coupÅÌ
188      * @return un AttackResult si tout c'est bien passé un
           refusedResult sinon
189      */
190     public Resultat attaquePerso(int idJoueur, int idAdversaire,
            Carte attaqueur, Carte receveur) {
191         Resultat res;
192
193         if (attaquePersoValide(idJoueur, idAdversaire, attaqueur,
                receveur)) {
194             Combattant att = (Combattant) attaqueur;
195             Cible attaquee = (Cible) receveur;
196             res = joueurList.get(idJoueur).attaque(att, attaquee);
197             joueurList.get(idAdversaire).MAJCartesPlancher();
198             this.prochainJoueur();
199         } else {
200             res = new RefuseResult("L'attaque n'est pas possible");
201         }
202
203         return res;
204     }
205
206     public boolean attaquePersoValide(int idJoueur, int idAdversaire,
            Carte attaqueur, Carte receveur) {
207
208         boolean coupP = ((this.aQuiLeTour() == idJoueur) && (idJoueur
                != idAdversaire));
209         coupP = coupP && (idAdversaire >= 0) && (joueurList.size() >=
                idAdversaire);
210         coupP = coupP && (attaqueur instanceof Combattant) &&
                (receveur instanceof Cible);
211         if (coupP) {
212             coupP = coupP &&
                    joueurList.get(idJoueur).carteDansJeu(attaqueur);
```

```
213             coupP = coupP &&
                    joueurList.get(idAdversaire).carteDansJeu(receveur);
214             coupP = coupP && ((Cible) receveur).peutEtreAttaque();
215         }
216         return coupP;
217
218     }
219
220     /**
221      * Permet d'attaquer un perso présent sur le deck
222      *
223      * @param idJoueur id du joueur effectuant l'action
224      * @param idAdversaire position relative sur le terrain de jeu de
             la carte
225      * attaquante
226      * @param attaqueur Carte effectuant l'attaque sur le joueur
227      * @return un AttackResult si tout c'est bien passé un
             refusedResult sinon
228      */
229     public Resultat attaqueJoueur(int idJoueur, int idAdversaire,
            Carte attaqueur) {
230         Resultat res;
231
232         if (attaqueJoueurValide(idJoueur, idAdversaire, attaqueur)) {
233             Combattant att = (Combattant) attaqueur;
234             res = joueurList.get(idJoueur).attaque(att,
                        joueurList.get(idAdversaire));
235             this.prochainJoueur();
236         } else {
237             res = new RefuseResult("L'attaque n'est pas possible");
238         }
239
240         return res;
241     }
242
243     public boolean attaqueJoueurValide(int idJoueur, int
            idAdversaire, Carte attaqueur) {
244
245         boolean verifValide = idJoueur == aQuiLeTour() &&
                idAdversaire >= 0 && idAdversaire <= joueurList.size() &&
                idJoueur != idAdversaire;
246         if (verifValide) {
247             verifValide =
                        joueurList.get(idJoueur).getCarteEnJeu().containsKey(attaqueur.getCardID());
248             verifValide = verifValide &&
                        joueurList.get(idAdversaire).peutEtreAttaque();
249             verifValide = verifValide && (attaqueur instanceof
                        Combattant);
250         }
251
252         return verifValide;
253     }
254
255     /**
256      * Permet d'ajouter une liste d'enchant sur un joueur
257      *
258      * @param idJoueur id du joueur qui va recevoir les enchants
259      * @param carteTouche La carte qui va recevoir les enchants
260      * @param enchant listes des positions relatives dans le deck des
             cartes
261      * d'enchantements à appliquer
262      * @return un list Result, chaqu'un contenant un EnchantResult si
             l'enchant
263      * fonctionne un refused Result sinon
264      */
265     public Resultat ajouterEnchantements(int idJoueur, Carte
            carteTouche, List<Carte> enchant) {
266         Resultat res;
267
268         if (peutAjouterEnchantements(idJoueur, carteTouche, enchant))
                {
269             List<Enchant> enchs = new ArrayList<>();
```

```
270             for (Carte c : enchant) {
271                 enchs.add((Enchant) c);
272             }
273             res = joueurList.get(idJoueur).ajouterEnchants(enchs,
                    carteTouche);
274             this.prochainJoueur();
275         } else {
276             res = new RefuseResult("Vous ne pouvez pas enchanter
                    cette cartes avec votre sélection.");
277         }
278         return res;
279     }
280
281     public boolean peutAjouterEnchantements(int idJoueur, Carte
            carteTouche, List<Carte> enchants) {
282         boolean verifValide = idJoueur == aQuiLeTour();
283         Joueur j = joueurList.get(idJoueur);
284         boolean carteDeploye = false;
285         for (Carte c : enchants) {
286             verifValide = verifValide && j.carteDansMain(c);
287         }
288         for (Joueur joueurAct : joueurList) {
289             carteDeploye = carteDeploye ||
                    joueurAct.carteDansJeu(carteTouche);
290         }
291         carteDeploye = carteDeploye && carteTouche instanceof Perso;
292         if (carteDeploye) {
293             Perso p = (Perso) carteTouche;
294             carteDeploye = p.getArme().peutAjouterEnchantement();
295         }
296
297         return verifValide && carteDeploye;
298     }
299
300     public Collection<Carte> getCartesMainJoueur(int idJoueur) {
301         return joueurList.get(idJoueur).getMain().values();
302     }
303
304     public Collection<Perso> getCartesJeuJoueur(int idJoueur) {
305         Perso p;
306         return joueurList.get(idJoueur).getCarteEnJeu().values();
307     }
308
309     public Collection<Carte> getCimetiereJoueur(int idJoueur) {
310         return joueurList.get(idJoueur).getCimetiere();
311     }
312
313     /**
314      * Permet de placer un personnage
315      *
316      * @param idJoueur id relatif du joueur dans la liste Joueur List
317      * @param personnage position relative dans la main du joueur à
           placer
318      * @param arme position relative dans la main de l'arme a placer
319      * @param enchants liste des cartes d'enchants à placer sur le
           perso
320      * @return un PersoDeploieResult si tout ce passe bien un
           refusedResult
321      * sinon
322      */
323     public Resultat placerPerso(int idJoueur, Carte personnage, Carte
            arme, List<Carte> enchants) {
324         Resultat res;
325         if (this.peutDeployerPerso(idJoueur, personnage, arme,
                enchants)) {
326             res = joueurList.get(idJoueur).placerPerso((Perso)
                    personnage, (Arme) arme, enchants);
327             this.prochainJoueur();
328
329         } else {
330             res = new RefuseResult("Vous n'avez pas le droit
                    d'utiliser une des cartes choisi.");
```

```
331         }
332         return res;
333
334     }
335
336     public boolean peutDeployerPerso(int idJoueur, Carte perso, Carte
        arme, List<Carte> enchants) {
337         boolean verif = idJoueur == aQuiLeTour();
338         boolean enchFac = false;
339         Joueur j = joueurList.get(idJoueur);
340         verif = verif && j.carteDansMain(perso) &&
                j.carteDansMain(arme);
341         verif = verif && (perso instanceof Perso) && (arme instanceof
                Arme);
342         for (Carte c : enchants) {
343             verif = verif && j.carteDansMain(c);
344             verif = verif && c instanceof Enchant;
345             enchFac = enchFac || c instanceof EnchantFacile;
346         }
347         if (verif) {
348             Perso p = (Perso) perso;
349             Arme a = (Arme) arme;
350             verif = !a.armeEstDeploye();
351             verif = verif && (a.peutUtiliserArme(p) || enchFac);
352         }
353
354         return verif;
355
356     }
357
358     /**
359      * Permet de défausser n Carte à partie de l'api
360      *
361      * @param idJoueur id du joueur souhaitant se défausser
362      * @param defausse Liste des positions relatives dans la main des
           cartes à
363      * defausser
364      * @return un defausseResult si tout ce passe bien un
           refusedResult en cas
365      * d'erreur
366      */
367     public Resultat defausserCartes(int idJoueur, List<Carte>
        defausse) {
368         Resultat res;
369         if (peutDefausserCartes(idJoueur, defausse)) {
370             res = joueurList.get(idJoueur).defausserCartes(defausse);
371             this.prochainJoueur();
372         } else {
373             res = new RefuseResult("La liste est soit vide, soit
                    rempli de cartes pas situés dans la main.");
374         }
375
376         return res;
377
378     }
379
380     public boolean peutDefausserCartes(int idJoueur, List<Carte>
        defausse) {
381
382         boolean coupValide = idJoueur == aQuiLeTour();
383         coupValide = coupValide && defausse != null;
384         coupValide = coupValide && !defausse.isEmpty();
385         Joueur joueurAct = joueurList.get(idJoueur);
386         for (Carte c : defausse) {
387             coupValide = coupValide && joueurAct.carteDansMain(c);
388             if (!coupValide) {
389                 break;
390             }
391         }
392
393         return coupValide;
394
```

```
395     }
396
397     /**
398      * Permet de soigner un personnage du jeu
399      *
400      * @param idJoueur id du joueur qui effectue l'action
401      * @param soigneur id relatif de la position de la carte sur le
           terrain de
402      * jeu
403      * @param soignee id relatif de la position de la carte sur le
           terrain de
404      * jeu ( carte à soigner)
405      * @return un SoinsResult si tout c'est bien passé un
           RefusedResult sinon
406      */
407     public Resultat soignerPerso(int idJoueur, Carte soigneur, Carte
           soignee) {
408         Resultat res;
409
410         if (peutSoignerPerso(idJoueur, soigneur, soignee)) {
411             Soigneur s = (Soigneur) soigneur;
412             Perso p = (Perso) soignee;
413             res = joueurList.get(idJoueur).soignerPerso(s, p);
414             this.prochainJoueur();
415
416         } else {
417             res = new RefuseResult("Vous n'avez pas le droit de faire
                    ce soin.");
418         }
419
420         return res;
421
422     }
423
424     public boolean peutSoignerPerso(int idJoueur, Carte soigneur,
           Carte soignee) {
425         boolean coupValide = idJoueur == aQuiLeTour();
426         coupValide = coupValide && soigneur != soignee;
427         Joueur joueurAct = joueurList.get(idJoueur);
428         coupValide = coupValide && joueurAct.carteDansJeu(soigneur)
429                 && joueurAct.carteDansJeu(soignee);
430         coupValide = coupValide && soigneur instanceof Soigneur &&
               soignee instanceof Perso;
431         if (coupValide) {
432             Soigneur s = (Soigneur) soigneur;
433             Perso p = (Perso) soignee;
434             coupValide = s.peutSoigner(p);
435         }
436         return coupValide;
437     }
438
439     /**
440      * Permet de liberer les informations de jeu et reinialisé les
           valeurs de
441      * l'api
442      */
443     public void finPartie() {
444         joueurList.clear();
445         joueurTour = 0;
446         partieEnMarche = false;
447     }
448 }
```

5 ResultUtils

5.1 Resultat.Java

```
1  package cardgame.ResultUtils;
2
3  /**
```

```
4   * Interface utilisé pour communiquer aux joueurs les conséquences
        des coups joués/refusés.
5   * Resultat est implémenté par une sous-classe pour chaque type de
        coup possible, afin de
6   * pouvoir transmettre la totalité des informations pertinentes.
7   * @author Mathieu Gravel GRAM02099206
8   * @author Nicolas Reynaud REYN23119308
9   * @version 1.0 08-Fév-2016 : 1.0 - Version initiale.
10  */
11  public interface Resultat {
12
13      /**
14       * Déclaration de coupAMarcher, Interface Result
15       *
16       * @return rien, non déclarée ici
17       */
18      public boolean coupAMarcher();
19
20      /**
21       * Déclaration de getDescription, Interface Result
22       *
23       * @return rien, non déclarée ici
24       */
25      public String getDescription();
26
27      /**
28       * Déclaration de coupJouerPar, Interface Result
29       *
30       * @return rien, non déclarée ici
31       */
32      public int coupJouerPar();
33
34      /**
35       * Déclaration de setJoueur, Interface Result
36       * Ce setter existe seulement afin d'éviter d'envoyer aux classes
37       * Cartes le Id du joueur.
38       * @param idJoueur non définie ici
39       */
40      public void setJoueur(int idJoueur);
41  }
```

5.2 AttaquePersoResult.Java

```
1   package cardgame.ResultUtils;
2
3   /**
4    * Implémentation de Resultat pour décrire les conséquences d'une
         attaque
5    * perso-joueur.
6    *
7    * @author Mathieu Gravel GRAM02099206
8    * @author Nicolas Reynaud REYN23119308
9    * @version 1.0
10   *
11   * 08-Fév-2016 : 1.0 - Version initiale.
12   */
13  public class AttaquePersoResult implements Resultat {
14
15      private final int dommageRecu;
16      private int attaqueur;
17      private final int idCarte;
18      private final int idCarteAttack;
19      private final boolean attaqueTuer;
20      private String desc;
21
22      public AttaquePersoResult(int dmg, int joueurId, int carteId, int
            persoCoupId, boolean attaqueTue) {
23          dommageRecu = dmg;
24          attaqueur = joueurId;
25          idCarte = carteId;
```

```
26          attaqueTuer = attaqueTue;
27          idCarteAttack = persoCoupId;
28          desc = "L'attaque de la carte " + carteId + "sur " +
                persoCoupId + "a causé " + dmg + "dégats.\n";
29          if (attaqueTue) {
30              desc = desc + "Le perso à été tué.";
31          }
32      }
33
34      public AttaquePersoResult(int dmg, int carteId, int persoCoupId,
            boolean attaqueTue) {
35          dommageRecu = dmg;
36          idCarte = carteId;
37          attaqueTuer = attaqueTue;
38          idCarteAttack = persoCoupId;
39          desc = "L'attaque de la carte " + carteId + "sur " +
                persoCoupId + "a causé " + dmg + "dégats.\n";
40          if (attaqueTue) {
41              desc = desc + "Le perso à été tué.";
42          }
43      }
44
45      /**
46       * Getter
47       *
48       * @return le dommage reÃğu par l'attaque.
49       */
50      public int getDmgEffectue() {
51          return dommageRecu;
52      }
53
54      /**
55       * @return l'attaque a-t-elle fait des dégats réels?
56       */
57      @Override
58      public boolean coupAMarcher() {
59          return dommageRecu > 0;
60      }
61
62      /**
63       * Getter
64       *
65       * @return Description de ce type de coup.
66       */
67      @Override
68      public String getDescription() {
69          return desc;
70      }
71
72      /**
73       * Getter
74       *
75       * @return l'identifiant du joueur qui a joué de coup.
76       */
77      @Override
78      public int coupJouerPar() {
79          return attaqueur;
80      }
81
82      /**
83       * Getter
84       *
85       * @return l'identifiant de la carte qui a attaqué.
86       */
87      public int getAttaqueurPerso() {
88          return idCarte;
89      }
90
91      /**
92       * Getter
93       *
```

```
94       * @return l'identifiant de la carte qui a reÃğu l'attaque ou -1
             si le joeur
95       * a pris le coup.
96       *
97       */
98      public int getPersonneAttaque() {
99          return idCarteAttack;
100     }
101
102     /**
103      * Getter
104      *
105      * @return True si le perso est mort par cette attaque,false
             sinon.
106      */
107     public boolean attaqueATuer() {
108         return attaqueTuer;
109     }
110
111     /**
112      * Setter
113      *
114      * @param joueurId l'identifiant du joueur qui a fait le coup.
115      */
116     @Override
117     public void setJoueur(int joueurId) {
118         attaqueur = joueurId;
119     }
120 }
```

5.3 AttaquePlayerResult.Java

```
1  package cardgame.ResultUtils;
2
3  /**
4   * Implémentation de Resultat pour décrire les conséquences d'une
        attaque
5   * perso-joueur.
6   *
7   * @author Mathieu Gravel GRAM02099206
8   * @author Nicolas Reynaud REYN23119308
9   * @version 1.0
10  *
11  * 08-Fév-2016 : 1.0 - Version initiale.
12  */
13 public class AttaquePlayerResult implements Resultat {
14
15      private final int dommageRecu;
16      private int attaqueur;
17      private final int idCarte;
18      private final int idAdversaire;
19      private final boolean attaqueTuer;
20      private  String desc;
21
22      public AttaquePlayerResult(int dmg, int joueurId, int carteId,
            int adversaireId, boolean attaqueTue) {
23          dommageRecu = dmg;
24          attaqueur = joueurId;
25          idCarte = carteId;
26          attaqueTuer = attaqueTue;
27          idAdversaire = adversaireId;
28          desc = "L'attaque de la carte " + carteId + "sur le joueur" +
                idAdversaire + "a causé " + dmg + "dégats.\n";
29          if (attaqueTue)
30              desc = desc + "Le joueur à été tué.";
31      }
32
33      public AttaquePlayerResult(int dmg, int carteId, int
            adversaireId, boolean attaqueTue) {
34          dommageRecu = dmg;
```

```
35          idCarte = carteId;
36          attaqueTuer = attaqueTue;
37          idAdversaire = adversaireId;
38          desc = "L'attaque de la carte " + carteId + "sur le joueur" +
                idAdversaire + "a causé " + dmg + "dégats.\n";
39          if (attaqueTue)
40              desc = desc + "Le joueur à été tué.";
41      }
42
43      /**
44       * Getter
45       *
46       * @return le dommage reÃğu par l'attaque.
47       */
48      public int getDmgEffectue() {
49          return dommageRecu;
50      }
51
52      /**
53       * @return l'attaque a-t-elle fait des dégats réels?
54       */
55      @Override
56      public boolean coupAMarcher() {
57          return dommageRecu > 0;
58      }
59
60      /**
61       * Getter
62       *
63       * @return Description de ce type de coup.
64       */
65      @Override
66      public String getDescription() {
67          return desc;
68      }
69
70      /**
71       * Getter
72       *
73       * @return l'identifiant du joueur qui a joué de coup.
74       */
75      @Override
76      public int coupJouerPar() {
77          return attaqueur;
78      }
79
80      /**
81       * Getter
82       *
83       * @return l'identifiant de la carte qui a attaqué.
84       */
85      public int getAttaqueurPerso() {
86          return idCarte;
87      }
88
89      /**
90       * Getter
91       * @return True si le perso est mort par cette attaque,false
               sinon.
92       */
93      public boolean attaqueATuer() {
94          return attaqueTuer;
95      }
96
97      /**
98       * Setter
99       * @param joueurId l'identifiant du joueur qui a fait le coup.
100      */
101     @Override
102     public void setJoueur(int joueurId) {
103         attaqueur = joueurId;
104     }
```

```
105 }
```

5.4 DefausseResult.Java

```
1  package cardgame.ResultUtils;
2
3  import cardgame.JeuxCartes.Carte;
4  import java.util.ArrayList;
5  import java.util.List;
6  import javax.json.JsonObject;
7
8  /**
9   * Implémentation de Resultat pour décrire les conséquences d'une
        action de
10  * défaussage.
11  *
12  * @author Mathieu Gravel GRAM02099206
13  * @author Nicolas Reynaud REYN23119308
14  * @version 1.0
15  *
16  * 08-Fév-2016 : 1.0 - Version initiale.
17  */
18 public class DefausseResult implements Resultat {
19
20      private final List<Integer> cartesId;
21      private final List<JsonObject> cartesJSON;
22      private int joueurId;
23      private final String description;
24      private final boolean coupAFonctionne;
25
26      public DefausseResult(int idJoueur, boolean coupCorrect,
            List<Carte> cartes) {
27          cartesId = new ArrayList<>();
28          cartesJSON = new ArrayList<>();
29          String cartesStr = "";
30          for (Carte cartePioche : cartes) {
31              cartesId.add(cartePioche.getCardID());
32              cartesJSON.add(cartePioche.toJSON());
33              cartesStr = cartesStr + cartePioche.toJSON().toString() +
                    "\n";
34          }
35          joueurId = idJoueur;
36          description = "Les cartes suivantes ont été défaussés :" +
                cartesStr;
37          coupAFonctionne = coupCorrect;
38      }
39
40      /**
41       * @return True si l'action a fonctionné,false sinon.
42       */
43      @Override
44      public boolean coupAMarcher() {
45          return coupAFonctionne;
46      }
47
48      /**
49       * Getter
50       *
51       * @return Description de ce type de coup.
52       */
53      @Override
54      public String getDescription() {
55          return description;
56      }
57
58      /**
59       * Getter
60       *
61       * @return l'identifiant du joueur qui a joué de coup.
62       */
```

```
63      @Override
64      public int coupJouerPar() {
65          return joueurId;
66      }
67
68      /**
69       * Getter
70       *
71       * @return les identifiant des cartes défaussés. (Utile pour une
             vue qui
72       * l'utilise en tandem avec le Json de la partie.)
73       */
74      public List<Integer> getCartesID() {
75          return cartesId;
76      }
77
78      /**
79       * Getter
80       *
81       * @return La représentation JSON des cartes défaussés. (Utile
             pour une vue
82       * qui veut faire des animations.)
83       */
84      public List<JsonObject> getCartesJSON() {
85          return cartesJSON;
86      }
87
88      /**
89       * Setter
90       *
91       * @param idJoueur l'identifiant du joueur qui a fait le coup.
92       */
93      @Override
94      public void setJoueur(int idJoueur) {
95          this.joueurId = idJoueur;
96      }
97  }
```

5.5 EnchantResult.Java

```
1   package cardgame.ResultUtils;
2
3   import cardgame.JeuxCartes.Carte;
4   import cardgame.JeuxCartes.Enchant;
5   import java.util.List;
6   import javax.json.JsonObject;
7
8   /**
9    * Implémentation de Resultat pour décrire les conséquences d'une
         action
10   * d'enchantement.
11   *
12   *
13   * @author Mathieu Gravel GRAM02099206
14   * @author Nicolas Reynaud REYN23119308
15   * @version 1.0
16   *
17   * 08-Fév-2016 : 1.0 - Version initiale.
18   */
19  public class EnchantResult implements Resultat {
20
21      private final String description;
22      private int joueurId;
23      private final boolean coupAFonctionne;
24      private final Carte carteEnchant;
25      private final List<Enchant> enchant;
26
27      public EnchantResult(boolean coupCorrect, Carte
            carteEnch,List<Enchant> enchants) {
28          coupAFonctionne = coupCorrect;
```

```
29          carteEnchant = carteEnch;
30          enchant = enchants;
31          String enchStr = "";
32          for (Enchant ench : enchants)
33              enchStr = enchStr + ench.toJSON().toString();
34          description = "La carte " + carteEnch.toJSON().toString() +
                "a recu les enchantements suivants :" + enchStr;
35      }
36
37      public EnchantResult(int jId, boolean coupCorrect, Carte
            carteEnch,List<Enchant> enchants) {
38          joueurId = jId;
39          coupAFonctionne = coupCorrect;
40          carteEnchant = carteEnch;
41          enchant = enchants;
42          description = "Enchantement d'une carte";
43      }
44
45      /**
46       * Getter
47       *
48       * @return identifiant de la carte enchanté.
49       */
50      public JsonObject getCarteEnchante() {
51          return carteEnchant.toJSON();
52      }
53
54      /**
55       * @return True si l'action a fonctionné,false sinon.
56       */
57      @Override
58      public boolean coupAMarcher() {
59          return coupAFonctionne;
60      }
61
62      /**
63       * Getter
64       *
65       * @return Description de ce type de coup.
66       */
67      @Override
68      public String getDescription() {
69          return description;
70      }
71
72      /**
73       * Getter
74       *
75       * @return l'identifiant du joueur qui a joué de coup.
76       */
77      @Override
78      public int coupJouerPar() {
79          return joueurId;
80      }
81
82      /**
83       * Setter
84       *
85       * @param idJoueur l'identifiant du joueur qui a fait le coup.
86       */
87      @Override
88      public void setJoueur(int idJoueur) {
89          this.joueurId = idJoueur;
90      }
91  }
```

5.6 ForfaitResult.Java

```
1   package cardgame.ResultUtils;
2
```

```java
/**
 * 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

```
1  package cardgame.ResultUtils;
2
3  /**
4   * Implémentation de Resultat pour décrire la fin d'une partie.
5   *
6   *
7   * @author Mathieu Gravel GRAM02099206
8   * @author Nicolas Reynaud REYN23119308
9   * @version 1.0
10  *
11  * 08-Fév-2016 : 1.0 - Version initiale.
12  */
13 public class FinDePartieResult implements Resultat {
14
15     private final String description;
16     private int joueurId;
17     private final boolean coupAFonctionne;
18     private final int joueurGagne;
19
20     public FinDePartieResult(int jId, boolean coupCorrect, int
           idJoueurGagne) {
21         joueurId = jId;
22         coupAFonctionne = coupCorrect;
23         joueurGagne = idJoueurGagne;
24         description = "Le joueur" + idJoueurGagne + "vient de gagner
               la partie";
25     }
26
27     /**
28      * @return True si l'action a fonctionné,false sinon.
29      */
30     @Override
31     public boolean coupAMarcher() {
32         return  coupAFonctionne;
33     }
34
35     /**
36      * Getter
37      *
38      * @return Description de ce type de coup.
39      */
40     @Override
41     public String getDescription() {
42         return description;
43     }
44
45     /**
46      * Getter
47      *
48      * @return l'identifiant du joueur qui a joué de coup.
49      */
50     @Override
51     public int coupJouerPar() {
52         return joueurId;
53     }
54
55     /**
56      * Getter
57      * @return l'identifiant du joueur  qui a gagné.
58      */
59     public int getJoueurQuiAGagne() {
60         return joueurGagne;
61     }
62
63     /**
64      * Setter
65      * @param idJoueur l'identifiant du joueur qui a fait le coup.
66      */
67     @Override
```

```
68      public void setJoueur(int idJoueur) {
69          this.joueurId = idJoueur;
70      }
71  }
```

5.8 PersoDeploieResult.Java

```
1   package cardgame.ResultUtils;
2
3   import cardgame.JeuxCartes.Carte;
4
5   /**
6    * Implémentation de Resultat pour décrire la conséquence d'un
         déploiment d'un
7    * perso et de son arme sur le jeu.
8    *
9    *
10   * @author Mathieu Gravel GRAM02099206
11   * @author Nicolas Reynaud REYN23119308
12   * @version 1.0
13   *
14   * 08-Fév-2016 : 1.0 - Version initiale.
15   */
16  public class PersoDeploieResult implements Resultat {
17
18      private final String description;
19      private int joueurId;
20      private final boolean coupAFonctionne;
21
22      public PersoDeploieResult(int jId, boolean coupCorrect, Carte
            perso) {
23          joueurId = jId;
24          coupAFonctionne = coupCorrect;
25          description = "Le joueur " + jId + "vient de déployer sur le
                jeu : " + perso.toJSON();
26      }
27
28      /**
29       * @return True si l'action a fonctionné,false sinon.
30       */
31      @Override
32      public boolean coupAMarcher() {
33          return coupAFonctionne;
34      }
35
36      /**
37       * Getter
38       *
39       * @return Description de ce type de coup.
40       */
41      @Override
42      public String getDescription() {
43          return description;
44      }
45
46      /**
47       * Getter
48       *
49       * @return l'identifiant du joueur qui a joué de coup.
50       */
51      @Override
52      public int coupJouerPar() {
53          return joueurId;
54      }
55
56      /**
57       * Setter
58       *
59       * @param idJoueur l'identifiant du joueur qui a fait le coup.
60       */
```

```
61      @Override
62      public void setJoueur(int idJoueur) {
63          this.joueurId = idJoueur;
64      }
65  }
```

5.9 PiocheResult.Java

```
1   package cardgame.ResultUtils;
2
3   import cardgame.JeuxCartes.Carte;
4   import java.util.ArrayList;
5   import java.util.List;
6   import javax.json.JsonObject;
7
8   /**
9    * Implémentation de Resultat pour décrire la conséquence d'une
         pioche.
10   *
11   *
12   * @author Mathieu Gravel GRAM02099206
13   * @author Nicolas Reynaud REYN23119308
14   * @version 1.0
15   *
16   * 08-Fév-2016 : 1.0 - Version initiale.
17   */
18  public class PiocheResult implements Resultat {
19
20      private final String description;
21      private int joueurId;
22      private final boolean coupAFonctionne;
23      private final List<Integer> cartesId;
24      private final List<JsonObject> cartesJSON;
25
26      public PiocheResult(int jId, boolean coupCorrect, List<Carte>
            cartes) {
27          joueurId = jId;
28          coupAFonctionne = coupCorrect;
29          cartesId = new ArrayList<>();
30          cartesJSON = new ArrayList<>();
31          String cStr = "";
32          for (Carte cartePioche : cartes) {
33              cartesId.add(cartePioche.getCardID());
34              cartesJSON.add(cartePioche.toJSON());
35              cStr = cStr + cartePioche.toJSON().toString();
36          }
37          description = "Le joueur " + jId + "vient de piocher les
                cartes suivantes : \n" + cStr;
38      }
39
40      /**
41       * Getter
42       *
43       * @return les identifiant des cartes pigés. (Utile pour une vue
             qui
44       * l'utilise en tandem avec le Json de la partie.)
45       */
46      public List<Integer> getCartesID() {
47          return cartesId;
48      }
49
50      /**
51       * Getter
52       *
53       * @return La représentation JSON des cartes pigés. (Utile pour
             une vue qui
54       * veut faire des animations.)
55       */
56      public List<JsonObject> getCartesJSON() {
57          return cartesJSON;
```

```
58      }
59
60      /**
61       * @return True si l'action a fonctionné,false sinon.
62       */
63      @Override
64      public boolean coupAMarcher() {
65          return coupAFonctionne;
66      }
67
68      /**
69       * Getter
70       *
71       * @return Description de ce type de coup.
72       */
73      @Override
74      public String getDescription() {
75          return description;
76      }
77
78      /**
79       * Getter
80       *
81       * @return l'identifiant du joueur qui a joué de coup.
82       */
83      @Override
84      public int coupJouerPar() {
85          return joueurId;
86      }
87
88      /**
89       * Setter
90       *
91       * @param idJoueur l'identifiant du joueur qui a fait le coup.
92       */
93      @Override
94      public void setJoueur(int idJoueur) {
95          this.joueurId = idJoueur;
96      }
97  }
```

5.10 RefuseResult.Java

```
1   package cardgame.ResultUtils;
2
3   /**
4    * Implémentation de Resultat pour décrire la conséquence d'un coup
         refusé
5    * puisqu'il était impossible.
6    *
7    *
8    * @author Mathieu Gravel GRAM02099206
9    * @author Nicolas Reynaud REYN23119308
10   * @version 1.0
11   *
12   * 08-Fév-2016 : 1.0 - Version initiale.
13   */
14  public class RefuseResult implements Resultat {
15
16      private final String description;
17      private int joueurId;
18
19      public RefuseResult(int idJoueur, String coupRefuse) {
20          description = coupRefuse;
21          joueurId = idJoueur;
22      }
23
24      public RefuseResult(String coupRefuse) {
25          description = coupRefuse;
26      }
```

```
27
28      /**
29       * @return True si l'action a fonctionné,false sinon.
30       */
31      @Override
32      public boolean coupAMarcher() {
33          return false;
34      }
35
36      /**
37       * Getter
38       *
39       * @return Description de ce type de coup.
40       */
41      @Override
42      public String getDescription() {
43          return description;
44      }
45
46      /**
47       * Getter
48       *
49       * @return l'identifiant du joueur qui a joué de coup.
50       */
51      @Override
52      public int coupJouerPar() {
53          return joueurId;
54      }
55
56      /**
57       * Setter
58       *
59       * @param idJoueur l'identifiant du joueur qui a fait le coup.
60       */
61      @Override
62      public void setJoueur(int idJoueur) {
63          this.joueurId = idJoueur;
64      }
65  }
```

5.11 SoinsResult.Java

```
1   package cardgame.ResultUtils;
2
3   /**
4    * Implémentation de Resultat pour décrire la conséquence d'un
         sortilège de
5    * soins.
6    *
7    *
8    * @author Mathieu Gravel GRAM02099206
9    * @author Nicolas Reynaud REYN23119308
10   * @version 1.0
11   *
12   * 08-Fév-2016 : 1.0 - Version initiale.
13   */
14  public class SoinsResult implements Resultat {
15
16      private final String description;
17      private int joueurId;
18      private final boolean coupAFonctionne;
19      private final int healerId;
20      private final int persoSoigneeId;
21
22      public SoinsResult(int jId, boolean coupCorrect, int hId, int
            cId) {
23          description = "Le personnage " + hId + " vient de soigner " +
                cId + ".";
24          joueurId = jId;
25          coupAFonctionne = coupCorrect;
```

```
26          healerId = hId;
27          persoSoigneeId = cId;
28      }
29
30      public SoinsResult(boolean coupCorrect, int hId, int cId) {
31          description = "Le personnage " + hId + " vient de soigner " +
                cId + ".";
32          coupAFonctionne = coupCorrect;
33          healerId = hId;
34          persoSoigneeId = cId;
35      }
36
37      /**
38       * Getter
39       * @return L'identifiant du perso qui a fait le sort de soins.
40       */
41      public int getHealerId() {
42          return healerId;
43      }
44
45      /**
46       * Getter
47       * @return L'identifiant du perso soignée.
48       */
49      public int getCarteSoigneeId() {
50          return persoSoigneeId;
51      }
52
53      /**
54       * @return True si l'action a fonctionné,false sinon.
55       */
56      @Override
57      public boolean coupAMarcher() {
58          return coupAFonctionne;
59      }
60
61      /**
62       * Getter
63       *
64       * @return Description de ce type de coup.
65       */
66      @Override
67      public String getDescription() {
68          return description;
69      }
70
71      /**
72       * Getter
73       *
74       * @return l'identifiant du joueur qui a joué de coup.
75       */
76      @Override
77      public int coupJouerPar() {
78          return joueurId;
79      }
80
81      /**
82       * Setter
83       *
84       * @param idJoueur l'identifiant du joueur qui a fait le coup.
85       */
86      @Override
87      public void setJoueur(int idJoueur) {
88          this.joueurId = idJoueur;
89      }
90
91  }
```