

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

Résumé	1
1 CarteJeux	1
1.1 Carte.Java	1
1.2 Perso.Java	2
1.3 Carte.Java	4
1.4 Enchant.Java	6
1.5 EnchantStase.Java	7
1.6 EnchantNeutre.Java	8
1.7 EnchantFacile.Java	8
1.8 EnchantDegatPlus.Java	9
1.9 EnchantDegatMoins.Java	9
1.10 Cible.Java	10
1.11 Soigneur.Java	10
1.12 Combattant.Java	11
1.13 Guerrier.Java	12
1.14 Pretre.Java	12
1.15 Paladin.Java	13
1.16 Deck.Java	14
1.17 Joueur.Java	16
2 Init	21
2.1 ArmeFactory.Java	21
2.2 EnchantFactory.Java	22
2.3 PersoFactory.Java	23
3 Regles	24
3.1 Regle.Java	24
3.2 TypeArme.Java	24
4 API	25
4.1 Jeux.Java	25

5	ResultUtils	32
5.1	Resultat.Java	32
5.2	AttaquePersoResult.Java	33
5.3	AttaquePlayerResult.Java	35
5.4	DefausseResult.Java	37
5.5	EnchantResult.Java	38
5.6	ForfaitResult.Java	39
5.7	FinDePartieResult.Java	41
5.8	PersoDeploieResult.Java	42
5.9	PiocheResult.Java	43
5.10	RefuseResult.Java	44
5.11	SoinsResult.Java	45

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
7   *   types de cartes
8   * du jeu. (La raison derrière le choix de classe abstraite et non
9   *   d'interface
10  * réside dans l'identifiant unique. Celle-ci nous permet de lier une
11  *   carte du
12  * modèle aux demandes du controleurs si nécessaire.)
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 abstract class Carte {
21
22     //Int statique utilisé pour s'assurer que chaque carte ait un Id
23     //unique.
24     static int SSID = 0;
25     private final int cardID;
26
27     /**
28     * Description des fonctions représentant le JSon des cartes, non
29     *   définie
30     * ici
31     *
32     * @return null, fonction non définie ici
33     */
34     public abstract JsonObject toJSON();
35
36     /**
37     * Constructeur par défaut. Initialise l'identifiant de la carte.
38     */
39     public Carte() {
40         cardID = SSID;
41         ++Carte.SSID;
42     }
43
44     /**
45     * Permet d'avoir l'id de la carte.
46     */

```

```

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 =
150             attaqueur.forceAttaque(this.armePerso.getTypeArme());
151         this.hp -= degat;
152         res = new AttaquePersoResult(degat, this.getCardID(),
153             attaqueur.getCardID(), estMort());
154         return res;
155     }
156 }

```

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
11   * par
12   * ArmeFactory (Afin d'attribuer les bons attributs pour chaque type
13   * d'armes),
14   * Arme permet de traiter la logique de : - Force d'attaque -
15   *   Équipage
16   * d'enchantements - Vérification de l'arme pour un déploiement.
17   *
18   * @author Mathieu Gravel GRAM02099206
19   * @author Nicolas Reynaud REYN23119308
20   * @version 1.2
21   *
22   * Historique : 08-Fév-2016 : 1.0 - Version initiale.
23   * 11-Fév-2016 : 1.1 - Découpage de ListUtilisateurs pour le placer
24   *   dans Perso.
25   * 1.2 - Ajout de fonctions pour vérifier si l'amr
26   *   est déployé.
27   */
28  public class Arme extends Carte {
29
30      protected TypeArme type;
31      /**
32       * Boolean notant explicitement si l'arme est stased pour fins
33       * d'efficacités.
34       */
35      protected boolean estStase;
36      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 lequel 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 ||
82             p.getArmesUtilisables().contains(this.type));
83     }
84
85     /**
86      * Permet d'ajouter un enchantement à l'arme courante
87      *
88      * @param ench Enchant à appliquer à l'arme
89      */
90     protected void ajouterEnchant(Enchant ench) {
91         if (!this.estStase) {
92             listEnchant.add(ench);
93             ench.placerEnchant(this);
94         }
95     }
96
97     /**
98      * Permet de réinitialiser l'arme à son état d'origine.
99      */
100     protected void reset() {
101         this.listEnchantStase = new ArrayList<>(this.listEnchant);
102         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,
121                 it.next().toJSON());
122             ++enchNum;
123         }
124         enchNum = 1;
125         it = listEnchantStase.iterator();
126         while (it.hasNext()) {
127             obj.add("Enchantement inactif #" + enchNum,
128                 it.next().toJSON());
129             ++enchNum;
130         }
131         return obj.build();
132     }
133
134     /**
135     * Getter
136     * @return Type d'arme
137     */
138     public TypeArme getTypeArme(){
139         return type;
140     }
141
142     /**
143     * Setter notant que l'arme a été déployé.
144     */
145     protected void deployerArme() {
146         armeUtilise = true;
147     }
148
149     /**
150     * Permet de savoir si une arme est Stase
151     *
152     * @return true si l'arme est stase false sinon
153     */
154     public boolean peutAjouterEnchantement() {
155         return !estStase;
156     }
157
158     /**
159     * Getter
160     * @return Bool dictant si l'arme est déployé sur un perso.
161     */
162     public boolean armeEstDeploye() {
163         return armeUtilise;
164     }

```

1.4 Enchant.Java

```

1 package cardgame.JeuxCartes;
2

```

```

3  import javax.json.*;
4  /**
5   * Classe abstraite, Enchant sert d'interface commun pour tout les
      types
6   * d'enchantements du jeu.
7   *
8   * Les implémentations de cette classes sont basés sur le patron
      Visiteur,
9   * afin de pouvoir ajouter dynamiquement une nouvelle opération à arme
10  * (ajout l'enchantement) sans toutefois modifier sa classe.
11  * Ceci nous permet d'assurer que tout ajouts d'enchantements basés
12  * sur des valeurs d'armes existantes n'auront pas besoin de modifier
      Arme.
13  * (https://sourcemaking.com/design_patterns/visitor)
14  *
15  * @author Mathieu Gravel GRAM02099206
16  * @author Nicolas Reynaud REYN23119308
17  * @version 1.0
18  *
19  * 08-Fév-2016 : 1.0 - Version initiale.
20  */
21  public abstract class Enchant extends Carte {
22
23      private final String description;
24
25      public Enchant(String desc) {
26          super();
27          description = desc;
28      }
29
30      /**
31       * Déclaration de la fonction placerEnchant, celle ci n'a aucun
          effet
32       *
33       * @param arme Arme sur lequel placer enchant
34       */
35      protected abstract void placerEnchant(Arme arme);
36
37
38      /**
39       * Permet d'avoir la représentation de la carte d'enchantement.
40       *
41       * @return le JJson représentant l'enchant de la carte
42       */
43      @Override
44      public JsonObject toJSON() {
45          JsonObjectBuilder obj = Json.createObjectBuilder();
46          obj.add("Id", this.getCardID());
47          obj.add("Nom", this.getClass().getCanonicalName());
48          obj.add("Description", description);
49
50          return obj.build();
51      }
52
53
54
55
56  }

```

1.5 EnchantStase.Java

```

1  package cardgame.JeuxCartes;
2
3  /**
4   * Implémentation de la classe abstraite Enchant.
5   *
6   * EnchantStase permet de placer l'effet de stase sur 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 EnchantStase extends Enchant {
15
16      public EnchantStase() {
17          super("Cette carte applique un effet de Stase sur l'arme
18              choisi.");
19      }
20
21      /**
22       * Modifie l'arme et la met en stase.
23       * @param arme arme qui va être mise sous stase.
24       */
25      @Override
26      protected void placerEnchant(Arme arme) {
27          arme.estStase = true;
28          arme.reset();
29      }
30  }

```

1.6 EnchantNeutre.Java

```

1  package cardgame.JeuxCartes;
2
3  import cardgame.Regles.TypeArme;
4
5  /**
6   * Implémentation de la classe abstraite Enchant.
7   *
8   * EnchantNeutre permet de changer le type de l'arme.
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 EnchantNeutre extends Enchant {
17
18      public EnchantNeutre() {
19          super("Cette carte rend cette arme neutre.");
20      }
21
22      /**
23       * Modifie l'arme sur lequel le triangle des dégâts ne sera plus
24       * appliqué.
25       * @param arme arme dont le triangle de dégât va être retiré
26       */
27      @Override
28      protected void placerEnchant(Arme arme) {
29          arme.type = TypeArme.Neutre;
30      }
31  }

```

1.7 EnchantFacile.Java

```

1  package cardgame.JeuxCartes;
2
3
4  /**
5   * Implémentation de la classe abstraite Enchant.
6   *
7   * EnchantFacile permet de changer les utilisateurs possibles de
8   * l'arme.

```

```

 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
18              1.");
19      }
20
21      /**
22       * Applique l'enchantement sur l'arme passé en paramètre.
23       *
24       * @param arme arme dont les degats vont etre diminué
25       */
26      @Override
27      protected void placerEnchant(Arme arme) {
28          arme.degat--;
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
 8   * au xperso et Joueurs.
 9   * (L'idée provient de l'équipe Philippe Pépos PetitClerc et Mehdi
10   * Ait Younes)
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 interface Cible {
18
19      /**
20       * @return True si la cible peut actuellement être attaqué.
21       * (Ex un Joueur doit avoir un jeu vide.)
22       */
23      public abstract boolean peutEtreAttaque();
24
25      /**
26       * recoitAttaque applique le dommage reçu, vérifie si le coup é
27       * tait
28       * fatal et retourne le résultat du coup.
29       * @param attaqueur Le combattant qui attaque la cible.
30       * @return Le résultat du coup reçu (Soit AttackPerso ou
31       * AttackJoueur)
32       */
33      public abstract Resultat recoitAttaque(Combattant attaqueur);
34
35      /**
36       * @return True si la cible est morte.
37       */
38      public abstract boolean estMort();
39  }

```

1.11 Soigneur.Java

```

 1  package cardgame.JeuxCartes;
 2

```

```

3  import cardgame.ResultUtils.SoinsResult;
4
5  /**
6   * Interface utilisé pour les fonctions reliés aux soins.
7   * Cette interface nous permet de découpler la logique des sortilèges
      de soins
8   * des Personnages, ce qui nous permettra d'ajouter de nouveaux
      métiers axés sur
9   * le support et les effets de status (Haste,Vitality etc...)
10  * (L'idée provient de l'équipe Philippe Pépos PetitClerc et Mehdi
      Ait Younes)
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 interface Soigneur {
18
19      /**
20       * Permet au perso de soigner un allié.
21       * Pour soigner, le perso a besoin d'avoir encore des points de
          magie.
22       *
23       * @param p Personnage allié à soigner.
24       * @return un SoinsResult si le soin à réussi, RefuseResult sinon.
25       */
26      public abstract SoinsResult soigner (Perso p);
27
28      /**
29       * Vérifie si le soigneur est capable de faire le sort de soins.
30       * @param p Le perso à soigner.
31       * @return True si le sort de soins peut être effectué.
32       */
33      public abstract boolean peutSoigner(Perso p);
34
35  }

```

1.12 Combattant.Java

```

1  package cardgame.JeuxCartes;
2
3  import cardgame.Regles.TypeArme;
4  import cardgame.ResultUtils.Resultat;
5  import java.util.List;
6
7  /**
8   * Classe abstraite qui extend Perso, Combattant nous permet de
9   * d.coupler la logique d'attaque hors des perso. Ceci nous
      permettrait
10  * alors dans le futur d'ajouter des classes qui ne peuvent attaquer,
11  * tel des troubadour ou Bardes.
12  *
13  * (Inspiré par les travaux de Phillipe Pépos PetitClerc et Zerrouk
      Rahdia.)
14  *
15  * @author Mathieu Gravel GRAM02099206
16  * @author Nicolas Reynaud REYN23119308
17  * @version 1.2
18  *
19  * Historique :
20  * 14-Fév-2016 : 1.0 - Version initiale
21  */
22  public abstract class Combattant extends Perso {
23
24      public Combattant(int _hp, int _mp, List<TypeArme> armes) {
25          super(_hp, _mp, armes);
26      }
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,
20              Arrays.asList(TypeArme.Contondant, TypeArme.Neutre));
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(),
29              allie.getCardID());
30          return resultat;
31      }
32
33      @Override
34      public boolean peutSoigner(Perso p) {
35          return this.getMp() > 0 && (this.getCardID() !=
36              p.getCardID());
37      }
38
39      @Override
40      public JsonObject toJSON() {
41          JsonObject json = super.toJSON();
42          JsonObjectBuilder addition = Json.createObjectBuilder();
43          addition.add("Type Personnage", "Pretre");
44          addition.add("General Info", json);
45          return addition.build();
46      }
47  }

```

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,
21              Arrays.asList(TypeArme.values()));
22      }
23
24      @Override
25      public SoinsResult soigner(Perso allie) {

```

```

25         SoinsResult resultat;
26         this.utiliserMagie();
27         allie.recevoirSoins();
28         resultat = new SoinsResult(true, this.getCardID(),
29             allie.getCardID());
30         return resultat;
31     }
32     @Override
33     public boolean peutSoigner(Perso p) {
34         return this.getMp() > 0 && (this.getCardID() !=
35             p.getCardID());
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
13   * classe
14   * permet d'initialiser le deck et de traiter la logique de pioche et
15   * de vie
16   * (puisque les points de vie sont == au nombre de cartes restantes.)
17   * sans
18   * donner accès à cette logique au joueur.
19   *
20   * @author Mathieu Gravel GRAM02099206
21   * @author Nicolas Reynaud REYN23119308
22   * @version 1.0 08-Fév-2016 : 1.0 - Version initiale.
23   *           1.1 10-Fév-2016 : 1.1 - Modification de InitialiserDeck
24   *           pour utiliser PersoFactory.
25   */
26  public class Deck {
27
28      /**
29       * Structure représentant le deck lui-même.
30       */
31      private final List<Carte> cartespioches;
32
33      public Deck(){
34          cartespioches = new ArrayList<>();
35          initialiserDeck();
36      }
37
38      /**
39       * Permet de créer le deck et d'initialiser son contenu.
40       * La classe initialise le nombre de cartes nécessaire de chaque
41       * type
42       * selon les règles du jeu et ensuite mélange le deck.
43       */
44  }

```

```

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,
49             1));
50         cartespioches.addAll(createurArmes.creerSetArmes(Regle.CARTEARMEDEUX,
51             2));
52         cartespioches.addAll(createurEnchants.creerSetEnchants(Regle.CARTEENCHANTEMENT));
53         Collections.shuffle(cartespioches, new
54             Random(System.nanoTime()));
55     }
56     /**
57      * Fonction qui permet de vider le deck.
58      */
59     public void viderDeck(){
60         this.cartespioches.clear();
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
70            restantes,
71            le nombre demandé ou le nombre maximal dans une main.*/
72         int nbAPiocher = Math.min(nbCartes, this.carteRestantes());
73         nbAPiocher = Math.min(nbAPiocher, Regle.CARTEMAIN);
74
75         List<Carte> nouvCartes = new ArrayList<>();
76
77         while ( nbAPiocher != 0) {
78             nouvCartes.add(cartespioches.remove(0));
79             --nbAPiocher;
80         }
81
82         return nouvCartes;
83     }
84     /**
85      * Permet d'appliquer les dégats reçu 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
111     * la triche.
112     * @return la représentation du Deck en JSon
113     */
114     public JsonObject toJson() {
115         JsonObjectBuilder obj = Json.createObjectBuilder();
116         obj.add("Nombre de cartes restantes a piger",
117             cartespioches.size());
118         return obj.build();
119     }
120 }

```

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 reÃgu du
15  * controlleur.
16  * Comparé à Jeux qui sert de facade pour le controlleur, Joueur
17  * traite les
18  * appels des joueurs dans le modèle et retourne ces conséquences.
19  *
20  * @author Mathieu Gravel GRAM02099206
21  * @author Nicolas Reynaud REYN23119308
22  * @version 1.1
23  * Historique : 08-Fév-2016 : 1.0 - Version initiale.
24  *              13-Fév-2016 : 1.1 - Réécriture des fonctions pour
25  *              fonctionner avec cible.
26  */
27 public class Joueur implements Cible {
28
29     private final int idJoueur;
30     private final Deck carteDeck;
31     private final Map<Integer, Carte> main;
32     private final Map<Integer, Perso> carteEnJeu;
33     private final List<Carte> cimetiere;
34
35     public Joueur(int i) {
36         idJoueur = i;
37         carteDeck = new Deck();
38         main = new ConcurrentHashMap<>();
39         cimetiere = new CopyOnWriteArrayList<>();
40         carteEnJeu = new ConcurrentHashMap<>();
41         List<Carte> mainDeb = carteDeck.piocherCarte(Regle.CARTEMAIN);
42         for (Carte c : mainDeb) {
43             main.put(c.getCardID(), c);
44         }
45     }
46
47     /**
48     * @return L'identifiant unique du joueur.
49     */
50     public int getIdjoueur() {
51         return idJoueur;
52     }
53 }

```

```

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
74         associées au joueur
75     */
76     public Map<Integer, Perso> getCarteEnJeu() {
77         return carteEnJeu;
78     }
79
80     /**
81     * Permet d'avoir le cimetiere du joueur
82     *
83     * @return la liste de carte présente dans le cimetiere du joueur
84     */
85     public List<Carte> getCimetiere() {
86         return cimetiere;
87     }
88
89     /**
90     * Permet de savoir si le joueur à perdu
91     *
92     * @return true si le joueur à perdu (autrement dit si il n'a
93         plus de carte
94     * null part ) [Cimetiere non compris] false sinon
95     */
96     public boolean aPerdu() {
97         return (main.isEmpty() && carteEnJeu.isEmpty() &&
98             carteDeck.deckEstVide());
99     }
100
101     /**
102     * Permet au joueur de defausser une liste de Carte
103     *
104     * @param defausse liste des cartes à défausser
105     * @return Un DefausseResult si la defausse s'est bien passé Un
106     * RefusedResult sinon
107     */
108     public Resultat defausserCartes(List<Carte> defausse) {
109         Resultat res;
110
111         for (Carte c : defausse) {
112             cimetiere.add(main.remove(c.getCardID()));
113         }
114
115         res = new DefausseResult(this.getIdjoueur(), true, defausse);
116         return res;
117     }
118
119     /**
120     * Permet au joueur de piocher des cartes
121     *
122     * @return un PiocheResult si tout s'est bien passé un
123     * RefusedResult sinon
124     */

```

```

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
135      * cartes
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
139      *         RefusedResult sinon
140      */
141     public Resultat attaque(Combattant attaqueur, Cible attaque) {
142         Resultat res;
143
144         res = attaqueur.Attaque(attaque);
145
146         return res;
147     }
148
149     /**
150      * Fonction auxiliaire appelé après chaque attaque reÃgu,
151      * MAJCartesPlancher
152      * enlève les perso mort de la partie.
153      */
154     public void MAJCartesPlancher() {
155         for (Perso pers : this.carteEnJeu.values()) {
156             if (pers.estMort()) {
157                 cimetiere.add(carteEnJeu.remove(pers.getCardID()));
158             }
159         }
160
161         /**
162          * Permet d'ajouter une liste d'enchant à un joueur
163          *
164          * @param enchs liste des positions des enchants dans la main
165          * @param carteTouchee carte étant affectée par l'enchant
166          * @return une Liste de Result, chaque'un étant soit un
167          *         EnchantResult si tout
168          *         c'est bien passé sinon un RefusedResult
169          */
170         public Resultat ajouterEnchants(List<Enchant> enchs, Carte
171             carteTouchee) {
172             Resultat res;
173             Arme arm;
174             if (carteTouchee instanceof Perso) {
175                 arm = ((Perso) carteTouchee).getArme();
176             } else if (carteTouchee instanceof Arme) {
177                 arm = (Arme) carteTouchee;
178             } else {
179                 return new RefuseResult("Erreur interne.");
180             }
181
182             for (Enchant ench : enchs) {
183                 arm.ajouterEnchant(ench);
184                 cimetiere.add(ench);
185                 main.remove(ench.getCardID());
186             }
187             res = new EnchantResult(true, carteTouchee, enchs);
188             return res;
189         }
190     }

```

```

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
195     *         Refusedresult
196     * sinon
197     */
198     public Resultat placerPerso(Perso personnage, Arme arm,
199     List<Carte> ench) {
200         Resultat res;
201
202         for (Carte c : ench) {
203             Enchant en = (Enchant) c;
204             arm.ajouterEnchant(en);
205             cimetiery.add(main.remove(en.getCardID()));
206         }
207
208         personnage.equiperArme(arm);
209         main.remove(personnage.getCardID());
210         main.remove(arm.getCardID());
211         carteEnJeu.put(personnage.getCardID(), personnage);
212
213         res = new PersoDeploieResult(this.getIdjoueur(), true,
214         personnage);
215         return res;
216     }
217
218     /**
219     * @param car La carte qu'on veut vérifier l'emplacement
220     * @return True si la carte est dans la main du joueur.
221     */
222     public boolean carteDansMain(Carte car) {
223         return main.containsKey(car.getCardID());
224     }
225
226     /**
227     * @param car La carte qu'on veut vérifier l'emplacement
228     * @return True si la carte est dans la jeu du joueur.
229     */
230     public boolean carteDansJeu(Carte car) {
231         return carteEnJeu.containsKey(car.getCardID());
232     }
233
234     /**
235     * @return True si le joueur a un deck vide
236     */
237     public boolean destVide() {
238         return carteDeck.deckEstVide();
239     }
240
241     /**
242     * Permet au joueur de declarerForfait Autrement dit, de passer
243     * toutes ces
244     * cartes dans le cimetiery.
245     *
246     * @return RefusedResult si le joueur a déjà perdu
247     *         FinDePartieResult si le
248     *         joueur a déclarer forfait
249     */
250     public Resultat declarerForfait() {
251         Resultat res;
252         main.clear();
253         carteDeck.viderDeck();
254         carteEnJeu.clear();
255         res = new FinDePartieResult(this.getIdjoueur(), true, -1);
256         return res;
257     }
258
259     /**

```

```

255     * Permet d'effectu   l'action de soin sur un personnage pr  sent
        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   tre soign  
261     */
262     public Resultat soignerPerso(Soigneur soins, Perso soignee) {
263
264         if (!soins.peutSoigner(soignee)) {
265             return new RefuseResult("Le soins ne peut pas   tre   
                ffectu  .");
266         }
267
268         return soins.soigner(soignee);
269     }
270
271     /**
272     * Permet d'avoir le JSon associ      un joueur
273     *
274     * @return le JSon objet repr  sentant 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 associ   au contenu de la main du joueur
287     *
288     * @return le JSon associ      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 associ   au contenu du cimetiere du
        joueur
305     *
306     * @return le JSon associ   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) &&
335             (!carteDeck.deckEstVide()));
336     }
337
338     @Override
339     public boolean peutEtreAttaque() {
340         return carteEnJeu.isEmpty();
341     }
342
343     @Override
344     public AttaquePlayerResult recoitAttaque(Combattant attaqueur) {
345         int degat = attaqueur.forceAttaque(TypeArme.Neutre);
346         List<Carte> cartePerdus = this.carteDeck.dommageJoueur(degat);
347         for (Carte c : cartePerdus) {
348             cimetiére.add(c);
349         }
350         AttaquePlayerResult res = new AttaquePlayerResult(degat,
351             idJoueur, attaqueur.getCardID(), idJoueur, estMort());
352         return res;
353     }
354
355     @Override
356     public boolean estMort() {
357         return ((carteDeck.deckEstVide() && main.isEmpty()) &&
358             (carteEnJeu.isEmpty()));
359     }
360 }

```

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
10   * dans une
11   * partie. ArmeFactory, classe basé sur le patron Factory, nous
12   * permet de
13   * découpler et cacher la logique de création des cartes des classes
14   * reliés à
15   * l'API.
16   *
17   * @author Mathieu Gravel GRAM02099206
18   * @author Nicolas Reynaud REYN23119308
19   * @version 1.0
20   * 08-Fév-2016 : 1.0 - Version initiale.
21   */
22  public class ArmeFactory {
23
24      /**
25       * Permet de créer une liste d'arme
26       *

```

```

25      * @param nbCopies Nombre d'ecopies de chacune des types d'armes
26      * à
27      * @param degats Nombre de dégats fait par ces armes.
28      * @return Liste de nbCopie éléments contenant les armes
29      * demandées.
30      */
31      public List<Arme> creerSetArmes(int nbCopies, int degats) {
32          List<Arme> armes = new ArrayList<>();
33
34          for (int copieAct = 0; copieAct < nbCopies; copieAct++) {
35              armes.add(new Arme(TypeArme.Contondant, degats));
36              armes.add(new Arme(TypeArme.Perforant, degats));
37              armes.add(new Arme(TypeArme.Tranchant, degats));
38          }
39
40          return armes;
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
9   * d'enchantelements dans
10  * une partie. EnchantFactory, classe basé sur le patron Factory,
11  * nous permet de
12  * découpler et cacher la logique de création des cartes de 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 EnchantFactory {
21
22      /**
23       * Permet de créer un ensemble de tout les types d'enchantelements
24       * du jeu.
25       *
26       * @param nbCopies nombre de copie des cartes Enchants à
27       * instancier.
28       * @return Une liste de nbCopie éléments qui contient tout les
29       * enchants.
30       */
31      public List<Enchant> creerSetEnchants(int nbCopies) {
32          List<Enchant> enchantements = new ArrayList<>();
33
34          for (int copieAct = 0; copieAct < nbCopies; copieAct++) {
35              enchantements.add(new EnchantNeutre());
36              enchantements.add(new EnchantStase());
37              enchantements.add(new EnchantDegatPlus());
38              enchantements.add(new EnchantDegatMoins());
39              enchantements.add(new EnchantFacile());
40          }
41
42          return enchantements;
43      }
44  }

```

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
12  *   cartes
13  * personnages. La classe est écrite de telle manière que tout ajout
14  *   de nouveaux
15  * personnages dans le jeu nécessite seulement d'être ajouté comme
16  *   fonction dans
17  * cette classe ainsi que d'inscrire ces règles dans la classe Règle.
18  *
19  * @author Mathieu Gravel GRAM02099206
20  * @author Nicolas Reynaud REYN23119308
21  * @version 1.1
22  *
23  * Historique : 8 Fév-2016 : 1.0 Version initiale.
24  * 13Fév-2016 : 1.1 Modification du code pour marcher avec les
25  * nouvelles classes Guerrier, Pretre et Paladin.
26  */
27 public class PersoFactory {
28
29     /**
30      * Retourne une liste de cartes Guerriers
31      *
32      * @param nbCopies nombre de copie de cartes guerriers
33      * @return liste de nbCopies d'instance de Guerriers.
34      */
35     public List<Perso> creerSetGuerrier(int nbCopies) {
36         List<Perso> guerriers = new ArrayList<>();
37
38         for (int copieAct = 0; copieAct < nbCopies; copieAct++) {
39             guerriers.add(new Guerrier());
40         }
41
42         return guerriers;
43     }
44
45     /**
46      * Retourne une liste de cartes Prêtres.
47      *
48      * @param nbCopies nombre de copie de cartes prêtres.
49      * @return liste de nbCopies d'instance de Prêtres.
50      */
51     public List<Perso> creerSetPretre(int nbCopies) {
52         List<Perso> pretres = new ArrayList<>();
53
54         for (int copieAct = 0; copieAct < nbCopies; copieAct++) {
55             pretres.add(new Pretre());
56         }
57
58         return pretres;
59     }
60
61     /**
62      * Retourne une liste de cartes Paladins.
63      *
64      * @param nbCopies nombre de copie de cartes paladins.
65      * @return liste de nbCopies d'instance de Paladins.
66      */
67     public List<Perso> creerSetPaladin(int nbCopies) {
68         List<Perso> paladins = new ArrayList<>();
69
70         for (int copieAct = 0; copieAct < nbCopies; copieAct++) {
71             paladins.add(new Paladin());
72         }
73     }
74 }

```



```

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 PRETREMHP = 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
5   * possibles dans
6   * le jeu, ainsi que leurs logiques personnelles, tel que leurs
7   * forces/faiblesses. La classe détient aussi la
8   * fonction de calcul du triangle d'attaque pour fins d'évolutions
9   * faciles.
10  *
11  * @author Mathieu Gravel GRAM02099206
12  * @author Nicolas Reynaud REYN23119308
13  * @version 1.0
14  *
15  * Historique :
16  *
17  * -8 Fév-2016 : 1.0 Version initiale.
18  */
19  public enum TypeArme {

```

```

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
44       *         ennemi, -1 si
45       *         l'arme ennemi est la faiblesse de l'arme actuelle, Sinon 0.
46       */
47      public int calculModificateur(TypeArme armeEnnemi) {
48          if (this.force.equals(armeEnnemi.nom)) {
49              return 1;
50          } else if (this.faiblesse.equals(armeEnnemi.nom)) {
51              return -1;
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
      * contrôleur. 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 faire une 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 effectuée 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 contrôleur si nécessaire.
38  * 13-Fév-2016 : 1.2 - Ajout de fonctions de validation de coup pour
    les contrôleurs
39  * (Remerciements : L'idée est inspirée 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
56              joueurList.get(idJoueur).getCarteDeck().carteRestantes();
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
106             pour quelqu'un d'autre.");
107     }
108 }
109
110 /**
111  * Permet de savoir l'état du jeu à tout moment ( sous un format
112  *   Json)
113  *
114  * @return le contenu de chaque joueur
115  */
116 public JsonObject getEtatJeu() {
117     JsonObjectBuilder obj = Json.createObjectBuilder();
118     obj.add("aQuiLeTour", this.joueurTour);
119     obj.add("partieEnCours", this.partieEnMarche);
120
121     Iterator<Joueur> j = joueurList.iterator();
122     int numJoueur = 1;
123     while (j.hasNext()) {
124         obj.add("Joueur #" + numJoueur, j.next().toJSON());
125         ++numJoueur;
126     }
127
128     return obj.build();
129 }
130
131 /**
132  * Permet de savoir si la partie est finie
133  *
134  * @return true si la partie est fini, autrement dit si un joueur
135  *   à gagné
136  * false sinon
137  */
138 public boolean partieFini() {
139     return getJoueurGagnant() != -1; // tout les joueurs ont
140     perdu sauf 1
141 }
142
143 /**
144  * Permet d'avoir l'id du joueur ayant gagné
145  *
146  * @return l'id du joueur gagnant -1 si aucun joueur n'a gagné
147  */
148 public int getJoueurGagnant() {
149     int joueursPerdants = 0;
150     int joueurGagnant = 0;
151     for (int i = 0; i < joueurList.size(); i++) {
152         if (!joueurList.get(i).aPerdu()) {
153             joueurGagnant = i;
154         } else {
155             joueursPerdants++;
156         }
157     }
158 }

```

```

155         return joueursPerdants == joueurList.size() -1? joueurGagnant
156             : -1;
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
163     *         la pioche
164     * un RefusedResult sinon
165     */
166     public Resultat piocherCartes(int idJoueur) {
167         Resultat res;
168         if (idJoueur != aQuiLeTour()) {
169             res = new RefuseResult("Ce n'est pas le tour de ce
170                 joueur.");
171             return res;
172         }
173         res = joueurList.get(idJoueur).piocher();
174         return res;
175     }
176     public boolean peutPiocherCartes(int idJoueur) {
177         return idJoueur == joueurTour &&
178             joueurList.get(idJoueur).peutPiocher();
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
185     *         la carte
186     * attaquante
187     * @param attaqueur carte attaquant la seconde carte
188     * @param receveur carte recevant le coup
189     * @return un AttackResult si tout c'est bien passé un
190     *         refusedResult sinon
191     */
192     public Resultat attaquePerso(int idJoueur, int idAdversaire,
193         Carte attaqueur, Carte receveur) {
194         Resultat res;
195         if (attaquePersoValide(idJoueur, idAdversaire, attaqueur,
196             receveur)) {
197             Combattant att = (Combattant) attaqueur;
198             Cible attaquee = (Cible) receveur;
199             res = joueurList.get(idJoueur).attaque(att, attaquee);
200             joueurList.get(idAdversaire).MAJCartesPlancher();
201             this.prochainJoueur();
202         } else {
203             res = new RefuseResult("L'attaque n'est pas possible");
204         }
205         return res;
206     }
207     public boolean attaquePersoValide(int idJoueur, int idAdversaire,
208         Carte attaqueur, Carte receveur) {
209         boolean coupP = ((this.aQuiLeTour() == idJoueur) && (idJoueur
210             != idAdversaire));
211         coupP = coupP && (idAdversaire >= 0) && (joueurList.size() >=
212             idAdversaire);
213         coupP = coupP && (attaqueur instanceof Combattant) &&
214             (receveur instanceof Cible);
215         if (coupP) {
216             coupP = coupP &&
217                 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
225  * la carte
226  * @param attaquante
227  * @param attaquateur Carte effectuant l'attaque sur le joueur
228  * @return un AttackResult si tout c'est bien passé un
229  * refusedResult sinon
230 */
231 public Resultat attaqueJoueur(int idJoueur, int idAdversaire,
232     Carte attaquateur) {
233     Resultat res;
234
235     if (attaqueJoueurValide(idJoueur, idAdversaire, attaquateur)) {
236         Combattant att = (Combattant) attaquateur;
237         res = joueurList.get(idJoueur).attaque(att,
238             joueurList.get(idAdversaire));
239         this.prochainJoueur();
240     } else {
241         res = new RefuseResult("L'attaque n'est pas possible");
242     }
243
244     return res;
245 }
246
247 public boolean attaqueJoueurValide(int idJoueur, int
248     idAdversaire, Carte attaquateur) {
249
250     boolean verifValide = idJoueur == aQuiLeTour() &&
251         idAdversaire >= 0 && idAdversaire <= joueurList.size() &&
252         idJoueur != idAdversaire;
253
254     if (verifValide) {
255         verifValide =
256             joueurList.get(idJoueur).getCarteEnJeu().containsKey(attaquateur.getCardID());
257         verifValide = verifValide &&
258             joueurList.get(idAdversaire).peutEtreAttaque();
259         verifValide = verifValide && (attaquateur instanceof
260             Combattant);
261     }
262
263     return verifValide;
264 }
265
266 /**
267  * Permet d'ajouter une liste d'enchant sur un joueur
268  *
269  * @param idJoueur id du joueur qui va recevoir les enchants
270  * @param carteTouche La carte qui va recevoir les enchants
271  * @param enchant listes des positions relatives dans le deck des
272  * cartes
273  * d'enchantelements à appliquer
274  * @return un list Result, chaque'un contenant un EnchantResult si
275  * l'enchant
276  * fonctionne un refused Result sinon
277 */
278 public Resultat ajouterEnchantements(int idJoueur, Carte
279     carteTouche, List<Carte> enchant) {
280     Resultat res;
281
282     if (peutAjouterEnchantements(idJoueur, carteTouche, enchant))
283     {
284         List<Enchant> enchs = new ArrayList<>();

```

```

270         for (Carte c : enchant) {
271             enchs.add((Enchant) c);
272         }
273         res = joueurList.get(idJoueur).ajouterEnchants(enchs,
274             carteTouche);
275         this.prochainJoueur();
276     } else {
277         res = new RefuseResult("Vous ne pouvez pas enchanter
278             cette cartes avec votre sélection.");
279     }
280     return res;
281 }
282
283 public boolean peutAjouterEnchantements(int idJoueur, Carte
284     carteTouche, List<Carte> enchants) {
285     boolean verifValide = idJoueur == aQuiLeTour();
286     Joueur j = joueurList.get(idJoueur);
287     boolean carteDeploye = false;
288     for (Carte c : enchants) {
289         verifValide = verifValide && j.carteDansMain(c);
290     }
291     for (Joueur joueurAct : joueurList) {
292         carteDeploye = carteDeploye ||
293             joueurAct.carteDansJeu(carteTouche);
294     }
295     carteDeploye = carteDeploye && carteTouche instanceof Perso;
296     if (carteDeploye) {
297         Perso p = (Perso) carteTouche;
298         carteDeploye = p.getArme().peutAjouterEnchantement();
299     }
300     return verifValide && carteDeploye;
301 }
302
303 public Collection<Carte> getCartesMainJoueur(int idJoueur) {
304     return joueurList.get(idJoueur).getMain().values();
305 }
306
307 public Collection<Perso> getCartesJeuJoueur(int idJoueur) {
308     Perso p;
309     return joueurList.get(idJoueur).getCarteEnJeu().values();
310 }
311
312 public Collection<Carte> getCimetiereJoueur(int idJoueur) {
313     return joueurList.get(idJoueur).getCimetiere();
314 }
315
316 /**
317  * Permet de placer un personnage
318  *
319  * @param idJoueur id relatif du joueur dans la liste Joueur List
320  * @param personnage position relative dans la main du joueur à
321  *     placer
322  * @param arme position relative dans la main de l'arme a placer
323  * @param enchants liste des cartes d'enchants à placer sur le
324  *     perso
325  * @return un PersoDeployeResult si tout ce passe bien un
326  *     refusedResult
327  * sinon
328  */
329 public Resultat placerPerso(int idJoueur, Carte personnage, Carte
330     arme, List<Carte> enchants) {
331     Resultat res;
332     if (this.peutDeployerPerso(idJoueur, personnage, arme,
333         enchants)) {
334         res = joueurList.get(idJoueur).placerPerso((Perso)
335             personnage, (Arme) arme, enchants);
336         this.prochainJoueur();
337     } else {
338         res = new RefuseResult("Vous n'avez pas le droit
339             d'utiliser une des cartes choisi.");
340     }
341 }

```

```

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  * Permet de défausser n Carte à partie de l'api
359  *
360  * @param idJoueur id du joueur souhaitant se défausser
361  * @param defausse Liste des positions relatives dans la main des
362  *   cartes à
363  *   défausser
364  * @return un defausseResult si tout ce passe bien un
365  *   refusedResult en cas
366  *   d'erreur
367  */
368 public Resultat defausserCartes(int idJoueur, List<Carte>
    defausse) {
369     Resultat res;
370     if (peutDefausserCartes(idJoueur, defausse)) {
371         res = joueurList.get(idJoueur).defausserCartes(defausse);
372         this.prochainJoueur();
373     } else {
374         res = new RefuseResult("La liste est soit vide, soit
            rempli de cartes pas situés dans la main.");
375     }
376
377     return res;
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
402     * terrain de
403     * jeu
404     * @param soignee id relatif de la position de la carte sur le
405     * terrain de
406     * jeu ( carte à soigner)
407     * @return un SoinsResult si tout c'est bien passé un
408     * RefusedResult sinon
409     */
410     public Resultat soignerPerso(int idJoueur, Carte soigneur, Carte
411     soignee) {
412         Resultat res;
413
414         if (peutSoignerPerso(idJoueur, soigneur, soignee)) {
415             Soigneur s = (Soigneur) soigneur;
416             Perso p = (Perso) soignee;
417             res = joueurList.get(idJoueur).soignerPerso(s, p);
418             this.prochainJoueur();
419         } else {
420             res = new RefuseResult("Vous n'avez pas le droit de faire
421             ce soin.");
422         }
423
424         return res;
425     }
426
427     public boolean peutSoignerPerso(int idJoueur, Carte soigneur,
428     Carte soignee) {
429         boolean coupValide = idJoueur == aQuiLeTour();
430         coupValide = coupValide && soigneur != soignee;
431         Joueur joueurAct = joueurList.get(idJoueur);
432         coupValide = coupValide && joueurAct.carteDansJeu(soigneur)
433         && joueurAct.carteDansJeu(soignee);
434         coupValide = coupValide && soigneur instanceof Soigneur &&
435         soignee instanceof Perso;
436         if (coupValide) {
437             Soigneur s = (Soigneur) soigneur;
438             Perso p = (Perso) soignee;
439             coupValide = s.peutSoigner(p);
440         }
441         return coupValide;
442     }
443
444     /**
445     * Permet de liberer les informations de jeu et reinialisé les
446     * valeurs de
447     * l'api
448     */
449     public void finPartie() {
450         joueurList.clear();
451         joueurTour = 0;
452         partieEnMarche = false;
453     }
454 }

```

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,
35                               boolean attaqueTue) {
36         dommageRecu = dmg;
37         idCarte = carteId;
38         attaqueTuer = attaqueTue;
39         idCarteAttack = persoCoupId;
40         desc = "L'attaque de la carte " + carteId + "sur " +
                persoCoupId + "a causé " + dmg + "dégats.\n";
41         if (attaqueTue) {
42             desc = desc + "Le perso à été tué.";
43         }
44     }
45     /**
46      * Getter
47      *
48      * @return le dommage reçu 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 reÃ§u l'attaque ou -1
95      * si le joueur
96      * a pris le coup.
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
106      * sinon.
107      */
108     public boolean attaqueATuer() {
109         return attaqueTuer;
110     }
111
112     /**
113      * Setter
114      *
115      * @param joueurId l'identifiant du joueur qui a fait le coup.
116      */
117     @Override
118     public void setJoueur(int joueurId) {
119         attaqueur = joueurId;
120     }

```

5.3 AttaquePlayerResult.Java

```

1  package cardgame.ResultUtils;
2
3  /**
4   * Implémentation de Resultat pour décrire les conséquences d'une
5   * attaque
6   * perso-joueur.
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 AttaquePlayerResult implements Resultat {
15
16      private final int dommageRecu;
17      private int attaqueur;
18      private final int idCarte;
19      private final int idAdversaire;
20      private final boolean attaqueTuer;
21      private String desc;
22
23      public AttaquePlayerResult(int dmg, int joueurId, int carteId,
24                                int adversaireId, boolean attaqueTue) {
25          dommageRecu = dmg;
26          attaqueur = joueurId;
27          idCarte = carteId;
28          attaqueTuer = attaqueTue;
29          idAdversaire = adversaireId;
30          desc = "L'attaque de la carte " + carteId + "sur le joueur" +
31                idAdversaire + "a causé " + dmg + "dégats.\n";
32          if (attaqueTue)
33              desc = desc + "Le joueur à été tué.";
34      }
35
36      public AttaquePlayerResult(int dmg, int carteId, int
37                                adversaireId, boolean attaqueTue) {
38          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 reçu 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
92     *         sinon.
93     */
94     public boolean attaqueATuer() {
95         return attaqueTuer;
96     }
97
98     /**
99     * Setter
100     * @param joueurId l'identifiant du joueur qui a fait le coup.
101     */
102     @Override
103     public void setJoueur(int joueurId) {
104         attaqueur = joueurId;

```

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
10   * action de
11   * défausse.
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,
27          List<Carte> cartes) {
28          cartesId = new ArrayList<>();
29          cartesJSON = new ArrayList<>();
30          String cartesStr = "";
31          for (Carte cartePioche : cartes) {
32              cartesId.add(cartePioche.getCardID());
33              cartesJSON.add(cartePioche.toJSON());
34              cartesStr = cartesStr + cartePioche.toJSON().toString() +
35                  "\n";
36          }
37          joueurId = idJoueur;
38          description = "Les cartes suivantes ont été défaussés : " +
39              cartesStr;
40          coupAFonctionne = coupCorrect;
41      }
42
43      /**
44       * @return True si l'action a fonctionné, false sinon.
45       */
46      @Override
47      public boolean coupAMarcher() {
48          return coupAFonctionne;
49      }
50
51      /**
52       * Getter
53       *
54       * @return Description de ce type de coup.
55       */
56      @Override
57      public String getDescription() {
58          return description;
59      }
60
61      /**
62       * Getter
63       *
64       * @return l'identifiant du joueur qui a joué de coup.
65       */

```

```

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
72      *         vue qui
73      *         l'utilise en tandem avec le Json de la partie.)
74      */
75     public List<Integer> getCartesID() {
76         return cartesId;
77     }
78
79     /**
80      *
81      * @return La représentation JSON des cartes défaussés. (Utile
82      *         pour une vue
83      *         qui veut faire des animations.)
84      */
85     public List<JsonObject> getCartesJSON() {
86         return cartesJSON;
87     }
88
89     /**
90      * Setter
91      *
92      * @param idJoueur l'identifiant du joueur qui a fait le coup.
93      */
94     @Override
95     public void setJoueur(int idJoueur) {
96         this.joueurId = idJoueur;
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
10   * action
11   * d'enchantement.
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
28                          carteEnch, List<Enchant> enchants) {
29          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() +
35             "a reçu les enchantements suivants :" + enchStr;
36     }
37     public EnchantResult(int jId, boolean coupCorrect, Carte
38         carteEnch, List<Enchant> enchants) {
39         joueurId = jId;
40         coupAFonctionne = coupCorrect;
41         carteEnchant = carteEnch;
42         enchant = enchants;
43         description = "Enchantement d'une carte";
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

```



```

3  /**
4   * Implémentation de Resultat pour décrire la conséquence d'un
      forfait.
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 ForfaitResult implements Resultat {
14
15      private final String description;
16      private int joueurId;
17      private final boolean coupAFonctionne;
18      private final int joueurPerdu;
19
20      public ForfaitResult(int jId, boolean coupCorrect, int
          idJoueurPerdu) {
21          joueurId = jId;
22          coupAFonctionne = coupCorrect;
23          joueurPerdu = idJoueurPerdu;
24          description = "Le joueur" + idJoueurPerdu + "vient de perdre
              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 getJoueurQuiAPerd() {
60          return joueurPerdu;
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  }

```

5.7 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
7   * déploiement d'un
8   * perso et de son arme sur le jeu.
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
23          perso) {
24          joueurId = jId;
25          coupAFonctionne = coupCorrect;
26          description = "Le joueur " + jId + "vient de déployer sur le
27              jeu : " + perso.toJSON();
28      }
29
30      /**
31       * @return True si l'action a fonctionné, false sinon.
32       */
33      @Override
34      public boolean coupAMarcher() {
35          return coupAFonctionne;
36      }
37
38      /**
39       * Getter
40       *
41       * @return Description de ce type de coup.
42       */
43      @Override
44      public String getDescription() {
45          return description;
46      }
47
48      /**
49       * Getter
50       *
51       * @return l'identifiant du joueur qui a joué de coup.
52       */
53      @Override
54      public int coupJouerPar() {
55          return joueurId;
56      }
57
58      /**
59       * Setter
60       *
61       * @param idJoueur l'identifiant du joueur qui a fait le coup.
62       */

```

```

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
10   * pioche.
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>
27          cartes) {
28          joueurId = jId;
29          coupAFonctionne = coupCorrect;
30          cartesId = new ArrayList<>();
31          cartesJSON = new ArrayList<>();
32          String cStr = "";
33          for (Carte cartePioche : cartes) {
34              cartesId.add(cartePioche.getCardID());
35              cartesJSON.add(cartePioche.toJSON());
36              cStr = cStr + cartePioche.toJSON().toString();
37          }
38          description = "Le joueur " + jId + "vient de piocher les
39              cartes suivantes : \n" + cStr;
40
41      /**
42       * Getter
43       *
44       * @return les identifiant des cartes pigés. (Utile pour une vue
45       * qui
46       * l'utilise en tandem avec le Json de la partie.)
47       */
48      public List<Integer> getCartesID() {
49          return cartesId;
50      }
51
52      /**
53       * Getter
54       *
55       * @return La représentation JSON des cartes pigés. (Utile pour
56       * une vue qui
57       * veut faire des animations.)
58       */
59      public List<JsonObject> getCartesJSON() {
60          return cartesJSON;
61      }
62

```

```

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
5   * refusé
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
5   * sortilège de
6   * soins.
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
23          cId) {
24          description = "Le personnage " + hId + " vient de soigner " +
25              cId + ".";
26          joueurId = jId;
27          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 " +
32             cId + ".";
33         coupAFonctionne = coupCorrect;
34         healerId = hId;
35         persoSoigneeId = cId;
36     }
37
38     /**
39     * Getter
40     * @return L'identifiant du perso qui a fait le sort de soins.
41     */
42     public int getHealerId() {
43         return healerId;
44     }
45
46     /**
47     * Getter
48     * @return L'identifiant du perso soignée.
49     */
50     public int getCarteSoigneeId() {
51         return persoSoigneeId;
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     * @return Description de ce type de coup.
65     */
66     @Override
67     public String getDescription() {
68         return description;
69     }
70
71     /**
72     * Getter
73     * @return l'identifiant du joueur qui a joué de coup.
74     */
75     @Override
76     public int coupJouerPar() {
77         return joueurId;
78     }
79
80
81     /**
82     * Setter
83     * @param idJoueur l'identifiant du joueur qui a fait le coup.
84     */
85     @Override
86     public void setJoueur(int idJoueur) {
87         this.joueurId = idJoueur;
88     }
89
90
91 }
```