Projet SQL

Power Mangeurs

BADOT BERTRAND Corentin
DANDOY Corentin

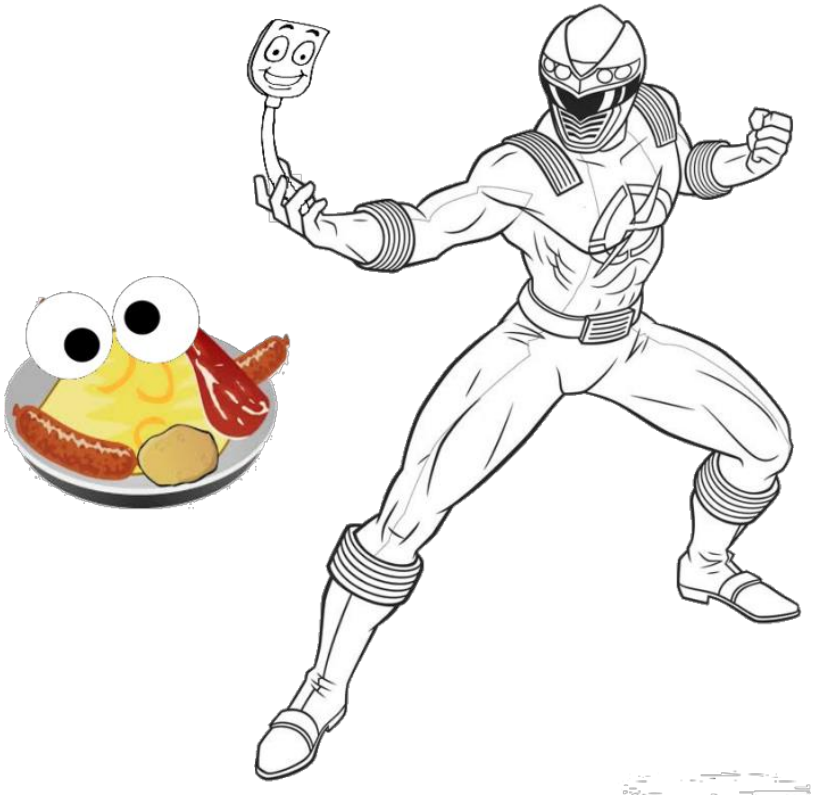


Table des matières

Introduction

Dans le cadre du cours de SQL de deuxième année, nous avons été amenés a développer une application à la fin du premier semestre. Cette application permet de gérer les combats de valeureux ninjas, les Power Mangeurs, contre des infâmes créatures, les Monstro-Nourriture. Chaque Power Mangeur pourra disposer d'un terminal à emporter qui permet de gérer le déroulement d'un combat ainsi que d'avoir des statistiques sur ses combats passés. L'application permettra aussi à l'ermite Tokiro, celui qui dirige les Power Mangeurs, de pouvoir avoir un aperçu de tous les Power Mangeurs.

Le but de ce projet est de pouvoir mettre en pratique les concepts vus en cours de SQL. En partant d'un énoncé, nous avons dû développer la structure de la base de données de notre application. Une fois la structure terminée, les différentes tables ont été développées et des contrôles supplémentaires pour garantir la cohérence des données ont été mis en place. En partant de cette base, nous avons pu développer l'application du côté serveur en utilisant des concepts tels que les vues, les procédures et les triggers. Une fois le côté serveur terminé, nous avons développé le côté client en Java en utilisant à chaque fois les capacités du serveur au maximum.

Ce document présente d'abord quelques clarifications que nous avons effectuées sur l'énoncé du projet ainsi qu'un schéma qui réprésente la structure de notre base de données PostgreSQL. Après, la partie SQL du projet sera traitée en commençant par le code SQL de tous les composants de la base de données, un script introduisant des données valides et un autre introduisant des données invalides pour tester les contraintes des tables. Et pour finir viendra le code source de l'application, c'est-à-dire, le code Java du terminal Tokiro et Power Mangeur.

Clarifications

L'énoncé contenait quelques points vagues que nous avons clarifiés après réflexion. Ces clarifications nous ont permis de mieux concevoir l'application, aussi bien au niveau du serveur SQL qu'au niveau des clients Java.

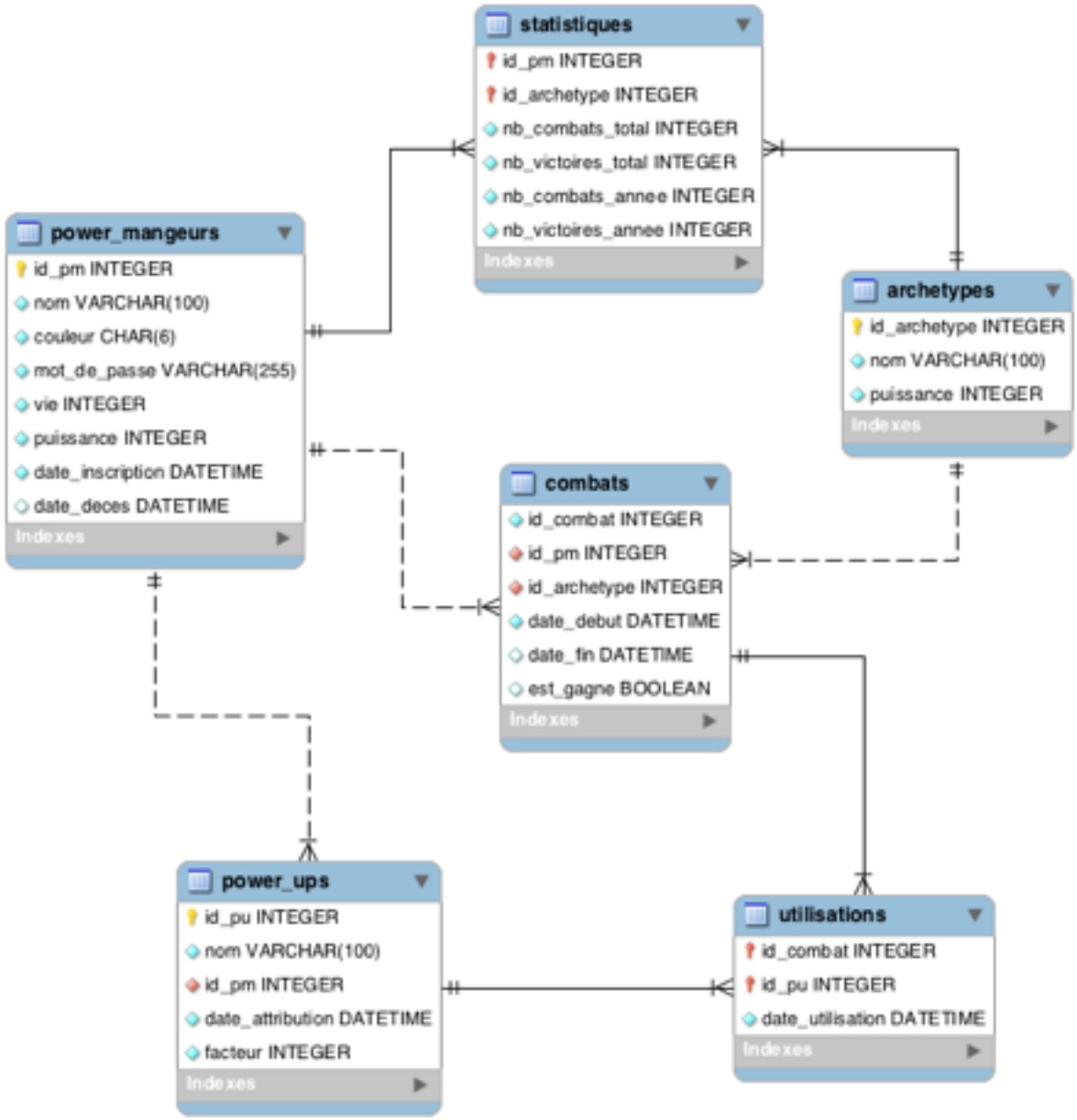
- Lors d'un combat, un archétype devait être selectionné au hasard parmi ceux qui étaient disponibles. Le hasard n'a pas été implémenté en Java grace à la classe Random, mais bien directement au niveau serveur avec une vue.

- Il n'était pas précisé directement quelles actions entreprendre dans le cas ou un combat était cloturé alors que le Power Mangeur était de puissance égale avec le monstro-nourriture. Nous avons décidé que le monstro-nourriture gagnerait en cas d'égalité.

- Pour éviter que des Power Mangeurs puissent être inscrits avec des noms trop courts, la taille minimale pour le nom devra être de 3 caractères.

- En ce qui concerne les extensions pour le terminal des Power Mangeurs, deux extensions ont été développées: l'une permettant de visualiser tous les Power Ups utilisés et l'autre offre une fonctionalité de 'Jackpot' permettant de gagner une vie grâce à un jeu de hasard.

Scripts SQL



Schéma

```sql
CREATE SCHEMA projet;
```

Séquences

```sql
CREATE SEQUENCE projet.id_power_mangeur;
CREATE SEQUENCE projet.id_archetype;
CREATE SEQUENCE projet.id_combat;
CREATE SEQUENCE projet.id_power_up;
```

Tables

```sql
CREATE TABLE projet.power_mangeurs
(
  id_pm            INTEGER PRIMARY KEY   DEFAULT
NEXTVAL('projet.id_power_mangeur'),
  nom              VARCHAR(100) NOT NULL UNIQUE CHECK (length(btrim(nom)) >=
3),
  couleur          CHAR(6)      NOT NULL UNIQUE,
  mot_de_passe     VARCHAR(150) NOT NULL CHECK (mot_de_passe <> ''),
  vie              INTEGER      NOT NULL DEFAULT 10 CHECK (vie >= 0 AND vie <=
10),
  puissance        INTEGER      NOT NULL DEFAULT 30 CHECK (puissance >= 30),
  date_inscription TIMESTAMP    NOT NULL DEFAULT LOCALTIMESTAMP,
  date_deces       TIMESTAMP,
  CHECK (date_inscription < date_deces)
);

CREATE TABLE projet.archetypes
(
  id_archetype INTEGER PRIMARY KEY DEFAULT NEXTVAL('projet.id_archetype'),
  nom          VARCHAR(100) NOT NULL UNIQUE CHECK (nom <> ''),
  puissance    INTEGER      NOT NULL CHECK (puissance >= 0)
);

CREATE TABLE projet.combats
(
  id_combat    INTEGER PRIMARY KEY DEFAULT NEXTVAL('projet.id_combat'),
  id_pm        INTEGER   NOT NULL REFERENCES projet.power_mangeurs (id_pm),
  id_archetype INTEGER   NOT NULL REFERENCES projet.archetypes (id_archetype),
  date_debut   TIMESTAMP NOT NULL  DEFAULT LOCALTIMESTAMP CHECK (date_debut <=
LOCALTIMESTAMP),
  date_fin     TIMESTAMP,
  est_gagne    BOOLEAN,
  CHECK (date_debut < date_fin)
);

CREATE TABLE projet.power_ups
(
  id_pu            INTEGER PRIMARY KEY   DEFAULT NEXTVAL('projet.id_power_up'),
  nom              VARCHAR(100) NOT NULL CHECK (nom <> ''),
  id_pm            INTEGER      NOT NULL REFERENCES projet.power_mangeurs
(id_pm),
  date_attribution TIMESTAMP    NOT NULL DEFAULT LOCALTIMESTAMP CHECK
(date_attribution <= LOCALTIMESTAMP),
  facteur          INTEGER      NOT NULL CHECK (facteur > 0),
  UNIQUE (nom, id_pm)
);

CREATE TABLE projet.utilisations
(
  id_combat        INTEGER   NOT NULL REFERENCES projet.combats (id_combat),
  id_pu            INTEGER   NOT NULL REFERENCES projet.power_ups (id_pu),
  date_utilisation TIMESTAMP NOT NULL DEFAULT LOCALTIMESTAMP CHECK
(date_utilisation <= LOCALTIMESTAMP),
  PRIMARY KEY (id_combat, id_pu)
);

CREATE TABLE projet.statistiques
(
  id_pm              INTEGER NOT NULL REFERENCES projet.power_mangeurs (id_pm),
  id_archetype       INTEGER NOT NULL REFERENCES projet.archetypes
```

```
(id_archetype),
  nb_combats_total   INTEGER NOT NULL DEFAULT 0,
  nb_victoires_total INTEGER NOT NULL DEFAULT 0 CHECK (nb_victoires_total >=
0),
  nb_combats_annee   INTEGER NOT NULL DEFAULT 0,
  nb_victoires_annee INTEGER NOT NULL DEFAULT 0 CHECK (nb_victoires_annee >=
0),
  PRIMARY KEY (id_pm, id_archetype),
  CHECK (nb_victoires_total <= nb_combats_total),
  CHECK (nb_victoires_annee <= nb_combats_annee),
  CHECK (nb_combats_annee <= nb_combats_total),
  CHECK (nb_victoires_annee <= nb_victoires_total)
);
```

Vues

```
-- Historique des combats d'un PM

CREATE VIEW projet.historique_combats (nom_pm, nom_archetype, date_debut,
date_fin, est_gagne) AS
  SELECT
    pm.nom AS "nom_pm",
    a.nom  AS "nom_archetype",
    c.date_debut,
    c.date_fin,
    c.est_gagne
  FROM projet.combats c
    INNER JOIN projet.power_mangeurs pm ON c.id_pm = pm.id_pm
    INNER JOIN projet.archetypes a ON c.id_archetype = a.id_archetype
  ORDER BY c.date_debut ASC;

-- Liste des PM décédés sur l'année

CREATE VIEW projet.liste_decedes AS
  SELECT
    pm.nom,
    pm.date_deces
  FROM projet.power_mangeurs pm
  WHERE pm.date_deces IS NOT NULL
        AND EXTRACT(YEAR FROM pm.date_deces) = EXTRACT(YEAR FROM NOW());

-- Tire un monstro-nourriture au hasard

CREATE VIEW projet.monstre_au_hasard AS
  SELECT a.*
  FROM projet.archetypes a
  ORDER BY RANDOM()
  LIMIT 1;

-- Historique d'utilisations de power ups

CREATE VIEW projet.historique_pu AS

      SELECT pu.id_pm, pu.nom, pu.facteur, u.date_utilisation
      FROM projet.power_ups pu
      INNER JOIN projet.utilisations u ON pu.id_pu=u.id_pu
      ORDER BY u.date_utilisation;
```

Procédures

```
-- Si nouvelle année, mets les stats annuelles de tout le monde à zéro

CREATE FUNCTION projet.verifier_stats_annee() RETURNS VOID AS $$
DECLARE
      _dernier_combat TIMESTAMP;
BEGIN

  SELECT date_debut INTO _dernier_combat FROM projet.combats ORDER BY
date_debut DESC LIMIT 1;
  IF (_dernier_combat IS NOT NULL AND EXTRACT(YEAR FROM _dernier_combat) <
EXTRACT(YEAR FROM NOW())) THEN
    UPDATE projet.statistiques SET nb_combats_annee = 0, nb_victoires_annee =
0;
  END IF;

END;
$$ LANGUAGE plpgsql;

-------------------------------------------------------------------------------
--                                    Toriko
-------------------------------------------------------------------------------


-- Inscription d'un PM

CREATE FUNCTION projet.inscrire_pm(VARCHAR(100), VARCHAR(150), CHAR(6)) RETURNS
INTEGER AS $$
DECLARE
  _nom      ALIAS FOR $1;
  _mdp      ALIAS FOR $2;
  _couleur  ALIAS FOR $3;
  _id       INTEGER;
BEGIN

      INSERT INTO projet.power_mangeurs (nom, couleur, mot_de_passe) VALUES
(_nom, _couleur, _mdp) RETURNING id_pm INTO _id;

      RETURN _id;

END;
$$ LANGUAGE plpgsql;

-- Ajout d'un archétype

CREATE FUNCTION projet.ajouter_archetype(VARCHAR(100), INTEGER) RETURNS INTEGER
AS $$
DECLARE
      _nom                ALIAS FOR $1;
      _puissance  ALIAS FOR $2;
      _id                   INTEGER;
BEGIN

      INSERT INTO projet.archetypes (nom, puissance) VALUES (_nom, _puissance)
RETURNING id_archetype INTO _id;

      RETURN _id;

END;
$$ LANGUAGE plpgsql;
```

```sql
-- Attribution d'un P-U

CREATE FUNCTION projet.attribuer_pu(VARCHAR(100), VARCHAR(100), INTEGER)
RETURNS INTEGER AS $$
DECLARE
      _nom_pm           ALIAS FOR $1;
      _nom_pu           ALIAS FOR $2;
      _facteur_pu ALIAS FOR $3;
      _id_pm            INTEGER;
      _id_pu            INTEGER;
BEGIN

      -- Vérifier l'existence du P-M
      SELECT id_pm INTO _id_pm FROM projet.power_mangeurs WHERE nom = _nom_pm;

      INSERT INTO projet.power_ups (nom, id_pm, facteur) VALUES (_nom_pu,
_id_pm, _facteur_pu) RETURNING id_pu INTO _id_pu;

      RETURN _id_pu;

END;
$$ LANGUAGE plpgsql;

-- Classement Power Mangeur

CREATE FUNCTION projet.classer_pm() RETURNS TABLE(nom VARCHAR(100), victoires
BIGINT) AS $$
DECLARE
      _dernier_combat   TIMESTAMP;
BEGIN

  PERFORM projet.verifier_stats_annee();


      RETURN QUERY SELECT
            pm.nom,
            COALESCE(SUM(s.nb_victoires_annee), 0) AS "victoires"
       FROM projet.statistiques s
            RIGHT JOIN projet.power_mangeurs pm ON s.id_pm = pm.id_pm
       WHERE pm.vie > 0
       GROUP BY pm.nom
       ORDER BY victoires DESC;

END;
$$ LANGUAGE plpgsql;

-------------------------------------------------------------------------------
--                                Power-Mangeurs
-------------------------------------------------------------------------------


-- Débuter un combat

CREATE FUNCTION projet.debuter_combat(INTEGER, INTEGER) RETURNS INTEGER AS $$
DECLARE
      _id_pm            ALIAS FOR $1;
      _id_arch          ALIAS FOR $2;
      _id_combat  INTEGER;
BEGIN

      -- Vérifier si P-M est déjà en plein combat et lever une exception si
c'est le cas
      IF EXISTS(SELECT * FROM projet.combats WHERE id_pm = _id_pm AND date_fin
```

```
IS NULL) THEN
            RAISE 'Combat en cours.';
      END IF;

      INSERT INTO projet.combats (id_pm, id_archetype) VALUES (_id_pm,
_id_arch) RETURNING id_combat INTO _id_combat;

      RETURN _id_combat;

END;
$$ LANGUAGE plpgsql;

-- Conclure un combat

CREATE OR REPLACE FUNCTION projet.conclure_combat(INTEGER,BOOLEAN) RETURNS
BOOLEAN AS $$
DECLARE
      _id_pm                  ALIAS FOR $1;
      _force_defaite          ALIAS FOR $2;
      _puissance_pm           INTEGER;
      _vie              INTEGER;
      _id_arch          INTEGER;
      _puissance_arch         INTEGER;
      _id_combat        INTEGER;
      _est_gagne        BOOLEAN;
BEGIN

      -- Vérifier que P-M est effectivement en plein combat et lever une
exception si ce n'est pas le cas
      SELECT c.id_combat, c.id_archetype, pm.puissance, a.puissance
      INTO _id_combat, _id_arch, _puissance_pm, _puissance_arch
      FROM projet.combats c
      INNER JOIN projet.power_mangeurs pm ON c.id_pm = pm.id_pm
      INNER JOIN projet.archetypes a ON c.id_archetype = a.id_archetype
      WHERE c.id_pm = _id_pm AND c.date_fin IS NULL;

      IF (_id_combat IS NULL) THEN
            RAISE 'Pas de combat en cours.';
      END IF;

      IF (_force_defaite = TRUE) THEN
            _est_gagne:=FALSE;
      ELSE
            _est_gagne:=(_puissance_pm>_puissance_arch);
      END IF;

      -- Mettre à jour issue combat
      UPDATE projet.combats SET date_fin = LOCALTIMESTAMP, est_gagne =
_est_gagne WHERE id_combat = _id_combat;

      -- Réinitialiser puissance P-M
      IF (_puissance_pm > 30) THEN
            UPDATE projet.power_mangeurs SET puissance = 30 WHERE id_pm =
_id_pm;
      END IF;

      RETURN _est_gagne;

END;
$$ LANGUAGE plpgsql;

-- Utiliser un P-U
```

```
CREATE OR REPLACE FUNCTION projet.utiliser_pu(INTEGER, INTEGER) RETURNS INTEGER
AS $$
DECLARE
      _id_pm                                    ALIAS FOR $1;
      _id_pu                                    ALIAS FOR $2;
      _id_combat                          INTEGER;
      _facteur                                      INTEGER;
      _derniere_utilisation   TIMESTAMP;
      _puissance  INTEGER;
BEGIN

  -- Vérifier que P-M est effectivement en plein combat et lever une exception
si ce n'est pas le cas
      SELECT id_combat INTO _id_combat FROM projet.combats WHERE id_pm = _id_pm
AND date_fin IS NULL;
      IF (_id_combat IS NULL) THEN
            RAISE 'Pas de combat en cours.';
      END IF;

      -- Vérifier que P-U n'a pas déjà été utilisé aujourd'hui
      SELECT date_utilisation INTO _derniere_utilisation FROM
projet.utilisations WHERE id_pu = _id_pu ORDER BY date_utilisation DESC LIMIT
1;

      -- 1x/jour : date_trunc('day', _derniere_utilisation) < date_trunc('day',
NOW())
      -- 1x/24h : (NOW()-_derniere_utilisation) > interval '1 day'
      IF (_derniere_utilisation IS NULL OR (NOW()-_derniere_utilisation) >
interval '1 day') THEN

            -- Selectionner le facteur du Power Up
    SELECT facteur INTO _facteur FROM projet.power_ups WHERE id_pu = _id_pu;

            INSERT INTO projet.utilisations (id_combat, id_pu) VALUES
(_id_combat, _id_pu);

            SELECT puissance INTO _puissance FROM projet.power_mangeurs WHERE
id_pm = _id_pm;
            _puissance := _puissance+_puissance*_facteur/100;

            UPDATE projet.power_mangeurs SET puissance = _puissance WHERE id_pm
= _id_pm;

      ELSE
            RAISE 'Ce Power-Up a déjà été utilisé aujourd''hui !';
      END IF;

      RETURN _puissance;

END;
$$ LANGUAGE plpgsql;

-- Encaisser un prix au JackPot

CREATE OR REPLACE FUNCTION projet.encaisser_jackpot(INTEGER) RETURNS INTEGER AS
$$
DECLARE
      _id_pm            ALIAS FOR $1;
      _vie        INTEGER;
      _date_fin   TIMESTAMP;
BEGIN

      -- Selectionner le nombre de vies du PM et la date de son dernier combat
```

```
	SELECT date_fin INTO _date_fin
	FROM projet.combats
	WHERE id_pm = _id_pm AND date_fin IS NOT NULL
	ORDER BY date_fin DESC
	LIMIT 1;

	-- Controle qu'il existe un dernier combat
	IF _date_fin IS NULL THEN
		RAISE 'Combat en cours ou inexistant';
	END IF;

	-- Controle timing
	IF (LOCALTIMESTAMP - _date_fin) > (INTERVAL '5 minutes') THEN
		RAISE 'JackPot seulement valable 5 minutes apres une fin de
combat';
	END IF;

	-- Incremente les vies
	UPDATE projet.power_mangeurs SET vie = vie+1 WHERE id_pm = _id_pm
RETURNING vie INTO _vie;

	-- Retourne le nombre de vies actuel
	RETURN _vie;

END;
$$ LANGUAGE plpgsql;

-- Visualiser l'historique du dernier combat

CREATE TYPE projet.liste_actions AS (date TIMESTAMP, action VARCHAR(255));

CREATE OR REPLACE FUNCTION projet.visualiser_combat(INTEGER) RETURNS SETOF
projet.liste_actions AS $$
DECLARE
	_id_pm			ALIAS FOR $1;
	_id_combat		INTEGER;
	_debut_combat		TIMESTAMP;
	_fin_combat		TIMESTAMP;
	_power_up		RECORD;
	_action			projet.liste_actions;
BEGIN

	SELECT id_combat, date_debut, date_fin INTO _id_combat, _debut_combat,
_fin_combat FROM projet.combats WHERE id_pm = _id_pm ORDER BY date_fin DESC
LIMIT 1;

	IF _id_combat IS NULL THEN
		RETURN;
	END IF;

	-- Retourner date début combat
	SELECT _debut_combat, 'Début du combat' INTO _action;
	RETURN NEXT _action;

	-- Retourner utilisations P-U
	FOR _power_up IN
		SELECT ut.date_utilisation, pu.nom FROM projet.utilisations ut
INNER JOIN projet.power_ups pu ON ut.id_pu = pu.id_pu WHERE id_combat =
_id_combat ORDER BY ut.date_utilisation ASC
	LOOP
		SELECT _power_up.date_utilisation, 'Activation du P-U ' ||
_power_up.nom INTO _action;
		RETURN NEXT _action;
```

```
      END LOOP;

      -- Retourner date fin combat
      SELECT _fin_combat, 'Fin du combat' INTO _action;
      RETURN NEXT _action;

      RETURN;

END;
$$ LANGUAGE plpgsql;

-- Statistiques PM

CREATE FUNCTION projet.stats_pm(INTEGER) RETURNS TABLE(nom_archetype
VARCHAR(100), nb_combats_total INTEGER, nb_victoires_total INTEGER,
nb_combats_annee INTEGER, nb_victoires_annee INTEGER) AS $$
DECLARE
      _id_pm ALIAS FOR $1;
BEGIN

  PERFORM projet.verifier_stats_annee();

  RETURN QUERY SELECT
            a.nom AS "nom_archetype",
            s.nb_combats_total,
            s.nb_victoires_total,
            s.nb_combats_annee,
            s.nb_victoires_annee
       FROM projet.statistiques s
            INNER JOIN projet.archetypes a ON s.id_archetype = a.id_archetype
           WHERE id_pm = _id_pm;

END;
$$ LANGUAGE plpgsql;

-- Calculer l'espérance de vie

CREATE FUNCTION projet.esperance_vie(INTEGER) RETURNS INTERVAL AS $$
DECLARE
      _id                           ALIAS FOR $1;
      _date_inscription TIMESTAMP;
      _vie                          INTEGER;
      _esperance              INTERVAL;
BEGIN

      SELECT date_inscription, vie INTO _date_inscription, _vie FROM
projet.power_mangeurs WHERE id_pm = _id;

      IF (_vie = 0) THEN
            RAISE EXCEPTION 'Vous êtes mort !';
      ELSIF (_vie = 10) THEN
            RAISE WARNING 'Vous êtes invincible !';
            RETURN 0;
      END IF;

      _esperance:=_vie*(NOW()-_date_inscription)/(10-_vie);

      RETURN justify_interval(_esperance);

END;
$$ LANGUAGE plpgsql;
```

Déclencheurs

```
-- Appel màj stats annuelles

CREATE FUNCTION projet.verifier_stats() RETURNS TRIGGER AS $$
DECLARE
      _dernier_combat TIMESTAMP;
BEGIN

      PERFORM projet.verifier_stats_annee();

      RETURN NEW;

END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER verifier_stats
  BEFORE INSERT ON projet.combats
  FOR EACH ROW
  EXECUTE PROCEDURE projet.verifier_stats();

-- Combat

CREATE FUNCTION projet.ajouter_combat() RETURNS TRIGGER AS $$
  BEGIN

    -- Créer ligne de stats si pas existante
      IF NOT EXISTS(SELECT * FROM projet.statistiques WHERE id_pm = NEW.id_pm
AND id_archetype = NEW.id_archetype) THEN
            INSERT INTO projet.statistiques (id_pm, id_archetype) VALUES
(NEW.id_pm, NEW.id_archetype);
      END IF;

      UPDATE projet.statistiques SET nb_combats_total = nb_combats_total+1,
nb_combats_annee = nb_combats_annee+1 WHERE id_pm = NEW.id_pm AND id_archetype
= NEW.id_archetype;

  RETURN NULL;

  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER ajouter_combat
  AFTER INSERT ON projet.combats
  FOR EACH ROW
  EXECUTE PROCEDURE projet.ajouter_combat();

-- Victoire

CREATE FUNCTION projet.ajouter_victoire() RETURNS TRIGGER AS $$
  BEGIN

    UPDATE projet.statistiques SET nb_victoires_total = nb_victoires_total+1,
nb_victoires_annee = nb_victoires_annee+1 WHERE id_pm = NEW.id_pm AND
id_archetype = NEW.id_archetype;

    RETURN NULL;

  END;
```

```
$$ LANGUAGE plpgsql;

CREATE TRIGGER ajouter_victoire
      AFTER INSERT OR UPDATE OF est_gagne ON projet.combats
      FOR EACH ROW
      WHEN (NEW.est_gagne = TRUE)
      EXECUTE PROCEDURE projet.ajouter_victoire();

-- Défaite

CREATE FUNCTION projet.ajouter_defaite() RETURNS TRIGGER AS $$
DECLARE
      _vie        INTEGER;
BEGIN

      UPDATE projet.power_mangeurs SET vie = vie-1 WHERE id_pm = NEW.id_pm
RETURNING vie INTO _vie;

      IF (_vie = 0) THEN
            UPDATE projet.power_mangeurs SET date_deces = LOCALTIMESTAMP WHERE
id_pm = NEW.id_pm;
      END IF;

      RETURN NULL;

END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER ajouter_defaite
      AFTER INSERT OR UPDATE OF est_gagne ON projet.combats
      FOR EACH ROW
      WHEN (NEW.est_gagne = FALSE)
      EXECUTE PROCEDURE projet.ajouter_defaite();
```

Données valides

```
-- Suppression ordonnée pour éviter violation des contraintes

DELETE FROM projet.utilisations;
DELETE FROM projet.combats;
DELETE FROM projet.power_ups;
DELETE FROM projet.archetypes;
DELETE FROM projet.power_mangeurs;

ALTER SEQUENCE projet.id_combat RESTART;
ALTER SEQUENCE projet.id_power_up RESTART;
ALTER SEQUENCE projet.id_archetype RESTART;
ALTER SEQUENCE projet.id_power_mangeur RESTART;

-- Power Mangeurs

INSERT INTO projet.power_mangeurs (nom, mot_de_passe, couleur, vie,
date_inscription, date_deces) VALUES
  ('Jean',
'984facf29f1f1701b7f474f64e3ba3f0e14375947b0f22392272a44892cf64802a921728b54459
20c0244675992da3ff0e3ab950c1d63d3c4a1d89ed9deee635', 'C04AD1', DEFAULT,
DEFAULT, NULL),
  ('Gerard',
```

```
'984facf29f1f1701b7f474f64e3ba3f0e14375947b0f22392272a44892cf64802a921728b54459
20c0244675992da3ff0e3ab950c1d63d3c4a1d89ed9deee635', '930C48', 8, '2014-12-03
18:15:12', NULL),
  ('Charles',
'984facf29f1f1701b7f474f64e3ba3f0e14375947b0f22392272a44892cf64802a921728b54459
20c0244675992da3ff0e3ab950c1d63d3c4a1d89ed9deee635', '2E69A3', 3, '2014-09-02
14:30:20', NULL),
  ('Hubert',
'984facf29f1f1701b7f474f64e3ba3f0e14375947b0f22392272a44892cf64802a921728b54459
20c0244675992da3ff0e3ab950c1d63d3c4a1d89ed9deee635', 'A8FF20', 0, '2014-08-15
17:03:20', '2014-11-30 11:20:54');

-- Archétypes

INSERT INTO projet.archetypes (nom, puissance) VALUES
  ('Tiramisu', 50),
  ('Couscous', 40),
  ('Poireau', 35),
  ('Concombre', 30),
  ('Courgette', 20),
  ('Yoagurth', 15);

-- Power-Ups

INSERT INTO projet.power_ups (nom, id_pm, date_attribution, facteur) VALUES
  ('Force obscure', 2, DEFAULT, 60),
  ('Orbe Rouge', 3, '2014-10-10 12:30:17', 70),
  ('Orbe Bleue', 3, '2014-11-29 19:08:20', 30),
  ('Mutantox', 4, '2014-08-22 09:48:25', 50);

-- Combats

INSERT INTO projet.combats (id_pm, id_archetype, date_debut, date_fin,
est_gagne) VALUES
  (2, 3, DEFAULT, NULL, NULL),
  (3, 4, '2014-11-30 20:17:01', '2014-11-30 20:22:35', TRUE),
  (3, 4, '2014-10-15 15:31:44', '2014-10-15 15:39:26', TRUE),
  (3, 4, '2014-10-15 18:14:17', '2014-10-15 18:16:43', FALSE),
  (3, 5, '2014-10-17 10:09:36', '2014-10-17 10:13:34', TRUE),
  (3, 5, '2014-10-18 06:36:21', '2014-10-18 06:46:21', FALSE),
  (3, 1, '2014-10-29 14:50:13', '2014-10-29 14:59:20', TRUE);

-- Utilisations

INSERT INTO projet.utilisations (id_combat, id_pu, date_utilisation) VALUES
  (2, 3, '2014-11-30 20:19:28'),
  (3, 3, '2014-10-15 15:34:21'),
  (7, 3, '2014-10-29 14:53:45'),
  (7, 2, '2014-10-29 14:57:12');
```

Données invalides

```
-- Table power_mangeurs

-- Controle que la date d'inscription soit inferieur a la date de deces
INSERT INTO projet.power_mangeurs (nom, couleur, mot_de_passe,
date_inscription, date_deces)
VALUES ('John', 'FFFFFF', '***', '2014-10-10 12:00:00', '2012-10-10 12:00:00');
```

```sql
-- Controle que le mot de passe ne soit pas vide
INSERT INTO projet.power_mangeurs (nom, couleur, mot_de_passe)
VALUES ('John', 'FFFFFF', '');

-- Controle que le nom (trimme) ait au moins 3 caracteres
INSERT INTO projet.power_mangeurs (nom, couleur, mot_de_passe)
VALUES ('  J  ', 'FFFFFF', '***');

-- Controle que la puissance soit au moins de 30 points
INSERT INTO projet.power_mangeurs (nom, couleur, mot_de_passe, puissance)
VALUES ('John', 'FFFFFF', '***', 15);

-- Controle que le nombre de vies ne soit pas negatif
INSERT INTO projet.power_mangeurs (nom, couleur, mot_de_passe, vie)
VALUES ('John', 'FFFFFF', '***', -4);

-- Controle que le nombre de vies ne soit pas superieur a 10
INSERT INTO projet.power_mangeurs (nom, couleur, mot_de_passe, vie)
VALUES ('John', 'FFFFFF', '***', 11);

-- Table archetypes

-- Controle pour voir si le nom de l'archetype n'est pas vide
INSERT INTO projet.archetypes (nom, puissance)
VALUES ('', 20);

-- Controle pour s'assurer que la puissance de l'archetype ne soit pas negative
INSERT INTO projet.archetypes (nom, puissance)
VALUES ('Couscous', -10);

-- Table power_ups

-- Controle que la date d'attribution ne soit pas dans le futur
INSERT INTO projet.power_ups (nom, id_pm, date_attribution, facteur)
VALUES ('Spatule', 1, '2020-10-10 12:00:00', 50);

-- Controle pour voir que le facteur soit entierement positif
INSERT INTO projet.power_ups (nom, id_pm, facteur)
VALUES ('Spatule', 1, 0);

-- Controle pour voir que le nom ne soit pas vide
INSERT INTO projet.power_ups (nom, id_pm, facteur)
VALUES ('', 1, 50);

-- Table combats

-- Controle pour s'assurer que la date de fin soit strictement superieur a la
date de debut
INSERT INTO projet.combats (id_pm, id_archetype, date_debut, date_fin)
VALUES (1, 1, '2014-10-10 12:00:00', '2014-10-10 10:00:00');

-- Controle pour voir si la date de debut ne se situe pas dans le futur
INSERT INTO projet.combats (id_pm, id_archetype, date_debut)
VALUES (1, 1, '2020-10-10 12:00:00');

-- Table utilisations

-- Controle que la date d'utilisation du power up ne se situe pas dans le futur
INSERT INTO projet.utilisations (id_combat, id_pu, date_utilisation)
VALUES (1, 1, '2020-10-10 12:00:00');

-- Table statistiques
```

```sql
-- Controle que le nombres de victoires total soit inferieur ou egal au nombre de combats
INSERT INTO projet.statistiques (id_pm, id_archetype, nb_combats_total, nb_victoires_total, nb_combats_annee, nb_victoires_annee)
VALUES (1, 1, 20, 21, 20, 10);

-- Controle que le nombres de victoires par annee soit inferieur ou egal au nombre de combats
INSERT INTO projet.statistiques (id_pm, id_archetype, nb_combats_total, nb_victoires_total, nb_combats_annee, nb_victoires_annee)
VALUES (1, 1, 20, 10, 20, 21);

-- Controle que le nombre de victoires au total soit positif
INSERT INTO projet.statistiques (id_pm, id_archetype, nb_combats_total, nb_victoires_total, nb_combats_annee, nb_victoires_annee)
VALUES (1, 1, 20, -5, 20, 10);

-- Controle que le nombre de victoires par annee soit positif
INSERT INTO projet.statistiques (id_pm, id_archetype, nb_combats_total, nb_victoires_total, nb_combats_annee, nb_victoires_annee)
VALUES (1, 1, 20, 10, 20, -5);

-- Note: il est inutile de controler que le nombre de combats total ou par annee soit positif,
-- car en controlant que le nombre de victoires ne puisse pas etre negatif, le nombre de combats
-- devra etre aussi plus grand que le nombre de victoires.

-- Controle que le nombre de combats par annee soit inferieur au nombre total de combats
INSERT INTO projet.statistiques (id_pm, id_archetype, nb_combats_total, nb_victoires_total, nb_combats_annee, nb_victoires_annee)
VALUES (1, 1, 20, 10, 30, 10);

-- Controle que le nombre de victoires par annee soit inferieur au nombre total de victoires
INSERT INTO projet.statistiques (id_pm, id_archetype, nb_combats_total, nb_victoires_total, nb_combats_annee, nb_victoires_annee)
VALUES (1, 1, 20, 10, 20, 30);
```

Applications Java

Toriko

```java
import java.sql.*;
import java.util.ArrayList;
import java.util.InputMismatchException;
import java.util.Scanner;

public class Terminal {

    private static final String DB_HOST = "localhost";
    private static final String DB_NAME = "dbcdandoy14";
    private static final String DB_USER = "cdandoy14";
    private static final String DB_PWD = "*******";

    private static Scanner scan;
    private static Connection db;
```

```
    private static ArrayList<PreparedStatement> statements;

    private static byte choix = -1;

    public static void main(String[] args) {

        try {
            Class.forName("org.postgresql.Driver");
        } catch (ClassNotFoundException e) {
            System.out.println("Driver PostgreSQL manquant !");
            System.exit(1);
        }

        try {
            db = DriverManager.getConnection("jdbc:postgresql://"+DB_HOST
+"/"+DB_NAME+"?user="+DB_USER+"&password="+DB_PWD);
        } catch (SQLException e) {
            System.out.println("Impossible de joindre le server !");
            System.exit(1);
        }

        try {
            statements = new ArrayList<PreparedStatement>(7);
            statements.add(null);
            statements.add(db.prepareStatement("SELECT projet.inscrire_pm(?, ?,
?);"));
            statements.add(db.prepareStatement("SELECT
projet.ajouter_archetype(?, ?);"));
            statements.add(db.prepareStatement("SELECT
projet.attribuer_pu(?, ?, ?);"));
            statements.add(db.prepareStatement("SELECT nom, victoires FROM
projet.classer_pm();"));
            statements.add(db.prepareStatement("SELECT nom, date_deces FROM
projet.liste_decedes"));
            statements.add(db.prepareStatement("SELECT nom_archetype AS
\"archetype\", date_debut AS \"date\", est_gagne AS \"issue\" FROM
projet.historique_combats WHERE nom_pm = ? AND date_debut BETWEEN ? AND ?"));
            statements.add(db.prepareStatement("SELECT nom_pm AS
\"power_mangeur\", DATE_TRUNC('DAY', date_debut) AS \"date\", COUNT(*) AS
\"nb_combats\" FROM projet.historique_combats WHERE nom_archetype = ? AND
date_debut BETWEEN ? AND ? GROUP BY \"power_mangeur\", \"date\"")); //
nb_combats: nombre de combats par Power Mangeur par jour
        } catch (SQLException e) {
            System.out.println("Probleme de requete preparee !");
            e.printStackTrace();
            System.exit(1);
        }

        scan = new Scanner(System.in);
        scan.useDelimiter(System.getProperty("line.separator"));

        launch();

        try {
            scan.close();
            db.close();
//            System.out.println("Fermeture de la connexion a la BDD
reussie.");
        } catch (SQLException e) {
            System.out.println("Fermeture de la connexion a la BDD echouee.");
        } finally {
            System.out.println("Au revoir !");
        }
```

```
    }

    public static void launch () {

        System.out.println("Bienvenue Toriko");
        System.out.println("----------------");

        afficherMenu();

        while (choix != 0) {

            System.out.print("\nCommande ");

            try {
                choix = scan.nextByte();
            } catch (Exception e) {
                scan.nextLine();
                choix = -1;
            }

            switch (choix) {
                case 0:
                    break;
                case 1:
                    inscrire_pm();
                    break;
                case 2:
                    ajouter_monstre();
                    break;
                case 3:
                    attribuer_pu();
                    break;
                case 4:
                    classement_pm();
                    break;
                case 5:
                    liste_deces();
                    break;
                case 6:
                    historique_pm();
                    break;
                case 7:
                    historique_archetype();
                    break;
                default:
                    System.out.println("Ceci n'est pas une commande valide.");
                    afficherMenu();
                    break;
            }

        }

    }

    private static void afficherMenu() {
        System.out.println("\nAide");
        System.out.println("
---------------------------------------------------- ");
        System.out.println("| 1 | Inscription d'un Power Mangeur
|");
        System.out.println("| 2 | Ajout d'un archetype
|");
```

```
        System.out.println("| 3 | Attribution d'un Power-Up
|");
        System.out.println("| 4 | Classement des Power Mangeurs
|");
        System.out.println("| 5 | Liste des Power Mangeurs decedes cette annee
|");
        System.out.println("| 6 | Historique des combats d'un Power Mangeur
|");
        System.out.println("| 7 | Historique des combats d'un archetype
|");
        System.out.println("
-----------------------------------------------------");
        System.out.println("| 0 | Quitter l'application
|");
        System.out.println("
-----------------------------------------------------");
    }

    private static void inscrire_pm() {

        System.out.println("\nInscription d'un Power Mangeur");
        System.out.println("-----------------------------\n");

        String nom = null, mdp, couleur;
        PreparedStatement statement = statements.get(choix);

        try {
            // NOM

            System.out.print("Nom : ");
            try {
                nom = scan.next();
            } catch (Exception e) {
                System.out.println("Probleme d'input.");
                return;
            }
            statement.setString(1, nom);

            // MOT DE PASSE

            System.out.print("Mot de passe : ");
            try {
                mdp = scan.next();
            } catch (Exception e) {
                System.out.println("Probleme d'input.");
                return;
            }
            mdp = CryptService.hash(mdp);
            statement.setString(2, mdp);

            // COULEUR

            System.out.print("Couleur : ");
            try {
                couleur = scan.next();
            } catch (Exception e) {
                System.out.println("Probleme d'input.");
                return;
            }
            statement.setString(3, couleur);

            System.out.println();
        } catch (SQLException e) {
```

```
            System.out.println("Erreur avec la base de donnees.");
            System.exit(1);
        }

        System.out.println("Inscription du PM en cours...");
        try {
            statement.execute();
            System.out.println("Inscription reussie !");
        } catch (SQLException e) {
            System.out.println("Probleme a l'inscription !");
            System.out.println(e.getMessage());
        }

    }

    private static void ajouter_monstre () {

        System.out.println("\nAjout d'un archetype de monstro-nourriture");
        System.out.println("-----------------------------------------\n");

        String nom;
        int puissance;
        PreparedStatement statement = statements.get(choix);

        try {
            System.out.print("Nom : ");
            try {
                nom = scan.next();
            } catch (Exception e) {
                System.out.println("Probleme d'input.");
                return;
            }
            statement.setString(1, nom);

            System.out.print("Puissance : ");
            try {
                puissance = scan.nextInt();
            } catch (InputMismatchException e) {
                scan.nextLine();
                System.out.println("Problème d'input.");
                return;
            } catch (Exception e) {
                System.out.println("Probleme d'input.");
                return;
            }
            statement.setInt(2, puissance);

            System.out.println();
        } catch (SQLException e) {
            System.out.println("Erreur avec la base de donnees.");
            System.exit(1);
        }

        System.out.println("Ajout de l'archetype en cours...");
        try {
            statement.execute();
            System.out.println("Ajout reussi !");
        } catch (SQLException e) {
            System.out.println("Probleme a l'ajout !");
            System.out.println(e.getMessage());
        }
    }
```

```
    private static void attribuer_pu () {

        System.out.println("\nCreation d'un Power-Up");
        System.out.println("----------------------\n");

        String nom_pu, nom_pm;
        int facteur_pu;
        PreparedStatement statement = statements.get(choix);

        try {

            // Liste les PM vivants et stop s'il n'en existe aucun
            ArrayList<String> liste = lister_pm(true);
            if (liste.isEmpty()) {
                System.out.println("Impossible donc d'attribuer de Power-Up.");
                return;
            }

            while (true) {
                System.out.print("Nom du Power Mangeur : ");
                try {
                    nom_pm = scan.next();
                } catch (Exception e) {
                    System.out.println("Probleme d'input.");
                    return;
                }
                if (! liste.contains(nom_pm)) {
                    System.out.println("Ce Power Mangeur n'existe pas.");
                    continue;
                }
                statement.setString(1, nom_pm);
                break;
            }

            System.out.print("Nom du Power-Up : ");
            try {
                nom_pu = scan.next();
            } catch (Exception e) {
                System.out.println("Probleme d'input.");
                return;
            }
            statement.setString(2, nom_pu);

            System.out.print("Facteur de multiplication du Power-Up : ");
            try {
                facteur_pu = scan.nextInt();
            } catch (InputMismatchException e) {
                scan.nextLine();
                System.out.println("Problème d'input.");
                return;
            } catch (Exception e) {
                System.out.println("Probleme d'input.");
                return;
            }
            statement.setInt(3, facteur_pu);

            System.out.println();
        } catch (SQLException e) {
            System.out.println("Erreur avec la base de donnees.");
            System.exit(1);
        }

        System.out.println("Creation du Power-Up en cours...");
```

```
        try {
            statement.execute();
            System.out.println("Creation reussie !");
        } catch (SQLException e) {
            System.out.println("Probleme a la creation :");
            System.out.println(e.getMessage());
        }
    }

    private static void classement_pm () {

        String nom_pm;
        int nb_victoires;

        System.out.println("\nClassement des meilleurs Power Mangeurs");
        System.out.println("---------------------------------------\n");

        try {
            ResultSet result = statements.get(choix).executeQuery();
            if (result.next()) {
                System.out.println(" ---------------------------- ");
                System.out.println("|  Power Mangeur  | Victoires |");
                System.out.println(" ---------------------------- ");
                do {
                    nom_pm = result.getString("nom");
                    nb_victoires = result.getInt("victoires");

                    System.out.print("| "+nom_pm);
                    for (int i = nom_pm.length(); i < 15; i++)
                        System.out.print(" ");
                    System.out.println(" |     "+nb_victoires+"     |");
                } while (result.next());
                System.out.println(" ---------------------------- ");
            } else
                System.out.println("Il n'y a aucun Power Mangeur
enregistre !");
        } catch (SQLException e) {
            System.out.println("Erreur avec la base de donnees.");
            System.out.println(e.getMessage());
        }

    }

    private static void liste_deces () {

        String nom_pm;
        Date deces;

        System.out.println("\nListe des deces sur l'annee");
        System.out.println("--------------------------\n");

        try {
            ResultSet result = statements.get(choix).executeQuery();
            if (result.next()) {
                System.out.println(" ---------------------------- ");
                System.out.println("|  Power Mangeur  | Date deces |");
                System.out.println(" ---------------------------- ");
                do {
                    nom_pm = result.getString("nom");
                    deces = result.getDate("date_deces");

                    System.out.print("| "+nom_pm);
                    for (int i = nom_pm.length(); i < 15; i++)
```

```
                        System.out.print(" ");
                    System.out.println(" | "+deces+" |");
                } while (result.next());
                System.out.println(" ------------------------------ ");
            } else
                System.out.println("Bonne nouvelle ! Aucun Power Mangeur n'est
mort cette annee !");
        } catch (SQLException e) {
            System.out.println("Erreur avec la base de donnees.");
        }

    }

    private static void historique_pm () {

        String nom_pm, nom_archetype, date_debut, date_fin, issue;
        Date debut, fin;
        PreparedStatement statement = statements.get(choix);

        System.out.println("\nHistorique des combats d'un Power Mangeur");
        System.out.println("----------------------------------------\n");

        try {

            // Liste tous les PM et stop s'il n'en existe aucun
            ArrayList<String> liste = lister_pm(false);
            if (liste.isEmpty())
                return;

            while (true) {
                System.out.print("Nom du Power Mangeur : ");
                try {
                    nom_pm = scan.next();
                } catch (Exception e) {
                    System.out.println("Probleme d'input.");
                    return;
                }
                if (! liste.contains(nom_pm)) {
                    System.out.println("Ce Power Mangeur n'existe pas.");
                    continue;
                }
                statement.setString(1, nom_pm);
                break;
            }

            while (true) {
                System.out.print("Debut de periode : ");
                try {
                    date_debut = scan.next();
                    debut = Date.valueOf(date_debut);
                } catch (IllegalArgumentException e) {
                    System.out.println("La date doit etre au format \"YYY-[M]M-
[D]D\".");
                    continue;
                } catch (Exception e) {
                    System.out.println("Probleme d'input.");
                    return;
                }
                statement.setDate(2, debut);
                break;
            }

            while (true) {
```

```
                System.out.print("Fin de periode : ");
                try {
                    date_fin = scan.next();
                    fin = Date.valueOf(date_fin);
                } catch (IllegalArgumentException e) {
                    System.out.println("La date doit etre au format \"YYY-[M]M-
[D]D\".");
                    continue;
                } catch (Exception e) {
                    System.out.println("Probleme d'input.");
                    return;
                }
                statement.setDate(3, fin);
                break;
            }

            System.out.println();

            if (debut.after(fin)) {
                System.out.println("Periode invalide !");
                return;
            }

        } catch (SQLException e) {
            System.out.println("Erreur avec la base de donnees.");
            System.exit(1);
        }

        try {
            ResultSet result = statement.executeQuery();
            if (result.next()) {
                System.out.println("
------------------------------------------- ");
                System.out.println("| Monstro-nourriture |    Date    |  Issue
|");
                System.out.println("
------------------------------------------- ");
                do {
                    nom_archetype = result.getString("archetype");

                    debut = result.getDate("date");
                    date_debut = debut.toString();

                    if (result.getObject("issue") == null)
                        issue = "En cours";
                    else
                        issue = result.getBoolean("issue") ? "Victoire" :
"Defaite ";

                    System.out.print("| "+nom_archetype);
                    for (int i = nom_archetype.length(); i < 18; i++)
                        System.out.print(" ");
                    System.out.println(" | "+date_debut+" | "+issue+" |");
                } while (result.next());
                System.out.println("
------------------------------------------- ");
            }
            else
                System.out.println("Aucun combat n'a ete trouve.");
        } catch (SQLException e) {
            System.out.println("Probleme avec la requete.");
            System.out.println(e.getMessage());
        }
```

```
    }

    private static void historique_archetype () {

        String nom_pm, nom_archetype, date, date_debut, date_fin;
        Date debut, fin;
        int nb_combats;
        PreparedStatement statement = statements.get(choix);

        System.out.println("\nHistorique des combats d'un archetype");
        System.out.println("-------------------------------------\n");

        try {

            // Liste tous les archétypes et stop s'il n'en existe aucun
            ArrayList<String> liste = lister_arch();
            if (liste.isEmpty())
                return;

            System.out.print("Nom de l'archetype : ");
            try {
                nom_archetype = scan.next();
            } catch (Exception e) {
                System.out.println("Probleme d'input.");
                return;
            }
            statement.setString(1, nom_archetype);

            while (true) {
                System.out.print("Debut de periode : ");
                try {
                    date_debut = scan.next();
                    debut = Date.valueOf(date_debut);
                } catch (IllegalArgumentException e) {
                    System.out.println("La date doit etre au format \"YYY-[M]M-
[D]D\".");
                    continue;
                } catch (Exception e) {
                    System.out.println("Probleme d'input.");
                    return;
                }
                statement.setDate(2, debut);
                break;
            }

            while (true) {
                System.out.print("Fin de periode : ");
                try {
                    date_fin = scan.next();
                    fin = Date.valueOf(date_fin);
                } catch (IllegalArgumentException e) {
                    System.out.println("La date doit etre au format \"YYY-[M]M-
[D]D\".");
                    continue;
                } catch (Exception e) {
                    System.out.println("Probleme d'input.");
                    return;
                }
                statement.setDate(3, fin);
                break;
            }

            System.out.println();
```

```
            if (debut.after(fin)) {
                System.out.println("Periode invalide !");
                return;
            }

        } catch (SQLException e) {
            System.out.println("Erreur avec la base de donnees.");
            System.exit(1);
        }

        try {
            ResultSet result = statement.executeQuery();
            if (result.next()) {
                System.out.println(" ---------------------------------- ");
                System.out.println("|    Power Mangeur    |    Date    |");
                System.out.println(" ---------------------------------- ");
                do {
                    nom_pm = result.getString("power_mangeur");
                    date = result.getDate("date").toString();
                    nb_combats = result.getInt("nb_combats");
                    if (nb_combats > 1)
                        nom_pm += " ("+nb_combats+"x)";

                    System.out.print("| "+nom_pm);
                    for (int i = nom_pm.length(); i < 19; i++)
                        System.out.print(" ");
                    System.out.println(" | "+date+" |");
                } while (result.next());
                System.out.println(" ---------------------------------- ");
            }
            else
                System.out.println("Aucun combat n'a ete trouve.");
        } catch (SQLException e) {
            System.out.println("Probleme avec la requete.");
            System.out.println(e.getMessage());
        }
    }

    private static ArrayList<String> lister_pm (boolean vivant) throws
SQLException {
        String statut, condition = vivant ? " WHERE vie > 0" : "";
        ResultSet liste = db.prepareStatement("SELECT * FROM
projet.power_mangeurs"+condition+" ORDER BY date_inscription
DESC").executeQuery();
        ArrayList<String> table = new ArrayList<String>();
        if (liste.next()) {
            System.out.println("Power Mangeurs :");
            do {
                statut = (liste.getInt("vie") > 0) ? "+" : "-";
                System.out.println("  "+statut+" "+liste.getString("nom"));
                table.add(liste.getString("nom"));
            } while (liste.next());
            System.out.println();
        } else {
            System.out.println("Aucun Power Mangeur rencense !");
        }
        return table;
    }

    /**
     * Sélectionne en BDD tous les archétypes et les liste par ordre
alphabétique
```

```
     * @return ArrayList
     * @throws SQLException
     */
    private static ArrayList<String> lister_arch () throws SQLException {
        ResultSet liste = db.prepareStatement("SELECT * FROM projet.archetypes
ORDER BY nom").executeQuery();
        ArrayList<String> table = new ArrayList<String>();
        if (liste.next()) {
            System.out.println("Archetypes :");
            do {
                System.out.println("  * "+liste.getString("nom"));
                table.add(liste.getString("nom"));
            } while (liste.next());
            System.out.println();
        } else {
            System.out.println("Aucun archetype rencense !");
        }
        return table;
    }
}
```

Power Mangeur

```
package terminalPM;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;

/**
 * Lance un terminal pour les Power Mangeurs
 *
 * Ce terminal, apres s'etre connecte au serveur, permettra au Power Mangeur de
 * lancer ou continuer un combat, de voir le deroulement de son dernier combat
 * et de visualiser des statistiques concernant ses combats.
 */
public class App {

      private static final String SERVER = "localhost";
      private static final String USER = "cbadot-14";
      private static final String PASS = "password";
      private static final String DBNAME = "dbcbadot14";

      private static final int CLOSE_ACTION = 0;
      private static final int BATTLE_ACTION = 1;
      private static final int HISTORY_ACTION = 2;
      private static final int STATS_ACTION = 3;
      private static final int JACKPOT_ACTION = 4;

      private static Scanner scanner = new Scanner(System.in);

      private Connection dbConnection;
      private int userID;

      /**
```

```
	 * Construit l'application et dirige son comportement
	 *
	 * Cette methode controle toute l'execution de l'application, de la
connection
	 * au serveur de bases de donnees PostgreSQL jusqu'a la fermeture de
l'application.
	 * Ceci est le déroulement classique de l'application:
	 * - Connection au serveur
	 * - Identification du Power Mangeur
	 * - Eventuellement continuer une bataille qui était en cours
	 * - Montrer le menu principal et executer autant d'actions que le Power
Mangeur souhaitera
	 * - Fermer l'application
	 */
	public App() {

		databaseConnect();

		userID = ( new LoginHandler(dbConnection) ).login();

		try {
			restartLastBattle();

			int selectedAction = showMenu();

			while( selectedAction != CLOSE_ACTION ) {

				executeAction(selectedAction);
				selectedAction = showMenu();
			}
		}
		catch(DeadException e) {
			System.out.println("\nDesole, tu viens de mourir...");
			System.out.println("Tu ne pourras donc plus te connecter a ce
terminal.");
		}

		closeApp();
	}

	/**
	 * Tente d'etablir une connexion a la base de donnees
	 *
	 * Avant d'etablir la connexion, un controle est effectue pour voir si le
	 * driver PostreSQL est bien installe.
	 * Si la connection est effectuee, elle est persistee dans l'attribut
	 * dbConnection de la classe App. Sinon, en cas d'erreur, le programme
est
	 * interrompu directement en terminant avec un code de retour 1.
	 *
	 * @return  void
	 */
	private void databaseConnect() {

		try {
			Class.forName("org.postgresql.Driver");
		}
		catch(ClassNotFoundException e) {
			System.out.println("Module PostgreSQL manquant");
			System.exit(1);
		}

		String databaseUrl="jdbc:postgresql://"+SERVER+"/"+ DBNAME +"?
```

```
user="+ USER +"&password="+ PASS;

            try {
                  this.dbConnection = DriverManager.getConnection(databaseUrl);
            }
            catch (SQLException e) {
                  System.out.println("Serveur PostgreSQL distant ne reponds
pas");
                  System.exit(1);
            }
      }

      /**
       * Relance le dernier combat en cours
       *
       * Un controle est effectue pour voir si le Power Mangeur possede encore
       * un combat qui n'est pas termine (qui n'a pas encore de date de fin).
       * Si c'est le cas, une bataille est lancee avec les informations de la
bataille
       * qui n'etait pas terminee. Sinon, si tous les combats sont acheves,
       * une simple notification est affichee.
       *
       * @return  void
       */
      private void restartLastBattle() throws DeadException {

            try {

                  PreparedStatement ps = dbConnection.prepareStatement("SELECT
id_combat FROM projet.combats WHERE id_pm=? AND date_fin IS NULL");
                  ps.setInt(1, this.userID);
                  ResultSet rs = ps.executeQuery();

                  if( !rs.next() )
                        return;

                  System.out.println("\nRecuperation du combat precedent en
cours...");
                  new BattleHandler(dbConnection, userID,
rs.getInt("id_combat") );
            }
            catch (SQLException e) {
                  e.printStackTrace();
            }
      }

      /**
       * Affiche le menu principal de l'application
       *
       * Cette methode a pour seul but d'afficher le menu principal de
l'application
       * ainsi que de renvoyer le numero de l'action choisi par l'utilisateur.
       * Il n'y a pas de controle pour regarder si le numero correspond bien a
une action.
       *
       * @return  int         Numero de l'action
       */
      private int showMenu() {

            System.out.println("\nTerminal Power Mangeur");
            System.out.println("----------------------\n");

            System.out.println("#"+ BATTLE_ACTION +"\tPOWER MANGEUR
```

```
ACTIVATION!");
            System.out.println("#"+ HISTORY_ACTION +"\tHistorique dernier
combat");
            System.out.println("#"+ STATS_ACTION +"\tStatistiques");
            System.out.println("#"+ JACKPOT_ACTION +"\tJackpot");
            System.out.println("#"+ CLOSE_ACTION +"\tQuitter");

            System.out.print("\nChoix: ");

            return scanner.nextInt();
      }

      /**
       * Execute une action specifique
       *
       * Execute l'action dont le numero est passe en parametre. Si le numero
       * n'est pas lie a une action, la methode ne fera rien.
       * Les actions disponibles sont:
       * - Lancement d'un nouveau combat [1]
       * - Affichage de l'historique d'un combat [2]
       * - Lancement du menu statistiques [3]
       *
       * @param   action      Un numéro correspondant a une action
       * @return  void
       * @throws  DeadException
       */
      private void executeAction(int action) throws DeadException {

            if(action == BATTLE_ACTION) {
                  new BattleHandler(dbConnection, userID);
            }
            else if(action == HISTORY_ACTION) {
                  new HistoryHandler(dbConnection, userID);
            }
            else if(action == STATS_ACTION) {
                  new StatsHandler(dbConnection, userID);
            }
            else if(action == JACKPOT_ACTION) {
                  new JackpotHandler(dbConnection, userID);
            }
      }

      /**
       * Ferme l'application
       *
       * Coupe la connection au serveur de bases de donnees et termine
l'application en
       * renvoyant un code 0 pour signifier que l'application s'est fermee sans
problemes.
       * En cas de problemes lors de deconnection au serveur, l'application se
terminera
       * avec un code 1.
       *
       * @return  void
       */
      private void closeApp() {

            try {
                  this.dbConnection.close();
            }
            catch (SQLException e) {
                  System.exit(1);
            }
```

```
            System.out.println("\nFermeture de l'application...");
            System.exit(0);
      }

      // LANCEMENT DE L'APPLICATION POWER-MANGEURS

      public static void main(String[] args) {

            new App();
      }

}
```

```java
package terminalPM;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Date;
import java.util.Scanner;

public class BattleHandler {

	private static final int BATTLETIME = 5;
	private static final int DEFAULTPOWER = 30;

	private static final int ACTION_CLOSE_BATTLE = 1;
	private static final int ACTION_USE_POWERUP = 2;

	private static Scanner scanner = new Scanner(System.in);

	private Connection dbConnection;
	private int userID;

	private long beginTimestamp;

	private int monsterID;
	private String monsterName;
	private int monsterPower;

	public BattleHandler(Connection connection, int userID) throws
DeadException {

		this.dbConnection = connection;
		this.userID = userID;

		if( setRandomMonster() == false ) {
			System.out.println("Yeah, aucun monstre a combattre pour
l'instant...");
			return;
		}

		if( startNewBattle() == false ) {
			System.out.println("Impossible de commencer un combat\nUn
combat est peut-etre deja en cours...");
			return;
		}

		executeBattle();
	}

	public BattleHandler(Connection connection, int userID, int battleID)
throws DeadException {

		this.dbConnection = connection;
		this.userID = userID;

		try {
			PreparedStatement ps = dbConnection.prepareStatement("SELECT
* FROM projet.combats WHERE id_combat=? AND date_fin IS NULL");
			ps.setInt(1, battleID);
			ResultSet rs = ps.executeQuery();

			if( !rs.next() )
				return;
```

```
                  this.beginTimestamp =
(rs.getTimestamp("date_debut")).getTime();
                  this.monsterID = rs.getInt("id_archetype");
            }
            catch(SQLException e) {
                  System.out.println( e.getMessage() );
                  return;
            }

            if( isTimeOver() ) {
                  closeBattle(true);
                  return;
            }

            executeBattle();
      }

      private void executeBattle() throws DeadException {

            int choice = -1;
            while(choice != ACTION_CLOSE_BATTLE) {

                  choice = displayMainMenu();

                  if( isTimeOver() ) {
                        this.closeBattle(true);
                        return;
                  }

                  if(choice == ACTION_USE_POWERUP) {
                        this.usePowerUp();
                  }
            }
            this.closeBattle(false);

            if( this.isDead() )
                  throw new DeadException();
      }

      private boolean isTimeOver() {

            long diff = ( (new Date()).getTime() - this.beginTimestamp )/1000;
            return ( diff > BATTLETIME*60 );
      }

      private boolean setRandomMonster() {

            try {

                  PreparedStatement ps = dbConnection.prepareStatement("SELECT
* FROM projet.monstre_au_hasard");
                  ResultSet rs = ps.executeQuery();

                  if( !rs.next() )
                        return false;

                  this.monsterID = rs.getInt("id_archetype");
                  this.monsterName = rs.getString("nom");
                  this.monsterPower = rs.getInt("puissance");

                  System.out.println("\nLancement d'un combat contre...\nun
monstre "+monsterName+" avec puissance "+monsterPower+"!");
```

```
            }
            catch(SQLException e) {

                  e.printStackTrace();
                  return false;
            }

            return true;
      }

      private boolean startNewBattle() {

            try {
                  PreparedStatement ps = dbConnection.prepareStatement("SELECT
* FROM projet.debuter_combat(?,?)");
                  ps.setInt(1, this.userID);
                  ps.setInt(2, this.monsterID);
                  ResultSet rs = ps.executeQuery();

                  rs.next();
                  this.beginTimestamp = ( new Date() ).getTime();

                  System.out.println("\nTu as une puissance "+DEFAULTPOWER);
            }
            catch(SQLException e) {

                  e.printStackTrace();
                  return false;
            }

            return true;
      }

      private int displayMainMenu() {

            System.out.println("\nMenu Combat");
            System.out.println("-----------\n");

            System.out.println("#1\tConclure combat");
            System.out.println("#2\tUtiliser Power-Up");

            System.out.println("\nChoix :");
            return scanner.nextInt();
      }

      private void closeBattle(boolean forceDeath) {

            if( forceDeath ) {
                  System.out.println("\nDuree maximale du combat depassee!");
            }

            try {
                  PreparedStatement ps = dbConnection.prepareStatement("SELECT
* FROM projet.conclure_combat(?,?)");
                  ps.setInt(1, userID);
                  if( forceDeath ) {
                        ps.setBoolean(2, true);
                  } else {
                        ps.setBoolean(2, false);
                  }
                  ResultSet rs = ps.executeQuery();

                  rs.next();
```

```
                  if( rs.getBoolean(1) == true )
                        System.out.println("\nLe combat est GAGNE!");
                  else
                        System.out.println("\nLe combat est PERDU...");
            }
            catch(SQLException e) {

                  System.out.println( e.getMessage() );
            }
      }

      private void usePowerUp() {

            try {
                  PreparedStatement ps = dbConnection.prepareStatement("SELECT
* FROM projet.power_ups WHERE id_pm = ?");
                  ps.setInt(1, userID);
                  ResultSet rs = ps.executeQuery();

                  if( !rs.next() ) {
                        System.out.println("\nDesole, aucun Power-Up
disponible");
                        return;
                  }

                  System.out.println("\nChoisir un Power-Up");
                  System.out.println("-------------------\n");

                  do {
                        int ID = rs.getInt("id_pu");
                        String name = rs.getString("nom");

                        System.out.println("#"+ID+"\t"+name);
                  }
                  while( rs.next() );
                  System.out.println("#0\tAnnuler");

                  System.out.println("\nChoix : ");
                  int choice = scanner.nextInt();

                  if( choice == 0 )
                        return;

                  ps = dbConnection.prepareStatement("SELECT * FROM
projet.utiliser_pu(?, ?)");
                  ps.setInt(1, userID);
                  ps.setInt(2, choice);
                  rs = ps.executeQuery();

                  rs.next();
                  System.out.println("\nTu as maintenant une puissance
"+rs.getInt(1));
            }
            catch(SQLException e) {

                  System.out.println("\n"+ e.getMessage() );
            }
      }

      private boolean isDead() {

            try {
```

```
                  PreparedStatement ps = dbConnection.prepareStatement("SELECT
vie FROM projet.power_mangeurs WHERE id_pm = ?");
                  ps.setInt(1, userID);
                  ResultSet rs = ps.executeQuery();

                  if( !rs.next() || rs.getInt(1) <= 0 )
                        return true;
            }
            catch(SQLException e) {

                  e.printStackTrace();
            }

            return false;
      }
}
```

```java
package terminalPM;

import java.math.BigInteger;
import java.security.GeneralSecurityException;
import java.security.InvalidParameterException;

import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;

public class CryptService {

	private static final int ITERATIONS = 1000;
	private static final String SALT = "FE26EEE87B528135";
	private static final int KEYLENGTH = 64*8;
	private static final String CIPHER = "PBKDF2WithHmacSHA1";

	public static String hash(String s) {

		if( s == null )
			throw new InvalidParameterException();

		PBEKeySpec spec = new PBEKeySpec(s.toCharArray(), SALT.getBytes(),
ITERATIONS, KEYLENGTH);

		byte[] hash;
		try {
			hash =
SecretKeyFactory.getInstance(CIPHER).generateSecret(spec).getEncoded();

		}
		catch(GeneralSecurityException e) { return null; }

		return toHex( hash );
	}

	private static String toHex(byte[] array)
    {
        String hex = ( new BigInteger(1, array) ).toString(16);
        int paddingLength = (array.length * 2) - hex.length();

        if(paddingLength > 0)
            return String.format("%0"  +paddingLength + "d", 0) + hex;

        return hex;
    }

}


package terminalPM;

@SuppressWarnings("serial")
public class DeadException extends Exception {

	public DeadException() {}

}
```

```java
package terminalPM;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;

public class HistoryHandler {

	private Connection dbConnection;
	private int userID;

	public HistoryHandler(Connection connection, int userID) {

		this.dbConnection = connection;
		this.userID = userID;

		launch();
	}

	private void launch() {

		try {

			PreparedStatement ps = dbConnection.prepareStatement("SELECT
* FROM projet.visualiser_combat(?)");
			ps.setInt(1, userID);
			ResultSet rs = ps.executeQuery();

			System.out.println("\nHistorique Dernier Combat");
			System.out.println("-------------------------\n");

			if( !rs.next() ) {
				System.out.println("Aucun historique disponible");
				return;
			}

			do {
				String action = rs.getString("action");
				Timestamp timestamp = rs.getTimestamp("date");

				System.out.println(timestamp+"\t"+action);
			}
			while( rs.next() );
		}
		catch (SQLException e) {
			System.out.println( e.getMessage() );
		}
	}
}
```

```java
package terminalPM;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Random;
import java.util.concurrent.TimeUnit;

public class JackpotHandler {

	private static final String DIAMONDS = "\u2666";
	private static final String CLUBS = "\u2663";
	private static final String SPADES = "\u2660";

	private Connection dbConnection;
	private int userID;

	public JackpotHandler(Connection connection, int userID) {

		this.dbConnection = connection;
		this.userID = userID;

		launch();
	}

	private void launch() {

		System.out.println("\nJackPot");
		System.out.println("-------\n");

		System.out.println("Si les 3 symboles correspondent,\nvous gagnez
une vie supplementaire.");
		System.out.println("Uniquement valable 5min. apres un combat.\n");

		boolean jackpotResult = this.runJackpot();

		if( jackpotResult == false ) {
			System.out.println("\n\nDesole, vous ne gagnez rien...");
			return;
		}

		int currentLives = collectJackpot();

		if(currentLives != -1) {

			System.out.println("\nBingo, vous gagnez une vie
supplementaire.");
			System.out.println("Vous avez donc maintenant "+currentLives
+" vies.\n");

			for(int i=0; i<currentLives; i++)
				System.out.print(LoginHandler.HEARTICON +" ");
		}

		System.out.println("");
	}

	private boolean runJackpot() {

		String s1 = generateRandomSymbol(), s2 = generateRandomSymbol(), s3
= generateRandomSymbol();
```

```
            try {
                  TimeUnit.SECONDS.sleep(1);
                  System.out.print(s1);

                  TimeUnit.SECONDS.sleep(1);
                  System.out.print(s2);

                  TimeUnit.SECONDS.sleep(1);
                  System.out.print(s3);

                  TimeUnit.SECONDS.sleep(1);
            }
            catch (InterruptedException e) {
                  e.printStackTrace();
            }

            return ( s1.equals(s2) && s1.equals(s3) );
      }

      private int collectJackpot() {

            try {

                  PreparedStatement ps = dbConnection.prepareStatement("SELECT
* FROM projet.encaisser_jackpot(?);");
                  ps.setInt(1, userID);
                  ResultSet rs = ps.executeQuery();
                  rs.next();

                  return rs.getInt(1);
            }
            catch(SQLException e) {
                  System.out.println(e.getMessage());
            }

            return -1;
      }

      private String generateRandomSymbol() {

          int randomNum = (new Random()).nextInt(3)+1;

          if( randomNum == 1 )
            return DIAMONDS;

          if( randomNum == 2 )
            return CLUBS;

          return SPADES;
      }

}
```

```java
package terminalPM;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;

public class LoginHandler {

	public static final String HEARTICON = "\u2764";

	private Connection dbConnection;
	private Scanner scan;

	private int userID;
	private String hashedPassword;
	private int lives;

	public LoginHandler(Connection c) {

		this.dbConnection = c;

		this.scan = new Scanner(System.in);
		scan.useDelimiter("\\n");
	}

	public int login() {

		System.out.println("\nIdentification requise");
		System.out.println("----------------------\n");

		while( this.userID == 0 || this.lives <= 0 ) {

			System.out.print("Nom: ");
			String name = scan.next();

			retrieveUserDate(name);

			if( this.userID == 0 || this.lives == 0 )
				System.out.println("Desole, ce Power Mangeur est mort
ou n'existe pas\n");
		}

		while(true) {

			System.out.print("\nMot de passe: ");
			String tempPass = scan.next();

			String hash = CryptService.hash(tempPass);

			if( hashedPassword != null && hash.equals(hashedPassword) )
				break;

			System.out.println("Desole, mot de passe incorrect");
		}

		System.out.println("Mot de passe correct");

		System.out.print("\nVies : ");
		for(int i=0; i<this.lives; i++)
			System.out.print(HEARTICON +" ");
		System.out.println("");
```

```
            return this.userID;
      }

      private void retrieveUserDate(String name) {

            if( name == null || name == "" )
                  return;

            try {

                  PreparedStatement ps = dbConnection.prepareStatement("SELECT
* FROM projet.power_mangeurs WHERE nom=?;");
                  ps.setString(1, name);
                  ResultSet r = ps.executeQuery();

                  if( r.next() ) {

                        this.userID = r.getInt("id_pm");
                        this.hashedPassword = r.getString("mot_de_passe");
                        this.lives = r.getInt("vie");
                  }
            }
            catch(SQLException e) {
                  e.printStackTrace();
            }
      }

}
```

```java
package terminalPM;

import java.sql.*;
import java.util.Scanner;

public class StatsHandler {

	private Connection dbConnection;
	private int userID;
	private Scanner scanner = new Scanner(System.in);

	public StatsHandler(Connection c, int ID) {

		this.dbConnection = c;
		this.userID = ID;

		launch();
	}

	private void launch() {

		int action = -1;

		while( action != 0 ) {

			action = this.showMenu();

			System.out.println("");

			if( action == 1 )
				showStatsMonster();

			else if( action == 2 )
				showStatsLife();

			else if( action == 3 )
				showPowerUpHistory();

			else if( action != 0 )
				System.out.println("Cette option n'existe pas");
		}
	}

	/**
	 * Affiche le menu des statistiques
	 *
	 * @return  int         L'identifiant de l'action choisie
	 */
	private int showMenu() {

		System.out.println("\nStatistiques");
		System.out.println("------------\n");

		System.out.println("#1\tStatistiques des combats");
		System.out.println("#2\tEsperance de vie");
		System.out.println("#3\tHistorique Power Ups");
		System.out.println("#0\tRetour");

		System.out.print("\nChoix: ");

		return scanner.nextInt();
	}
```

```
	/**
	 * Affiche les statistiques sur le nombre de combats
	 *
	 * @return  void
	 */
	private void showStatsMonster() {

		System.out.println("\nStatistiques des combats");
		System.out.println("------------------------\n");

		try {
			PreparedStatement ps = dbConnection.prepareStatement("SELECT
* FROM projet.stats_pm(?)");
			ps.setInt(1, userID);
			ResultSet rs = ps.executeQuery();

			if( !rs.next() ) {
				System.out.println("Aucune statistique disponible");
				return;
			}

			do {

				System.out.println("- Monstre "+
rs.getString("nom_archetype") );
				System.out.println("\t- Total combats\t\t"+
rs.getInt("nb_combats_total") );
				System.out.println("\t- Total victoires\t"+
rs.getInt("nb_victoires_total") );
				System.out.println("\t- Combats annee\t\t"+
rs.getInt("nb_combats_annee") );
				System.out.println("\t- Victoires annee\t"+
rs.getInt("nb_victoires_annee") );
			}
			while( rs.next() );
		}
		catch (SQLException e) {
			e.printStackTrace();
		}
	}

	/**
	 * Affiche l'esperance de vie du Power Mangeur
	 *
	 * L'esperance de vie est calculee...
	 *
	 * @return  void
	 */
	private void showStatsLife() {

		System.out.println("\nEsperance de vie");
		System.out.println("----------------\n");

		try {
			PreparedStatement ps = dbConnection.prepareStatement("SELECT
* FROM projet.esperance_vie(?)");
			ps.setInt(1, userID);
			ResultSet rs = ps.executeQuery();
			rs.next();

			String e = rs.getString("esperance_vie");

			if( e.equals("00:00:00") )
```

```
                        System.out.println("Esperance de vie indeterminee");
                  else
                        System.out.println( rs.getString("esperance_vie") );
            }
            catch (SQLException e) {
                  e.printStackTrace();
            }
      }


      private void showPowerUpHistory() {

            System.out.println("\nHistorique des Power Up");
            System.out.println("----------------------\n");

            try {
                  PreparedStatement ps = dbConnection.prepareStatement("SELECT
* FROM projet.historique_pu WHERE id_pm = ?;");
                  ps.setInt(1, userID);
                  ResultSet rs = ps.executeQuery();

                  if( !rs.next() ) {
                        System.out.println("Aucun historique d'utilisation de
Power Up");
                        return;
                  }

                  do {

                        String name = rs.getString("nom");
                        int f = rs.getInt("facteur");
                        Timestamp timestamp =
rs.getTimestamp("date_utilisation");

                        System.out.println(name +"  [+"+ f +"%]  "+ timestamp);
                  }
                  while( rs.next() );
            }
            catch(SQLException e) {
                  e.printStackTrace();
            }
      }

}
```

Conclusion

Nous estimons à 25-30 heures le temps que nous avons travaillé sur ce projet. Ce temps étant principalement puisé sur les heures de séances prévues explicitement pour cela.
Nous avons tout deux appris un langage dont nous ne soupçonnions pas l'existence, ou, tout du moins, l'utilité.
Etant de même niveau, nous n'avons eu aucun problème de coordination et nous avons pû travailler à notre maximum sans laisser quelqu'un derrière.
En conclusion, la réalisation de ce projet nous a plû et nous a permis de nous instruire dans un domaine inconnu.