Table des matières

I. Game Design Document

Prodigium *(IPO 2020/2021 G8)*

PRÉSENTATION GÉNÉRALE

Type de jeu

Aventure, semblable à Pokémon, ou encore Temtem.

Thème du jeu

Un enfant de 13 ans doit venger sa famille avec l'aide de créatures mystiques, tout en libérant le monde de l'emprise d'une multinationale (Tenebris) qui plongent le monde dans une ère sombre (au sens propre, il n'y a plus de lumière naturelle à cause de la pollution) depuis des siècles.

HISTOIRE ET GAMEPLAY

Univers du jeu

Le protagoniste (Ray Luminis) évolue dans une contrée lointaine du nom de Terrafernum (contraction du latin “d'Enfer", “Infernum”, et du latin de “Terre”, “Terra”). Dans cette contrée, l’exploitation des êtres vivants de toutes catégories est normale. Malgré quelques révoltes au cours de son histoire, cette région de l’univers est en paix. Les créatures mystiques qui la peuplent y sont réduit en esclavage, et leurs pouvoirs sont exploités jusqu’à la mort de la créature.

Orphelin depuis ses 8 ans, il vit avec des créatures mystiques aux pouvoirs variés, les Prodigiums (du latin “monstre”). Ses parents sont morts, exploités par une multinationale, tout comme leurs parents. Après s’être enfui de chez lui sur les conseils de ses parents, il s’est réfugié avec des Prodigiums dans une grotte inconnue de tous. Il a ensuite été intégré à la société des Prodigiums. A l’âge de ses 13 ans, il décide de partir venger sa famille, et libérez tous les Prodigiums, en asservissant tous ceux qui lui bloquent le chemin. Pour cela, il peut compter sur l’aide des créatures mystiques qui l'ont recueilli il y a 5 ans.

L’enfant (le joueur) peut sembler « extrémiste » dans un premier temps, mais il ne faut pas oublier que ce monde est littéralement le reflet malsain de notre monde actuel amplifié. Esclavage, meurtre, torture, et d’autres choses horribles sont inscrites dans leur mœurs.

Le joueur

Le joueur, nommé “Ray Luminis” (en latin “rayon de lumière”), n’aura pas de pouvoir particulier ou de caractéristiques surhumaines. Ce qui le rend unique c’est son lien avec les Prodigiums, qui sont près à mourir pour sa cause. Le joueur aura donc une équipe de Prodigiums pouvant être composée de 10 créatures maximum. Le joueur possédera également un inventaire pouvant contenir des baies, avec une limite de capacité de 10 baies de chaque type, 5 pour les plus rares. Il possède également une intégrité morale, qu’il pourra perdre ou gagner.

Prodigiums

Les Prodigiums sont des créatures mystiques possédant différents pouvoirs. Ils peuvent être répartie en “Type”, et en catégorie. Chaque Prodigium aura 4 attaques maximums, qu’il débloque en fonction de son niveau d’expérience. Plus le Prodigium utilise des attaques, plus sa stamina diminue : elle est régénérée à chaque fin de combat.

- **Type**

  Les “Types” sont des catégories qui classent les Prodigiums en fonction de leurs pouvoirs. Les “types” sont : Eau, Terre, Feu et Air. De plus, chaque type à un boost de

dégât contre un autre type : par exemple, l'eau à l'avantage sur le feu. L'inverse existe aussi : le type feu à un malus de dégâts sur le type eau. Ce phénomène n'est applicable qu'aux attaques.

- **Attaques**

  Chaque attaque possède également un "Type". Elle a certaines propriétés comme : son taux de dégâts, et son coût en stamina. Elle peut être lancée par un Prodigium lors d'un combat.

- **Vie**

  Il est impossible de soigner un Prodigium autrement qu'avec des baies. Un Prodigium mort au combat ne peut pas être ramené à la vie après le combat. C'est uniquement lors de sa mort, que le joueur peut choisir d'utiliser une baie spéciale pour le ramener à la vie. Rien ne l'empêche cependant, de remourir durant le même combat.

- **Remerciements**

  Vous pouvez remercier un Prodigium de votre équipe pour le laisser partir vivre sa vie libre. Il quitte ainsi votre équipe et libère une place.

Baies

Les baies peuvent être trouvées dans chaque lieu extérieur que le joueur visite. Il existe plusieurs types de baies, chacune ayant un niveau de rareté différent.

- **Baie de soin** `( Rareté : Commune (1) )`

  La baie de soin permet de soigner un Prodigium pendant ou après un combat. Elle restaure 25% de la vie maximale du Prodigium.

- **Baie boostante** `( Rareté : Commune (1) )`

  Cette baie permet de régénérer la stamina d'un Prodigium de 30%.

- **Baie divine** `( Rareté : Mythique (3) )`

  La baie divine permet au joueur de ressusciter un Prodigium venant de mourir au combat.

Rencontre

Le joueur peut rencontrer des Prodigiums lors de son aventure. Il en existe deux types : les Prodigiums Libres et les Prodigiums Asservis

- **Libres**

  Les Prodigiums Libres peuvent se joindre à votre cause. Il faudra cependant les convaincre que votre cause est juste. Attention : vous n'avez qu'une chance pour les convaincre, sinon, ils ne vous rejoindront pas.

Pour cela, vous aurez deux choix possibles : dire la vérité (chance de réussite faible mais vous gardez votre intégrité morale), mentir (chance de réussite élevée mais vous perdez un peu de votre intégrité morale). Tout sera une question d'équilibre.

Plus le Prodigium Libre a une vie élevée, plus il sera difficile à convaincre. Et inversement.

Vous ne pouvez pas rencontrer de Prodigiums Libres si votre équipe est déjà composée de 10 Prodigiums.

- **Asservis**

  Le joueur peut uniquement les combattre. Ces derniers ont été réduit en esclavage et ont subi un "lavage de cerveau", il est donc impossible de les sauver. Combat de type 1v1, avec un unique Prodigium adverse.

Le joueur aura aussi la possibilité de rencontrer des "Gardiens". Chaque Gardien, tout comme le joueur, possède une équipe de Prodigiums. Il faudra donc que le joueur se batte contre chacun des Prodigiums adverses les uns à la suite des autres.

De même il existera des "Légendes", similaires aux Gardiens mais avec une équipe plus grande et un niveau plus élevé.

Combat

Un combat peut être déclenché quand le joueur rencontre un Prodigium Asservis, un Gardiens, ou encore une Légende. Chaque combat opposera un Prodigium allié contre un Prodigium adverse (1 vs 1). Cependant, on remarque quelques spécificités :

- Les combats se déroulent en plusieurs phases : choix de l'attaque, exécution de l'attaque allié, exécution de l'attaque ennemie. Tout cet enchaînement d'action est appelé un "tour".
- A la fin d'un combat avec un Gardien (ou une Légende), le joueur pourra choisir d'exécuter le Gardien (ou la Légende), perdant ainsi énormément d'intégrité morale, ou de le (la) laisser en vie, gardant ainsi son intégrité morale.

  Cette fonctionnalité est superflue, mais cela renforce le côté malsain de l'univers du jeu.

- Le joueur aura la possibilité de changer de Prodigium durant le combat, utilisant ainsi un tour de combat. De même, il pourra utiliser des baies durant le combat, en appliquant ainsi ses effets au Prodigium actuellement sur le terrain.

Fin du jeu

Il y a Victoire quand le joueur libère tous les Prodigiums et venge sa famille en libérant le monde de l'emprise de Tenebris.

Il y a Défaite quand le joueur fait trop de mauvaises décisions, le but n'est pas qu'ils remplacent ceux contre qui il se bat ; ou quand le joueur est vaincu au combat, il n'y a pas de seconde chance.

UNIVERS GRAPHIQUE

Graphisme des environnements

Les différents lieux seront la totalité du temps en extérieur. Le design sera en pixel art, sans trop de détails. Ils seront cependant comparables aux “vrais” éléments extérieurs dans la forme : les arbres ressembleront à des arbres. Cependant, leurs couleurs changeront : les feuilles des arbres pourront alors être rouges, ou violettes, etc.

Graphisme des personnages et des Prodigiums

Toujours en pixel art mais en cubique. Des personnages et Prodigiums à l’allure de Steve (Minecraft) mais en 2D. Les couleurs des habits que les personnages portent pourront varier. Les Prodigiums ne portent aucun habit, et leurs couleurs seront décidées en fonction de leur type.

Interface

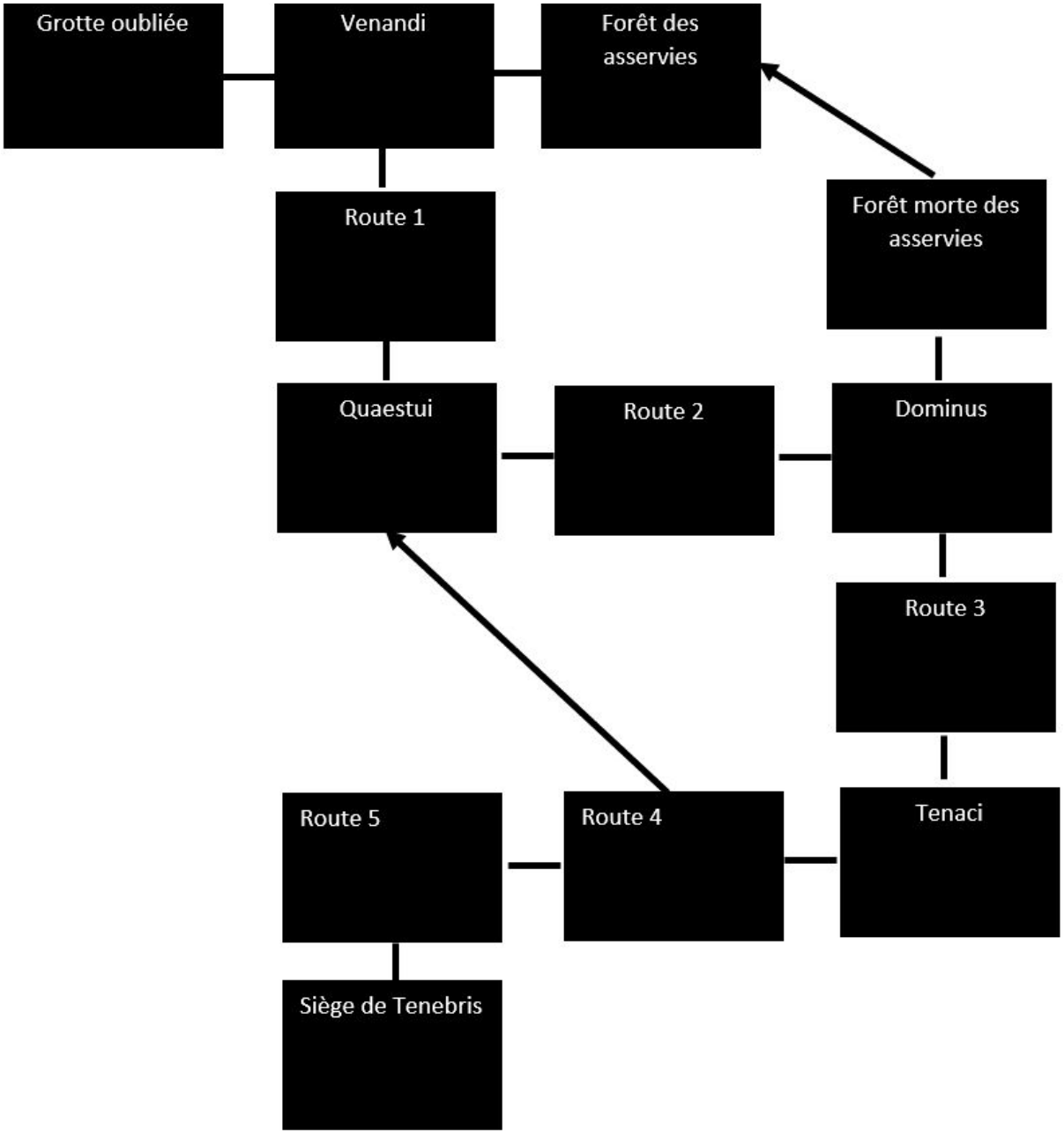
Une interface aventure, où on voit le contenu de notre sac à dos, et le nombre de Prodigiums dans notre équipe.

Une interface de combat, où on voit le nombre de points de vie et de points de stamina de notre Prodigium et du Prodigium adverse, sous forme de barre verte (vie) et jaune/orange (stamina).

Une interface de négociation, où on voit la vie du Prodigium Libre, et où on peut choisir de lui dire la vérité ou de lui mentir.

Une zone de texte, qui retracera tous les événements causés ou subis par le joueur, ou son équipe.

CARTE DU JEU



Grotte oubliée : Une grotte banale d'apparence mais qui renferme toute une société de Prodigiums. J'y ai passé mes 5 dernières années en leur compagnie. Il est maintenant temps que je parte à l'aventure et que je me venge !

Venandi : On dirait un petit village, principalement constitué de chasseurs de Prodigiums. L'odeur de la chair des Prodigiums en putréfaction est insoutenable !

Forêt des asservies : *vous lisez un panneau à l'entrée* "Forêt des asservies : défense d'entrer !" Une forêt magnifique...ment glauque… On dirait bien que c'est le terrain de chasse des gens du village que l'on vient de traverser...

Route 1 : C'est la route qui relie Venandi à Quaesti.

Quaesti : *tousse* L'air est irrespirable ici ! ça pourrait expliquer pourquoi tout le monde a l'air malade et pauvre dans le coin. C'est certainement une ville d'esclaves. Je ne devrais pas trop m'attarder par ici…

Route 2 : On dirait bien que c'est la route entre Quaesti et Dominus.

Dominus : Ah… L'air est beaucoup plus respirable dans le coin ! Cette ville là, a l'air beaucoup plus riche que celle d'avant… Peut-être est-ce la ville des maîtres ? Si c'est le cas, je devrais trouver pas mal de Prodigiums à sauver !

Forêt morte des asservis : On dirait comme une partie de la Forêt des asservies… mais elle est complètement déserte et morte… C'était peut-être leur ancien terrain de chasse ?

Route 3 : Voilà la route qui relie Dominus à Tenaci !

Tenaci : On dirait bien que ce n'est pas réellement une ville ! C'est plus une sorte de zone industrielle. Il faut faire cesser l'exploitation des Prodigiums dans le coin ! Je ne repartirai pas tant qu'ils ne seront pas tous libérés.

Route 4 : La route qui précède la route 5, il se pourrait bien que je puisse retourner à Quaestui en l'empruntant.

Route 5 : Enfin ! Je sens que je touche au but ! C'est cette route qui mène jusqu'au siège de Tenebris ! Ils feraient mieux de se préparer à ma visite...

Siège de Tenebris : On dirait bien que ma Révolution est en marche ! Et ce n'est que le début… Je ne peux plus faire marche arrière !

II. Rapport

Prodigium *(IPO 2020/2021 G8)*

Exercice 7.5

Afin de mener à bien cet exercice j'ai créé une procédure printLocationInfo(), dans la classe Game, qui affiche le nom et la description de la salle actuelle, ainsi que les différentes directions disponibles. Cela m'a permis de remplacer les instructions qui effectuaient la même chose dans goRoom() et printWelcome(), évitant ainsi la duplication de code (l'ennemie n°1).

Exercice 7.6

Dans cet exercice, j'ai ajouté une fonction, avec le type de retour Room, me permettant de "prendre" la Room correspondant à la sortie choisie grâce au paramètre pDirection (une String). Cela permet notamment de rendre les attributs, correspondants aux sorties, privés.

Exercice 7.7

Cet exercice a été réalisé en créant une nouvelle fonction getExitString(), dans la classe Room, permettant d'obtenir une String contenant les sorties disponibles. Ainsi, j'ai dû modifier la méthode printLocationInfo(), de la classe Game, afin d'éviter la duplication de code, en appelant la fonction nouvellement créée sur aCurrentRoom, et en y affichant son résultat.

Exercice 7.8

Dans cet exercice, et dans la classe Room, j'ai créé une HashMap<String, Room> afin d'y stocker les sorties possibles. Par conséquent, j'ai supprimé les quatre attributs de type Room représentant les sorties (= aNorthExit, …), et mis à jour le constructeur afin que cette HashMap ai une valeur par défaut. De plus, j'ai supprimé setExits(....), étant devenu obsolète suite à l'ajout de setExit(..). Ensuite, j'ai modifié la méthode getExitString(), afin qu'elle puisse afficher n'importe quelle direction disponible dans la HashMap. Pareil pour la fonction getExit(.), qui retourne maintenant la valeur, stockée dans la HashMap, associée à la direction. J'ai également dû changer la méthode createRooms() dans la classe Game, afin de la rendre compatible avec ce nouveau système.

J'ai également dû ajouter une pièce, afin de pouvoir se déplacer verticalement. Pour cela , j'ai modifié la méthode createRooms() dans la classe Game, et utilisé la méthode setExit(..) de la classe Room, sur une des pièces créées.

```java
private HashMap<String, Room> aExits;

/**
 * Create a Room object
 * @param pName the name of the room
 * @param pDescription the description of the room
 */
public Room( final String pName, final String pDescription )
{
    this.aName = pName;
    this.aDescription = pDescription;
    this.aExits = new HashMap<String, Room>();
} // Room(..)
```

```java
/**
 * Set an exit in a specific direction
 * @param pDirection direction of the exit
 * @param pRoom the exit
 */
public void setExit( final String pDirection, final Room pRoom )
{
    if(pRoom != null){
        this.aExits.put( pDirection, pRoom );
    }
} // setExit(..)
```

```java
/**
 * Get exit from a direction
 * @param pDirection exit direction
 * @return the exit room or null
 */
public Room getExit ( final String pDirection )
{
    return this.aExits.get(pDirection);
} // getExit(.)
```

Exercice 7.9

Extrait de l'exercice 7.8 : "De même, j'ai modifié la méthode getExitString(), afin qu'elle puisse afficher n'importe quelle direction disponible dans la HashMap.". Pour cela j'ai dû utiliser une boucle for each, afin de boucler à travers le set de clé (=toutes les clé) de ma HashMap, et ainsi ajouter toutes les sorties disponibles dans ma HashMap à ma String servant de valeur de renvoi.

```java
/**
 * Show every direction available from this room
 * @return the serialized String with every exits
 */
public String getExitString ()
{
    String vExit = "Vous voyez différents chemins :\n";
    Set<String> vKeys = this.aExits.keySet();

    for ( String key : vKeys ){
        vExit += key + " : " + this.aExits.get(key).getName() + "\n";
    }

    return vExit;
} // showDirections()
```

Exercice 7.10

Dans cet exercice, je n'ai pas eu beaucoup de choses à faire, étant donné que je maintiens la javadoc à jour à chaque exercice. J'ai donc seulement corrigé certaines fautes d'orthographe et généré la javadoc.

Exercice 7.11

Afin de mener à bien cet exercice, j'ai dû créer une nouvelle fonction, getLongDescription(), qui retourne une String construite sous la forme : nom de la pièce (retour à la ligne) + description de la pièce (retour à la ligne) + sorties disponibles. Afin de profiter de cette nouvelle fonction, et en évitant ainsi la duplication de code, j'ai modifié la classe Game, et plus précisément la méthode printLocationInfo(), afin qu'elle utilise la fonction nouvellement créée, en affichant juste son résultat.

```
/**
 * Get the long description of the current room
 * It's composed of the name, the description and the exits of the room
 * @return the room's long description
 */
public String getLongDescription()
{
    String vLongDescription = "***** " + this.getName() + " *****\n";
    vLongDescription += this.getDescription() + "\n";
    vLongDescription += this.getExitString() + "\n";

    return vLongDescription;
} // getLongDescription()
```

```
/**
 * Show every direction available from the current room
 */
private void printLocationInfo()
{
    System.out.println( this.aCurrentRoom.getLongDescription() );
} // printLocationInfo()
```

Exercice 7.14

Dans cet exercice j'ai ajouté la commande "regarder". Pour cela, j'ai dû ajouter le mot "regarder" dans le tableau aValidCommands, de la classe CommandWords, afin que cette nouvelle commande soit reconnue par le Parser. De plus, j'ai dû créer une nouvelle méthode dans la classe Game, afin d'effectuer une action lorsque cette commande est exécutée. J'ai également ajouté un "else if(...)" dans la procédure processCommand(.) afin de pouvoir exécuter la méthode correspondant à la commande "regarder".

```
else if(pCommand.getCommandWord().equals("regarder")){
    this.look();
}
```

```
/**
 * Method associated with the look command
 */
private void look()
{
    this.printLocationInfo();
} // look()
```

Exercice 7.15

Similaire à l'exercice 7.14, mais avec la commande "manger" et la création de la méthode eat().

```
/**
 * Method associated with the eat command
 */
private void eat()
{
    System.out.println("Vous venez de manger. Vous êtes rassasié.");
} // eat()
```

Exercice 7.16

Pour cet exercice, j'ai créé une méthode dans la classe CommandWords, nommée showAll(), qui affiche toutes les commandes disponibles. J'ai ensuite créé une procédure dans la classe Parser, afin d'appeler la méthode showAll() via l'attribut aValidCommands de la classe Parser. Enfin, j'ai mis à jour la méthode printHelp() de la classe Game, afin qu'elle se serve de la procédure précédemment créée.

```
/**
 * Show every available commands
 */
public void showAll()
{
    for(String vCmd : this.aValidCommands){
        System.out.print(vCmd + " ");
    }

    System.out.println();
} // showAll()
```

Exercice 7.18.0

Pour cet exercice j'ai dû modifier la classe CommandWords, et plus précisément la méthode showAll(), que j'ai transformée en fonction, getCommandList(), qui retourne une String contenant la liste des commandes. J'ai alors été obligé de modifier la méthode showCommands() de la

classe Parser, la transformant ainsi en fonction, getCommandList(), retournant une String contenant le résultat de la fonction getCommandList(), décrite ci-dessus. J'ai ensuite modifié la classe Game, et plus particulièrement la méthode printHelp(), afin d'afficher le résultat de la fonction getCommandList() de la classe Parser.

```
/**
 * Get every available commands
 * @return the command list
 */
public String getCommandList ()
{
    String vCmds = "";

    for(String vCmd : this.aValidCommands){
        vCmds += vCmd + " ";
    }

    return vCmds;
} // getCommandList()
```

Exercice 7.18.1

Dans cet exercice, j'ai uniquement regardé chaque classe du projet zuul-better, et j'ai effectué des petites modifications, afin de me rapprocher du projet zuul-better.

Exercice 7.18.4

Le titre du jeu sera le nom des créatures mystiques, soit : "Prodigium".

Exercice 7.18.5

Dans cet exercice j'ai créé une HashMap<String, Room> me permettant de stocker chaque Room à sa création. Pour cela, j'ai également modifié le constructeur de la classe Room en ajoutant un paramètre, de type Game, puis en appelant dans ce constructeur la méthode addRoom() de la classe Game, sur l'objet Game passé en paramètre, afin d'ajouter la Room actuelle (this) à notre HashMap aRooms.

```java
/**
 * Add a room to the map
 * @param pRoom room to add
 */
public void addRoom ( final Room pRoom )
{
    this.aRooms.put( pRoom.getName(), pRoom );
} // addRoom(.)
```

```java
/**
 * Create a Room object
 * @param pGame the game instance
 * @param pName the name of the room
 * @param pDescription the description of the room
 */
public Room( final Game pGame, final String pName, final String pDescription )
{
    this.aName = pName;
    this.aDescription = pDescription;
    this.aExits = new HashMap<String, Room>();
    pGame.addRoom(this);
} // Room(..)
```