Titre provisoire + lien page web

- Titre provisoire : *La Roue des Saisons*
- Lien page web :

I.A) Auteur(s)

Jeu réalisé par : Simon ROPIOT (Tristan → dans le jeu seulement).
Étudiant en 1re année à l'ESIEE Paris, unité IPO

I.B) Thème (phrase-thème validée)

Dans un royaume médiéval-fantastique où les saisons ont disparu : Tristan, apprenti herboriste de 19 ans, doit récupérer les 4 sceaux des Esprits des saisons et les déposer au sanctuaire du village central pour réactiver la Roue des Saisons.

I.C) Résumé du scénario (complet)

Le monde de ce royaume médiéval-fantastique est déréglé : les saisons ont disparu, laissant une météo neutre et morne sur tout le territoire.
Au centre du monde se trouve un village neutre avec un sanctuaire dédié à la Roue des Saisons, aujourd'hui éteinte.

Le joueur incarne Tristan, un jeune apprenti herboriste de 19 ans, allergique au pollen, choisi pour sa connaissance des plantes et des esprits.
Quatre royaumes entourent le village : le Royaume de l'Hiver, du Printemps, de l'Été et de l'Automne, chacun avec son ambiance, ses dangers simples et son Esprit de saison.

Dans chaque royaume, Tristan doit réussir une petite épreuve (traverser le froid, gérer le pollen, supporter la chaleur, trouver son chemin dans les feuilles) pour obtenir le sceau de la saison correspondante.
Le jeu est gagné lorsque les quatre sceaux ont été ramenés au sanctuaire du village et placés sur l'autel, ce qui réactive la Roue des Saisons et rétablit l'équilibre du monde.

I.D) Plan (7 rooms)

On vise 7 rooms :

1. VillageCentral (village neutre + sanctuaire)
2. ForetHiver (accès au royaume de l'Hiver)
3. GrotteHiver (lieu de l'Esprit de l'Hiver)
4. ClairierePrintemps (Esprit du Printemps, fleurs et pollen)
5. OasisEte (royaume de l'Été, chaleur)
6. ForetAutomne (forêt aux feuilles mortes)
7. SanctuaireAutomne (lieu de l'Esprit de l'Automne)

Il n'y a pas de partie "réduite" spécifique, tout le jeu tient dans ces 7 rooms.

I.E) Scénario détaillé

Début du jeu

- Le joueur commence dans VillageCentral.
- On lui explique que la Roue des Saisons est éteinte et qu'il doit récupérer les 4 sceaux (Hiver, Printemps, Été, Automne).
- Il peut aller dans les directions menant aux royaumes (nord = Printemp, est = été, sud = Automne, ouest = Hiver).

Royaume de l'Hiver (ForetHiver)

- GrotteHiver :
    - Petite "épreuve" : un mini-labyrinthe très simple (2 entrées possibles, une seule mène à l'Esprit).
    - Une fois le bon chemin trouvé, l'Esprit de l'Hiver donne le SceauHiver.

Royaume du Printemps (ClairierePrintemps)

- ClairierePrintemps :
    - Ambiance : fleurs, pollen, ruisseau.
    - Tristan est allergique au pollen : sans Masque (ou foulard), il éternue et est renvoyé au VillageCentral quand il essaye d'avancer vers la zone des fleurs.
    - En explorant, il peut trouver un Masque dans le Village ou en parlant à un PNJ (par exemple un ancien herboriste).
    - Une fois protégé, il peut atteindre l'arbre sacré du Printemps et recevoir le SceauPrintemps après un simple "rituel" (posant une graine / un peu d'eau).

Royaume de l’Été (OasisEte)

- OasisEte :
    - Ambiance : chaleur très forte, zone désertique avec une oasis.
    - Sans GourdeEau, le joueur est “déshydraté” et renvoyé au VillageCentral (message de malaise).
    - En ayant récupéré une GourdeEau au village, il peut traverser pour atteindre l’oasis.
    - L’Esprit de l’Été, satisfait de sa prudence, lui remet le SceauEte.

Royaume de l’Automne (ForetAutomne + SanctuaireAutomne)

- ForetAutomne :
    - Forêt de feuilles mortes, vent, chemins peu lisibles.
    - Simple “puzzle de direction” : suivre les indices dans la description (ex : “le vent souffle vers le couchant”, donc aller à l’ouest, etc.).
    - Mauvaise direction → on retombe sur l’entrée de la forêt.
- SanctuaireAutomne :
    - Petit sanctuaire au milieu de la forêt.
    - L’Esprit de l’Automne donne le SceauAutomne après un court dialogue sur le changement et la fin des choses.

Fin du jeu

- Quand le joueur revient à VillageCentral avec les 4 sceaux (Hiver, Printemps, Été, Automne), il peut les placer sur l’autel du sanctuaire.
- Condition de victoire : le jeu vérifie que le joueur possède les 4 sceaux → message final décrivant le retour des saisons et fin de la partie gagnante.

I.F) Détail des lieux, items, personnages

Lieux (Rooms)

1. VillageCentral
    - Rôle : point de départ, sanctuaire, lieu de retour.
    - Accessoires de base, Manteau, GourdeEau et autel final.
2. ForetHiver
    - Rôle : zone de transition vers GrotteHiver.
    - Particularité : nécessite le Manteau pour rester longtemps.
3. GrotteHiver

    - Rôle : lieu de l’Esprit de l’Hiver.
    - Petit labyrinthe très simple (2–3 sorties maximum).
4. ClairierePrintemps
    - Rôle : accès à l’Esprit du Printemps.
    - Particularité : nécessite le Masque pour franchir la zone de fleurs.
5. OasisEte
    - Rôle : lieu de l’Esprit de l’Été.
    - Particularité : nécessite la GourdeEau pour survivre à la chaleur.
6. ForetAutomne
    - Rôle : zone de “labyrinthe léger” pour atteindre SanctuaireAutomne.
    - Indices de direction dans la description (vent, couleurs).
7. SanctuaireAutomne
    - Rôle : lieu de l’Esprit de l’Automne.

Items

- Manteau : protège du froid (Hiver).
- Masque (ou foulard) : protège du pollen (Printemps).
- GourdeEau : permet de traverser la chaleur (Été).
- SceauHiver, SceauPrintemps, SceauEte, SceauAutomne : objets-clefs de victoire.

Personnages

- Tristan : apprenti herboriste de 19 ans, personnage joueur.
- Ancien du village : PNJ, explique la situation et peut donner des indices / objets.
- Esprit de l’Hiver, Esprit du Printemps, Esprit de l’Été, Esprit de l’Automne : PNJ de fin de zone, donnent les sceaux.

I.G) Situations gagnantes et perdantes

- Situation gagnante :
    - Le joueur possède SceauHiver, SceauPrintemps, SceauEte, SceauAutomne et les place sur l’autel du sanctuaire dans VillageCentral → message de victoire, réactivation de la Roue des Saisons.
- Situations “perdantes” simples (sans game over définitif, juste retour en arrière) :
    - Aller dans ForetHiver sans Manteau → trop froid, retour au VillageCentral.

- Essayer de traverser la zone de fleurs du Printemps sans Masque → crise d'allergie, retour au VillageCentral.
- Essayer de traverser la zone d'Été sans GourdeEau → déshydratation, retour au VillageCentral.

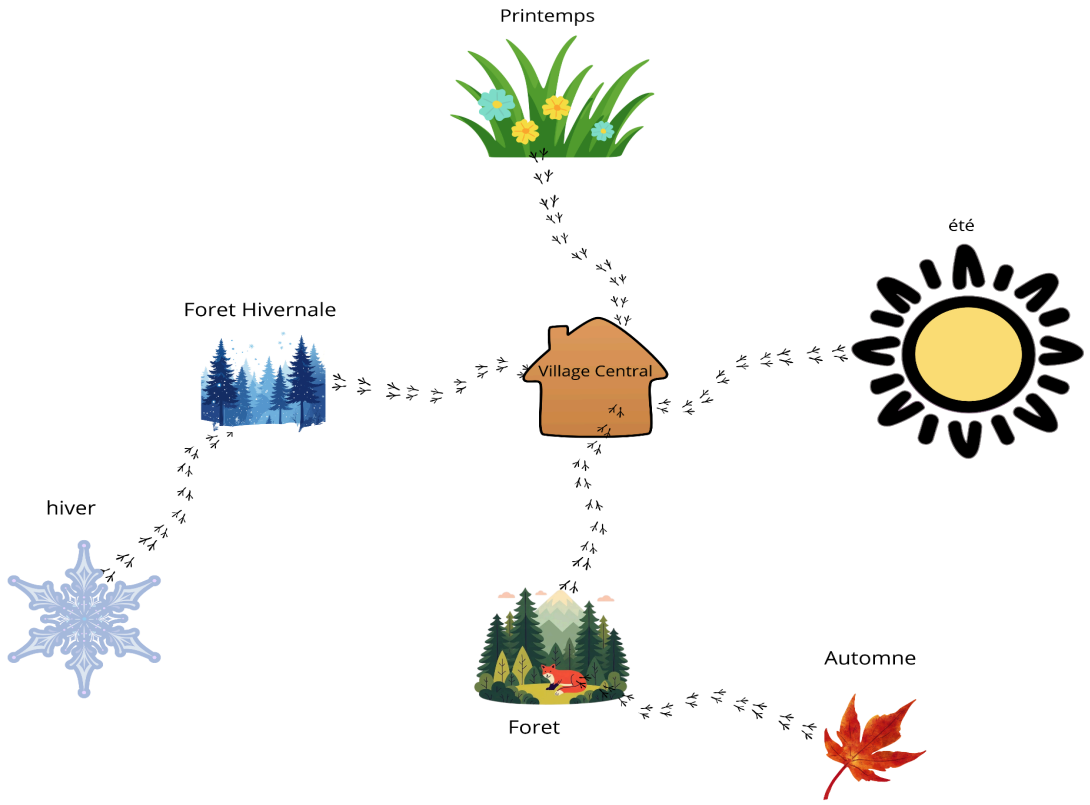
I.H) Énigmes, mini-jeux, combats, etc.

- Mini-labyrinthe très simple dans GrotteHiver (choix de chemin).
- Puzzle de protection dans ClairierePrintemps (avoir le Masque).
- Gestion de la chaleur dans OasisEte (avoir la GourdeEau).
- Puzzle d'orientation dans ForetAutomne (suivre le vent / couleurs).

I.I) Commentaires

- Partie technique : jeu basé sur 7 rooms, quelques objets et conditions simples → totalement réalisable dans le cadre du projet Zuul.
- Possibles extensions si temps : ajouter une 8e room optionnelle, plus d'items décoratifs, ou des descriptions plus riches.

Schéma de la Maps du jeu



Exercices

Exercice 7.5:

Le code pour afficher les informations du lieu courant (description + sorties) était dupliqué dans deux méthodes : printWelcome() et goRoom(). Cette duplication posait un problème de maintenabilité : toute modification future nécessiterait de changer le code à plusieurs endroits. Nous avons créé une nouvelle méthode printLocationInfo() dans la classe Game qui centralise l'affichage des informations du lieu courant (description de la pièce et liste des sorties disponibles). Le code dupliqué dans printWelcome() et goRoom() a été remplacé par un simple appel à printLocationInfo(). Toute modification future de l'affichage des informations de lieu ne nécessite maintenant qu'une seule modification dans printLocationInfo(), au lieu de devoir modifier plusieurs endroits du code.

Exercice 7.6:

Nous avons amélioré l'encapsulation de la classe Room en rendant les attributs de sortie privés. Nous avons créé une méthode getExit(String direction) qui retourne la pièce correspondant à une direction donnée, retournant null s'il n'y a pas de sortie. Dans la classe Game, nous avons simplifié la méthode goRoom() en remplaçant les multiples if/else par un simple appel à getExit(direction), ce qui réduit considérablement la duplication de code. Dans notre projet "La Roue des Saisons", cette méthode permet de naviguer entre les 7 salles (villageCentral, foretHiver, grotteHiver, clairierePrintemps, oasisEte, foretAutomne, sanctuaireAutomne) de manière uniforme. Ces modifications réduisent le couplage entre les classes Room et Game, permettant de modifier l'implémentation interne de Room sans impacter Game, conformément au principe d'encapsulation.

Exercice 7.7:

Nous avons créé la méthode getExitString() dans la classe Room qui construit et retourne une chaîne de caractères contenant toutes les sorties disponibles de la pièce. Cette méthode parcourt les attributs de sortie et ajoute à la chaîne le nom de chaque direction dont la sortie existe. Dans la classe Game, nous avons simplifié la méthode getLongDescription() (utilisée dans printWelcome() et goRoom()) en utilisant getExitString(). Dans notre projet "La Roue des Saisons", cela permet d'afficher dynamiquement les sorties disponibles pour chaque salle du royaume médiéval-fantastique. Cette modification respecte le principe de cohésion : la classe Room est responsable de préparer les informations qu'elle détient (ses sorties), tandis que la classe Game est responsable de l'affichage. Cela réduit le couplage car Game n'a plus besoin d'accéder directement aux détails internes de Room, rendant le code plus maintenable et extensible.

Exercice 7.8:

Nous avons remplacé les quatre attributs individuels de sortie de la classe Room par une seule HashMap<String, Room> qui stocke toutes les sorties possibles. Une HashMap associe une clé (le nom de la direction en String) à une valeur (la pièce voisine de type Room). Dans le constructeur de Room, nous avons initialisé cette HashMap. Nous avons créé une méthode setExit(String direction, Room neighbor) qui ajoute une sortie à la fois dans la HashMap avec put(). La méthode getExit() utilise get(direction) pour récupérer la pièce correspondante, et getExitString() parcourt toutes les directions avec une boucle for sur keySet(). Dans createRooms() de notre projet "La Roue des Saisons", nous appelons setExit() plusieurs fois pour créer les connexions entre le villageCentral et les quatre royaumes saisonniers (nord vers clairierePrintemps, est vers oasisEte, ouest vers foretHiver, sud vers foretAutomne), ainsi que les connexions entre foretHiver et grotteHiver, et entre foretAutomne et sanctuaireAutomne. Cette approche rend le code plus flexible car on peut facilement ajouter de nouvelles directions comme "up" ou "down" sans modifier la structure de la classe Room.

Exercice 7.8.1:

Dans cet exercice, nous avons ajouté une dimension verticale au jeu en implémentant des déplacements "up" et "down". Grâce à la HashMap mise en place dans l'exercice 7.8, l'ajout de ces nouvelles directions a été très simple et ne nécessite aucune modification de la structure de la classe Room. Nous avons d'abord ajouté les commandes "up" et "down" dans le tableau aValidCommands de la classe CommandWords, puis créé la méthode showAll() qui affiche toutes les commandes disponibles. Dans la classe Parser, nous avons ajouté la méthode showCommands() qui appelle showAll(). Dans createRooms() de notre projet "La Roue des Saisons", nous avons créé une connexion verticale entre la grotteHiver et le sanctuaireAutomne : grotteHiver.setExit("down", sanctuaireAutomne) et sanctuaireAutomne.setExit("up", grotteHiver). Le joueur peut maintenant descendre depuis la grotte de l'Hiver vers le sanctuaire de l'Automne en profondeur, ajoutant une dimension d'exploration verticale au royaume médiéval-fantastique. La méthode getLongDescription() de Room affiche automatiquement "up" et "down" dans la liste des sorties disponibles grâce à getExitString() qui parcourt toutes les clés de la HashMap. Cette flexibilité démontre l'avantage de la HashMap : ajouter de nouvelles directions ne nécessite que l'ajout de nouvelles entrées dans la HashMap, sans modifier le code existant de la classe Room.

Exercice 7.9:

La méthode keySet() de la classe HashMap renvoie un ensemble (Set) contenant toutes les clés présentes dans la Map.
Dans notre cas, exits.keySet() nous donne la liste de toutes les directions possibles (les "clés") enregistrées pour la pièce (ex: "north", "west"), sans se soucier des pièces vers lesquelles elles mènent.

Exercice 7.10:

La méthode getExitString présentée dans le Code 7.7 fonctionne de la manière suivante :
Elle commence par déclarer une variable de type String nommée returnString et l'initialise avec le texte "Sorties :".
Ensuite, elle appelle la méthode keySet() sur la HashMap exits. Cela permet de récupérer l'ensemble (Set) de toutes les clés (c'est-à-dire les directions comme "nord", "ouest") qui sont stockées dans la map. Ce résultat est stocké dans une variable keys.
Elle utilise une boucle for-each pour parcourir cet ensemble keys. À chaque itération, la variable exit prend la valeur d'une des clés (une direction).
Dans la boucle, elle concatène (ajoute à la suite) un espace vide " " puis la direction courante exit à la variable returnString.
Une fois toutes les clés parcourues, la méthode retourne la chaîne returnString complète, qui contient maintenant la liste de toutes les sorties disponibles (par exemple : "Sorties : nord ouest").

Exercice 7.11

Dans cet exercice, nous avons modifié la classe Room pour qu'elle soit responsable de sa propre description complète (incluant les sorties). Nous avons créé la méthode getLongDescription() qui retourne une chaîne formatée contenant la description du lieu et la liste des sorties disponibles. Ensuite, dans la classe Game, nous avons remplacé les appels manuels à getDescription() par getLongDescription() dans les méthodes printWelcome et goRoom. Cela réduit le couplage : la classe Game n'a plus besoin de savoir comment construire la description d'une pièce, elle se contente de l'afficher.

java
// Extrait de la modification dans Game.java
System.out.println(currentRoom.getLongDescription());

Exercice 7.13:

Lors de l'exécution d'une commande go, le diagramme d'objets change simplement par la modification de la référence currentRoom dans l'objet Game.
Avant la commande, currentRoom pointe vers la pièce actuelle (ex: villageCentral).
Après go north, currentRoom pointe désormais vers la nouvelle pièce (ex: clairierePrintemps).
C'est tout : seule la flèche (référence) currentRoom se déplace pour pointer vers un autre objet Room. Les objets eux-mêmes (Game et les Room) ne changent pas.

Exercice 7.14:

Pour ajouter la commande "look", nous avons d'abord modifié le tableau aValidCommands dans la classe CommandWords en y ajoutant la chaîne "look". Ensuite, dans la classe Game, nous avons implémenté une méthode privée look(Command command) qui appelle currentRoom.getLongDescription() pour réafficher la description du lieu. Nous avons également traité le cas optionnel où l'utilisateur tape un second mot (ex: "look tree") en affichant un message d'erreur approprié. Enfin, nous avons mis à jour la méthode processCommand pour détecter le mot-clé "look" et appeler notre nouvelle méthode.

```
// Extrait de la méthode look() dans Game.java
if (command.hasSecondWord()) {
    System.out.println("I don't know how to look at something in particular yet.");
} else {
    System.out.println(currentRoom.getLongDescription());
}
```

Exercice 7.15:

Nous avons méthodiquement fait la même chose qu’au point 7.14 mais pour “eat”

Exercice 7.16:

Dans cet exercice, l’objectif était d’éviter le couplage implicite entre les classes Game, Parser et CommandWords pour l’affichage de l’aide. La solution consiste à faire porter la responsabilité de la liste des commandes à CommandWords, puis à déléguer cet affichage

via Parser. Dans notre code, cette architecture est déjà en place : CommandWords possède une méthode showAll() qui parcourt le tableau aValidCommands et affiche toutes les commandes, Parser fournit une méthode showCommands() qui appelle aValidCW.showAll(), et la méthode printHelp() de Game affiche simplement un texte d'introduction puis appelle parser.showCommands(). Ainsi, lorsqu'une nouvelle commande est ajoutée dans aValidCommands, elle apparaît automatiquement dans l'aide, sans modifier Game.

```
// Extrait de Game.printHelp()
System.out.println("Vos mots de commande sont :");
parser.showCommands();
```

Exercice 7.17:

Si j'ajoute maintenant une nouvelle commande (par exemple sleep), je dois encore modifier la classe Game, mais uniquement pour lui donner un comportement (nouvelle méthode sleep(...) et un else if dans processCommand). En revanche, je n'ai plus besoin de modifier Game pour que la commande apparaisse dans le message d'aide : la liste des commandes est générée automatiquement par CommandWords.showAll() appelé via parser.showCommands() dans printHelp(). Donc, pour l'affichage de l'aide, la classe Game ne dépend plus de la liste des commandes, ce qui réduit le couplage implicite.

Exercice 7.18:

Dans cet exercice, nous avons amélioré la conception en séparant la production et l'affichage de la liste des commandes. Dans CommandWords, la méthode showAll() a été remplacée par getCommandList(), qui construit et renvoie une chaîne contenant toutes les commandes (aValidCommands) séparées par des espaces. La classe Parser a ensuite été modifiée pour que showCommands() affiche cette chaîne avec System.out.println(this.aValidCW.getCommandList());. La classe Game ne change pas : sa méthode printHelp() appelle toujours parser.showCommands(). Cette nouvelle organisation suit le même principe que getExitString pour les sorties des pièces : l'information est produite au plus près des données (CommandWords), transmise par Parser, puis affichée par Game, ce qui facilitera un changement futur d'interface (par exemple passer à une interface graphique).

Exercice 7.18.1:

En comparaison avec le projet zuul-better, aucune modification n'a été réalisée sur mon projet.

Exercice 7.18.2:

Dans cet exercice, j'ai étudié la classe StringBuilder afin d'optimiser la construction de chaînes de caractères répétitives. Les objets String sont immuables en Java : chaque concaténation crée une nouvelle chaîne, ce qui est peu efficace dans une boucle. À l'inverse, StringBuilder est mutable et permet d'ajouter du texte avec la méthode append sans recréer d'objet à chaque fois. J'ai donc réécrit la méthode getExitString() de la classe Room pour utiliser un StringBuilder : je crée d'abord un StringBuilder vide, je parcours l'ensemble des directions (les clés de la HashMap aExits) et j'ajoute chaque direction suivie d'un espace avec append, puis je retourne le résultat avec toString(). Cette modification ne

change pas le comportement du jeu mais rend la construction de la chaîne de sorties plus propre et plus efficace.

Exercice 7.18.3:

Le choix des images est déjà défini.

Exercice 7.18.4:

Nom: *La Roue des Saisons.*
Message de bienvenue: *La Roue des Saisons est un jeu d'aventure, incroyablement ennuyeux.*