

AP 2018-2019 s1

Auteur

Clément Chomicki

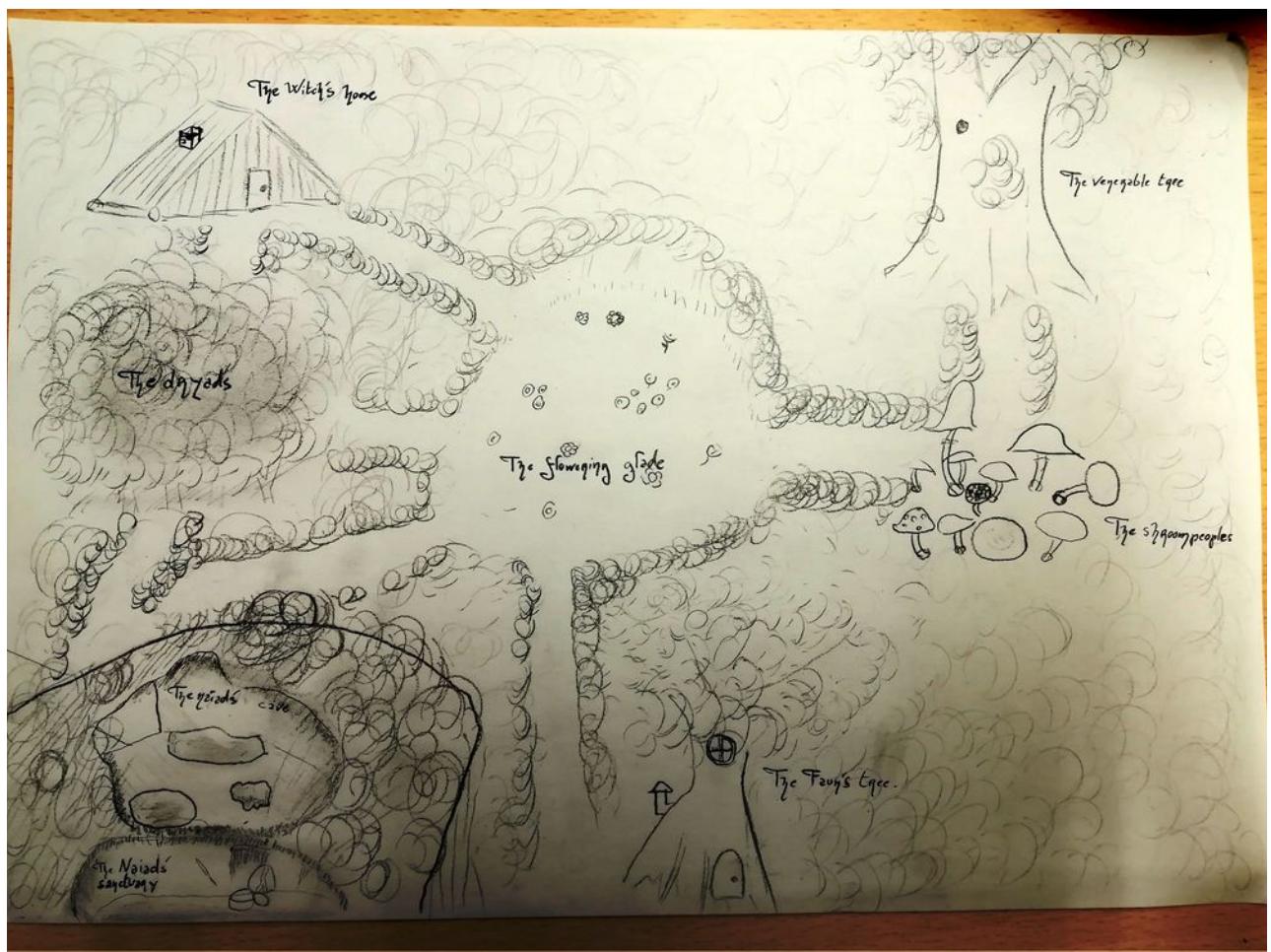
Thème

Dans une forêt onirique,
un humain perdu cherche son chemin
et fait d'étranges rencontres.

Résumé du scénario

Le joueur se retrouve prisonnier d'un bois magique, appellé par un esprit sylvestre pour l'aider à contrer la sorcière régnant sur la forêt.

Plan complet



Scénario

Le personnage principal se réveille dans une forêt enchantée, réveillé par un esprit sylvestre. Ce dernier lui demande de l'aider à mettre un terme à la sorcière régnant sur ces bois.

Le joueur va errer dans cette forêt enchanté à la recherche d'un moyen de battre la sorcière, et possiblement aider ou juste sympathiser avec ses habitants.

Le joueur pourra trouver une arme auprès de l'écureuil habitant le grand arbre, après l'avoir aidé à récupérer sa provision de noix, volée par la tribu mycéienne.

Le joueur pourra aider le Faune à récupérer sa théière, empruntée par les naiades depuis des jours.

Les naiades auront perdu cette théière dans le territoire des dryades lors d'une précédente sortie pic-nique.

Le joueur se fera agresser par une dryade en cherchant dans leur territoire, avant de se faire secourir

par la naïade l'accompagnant pour les recherches.

Il sera rammené au sanctuaire des naïades, et baigné dans une eau enchantée, guérira de ses blessures.

Il pourra par la suite rammener la théière au faune.

Quand il le désire, le joueur peut décider d'approcher la cabane de la sorcière, voir même la défier.
L'issue de la confrontation dépendra des actions précédemment effectuées.

Le joueur peut aussi décider d'affronter l'esprit sylvestre.
Il le trouvera en se perdant dans la forêt après avoir aidé l'écureuil.
L'issue de la confrontation dépendra des actions précédemment effectuées.

Détail des lieux, items et personnages

Lieux

Au coeur de la forêt se trouve une clairière fleurie.
Il semble y faire toujours beau et chaud.

Au sud se trouve l'arbre-maison du faune.
C'est comme si cet arbre avait poussé dans le seul et unique but de former une maison douillette.

Au sud-ouest se trouve la grotte des naïades.
Elles y vivent car en dessous, dans les parties les plus reculées de la grotte, se trouve leur sanctuaire, et surtout à cause des sources chaudes présentes.

À l'ouest se trouve la partie la plus sauvage et la plus ancienne de la forêt.
C'est le territoire des dryades.

Au nord-ouest se trouve la cabane de la sorcière.

À l'est se trouve le royaume champignon. Y vivent un peuple de petit être mi-hommes mi-bolets.

Au nord-est se trouve un arbre si grand qu'on le voit à des kilomètres.

Partout ailleurs est la forêt. Quand le joueur ne suit pas les chemins et décide de s'enfoncer dans les bois, il se perd et met plusieurs déplacement à retrouver son chemin.
Il se peut qu'un cour d'eau soit présent parfois.

Items

Une dague runique, arme donnée par l'écureuil
La théière du faune
Une fleur prise dans la clairière
Des noix

Personnages

Edmond le faune

Sympatique et patient, il passe la majeure partie de son temps à boire du thé et philosopher.

Les naïades

*Agape, Philia et Eris sont trois nymphes de l'eau habitant la grotte.
Elles aiment jouer des tours au faune et ne rien faire.*

Les dryades

*Leurs noms n'est pas connu par le joueur.
Elles sont sauvages, violentes et territoriales.*

Arglaë la sorcière

On ne sait pas grand chose d'elle, mais son pouvoir est dit immense.

Les Shrooms

*Ces êtres mi-hommes, mi-champignons suivent aveuglément leur délégué général.
Ils aiment les bien de production et les réunions syndicales.*

L'écureuil

*Il n'a pas de nom. Trop bête pour ça.
Il vit sur le grand arbre.*

L'esprit sylvestre

*On ne sait pas grand chose de lui.
Mais il est déterminé à mettre un terme à la sorcière qui l'a banni.*

Situations gagnantes

Il y a quatre façons de gagner:

- * Affronter la sorcière avec l'arme récupérée auprès de l'écureuil et après avoir été baigné dans le sanctuaire.
- * Abandonner l'idée de la confrontation et rester vivre avec les habitants de la forêt.
- * Affronter l'esprit sylvestre sous les mêmes conditions que pour la sorcière.
- * Être en haute estime des naïades et suivre la rivière que l'on trouve en se perdant dans la forêt.

Situations perdantes

Il y a trois façons de perdre:

- * Affronter la sorcière sans avoir remplis les conditions.
- * Affronter l'esprit sylvestre sans avoir été baigné dans le sanctuaire.
- * Interrompre à main armée la réunion syndicale des Shrooms sans carte du parti.

Questions

#7.1

Lecture du début du chapitre 7 du livre.

#7.1.1

Choix du thème.

#7.2

TP 3.1 et 3.2 terminés

#7.2.1

Emplois de la classe Scanner pour récupérer le texte entré en console.

#7.3

Conception du scénario.

#7.3.1

Création de ce rapport.

#7.3.2

Dessin du plan des lieux (sous forme de dessin au crayon scanné)

#7.4

Mis mes pièces à la place de celles de zuul-bad.

#7.5

Ajouté la méthode printLocationInfos dans la classe Game
pour remplacer les duplcats dans goRoom et printWelcome

#7.6

Ajout de la méthode getExit dans la classe Room.

Remplacement de la série de test de Game.goRoom() par un appel de getExit.

#7.7

Il est logique que Room soit responsable d'exploiter ses propres attributs,
et ne pas lui donner la gestion de l'affichage est normal car Room ne décrit que des pièces,
et l'affichage d'un jeu vidéo n'a aucun rapport avec des pièces.

C'est Game qui a pour rôle de gérer entre autre l'affichage.

Lui demander de produire des infos relatives à des attributs d'autres classes serait lui donner trop d'importance,
et en ferait un monstre de code tentaculaire dur à gérer dans l'optique du respect de l'encapsulation.
Chacun son métier et les vaches seront bien gardées.

#7.8

On implémente les pièces voisines dans un hashmap attribut de Room aExits en utilisant les
directions comme clées, afin de rendre le code plus extensible (et élégant).

On créa alors une méthode setExit qui prend en élément une chaîne de caractère (la direction)
et un objet Room, et qui ajoute ce dernier au hashmap de l'objet room courant en utilisant la direction comme clé.

#7.8.1

On ajoute sans difficultées des liens de voisinages verticaux entre certaines pièces
en utilisant "down" et "up" comme clés (ce qui fait maintenant 6 directions, avec "east", "west", "north" et "south").

#7.9

On améliore la méthode de Room getExitString en utilisant la méthode keySet de Hashmap
qui retourne un objet itérable par une boucle for et contenant les clés du hashmap sur lequel on l'appelle.

Ceci permet de réduire le couplage en laissant la liberté à `getExitString` de ne plus se préoccuper du nombre et de la formes des sorties.

#7.10

```
public String getExitString()
{
    String vS = "Exits:";
    for (String vKey : this.aExits.keySet()) {
        vS += " " + vKey;
    }
    return vS;
} // getExitString()
```

`getExitString` créé une chaîne de caractère `vS` initialisée à "Exits: ", puis pour chaque élément dans `aExits` (ie pour chaque sortie) `vS` est concaténée avec la clé correspondante et un espace, à l'aide d'une boucle `for` itérant dans l'ensemble des clés de `aExits`.

On retourne `vS` à la fin, qui est alors de la forme "Exits: direction1 direction2 ... directionx"

#7.10.1&2

Completion et génération de la javadoc

#7.11

On donne à `Room` via `getLongDescription` le rôle de construire la String à afficher pour décrire la scène, étant donné que c'est à `Room` qu'appartient les éléments à partir desquels on la construit.

#7.14

On ajoute "look" aux tableau des commandes valides (dans `CommandWords.java`)

On ajoute ensuite une méthode 'look' dans `Game`, qui pour l'instant ne fait que appeler `printLocationInfo`.

On ajoute le cas "look" dans `processCommand` pour appeler la méthode `look` quand on entre la commande 'look'.

#7.15

On ajoute une commande "eat" de la même manière que pour la commande "look".

"eat" ne fait pour l'instant qu'afficher un texte indiquant qu'on a mangé.

#7.16

On donne à la classe `CommandWords` le rôle d'afficher les commandes autorisées via une nouvelle procédure `showAll`. Parser étant la seule classe à utiliser `CommandWords`, on lui donne le rôle d'appeler `showAll` via une nouvelle méthode `showCommands`.

On modifie `printHelp` pour que la liste des commandes valides soit affichée par un appel de `showCommands` sur l'attribut `aParser`.

#7.17

Quand on ajoute une nouvelle commande, on doit toujours implémenter la méthode appellée par la commande dans `Game` et on doit modifier le test dans `processCommand` (donc dans `Game`) pour inclure la nouvelle venue.

On doit donc à ce stade toujours modifier `Game` à l'ajout d'une commande.

#7.18

On remplace la procédure d'affichage de 7.16 'showAll' par une fonction de type `String` construisant et retournant la chaîne de caractère à afficher plutôt que de laisser la tâche d'affichage à `CommandWords`, qui n'a absolument rien à voir avec le concept d'affichage.

On remplace donc en conséquence 'showCommands' dans `Parser` par une fonction de type `String 'getCommandList'` qui demande à 'makeCommandList' la construction de la chaîne à afficher et la renvoie, pour toujours ne laisser que `Parser` avoir accès à `CommandWords`. Et on finit par remplacer l'appel de 'showCommandList' dans la procédure 'printHelp' de `Game` par " `System.out.println(this.aParser.getCommandList())` " .

Le programme fonctionne de la même façon qu'avant, mais les tâches sont désormais un peu mieux distribuées.

#7.18.1

Rien ne venant de zuul-better n'est vraiment mieux que la version actuelle du projet.

#7.18.2

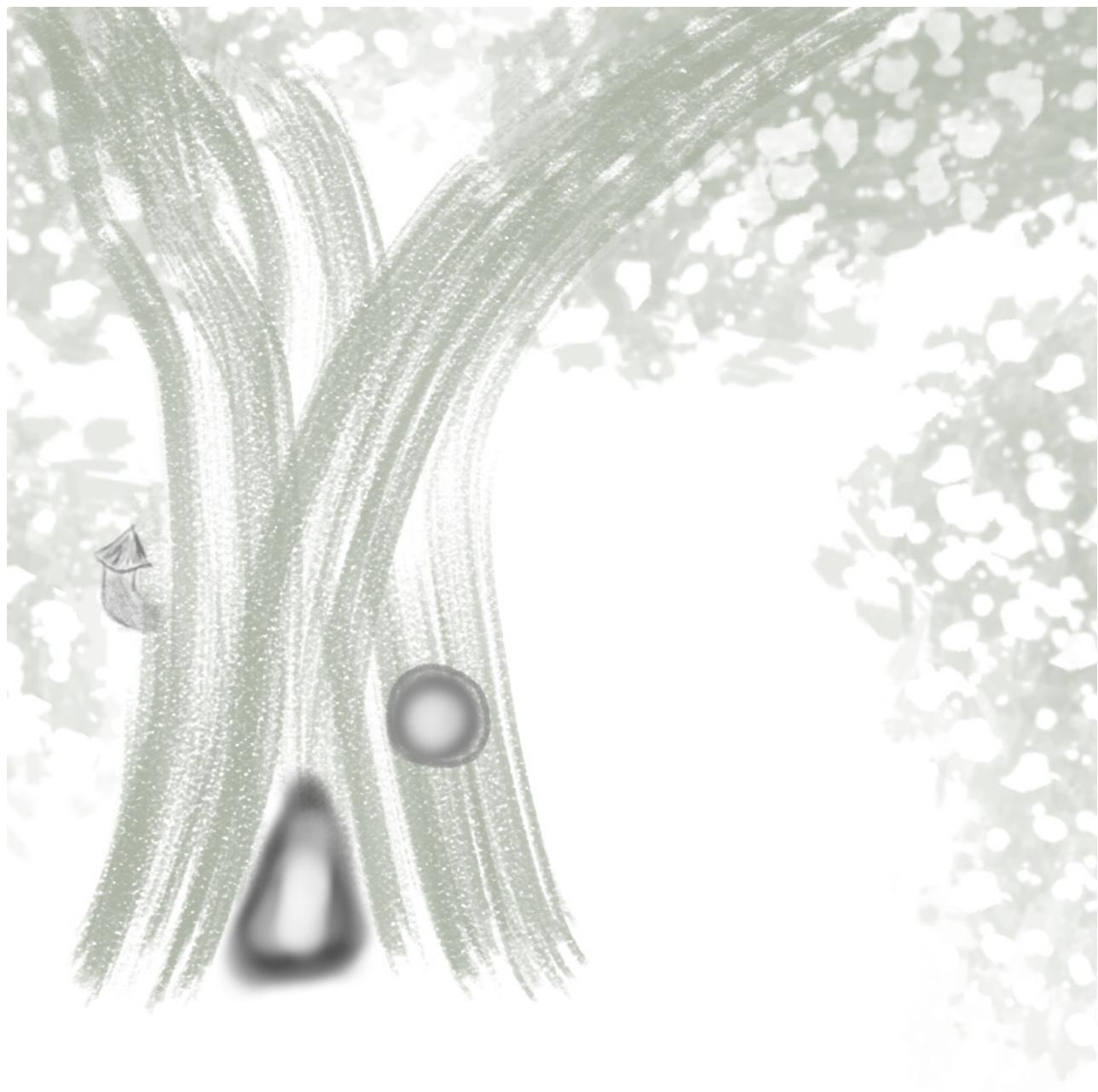
La classe `StringBuilder` est une implémentation de `String` mutable, permettant l'ajout de caractères sans que Java aie à recréer une `String` à chaque modification.

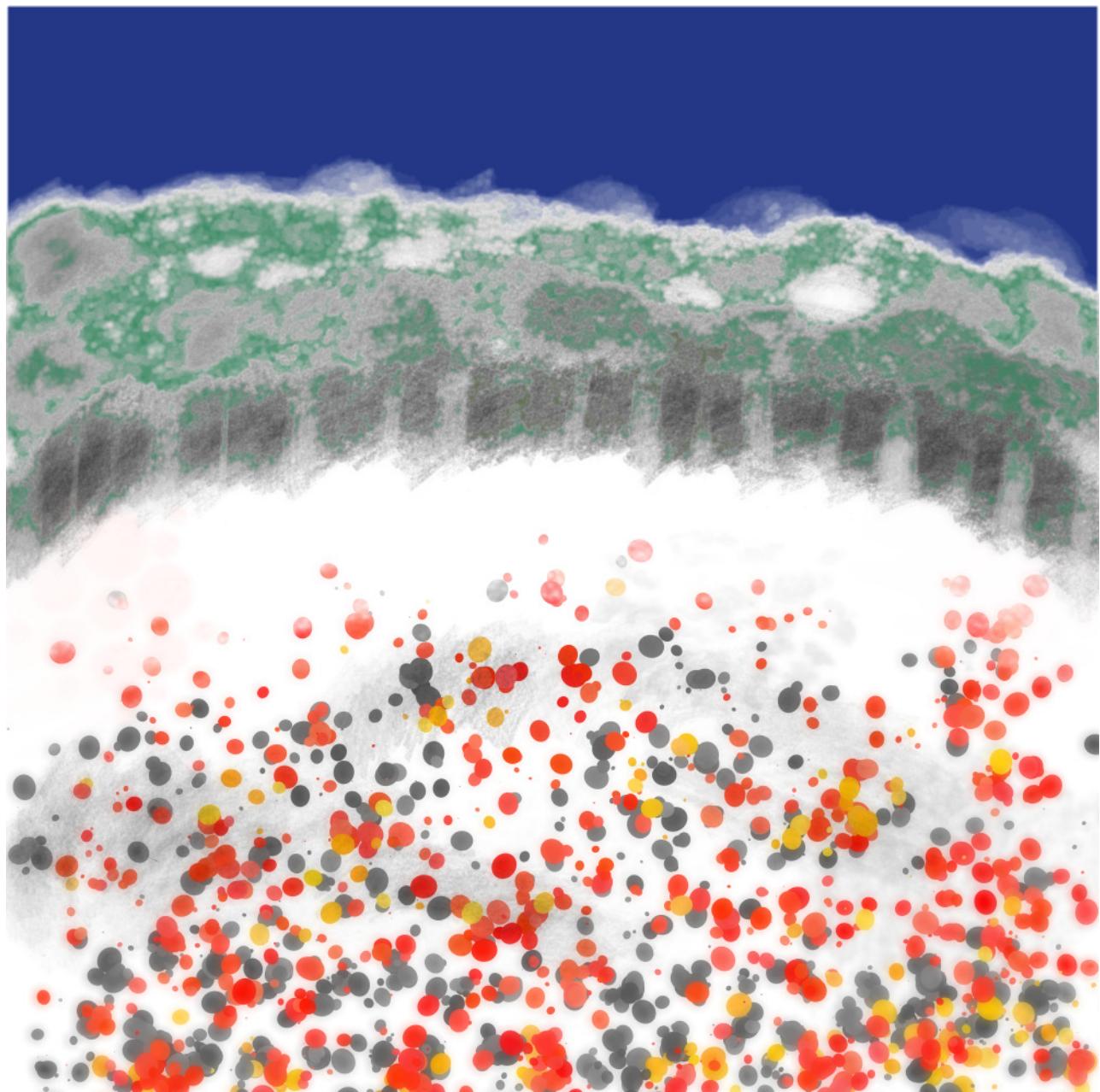
L'utiliser permettrait d'optimiser conséquemment certaines méthodes, comme 'Room.getExitString', mais l'intérêt serait moindre étant donné du faible nombre de concaténations faites par cette méthode, la faible fréquence d'appel à cette méthode et les performances peu exigeantes requises pour le moment. Ce genre d'optimisations seront peut-être implémentées une fois le jeu en voie de finition. "Premature optimization is the root of all evil" -- Donald Knuth

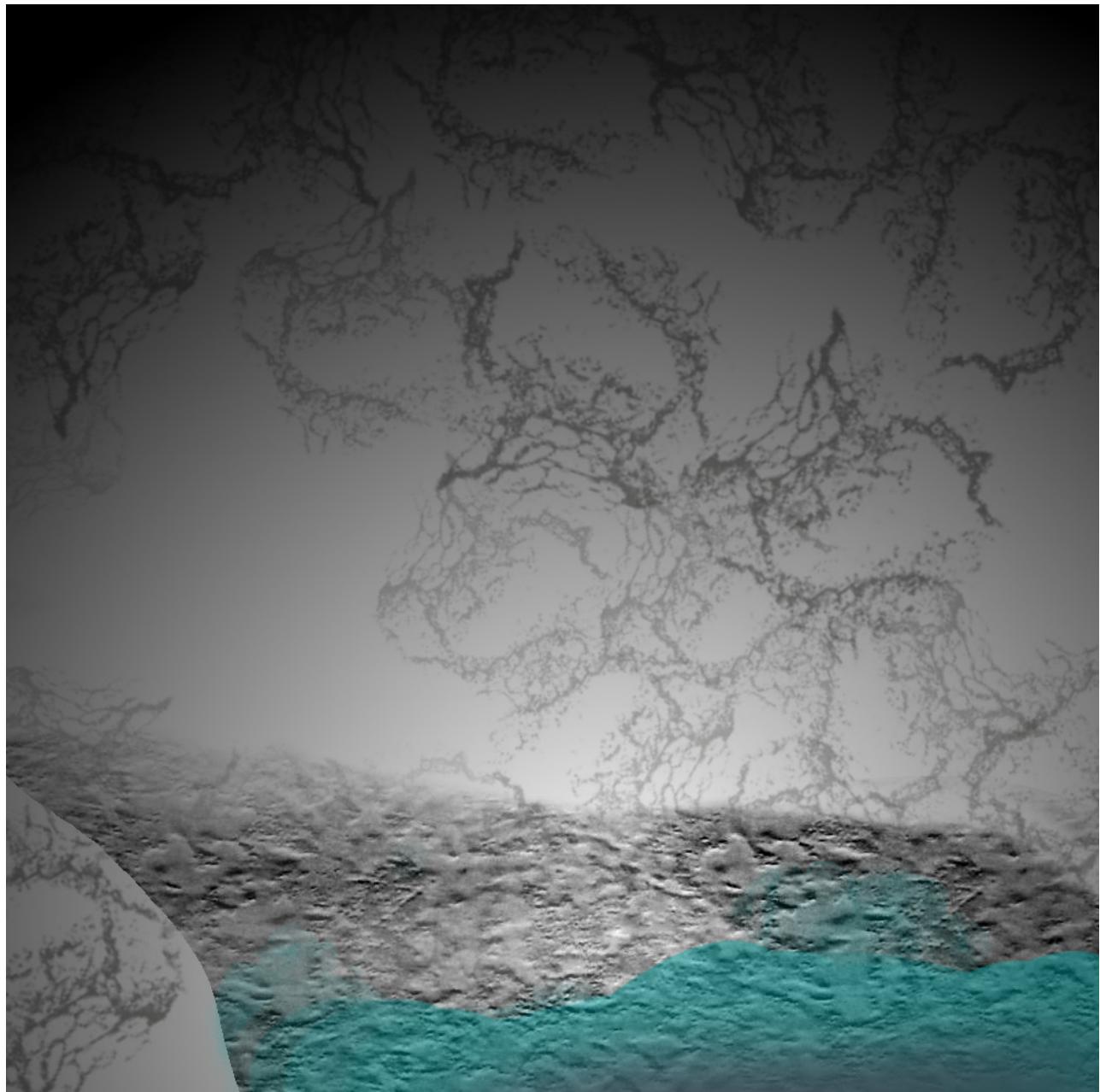
#7.18.3

Illustration (au moins temporaires) peintes.



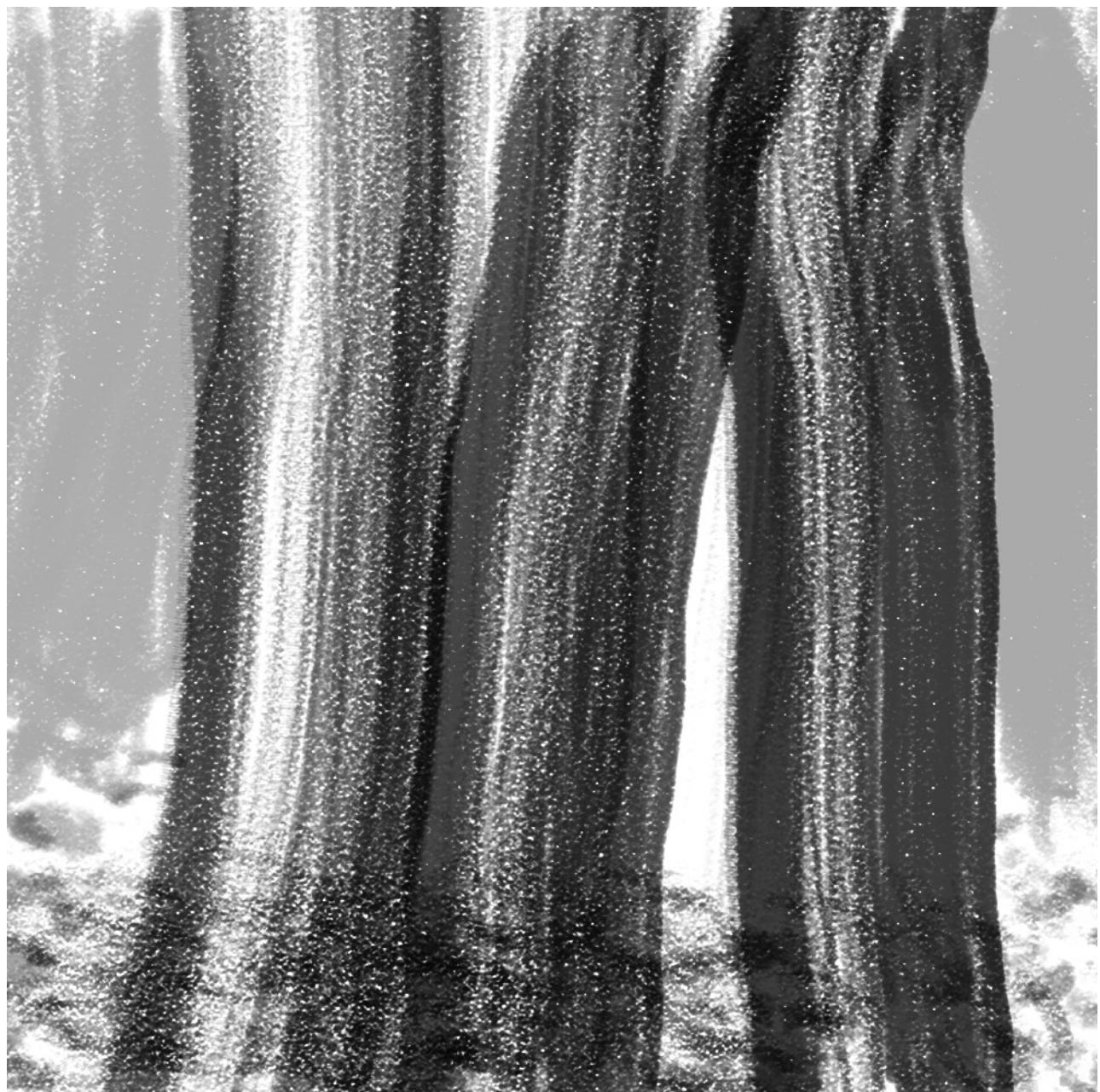


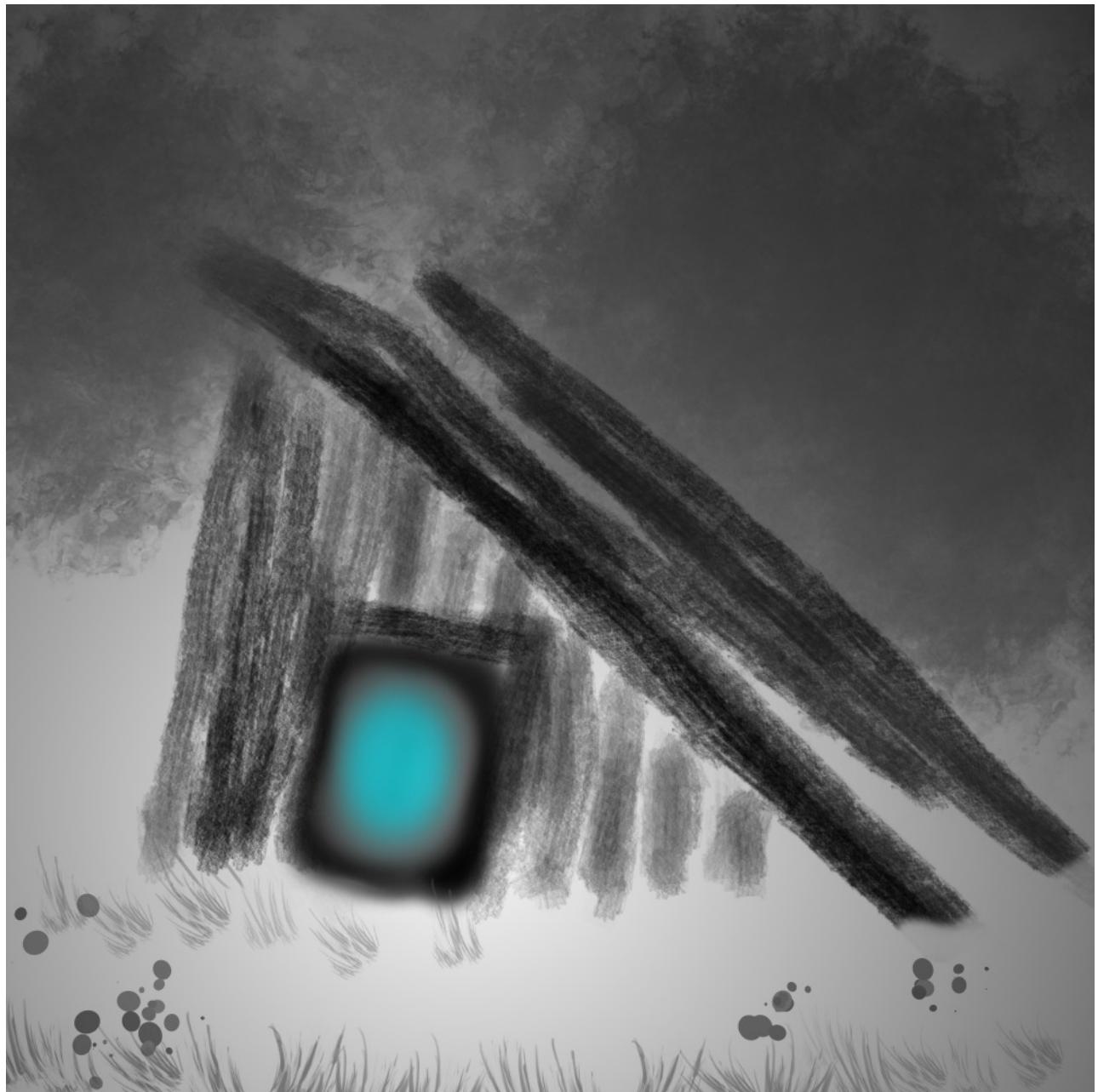












#7.18.4

Le titre du jeu sera "Within the woods"

#7.18.5

On ajoute un attribut aRooms à la classe Game, de type `HashMap<String, Room>`.

Ce hashmap permet de stocker les pièces créées dans `createRooms` dans un objet

qui sera accessible dans tout Game, et donc au moins d'y avoir accès autre part que dans `createRooms`, ce qui n'était pas possible avant.

On n'oublie pas de l'initialiser avec:

```
this.aRooms = new HashMap<String, Room>();
```

Dans le constructeur de Game.

Déclaration anti-plagia

L'entièreté du code présent dans ma version du projet est de mon cru, sauf celui issu des indications données par les exercices, qui est alors une simple application des consignes et dont je ne suis que l'exécutant.