Rapport TOB : Generation Procedurale

Nicolai Beuhorry-Sassus

May 2024

I Introduction

Une des valeurs ajoutées majeure de notre projet est le fait que chaque étage de notre Donjon est généré de manière unique. Pour répondre à ce critère nous avons décidé mettre en place un algorithme de génération procédurale. Ainsi dans cette partie, nous traiterons de la conception à l'implantation de l'algorithme de génération procédurale en passant également par l'implantation des différentes classes concourant à son bon fonctionnement.

En lien avec les fonctionnalités annoncées du projet (c.f. : voir le livrable *fonctionnalites.pdf* pour plus de détails), ainsi que les features et user stories relatives à l'application des méthodes agiles, cette partie est en lien avec les points suivants :

fonctionnalités en lien avec cette partie :

- Salles classiques
- Salles Spéciales
- Carte

Features traitées dans cette partie :

- Etage
- Portes
- Salles Spéciales

II Conception

Nous aborderons dans cette partie la manière dont nous avons travaillé pour concevoir l'algorithme avant de vous présenter le raffinage vers lequel nous avons abouti.

Pour commencer suite à une première réunion de repartition des tâches pour le projet, nous avons commencé à travailler M. Pointeau et moi sur un premier

raffinage et une première implantation qui ne se sont pas montrés concluants. Le modèle globale de l'étage sur lequel nous étions partis était beaucoup trop complexe ce qui rendait la tâche de relier les portes des salles entre elles, et ce de manière cohérente, presque impossible.

Par la suite, lors d'une seconde réunion M. Moussu s'est joint à notre groupe et c'est suite à une seconde séance de brainstorming que nous avons mis au point un modèle d'étage plus simple ainsi qu'une première version du raffinage finale.

Le modèle de l'étage était le suivant : visualiser l'étage comme un grille géante ou chaque case représente un emplacement envisageable pour y mettre une salle.

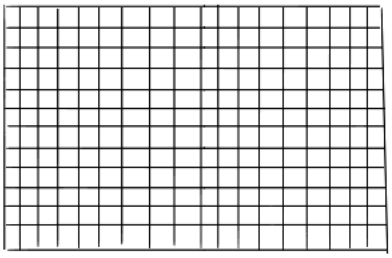


Figure 1: Illustration du modèle de l'étage sous forme de grille avec des cases

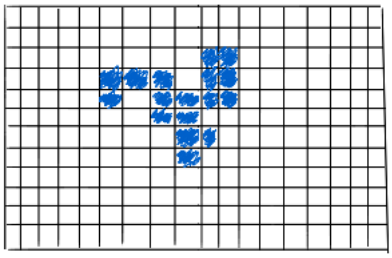


Figure 2: Illustration du modèle de l'étage avec des salles placées (carré bleu)

A partir de cet instant, nous avons implanté la première version du raffinage de l'algorithme de génération procédurale puis après avoir aporté de légères modifications durant l'implantation, nous avons concu l'algorithme suivant :

- **R0 :** Génerer de manière procédurale un étage
- **R1 :** Comment **R0** ?

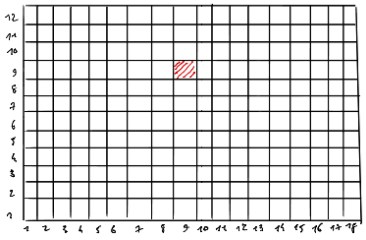
  Choisir une case de départ
  Construire la salle de départ
  Choisir la case suivante
  Initialiser le premier couple (case source, case libre)

TANT QUE le nombre de salles est inférieure au nombre de salles maximum dans l'étage **FAIRE**

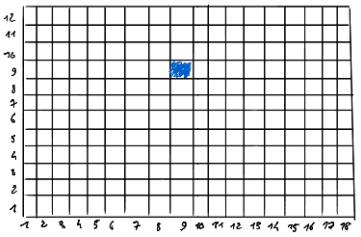
Tirer un couple (case source, case libre)
Choisir une nouvelle salle
Relier la nouvelle salle avec la salle source
Creer de nouveau couple (case source, case libre) à partir des cases adjacentes de la nouvelle salle

FIN TANT QUE
Placer la salle étage suivant

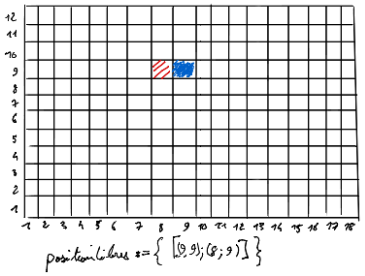
Illustrons cet algorithme dans le cas où le nombre maximum de salles dans l'étage est 4 (Figure 3 et Figure 4):



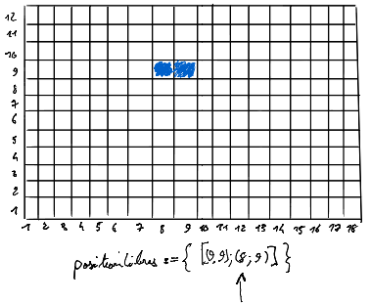
(a) Illustration de la première action complexe du raffinage



(b) Illustration de la seconde action complexe du raffinage



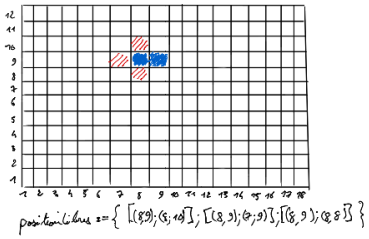
(c) Illustration de la troisième et quatrième action complexe du raffinage



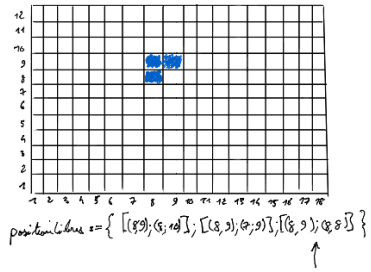
(d) Illustration des trois premières actions complexes du raffinage pour la première itération de la boucle TANT QUE

Figure 3: Illustration de l'algorithme de génération procédurale pour quatre salles dans l'étage (première partie)

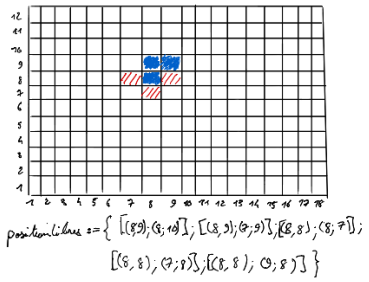
En ce qui concerne l'implatation à proprement parler de la génération procédurale je vous invite à aller consulter le code et ses commentaires situé dans le répertoire *src/core/src* dans le package *com.libgdx.roguelike*, dans la classe *Etage* à la



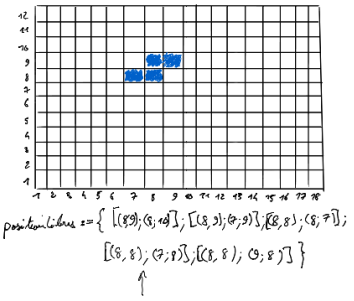
(a) Illustration de la quatrième action complexe du raffinage pour la première itération de la boucle TANT QUE



(b) Illustration des trois premières actions complexes du raffinage pour la deuxième itération de la boucle TANT QUE



(c) Illustration de la quatrième action complexe du raffinage pour la deuxième itération de la boucle TANT QUE



(d) Illustration de la dernière action complexe du raffinage

Figure 4: Illustration de l'algorithme de génération procédurale pour quatre salles dans l'étage (deuxième partie)

méthode *generer()* ligne 162, si vous souhaitez en savoir plus. Vous y retrouverez notamment en commentaire les étapes majeures du raffinages avec en dessous l'implantation correspondante en *java*.

III Structure du projet concernant la partie génération procédurale

L'implantation de la génération procédurale a bien évidemment nécéssité l'implantation de différentes classes et de différents packages afin de pouvoir manipuler 'numériquement' les entités étage, salle, porte, case et les couples (case source, case libre) qui est implanté par la classe *Position*. Le diagramme de classe ci-dessous (Figure 5) permet de synthétiser les différentes classes impliquées entre autre dans la génération procédurale. Pour plus de détails je vous invite à aller consulter le code dans le repertoire *src/core/src* dans le package *com.libgdx.roguelike*.

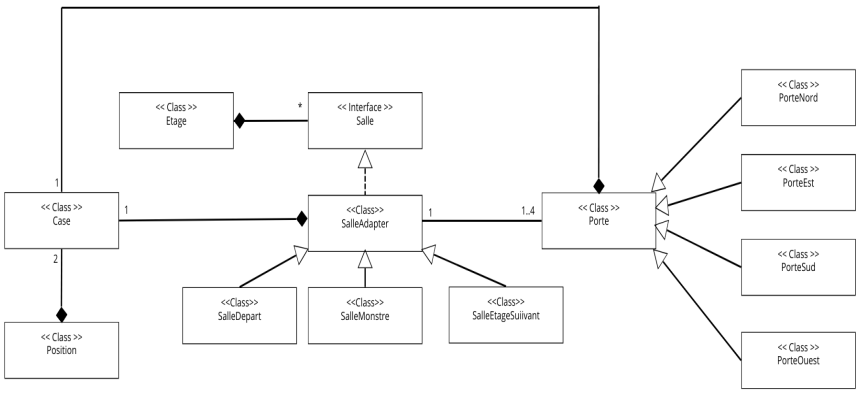


Figure 5: Diagramme de classe des objets impliqués dans la génération procedurale

IV Validation de la génération procédurale

Afin de valider le bon fonctionnement de la génération procédurale, nous avons dans un premier temps visualisé le résultat obtenu en implantant une carte qui était initialement destinée uniquement à la fonction de test (résultat Figure 6).

Cette première visualisation du placement des salles dans l'étage, nous a permis de valider la cohérence du placement des salles sur l'échiquier de cases de l'étage.

Nous avons voulu nous assurer par la suite que les différentes salles étaient correctement reliées entre elles. C'est à dire qu'on pouvait accéder à toutes les salles en partant de n'importe quelle salle et que lorsque deux salles étaient reliées par une porte, que cette porte était présente de manière symétrique à la fois dans l'une est dans l'autre des deux salles.

Pour ce faire, nous avons modifié certaines classes ainsi que 'la carte test' afin d'afficher également les portes de chaque salle. Il n'était cependant pas possible de vérifier directement le critère de symétrie pour les portes car l'affichages de ces dernières se superposait lorsqu'elles reliaient un même couple de salles. Le résultat obtenu correspond à celui de la Figure 7.

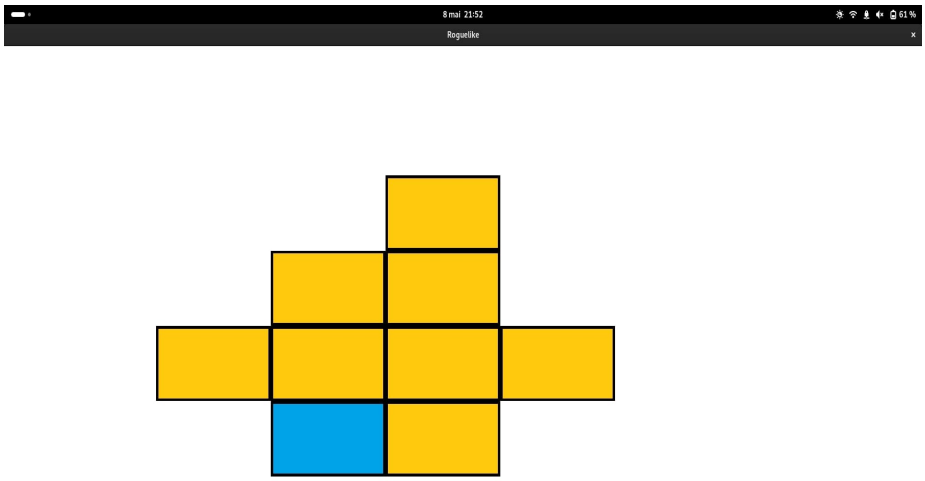


Figure 6: Image du résultat affiché par la carte test pour un étage de neuf salles. *Légende : Rectangle bleu : salle départ; Rectangle jaune : salle monstre*

Ce dernier affichage nous a permis de valider le fait que toutes les portes étaient correctement reliées.

Cependant, afin de valider les critères d'acceptations des classes impliquées dans la génération procedurale et de nous assurer de certaines propriétés sur le résultat de l'algorithme, des programmes de tests à l'aide de *JUnit* ont été réalisés. Pour plus de détails je vous invite à consulter la classe *GenerationProceduraleTest* qui se trouve dans le répertoire *src/core/ src/test/java* du projet.

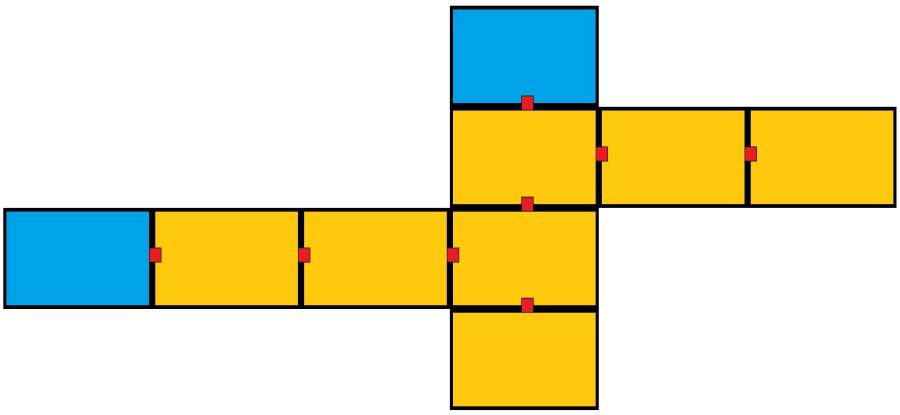


Figure 7: Image du résultat affiché par la carte test avec les portes pour un étage de neuf salles.
Légende : Rectangle bleu : salle départ/étage suivant; Rectangle jaune : salle monstre; Carré rouge : porte