# Projet de programmation: MapMaker

L'objectif de ce projet est d'implémenter une interface graphique de création de carte du monde et un algorithme de complétion automatique de carte.

---

## *MapMaker*

*MapMaker* est un programme qui doit permettre la création interactive d'une carte du monde, à partir de morceaux de paysages prédéfinis. La carte sera créée sur une grille, que l'on considérera initialement de taille 10 cases par 10 cases. Chacune de ces cases contiendra une image représentant une partie de la carte : ces images sont appelées *tuiles*. On donne ci-dessous un exemple de carte pouvant être générée par le programme.

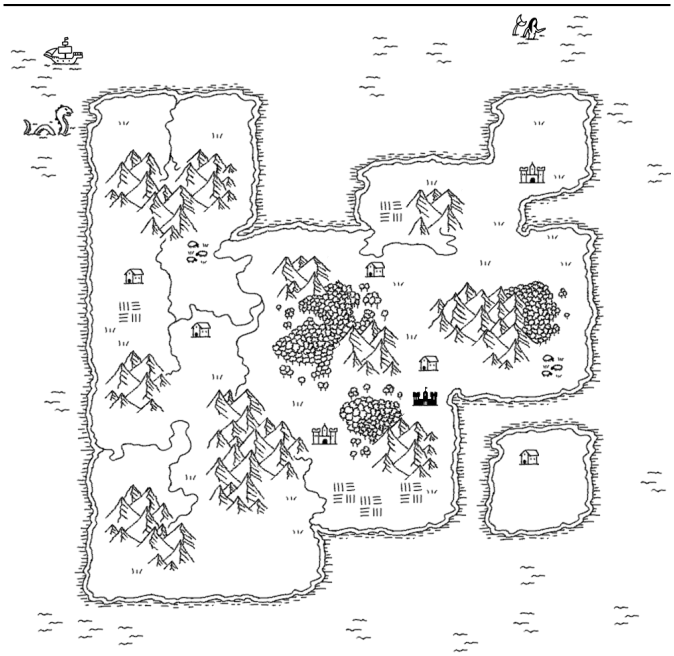


*Fig. 1 : Un exemple de carte générée par MapMaker*

La première fonctionnalité attendue de *MapMaker* est la création manuelle de carte : lorsque l'utilisateur clique sur une case vide, le programme devra lui proposer une sélection de tuiles qui pourront être positionnées dans cette case. L'utilisateur pourra ainsi dessiner la carte case par case. On propose un exemple d'interface permettant de sélectionner une nouvelle tuile :

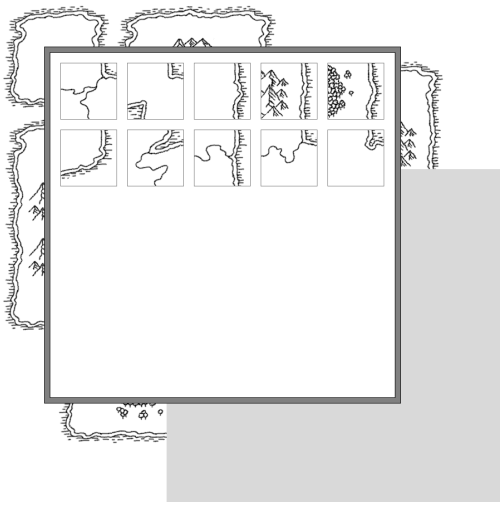


*Fig. 2 : Une interface possible pour la sélection de tuiles*

Pour assurer une carte harmonieuse et cohérente, on vérifiera à chaque placement d'une nouvelle tuile qu'elle est compatible avec les tuiles déjà présentes sur la carte. Par exemple, on évitera d'accoler une tuile avec un bord de forêt à un bord de montagne.

On veillera également à ce qu'il soit possible de **retirer une tuile déjà placée**, pour pouvoir modifier la carte en cours de création.

La deuxième fonctionnalité de MapMaker est la **complétion automatique de carte**. À tout moment lors de la création de carte, l'utilisateur peut demander au programme de compléter automatiquement la carte en cours : toutes les cases vides sont alors remplies par des tuiles adéquates, de manière à obtenir une carte complète.

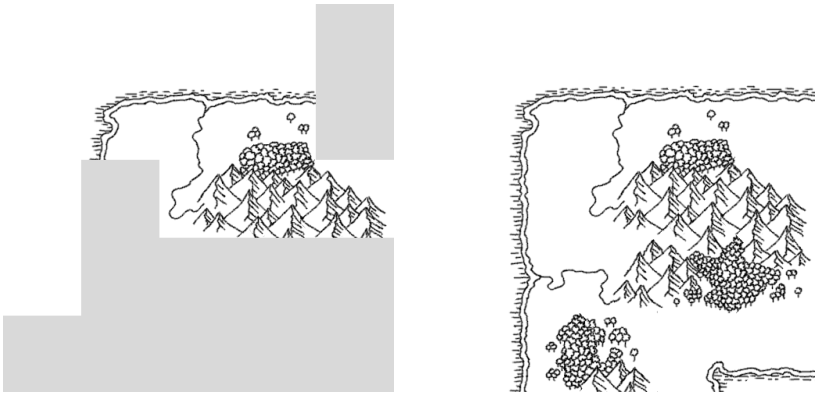


*Fig. 3 : Une carte incomplète…* *…et sa complétion.*

Il peut aussi arriver qu'il soit impossible de compléter la carte en respectant les tuiles déjà placées,

par exemple comme sur la figure 4 : il faudrait une tuile contenant une mer en haut et en bas, une forêt à droite, et une plaine à gauche, et il n'existe pas de telle tuile.

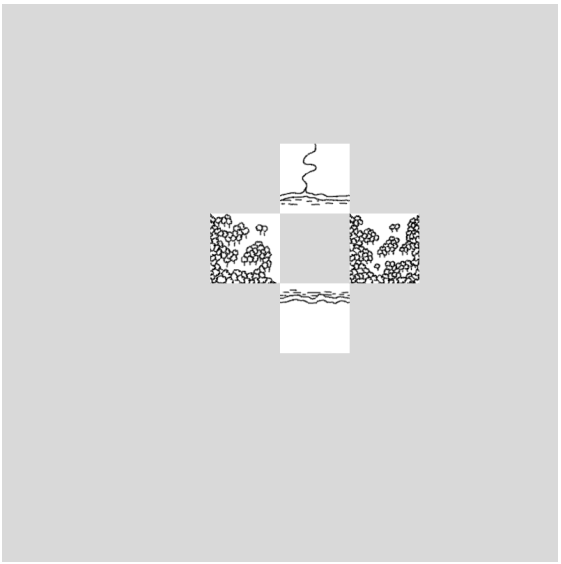


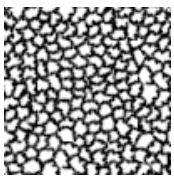
*Fig. 4 : Une carte qu'il n'est pas possible de compléter*

On s'assurera de **détecter correctement ces situations**, et de notifier l'utilisateur qu'aucune solution n'est possible.

## Les tuiles

Les tuiles contiennent des éléments de paysage de plusieurs types : des rivières, des forêts, des plaines, des montagnes, des côtes et de la mer. On nomme ces différents éléments de paysage des *biomes*. L'assemblage de ces différentes tuiles doit être fait de sorte à ce que le raccord entre deux tuiles soit invisible à l'oeil nu, afin d'obtenir une carte d'un seul tenant. Chaque tuile possède 4 côtés, correspondant chacun à un biome précis. La tuile de gauche comprend donc des biomes de Montagne, Rivière, et Plaines, tandis que celle de droite ne contient que des biomes de Forêt.

Une tuile de montagne, rivière, et plaines



Une tuile de forêts

L'ensemble des tuiles vous est fourni avec le sujet, et est téléchargeable sous forme d'archive sur e-learning. Pour qu'ils soient plus simples à identifier, les fichiers contenant les images de tuiles suivent une convention de nommage précise. Leur nom est toujours composé de 4 lettres. La

première lettre correspond au biome du haut de la tuile, le deuxième à celui de droite, le troisième à celui du bas, et le dernier à celui de gauche.

Chacune de ces lettres correspond à un biome spécifique :

- **F** pour **F**orêt
- **M** pour **M**ontagne
- **P** pour **P**laine
- **R** pour **R**ivière
- **S** pour **S**ea (biome de mer)

On dispose de plus, afin de séparer les parties terrestres des parties aquatiques, d'un biome de côte. Afin de garder l'orientation des côtes cohérentes entre les tuiles, on dispose d'un biome spécifique pour chaque orientation de côte :

- **D** pour une côte située à **D**roite de l'île
- **G** pour une côte située à **G**auche de l'île
- **B** pour une côte située en **B**as de l'île
- **H** pour une côte située en **H**aut de l'île

Attention, les directions données correspondent aux positions des côtes par rapport à l'île. Par exemple, une côte **D**roite touchera forcément le haut ou le bas d'une tuile.

On donne ci-dessous quelques exemples de tuiles avec leur convention de nommage détaillée:

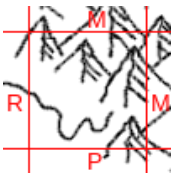



*Fig. 5a : La tuile* `MMPR`

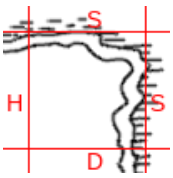



*Fig. 5b : La tuile* `SSDH`

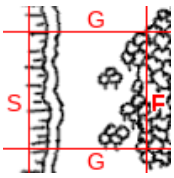



*Fig. 5c : La tuile* `GFGS`

Notez que l'archive fournie ne contient pas toutes les tuiles possibles. Par exemple, la tuile `MRFM`, qui correspondrait à une tuile de montagnes en haut et à gauche, une rivière à droite, et une forêt en bas, n'existe pas. L'absence de certaines de ces tuiles pourra causer des cartes impossibles à compléter.

## Travail attendu et évaluation

Le projet est constitué de quatre tâches principales. Chaque tâche contient plusieurs niveaux de perfectionnement. Vous devez avoir réalisé au moins le niveau 1 de toutes les tâches obligatoires avant d'aborder les améliorations facultatives. Les niveaux au-dessus du niveau 1 sont volontairement moins guidés : à vous de concevoir une solution adaptée !

Pour chacune des tâches, on donne des indications d'algorithmes et de structures de données adaptés au problème. Celles-ci sont données à titre indicatif : vous êtes libre de ne pas les suivre, mais assurez-vous de justifier vos choix et de les valider auprès de vos encadrants ou encadrantes de TP avant de vous lancer dans leur réalisation.

Il sera également possible d'ajouter plusieurs fonctionnalités additionnelles à *MapMaker*. On détaille ces améliorations possibles à la fin de ce sujet.

## Tâche 1 : Lecture de fichier et parcours de dossier

La première tâche consiste à parcourir les fichiers fournis pour récupérer toutes les tuiles disponibles.

On propose de construire un dictionnaire dont les clefs sont les noms des tuiles (par exemple, `'MMPR'` pour la tuile de la Fig. 5a) et les valeurs associées sont les chemins d'accès relatifs des images des tuiles correspondantes (par exemple, `'tuiles/MMPR.png'`).

*Indication* : la fonction `listdir` du module `os` permet de récupérer une liste de tous les fichiers se trouvant dans un répertoire donné.

Implémentez une fonction `cree_dico(chemin)` qui renvoie ce dictionnaire à partir du `chemin` d'accès du répertoire contenant les tuiles.

**Attention** : pour implémenter les niveaux supérieures de certaines tâches, il pourra être nécessaire d'enrichir cette structure de données.

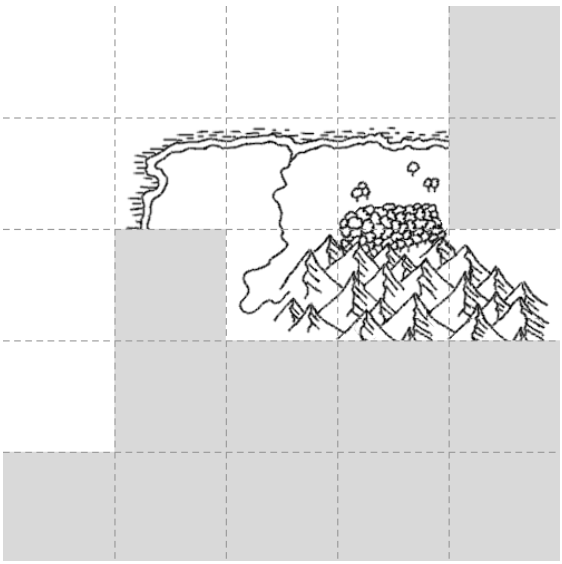
## Tâche 2 : Moteur et placement des tuiles

La deuxième tâche du projet consiste à programmer la logique interne de *MapMaker* (le *moteur*), c'est-à-dire la partie qui permet de représenter l'état de la carte dans des structures de données appropriées, et de les modifier durant l'utilisation du programme.

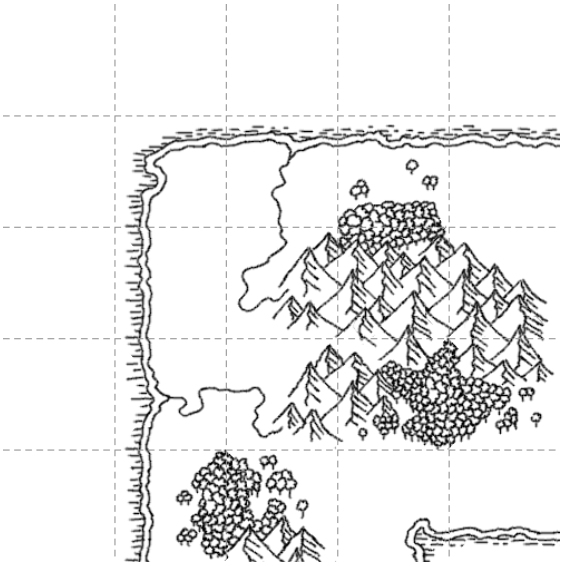
### Représentation de l'état de la carte

La **grille** est représentée par une liste de listes de chaînes de caractères. Les dimensions des listes correspondent aux dimensions de la carte représentée (initialement : 10 par 10). La valeur `grille[i][j]` correspond au nom de la tuile présente dans la case `(i, j)` de la grille. On utilisera la valeur `None` pour représenter une case non-remplie.

Par exemple, la figure ci-dessous montre deux grilles de taille $5 \times 5$ et leurs représentations par des listes de listes.



Grille 1



Grille 2

```
[['SSSS', 'SSSS', 'SSSS', 'SSSS', None  ],
 ['SSSS', 'SHGS', 'SHRH', 'SHFH', None  ],
 ['SSSS', None  , 'RMPP', 'FMMM', 'PPMM'],
 ['SSSS', None  , None  , None  , None  ],
 [None  , None  , None  , None  , None  ]]
```

```
[['SSSS', 'SSSS', 'SSSS', 'SSSS', 'SSSS'],
 ['SSSS', 'SHGS', 'SHRH', 'SHFH', 'SHPH'],
 ['SSSS', 'GPGS', 'RMPP', 'FMMM', 'PPMM'],
 ['SSSS', 'GRGS', 'PMPR', 'MFPM', 'MPPF'],
 ['SSSS', 'GFGS', 'PPMF', 'PBDP', 'PBSB']]
```

### Gestion des tuiles

Le moteur doit permettre de détecter les tuiles valides dans chaque case, de placer des tuiles, et d'en retirer.

- **Niveau 1 : vérification des raccords**

  Pour vérifier qu'une tuile peut être posée dans une case, il faut s'assurer que chaque biome apparaissant sur un des côtés de cette tuile coïncide avec les biomes apparaissant sur les tuiles adjacentes :

  - Le biome du haut de la tuile doit correspondre au biome du bas de la tuile supérieure.
  - Le biome de droite de la tuile doit correspondre à celui de gauche de la tuile adjacente.
  - Le biome de gauche de la tuile doit correspondre à celui de droite de la tuile adjacente.
  - Le biome du bas de la tuile doit correspondre à celui du haut de la tuile inférieure.

  Pour ce faire, on propose d'implémenter au minimum les fonctions suivantes :

  - `emplacement_valide(grille, i, j, nom_tuile)` : Vérifie si la tuile `nom_tuile` se connecte correctement aux tuiles déjà posées dans les cases voisines de la case `(i, j)`. Renvoie `True` si c'est le cas et `False` sinon.
  - `tuiles_possibles(grille, i, j)` : Renvoie la liste de toutes les tuilles qui peuvent être positionnées à la case `(i, j)` dans la grille en respectant les règles d'adjacence.

- **Niveau 2 : rivières naturelles**

  Pour ce niveau, on veut imposer une restriction supplémentaire sur la construction des rivières, qui garantit qu'elles suivent des règles de génération vraisemblables : les rivières prennent leur source dans une montagne, ne se séparent jamais en deux, peuvent éventuellement rejoindre d'autres rivières, et finissent par se jeter dans la mer.

  Les contraintes à vérifier sont donc les suivantes :

  - Les rivières ne peuvent pas former de boucles.
  - Toute rivière doit démarrer dans une montagne (ou hors de la carte).
  - Toute rivière doit terminer dans la mer (ou hors de la carte).
  - Une rivière peut rejoindre une autre rivière, mais ne peut pas se séparer en deux.

  *Indication* : pensez à la coloration de zones !

  La figure 6a donne quelques exemples de rivières **impossibles**.

Fig. 6a : pas de source Fig. 6b : une séparation Fig. 6c : une boucle

**Attention**, cette amélioration doit être compatible avec le solveur (tâche 4) et peut grandement en affecter les performances. Prévoyez une méthode simple pour activer ou désactiver la gestion des rivières.

- **Niveau 3 : décors**

  Pour ce niveau, on veut offrir la possibilité d'améliorer le visuel de la carte en y plaçant des décors. Dans chaque biome, certains décors pourront être ajoutés. L'archive sur e-learning fournit, en plus de la liste des tuiles, une liste des décors suivants, chacun spécifique à un certain biome :

  - Des vagues, un bateau, une sirène et un monstre marin pour les biomes de mer.
  - Des chateaux, de l'herbe, des moutons et des champs pour les biomes de plaine.

  Notez que tous les décors sont plus petits que les tuiles : il est donc attendu de pouvoir les placer plus finement que dans les cases de la grille. On s'assurera qu'ils peuvent apparaître à plusieurs endroits à l'intérieur d'une tuile, voir à cheval sur plusieurs tuiles. Par contre, les décors ne doivent jamais recouvrir un élément déjà dessiné (rivière, montagne, arbre, autre décor…)

  Deux fonctionnalités sont attendues :

  1. Placement manuel de décor pendant la conception de la carte
  2. Placement automatique de décors respectant les règles sur une carte finie

*Indication* : attention, cette amélioration nécessite d'enrichir les structures de données utilisées pour représenter l'état de la carte, et a des répercussions sur l'interface graphique.

## Tâche 3 : Affichage graphique et interface utilisateur

**Toute la partie graphique du projet doit être réalisée avec le module `fltk`.**

Le but de cette tâche est de permettre l'**affichage** graphique de la carte. Pour avoir un affichage continu de la carte, il est important que les délimitations entre les différentes cases de la grille ne soient pas visibles sur l'affichage. Les tuiles sont conçues de sorte à ce que tous les raccords se fassent de façon fluide.

L'utilisateur pourra, par de simples clics, ajouter des tuiles case par case en les sélectionnant parmi une liste de tuiles compatibles. On s'assurera aussi de pouvoir supprimer des tuiles, par exemple à l'aide d'un clic droit. La gestion des décors (si réalisée à la tâche 2), et le lancement du solveur, détaillés dans les autres parties, devront être accessibles facilement, soit via un bouton sur l'interface graphique, soit via des touches de clavier.

- **Niveau 1** : Lors d'un clic gauche sur une case, une tuile compatible est choisie au hasard et placée à l'emplacement correspondant. Si aucune tuile ne convient, un message s'affiche pour avertir l'utilisateur.
- **Niveau 2** : Lors d'un clic, un sous-menu apparaît avec une sélection (aléatoire s'il y en a trop) de tuiles compatibles. L'utilisateur peut alors sélectionner n'importe laquelle de ces tuiles pour l'ajouter à la carte.
- **Niveau 3** : Amélioration du niveau 2 : s'il y a trop de tuiles, le menu contient un système de défilement pour parcourir toutes les tuiles disponibles.

## Tâche 4 : Complétion automatique

Le but de cette tâche est de programmer un "solveur", à savoir une procédure permettant de **compléter automatiquement une carte** partiellement remplie.

Pour effectuer cette complétion automatique, on implémentera un algorithme récursif de complétion appelé algorithme de recherche *par backtracking*, qui essayera de compléter la carte case par case, en essayant d'ajouter une tuile valide à chaque étape. Si on arrive sur une case dans laquelle on ne peut placer aucune tuile, on annulera la dernière modification effectuée en revenant en arrière pour tenter une autre piste (c'est le *backtracking*). Si toutes nos tentatives échouent, on en déduira qu'il n'y a pas de solution possible pour compléter la carte, et on renverra un message d'erreur.

L'algorithme procède comme suit, à partir d'une grille `grille` donnée :

1. On vérifie si toutes les cases sont remplies. Si c'est le cas, on a réussi à trouver une solution, et on renvoie `True`.
2. Sinon, on choisit une case `(i, j)` vide à remplir.
3. On récupère la liste `lst_tuiles` de toutes les tuiles que l'on peut placer dans la case `(i, j)`.
4. On parcourt de manière itérative chaque tuile `t` dans `lst_tuiles` comme suit:

- On ajoute `t` dans la case `(i, j)` de `grille`.
- On relance récursivement la recherche de solution sur la grille. Si elle réussit, on renvoie `True`. Sinon, on passera à la prochaine étape de la boucle, où on sélectionnera une autre tuile `t`.

*Notez que s'il n'y a aucune tuile possible à placer dans la case `(i, j)`, cette étape se finit instantanément, et on passe immédiatement à l'étape 5 (c'est un cas d'échec).*

5. Si on sort de la boucle, c'est qu'on n'a pas réussi à trouver de solution. Dans ce cas, on vide la case `(i, j)` en la réinitialisant à `None` et on renvoie `False`. (*On revient en arrière dans la recherche : c'est l'étape de backtracking*)

Pour cette tâche, on propose deux axes d'amélioration différents et complémentaires.

### Axe 1 : Complexité et choix des tuiles

La façon de sélectionner la prochaine case lors de l'étape 2 peut grandement influencer la complexité de l'algorithme.

- **Niveau 1 : sélection séquentielle**

  Pour l'étape 2, on se contente de reparcourir la grille de haut en bas, et de la gauche vers la droite, jusqu'à trouver une case vide.

- **Niveau 2 : sélection séquentielle efficace**

  Pour éviter le parcours systématique de la grille, on pourra proposer une solution pour trouver efficacement la prochaine case à remplir. Vous pourrez ajouter un paramètre supplémentaire à la fonction.

- **Niveau 3 : sélection de la case la plus contrainte**

  Ce n'est en fait pas le parcours de la grille qui est coûteux en temps, mais le nombre d'appels récursifs. Pour les limiter au maximum, on pourra s'assurer de sélectionner lors de l'étape 2 la case **la plus contrainte**, c'est-à-dire celle pour laquelle il y a le moins de tuiles possibles.

- **Niveau 4 : sélection efficace de la case la plus contrainte**

  On veut améliorer le niveau précédent de manière à éviter de recalculer systématiquement la liste des tuiles possibles pour chaque case, ce qui est particulièrement dommage pour les cases dont les voisines n'ont pas changé. Concevez une structure de données adaptée pour réaliser cette amélioration. Vous pourrez ajouter un paramètre supplémentaire à la fonction.

### Axe 2 : Utilisation du solveur

- **Niveau 0 : Génération de carte aléatoire**

  L'utilisation la plus simple du solveur est de le lancer sur une grille initialement vide, de façon à générer une carte aléatoire respectant les contraintes.

- **Niveau 1 : Complétion de carte**

  L'utilisation souhaitée du solveur est pour la complétion de carte. Il devra être possible pour l'utilisateur de commencer la création manuelle d'une carte, puis à tout moment d'exécuter le solveur pour compléter de façon cohérente la carte en cours. L'appel du solveur ne doit pas modifier les tuiles déjà placées, et doit renvoyer un message d'erreur si la carte courante ne peut pas être complétée.

- **Niveau 2 : Défilement infini**

  On souhaite utiliser le solveur pour donner l'illusion d'une carte infinie. L'utilisateur peut faire défiler la carte dans une des directions cardinales, et le solveur est exécuté pour générer automatiquement la suite de la carte (*voir vidéo de démonstration sur la chaine du projet*).

  Attention, l'ordre du parcours influence grandement la fluidité du défilement. Il est attendu que le défilement se fasse sans temps de chargement visible. *Indication* : vous pourrez implémenter une version spécifique du solveur pour cette fonctionnalité.

- **Niveau 3 : Assistant de conception**

  Pour ce dernier niveau, le solveur est utilisé comme un assistant pendant la conception de la carte. Le solveur indiquera visuellement les cases vides les plus contraintes et avertira automatiquement l'utilisateur dès que la carte en cours ne peut plus être complétée.

  Attention, cette fonctionnalité ne doit pas dégrader les performances du reste du programme.

## Tâches complémentaires

Cette section propose des améliorations du projet, à aborder uniquement si toutes les tâches obligatoires ont été terminées **au moins au niveau 1**. Les étoiles ($\star$) devant chaque tâche donnent une indication sur leur difficulté.

- Ajouter un système de sauvegarde de cartes. On peut en envisager deux :

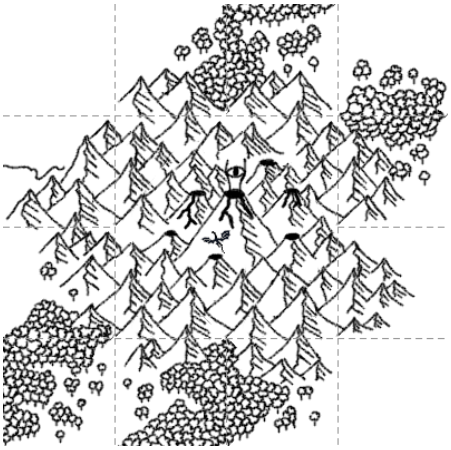
  1. ($\star$) Un système de sauvegarde des cartes partiellement remplies, que l'on puisse ouvrir a posteriori pour reprendre l'édition.
  2. ($\star$) Un système de sauvegarde des images de cartes, pour pouvoir exporter les images sous un format réutilisable (png, jpg, …)

- Gérer l'affichage sur une grille de taille paramétrable :

  1. ($\star$) Le programme gère des grilles de taille différentes de $10 \times 10$ ou rectangulaire, mais qui peuvent toujours être affichées en entier dans une fenêtre.
  2. ($\star\star$) Le programme gère des cartes de très grandes dimensions (par ex, $50 \times 50$) en proposant à l'utilisateur un moyen de zoomer sur la partie qu'il édite, et de dézoomer pour voir l'ensemble de la carte. *Attention : le redimensionnement des images par* `fltk` *nécessite la bibliothèque* `PIL`*.*

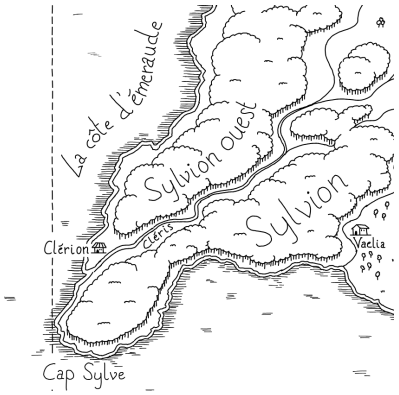
- ($\star$) Ajouter une visualisation au solveur. On affichera à chaque étape la case en cours de résolution, et on s'assurera de ralentir assez les affichages pour que l'on puisse suivre à l'oeil les étapes de l'algorithme. *Cette fonctionnalité devra pouvoir être facilement activée ou désactivée.*

- (⋆) Ajouter une prévisualisation du résultat de la complétion automatique. Lorsque l'utilisateur demande une complétion automatique, les tuiles proposées par le solveur sont indiquées par un visuel différent. L'utilisateur peut alors soit valider la proposition du solveur, soit demander une solution différente.
- (⋆⋆) Ajouter une implémentation de "méga-tuiles". On pourra ajouter à la liste des tuiles certaines tuiles composées de plusieurs cases pour enrichir le visuel de la carte finale. On fournit une de ces "méga-tuiles" en exemple. C'est une tuile de taille $2 \times 2$, dont tous les biomes aux bords sont des montagnes. *Cette fonctionnalité demande d'enrichir votre structure de données et doit être compatible avec le solveur.*



Une petite carte avec la méga tuile au centre.

- (⋆⋆) Ajouter une fonctionnalité permettant de tracer automatiquement des routes entre des points d'intérêts sur la carte, par exemple, entre les chateaux et les villages. Ces routes ne doivent pas couper d'autres éléments de décors (rivières, arbres, moutons…).
- (⋆⋆) Améliorer la fonctionnalité "défilement infini" de sorte à mémoriser les pans de cartes déjà générés. Il doit être possible de retourner en arrière et de revoir à l'identique les parties précédentes de la carte.
- (⋆⋆) Ajouter des contraintes de placement des décors pour obtenir un résultat plus naturel. Par exemple : les villes et villages doivent être proches d'un point d'eau, les champs et moutons doivent être proches d'un village, il ne doit pas y avoir deux villes trop proches l'une de l'autre, etc…
- (⋆⋆⋆⋆) On veut détecter automatiquement les points d'intérêts principaux de la carte (chaînes de montagnes couvrant plus de $n$ tuiles, groupements de plusieurs villages, etc.), leur attribuer un nom généré aléatoirement, et afficher élégamment le nom sur la carte. Par exemple, l'affichage devra suivre la géométrie de l'élément : il ne devra pas cacher d'autres éléments importants, et épouser la forme du paysage. *Indication : c'est beaucoup plus difficile qu'il n'y paraît !*




Un exemple de noms épousant la géométrie de la carte.
*Cet exemple n'a pas été généré avec MapMaker !*

- ($\star$ – $\star\star\star\star$) Créez un petit jeu à jouer sur votre carte du monde. Soyez créatifs. L'évaluation de cette amélioration dépend de la difficulté des fonctionnalités ajoutées. N'hésitez pas à en discuter avec votre chargé(e) de TP.

Toute autre amélioration est possible selon vos envies. N'hésitez pas à en parler avec vos encadrant(e)s, qui pourront vous conseiller sur la manière de la réaliser et sur sa difficulté.