Projet : Wall is you.

Le but de ce projet est de réaliser un jeu nommé *Wall Is You*. Dans un scénario classique, le joueur incarne un aventurier courageux qui doit s'engouffrer dans un donjon sinistre pour vaincre les créatures maléfiques qui l'occupent ; mais dans *Wall Is You*, les rôles sont inversés : le joueur joue le rôle d'un donjon bienveillant qui doit aider un aventurier particulièrement stupide à le débarrasser des monstres qui l'habitent. Le jeu alterne entre les tours du donjon, contrôlé par le joueur, et les tours de l'aventurier, contrôlé par le programme, qui doit réaliser les actions lui ayant été dictées par le donjon.

Le projet est à réaliser en binôme et en plusieurs phases. Lisez bien le sujet dans son intégralité avant de commencer le travail, car les choix que vous effectuerez au début auront un impact sur la suite. Ne vous inquiétez pas des notions du langage Python que vous ne connaîtriez pas encore : elles seront vues au fur et à mesure au cours.

1 Aperçu du jeu

Voici les grandes lignes du déroulement du jeu, afin de réaliser une première version basique que vous serez amenés à faire évoluer par la suite.

1.1 Déroulement d'une partie

Comme précisé dans l'introduction du sujet, chaque tour fait alterner les actions du donjon et celles de l'aventurier : le donjon modifie éventuellement ses salles, et donne ses instructions à l'aventurier, qui doit ensuite les exécuter. Voici plus de détails sur leurs rôles et leurs caractéristiques.

Le donjon. Le donjon est une grille rectangulaire de salles, avec les propriétés suivantes :
- chaque salle comporte au moins un passage vers chacune des quatre directions cardinales (haut, droite, bas, gauche) dont l'encodage sera précisé en section 1.2.3 ;
- chaque salle peut contenir un dragon ;
- chaque dragon a un niveau, et tous les dragons d'un donjon sont de niveaux différents.

Au début du tour d'un joueur, le joueur peut cliquer une ou plusieurs fois sur autant de salles qu'il le souhaite pour les faire pivoter. Chaque clic fait pivoter la salle de 90° dans le sens horaire. Plusieurs clics successifs permettent donc de faire pivoter la salle de 180°, ou 270°, ou de la ramener dans sa position d'origine, comme le montre la figure 1. Le joueur indique la fin du tour du donjon en appuyant sur la touche Espace.

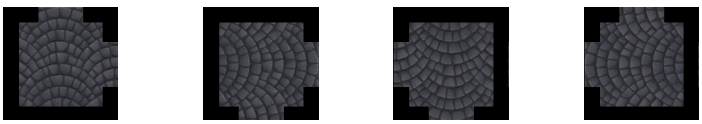


FIGURE 1 – Les quatre orientations possibles d'une salle en L.

L'aventurier. L'aventurier se trouve dans l'une des salles du donjon et commence au niveau 1. Il cherche toujours à aller vers le dragon le plus fort qui lui est accessible. À chaque instant, l'aventurier indique son intention par une ligne rouge. Par exemple, sur la figure 2, l'intention de l'aventurier est d'affronter le dragon de niveau 3. Si aucun dragon n'est accessible, l'aventurier n'a pas d'intention.

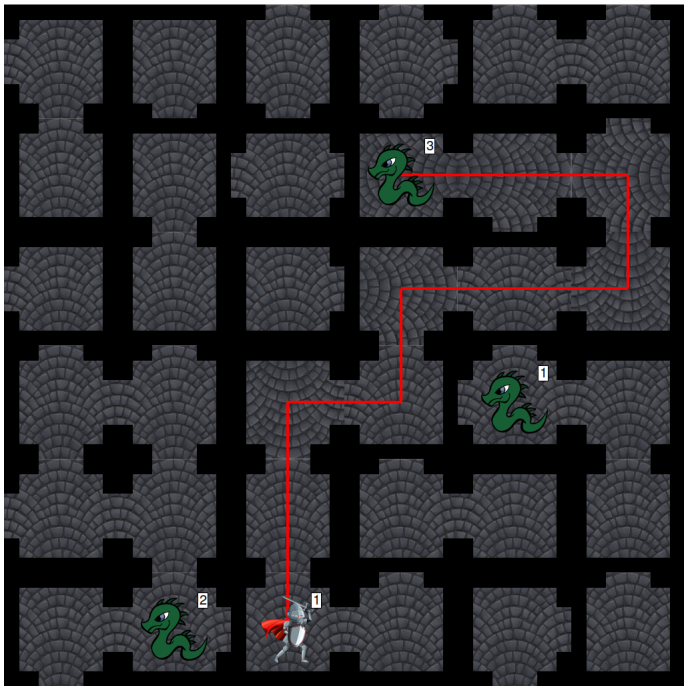


FIGURE 2 – Une partie en cours.

À son tour, l'aventurier se déplace de salle en salle le long de la ligne rouge, jusqu'à atteindre un dragon. S'il rencontre un dragon de niveau inférieur ou égal au sien, il le tue et gagne un niveau. S'il rencontre un dragon de niveau supérieur, il est tué et la partie est perdue. La partie est gagnée lorsque tous les dragons ont été tués.

1.2 Réalisation

Pour faciliter la réalisation de cette première phase, voici une découpe en tâches principales, toutes obligatoires.

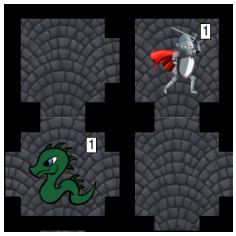
1.2.1 Tâche 1 : Réalisation du moteur de jeu

La première tâche du projet consiste à programmer la logique interne du jeu (le modèle), c'est-à-dire la partie qui permet de représenter l'état du jeu dans des structures de données appropriées, et de les modifier pour appliquer les effets des actions du joueur et de l'aventurier.

Représentation de l'état du jeu. Voici une suggestion de structure de données permettant de représenter l'état du jeu `(donjon, personnages)`. Vous êtes toutefois libres d'en changer, à condition d'expliquer et de motiver vos choix.

Chaque salle du donjon est représentée par un tuple de 4 booléens (haut, droite, bas, gauche), indiquant les sorties possibles pour cette salle. Le donjon est représenté par une liste `donjon` de listes de salles. Les dimensions des listes correspondent aux dimensions du donjon représenté. La valeur `donjon[i][j]` décrit la salle se trouvant à la ligne `i` et à la colonne `j`. La figure 3 donne un exemple de donjon avec son encodage associé.

Ainsi, la valeur `donjon[1][0]` représente la salle en bas à gauche du donjon (celle contenant le dragon). Dans l'exemple, elle vaut `(True, True, False, True)` pour indiquer que la salle permet les passages vers le haut, la droite et la gauche, mais pas vers le bas. Chaque personnage (aventurier ou dragon) est représenté par une liste de deux éléments : le premier indique sa position à l'aide d'un couple `(ligne, colonne)`, et le second indique son niveau par



```
donjon = [
    [(False, True, True, False), (False, False, True, False)],
    [(True , True, False, True), (True , True, True, False)]
]
```

FIGURE 3 – Un tout petit donjon, et son encodage.

un entier. Par exemple, l'aventurier de la figure 3 est représenté par `aventurier = [(0,1), 1]`. Comme le donjon peut contenir plusieurs dragons, les listes correspondant à chaque dragon sont regroupés dans une liste `dragons`.

Gestion du donjon. Le moteur du jeu doit permettre de faire pivoter les salles du donjon et de savoir facilement si deux salles sont connectées. Il vous faudra donc prévoir les fonctions adéquates pour ce faire.

Intention de l'aventurier. La partie la plus délicate du moteur du jeu est le calcul de l'intention de l'aventurier, c'est-à-dire le chemin qu'il compte emprunter lorsque le donjon aura fini de jouer. Dans cette première version de base, vous tracerez vous-même dans l'interface, à l'aide de la souris, le chemin que devra suivre l'aventurier, qui sera encodé par une liste de positions successivement connectées. Dans une phase ultérieure, ce calcul devra être réalisé automatiquement. Lors de son tour, l'aventurier suit son intention, et en applique les conséquences.

1.2.2 Tâche 2 : Interface graphique

Le but de cette tâche est de programmer l'interface graphique du jeu (la vue). Une interface ergonomique est attendue. Au lancement du programme, le joueur devra être accueilli par un menu lui permettant de charger un donjon de son choix parmi un ensemble de donjons disponibles. Une fois le jeu lancé, le joueur pourra jouer en cliquant sur les salles à faire pivoter. À chaque coup joué, le programme se chargera de mettre à jour l'intention de l'aventurier et l'affichage en conséquent. Le joueur indiquera ensuite la fin de son tour en appuyant sur la touche Espace, et le programme se chargera de jouer et d'afficher le tour de l'aventurier. Le joueur pourra recommencer le donjon choisi depuis le début avec la touche R, et revenir au menu pour choisir un autre donjon avec la touche Esc. En cas de victoire ou de défaite, un message approprié doit s'afficher et proposer au joueur de revenir au menu. L'aspect esthétique est laissé à votre appréciation ; en cas de besoin, vous pouvez vous servir d'éléments graphiques disponibles en accès libre [1]. Pensez bien à utiliser une image par salle.

1.2.3 Tâche 3 : Représentation et chargement des donjons

Le but de cette tâche est de permettre au programme de lire et de charger des donjons prédéfinis et stockés dans des fichiers. Ainsi, un donjon de *Wall Is You* sera représenté par un fichier texte constitué de deux parties :

1. La première partie représente l'agencement du donjon, dessiné avec les caractères spéciaux ╬, ╠, ╔, ║, ╦ et leurs rotations [2]. Chacun de ces caractères représente une salle avec les passages existants vers d'autres salles. Par exemple :
   — ╬ : les quatres salles adjacentes sont accessibles ;

1. Voir par exemple https://opengameart.org/.
2. Voir https://en.wikipedia.org/wiki/Box-drawing_characters.

— ╔ : on ne peut aller que vers la droite ou vers le bas ;
— ╥ : on ne peut aller que vers le bas.
Voici tous les caractères utilisés avec leurs rotations :
— ╬ : 4 passages ;
— ╠ ╦ ╣ ╩ : 3 passages ;
— ╔ ╗ ╚ ╝ ═ ║ : 2 passages ;
— ╥ ╨ ╡ ╞ : 1 passage : le double trait indique l'unique passage autorisé.

2. La seconde partie donne les coordonnées et niveaux des personnages. Par exemple, A 2 5 indique que l'aventurier se trouve à la ligne 2 et à la colonne 5, tandis que D 2 0 2 indique que la salle à la ligne 2 et colonne 0 contient un dragon de niveau 2. On ne précise pas le niveau de l'aventurier, qui commence toujours au niveau 1. La figure 4 montre un exemple de fichier avec le donjon correspondant.

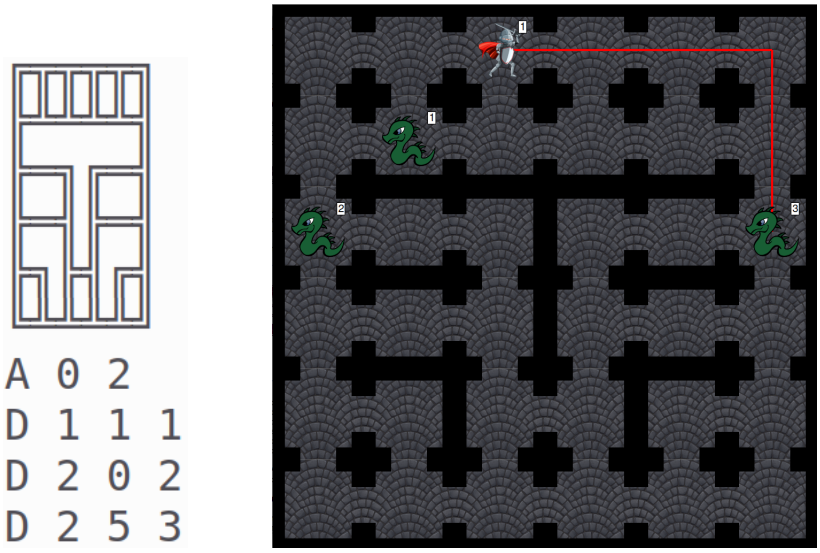


FIGURE 4 – Le fichier d'un donjon, et le donjon correspondant. Attention, les caractères **ne représentent pas les murs**, mais bien les sorties possibles pour une position donnée.

Votre programme doit permettre de lire des fichiers écrits dans le format spécifié ci-dessus.

1.2.4 Tâche 4 : Calcul automatique de l'intention

L'intention de l'aventurier doit être calculée automatiquement par le donjon une fois les éventuelles reconfigurations de salles terminées, Pour ce faire, on propose d'écrire une fonction `intention(donjon, position, dragons)` renvoyant un chemin (une liste de positions successivement connectées) menant l'aventurier jusqu'au dragon accessible de plus haut niveau, ou `None` si aucun dragon n'est accessible.

Une fois cette tâche réalisée, l'intention de l'aventurier doit toujours être affichée si un dragon est accessible, et doit également se mettre à jour automatiquement après reconfiguration des salles.

2 Variantes

Dans un deuxième temps, vous devrez fournir une version du jeu avec un menu initial qui permet d'utiliser une combinaison des alternatives suivantes. Les choix d'implémentation sont laissés à votre appréciation.

1. **Trésors** : Le jeu présenté ne laisse en fait que très peu de choix au joueur : il s'agit simplement de trouver un chemin allant de l'aventurier au premier dragon, puis du premier dragon au deuxième, etc. On souhaite enrichir le jeu en ajoutant des trésors, régis par les règles suivantes :
   — Le donjon dispose d'un certain nombre de trésors, fixé à l'avance.
   — Pendant son tour, le donjon peut, par un clic droit, placer un trésor dans une salle inoccupée. Il ne peut y avoir au plus qu'un seul trésor présent dans le donjon.
   — L'aventurier est cupide, et est attiré en priorité par les trésors, même si des dragons sont accessibles.
   — Lorsque l'aventurier rencontre un trésor, il le récupère (le trésor est définitivement retiré) et termine son tour.

   Les trésors sont donc un outil pour le donjon, lui permettant d'attirer l'aventurier dans une salle de son choix, et donc de réaliser des itinéraires normalement impossibles. Dans l'exemple ci-dessous, le joueur a placé un trésor pour attirer l'aventurier jusqu'à la dernière salle de la dernière ligne, puis a ensuite pivoté cette salle pour forcer l'aventurier à combattre le dragon de niveau 1 en premier. On peut facilement se convaincre que ce donjon ne peut pas être résolu sans trésor : en effet, l'aventurier veut toujours combattre le en premier le dragon le plus fort parmi ceux qui sont accessibles, mais sera tué par le dragon de niveau 2 qui est plus fort que lui. En revanche, l'envoyer vers un trésor puis lui barrer l'accès au dragon de niveau 2 permet de l'obliger ensuite à se battre contre le dragon de niveau 1.

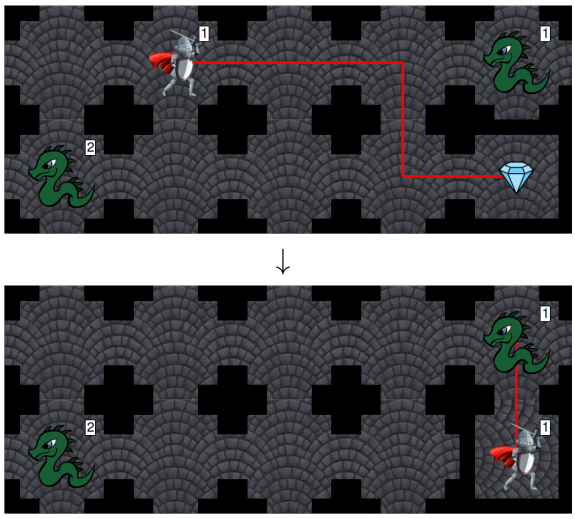


FIGURE 5 – Utilisation d'un trésor.

   Cette amélioration requiert d'adapter les différentes structures de données et algorithmes proposés, et de proposer des nouveaux donjons prédéfinis avec un nombre de trésors approprié.

2. **Sauvegarde** : Permettre au joueur d'enregistrer une partie en cours ou de charger une partie enregistrée. L'enregistrement devra être fait dans un fichier de manière à pouvoir être récupéré lors d'une autre session de jeu.
3. **Déplacement des dragons** : après que l'aventurier ait joué son tour, et avant que le donjon ne prenne la main, chaque dragon qui subsiste se déplace d'une case dans une direction choisie aléatoirement parmi les directions accessibles. Si cela mène un dragon à rencontrer l'aventurier, les règles de base s'appliquent : un combat a lieu, et le jeu ne se poursuit que si l'aventurier en sort vainqueur.

3 Bonus et améliorations

Finalement, vous devrez implémenter au moins deux des améliorations suivantes :

1. **Édition de donjons** : Implémenter un éditeur de donjons, c'est-à-dire une interface graphique permettant à l'utilisateur de choisir la dimension du donjon, l'agencement des salles, et où placer les dragons et l'aventurier. Une fois la réalisation terminée, la grille sera enregistrée dans un fichier.
2. **Indices** : Implémenter une fonction "Indice", indiquant au joueur un chemin possible jusqu'au prochain dragon, même si les salles ne sont pas actuellement correctement tournées.
3. **Génération de niveaux** : Implémenter un générateur de niveaux aléatoires. Les niveaux générés doivent toujours admettre une solution.
4. **Mode "tour unique"** : dans ce mode, le donjon ne peut reconfigurer les salles qu'une seule fois. Une fois qu'il a terminé, l'aventurier effectue toutes ses actions sans aucune autre interaction jusqu'à la fin de la partie : c'est-à-dire qu'à chaque itération, il affiche son intention (calculée automatiquement), la réalise, et recommence jusqu'à ce qu'il ait tué tous les dragons ou se soit fait tuer.
5. **Nouvelles mécaniques de jeu** : Implémenter de nouvelles mécaniques de jeu qui n'attirent pas l'aventurier et n'interrompent pas son tour, mais sont activées lorsqu'il passe dessus. Par exemple : des leviers permettant d'ouvrir ou de fermer des portes, des élixirs permettant de vaincre le prochain dragon même s'il est de niveau supérieur, etc. Cette amélioration nécessite de proposer des donjons adaptés et intéressants.
6. **Plus courts chemins** : l'algorithme naïf utilisé pour le calcul automatique de l'intention de l'aventurier ne produit pas nécessairement le chemin le plus court pour rejoindre le dragon visé. Ce défaut conduit à des situations dans lesquelles l'aventurier semble faire des détours incompréhensibles, comme dans la figure 6.

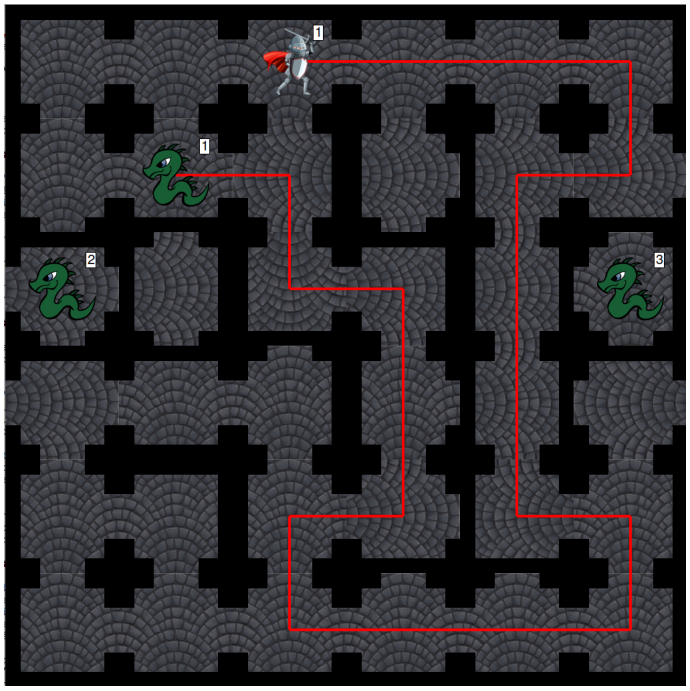


FIGURE 6 – Un détour inutile.

Proposez un nouvel algorithme permettant de trouver un chemin de longueur minimale vers le dragon de plus haut niveau.

4 Consignes de rendu

Le projet se déroulera en trois grandes phases, détaillées en sections 4.1, 4.2 et 4.3. Pour les rendus, il sera impératif de suivre les consignes précisées ci-dessous, sans quoi vous perdrez des points. En particulier :

- Ce projet est à réaliser en binôme (s'il est nécessaire de faire une exception, contactez les enseignants au plus vite). Vous devrez sélectionner votre binôme sur e-learning dans la semaine suivant la publication du sujet. Pour des raisons d'organisation, les binômes doivent se constituer de membres du même groupe de TP.
- **Vous DEVEZ utiliser `fltk`**. L'utilisation de modules autres que les modules standards de Python est interdite ; en cas de doute, n'hésitez pas à poser la question.
- L'utilisation de modules standards non couverts au cours est autorisée ; en revanche, par souci d'équité, les notions avancées non couvertes au cours (classes, décorateurs, …) sont interdites. En cas de doute, n'hésitez pas à poser la question.
- **Votre programme devra impérativement s'exécuter sans problème sur les machines de l'IUT.** Prévoyez donc bien de le tester sur ces machines en plus de la vôtre et d'adapter ce qui doit l'être le cas échéant (par exemple, les vitesses de déplacement), et faites-le suffisamment tôt. Il sera donc utile de permettre à l'utilisateur de préciser certaines options auxquelles vous attribuez des valeurs par défaut plutôt que de devoir modifier le code à chaque exécution.

4.1 Premier rendu

L'objectif général à atteindre pour le premier rendu est d'avoir un jeu fonctionnel et robuste avec :
- **la tâche 1 (section 1.2.1) et tâche 2 (section 1.2.2) ;**
- **au moins une variante de la liste donnée en section 2.**

Ne commencez pas les variantes de la section 2 avant d'avoir réalisé le jeu de base.

La date limite pour ce premier rendu est le **vendredi 21 novembre 2025 à 23h55**. Passé ce délai, la note attribuée à votre projet sera 0. Les séances de SAÉ de la semaine suivante seront consacrées à la vérification de votre rendu et à amorcer la seconde partie. Vous déposerez sur la plate-forme e-learning une archive contenant :

1. **les sources de votre projet**, commentées de manière pertinente (quand le code s'y prête, pensez à écrire les doctests). De plus :
   - les fonctions doivent avoir une longueur raisonnable ; n'hésitez pas à morceler votre code en plusieurs fonctions auxiliaires pour y parvenir ;
   - plus généralement, le code doit être facile à comprendre : utilisez des noms de variables parlants, limitez le nombre de tests, et veillez à simplifier vos conditions ;
   - quand cela se justifie, regroupez les fonctions par thème dans un même module ;
   - chaque fonction et chaque module doit posséder sa *docstring* associée, dans laquelle vous expliquerez leur fonctionnement, et pour les fonctions, ce que sont les paramètres ainsi que les hypothèses faites à leur sujet.
2. un **fichier texte** `README.txt` (ou `README.md`) expliquant :
   - l'organisation du programme ;
   - les choix techniques ;

— les éventuels problèmes rencontrés.
Vous pouvez y ajouter tous les commentaires ou remarques que vous jugez nécessaires à la compréhension de votre code.

Si le code ne s'exécute pas, la note de votre projet sera 0. Vous perdrez des points si les consignes ne sont pas respectées, et il va sans dire que si deux projets trop similaires sont rendus par 2 binômes différents, la note de ces projets sera 0.

4.2 Second rendu

L'objectif pour le second rendu est toujours d'avoir un jeu fonctionnel et robuste avec le jeu de base, ainsi que :
- **la tâche 3 (section 1.2.3) ;**
- **toutes les variantes de la section 2.**

Les consignes données pour le premier rendu restent valables pour le second rendu. La date limite pour le second rendu est le **vendredi 19 décembre 2025 à 23h55**.

4.3 Troisième rendu

L'objectif est toujours d'avoir un jeu fonctionnel et robuste avec le jeu de base et toutes les variantes, ainsi que :
- **la tâche 4 (section 1.2.4) ;**
- **un rapport synthétique ;**
- **au moins trois bonus de la section 3.**

Le rapport devra expliquer votre programme dans les grandes lignes et fournir une analyse de la complexitè des algorithmes principaux que vous utilisez.

La date limite pour le dernier rendu est le **vendredi 9 janvier 2026 à 23h55**.Des mini-soutenances seront organisées la semaine suivante, au cours desquelles vous ferez une démonstration sur une machine des salles de TP et serez interrogés sur le projet (les choix techniques et algorithmiques que vous avez faits, les difficultés rencontrées, l'organisation de votre travail, …). Les contributions de chaque participant seront évaluées, et il est donc possible que tous les participants d'un même groupe n'aient pas la même note. Aucune présentation PowerPoint ne sera demandée : seul le code et le programme en action seront examinés.

Les mini-soutenances auront lieu le **jeudi 15 janvier 2026**, suivant un horaire et un ordre de passage qui vous seront communiqués en temps voulu.