CODINGWEEK2026

TELECOM Nancy — 5 janvier au 9 janvier 2026

> “Vous n’êtes pas le héros…
> Vous êtes le seigneur du donjon qui construit les pièges et organise la défense.”

L’objectif de ce projet est de concevoir et développer un petit jeu de gestion/simulation intitulé *Dungeon Manager*. Le principe est volontairement “à contre-emploi” : vous ne jouez pas un aventurier qui explore un donjon, vous jouez la personne qui le conçoit et le défend.

Le jeu se déroule sur une grille 2D représentant le donjon. Le joueur construit un parcours en plaçant des éléments (au minimum des murs et des pièges), puis lance des vagues de héros (PNJ) qui cherchent à atteindre le trésor. Le joueur cherche à protéger le trésor tout en maximisant un score.

L’application doit proposer deux interfaces au choix : une interface en terminal (texte uniquement, mais ergonomique) et une interface Web (HTML/CSS/JavaScript, framework autorisé) servie par un serveur local ou distant. Quel que soit le choix d’interface, on attend une conception et une architecture modulaire (inspirée MVC), une utilisation sérieuse de Git, et un usage raisonné d’assistants d’IA de programmation, incluant un journal d’usage de l’IA et un retour critique de vos sessions *vibe coding*.

Règles du jeu (version minimale)

Le donjon

Le donjon est une grille 2D (par exemple 32 × 32) comportant une case entrée et une case trésor. Le joueur édite la grille en y plaçant des éléments. Les murs sont bloquants ; les pièges infligent des dégâts lorsqu’un héros passe sur leur case. Vous pouvez ajouter des monstres si vous le souhaitez (dégâts, portée, caractéristiques), à condition de conserver un moteur simple, testable et bien structuré.

Le joueur dispose d’un budget de construction. Chaque élément placé a un coût ; l’interface doit permettre de visualiser le budget restant et empêcher les placements qui dépassent le budget.

Les héros (PNJ)

Chaque héros possède des points de vie (PV) et une position (ligne, colonne). À chaque tour, il choisit une case voisine où se déplacer, puis subit les effets éventuels de la case (piège, monstre). Un héros meurt si ses PV tombent à 0 ou moins. La vague s’arrête lorsque tous les héros sont morts (victoire du donjon) ou lorsque au moins un héros atteint le trésor (défaite du donjon).

IA des héros

Vous devez implémenter au moins deux stratégies d'IA de déplacement. Une stratégie "plus court chemin" (BFS/Dijkstra simple ou heuristique) est un bon point de départ. Une seconde stratégie peut chercher un chemin "le moins dangereux", en pénalisant les cases piégées (quitte à faire un détour). Le joueur doit pouvoir choisir la stratégie utilisée pour la vague via l'interface.

Score et feedback

À la fin d'une vague, vous calculez un score (formule libre) et vous affichez un récapitulatif lisible. La formule peut par exemple récompenser le nombre de héros vaincus, pénaliser l'atteinte du trésor, et tenir compte du coût de construction.

Le récapitulatif doit au minimum indiquer le nombre de héros engagés, le nombre de héros vaincus, si le trésor a été atteint, et le score total.

Travail demandé

Fonctionnalités minimales (MVP obligatoire)

Vous devez livrer un MVP complet permettant d'éditer un donjon et de simuler une vague. L'éditeur doit afficher la grille, permettre de placer et retirer des éléments (murs, pièges, et éventuellement monstres) et afficher clairement le budget restant. La simulation doit créer un ou plusieurs héros à l'entrée, les déplacer tour par tour jusqu'à la mort ou l'atteinte du trésor, appliquer les dégâts, puis s'arrêter lorsque la condition de fin est atteinte.

Le joueur doit pouvoir choisir la stratégie utilisée pour la vague, et l'application doit fournir au moins deux stratégies réellement distinctes. Enfin, vous devez calculer un score en fin de vague, afficher un résumé clair, et proposer une persistance permettant de sauvegarder et recharger un donjon (JSON ou autre format textuel).

Fonctionnalités complémentaires (si le temps le permet)

Si votre MVP est stable et bien testé, vous pouvez enrichir votre jeu : différents types de héros, amélioration des pièges/monstres (niveaux, dégâts, portée), événements aléatoires, messages d'ambiance, ou d'autres stratégies d'I.A. Ces ajouts sont optionnels et ne doivent pas se faire au détriment de la qualité globale du livrable final.

Contraintes techniques

Langage et environnement

Le langage choisi doit être orienté objet (Java, Python, ou autre si validé par l'enseignant). Deux interfaces utilisateur sont obligatoires : l'une Web (framework autorisé) avec un serveur, l'autre en mode terminal (grille ASCII, menus…). Le moteur principal contenant la logique métier de votre application doit être partagé par les deux interfaces.

Architecture et patrons

Vous proposerez une architecture claire, inspirée de MVC : un modèle regroupant le donjon, les cases, les héros, les pièges, la simulation, le score et la persistance ; une vue (Web ou terminal) ; et un contrôleur orchestrant les actions utilisateur et le déroulement du jeu.

Vous pouvez faire usage des patrons de conception comme Stratégie (IA des héros), Observateur (ou équivalent) pour les mises à jour de la vue lorsque le modèle change, et Fabrique (ou Monteur) pour créer proprement héros et éléments. Vous fournirez un diagramme de classes UML et un à deux diagrammes de séquence (par exemple "édition d'une case" et "lancement d'une vague").

Qualité et tests

Vous écrirez des tests unitaires sur le moteur de simulation (sans UI) afin de valider, par exemple, l'application des dégâts, la mort/survie, le calcul du score et le comportement minimal d'une stratégie. Vous pourrez également si le temps vous le permet réaliser des tests de bout en bout de votre application Web.

Lorsque c'est possible, vous utiliserez un *linter/formatter* et vous réaliserez au moins une revue de code via Merge Request (ou équivalent).

Git et collaboration

Un dépôt Git a été créée pour chaque groupe déclaré sur la plateforme gitlab de l'école (https://gibson.telecomnancy.univ-lorraine.fr/). L'organisation du travail doit être visible, par exemple, via des issues (backlog minimal : tâches, bugs), des branches de fonctionnalités et des *Merge Requests*. L'historique doit refléter une participation effective et régulière de **tous** les membres.

Assistants d'IA et "vibe coding"

Vous êtes encouragés à utiliser des assistants d'IA de programmation, avec une exigence de recul critique. Vous devez dans ce cas tenir un journal d'usage `IA_USAGE.md` de l'IA, décrivant quelques cas représentatifs (contexte, prompt, réponse, ce qui est gardé/modifié/rejeté, justification).

Vous restez responsables de votre code : un copier-coller non compris (Internet/IA) pourra être pénalisé.

Organisation sur la semaine

Présence et assiduité

La semaine est bloquée pour vous permettre de vous concentrer sur cette épreuve. Votre groupe doit être joignable et présent aux points de synchronisation. Les modalités pratiques (salles, créneaux) seront précisées en début de semaine.

Travail et collaboration

Le développement se fait de façon itérative et incrémentale : une version intégrée et démontrable doit exister à la fin de chaque journée. Votre dépôt doit rester exploitable en permanence (structure claire, `README.md` à jour, scripts si nécessaire) et votre organisation doit être visible via une *roadmap* simple `ROADMAP.md`, des issues, et des *Merge Requests*. Les commits doivent être fréquents et attribués correctement.

Le développement du projet doit se faire en suivant une méthode agile de type Scrum simplifié où chaque jour correspond à un sprint. Pour chaque journée de travail, il est attendu qu'un compte rendu de la réunion de planification du sprint soit rédigé puis commité sur le dépôt Git **au plus tard à 9h30** (sauf le premier jour), sous la forme d'un fichier `SPRINT_PLANNING_DAY_X.md` (avec `X` le numéro du jour/sprint). **En fin de journée**, un compte rendu de la réunion de rétrospective devra également être produit et déposé dans le fichier `SPRINT_RETROSPECTIVE_DAY_X.md`. Par ailleurs, la roadmap du projet devra être maintenue dans le fichier `ROADMAP.md`, qui décrit le backlog global et la répartition des tâches entre les différents sprints, de façon à rendre visible l'évolution progressive du projet et les priorisations décidées par l'équipe.

L'ensemble de ces documents seront stockés dans un répertoire `docs/` situé à la racine de votre projet.

Piste d'organisation proposée (non obligatoire)

Jour 1 – Compréhension et éléments de conception

- Lecture du sujet, questions/réponses
- Définition des règles de jeu (détails, variantes)
- Identification des entités et du modèle de données
- Choix des technologies d'interface (Web/terminal)
- Croquis rapides des maquettes d'interface
- Mise en place du dépôt Git, réalisation d'un premier backlog des tâches

Jours 2 à 4 – Modèle, moteur de simulation, interfaces utilisateur, gameplay

- Conception et implémentation du modèle (donjon, cases, héros, pièges)
- Études et implémentations des stratégies d'IA
- Réalisation des tests unitaires
- Utilisation ponctuelle d'un assistant IA pour proposer/vérifier l'architecture (documenter l'usage), rédiger des tests unitaires…
- Session de *vibe coding* dédiée au refactoring

Jour 5 – Stabilisation et démonstration

- Nettoyage du code (dead code, commentaires, structure)
- Vérification des tests
- Finalisation de la documentation et du journal d'usage de l'IA
- Préparation d'une démonstration de 10–15 minutes

Communication avec l'équipe pédagogique

Afin de poser vos questions et de discuter durant la semaine, nous avons créé un serveur Discord. N'hésitez pas à y poser des questions sur le sujet ou sur des points techniques.

Pour rejoindre le serveur Discord, vous devrez utiliser le lien d'invitation suivant :

Évaluations et rendus

Rendus intermédiaires

Chaque groupe effectuera un rendu quotidien sous la forme de livraisons Git. Vous utiliserez la convention `RELEASE_DAY_1`, `RELEASE_DAY_2`, …, `RELEASE_DAY_5` (tag Git sur la branche principale). Chaque livraison doit être exécutable et opérationnelle.

Rendu final

Le rendu final sera étiqueté `RELEASE_FINAL`. Il devra comporter a minima :

- le code source (moteur + UIs),
- les tests unitaires,
- un `README.md` expliquant installation/lancement/usage,
- les documents de conception (UML, choix d'architecture, patrons) dans un répertoire `docs/`,
- la roadmap journalière suivie du projet `ROADMAP.md` et les comptes rendus de planification et de rétrospective,
- le journal d'usage de l'IA `IA_USAGE.md`,
- ainsi qu'une vidéo de démonstration d'une durée comprise entre 10 et 15 minutes dont le lien figure dans le fichier `DEMONSTRATION.md` présent sur le dépôt.

La vidéo de démonstration doit être scénarisée : elle ne consiste pas en une simple exploration au hasard de l'application, mais en la présentation claire d'un ou plusieurs scénarios d'usage représentatifs. Vous devez donc choisir un "fil conducteur" (par exemple : préparation d'une nouvelle vague, placement de pièges, lancement de la simulation, analyse du score) et montrer, dans un ordre logique, comment l'utilisateur interagit avec les différentes fonctionnalités de votre application. Pour la réalisation technique, vous pouvez par exemple utiliser un logiciel comme OBS Studio qui permet d'enregistrer le terminal ou le navigateur, et d'obtenir une vidéo fluide illustrant clairement le fonctionnement de votre projet.

Critères d'évaluation

Les critères portent notamment sur les fonctionnalités livrées (MVP opérationnel), le respect des méthodes (itérations, intégration, organisation), la collaboration (activité Git, MR, répartition), la qualité du code

(architecture, patrons, lisibilité), la mise en œuvre de tests, l'ergonomie, ainsi que la qualité du retour critique sur l'usage des assistants IA.

Annexes

Maquettes d'interface · exemples

Les schémas ci-dessous sont fournis à titre illustratif. Vous pouvez proposer autre chose.

```
========== Dungeon Manager (Terminal) ==============

Stratégie IA : [1] Plus court chemin
               [2] Chemin le moins dangereux
Budget restant : 120 or

Légende :  E=Entrée  T=Trésor  •=Sol  #=Mur
^=Piège  M=Monstre

    0 1 2 3 4 5 6 7 8 9
 0  E . . # ^ . . . . T
 1  . . . . # . ^ . . .
 2  . # . . . . . . . .
 3  . . . ^ . . # . . .
 4  . . . . . . . . . .
 5  . . . . . . . . . .
 6  . . . . . . . . . .
 7  . . . . . . . . . .
 8  . . . . . . . . . .
 9  . . . . . . . . . .

Actions :
 [P] Placer un élément  [S] Sauvegarder  [C]
Charger
 [L] Lancer la vague    [Q] Quitter

> Choix de l'action :
```

```
=============== Simulation - Vague 1 ===============

Tour 1 :
E H . # ^ . . . . T
. . . . # . ^ . . .
...

Héros 1 (PV=100) en (0,1)
Héros 2 (PV=80)  en (0,1)

[Entrée pour continuer, Q pour arrêter] _
```

Listing 1: Exemple terminal (éditeur + simulation)

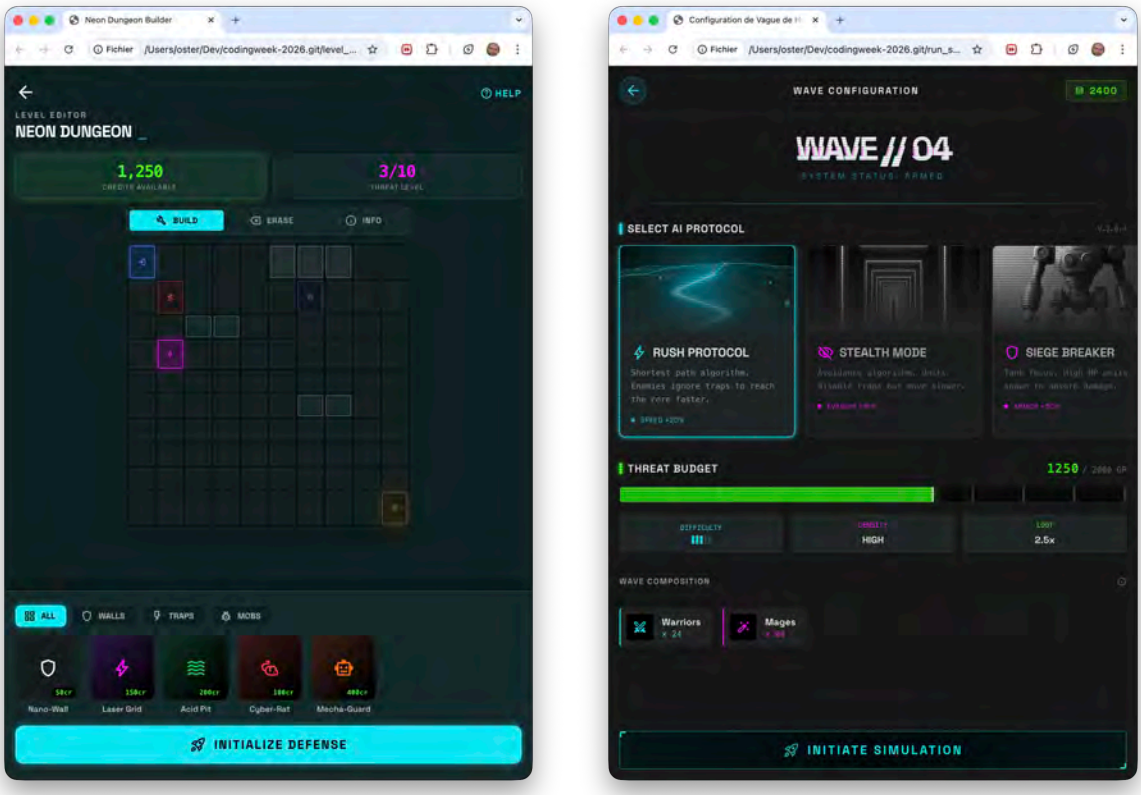

Figure 1: Exemple Web (éditeur + simulation)

Exemple de journal d'usage de l'IA (non obligatoire)

```
## Interaction #42

**Contexte :** calcul du score de vague.

**Prompt :**
"Nous avons la fonction compute_score(wave_stats)... (coller le code).
Propose 5 cas de test unitaires sous forme de tableau."

**Réponse IA :**
(tableau résumé, pas besoin de tout recopier)

**Décision :**
- Gardé : tests 1, 2, 4 (raison : couvrent cas normal, 0 héros, score négatif).
- Modifié : test 3 (mauvaise compréhension des pénalités).
- Rejeté : test 5 (hors scope du projet).

**Remarques :**
L'IA avait oublié le cas "budget négatif". Nous l'avons ajouté manuellement.
```

Exemple de format de roadmap (non obligatoire)

```
# ROADMAP – Projet Dungeon Manager

> Ce document décrit la vision globale du projet et le découpage en sprints (1 jour = 1 sprint).
> Il doit rester à jour tout au long de la semaine.

## 1. Vision du projet

- Objectif général :
  _Ex : Concevoir un jeu de gestion de donjon jouable en TUI/Web, mettant en œuvre POO, patterns et simulation._
- Public cible / utilisateur :
  _Ex : Joueur qui veut concevoir, tester et optimiser un donjon contre des vagues de héros._
- Résultat attendu en fin de semaine :
  _Ex : Application jouable + vidéo de démo scénarisée + tests unitaires de base._

## 2. Conventions

- **Statuts des tâches** :
  - `TODO`  : à faire
  - `DOING` : en cours
  - `DONE`  : terminé
- **Sprints** :
  - `DAY_1` à `DAY_5` (1 sprint par jour)
- Les détails de chaque sprint sont dans :
  `SPRINT_PLANNING_DAY_X.md` et `SPRINT_RETROSPECTIVE_DAY_X.md`.

## 3. Backlog global

> Liste des fonctionnalités et tâches importantes, avec une cible de sprint.
> À mettre à jour au fur et à mesure.

| ID  | User story / tâche                                          | Sprint cible | Priorité | Etat |
|-----|-------------------------------------------------------------|--------------|----------|------|
| US1 | En tant que joueur, je peux afficher une grille de donjon | DAY_1        | Haute    | TODO |
| US2 | En tant que joueur, je peux placer des murs/pièges        | DAY_1        | Haute    | TODO |
| US3 | Simuler le déplacement d'un héros dans le donjon          | DAY_2        | Haute    | TODO |
| US4 | Implémenter la stratégie IA "plus court chemin"           | DAY_2        | Haute    | TODO |
| US5 | Calculer et afficher le score de la vague                 | DAY_3        | Moyenne  | TODO |
| US6 | Sauvegarder/charger un donjon                             | DAY_3        | Moyenne  | TODO |
| US7 | Implémenter la deuxième stratégie IA                      | DAY_4        | Moyenne  | TODO |
| US8 | Ajouter un log des événements                             | DAY_4        | Basse    | TODO |
| US9 | Préparer la vidéo de démonstration                        | DAY_5        | Haute    | TODO |
| US10| Finaliser la documentation et nettoyage du code           | DAY_5        | Haute    | TODO |

## 4. Vue par sprint (résumé)

> À maintenir en cohérence avec les fichiers de planning/retrospective.

### Sprint DAY_1 – [date]

- **Objectif du jour** :
  _Ex : Avoir une grille affichée + dépôt Git structuré._
- **User stories ciblées** : `US1`, `US2`, ...
```

```
- **Critères de succès** :
  - Le projet se compile / s'exécute.
  - La grille apparaît (même minimale).
  - Un premier commit structuré existe sur le dépôt.

### Sprint DAY_2 – [date]

- ...

### Sprint DAY_3 – [date]

- ...

### Sprint DAY_4 – [date]

- ...

### Sprint DAY_5 – [date]

- ...

## 5. Dette technique et améliorations futures

> Tâches non prioritaires pendant la semaine, mais à garder en mémoire.

- [ ] Ajouter une troisième stratégie IA.
- [ ] Améliorer l'ergonomie de l'IHM (raccourcis, couleurs, etc.).
- [ ] Ajouter plus de tests (cas extrêmes, erreurs utilisateur).
```