CENTRALE NANTES

MEDEV

TP Noté

Myriam Servières \ Carito Guziolowski

Janvier 2026

V 1.0

Jeu du pendu

1 Présentation du jeu

[extrait de wikipedia]

Le pendu est un jeu de devinette dans lequel un joueur doit retrouver un mot secret en proposant successivement des lettres. Le jeu se joue traditionnellement à deux, avec un papier et un crayon. Il peut aussi se jouer en ligne contre un ordinateur.

À chaque proposition :

- si la lettre appartient au mot, toutes ses occurrences sont révélées ;
- sinon, le joueur commet une erreur.

Le joueur gagne s'il parvient à découvrir l'intégralité du mot avant d'atteindre le nombre maximal d'erreurs autorisées.
Le nombre maximal d'erreurs autorisées est généralement de 6 ou 7, selon les variantes du jeu. Chaque erreur entraîne le dessin progressif d'un pendu sur une potence. Lorsque le dessin est complet, le joueur a perdu la partie.

Exemple de dessin du pendu en début de partie :

```
 +---+
 |   |
     |
     |
     |
     |
=======
```

Exemple de dessin du pendu en fin de partie (le joueur a perdu) au bout de 7 erreurs :

```
 +---+
 |   |
 O   |
/|\  |
 |   |
/ \  |
=======
```

2 Règles du jeu à implémenter

2.1 Modes de jeu

Votre programme devra proposer au minimum les deux modes suivants :

- Mode 1 joueur :
  le mot secret est tiré aléatoirement depuis un dictionnaire externe.
- Mode 2 joueurs :
  un joueur choisit le mot secret, l'autre tente de le deviner.

2.2 Règles obligatoires

- Le nombre maximal d'erreurs est paramétrable et la visualisation proposée en conséquence.
- Une lettre déjà proposée ne doit pas être comptée comme une erreur supplémentaire.
- Les lettres proposées doivent être :
  - insensibles à la casse,
  - validées (caractère alphabétique uniquement).
- Le jeu doit gérer correctement les mots contenant :
  - des lettres répétées,
  - une longueur variable.

2.3 Déroulement d'une partie

À chaque tour :

1. l'état courant du mot est affiché (ex. `_ A _ _ E`) ;
2. la liste des lettres déjà proposées est affichée ;
3. le joueur propose une lettre ;
4. l'état du jeu est mis à jour.

La partie se termine lorsque :

- le mot est entièrement découvert (victoire) ;
- le nombre maximal d'erreurs est atteint (défaite).

3 Contraintes de conception

3.1 Séparation des responsabilités

Votre application devra **impérativement** séparer : - le moteur du jeu (règles, état, transitions), - la gestion des entrées/sorties (interface texte), - la gestion des données (dictionnaire, paramètres).

Toute logique métier intégrée directement dans l'interface texte sera pénalisée.

3.2 Modélisation de l'état du jeu

L'état d'une partie devra être clairement modélisé, incluant notamment :

- le mot secret ;
- l'ensemble des lettres proposées ;
- l'état courant du mot (lettres révélées / cachées) ;
- le nombre d'erreurs restantes ;
- l'état global de la partie (en cours, gagnée, perdue).

Les invariants de cet état devront être explicitement respectés.

3.3 Dictionnaire

En mode 1 joueur :

- les mots devront être chargés depuis un **fichier texte externe** à réaliser ;
- le chargement devra être robuste (test de fichier absent, vide, mots invalides) ;
- la sélection du mot devra être aléatoire.

4 Rendu

Le travail est à réaliser **en binôme**. Vous devrez utiliser **Git** pour le travail collaboratif (commits réguliers). Le dépôt devra être public ou accessible aux enseignantes (login github *mycmac* et *cguziolo*) et son **URL indiquée clairement** dans votre rapport.

Votre rendu devra comporter :

- les spécifications et documents de conception de votre proposition
- le code (documenté en javadoc) et les tests unitaires JUnit (sur GitHub) couvrant notamment :

  - les transitions d'état du jeu ;
  - les cas de lettres répétées ;
  - les lettres invalides ;
  - la détection de victoire et de défaite ;
  - la non-régression sur les règles principales. Les tests ne devront pas dépendre de l'interface utilisateur.

- le fichier de configuration de Ant ou de Maven permettant à minima l'automatisation du build et la création d'un jar mais idéalement aussi l'exécution des tests unitaires et la génération de la documentation.
- Tout au long du tp, vous devrez analyser votre code avec Sonar et montrer par des captures dans votre compte rendu l'évolution de votre projet (par exemple à 14h, 15h, 16h, 17h et en fin de projet).

Remarques : Des points supplémentaires seront accordés si les fichiers d'automatisation, comme ceux de configuration Ant ou Maven, incluent l'automatisation de la compilation des tests et la génération de rapports détaillés. Le fichier Ant doit être clairement commenté, tandis que les étapes nécessaires à la génération d'un fichier Maven doivent être expliquées en détail. Par ailleurs, les fichiers XML générés automatiquement par des IDE tels que NetBeans ne seront pas acceptés pour valider cette partie.

Le compte rendu sera à déposer sur hippocampus cours MEDEV_INFOSI (après s'être inscrit à un groupe de rendu) https://hippocampus.ec-nantes.fr/course/view.php?id=3001#section-7 pour 18h le 12 janvier 2026 au plus tard.