Introduction à la programmation objet

Pédagogie

- Apprentissage par projet (APP)
  - Réalisation d'un petit jeu vidéo
  - Découpage en mondes et niveaux
    - ★ Objectifs à coder
    - ★ Notions à maîtriser
- Déroulement des séances
  - Rendu des mondes à valider (devoir Teams)
  - Grille de validation (feuille Excel)
  - Utilisation avec parcimonie des outils d'IA générative
    - ★ Pour apprendre, pas pour faire à votre place
    - ★ Comprendre pour maîtriser, pas juste consommer
- Cours et TP (avec corrigé) sur Teams
  - À travailler en parallèle
  - QCM d'auto-évaluation et examens de l'an dernier
- Évaluation
  - Examen terminal écrit (sur feuille), sans document

Bénéfices

- Motivation intrinsèque
  - Donner du sens, de l'autonomie et un objectif clair
- Apprentissage actif
  - Plus efficace que l'enseignement purement magistral
- Pédagogie différenciée
  - Adapter l'enseignement aux forces/faiblesses des élèves
  - Feedback fréquent et individualisé

Jeu vidéo

Le joueur contrôle un personnage qu'il déplace dans un niveau où se trouvent des ennemis à combattre et des items à ramasser.

Carte

- Objectifs
  - Afficher `Hello World!` dans la console (IDE) / un terminal
- Notions
  - Classe
    - ★ Convention de nommage
    - ★ Méthode principale `main`
  - Flot de sortie standard `System.out`
  - Compilation d'une classe Java (`javac`)
    - ★ À partir d'un IDE ou d'un terminal
  - Exécution d'une classe Java (`java`)
    - ★ À partir d'un IDE ou d'un terminal

Carte

- Objectifs
  - Modéliser un joueur caractérisé par un nom (chaîne de caractères) et un score (entier) initialement nul
  - Créer une joueuse `Alice` et un joueur `Bob`
    - ★ Afficher les joueurs `Alice` et `Bob` dans la console
    - ★ Afficher le nom et le score des deux joueurs dans la console
    - ★ Ajouter des points au score d'`Alice`, retirer des points au score de `Bob`
    - ★ Afficher de nouveau le nom et le score des deux joueurs dans la console
- Notions
  - Objet/instance
    - ★ Convention de nommage
    - ★ Attributs d'instance (état)
    - ★ Constructeur
    - ★ Objet receveur `this`
  - Instanciation : opérateur `new`
  - Type primitif vs type référence
  - Éléments syntaxiques de base
    - ★ Déclaration et initialisation de variables
    - ★ Affectation et opérateurs arithmétiques

- Objectifs
  - Garantir que le nom d'un joueur n'est pas modifiable
  - Garantir que le score d'un joueur ne peut pas devenir négatif depuis une autre classe (cliente)
    - ★ Si le joueur perd plus de points qu'il n'en a, son score devient nul
  - Pas de duplication de code (principe DRY - *Don't Repeat Yourself*)
- Notions
  - Encapsulation
  - Modificateurs d'accès `public`/`private`
  - Attribut constant `final`
  - Méthodes d'instance (comportement)
    - ★ Accesseurs en lecture et en écriture
  - Éléments syntaxiques de base
    - ★ Conditionnelle `if`/`else` ou opérateur conditionnel ternaire
    - ★ Opérateurs de comparaison

- Objectifs
  - Commenter et documenter le code
  - Générer la documentation HTML
- Notions
  - Commentaires
  - Documentation HTML (`javadoc`)

- Objectif
  - Convertir un joueur en une chaîne de la forme *nom* : *score* `pts`
    - ★ Attention à l'accord de `pt(s)` en fonction du score
  - Afficher les joueurs `Alice` et `Bob` dans la console
    - ★ `Alice` a un score de 1 et `Bob` de 2
- Notions
  - Méthode `toString()`
  - Redéfinition de méthode
  - Opérateur de concaténation +

- Objectifs
  - Tester l'égalité entre joueurs : deux joueurs sont égaux s'ils ont le même nom (insensible à la casse)
    - ★ L'égalité d'un joueur avec un objet non joueur est toujours fausse
  - Tester que :
    - ★ La joueuse `Alice` n'est pas égale à la chaîne `"Alice"`
    - ★ La joueuse `Alice` n'est pas égale au joueur `Bob`
    - ★ Le joueur `Bob` est égal à un nouveau joueur `BOB`
    - ★ Le joueur `Bob` n'est pas `==` au joueur `BOB`
    - ★ Le joueur `Bob` est `==` à `b` où `b` est une nouvelle référence au joueur `Bob`
  - Déterminer le nombre de joueurs créés
  - « Supprimer » un joueur
- Notions
  - Méthode `equals(Object)`
  - Opérateur `instanceof`
  - Transtypage (*downcast*)
  - Opérateur `==`
  - Référence vs objet
  - Référence `null`
  - Ramasse-miettes

- Objectifs
  - Compter le nombre total de joueurs créés (peu importe où)
  - Avant de créer un seul joueur, afficher le nombre total de joueurs dans la console (*i.e.*, zéro)
  - Créer des joueurs sans donner de nom (en plus d'`Alice` et `Bob`)
    - ★ Leur nom sera `Joueur`*N* où *N* est le nombre total de joueurs créés
    - ★ Si le premier joueur est créé sans donner de nom, son nom sera `Joueur1`
  - Afficher chaque joueur créé dans la console
  - Afficher le nombre total de joueurs créés dans la console
- Notions
  - Attributs de classe `static`
  - Méthodes de classe `static`
  - Constructeur par défaut
  - Chaînage de constructeurs `this(...)`

Carte

- Objectifs
  - Modéliser un niveau caractérisé par une grille de caractères
    - ★ # pour les murs, ␣ (espace) pour le vide
  - Créer des niveaux de taille et de structure différentes
  - Afficher les niveaux dans la console
- Notions
  - Tableaux
  - Boucle `for`

- Objectifs
  - Placer un joueur (caractère 1) dans le niveau à sa création
    - ★ Si le joueur est placé sur un mur, hors des limites de la grille ou absent du niveau, signaler le problème au code appelant avec un message d'erreur approprié, sans l'obliger à le traiter
  - Afficher un niveau avec un joueur dedans dans la console
  - Afficher également le joueur du niveau avec sa position dans la grille
- Notions
  - Composition
  - Exceptions non vérifiées (vs vérifiées)
  - Levée d'une exception `throw`

- Objectifs
  - Déplacer le joueur dans le niveau, case par case, dans les 4 directions (haut, gauche, bas, droite)
    - ★ Le joueur ne peut pas traverser les murs
    - ★ Le joueur ne peut pas aller hors des limites de la grille
  - Afficher le niveau dans la console après chaque déplacement du joueur
    - ★ Même si le joueur est resté à la même position
- Notions
  - Type énuméré `enum`
  - Conditionnelle `switch/case`

- Objectifs
  - Demander en continu à l'utilisateur de déplacer le joueur dans le niveau à l'aide des touches du clavier (ZQSD)
    - ★ Attention à la gestion des entrées de l'utilisateur
  - Afficher le niveau dans la console après chaque déplacement du joueur
    - ★ Même si le joueur est resté à la même position
  - Ne pas mélanger le code du noyau applicatif avec le code d'interaction avec l'utilisateur
- Notions
  - Flot d'entrée standard `System.in`
  - Boucle `while`

- Objectifs
  - Créer un niveau à partir d'un fichier texte
    - ★ Le fichier texte est supposé correctement formaté
    - ★ Si le fichier n'existe pas, un message d'erreur est affiché dans la console
  - Jouer au niveau du fichier texte
- Notions
  - API `java.nio.file` pour la lecture(/écriture) de fichier
  - Exceptions vérifiées (vs non vérifiées)
  - Capture d'une exception `try/catch`
  - Flot d'erreur standard `System.err`

- Objectifs
  - Créer un exécutable du jeu
  - Lancer le jeu en fournissant le fichier texte contenant le niveau en argument du programme
    - ★ Le nom du fichier texte est arbitraire
    - ★ Si aucun fichier n'est fourni, un message d'utilisation est affiché dans la console
- Notions
  - Paramètre de la méthode principale `main`
  - Exportation d'une archive jar exécutable
  - Exécution d'une archive jar exécutable

Carte

- Objectifs
  - Ajouter des pièces (caractère .) dans le niveau que le joueur peut ramasser pour gagner des points
    - ★ Chaque pièce ramassée augmente le score du joueur de 10 points
    - ★ Les pièces ramassées disparaissent du niveau
  - Afficher `NIVEAU TERMINÉ` dans la console lorsque le joueur a ramassé toutes les pièces du niveau

- Objectifs
  - Ajouter des pièges (caractère $*$) dans le niveau qui font perdre 2 vies au joueur quand il passe dessus
    - ★ Les pièges sont détruits après que le joueur passe dessus
  - Faire commencer le joueur avec 5 vies
  - Ramener le joueur à sa position de départ dès qu'il perd une vie
  - Afficher `GAME OVER` dans la console lorsque le joueur n'a plus de vie
  - Après un *game over*, demander à l'utilisateur s'il souhaite recommencer le jeu de zéro ou quitter

- Objectifs
  - Passer plusieurs fichiers texte contenant chaque niveau du jeu en argument du programme
  - Au début du jeu, demander à l'utilisateur d'entrer un nom de joueur
  - Dès qu'un niveau est terminé, passer au niveau (fichier) suivant
    - ★ Le joueur conserve ses données (score, vies...) d'un niveau à l'autre
  - Lorsque tous les niveaux sont terminés, remercier l'utilisateur d'avoir joué et quitter le jeu

- Objectifs
  - Modéliser une cellule d'une grille caractérisée par sa position (ligne/colonne) dans la grille, la présence ou non d'une pièce, et un type (vide, mur, piège...)
  - Remplacer la grille de caractères d'un niveau par une grille de cellules

- Objectifs
  - Ajouter un nouveau type de cellule, porte verrouillée (caractère `D`), qui, comme les murs, ne peut pas être traversé par le joueur
    - ★ Sans avoir besoin de mettre à jour le code de collision

- Objectifs
  - Considérer la grille de cellules comme un tore
    - ★ Les cellules sur les bords de la grille ont également 4 voisines (cellules des bords opposés)

Carte

- Objectifs
  - Modéliser un ennemi qui se déplace aléatoirement dans le niveau, case par case, dans les 4 directions
    - ★ Un ennemi a un nom unique et une vie
    - ★ Un ennemi ne peut pas traverser les murs ni les portes verrouillées
    - ★ Un ennemi ne peut pas passer sur les pièges
    - ★ Un ennemi ne ramasse pas les pièces, mais peut aller sur des cases qui en contiennent
  - Placer un ennemi (caractère `R`) dans le niveau
  - Déplacer l'ennemi après chaque déplacement du joueur
    - ★ Même si le joueur est resté à la même position
- Notions
  - Héritage
  - Modificateur d'accès `protected`
  - Surcharge de méthode (vs redéfinition de méthode)
  - Classe abstraite

- Objectifs
  - Faire perdre 1 vie au joueur quand il entre en collision avec un ennemi
    - ★ Le joueur et un ennemi se trouvent sur une même case
  - Afficher les ennemis entrés en collision avec le joueur dans la console
  - Ramener les ennemis à leur position de départ chaque fois que le joueur perd des vies
    - ★ Y compris si c'est en passant sur un piège
  - Placer plusieurs ennemis dans le niveau et déplacer-les après chaque déplacement du joueur
    - ★ Les ennemis peuvent se retrouver sur une même case
- Notions
  - Collections
    - ★ Liste (séquence)
    - ★ Types génériques
    - ★ Itérateur

- Objectifs
  - Stocker dans le niveau l'ensemble des cellules occupées par les ennemis
- Notions
  - Collections
    - ★ Ensemble (sans doublon)
  - Méthode `hashcode()`

- Objectifs
  - Modéliser un ennemi *fantôme* qui se déplace dans une même direction sans être bloqué par quoi que ce soit et ne change de direction que lorsque qu'il se trouve sur la même ligne/colonne que le joueur pour aller vers lui
  - Placer un fantôme (caractère `G`) dans le niveau
- Notions
  - Polymorphisme
  - Type statique vs type dynamique
  - Transtypage (*upcast*)

- Objectifs
  - Modéliser un ennemi *chasseur* qui poursuit le joueur
    - Il a 3 vies
    - Il fait perdre 2 vies au joueur quand il entre en collision avec lui
  - Placer un chasseur (caractère `C`) dans le niveau
- Notions
  - Algorithmes : BFS, Dijkstra, A*, *etc.*

- Objectifs
  - Structurer le projet en regroupant les classes liées entre elles
  - Ne plus avoir aucune classe à la racine du projet
- Notions
  - Paquetages
  - Modificateur d'accès par défaut *package-private*

Carte

- Objectifs
  - Ajouter au joueur un inventaire limité pour stocker des items et des compétences
    - ★ Capacité max : 5 éléments
    - ★ Les items sont ramassés par le joueur
    - ★ Certains items ramassés ne sont pas stockés dans l'inventaire, mais immédiatement consommés par le joueur (*e.g.*, les pièces)
    - ★ Une cellule ne peut comporter qu'un seul item à ramasser à la fois
    - ★ Les compétences, quant à elles, sont gagnées par le joueur
    - ★ Si l'inventaire est plein, l'item à stocker n'est pas ramassé ou la compétence obtenue n'est pas gagnée
    - ★ Les items et les compétences de l'inventaire pourront être utilisés par le joueur avant chaque déplacement
    - ★ Après utilisation, seuls les items disparaissent de l'inventaire
  - Rendre l'inventaire *type-safe* (*i.e.*, sûr du point de vue du typage)
- Notions
  - Interface
  - Polymorphisme

- Objectifs
  - Modéliser un item *arme* qui permet automatiquement de faire perdre une vie à un ennemi entré en collision avec le joueur
    - ★ Le joueur, quant à lui, ne perd aucune vie
  - Ramener un ennemi à sa position de départ dès qu'il perd une vie
  - Supprimer du niveau un ennemi qui n'a plus de vie
  - Placer plusieurs armes (caractère `W`) dans le niveau

- Objectifs
  - Modéliser un item *sablier* qui permet de stopper les ennemis durant dix déplacements du joueur à partir du moment où il l'utilise
  - Permettre au joueur d'utiliser l'item avant de se déplacer
  - Placer un sablier (caractère `H`) dans le niveau

- Objectifs
  - Modéliser une compétence *crochetage* qui permet au joueur de traverser automatiquement les portes verrouillées
    - ★ Les portes restent verrouillées pour les ennemis
  - Faire gagner cette compétence au joueur dès que son score dépasse 100

- Objectifs
  - Modéliser une compétence *téléportation* qui permet au joueur de se déplacer aléatoirement sur une cellule vide chaque fois qu'il l'utilise
  - Faire gagner cette compétence au joueur dès qu'il a vaincu 3 ennemis
  - Permettre au joueur d'utiliser cette compétence avant de se déplacer

- Objectifs
  - Trier les éléments de l'inventaire selon leur type (d'abord les items, puis les compétences) d'une part, et leur nom d'autre part
  - Terminer certains niveaux en tuant tous les ennemis ou en ramassant un item particulier
    - ★ Au début de chaque niveau, la condition de réussite est affichée
- Notions
  - `Comparable<T>`, `Comparator<T>`

Carte

- Objectifs
  - Afficher `Hello World!` dans une fenêtre graphique
- Notions
  - JavaFX
    - ★ `Application`, `Stage`, `Scene`, `Node`

- Objectifs
  - Afficher le niveau avec le joueur, les ennemis et les items dans une fenêtre graphique
  - Appliquer le modèle d'architecture logicielle PAC
- Notions
  - `Canvas`
  - Modèle d'architecture logicielle PAC

- Objectifs
  - Déplacer le joueur dans le niveau à l'aide des flèches du clavier
  - Déplacer les ennemis dans le niveau
- Notions
  - Gestionnaires d'événement

- Objectifs
  - Afficher le nom du joueur avec son score et ses vies
  - Afficher l'inventaire du joueur
- Notions
  - Panneaux
  - CSS

- Objectifs
  - Mettre à jour les données (*i.e.*, score, vie, inventaire) du joueur
  - Utiliser un item ou une compétence en cliquant dessus dans l'inventaire
  - Faire disparaître du niveau les items ramassés et les ennemis vaincus
- Notions
  - Liaison de données `Property`

- Objectifs
  - Ajouter un écran de démarrage avec un menu et de fin de jeu (crédits)
  - Ajouter des boites de dialogue pour communiquer avec l'utilisateur
    - ★ Entrer le nom du joueur
    - ★ Recommencer le jeu de zéro ou quitter
  - Déplacer le joueur en continu selon sa dernière direction
    - ★ Les flèches du clavier permettent dorénavant de définir la prochaine direction souhaitée du joueur
- Notions
  - Boîtes de dialogue `Alert`
  - Minuteur `AnimationTimer`

Carte

- Objectifs
  - Comprendre le message d'erreur affiché dans la console
  - Déboguer en ajoutant des points d'arrêt dans le code afin de suivre l'évolution des variables (*i.e.*, l'état du programme)
  - Corriger la faute dans le code
- Notions
  - Débogueur (`jdb`)