Master 1 GIL - Normes et spécifications de codage
Agora V3-1

Groupe 1 Agora V3-1 : Partie bibliothèque de jeux

20 janvier 2024

Version	2
Date	20 janvier 2024
Rédigé par	MAZUY Axelle

Mises à jour du document

Version	Date	Modification réalisée
1	17 janvier 2024	Création du document
2	20 janvier 2024	Mise à jour nommage des dossiers

Table des matières

1 Normes générales concernant l'organisation du code

1.1 Taille des fichiers

Les fichiers de code devant être partagés, écrits et relus par un certain nombre d'intervenants, on peut s'accorder sur des fichiers n'excédant pas 1000 lignes de code. Au delà, il devient difficile pour quiconque de se retrouver dans le fichier, et cela relèverait également d'un mauvais découpage de code. Il est donc important de modulariser les fichiers.

1.2 Taille des méthodes

De même qu'il est nécessaire de ne pas dépasser une certaine longueur de fichier pour faciliter la relecture, il est essentiel de limiter la taille des méthodes.

Au delà de 100 lignes de code, une méthode est considérée bien trop longue et là encore, témoigne d'un mauvais découpage ou d'une mauvaise programmation.

Il est essentiel de créer des fonctions et méthodes par fonctionnalité.

En cas de fonctions trop longues, il faudrait envisager si c'est possible de recourir à des méthodes/classes outils, ou recourir à un service.

1.3 Spécifications des classes et méthodes

Il est nécessaire de maintenir des spécifications dans le code en en-tête de classes et de méthodes. Cela permet de spécifier le comportement attendu et ainsi d'expliquer à de futurs intervenants ou à de futurs relecteurs l'intérêt du code fourni.

Toutefois, le code doit pouvoir se suffire à lui-même en partie : les noms de variables doivent être explicites, le code clair et compréhensible, les commentaires ne doivent pas alourdir le code mais aider à la compréhension - et doivent être mis à jour si besoin.

L'en-tête de spécification doit contenir les informations clés nécessaires à sa compréhension, sans pour autant comprendre la manière dont le code est réalisé. On retrouvera ainsi pour les méthodes :

— Une brève description de ce que réalise la méthode
— La liste des paramètres ainsi que leur signification si ce n'est pas explicite
— La valeur de retour
— Les exceptions si la méthode en renvoie

Pour les classes :

— Un descriptif de ce que gère la classe, son utilité
— Les invariants de la classe
— L'explication du calcul des attributs dérivés

2 Normes générales de nommage - Nomenclature

De même qu'il est important de spécifier son code, il est important de correctement nommer les différentes entités que l'on peut rencontrer pour comprendre rapidement de quoi on parle.

Il est également important de noter que les noms doivent être en anglais, de manière uniforme sur tous les fichiers et packages.

2.1 Package

Le nom des packages sont en minuscules pour les packages englobants, mais la camel case est acceptée. On mettra toutefois une majuscule au début des noms de sous-dossiers de src et templates (Controller, Entity, ...).

On pourra donc retrouver des noms comme "resources" ou "gameResources" pour les noms de dossiers en dehors de templates et src. En revanche, le nom doit commencer par une minuscule.
De même, on n'utilisera que les caractères [a-zA-Z][0-9] et le point.

2.2 Classe

Tout comme les packages, les classes n'utilisent que les caractères [a-zA-Z][0-9].
Les classes ont un nom commençant par une majuscule et acceptent également la camel case.
De même, on suffixera une classe par son type : Controller si c'est un contrôleur, Service si la fonction contient du code métier, Repository si on effectue des requêtes en base de données, Test s'il s'agit d'une classe de tests, etc. Le nom d'une classe est donc composée d'un nom simple et descriptif, ainsi que du type de classe. En revanche, le nom d'une classe ne sera pas une action ou un verbe, on privilégiera les noms.

2.3 Méthode

Les méthodes ont une norme similaire aux classes concernant les caractères utilisés, toutefois les noms de méthodes commencent par une minuscule.
On adaptera le nom de la méthode à son type :

- Une requête commencera par get
- Une commande aura pour nom un verbe décrivant l'action. Si cette méthode ajoute un élément à une structure de données, on privilégiera le mot "add". Si cette méthode remplace la valeur d'un objet, on privilégiera le mot "set".

2.4 Variable

Les variables respectent une syntaxe similaire aux méthodes : elles sont donc écrites en minuscule avec des camel cases, en respectant l'usage [a-zA-Z][0-9].

On évitera les variables à une seule lettre si celles-ci ne sont pas pour un usage précis et local (comme les tours de boucle). On privilégiera un nom descriptif.

2.5 Constante

Les constantes sont des attributs particuliers : publics, statiques, finaux, ils sont écrits en majuscule et leur valeur est déterminée dès la déclaration.
Les différents mots composant le nom d'une contantes sont séparés par un underscore. Là encore, on donnera un nom descriptif et bref.

3 Normes générales de nommage des commits

L'espace de code partagé Git est également un lieu sujet à la compréhension des différents intervenants. Il est donc essentiel d'adopter une nomenclature de commits particulière.

On adoptera donc pour chaque commit la syntaxe suivante :

```
[CODE JIRA] [Type de commit (feature, fix, etc)] [Message du commit] \n
\n
[Description complète]

Où \n est un retour à la ligne.
```

4 Conventions relatives à la lisibilité

4.1 Indentation du code

L'ensemble du projet doit être indenté de manière similaire. De plus, les environnements de travail devraient être configurés pour remplacer les tabulations par des caractères blancs afin d'assurer la compatibilité et lisibilité sur tous les environnements de travail.
On optera donc pour une tabulation valant 4 espaces.

4.2 Taille des lignes

Des lignes trop longues sont lourdes à relire et comprendre, de plus il est assez pénible de coder et lire en utilisant une barre horizontale de défilement. C'est pourquoi on limitera à 180 caractères maximum une ligne, toutefois il serait préférable d'avoir des lignes de code d'en moyenne 80 caractères pour plus de lisibilité peu importe la taille de la fenêtre.

4.3 Retour à la ligne

Il est à noter qu'il faudra respecter les retours à la limite de manière uniforme sur l'ensemble du projet.
Ainsi, on optera pour une manière de coder similaire à du code généré par Symfony :

— Les accolades seront renvoyés en début de ligne suivante pour les méthodes
— Les accolades seront mises en fin de ligne pour les structures conditionnelles et itératives
— Pas d'espace avant et après les flèches (->)
— Un espace avant et après le symbole d'égalité (=)
— Pas d'espace avant le symbole de fin de ligne de code (;)
— Pas d'espace après la possible nullité d'un attribut/paramètre (?)
— Pas d'espace avant la déclaration du type de retour (:)
— Les symboles fermants ')', ']', '}' sont renvoyés en début de ligne suivante
— Pour les balises html, on ne mettra pas d'espace avant et après les éléments contenus dans les balises.

4.4 Encodage des fichiers

Il est également important que tous les fichiers suivent un encodage identique afin de ne pas avoir de problèmes de compatibilité de caractères.