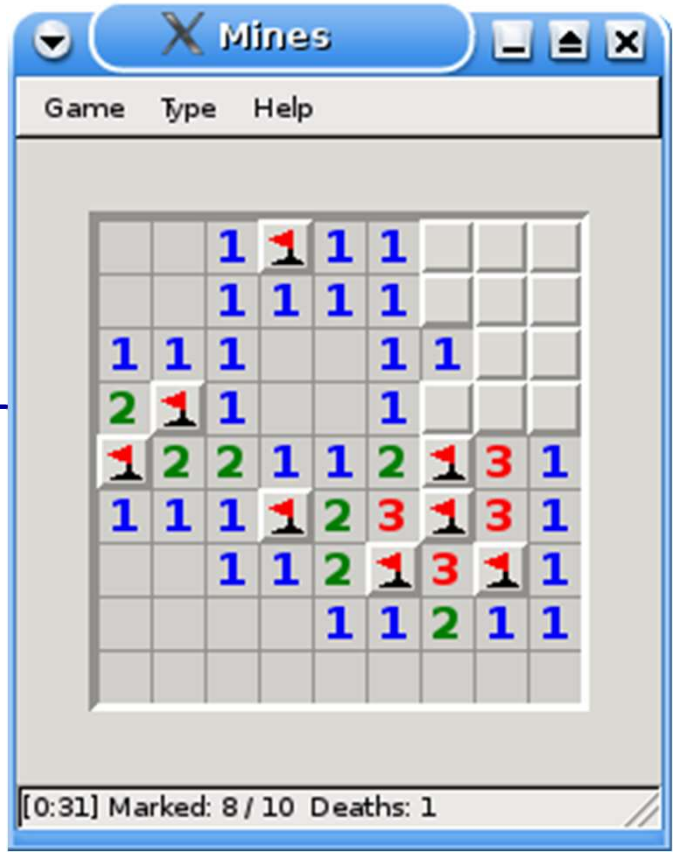
Module R6.05
Introduction à Dart & Flutter

Modalités du Cours

- 6 séances de cours/TP de 3h
- Un seul projet (démineur) enrichi à chaque séance
- Plan :
  - TP1 : Découverte de DART - codage du modèle et interface CLI
  - TP2 : Introduction à Flutter - codage d'un StatefulWidget
  - TP3 : Navigation (basic) et Gestion de l'état (basic)
  - TP4 : Listes, Formulaires
  - TP5 : Navigation (Navigator), Gestion de l'état (RiverPod)
  - TP6 : Persistance avec FireBase

- Evaluation : rendu du projet

TP 1
Coder le Modèle du Démineur



Introduction

1. **Origines :** langage de programmation open source développé par Google. Créé en 2011 avec pour objectif d'améliorer le développement web (projet initial de VM Dart intégré à Chrome, abandonné depuis)
2. **Polyvalence :** Dart peut être utilisé pour le développement d'applications web (front mais back aussi) et mobiles
3. **Syntaxe Similaire à d'autres langages :** syntaxe intuitive qui ressemble à celle de nombreux autres langages => transition facile
4. **Multi plates-formes :** Dart peut être compilé de façon native (Linux, macOS, Windows, Android, iOS) ou exécuté dans un navigateur Web en étant compilé en JS. VM pour exécuter du DART en ligne de commande.
5. **Orienté objet :** modèle objet complet, garbage collector, …
6. **Fort Typage :** langage fortement typé => code plus fiable
7. **Asynchrone par Conception :** prise en charge native de la programmation asynchrone
8. **CLI, Gestionnaire de paquets** : le SDK fournit une interface en ligne de commande et un gestionnaire de paquets (pub)
9. **Communauté Active :** nombreuses ressources en ligne
10. **Documentation** : https://dart.dev/language

Installation TP1

- Utilisation de VS Code et Installation de Dart+Flutter *via* VS code fortement recommandée
- Installer VS Code : https://code.visualstudio.com/download
- Installer Dart + Flutter (Type App : Mobile) : https://docs.flutter.dev/get-started/install
- Décompresser l'archive **`tp01.zip`** fournie sur Chamilo
  => un répertoire **tp01** est créé
- Ouvrir le dossier **tp01** dans VS Code
- Dans le terminal de VS Code, taper la commande pour installer les dépendances du projet : **`dart pub install`** **`get`**
- Le projet est prêt à être exécuté :
  - Interface VS Code : Ctrl+Shift+D (configuration tp01)
  - Dans le terminal : **`dart run`**

Principe du Jeu du Démineur

- Le champ de mines du Démineur est représenté par une **grille** à deux dimensions (pavage rectangulaire).
- Chaque **case** de la grille peut soit cacher une mine (être **minée**), soit être vide.
- Au départ, le contenu de toutes les cases est « couvert » (caché)
- Le but du jeu est de **découvrir** toutes les cases **non minées**
- Lorsque le joueur **découvre** une case vide **qui comporte au moins une mine autour** (dans son 8-voisinage), un chiffre apparaît dans cette case, indiquant le « nombre de mines autour »
- Le joueur peut aussi **marquer** les cases couvertes dont il croit qu'elles sont minées (et supprimer une(des) marque(s) plus tard s'il a changé d'avis).
- Lorsqu'un joueur **découvre** une case vide **qui ne comporte aucune mine autour** :
  - Toutes les cases **voisines** qui ne sont **pas marquées**, **pas minées** et **pas encore découvertes**, sont automatiquement **découvertes**
  - Ce processus automatique est **répété** récursivement mais uniquement sur les voisines qui remplissent les 3 critères précédents **et qui en plus n'ont pas de mines autour**
- En croisant les informations affichées au fur et à mesure, le joueur peut progresser dans le déminage du terrain.
- Finalement, la partie s'arrête et le joueur a :
  - **Perdu** s'il a découvert une case minée
  - **Gagné** s'il a découvert **toutes** les cases non minées

Cahier des charges

- On doit pouvoir choisir la taille de la grille
- On doit pouvoir choisir le nombre de mines placées aléatoirement dans la grille
- On doit pouvoir jouer avec une interface en mode texte pour :
  - Saisir le coup d'un joueur : coordonnées de la case dans la grille et action réalisée (**D**écouvrir ou **M**arquer)
  - Afficher le contenu de la grille :
    - Une case masquée : *
    - Une case marquée : **M**
    - Une case découverte sans mine autour : (*espace*)
    - Une case découverte avec des mines autour : **n** (où **n** est le nombre de mines autour de la case)

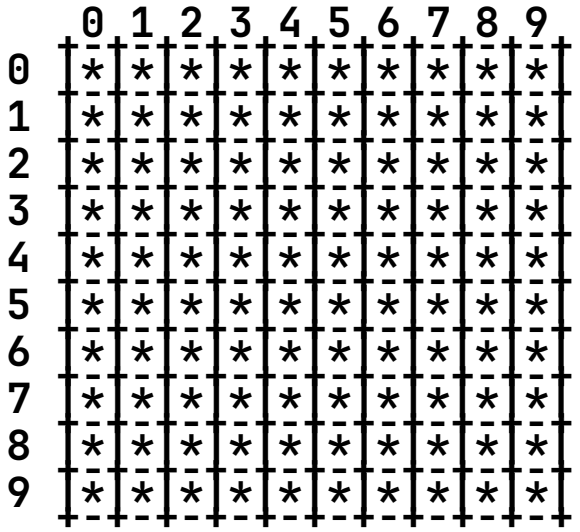
Pour tester et bien comprendre le jeu : https://minesweeper.online/

Les étapes

- Avant de développer l'Interface Utilisateur « cross platform » de cette application avec Flutter, nous allons d'abord :
  - Développer le **modèle** en Dart (sera réutilisé avec l'application Flutter) :
    => ***lib/modele***
  - Tester ce modèle avec une **Interface Utilisateur** en mode texte (*fournie*)
    => ***lib/interface***
  - Le programme principal (main, fourni) code le déroulement du jeu :
    => **bin/tp01.dart**

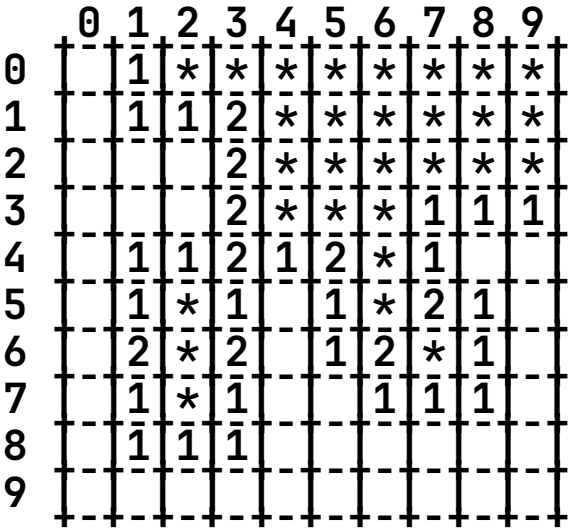
1

```
Taille de la grille :10
Nombre de mines     :10
```



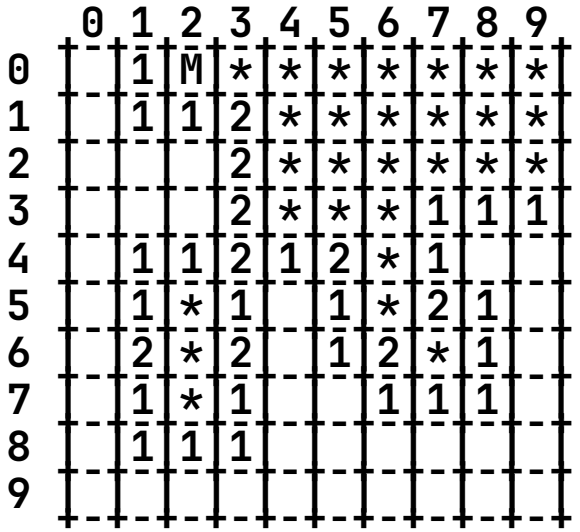
2

```
Quel coup jouez-vous : 0 0 D
Il faut continuer a deminer...
```



3

```
Quel coup jouez-vous : 0 2 M
Il faut continuer a deminer...
```



4

```
Quel coup jouez-vous :5 2 D
BOUM ! Vous avez perdu !
```

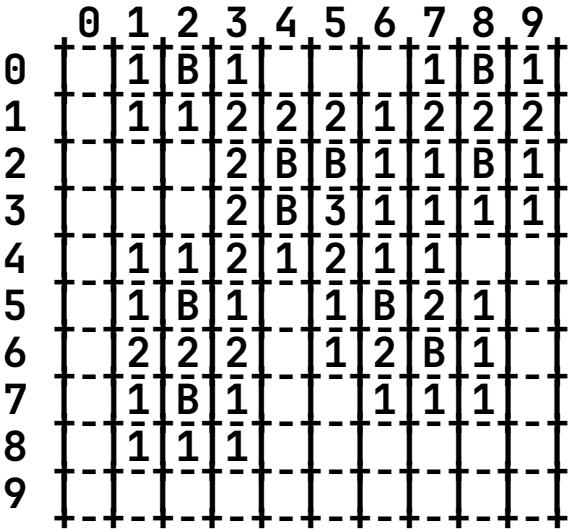
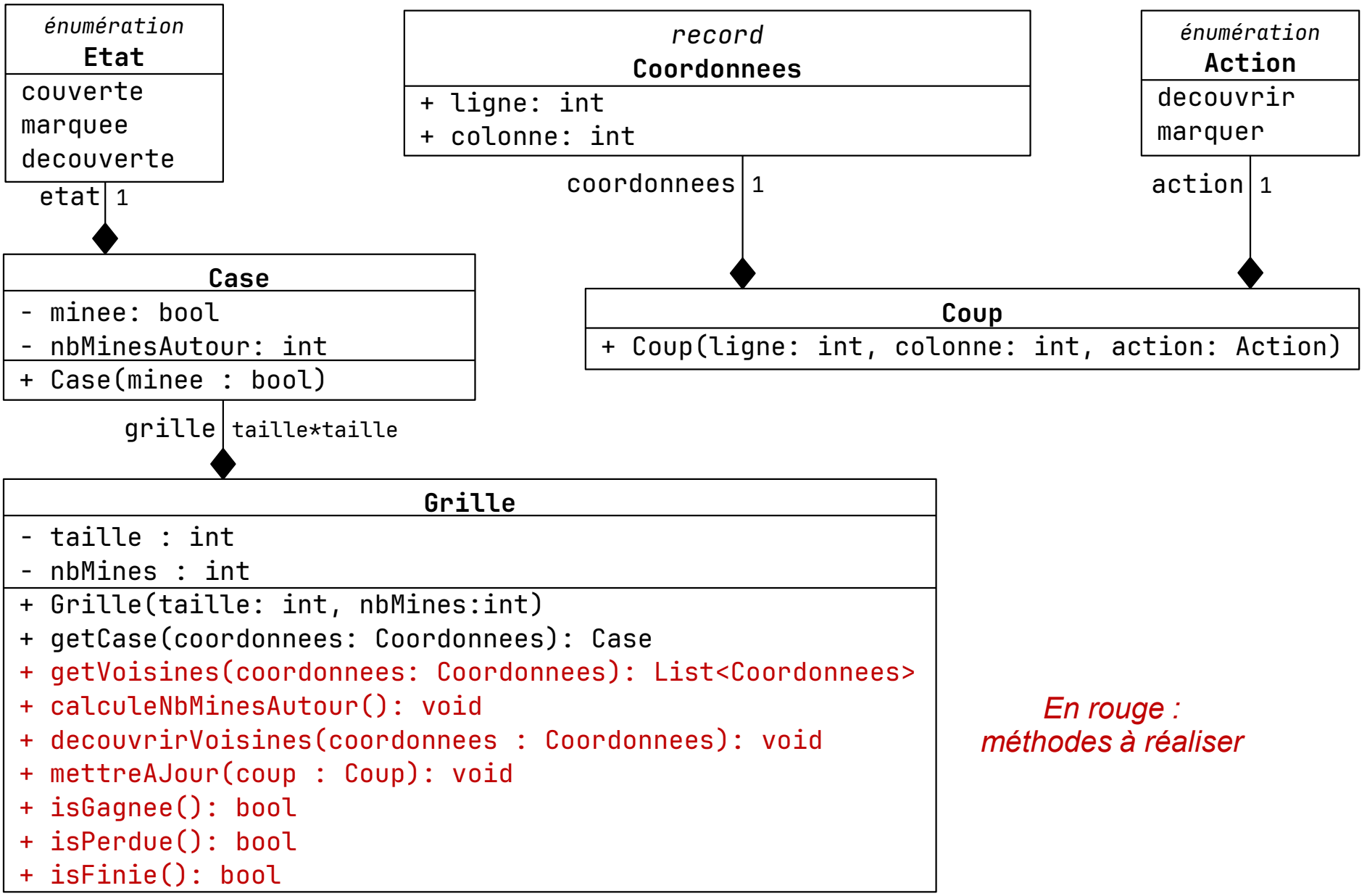


Diagramme de Classe (lib/modele.dart)



En rouge :
méthodes à réaliser

Interface Utilisateur « texte » (console_ui.dart)

```dart
// Tailles MIN et MAX de la grille
const tailleMin = 2;
const tailleMax = 10;
// Caractère affiché selon l'état de la case
const caseVide = ' ';
const caseMinee = 'B';
const caseMarquee = 'M';
const caseCouverte = '*';
// Délai maximum pour répondre (en secondes)
const delaiMax = 10;

/// Record pour représenter la taille d'une grille et son nombre de mines
typedef ParametresGrille = ({int taille, int nbMines});

/// Saisie asynchrone sur [stdin] de la taille et du nombre de mines
/// - Si pas de saisie au bout de [delaiMax], on utilise des valeurs par défaut
/// - Le résultat est renvoyé sous forme de record
Future<ParametresGrille> saisirParametres() async {…

/// - Saisie asynchrone sur [stdin] du coup d'un joueur :
/// - deux entiers ligne,colonne compris entre 0 et taille-1
/// - un caractère pour l'action : d (découvrir) ou m (marquer)
Future<Coup> saisirCoup(int taille) async {…

/// - Affiche sur [stdout] la [grille]
/// - Montrant la solution si [montrerSolution]
void afficher(Grille grille, {bool montrerSolution = false}) {…

/// Affiche le résultat selon l'état de la [grille] après un coup joué
void afficherResultat(Grille grille) {…
```


Programme principal

```dart
void main(List<String> arguments) async {
  // Saisie des paramètres : taille et nombre de mines (délai pour choisir)
  ParametresGrille params = await saisirParametres();
  // Initialisation de la grille
  Grille laGrille = Grille(taille: params.taille, nbMines: params.nbMines);
  // Pour tester/déboguer : on affiche tout de suite la solution
  afficher(laGrille, montrerSolution: true); // à commenter pour jouer vraiment
  // Déroulement d'une partie
  do {
    afficher(laGrille);
    Coup coup = await saisirCoup(laGrille.taille);
    laGrille.mettreAJour(coup);
    afficherResultat(laGrille);
  } while (!laGrille.isFinie());
  // A la fin on affiche la solution
  afficher(laGrille, montrerSolution: true);
  exit(0);
}
```


Travail à réaliser

- Etudiez le code fourni
- Lisez la documentation du langage quand c'est nécessaire: https://dart.dev/guides
- Complétez les méthodes de la classe Grille (`lib/grille.dart`)
- Modifiez l'interface pour que, si le joueur n'a pas commencé à saisir un coup au bout de 10 secondes, une aide lui soit proposée en découvrant « automatiquement » une case (couvert et non minée évidemment).
- Compléter le modèle et l'interface pour :
  - Gérer la notion de joueur (un pseudo, un best score)
  - Calculer le score d'un joueur
  - Gérer une liste de joueur (palmares)