

M2 CCI

UFR IM2AG

Projet de Programmation

(du 24/11/25 au 28/11/25)

Encadrement :

Catherine Vigouroux

Cristian Ene

Laurent Mounier

et avec le support de Jean-Marie Favre pour la gestion des dépôts git ... !

1

1

Organisation générale

Planning global proposé:

- lundi 24 :

prise en main du sujet, des documents fournis, des dépôts git ...
planning prévisionnel, partage du travail, organisation
premières lignes de code ...

- du mardi 25 au vendredi 28 :

(codage, tests, intégration, révision du planning et du partage des tâches)

- lundi 1 après-midi :

évaluation sous forme de **démos commentées** (30 minutes / groupes)

Support / Ressources :

- [permanences enseignant](#) (~ 1h30 / demi-journées en salles de TP)
- [Moodle](#) ...

3

3

Objectifs (pédagogiques)

Programmer :

- à plusieurs
- un logiciel de taille « moyenne »
- à plein temps, sur une semaine

Les enjeux :

- taille et complexité du code
- partage du travail (interfaces, gestion des versions)
- utilisation de code fourni
- autonomie (choix des objectifs et des solutions, gestion du temps)

→ **Progression / entraînement :**

- en algorithmique
- en programmation (mise au point, tests, ...)
- en travail en équipe (projet d'intégration en fin de semestre)

2

2

Thème du projet :

programmer des « réussites »

Cahier des charges = règles du jeu des « réussites »

Couvre **differents contextes** de programmation :

- **structures de données** (ensembles de cartes, piles/tas de carte, etc.)
interfaces et implémentation
- **algorithmique** propre à chaque « réussite »
- **analyse comparative** des « réussites »
- utilisation d'une **librairie graphique**

→ sujet à **géométrie variable**, ne pas chercher à tout faire !!!

4

4

Les ressources fournies

- du **code** (dépôt *m2cci-2526-pr-root*)
 - des types de base pour représenter des cartes
 - une 1ère implémentation de la structure « tas de cartes »
 - une interface d'accès à une librairie graphique
 - un exemple complet de code d'une réussite (Relais des 7)
 - un *Makefile*
- un **document papier**
 - une description du code fourni
 - une description du travail à faire
 - les règles des différentes réussites ...

5

Démo time !

- des ressources fournies
- du code fourni
- du « Relais des 7 »

6

Quelques conseils ...

Travailler à l'UFR (salles machine réservées)
 - environnement de travail adapté
 - permanence des enseignants, présence d'autres étudiants !

Réfléchir avant de programmer ... !

Tester dès que possible, utiliser des **outils d'aide à la mise au point** (VSCode, gdb, ...)

Discuter beaucoup (au sein des groupes ... et entre groupes !)

N'hésiter pas à **adapter/personnaliser** le travail demandé (variantes, autres réussites ?)

Bien « **versionner** » tout ce qui pourra être « **montrable** » le jour de la démo !

7

Test et mise au point ?

Plusieurs difficultés :

1. Pouvoir tester et mettre au point des parties spécifiques du code sans attendre que l'ensemble du code soit disponible (et intégré)
 → « tests unitaires »
2. Contrôler la génération aléatoire des entrées (ex : tirage des cartes)
 → pouvoir « rejouer » des entrées données (utile aussi pour une démo !)
3. Pouvoir placer l'application dans un « état » donné
 ex : une configuration de jeu qui semble produire une erreur

8

7

8

Test unitaire

Objectif : tester une **fonctionnalité** précise **en cours de développement**

exemples :

- affichage d'un tas, ou de l'ensemble du plateau de jeu
- analyse statistique d'un ensemble de parties
- modification de l'état du plateau de jeu (déplacement de cartes)

Solution :

- écrire **un programme principal** minimal dédié à ce test
 - ouvrir une fenêtre graphique
 - produire un ensemble de valeurs dans un tableau
 - initialiser le plateau de jeu dans un état donné
- compiler ce programme principal **avec uniquement le code à tester** (produit un **exécutable** dédié au test)

9

Contrôler la génération aléatoire (fichier lib/Alea.c)

```
void InitAlea()
{
    int g;
    time_t t;
    g = (unsigned int) time(&t); // germe choisi « aléatoirement »
    srand(g);
}
```

→ ajouter une fonction d'initialisation avec **contrôle du germe**

```
void InitAleaControl (unsigned int g) // germe fourni par l'utilisateur
{
    srand(g);
}
```

Remarque : possible également d'**afficher** les germes utilisés par InitAlea (rejet)

10

10

Bon projet !

Et n'oubliez pas
d'apporter vos jeux de
cartes pour vous
imprégner des
réussites proposées ...



11

11