# Projet Informatique – Sections Electricité et Microtechnique

Printemps 2025 : *Linked-Crossing* © R. Boulic & collaborators

## Rendu3 (25 mai 23h59)

Table des matières :

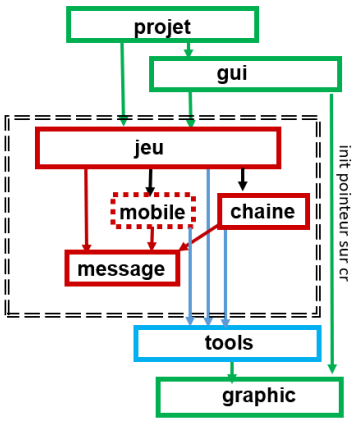


***Objectif de ce document :*** *Ce document utilise l'approche introduite avec la série théorique sur les méthodes de développement de projet qu'il est important d'avoir faite avant d'aller plus loin.*

*En plus de préciser ce qui doit être fait, ce document identifie des tests élémentaires à effectuer pour vérifier individuellement chaque fonctionnalité demandée ; c'est à vous de mettre au point les fichiers de test pour ces vérifications. Un conseil : concevez les tests les plus simples possibles car en cas de bug cela simplifie l'analyse du problème. Vous devz continuer à respecter l'architecture minimale du projet (donnée Fig 11b).*

## *1. Buts du rendu3 : finalisation du jeu et documentation dans le rapport final*

Votre approche peut évoluer entre ce que vous avez décrit pour le rendu2 en matière de structuration des données et ce que vous mettez en œuvre finalement du moment que vous respectez les responsabilités des différents modules (Fig 11b) visible ci-contre =>




**Lancement et comportement attendu:**
Le programme sera lancé soit en indiquant un nom de fichier à ouvrir (rendu2) :

```
./projet test1.txt
```

Soit sans aucun argument sur la ligne de commande :

```
./projet
```

Dans ce second cas, l'interface est créée et le programme attend qu'on lui demande d'ouvrir un fichier avec le bouton Open.

Une fois le fichier ouvert, on dit pouvoir construire et guider la chaine pour l'attirer vers son but, traiter le cas de collision avec un faiseur, et gérer les états du jeu (ONGOING, WON,LOST).

Les fonctionnalités demandées sont décrites dans la donnée générale. La vérification supplémentaire suivante doit aussi être effectuée sur la configuration de la chaîne:

- **Toutes les articulations de la chaîne doivent toujours être à l'intérieur de l'arène** ; cela est garanti en mode construction mais en mode guidage il faut ajouter la vérification suivante :
  - Dès qu'une articulation (après la racine) n'est plus à l'intérieur de l'arène, alors la mise à jour produite par l'algorithme de guidage est annulée.

- Cette approche simple permet de ne pas avoir à modifier le code qui calcule le but intermédiaire.

La majorité des scénarios d'évaluation du rendu3 sera constitué des cas très simples discutés dans la section 2. Seuls les deux scénarios qui servent à illustrer le rapport (section 1.1) ont une complexité de type « jeu complet ».

**1.1 Rapport final (MAX 2 pages)**:

Le Rapport ne contient PAS de page de titre, ni de table des matières. Il est écrit avec une police 11 au minimum et 14 au maximum, interligne simple. Le rapport est écrit en français ou en anglais ; une orthographe ou une grammaire défaillante peut induire les correcteurs en erreur. Le Rapport ne duplique pas les précédents. Il est organisé en 2 parties, chacune de 1 page maximum:

Première partie sur l'éventuelle évolution de l'organisation du code et sur l'exécution (max 1 page):

- **Optionnel** : liste des éventuels changements dans la structuration de vos données ou de vos algorithmes.
- **Description de l'exécution de votre programme pour les fichiers de test suivants :** insérer les parties utiles des captures d'écran demandées (pas de photo) et décrire brièvement le comportement du programme.
  - **t22.txt :** montrer 4 images de la chaîne à différentes étapes de sa construction, donc avec des nombres d'articulation différents. Deux des étapes doivent aussi avoir utilisé le mode de guidage pour modifier la configuration de la chaine (repasser en mode construction pour pouvoir faire la capture d'écran).
  - **t35.txt :** même contenu que t20.txt avec CONSTRUCTION à la place de GUIDAGE. Montrer l'image de l'arène pour l'état initial puis rester en mode construction et utiliser start/stop/step pour montrer l'image de l'état juste avant la destruction de la chaine (score 4852) par un faiseur puis la mise à jour suivante qui détruit la chaine (le jeu doit pouvoir continuer).

Deuxième partie sur la méthodologie de travail, les contributions individuelles et la conclusion (max 1 page):

- **Activité individuelle** (max 0.5 page en début de cette 2ième partie) de chaque membre selon ce message edstem décrivant la méthode de travail recommandée.

- **Méthodologie**: si vous ne l'avez pas précisé ci-dessus, indiquez comment vous avez organisé votre travail à plusieurs, avec quels outils ? (git?, VSCode ? etc…) ; par quels modules avez-vous commencé, comment les avez-vous testés. Indiquer la proportion de travail simultané en groupe (c'est à dire côte à côte ou en-ligne sur le même code) par rapport au travail indépendant (chacun de son coté). Avec le recul, est-ce que vous modifieriez cette proportion?

  - Quel était le bug le plus fréquent, pourquoi ? Quel est celui qui vous a posé le plus de problème et comment a-t-il été résolu, …).

  - En cas d'usage d'outil d'IA, avez-vous eu l'impression qu'elle vous a aidé ou au contraire fait perdre du temps ? Essayez de quantifier (un peu, beaucoup, etc) même si ça reste très subjectif.

- **Conclusion:** fournissez une brève auto-évaluation de votre travail (dont la robustesse et la performance du code) et de l'environnement mis à votre disposition (points forts, points faibles, améliorations possibles).

Le rapport final doit être inclus dans le fichier archive du rendu final (en format pdf).

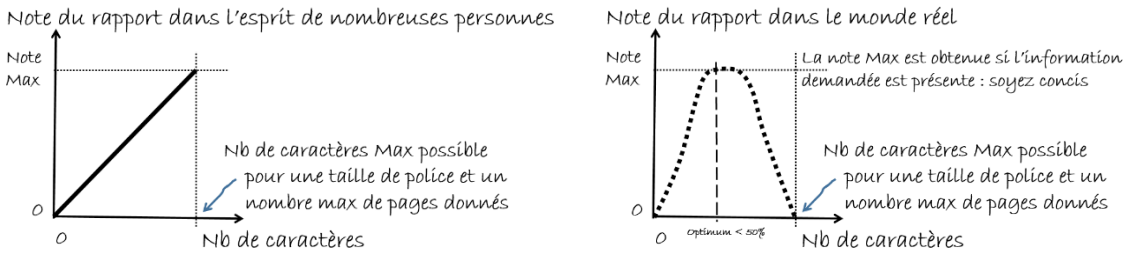



Figure rendu3.1 : un mauvais rapport dilue l'information utile jusqu'à atteindre le nombre maximum de caractères possibles sur la page (fig gauche) : en fait ce type de rapport sera pénalisé parce qu'il est peu lisible ; un bon rapport est celui qui fournit les informations demandées avec concision avec une mise en page aérée et lisible (fig droite)

## *2. Organisation du travail*

Nous recommandons de tester votre jeu avec le bouton Step (se souvenir aussi que le clic gauche/droit de souris doit produire une seule mise à jour équivalente à l'appel effectué pour Step quand le jeu est à l'arret). Cette approche permet de controller chaque état successif du jeu sur l'affichage graphique car le dessin est un puissant outil de mise au point.

N'hésitez pas à compléter avec du code de scaffolding qui affiche la valeurs des variables importantes dans le terminal pour compléter vos outils de mise au point. D'ailleurs, en cas de bug, l'affichage dans le terminal devrait être redirigé vers un fichier de texte (cf cours redirection de la sortie) pour pouvoir l'ouvrir avec un éditeur de texte après le crash du programme et analyser l'évolution des valeurs de vos variables.

Utilisez les fichiers de test déjà fournis et complétez-les avec vos propres fichiers de test, les plus simples possible, pour vérifier les scenarios décrits dans la table ci-dessous.

| Fonctionalité (section 2.2 donnée générale et complément en page1) | Forme minimal du test |
|---|---|
| Fig6a et Fig7a => peu importe le mode CONSTRUCTION/GUIDAGE => dessin de la region de capture (cercle rouge) et du but (point noir à l'opposé du cercle) en fonction de la position de la souris. Le dessin du but est aussi très utile pour faire un bon choix de capture. Un petit risque de division par zero existe si la souris se trouve sur l'origine (cas à détecter => ne rien calculer sinon ça fait planter le programme ). Le traitement de cet événement de **movement de la souris** ne produit aucune mise à jour complète du jeu si celui-ci est à l'arret. | Pas besoin d'entités ; t00.txt suffit mais t25.txt facilite le test du passage sur l'origine. On doit voir les deux éléments bouger à chaque movement de la souris. |
| Fig6a->Fig6b et Fig7a => mode CONSTRUCTION: avec un clic du bouton gauche de la souris on capture une particule constituant la racine de la chaine (rappel: le score diminue d'une unite car une mise à jour complète est faite à chaque **clic de bouton de souris**). On doit constater la diminution du nombre de particule, l'augmentation du nombre d'articulation, le changement de couleur de particule vers la couleur de l'articulation et le cercle de capture doit être centré sur la particule capturée. Tout comme la racine le but devient fixe de l'autre coté de l'arène. | Il suffit d'une particule immobile proche d'un bord de l'arène. |
| Même test mais avec 2 particules assez proches (ou superposées) ; la capture ne doit pas être possible tant que le cercle de capture contient 2 particules. | Ajouter une seconde particule proche de celle du test ci-dessus. |
| Fig6c : une des 2 particules est mobile et permet de créer une seconde articulation quand passe dans le cercle de capture centrée sur la racine. Le cercle de capture doit se déplacer sur la dernière articulation (l'effecteur). | modifier le fichier precedent pour que la seconde particule entre dans le cercle de capture de la racine pour être capturée comme seconde articulation. |
| Fig7b: on continue avec l'état final du cas ci-dessus pour un premier test du mode de GUIDAGE. On doit constater que les clics gauche et droit de la souris modifient le mode indiqué dans l'interface. En mode construction, rien ne bouge car il n'y a plus de particule à capturer. Par contre en mode guidage (et jeu à l'arret) chaque clic droit de la souris provoque une mise à jour du jeu avec un alignement du segment selon la direction reliant la racine au **but intermédiaire**. Du fait du calcul du but intermédiaire, il se peut qu'il faille 2 mises à jour pour aligner l'effecteur en direction du point où se trouve la souris. On testera aussi des cas qui font sortir l'effecteur de l'arène où qui causent une division par zéro ; ils doivent donc être ignorés (la chaine ne bouge pas). | Utiliser le même fichier que ci-dessus. |
| Poursuivre avec des tests qui permettent de créer une chaine avec une articulation supplémentaire. Reprendre le test du guidage à chaque fois qu'on rajoute une articulation. La Fig8 donne le principe de l'algorithme qu'il est relativement facile de simuler sur le papier pour prédire la configuration attendue pour la chaine. | Fichiers ajoutant une particule supplémentaire qui sera facile à capturer. |
| Tester le cas où on gagne le jeu en ayant le but à l'intérieur de la zone de capture. Changer l'état du jeu à WON et stopper les mises à jour. Autoriser seulement exit, Open et restart. | Fichier avec suffisamment de particules bien disposées pour être capturées et ainsi atteindre le but. |
| Tester le cas de destruction de la chaine par un faiseur en commençant par le cas le plus simple : un faiseur superposé à une particule immobile supprime une chaine dès qu'elle existe. Ensuite poursuivre avec : 1) une chaine avec plus d'une articulation qui atteint un faiseur immobile, puis 2) une chaine immobile qui est atteinte par un faiseur mobile. Toutes les articulations doivent être testées. Seule la position des articulations (Fig5b) est utilisée pour ces tests (pour les simplifier : pas de problème pour Fig5a). | Éditer les fichiers précédents qui conviennent. |
| Les deux scenarios ci-dessous doivent être utilisés pour illustrer le rapport | |
| t22: nous testerons ce cas plus complet en jeu continu (start) | t22 |
| t35 :nous testerons ce cas plus complet en jeu continu (start) au moins jusqu'à la destruction de la chaine | t35 |

## *3. Forme du rendu3*

**Convention de style :** il est demandé de respecter les conventions de programmation du cours.

- On autorise *une seule* fonction/méthode de plus de 40 lignes (max 80 lignes) car le module gui est fourni et respecte déjà la consigne de longueur des fonctions/méthodes.

**Documentation** : l'entête de vos fichiers source doit indiquer le nom du fichier et les noms des membres du groupe avec, **pour les fichiers .cc**, une estimation du pourcentage de contribution de chaque membre du groupe au code de ce fichier.

**Rendu** : pour chaque rendu **UN SEUL membre d'un groupe** (noté **SCIPER1** ci-dessous) doit téléverser un fichier **zip**[1] sur moodle (pas d'email). Le non-respect de cette consigne sera pénalisé de plusieurs points. Le nom de ce fichier **zip** a la forme :

**SCIPER1_ SCIPER2.zip**

Compléter le fichier fourni **mysciper.txt** en remplaçant 111111 par le numéro SCIPER de la personne qui télécharge le fichier archive et 222222 par le numéro SCIPER du second membre du groupe.

Le fichier archive du rendu2 doit contenir (**aucun répertoire**) :

- Fichier pdf du rapport (section 1.1, p2)
- Fichier texte édité **mysciper.txt**
- Votre fichier **Makefile** produisant un executable **projet**
- Tout le code source (.cc et .h) nécessaire pour produire l'exécutable.

*On doit obtenir l'exécutable **projet** après décompression du fichier **zip** en lançant la commande* **make** *dans un terminal de la VM (sans utiliser VSCode).*

**Auto-vérification :** Après avoir téléversé le fichier **zip** de votre rendu sur moodle (upload), récupérez-le (download), décompressez-le et assurez-vous que la commande **make** produit bien l'exécutable lorsqu'elle est lancée dans un *terminal de la VM* (sans utiliser VSCode) et que celui-ci fonctionne correctement.

**Exécution sur la VM**: votre projet sera évalué sur la VM à distance (compilation avec l'option -std=c++17).

**Backup** : Il y a un backup automatique sur votre compte myNAS.

**Debugging :** Visual Studio Code offre un outil intéressant pour la recherche de bug (VSCode tuto).

[1] Nous exigeons le format zip pour le fichier archive