Conseils pour réaliser un projet

— Une fois que le cahier des charges est bien clair et établi, prendre le temps de réfléchir à la conception du projet avant l'écrire. Établir les fonctions et sous fonctions, la hiérarchie des structures/classes sans forcément écrire leur "corps". Écrire les noms de méthode/fonctions/classe vide et les commenter avant de remplir leur corps.

— Lorsque vous écrivez du code pensez à anticiper les améliorations que vous pourrez ajouter plus tard, quitte à écrire des fonctions vide sur le moment et à les remplir après. Dans certains langages les outils d'héritages et d'encapsulations (POO) sont faits pour faciliter l'écriture de ces anticipations.

— Organiser les fonctions de la manière la plus indépendante possible. (En général plus il est facile de décrire la fonction par un texte plus elle est indépendante)

— Essayer de séparer la "physique" du programme (si possible dans une structure/classe) de son interface graphique.

— Une fois la phase conception de votre projet finie répartissez le travail entre les membres du groupe pour l'écriture des fonctions.

— Écrire des fonctions courtes. Plus la fonction est courte plus elle est agréable et facile à lire, à modifier et corriger. Décomposez, si possible, les grandes fonctions en plusieurs plus petites.

— Essayer d'écrire le moins de lignes de code possible (sans perdre en efficacité). Plus le code est court plus il est facile à modifier et à corriger.
  — Si vous avez à choisir entre un code (ou un algorithme ou une structure de donnée) rapide à l'exécution mais long à écrire et un code plus court à écrire mais plus long à l'exécution, pesez votre choix.
  Par exemple, supposons que vous ayez besoin de faire une tâche qui sera exécutée au plus : 1 fois par seconde et que pour cette tâche vous ayez le choix entre 2 algorithmes. L'un très naïf et simple à écrire peut s'écrire en 10 lignes et s'exécute en 300 opérations (élémentaires).
  L'autre plus complexe s'écrit en 100 lignes et s'exécute en 1 opérations (un peu exagéré).
  Le choix penche sans hésitation pour l'utilisation de l'algorithme naïf (facile à corriger, mettre à jour) car finalement la différence entre 1 opération par seconde ou 300 opérations par seconde ne sera pas perceptible pour un utilisateur.
  — Pour minimiser le code vous pouvez aussi utiliser le mécanisme d'héritage des classes.

— Faite régulièrement des essais d'exécution de vos codes. Il vaut mieux améliorer petit à petit votre programme et faire de nombreux tests que d'écrire tout d'un coup. Les erreurs d'exécutions sont plus faciles à cerner lorsque les périodes d'écriture-test sont courtes.

— Pour détecter les erreurs d'exécution utiliser assert. Il faut utiliser la macro assert judicieusement : ce n'est pas pour rien que c'est une macro. Elle n'est pas destinée à être utilisée dans le produit fini. Pour "désactiver" assert utiliser l'option `gcc -DNDEBUG`

— Faite attention à prévenir tous les comportements possibles de l'utilisateur. Pensez toujours au cas "autre".

— Entrer et gérer des commandes et lire un texte formaté peut se faire grâce aux méthodes de flux de la bibliothèque standard. Vous pouvez également utiliser les méthodes printf et scanf de la bibliothèque standard.

— Utilisez les outils de la bibliothèque standard du C++ qui est installée et liée par défaut (avec g++). `http://www.cplusplus.com/reference/`

— Comme bibliothèque graphique vous pouvez utiliser la SDL (Simple DirectMedia Layer), la SFML (Simple and

Fast Multimedia Library) (peut-être) plus facile d'accès.

— Si vous utiliser une interface graphique, vous pourrez utiliser les images issues de `https://opengameart.org/`

— On peut modifier le comportement du terminal par exemple pour "dessiner". Pour cela voir les protocoles associés au terminal VT100 `https://espterm.github.io/docs/VT100%20escape%20codes.html`

— Envoyez une version dès que possible et n'attendez pas le dernier moment, vous pourriez avoir de mauvaises surprises lors de l'envoi : Serveur en panne, Pas de Connections,...

— Certaines erreurs d'exécution peuvent être détectées par le compilateur gcc/g++ si on le lui indique. Pour activer cette option il suffit d'écrire, dans la ligne de commande de compilation, les mots `-Wall` et `-Wextra`. Exemple : `g++ fichier.cpp -Wall -Wextra`.

— **Important** Ne vous fiez pas qu'aux tests sur une seule machine. Il y a des erreurs d'utilisation de la mémoire qui n'apparaissent pas sur certains système et qui font planter le programme sur d'autres. Utiliser les assertions (la macro `assert`, écrivez des fonctions de test automatiques (que vous n'utiliserez pas dans le programme final). Utiliser des outils comme Valgrind ou l'option `-fsanitize=address` de gcc (qui détecte certaines erreurs mémoires).