

# Programmation C – Projet 1

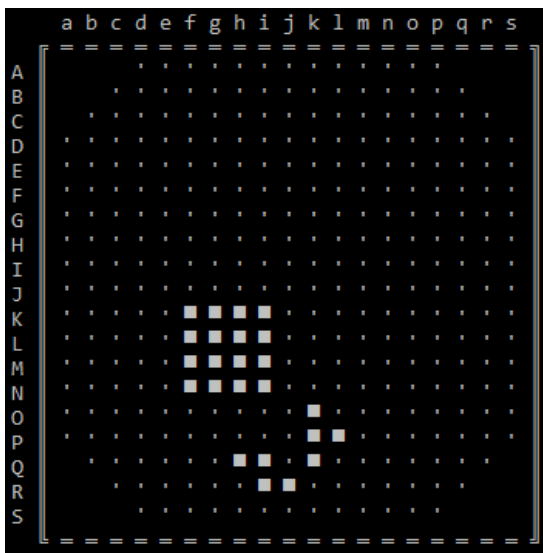
## Comme un air de Tetris

### Description :

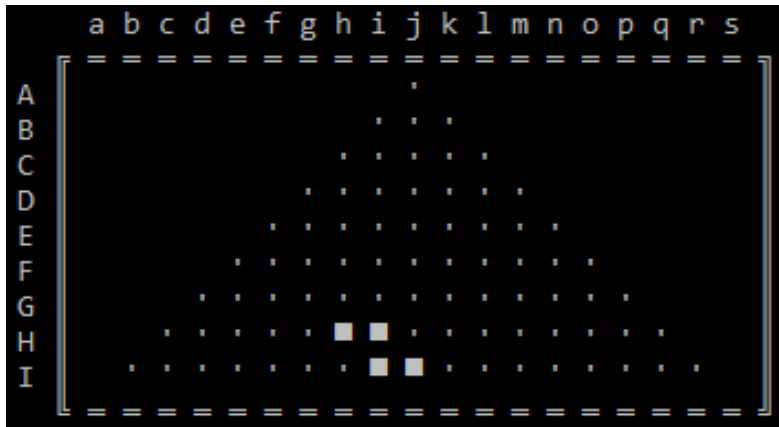
Le Tetris est un jeu qui se présente sous forme d'une matrice où des blocs de différentes formes doivent être posés de sorte que le plateau soit gardé le plus longtemps possible non plein. L'idée est de placer chaque bloc à l'emplacement qui permet d'éliminer un maximum de lignes et/ou de colonnes. Ces dernières sont supprimées automatiquement lorsqu'elles sont pleines.

Dans ce projet, nous souhaitons s'inspirer du Tetris et lui créer une toute nouvelle version.

Il s'agit de partir d'un plateau à 2 dimensions de taille **minimum** 21 x 21 cases - dont les lignes sont désignées par des lettres majuscules ('A', 'B' ...) et les colonnes par des lettres minuscules ('a', 'b', ...) - sur lequel sera délimitée une surface de jeu valide. Cette surface de jeu valide peut prendre trois formes différentes : cercle, losange ou triangle (voir les trois figures ci-dessous).



Dans ce jeu, l'utilisateur dispose d'un ensemble de blocs qu'il devra placer tour à tour sur la surface valide du plateau en saisissant les coordonnées de l'endroit où il veut les insérer. Certains blocs à poser sont communs aux trois formes, mais à chacune d'entre elles d'autres formes plus adaptées peuvent s'ajouter (**Cf. section 3**).



## Application à réaliser :

Au lancement du jeu, un écran d'accueil doit s'afficher proposant deux options :

- Commencer à jouer
- Afficher les règles du jeu.

Le contenu de cet écran d'accueil est laissé ouvert et chacun peut proposer le design qu'il souhaite.

Lorsque le jeu commence, l'écran d'accueil doit disparaître et l'utilisateur est invité à paramétrer son jeu :

1. Il doit d'abord choisir la dimension de son plateau de jeu et sa forme parmi :

- Cercle
- Losange
- Triangle

2. Choisir parmi deux politiques de suggestion des blocs :

- Afficher à chaque tour de jeu l'ensemble des blocs disponibles et l'utilisateur en sélectionne un.
- Afficher uniquement 3 blocs sélectionnés aléatoirement.

**Note** : le paramétrage réalisé à cette étape reste inchangé durant toute la période de jeu.

Pendant la partie, l'application doit être capable de vérifier si les coordonnées saisies par l'utilisateur sont valides :

- Elles sont valides si les cases du bloc choisi peuvent être toutes placées sur des cases vides se trouvant sur la surface valide du jeu.

- Elles ne sont pas valides si certaines - ou toutes les - cases du bloc choisi se situent en dehors de la surface valide du jeu ou que l'emplacement sélectionné n'a pas suffisamment de cases pour accueillir le bloc choisi.

A chaque insertion d'un bloc, un test est effectué pour vérifier s'il y a des lignes et/ ou des colonnes pleines. Dans ce cas, celles-ci doivent être annulées (réinitialisées). Dans le cas de l'annulation de la ligne, les cases pleines se retrouvant au-dessus doivent descendre pour s'empiler au fond de la surface de jeu. En revanche, à l'annulation de la colonne, aucun décalage n'est à prévoir.

De plus, l'annulation des lignes/colonnes doit entraîner le calcul d'un score qui s'affiche à côté du plateau de jeu. Une fonction simple pour celui-ci serait de compter le nombre de cases annulées. Cependant, la proposition de toute autre fonction est acceptée.



### Condition d'arrêt :

Pour insérer un bloc sur le plateau, l'utilisateur dispose de trois tentatives pour saisir des coordonnées valides. Si à l'issue de 3 tentatives successives, les positions choisies sont à chaque fois invalides, alors le jeu s'arrête.

A la fin de la partie, un message doit s'afficher à l'écran rappelant le score obtenu.

## Aspects techniques :

La réalisation de ce projet nécessite de bien réfléchir à la structure de stockage qui représentera tous vos éléments, ainsi qu'une segmentation de votre code en fonctions.

### 1. Structure de données

- D'abord, chaque bloc est vu comme une matrice 2D **créée dynamiquement** de taille  $N \times M$  ( $N$  = plus grande hauteur parmi tous les blocs proposés dans la série,  $M$ =plus grande largeur parmi tous les blocs proposés dans la série).
- Puis, l'ensemble des blocs à utiliser dans le jeu est stocké dans un seul tableau global de taille **nb\_blocs**.

Exemple : Si l'on dispose des blocs :

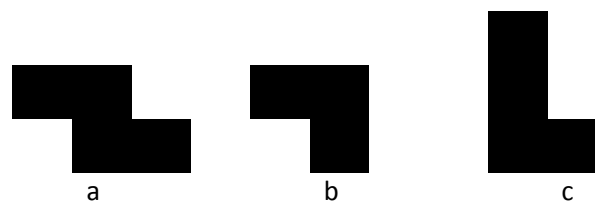
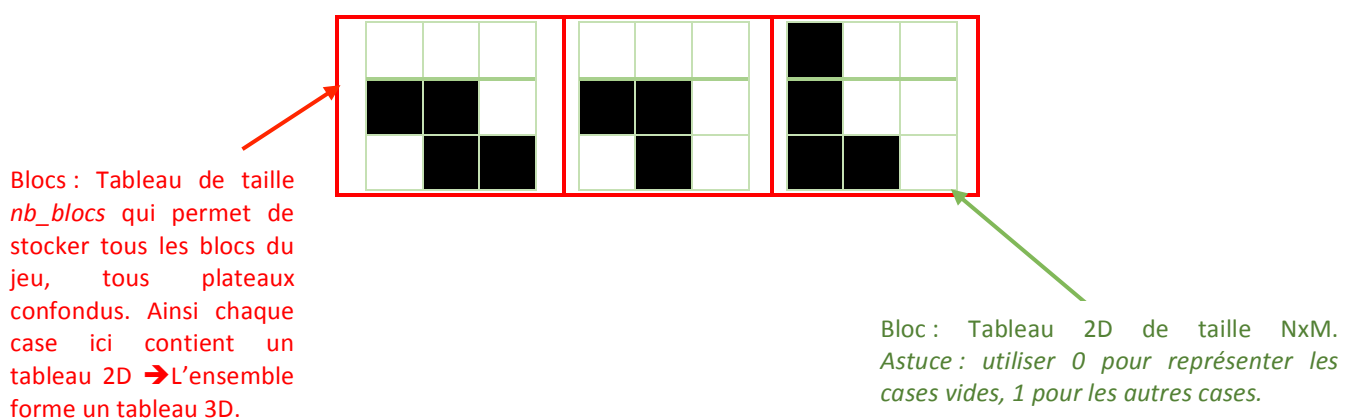


Figure 1

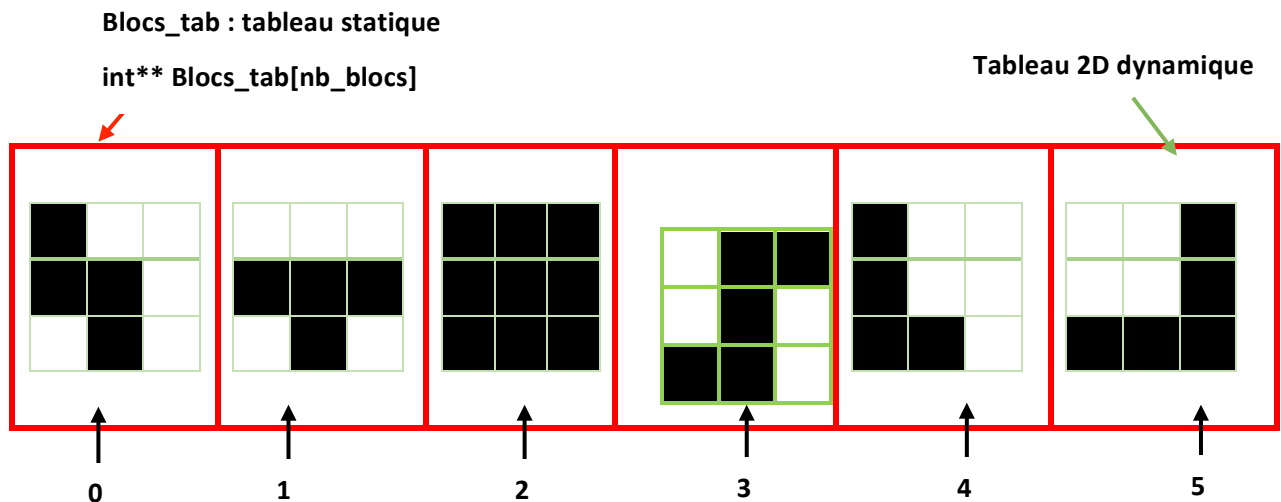
La taille des matrices où ils seront stockés sera de  $3 \times 3$  car le bloc de la figure 1.c est le plus grand en hauteur ( $=3$ ) et le bloc de la figure 1.a est le plus large (largeur = 3).

A leur tour, tous les blocs sont stockés dans un seul tableau comme le montre la figure ci-dessous.

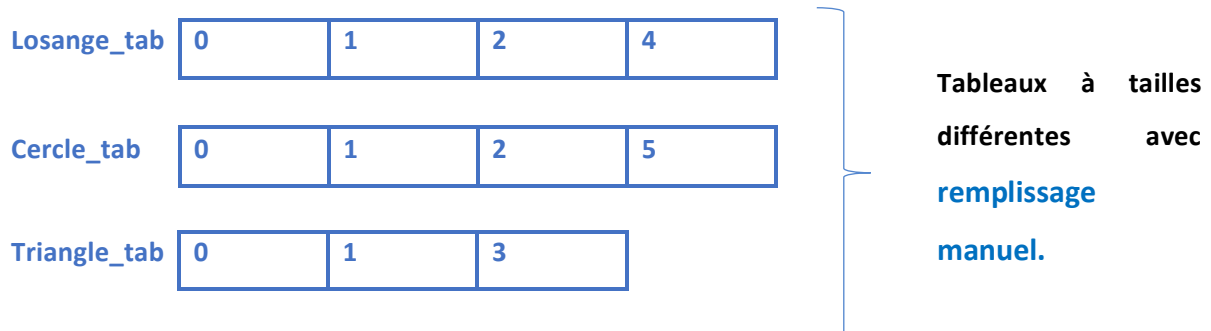


- Pour chaque forme de plateau, un tableau est associé dont le contenu des cases représente les indices où sont stockés les blocs qui lui sont autorisés (**Cf. section 3**). Dans ce jeu, il y aura donc trois tableaux de tailles différentes : **losange\_tab**, **cercle\_tab** et **triangle\_tab**.

**Exemple** : Supposons que le tableau **Blocs\_tab** contienne les blocs suivants :



Les trois tableaux associés aux différents plateaux de jeu seront dans cet exemple de la forme suivante :



Ainsi, l'accès au bloc d'indice 5 pour le plateau de forme Cercle par exemple, se fera par : **Blocs\_tab[Cercle\_tab[3]]** .

## 2. Fonctions

- Ecrire une fonction **creer\_tab2D\_dyn** qui permet de créer un tableau 2D de façon dynamique. Notez que dans ce jeu, le plateau de jeu et les différents blocs doivent être créés dynamiquement.
- Pour chaque forme de bloc **forme**, écrire une fonction **creer\_bloc\_ forme** qui prend en paramètres les dimensions d'un bloc donné, crée un tableau 2D dynamiquement et le

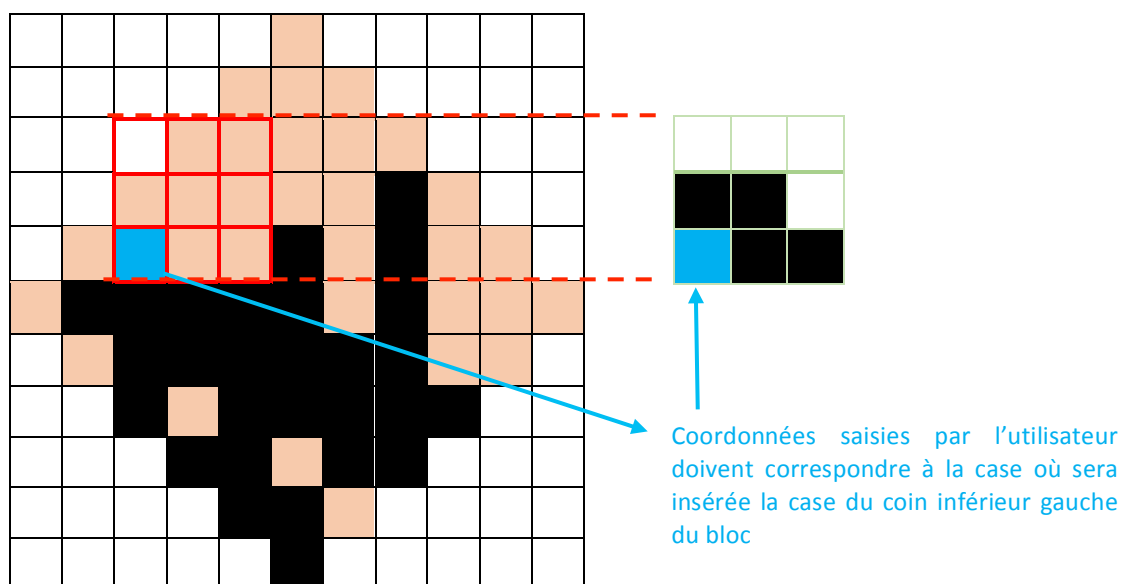
remplit par des 1 et des 0 en fonction de la forme souhaitée. Cette fonction pourra être utilisée pour la création du plateau de jeu, mais également pour la création des blocs.

- Ecrire une fonction **remplir\_case\_tab** qui prend en paramètres un tableau 2D et un indice i et qui permet, en fonction de l'indice i, d'appeler une des fonctions précédentes pour construire le bloc associé (utiliser un switch case sur l'indice i pour donner la possibilité de créer l'ensemble des blocs).
- Ecrire une fonction **afficher\_plateau** qui prend en paramètres un plateau 2D et qui l'affiche à l'écran. L'affichage se fera en utilisant les codes ASCII des différents symboles et caractères. Le choix de ses derniers est libre, la seule condition est que l'affichage soit clair à l'écran.
- Ecrire une fonction **afficher\_blocs** qui, en fonction de la forme du plateau choisi, affiche la liste de tous les blocs qui lui sont associés.
- Ecrire une fonction **selectionner\_blocs** qui permet de sélectionner les blocs à proposer à l'utilisateur selon l'une des 2 politiques expliquées ci-dessus.
- Ecrire une fonction **verif\_validite** qui prend en paramètres un bloc et les coordonnées sur le plateau où il doit être placé. Elle retourne 1 si le bloc peut être posé sur la zone valide du plateau, 0 sinon.

#### Règle de placement d'un bloc sur le plateau :

- On suppose que l'insertion d'un bloc sur le plateau se fera par la superposition de la matrice qui le contient sur l'endroit où l'on veut l'insérer. Pour cela, il faudra faire coïncider la cellule du coin inférieur gauche du bloc avec les coordonnées saisies par l'utilisateur.

#### Exemple :


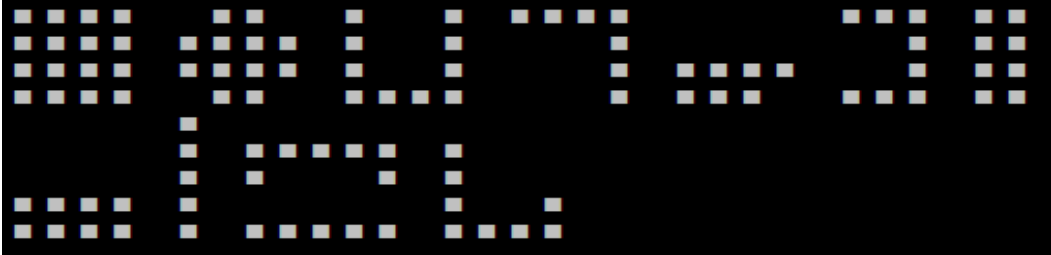
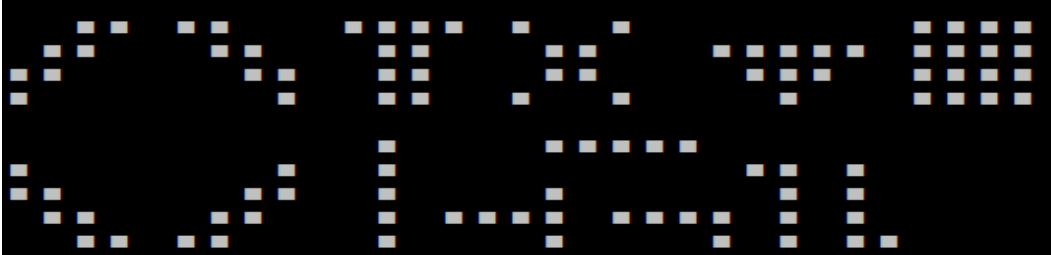
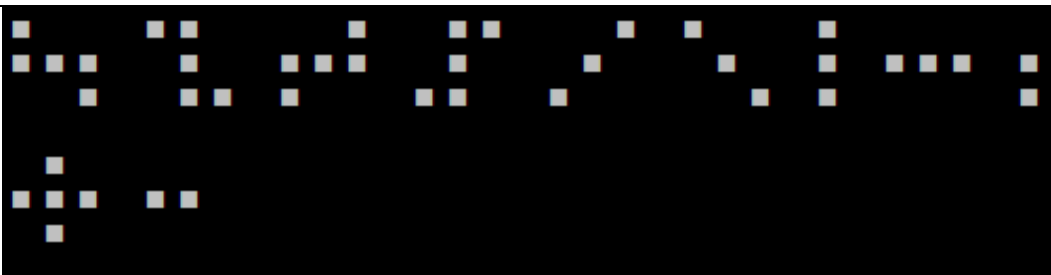


- Ecrire une fonction **placer\_bloc** qui permet de placer un bloc B dans le plateau P à une position (i, j). Les cases occupées par le bloc peuvent prendre la valeur 2. A l'affichage, elles doivent se distinguer par un symbole différent de celui du plateau.
- Ecrire une fonction **etat\_ligne** qui retourne 1 si la ligne donnée en paramètre est pleine, 0 sinon.
- Ecrire une fonction **etat\_colonne** qui retourne 1 si la colonne donnée en paramètre est pleine, 0 sinon.
- Ecrire une fonction **annuler\_ligne** qui permet de réinitialiser la ligne donnée en paramètre.
- Ecrire une fonction **annuler\_colonne** qui permet de réinitialiser la colonne donnée en paramètre.
- Ecrire une fonction **decaler\_lignes** qui à partir d'un numéro de ligne donné en paramètre, permet de faire des décalages vers le bas. Les cases dont le décalage risque de les mettre hors de la zone de jeu valide, ne doivent pas changer de position.
- Ecrire une fonction **calcul\_score** qui permet de mettre à jour le score à chaque annulation de lignes/colonnes.
- Ecrire le programme principal en utilisant les fonctions développées précédemment, et d'autres si nécessaire, qui permet le lancement du jeu.

**IMPORTANT !!**

1. Le code doit être organisé en trois fichiers : **projet1c.h**, **projet1c.c** et **main.c**
2. Dans toutes les fonctions et le programme principal, la saisie doit être sécurisée selon les types et les valeurs autorisées des variables.

### 3. Blocs associés à chaque plateau de jeu

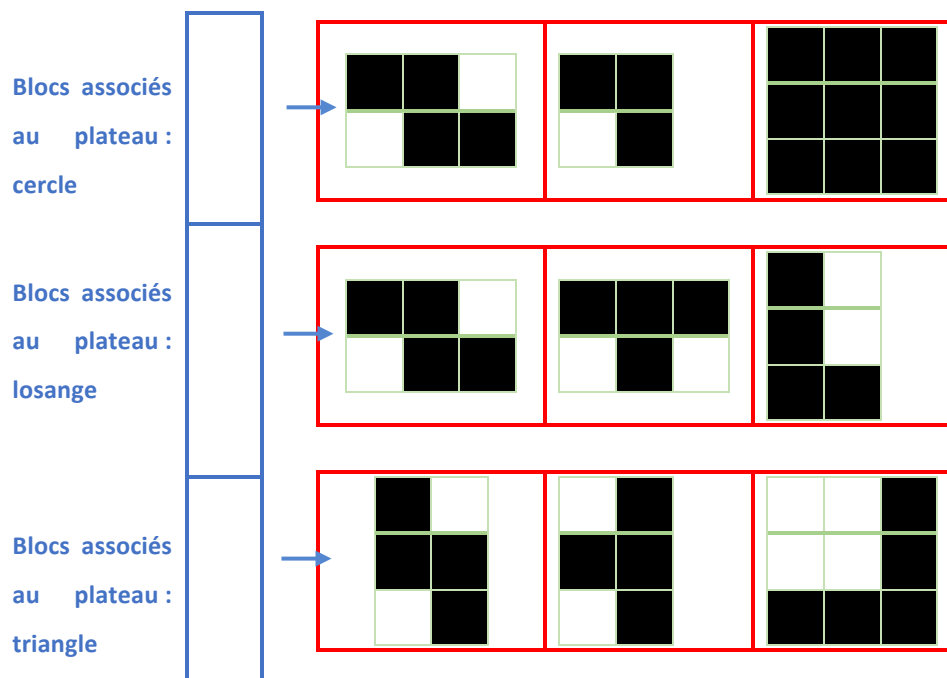
Plateau	Liste des blocs
Tous (Blocs communs à tous les plateaux)	
Cercle	
Losange	
Triangle	

### Optionnel

Ci-dessous quelques fonctionnalités additionnelles que les plus motivés peuvent intégrer pour avoir une version du jeu plus poussée :

- En fonction de la valeur du score, ajouter de la difficulté au jeu en suggérant des blocs pas toujours faciles à insérer.
- Afin de généraliser ce jeu pour différents plateaux et différentes formes de blocs, le représenter par une structure où tous les tableaux sont créés dynamiquement. Aussi, les tailles des tableaux associés aux formes utilisées dans le jeu, ne sont pas forcément de la même taille.





- Ajouter une 3<sup>ème</sup> politique de suggestion des blocs qui permet d'afficher à l'utilisateur à chaque tour de jeu 3 blocs uniquement parmi ceux qui peuvent être insérés sur le plateau.
- Changer la condition de validité des coordonnées saisies par l'utilisateur telle que : les coordonnées saisies ne sont acceptées pour placer un bloc que s'il existe un chemin que le bloc peut emprunter depuis le haut du plateau jusqu'à la position souhaitée. Pour placer le bloc en question, simuler son déplacement sur le chemin préalablement calculé.

## Consignes générales

- 1) Le projet est à réaliser en binôme. Un seul groupe à trois étudiants pourrait être accepté dans le cas d'une imparité dans le groupe.
- 2) La remise de votre travail doit inclure:
  - Un fichier .zip contenant l'ensemble des fichiers source (fichiers .h et .c).
  - Un rapport de 10 à 15 pages permettant de:
    - détailler vos solutions (en langage algorithmique) et vos choix de structures de données.
- 3) Tout dossier remis doit être renommé comme suit : NOM1\_NOM2 (avec NOM1 et NOM2 : noms des étudiants qui ont réalisé le projet).

- 4) Le projet est à rendre sur Moodle.

Les travaux sont à rendre au plus tard le 23/04/2020 à 23:59

## **Planning du projet**

- Dimanche 22/03/2020: Dépôt du projet sur moodle
- Semaine du 30/03/2020: Suivi du projet
- Semaine du 20/04/2020: Soutenances du projet

## **Modalités d'évaluation**

La note finale du projet N°1 sera composée de :

- . 1) La note du code (/12)
- . 2) La note du rapport (/4)
- . 3) La note de soutenance (/4)