Projet de C jeu Bataille Navale

Le but de la bataille navale est de couler tous les bateaux de son adversaire. Dans un premier temps, on place les bateaux. Votre adversaire, sans voir votre placement, place aussi ses propres bateaux. Ensuite, chacun à son tour, vous essayez de trouver et couler les bateaux de l'adversaire. Les règles du jeu sont à consulter sur internet.

Objectif

Jouer, paramétrer des options du jeu, sauver des parties, des scores de joueur, rejouer des parties.

Vous devez implanter la version suivante de ce jeu :

- Un joueur contre la machine
- Le joueur et le programme doivent placer chacun 4 bateaux de taille 2 à 5 cases sur des grilles respectives de taille 10×10 (cf. Figure 1)
- Les bateaux sont placés horizontalement ou verticalement
- Dans une version ultérieure le nombre de bateaux de chaque taille sera paramétrable (2,3...)
- Le joueur aura la possibilité d'annuler un coup et ce autant de fois qu'il le souhaite (possibilité de revenir de *n* coups en arrière pas à pas).

Règles de programmation à respecter :

1. Respecter une programmation modulaire
2. Séparer saisies, traitements et affichages
3. Respecter les normes (MVC Modèle Vue Contrôleur)
4. Proposer des fonctions de test et des jeux de test au moins pour la partie contrôleur.
5. Pas de saisie ni d'affichage dans une fonction de traitement ; des pénalités seront appliquées dans le cas contraire
6. Les saisies et affichages sont uniquement dans les fichiers associés à la vue
7. Tous les fichiers seront documentés avec : leur rôle et pour chaque fonction un commentaire correspondant à ce que vous avez vu en premier semestre. De même la norme du programmeur du semestre 1 reste à respecter
8. Implantation dynamique au minimum de la pile de coups.

Modèle MVC séparer :

- Le modèle (les structures de données représentant le jeu : grille, bateaux, la pile des coups joués),
- La vue (affichages) et

- Le contrôleur (le chef d'orchestre en particulier le programme principal).

Afin de vous aider dans votre programmation nous proposons un découpage en module et une analyse conception incomplète.

1. Les fonctionnalités du jeu

1.1. Menu principal

Menu de démarrage :

Permet de créer une nouvelle partie, de charger une partie enregistrée dans un fichier, de consulter les meilleurs scores, de consulter les règles du jeu ou de quitter le programme.

- Créer une nouvelle partie
- Charger partie (recommence une partie sauvegardée au moment où elle s'est arrêtée)
- Consulter les meilleurs scores[1]
- Afficher règles
- Quitter

Vous pouvez faire des sous menus mais à tout instant on doit pouvoir quitter une partie en l'enregistrant ou pas dans un fichier et en mettant à jour les meilleurs scores.

1.2. Créer une nouvelle partie

Permet de lancer une nouvelle partie (débute le jeu) après avoir :

- paramétré (changer les couleurs des bateaux, leur nombre, ...) le jeu à partir de valeurs saisies ou du fichier de paramètres par défaut [2]
- Saisi les joueurs (nom, prénom)
- Saisi les positions des bateaux du joueur

2. Exemple d'interface (vue)

2.1. Exemple d'écran d'affichage

Figure 1 est présenté un exemple d'écran de jeu. Cet écran contient :

1. Une grille pour le joueur qui affiche les bateaux du joueur et les coups de la machine,

[1] En fin de partie si le score est un des meilleurs scores parmi les 10 meilleurs scores enregistrés dans un fichier meilleurs.dat alors mettre à jour ce fichier. On retourne alors sur le menu initial.

[2] Le fichier des paramètres par défaut se nommera defaut.dat.

2. Une grille pour la machine qui affiche les missiles tirés par le joueur et indique s'ils sont à l'eau ou si ils ont atteint une cible avec des couleurs différentes pour touché et coulé,
3. Un affichage des informations sur le score du joueur (nombre de missiles tirés, nombre de ceux qui ont atteint une cible, score du joueur),
4. Un affichage similaire pour le score de la machine,
5. Un espace pour les affichages des messages et la saisie des coordonnées des tirs.

Bataille Navale

Joueur

	A	B	C	D	E	F	G	H	I	J
1										
2			*							
3										
4										
5						*				
6										
7										
8										
9										
10										

Machine

	A	B	C	D	E	F	G	H	I	J
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										

Nombre de coups	2	Nombre de coups	2
Nombre de touche	1	Nombre de touche	1
Score	...	Score	...

Saisie et Affichages

Figure 1 : Exemple d'écran de jeu

3. Calcul des scores

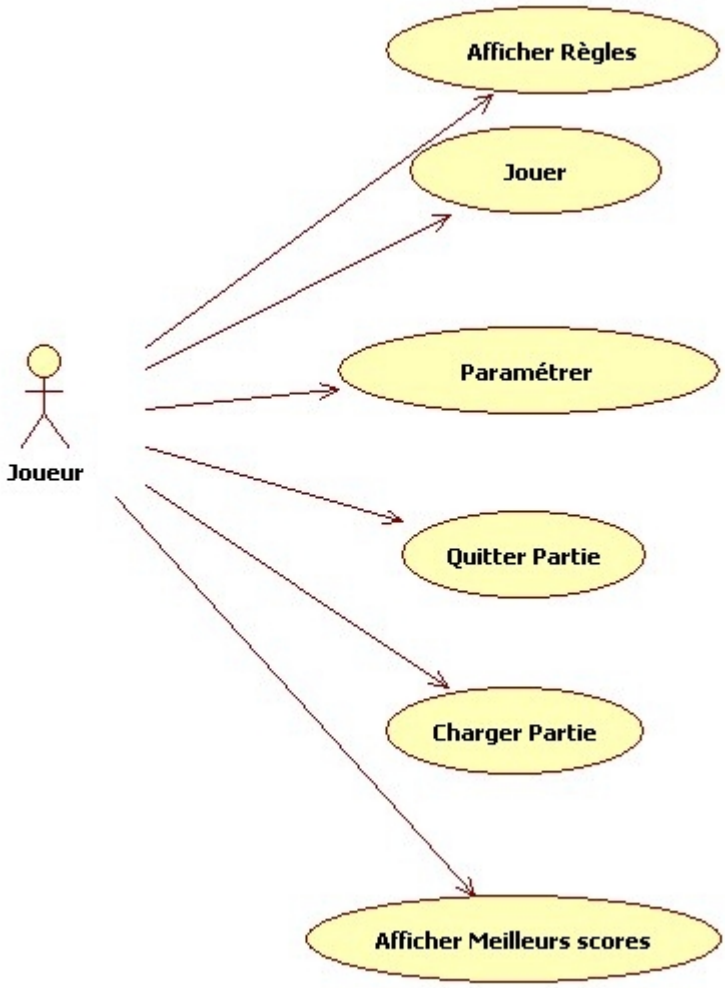
On propose de calculer le score de la façon suivante :

Nombre de cases de la grille de jeu – nombre de missiles envoyés

4. Diagramme UML

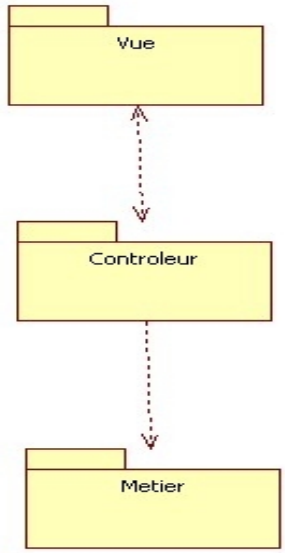
Le modèle proposé est volontairement incomplet. Il représente le découpage minimum demandé, tout découpage supplémentaire en vue de donner une meilleure lisibilité du projet est encouragé.

4.1. Uses Cases

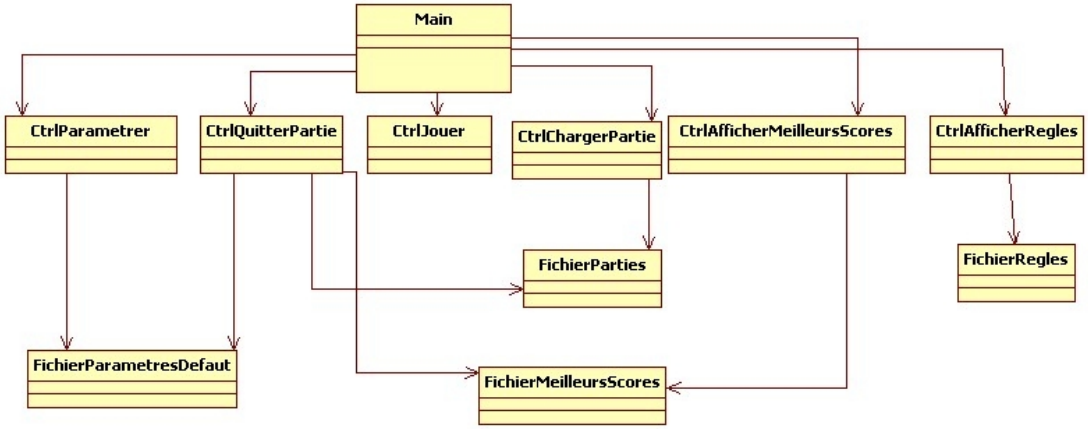


4.2. Diagramme de classe

4.2.1. Packages

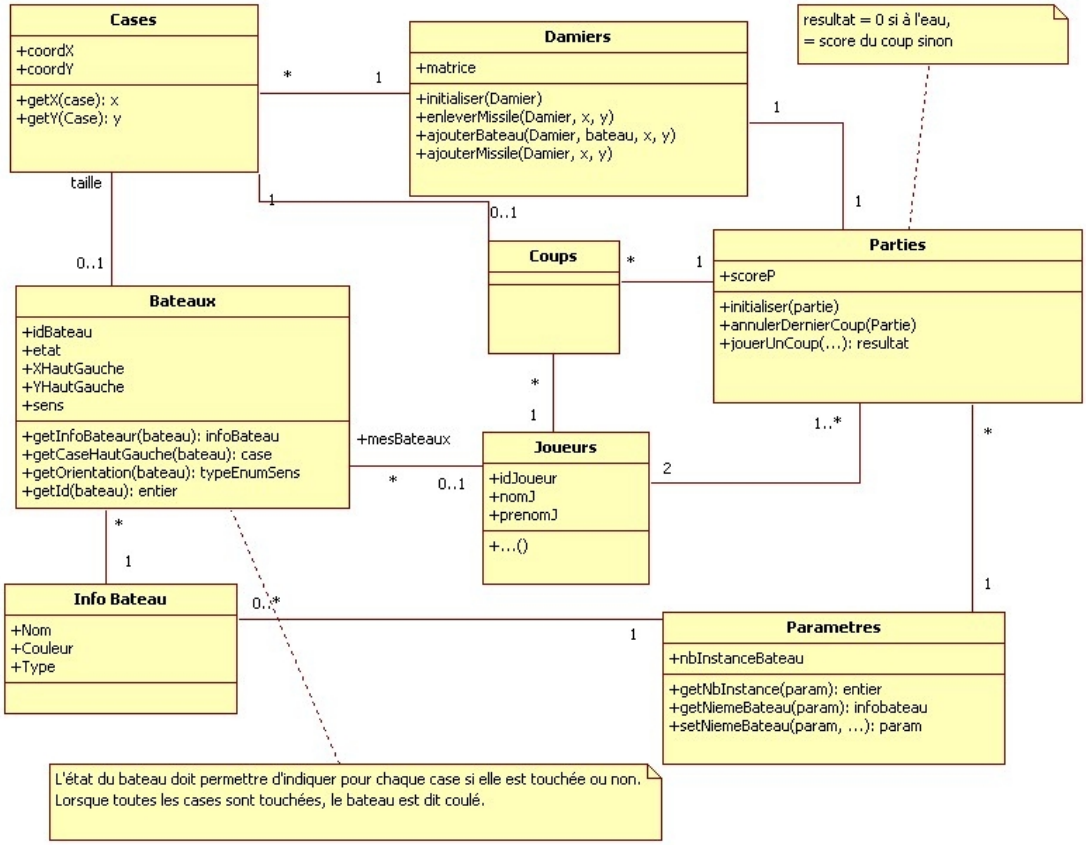


4.2.2. Contrôleur



4.2.3. Vue

4.2.4. Métier (Modèle incomplet)



5. Conception : Découpage du travail demandé

Semaine 1-2 : Paramétrage de la partie.
Le module parametre contient les paramètres de la partie. Il est constitué des fichiers : **parametre.h** et **parametre.c.**
Les paramètres doivent pouvoir être saisis, affichés et donc une vue est associée à ces paramètres. **vueParam.h vueParam.c**
De plus les paramètres doivent pouvoir être sauvegardés et chargés depuis un fichier. En effet, lorsque le joueur ne veut pas changer le paramétrage, on veut pouvoir charger des paramètres par défaut. De même, en cas de reprise d'une partie, il faut retrouver son paramétrage.

Les paramètres nécessaires à une partie sont :

1. Le joueur et la machine ont par défaut un bateau de chaque type (remorqueur, porte-avion, sous-marin, cargot). Le type du bateau conditionne sa taille. Prévoir, en cas d'extension du projet de pouvoir paramétrer le nombre d'instances de chaque type de bateau.
2. Une description constructeur de chaque bateau comportant son nom, son type et sa couleur.
3. Le nom du joueur (« anonyme » par défaut).

Avant de définir le type paramètres, nous devons donc définir un type descriptif bateau (Info Bateau dans le modèle métier).

Pour cela on définit le type EType

```
typedef enum {remorqueur=2, porteAvion, sousMarin, cargot} Etype;
```

Ce type énuméré entraîne que remorqueur=2, porteAvion=3... ces valeurs sont supposées être les longueurs respectives des bateaux. Ce sont des constantes de type « entier ».
On suppose le module couleurs donné, ce module contient un tableau constant qui met en correspondance, les noms des couleurs (chaîne de caractères, une lettre pour un choix facilité par menu, et une valeur utile au système pour afficher en couleur).
Ce module vous est donné sous moodle vous pourrez donc l'intégrer à vos projets. Un certain nombre de fonctions utiles sont fournies.

Ce module contient donc un type couleur et un tableau de valeurs prédéfinies.

//tableau

```
const Couleur tableCouleurs[KCOULEURS_NBCOULMAX] = {
        {'F',"FUSHIA", MAGENTA} ,
        {'R',"ROUGE", RED},
        {'B', "BLEU", BLUE},
        {'C', "CYAN",CYAN },
        {'M',"MARRON",BROWN},
        {'J', "JAUNE" ,YELLOW},
        {'V',"VERT", GREEN}
} ;
```

//fichier couleur.h

```
#ifndef _COULEURS_H
#define _COULEURS_H
#include "coniocouleur.h"
#define KCOULEURS_NBCOULMAX 7
#define KCOULEURS_LGNOMCOUL 10

        typedef     struct {
                    char lettre;
                    char nom[KCOULEURS_LGNOMCOUL+1];
                    COLORS num;
            }Couleur;

        //  retourne la couleur associée à la lettre dans le tableau des couleurs tableCouleurs
        Couleur lettreToCouleur (char pLettre);

        //retourne la couleur à l'indice pI dans le tableau des couleurs tableCouleurs
        Couleur getCouleurFromNum (int pI);

        // fonction d'accès à chacun des champs de la structure couleur
        COLORS getColor (Couleur pCouleur);
        char getChar (Couleur pCouleur);
        void getNom (Couleur pCouleur, char pNom[]);
#endif
```

Objectif définir et tester toutes les fonctions des modules parametre et vueParam.
Chaque programmeur développe un module et teste le module développé par l'autre programmeur.
Vous devez donc développer également des modules testParametre et testVue.

Vous pouvez compléter les modules fournis

```
// parametre.h
#ifndef _PARAM_H
#define _PARAM_H
#include <stdio.h>
#include "couleurs.h"
```

```
#define K_NBCOULEURS 4
#define K_NBTYPEBATEAUX 4
#define K_LGNOM 25
typedef enum {remorqueur=2, porteAvion, sousMarin, cargot} EType;

typedef struct
{
      int mCouleur /*indice dans la table des couleurs*/;
      EType mType;
      char mNom[K_LGNOM];
}      TInfoBateau;

//************************
/*        N : getCouleur
          D : donne le l'index de la couleur dans la table des couleurs de couleurs.h/.c de la couleur de l'info bateau pB
          E : pB
          S :
          R : le numéro de la couleur
          Prec : -
*/
int getCouleur(const TInfoBateau * pB);

//*****************************
/*        N : getType
          D : donne le type de l'info bateau pB
          E : pB
          S :
          R : le numéro du type
          Prec : -
*/
EType getForme(const TInfoBateau *pB);

//*****************************
/*        N : getNom
          D : donne nom du bateau pB
          E : pB
          S : pNom
          R :
          Prec : -
*/
void getBNom(const TInfoBateau * pB, char pNom[]); //voir tp sur les chaines de caractères dynamique*/

//*****************************
/*        N : setInfoBateau
          D : affecte les infos pNom pCouleur et pType à l'info bateau pB
          E :  pNom pCouleur et pType
          S : pB
          R :
          Prec : -
*/
void setInfoBateau (TInfoBateau *pB, char pNom[], int pCouleur , EType pType);

typedef struct
{
   //ajouter char mNomJoueur[KLGMAXNOM]; et les fonctions nécessaires
     int mNombreInstanceBateaux;
     TInfoBateau * mBateauxDuJoueur;
          /*tableau dynamique de mNombreInstanceBateaux*K_NBTYPEBATEAUX bateaux du joueur*/
     TInfoBateau * mBateauxMachine;
          /*tableau dynamique de mNombreInstanceBateaux*K_NBTYPEBATEAUX bateaux de la machine*/
}Tparam;
```

```
/* On va supposer que les bateaux du joueur sont numérotés de 0 à mNombreInstanceBateaux*K_NBTYPEBATEAUX
-1. Ce numéro correspond à idBateau de la classe métier Bateaux. L'idBateau sera l'indice dans le tableau
mBateauxDuJoueur.
Les bateaux de la machine sont numérotés de mNombreInstanceBateaux*K_NBTYPEBATEAUX à (2*
mNombreInstanceBateaux*K_NBTYPEBATEAUX) -1. De même ce numéro correspond à idBateau. Pour calculer
l'indice dans le tableau mBateauxMachine du bateau d'identifiant idBateau on utilise la formule idBateau -
mNombreInstanceBateaux*K_NBTYPEBATEAUX */

//*****************************
/*      N : getInfoBateau
        D : donne les information sur le pNum eme bateau des parametres de la partie
        E : pNum , pParam
        S :*
        R : les informations sur le bateau
        Prec : le pNum eme bateau existe
*/
TInfoBateau *getInfoBateau(int pNum ,const Tparam * pParam);

//*****************************
/*      N : getNBInstances
        D : donne le nombre d'instances de chaque bateau pour un joueur pour les parametres de partie pParam
        E : pParam
        S :
        R : ce nombre
        Prec : -
*/
int getNBInstances (const Tparam * pParam);

//*****************************
/*      N : chargerParam
        D : lit les paramètres de la partie dans un descripteur de fichier pDesc
        E :pDesc
        S : pP
        R :
        Prec : pDesc est un descriteur de fichier ouvert en lecture placé sur la lecture des paramètres de la partie
*/
void chargerParam(FILE * pDesc, Tparam *pP);

//*****************************
/*      N : memParam
        D : sauve les paramètres de la partie pParam dans un fichier de nom pNom
        E :pNom , pParam
        S :
        R :
        Prec : pDesc est un descriteur de fichier ouvert en écriture
*/
void memParam(const Tparam * pParam, FILE * pDesc);

//*****************************
/*      N : newTParam
        D : creer un Tparam pour pNbInstances de chaque bateau pour chacun : remarque devra allouer les tableaux dynamiques
        E :pNbInstances,
        S : pP
        R :
        Prec : -
*/
void newTParam(int pNbInstances , Tparam * pP);
#endif
```

Les saisies des informations pour remplir les tableaux doivent être faites par la vue. Le contrôleur va donc demander à la vue d'effectuer les saisies puis

appeler les fonctions du module parametre pour initialiser le nième bateau de la liste des paramètres etc => on passe à la vue.
De la même façon les affichages sont faits par la vue.

//fichier vueparam.h

```
#include "parametre.h"
// pour les couleurs
//*****************************
/*        N : afficherCouleur
          D :  affiche le nom de la couleur d'index i
          E : pB
          S :
          R :  -
          Prec : -
*/
void afficherCouleur(int pI);


// pour les types
//*****************************
/*        N : afficherType
          D :  affiche le nom de la type de bateau d'index i
          E : pB
          S :
          R :  -
          Prec : -
*/
void afficherType(int pI);

// Pour le information sur un bateau
//*****************************
/*        N : afficherInfoBateau
          D :  affiche les  Informations sur à l'info bateau pB
          E : pB
          S :
          R :  -
          Prec : -
*/
void afficherInfoBateau(const TInfoBateau *pB);

//*****************************
/*        N : afficherBateau
          D :  affiche les  Informations sur un bateau pB
          E : pB, pParam
          S :
          R :  -
          Prec : -
*/
void afficherBateau(const TBateau *pB,const Tparam * pParam);

//*****************************
/*        N : afficherParam
          D :  affiche les  param
          E : param
          S :
          R :  -
          Prec : -
*/
void afficherParam(const Tparam * pParam);

//*****************************
/*        N : saisirInfoBateau
          D : ...
          E :
```

```
          S : ...
          R :  -
          Prec : -
*/
void saisirInfoBateau(TInfoBateau *pB);

//*****************************
/*        N : setIemeInfoBateauTParam
          D : COMPLETER
          E   pIdBateau   pNom[] pCouleur   pType
          S : pP
          R :
          Prec : ne peut etre appelé qu'apres newTParam
*/
void setIemeInfoBateauTParam(int pIdBateau , Tparam * pP, const char pNom[], int pCouleur , EType pType);
```

…

Semaine 3-4 :

Jouer, un coup, des coups, annuler, tests et jeu d'essai

Semaine 5 :

Calcul du score, affichage du gagnant, sauvegarde de la partie, des meilleurs scores.

Remarque : lorsqu'on sauvegarde une partie il faut sauver les paramètres, les joueurs, la pile des coups... et pouvoir reprendre la partie, annuler des coups ...

Semaine 6 :

Les derniers tests et jeu d'essai

Semaine 7 :

Le dossier (voir le contenu attendu)

Semaine 8 :

Validation