# Introduction à la Programmation

—

# Examen partiel sur machine

—

G1: monasse*(at)*imagine.enpc.fr G2: nicolas.audebert*(at)*onera.fr
G3: alexandre.boulch*(at)*enpc.fr G4: maxime.ferrera*(at)*onera.fr
G5: laurent.bulteau*(at)*u-pem.fr G6: pierre-alain.langlois*(at)*eleves.enpc.fr

—

09/11/18

—

## 1 Enoncé

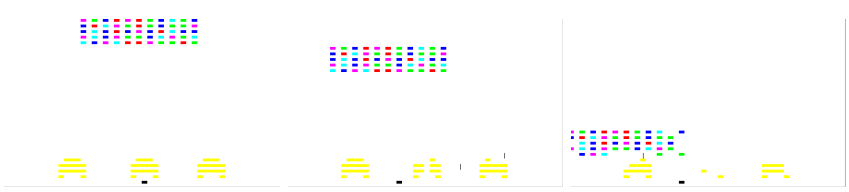
### 1.1 Space Invaders

Journal *Le Monde* du 6 novembre 2018 : "L'hypothèse de deux astronomes : une « sonde extraterrestre » existe dans le système solaire". Les intentions de ces extraterrestres pouvant être belliqueuses, préparons-nous au pire en nous entraînant au jeu classique *Space Invaders*, que nous allons reprogrammer ! Créez un projet Imagine++ avec un fichier contenant le `main`. Prenez le `CMakeLists.txt` d'un projet existant et adaptez-le.

**Il est plus important de livrer un code clair (commenté et indenté) et qui compile sans warning, même s'il ne répond pas à toutes les questions. Pour cela, vérifiez à chaque étape que votre programme compile et se lance correctement. À la fin de l'examen, nettoyez votre code (indentation...) et vérifiez qu'il compile. Créez alors une archive portant votre nom et numéro de groupe.**

### 1.2 Une armada

1. Chaque objet sera dans une grille de cases de taille $50 \times 30$ (utiliser des constantes `largeur` et `hauteur` pour ces dimensions), chaque case occupant $10 \times 10$ pixels (constante `zoom`$= 10$). L'armada part de la hauteur 0 et le défenseur est en ligne `hauteur`$-1$. Votre fonction `main` ouvre une fenêtre de la taille adéquate et appelle `endGraphics` en fin de programme.
2. Dans des fichiers à part, définir une structure `Vaisseau` comportant une position (coordonnées de case), une couleur, et un bool indiquant si le vaisseau est "vivant".
3. Définir des fonctions d'affichage (s'il est vivant) et d'effaçage du `Vaisseau`, un rectangle de taille `zoom`/2 en hauteur.



```
void Clavier(int& dx, bool& tir) {
  dx = 0; tir=false;
  Event e;
  do {
    getEvent(0, e);
    if(e.type!=EVT_KEY_ON) continue;
    if(e.key=='s'||e.key=='S') dx=-1;
    if(e.key=='d'||e.key=='D') dx=+1;
    if(e.key==' ') tir=true;
  } while(e.type!=0 && dx==0 && !tir);
}
```

FIGURE 1 – Position initaile du jeu avec l'armada, le capitaine et les trois bunkers, puis en cours de jeu avec des bunkers endommagés et deux tirs ennemis en route, enfin l'armada dangereusement proche de réussir son invasion. À droite, le code de la fonction d'interaction avec les touches du clavier.

4. Définir une fonction `deplace` faisant bouger le vaisseau de paramètres `dx` et `dy` de la fonction.
5. Dans des fichiers à part, définir une structure `Armada` stockant un nombre variable de vaisseaux, et un bool indiquant si l'armada bouge vers la droite ou la gauche. Limiter le nombre maximum de vaisseaux à 100.
6. Définir une fonction d'ajout d'un vaisseau à une armada, vérifiant par `assert` qu'on ne déborde pas.
7. Définir les fonction d'affichage et d'effaçage de l'armada.
8. Définir une fonction `bornes` calculant pour une armada les abscisses les plus à gauche, droite et l'ordonnée la plus basse de ses vaisseaux vivants.
9. Définir une fonction `deplace` pour une armada. Les vaisseaux bougent tous d'un case vers la droite ou la gauche (contrôlé par la variable dans `Armada`), sauf quand cela ferait sortir un vaisseau (vérifier avec la fonction précédente), auquel cas l'armada bouge d'une case vers le bas et la direction de déplacement sera inversée (gauche↔droite). Cette fonction indique si la partie continue, c'est-à-dire que l'armada n'atteint pas une case à `hauteur`$-1$.
10. Dans votre `main.cpp`, créer une fonction `forme_armada` qui crée une armada de 11 vaisseaux en largeur, 5 en hauteur, partant du centre haut de la fenêtre. Les vaisseaux sont espacés de deux cases en largeur. Attribuer à chacun une couleur aléatoire parmi 5 préétablies. À ce stade, vérifiez que votre armada peut procéder à son invasion tranquillement, s'arrêtant quand elle atteint le bas de l'écran.

### 1.3 Gestion des tirs

11. Dans des fichiers à part, définir une structure `Tir`, stockant une position (une case) et une direction `dy`$= \pm 1$, car certains tirs vont vers le haut (défense), d'autres vers le bas (envahisseurs).
12. Définir les fonctions d'affichage (trait vertical centré en largeur dans la case), effaçage et déplacement d'un tir.
13. Un tir est dit actif si sa position est dans la grille et que sa direction est non nulle, écrire une fonction testant cela.
14. Écrire une fonction `explosion` testant si un tir rencontre un vaisseau vivant. Dans ce cas, elle rend le tir inactif et tue le vaisseau. Elle indique si l'explosion a eu lieu.
15. Idem pour une armada, la fonction correspondante indique si l'un de ses vaisseaux a explosé.
16. Le `main` crée un tir (celui du capitaine défenseur, vous !) partant du bas vers le haut. À chaque étape de l'animation, le tir se déplace, on teste l'impact éventuel avec l'armada (réafficher dans ce cas), puis l'armada se déplace et on reteste l'impact (réafficher si nécessaire).
17. Votre `main` crée un capitaine (vaisseau noir) centré en dernière rangée.
18. Définir une fonction `gere_capitaine`, qui obéit aux ordres de l'utilisateur : le capitaine peut se déplacer horizontalement avec les touches `s` et `d` sans sortir de la grille, et peut tirer avec la touche espace. Cependant, il ne peut tirer que lorsque son tir précédent est devenu inactif (explosion ou sortie de grille). Utiliser la fonction `Clavier` de l'illustration.
19. Votre animation permet de contrôler le capitaine. Chaque étape pourra appeler deux fois `gere_capitaine`, de manière à avoir un avantage de vitesse sur l'envahisseur.

### 1.4 Le jeu final

20. Vous pouvez créer 3 bunkers, qui sont simplement des armadas fixes, constitués chacun d'une rangée de 3 vaisseaux, suivie de deux rangées de 5, et de deux vaisseaux aux extrémités en dernière rangée, tous jaunes. Gérer l'impact du tir sur un bunker.
21. L'armada est susceptible de tirer au maximum `NB_TIRS`$= 2$ tirs simultanément. Stocker ces tirs dans `Armada` et n'oubliez pas d'initialiser ces tirs comme inactifs pour l'armada et pour les bunkers.
22. Faire en sorte que la fonction `deplace` pour l'armada fait aussi évoluer ses tirs.
23. La boucle du jeu vérifie les impacts des tirs de l'armada sur les bunkers et sur le capitaine. Le jeu est perdu dans ce dernier cas.
24. Chaque tir inactif lors de l'appel de `deplace` sur l'armada peut donner lieu à un nouveau tir avec une probabilité de $1/5$. Si c'est le cas, c'est un des vaisseaux vivants les plus en bas de chaque colonne de cases qui tire. Trouver l'indice du vaisseau le plus en bas de chaque colonne, compter leur nombre, et en choisir un au hasard comme tireur.