



## **HLIN302 – Projet noté « Casse-Briques »**

**Programmation impérative avancée**  
**Alban MANCHERON et Pascal GIORGI**

---

L'objectif de ce projet est la réalisation d'un « Casse Briques » en C++, permettant à un utilisateur (humain) de jouer dans un terminal Unix.



Un casse-briques se présente sous la forme d'un terrain (généralement une zone rectangulaire) sur lequel sont placées des briques (représentées par des rectangles), une raquette (symbolisée par une barre horizontale) et une balle. Les briques sont positionnées dans la partie haute du terrain et la raquette est positionnée en bas du terrain. Celle-ci peut être déplacée latéralement (par l'utilisateur), sans toutefois jamais sortir du terrain. La balle est initialement accolée sur le dessus de la raquette et une fois lancée (action de l'utilisateur), elle se déplace en ligne droite jusqu'à rencontrer un obstacle (bords gauche, supérieur et droit du terrain, une brique ou la raquette). La balle rebondit alors, ce qui change son orientation (et éventuellement sa vitesse). Si la balle rebondit sur une brique, la brique est abîmée et finit par se désagréger (disparaître). Si la balle arrive au bord inférieur du terrain, alors celle-ci sort du terrain et est perdue.

L'objectif est donc de détruire toutes les briques, et cela avec un nombre de balles fini. Si le joueur n'a plus de balle, la partie est perdue. Lorsqu'il n'y a plus de brique sur un terrain, alors un nouveau terrain est proposé (qui peut-être plus compliqué – ou pas). Le score du joueur est calculé par rapport au nombre de briques démolies sur l'ensemble de la partie.

Avant de définir les attentes pédagogiques, il convient donc de présenter le...

## **1 Cahier des charges**

Afin de pouvoir jouer, il faut donc visualiser le terrain de jeu, ainsi qu'une zone de dialogue permettant d'afficher des messages et instructions à l'attention du joueur humain.

Cela implique donc dans un premier temps, de définir les caractéristiques du terrain (dimensions, forme particulière, ...). Les briques peuvent être de formes différentes et plus ou moins résistantes (certaines briques disparaissent au premier choc de la balle, d'autres peuvent se désagréger par partie ou bien après un nombre fini de chocs). Bien évidemment, plus la brique est difficile à faire disparaître, plus elle permet de rapporter des points.

Le positionnement des briques peut-être basique (*e.g.* : 3 lignes de briques de résistance 1 espacées régulièrement d'une valeur fixe) mais également plus sophistiqué (*e.g.* : 1 ligne de 5 briques triangulaires espacées

régulièrement avec une résistance de 3, suivie de 2 lignes de briques creuses qui se fragmentent en 4 sous-briques [les bords] espacées aléatoirement d'une valeur comprise dans un intervalle donné, suivi d'1 ligne de 3 briques simples de résistance 1 espacées d'une valeur fixe). Le placement peut également correspondre à une description plus bas niveau (*e.g.* : placer une brique triangulaire de résistance 2 à la position  $27 \times 19$ ).

Le placement d'un (ou plusieurs) terrain(s) donné(s) doit donc être défini dans un fichier de configuration dont le format pourrait ressembler à l'exemple donné dans le fichier 1.





Dans l'hypothèse où le fichier de configuration comporterait des erreurs, il convient de les signaler à l'utilisateur le plus explicitement possible.

De même, si plusieurs fichiers de configuration sont lus, alors les informations qu'ils contiennent se combinent. Il est ainsi possible d'imaginer de faire un fichier pour les formes de briques, un pour les niveaux, voire un par forme et un par niveau. . . Le cas échéant, l'ordre de lecture des fichiers aura un impact important.

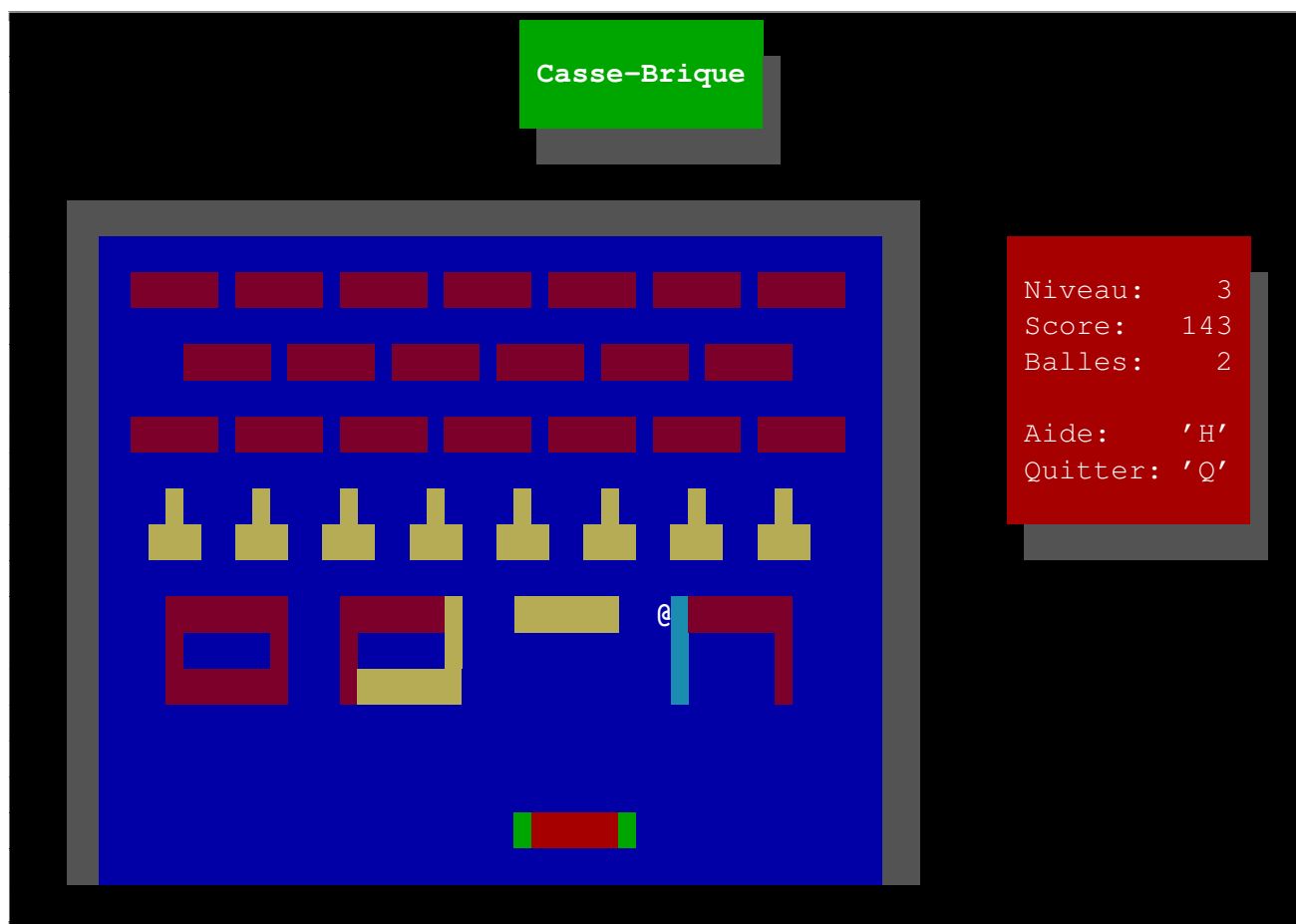
Enfin, un fichier, bien que valide, peut tout à fait ne pas pouvoir aboutir à un placement correct (nombre de lignes trop important par rapport à la dimension du terrain, utilisation de niveaux combinés incompatibles (notamment avec les placements en position absolue, . . .)). Dans ce cas, il faut également pouvoir notifier l'utilisateur de la difficulté rencontrée.

L'exemple 1 vous donne une idée de ce à quoi peut ressembler votre interface.

L'utilisateur doit pouvoir :

- lancer la balle à chaque début de partie (et à chaque nouvelle balle) en appuyant sur la touche espace de son clavier : ;
- déplacer sa raquette sur le terrain en utilisant les flèches de son clavier :  et ;
- Quitter la partie en cours en appuyant sur la touche  de son clavier.

### Exemple 1 (Exemple d'écran d'une partie)



Une fois l'initialisation du terrain réalisée, la partie *stricto sensu* peut débuter.

Le joueur peut déplacer sa raquette longitudinalement avec les flèches gauches et droites de son clavier et lâcher la balle en appuyant sur la touche espace. La balle doit partir nécessairement vers le haut (l'inclinaison peut être aléatoire ou bien dépendre des précédents mouvements de la raquette, à l'instar de sa vitesse).

Une cycle de calcul consiste à calculer la prochaine position de la balle s'il n'y a pas d'obstacle. Si cette position est voisine d'un (ou plusieurs) obstacle(s), alors il faut recalculer l'orientation et la vitesse de la balle pour le cycle d'après. Si parmi les obstacles se trouve une brique, il faut abaisser le niveau de résistance du bloc considéré de 1 unité (dans la vue, il est possible d'associer un code couleur dépendant de la résistance du bloc courant). Si la résistance est nulle, alors la brique (ou le composant de la brique) doit tout simplement disparaître du terrain au prochain cycle (il est également possible de représenter le contact avec un code couleur dans la vue). Il faut également pouvoir déterminer à tout moment combien de briques sont encore placées sur le terrain. S'il n'y a plus de brique, il faut passer au niveau suivant (et éventuellement accorder un bonus de points au joueur).

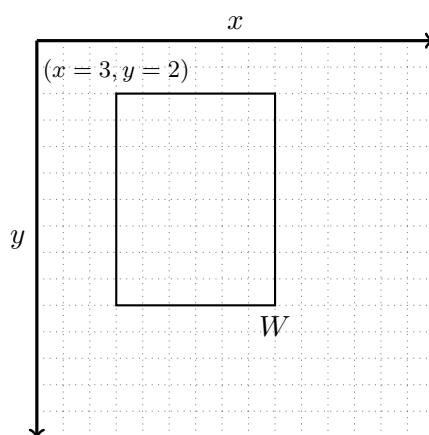
Si la balle sort du terrain, alors selon qu'il reste des balles ou pas, il faut réinitialiser la position de la balle par rapport à la raquette (ainsi que sa direction et sa vitesse) ou bien afficher un message à l'utilisateur lui indiquant le niveau atteint, ainsi que son score (et toute statistique que vous jugerez utile).

## 2 Mise en œuvre

Afin de réaliser l'affichage graphique ainsi que l'interaction homme-machine, vous devrez vous appuyer sur une bibliothèque permettant la manipulation de votre terminal. Vous devrez également modéliser les différents composants du jeu (terrain, raquette, balle, briques, niveaux, joueur, ...)

### Gestion de l'interface

La bibliothèque permettant de manipuler votre terminal et les interactions avec le clavier se nomme `ncurses`<sup>1</sup> et elle est fournie librement dans toutes les distributions Unix. Pour utiliser cette bibliothèque, il vous suffit d'inclure le fichier `ncurses.h` et compiler votre programme en ajoutant l'option `-lncurses`. Comme la prise en main de cette bibliothèque n'est pas évidente de prime abord, nous vous fournissons une classe C++ permettant une gestion simplifiée. La classe `Window` définit des objet permettant de gérer des fenêtres dans votre terminal. Par exemple, si votre terminal est de dimension  $15 \times 15$ , la déclaration `Window W(3, 2, 6, 8)` permet de définir une fenêtre de taille  $6 \times 8$  dont le coin supérieur gauche est positionné à la coordonnée  $(x = 3, y = 2)$ . Attention, la coordonnée du coin supérieur gauche de votre terminal est à la position  $(x = 0, y = 0)$  et l'axe des  $x$  est orienté vers la droite alors que l'axe des  $y$  est orienté vers le bas. La figure ci-dessous illustre la construction de notre exemple.



À partir d'un objet `Window` il est possible d'écrire du texte ou un caractère à une position donnée dans la fenêtre. Pour cela, il faut utiliser la méthodes `print(x, y, s)` où  $(x, y)$  représente la position où l'on souhaite écrire dans la fenêtre (la position  $(0, 0)$  correspond au coin supérieur gauche de la fenêtre) et  $s$  représente le caractère ou la chaîne de caractère que l'on veut afficher.

1. <https://fr.wikipedia.org/wiki/Ncurses>

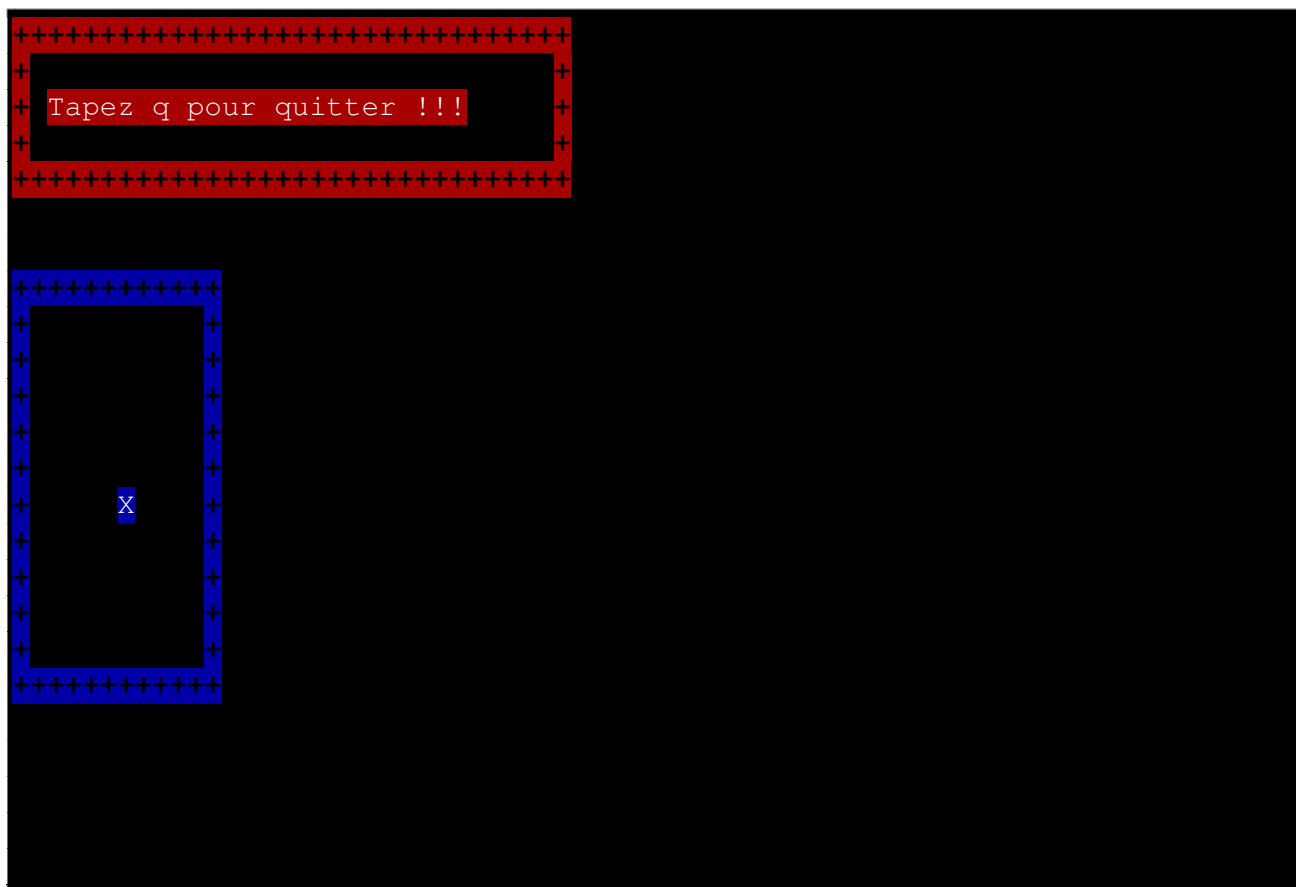
La fenêtre utilise un système d’affichage de couleur : une couleur de texte et une couleur de fond. Il est possible de changer cette couleur via la méthode `setCouleurFenetre`. De même, on peut changer la couleur de la bordure de la fenêtre via la méthode `setCouleurBordure`. On peut également spécifier une couleur `c` à la méthode `print(x, y, s, c)` pour écrire dans une couleur choisie. Enfin, on peut supprimer l’ensemble des textes écrits dans une fenêtre en utilisant la méthode `clear`. Le code de la classe `Window` est disponible dans l’espace projet sur l’ENT/Moodle du cours : fichiers `window.h` et `window.cpp`. Le fichier *header* comporte des commentaires vous permettant d’utiliser correctement cette classe et les couleurs d’affichage. Si vous le souhaitez, vous pouvez améliorer cette classe pour qu’elle gère plus d’options d’affichage de la bibliothèque `ncurses`.



Pour que votre programme fonctionne correctement avec la classe `Window` il faut que votre programme ressemble à celui-ci :

```
1 #include "window.h"
2
3 int main(int argc, char** argv) {
4     startProgramX();
5     myprogram();
6     stopProgramX();
7     return 0;
8 }
```

Vous devrez remplacer l’appel à la fonction `myprogram()` par la fonction principale de votre jeu. Vous trouverez également un exemple de programme utilisant la classe `Window` dans l’espace projet sur l’ENT/Moodle du cours. Rappel, pour compiler il vous faudra ajouter l’option `-lncurses`. Ce programme vous montrera également comment vous pouvez interagir avec les actions au clavier. Voici un rendu graphique du programme :



Pour la réalisation de votre programme, vous devez impérativement utiliser la classe `Window`. Si vous souhaitez utiliser d’autres possibilités de la bibliothèque `ncurses`, il vous faudra les intégrer dans la classe `Window`.

## Modélisation du programme

Vous devez vous appuyer sur les concepts vus en cours, TD et TP. Le langage étant imposé (C++ pour ceux qui auraient oublié), il semble évident qu'il faut mettre en œuvre les paradigmes du langage (modélisation objet, polymorphisme [dont la surcharge d'opérateurs], allocation dynamique, ...).

Il vous faudra donc fournir plusieurs classes permettant de modéliser le jeu.

## 3 Travail à réaliser

Le projet consiste donc à implémenter un programme complet permettant de jouer au casse-brique en C++ en utilisant les concepts abordés pendant l'ensemble des cours, TD, TP.

Pour mener ce projet à bien, il vous faudra bien évidemment définir les étapes clés du projet. Mais également appréhender des concepts qui ne relèvent pas directement de cette UE, comme l'utilisation d'une librairie tiers, la gestion des codes d'échappement permettant de modifier les couleurs d'avant-plan et d'arrière-plan d'un terminal<sup>2</sup>.

Le travail à réaliser devra correspondre à plusieurs étapes. Chaque étape correspond à une logique de conception, avec des objectifs spécifiques et des niveaux de difficulté variés. Plus les étapes seront réalisées correctement, meilleure sera la note finale du projet.

**(Étape 1)** Le jeu doit être jouable avec les caractéristiques suivantes :

- chaque composant du jeu doit être modélisé et intégré dans une classe : brique, niveau, terrain, joueur, raquette, balle, fenêtre de jeu, message, menu, score, ... Aucune variable globale ne doit être utilisée ;
- le jeu permet le passage de paramètres au lancement du programme pour connaître l'aide, la version du jeu et le nom des auteurs ;
- le positionnement des briques doit respecter les règles de non chevauchement et de bordure ;
- tous les messages doivent être fonctionnels : choix d'un niveau, affichage des scores, aide du jeu, fin de partie, erreurs dans les fichiers de configuration ou problème de placement des briques, ...

**(Étape 2)** Les options suivantes doivent être ajoutées au jeu :

- on doit pouvoir passer un ou plusieurs fichiers de configuration au programme qui décrit *a minima* l'ensemble des niveaux du jeu. Bien évidemment, pouvoir spécifier la dimension du terrain de jeu, modifier les codes couleurs, la représentation de la balle ou de la raquette est un plus. Le formalisme proposé dans le fichier 1 n'est pas imposé, vous avez toute latitude pour définir la syntaxe utilisée, toutefois, vous veillerez à ce que cette configuration soit compréhensible par l'utilisateur. Le fichier de configuration doit pouvoir permettre de définir plusieurs niveaux, composés de briques rectangulaires (de tailles paramétrables) placées soit par coordonnées explicites ou bien de manière automatique.
- on doit pouvoir sauvegarder les 5 meilleurs scores dans un fichier et les afficher dans le jeu. Il faudra prévoir la saisie des noms des joueurs ;
- lors du choix de position pour une brique, il ne doit pas être possible de se positionner sur une brique déjà existante ou bien en dehors du terrain. Il peut également être judicieux de contraindre les briques à être à une distance minimale fixée de la ligne de déplacement de la raquette.

**(Étape 3)** Les options avancées suivantes doivent être possibles :

- il est possible de sauvegarder une partie (dans un fichier) et de la reprendre plus tard ;
- l'ensemble des actions du jeu doivent être enregistrées dans un fichier journal (fichier *log*) permettant le suivi exact de l'évolution de la partie. Le nom du fichier journal pourra être spécifié par une option de la ligne de commande ;

---

2. Vous trouverez les informations nécessaires à l'adresse [http://en.wikipedia.org/wiki/ANSI\\_escape\\_code](http://en.wikipedia.org/wiki/ANSI_escape_code).

**(Étape 4)** Les options suivantes seront nécessaires pour obtenir la note maximale :

- Le fichier de configuration doit permettre de définir des formes de briques différentes et des compositions de briques.
- Le fichier de configuration doit permettre de définir des combinaisons de placements.

## Le bon sens en action

*A minima*, il vous faut implémenter les demandes exprimées dans les quatre étapes précisées dans le cahier des charges<sup>3</sup>.

Vous veillerez à coder proprement (respect des normes ANSI, commentaires utiles, indentation cohérente, fichiers structurés<sup>4</sup>, ...).



Vous veillerez également à réaliser des tests unitaires pour chaque classe que vous aurez développée.

Vous ne devez en aucun cas recopier du code déjà tout prêt. Toute utilisation de concepts non abordés en cours (*e.g.* : héritage, polymorphisme non paramétrique) ou code non C++ (*e.g.* : code C pur) est proscrit et entraînera la nullité du projet : note de 0/20.

Vous avez le droit d'utiliser la bibliothèque STL à l'exception de la classe `vector`. Bien entendu, il est proscrit d'utiliser tout autre structure de données de la STL pour remplacer la classe `vector`. Il vous sera donc indispensable d'utiliser des tableaux dynamiques pour remplacer la classe `vector`. Bien évidemment, vous pourrez encapsuler vos tableaux dynamiques dans une classe pour simplifier leurs utilisations.

## Boni

Le cahier des charges peut toujours être complété si vous en avez le temps, l'envie et la compétence (il va de soi que cela permet de gagner des points en plus, mais ne compensera **en aucune manière** un manquement au cahier des charges).

Par exemple, la librairie `ncurses` permet d'interagir avec la souris. Rien ne vous interdit d'explorer ces aspects afin de rendre l'interaction avec le terminal accessible à vos proches. Il est aussi envisageable de laisser la possibilité à l'utilisateur de modifier les touches du clavier permettant de contrôler l'interface *via* un menu « paramètres ». Ainsi un joueur chevronné pourra définir la touche  pour déplacer la raquette vers la gauche, et la touche  pour se déplacer à droite...

Et si vous souhaitez vraiment dépasser le cadre de l'UE, vous pouvez également implémenter une vue en utilisant une librairie graphique plus élaborée (*e.g.* : `wxWidgets`, `Qt`, ...) du moment que l'interface `ncurses` est également implémentée.

Ceci nous amène le plus naturellement du monde à aborder les aspects liés à votre...

## 4 Organisation



Concernant votre organisation, le travail est à réaliser en groupes de **4 étudiants**. Vous devrez saisir la composition de votre groupe de travail sur l'espace projet de l'ENT/Moodle du site du cours au plus tard le 31 octobre 2018 à 23h59. Les retardataires verront leur note de projet pénalisée de 2 points.

Enfin, vous devrez rendre sur l'ENT – au plus tard le 6 janvier 2019 à 23h59 – une archive compressée (au format **tar.gz** ou **tar.bz2** uniquement) contenant un répertoire portant le nom de votre groupe (*e.g.* : `groupe-12`) et dans lequel on trouvera<sup>5</sup> :

- les sources de votre programme ;
- un fichier texte<sup>6</sup> mentionnant vos noms et prénoms ;

3. Cela va sans dire, mais c'est toujours mieux en le disant...

4. La règle une classe = 1 .h + 1 .cpp est raisonnable.

5. Le non-respect de ces consignes sera indubitablement sanctionné.

6. Typiquement, un fichier `Auteurs.txt`.

- un fichier texte<sup>7</sup> expliquant comment compiler (en ligne de commande) et utiliser votre programme<sup>8</sup> ;
- un rapport<sup>9</sup> au format **pdf** (10 pages maximum, annexes comprises, hors pages liminaires, police 11 points pour le corps, marges à 2cm) mentionnant vos noms et prénoms et décrivant votre analyse, vos choix, vos tests, les remarques et commentaires, ... Vous discuterez notamment dans ce rapport de l'algorithme que vous aurez défini pour calculer la direction et la vitesse de la balle ainsi que du formalisme que vous aurez choisi pour votre fichier de configuration (notamment pour représenter les briques et les placements). Vous mettrez également en avant les différentes étapes que vous avez réalisées et leur degré d'achèvement.

Et comme tout jeu, vous vous devez d'y jouer pour comprendre ce que vous aurez à implémenter. Pour cela, vous pouvez par exemple tester *LBreakOut* (<http://lgames.sourceforge.net/LBreakout2/>).

*Bon Courage. . .*

---

7. Typiquement, un fichier `LisezMoi.txt`.

8. L'utilisation d'un `Makefile` est fortement conseillée.

9. Le rapport devra être écrit en français et respecter au mieux les règles typographiques françaises.

## Fichier 1 – Exemple de fichier de configuration

```

1 # Un fichier de config peut (et doit) contenir des commentaires.
2 # un commentaire débute par le symbole '#' et se termine à la fin de la ligne.
3 # Bien évidemment, les lignes vides sont tout simplement ignorées
4
5 # On définit une "brique-de-base" comme étant un rectangle de hauteur 1 et de largeur 2.
6 # Sa résistance sera par défaut de 1 en l'absence de spécification.
7 [Brick brique-de-base]
8     Shape: 1x2 # rectangle de dimension 1 (hauteur) par 2 (largeur)
9 [End Brick]
10
11 # On définit une "brique-triangle" comme étant la superposition d'une brique
12 # rectangulaire de 1 par 3, puis d'une brique rectangulaire de 1 par 1, avec une
13 # résistance par défaut de 2.
14 [Brick brique-triangle]
15     Hardness: 2
16     Shape: " # \n###" # le symbole '#' a été pris "au hasard", c'est un symbole
17                     # visible autre que l'espace.
18 [End Brick]
19
20 # On définit une "brique-creuse" qui se fragmentent en 4 sous-briques (les
21 # bords) et dont chaque morceau a une résistance par défaut de 2.
22 [Brick brique-creuse]
23     Hardness: 2
24     Shape: "LTTTR\nL    R\nLBBBR" # Ici, il y a 4 types distincts composant la brique
25                                     # ('L', 'T', 'B', 'R'). Lorsqu'un morceau de type 'L' est
26                                     # touché par la balle, tous les morceaux de type de la
27                                     # brique 'L' perdent 1 point de résistance.
28 [End Brick]
29
30 # 3 lignes de briques de résistance 1 espacées régulièrement d'une valeur fixe
31 [Level Niveau-1]
32     Brick: brique-de-base # Type de brique à utiliser
33     NbBricks: auto # essaie de positionner le plus de briques possibles
34     NbLines: 3 # Nombre de lignes
35     Padding: 1 # Espacement entre les briques
36 [End Level]
37
38 # 1 ligne de 5 briques triangulaires espacées régulièrement avec une résistance de 3
39 [Level Niveau-2-a]
40     Brick: brique-triangle{3} # brique-triangle mais avec une résistance de 3 (par défaut,
41                             # la brique-triangle a été définie avec une résistance de 2.
42     NbBricks: 5 # Nombre de briques par ligne
43     NbLines: 1 # Nombre de lignes
44     Padding: auto # Espacement entre les briques
45 [End Level]
46
47 # 2 lignes de briques creuses espacées aléatoirement de 1 à 3 unités.
48 [Level Niveau-2-b]
49     Brick: brique-creuse
50     NbBricks: auto # Nombre de briques par ligne
51     NbLines: 2 # Nombre de lignes
52     Padding: rand(1,3) # Espacement entre les briques
53 [End Level]
54
55 # 1 ligne de 3 briques simples de résistance 1 espacées d'une valeur fixe
56 [Level Niveau-2-c]
57     Brick: brique-de-base # brique de base
58     NbBricks: 3 # Nombre de briques par ligne
59     NbLines: 1 # Nombre de lignes
60     Padding: 2 # Espacement entre les briques
61 [End Level]
62
63 # Niveau combiné:
64 [Level Niveau-2]
65     Use: Niveau-2-a
66     Use: Niveau-2-b
67     Use: Niveau-2-c
68 [End Level]
69
70 # 1 brique triangulaire de résistance 2 à la position 27x19.
71 [Level Niveau-2-c]
72     Brick: brique-triangle # Type de brique
73     NbBricks: 1 # Nombre de briques par ligne
74     NbLines: 1 # Nombre de lignes
75     StartPosition: 27x19
76 [End Level]

```