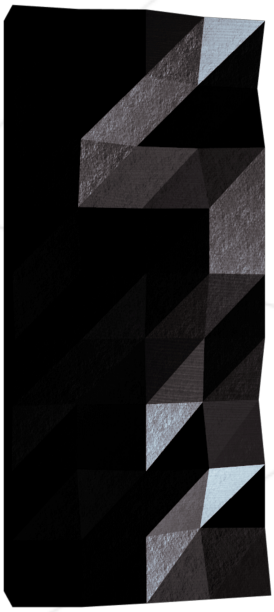

Filler

Like Fillit, but Filler

42 Staff pedago@staff.42.fr

Résumé: Créez votre programme joueur pour affronter d'autres étudiants sur le célèbre (ou pas) plateau du Filler. Le principe est simple : deux joueurs s'affrontent sur un plateau, et doivent placer, tour à tour, la pièce que le maître du jeu (fourni sous la forme d'un exécutable Ruby) leur donne, gagnant ainsi des points. La partie s'arrête dès qu'une pièce ne peut plus être placée. Petit projet ludique !

Table des matières

Chapitre I

Préambule

FIGURE I.1 – Extra Fabulous Comics.

Chapitre II

Introduction

`Filler` est un jeu algorithmique qui consiste à remplir une grille d'une taille connue à l'avance avec des pièces de taille et de formes aléatoires, sans que les pièces ne se superposent de plus d'une seule case et sans qu'elles ne depassent de la grille. Si l'une de ces conditions n'est pas remplie, la partie s'arrête.

Chaque pièce posée avec succès rapporte un certain nombre de points, et a un seul joueur, le but du jeu pourrait être d'obtenir le meilleur score possible. Cepedant, c'est avec deux joueurs que le filler prend tout son intérêt. Chaque joueur a pour but de poser un maximum de pièces tout en tentant d'empêcher son adversaire de le faire. A la fin de la partie, celui avec le plus de point remporte le match...

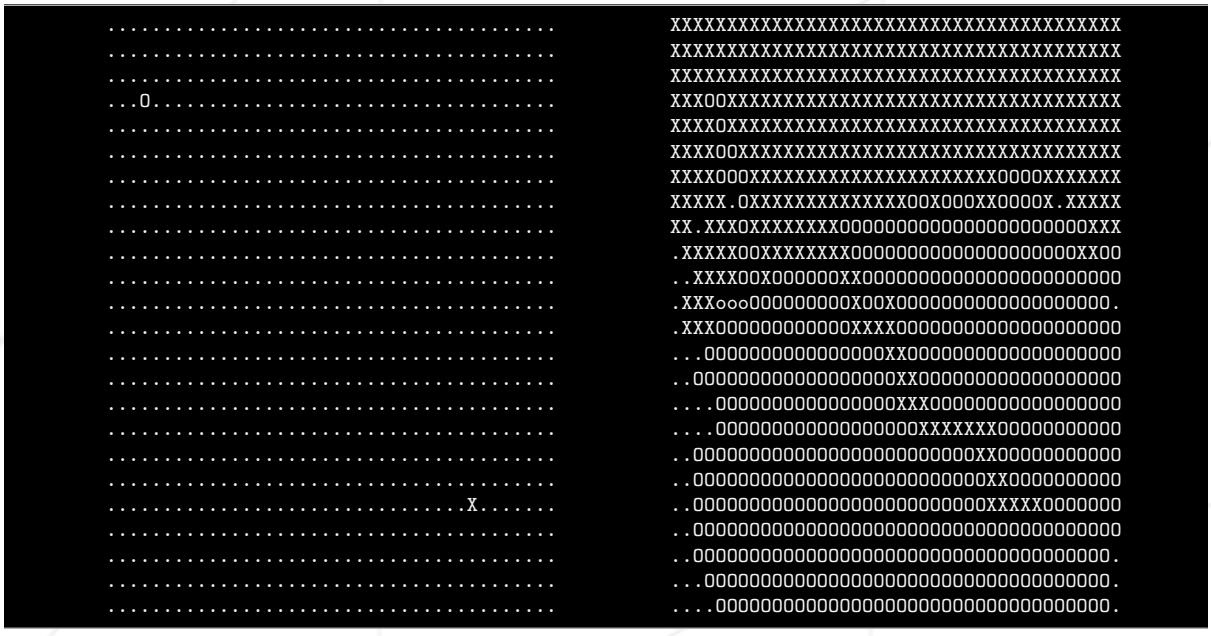

FIGURE II.1 – A gauche l'état initial d'une grille et à droite le résultat de l'affrontement de deux joueurs.

Chapitre III

Objectifs

Ecrire un joueur de `Filler` est un défi algoritmique très intéressant. A chaque tour, le joueur actif reçoit l'état de la grille et doit maximiser ses points tout en tentant de minimiser ceux de l'adversaire en l'éliminant le plus vite possible.

Les objectifs de ce projet rassemblent toujours les objectifs usuels des projets de début de cursus : rigueur, pratique du `C` et pratique d'algorithmes élémentaires. Mais à la différence d'un jeu plus simple comme le `Fillit`, il ne s'agit plus de simplement agencer ses pièces le plus efficassement possible, mais maintenant d'empêcher son adversaire de le faire ! Il vous faudra donc créer votre propre algorithme de remplissage pour contrer l'algorithme ennemi.

Et il faudra également apprivoiser la VM fournie avec ce sujet...

Chapitre IV

Consignes générales

- Ce projet ne sera corrigé que par des humains. Vous êtes donc libres d'organiser et de nommer vos fichiers comme vous le désirez, en respectant néanmoins les contraintes listées içi.
- Vous devez rendre, à la racine de votre dépôt de rendu, un fichier `auteur` contenant votre login suivi d'un retour à la ligne :

```
$>cat -e auteur
xlogin$
$>
```

- L'exécutable doit s'appeller `<login>.filler`. Evitez d'être bête et remplacez le placeholder `<login>` par votre login...
- Cet exécutable doit se trouver à la racine du dépôt après la compilation.
- Vous devez rendre un Makefile.
- Votre Makefile doit compiler le projet, et doit contenir les règles habituelles. Il ne doit recompiler et/ou relinker le programme qu'en cas de nécessité.
- Si vous êtes malin et que vous utilisez votre bibliothèque `libft` pour votre `filler`, vous devez en copier les sources et le `Makefile` associé dans un dossier nommé `libft` qui devra être à la racine de votre dépôt de rendu. Votre `Makefile` devra compiler la bibliothèque, en appelant son `Makefile`, puis compiler votre projet.
- Votre devez coder en `C`, à la Norme.
- Les variables globales sont interdites.
- Votre programme ne doit pas avoir de fuites mémoire.
- Vous devez gérer les erreurs de façon stricte. En aucun cas votre programme ne doit quitter de façon inattendue (Segmentation fault, etc...).
- Dans le cadre de la partie obligatoire, vous avez le droit d'utiliser les fonctions suivantes de la `libc` :
  - `write`
  - `malloc`
  - `free`
  - `read`
  - `perror`
  - `strerror`

Chapitre V

Partie obligatoire

V.1 Le but du jeu

- Deux adversaires s'affrontent à tour de rôle.
- L'objectif est de gagner le plus de points en remplissant le plateau de jeu avec le plus de pièces possible.
- Le plateau est défini par `X` colonnes et `Y` lignes. Il fera donc `X x Y` cases.
- A chaque tour, le programme du joueur actif se voit fournir l'état actuel du plateau et une pièce à placer.
- La VM représente les pièces posées par un des programmes joueurs avec des charactères `'X'` et celles posées par l'autre avec des charactères `'O'`.
- Une pièce est définie par `x` colonnes et `y` lignes. Dans chaque pièce, une forme d'une ou plusieurs cases est représentée avec le charactère `'*'`.
- Pour pouvoir poser une pièce, il faut qu'une case de la forme recouvre une case, et une seule case, d'une forme précédemment posée.
- La forme doit rentrer intégralement dans le plateau.
- Le plateau contient une première forme pour chaque programme joueur pour initier la partie.
- La partie s'arrête à la première erreur : dès qu'une pièce ne peut plus être posée ou a été mal posée.
- La VM calcule ensuite le score pour chacun des deux programmes joueurs. Le score le plus élevé remporte la partie.

V.2 Le Plateau

Un plateau est donc une grille de deux dimensions avec un nombre de lignes et de colonnes arbitraires. Pour lancer la partie un plateau initial doit être passée en argument à la VM. Ce plateau initial doit comporter une forme de départ pour chaque joueur. Voici un exemple de plateau initial de 14 par 30 :

```
Plateau 14 30:
    012345678901234567890123456789
000 ..............................
001 ..............................
002 ..X...........................
003 ..............................
004 ..............................
005 ..............................
006 ..............................
007 ..............................
008 ..............................
009 ..............................
010 ..............................
011 ............................O..
012 ..............................
013 ..............................
```

V.3 Les pièces

Les pièces sont générées aléatoirement par la VM. Vous ne pouvez pas prévoir leur taille ni leur forme avant que la VM ne les transmettent à votre programme. Voiçi quelques exemples arbitraires de pièces possibles pour vous donner une idée :

```
Piece 4 7   Piece 4 5:  Piece 3 6:
 ...*...      .**..       .****.
 ...*...      .***.       **....
 ...*...      ..*..       *.....
 ..***..      .....
```

V.4 Le programme joueur

Vous comprenez donc maintenant que le but du projet `Filler` est d'écrire un programme joueur capable de battre son adversaire. Ce programme joueur est passé en paramètre à la VM qui se chargera d'exécuter chaque programme joueur, puis de lui transmettre les informations nécessaires au calcul du prochain coup à chaque tour. A chaque tour, votre programme joueur devra donc :

- Lire le plateau et les pièces sur l'entrée standard. Ces informations proviennent bien évidemment de la VM.
- Ecrire sur la sortie standard les coordonnées pour placer la pièce. Le format sera sous la forme suivante : `X Y\n"`.

Faites bien attention d'écrire les coordonnées de la pièce et non de la forme !

Lorsque votre programme joueur lira sur l'entrée standard, pour la première fois, il lira son numéro de joueur et son nom transmis par la VM au format suivant :

```
$$$ exec p[PLAYER_NUMBER] : [PLAYER_NAME]
```

- Si vous êtes le premier joueur, votre programme sera représenté par les caractères `'o'` et `'O'`.
- Si vous êtes le second joueur, votre programme sera représenté par les caractères `'x'` et `'X'`.
- Les `'o'` et `'x'` minuscules servent à identifier la dernière pièce posée par votre adversaire pour vous permettre d'adapter votre stratégie en fonction de la sienne.

A chaque tour, la VM envoie la carte mise à jour et une nouvelle pièce au joueur concerné selon le format suivant :

```
Plateau 14 30:
    01234567890123456789012345678 9
000 ..............................
001 ..............................
002 ..............................
003 ..............................
004 ..............................
005 ..............................
006 ..............................
007 ..............................
008 ..............O...............
009 ..............................
010 ..............................
011 ..............................
012 ..............................
013 ..............................
Piece 4 7:
...*...
...*...
...*...
..***..
```

V.4.1 La VM

Vous trouverez en annexes de ce sujet une tarball `filler.tar.gz` contenant :

filler_vm : La VM de `Filler` que vous utiliserez pour développer votre programme joueur.

```
$>./filler_vm, made by Hcao and Abanlin, version 1.1

Usage: ./filler_vm -f path [-i | -p1 path | -p2 path] [-s | -q | -t time]

-t  --timeset timeout in second
-q  --quietquiet mode
-i  --interactiveinteractive mode(default)
-p1 --player1use filler binary as a first player
-p2 --player2use filler binary as a second player
-f  --fileuse a map file (required)
-s  --seeduse the seed number (initialization random) (man srand)
$>
```

maps : Des plateaux initiaux pour tester votre programme joueur dans des conditions différentes.

players : Des programmes joueurs de complexités très hétérogènes pour affronter le votre.

Voici un exemple concret de l'utilisation de la VM :

```
$>./filler_vm -f maps/map00 -p1 players/hcao.filler -p2 players/superjeannot.filler
# -------------- VM  version 1.1 ------------- #
#                                              #
# 42 / filler VM Developped by: Hcao - Abanlin #
#                                              #
# -------------------------------------------- #
launched players/hcao.filler
$$$ exec p1 : [players/hcao.filler]
launched players/superjeannot.filler
$$$ exec p2 : [players/superjeannot.filler]
Plateau 15 17:
    01234567890123456
000 .................
001 .................
002 .................
003 .................
004 .................
005 .................
006 .................
007 .................
008 ..O..............
009 .................
010 .................
011 .................
012 ..............X..
013 .................
014 .................
Piece 2 2:
*.
*.
<got (0): [7, 2]

    < ... LATER ... >

Plateau 15 17:
    01234567890123456
000 .................
001 OO...............
002 OOOOO............
003 .O...............
```

```
004 .O..............
005 .O..............
006 .O..............
007 .OO.............
008 ..O.....XXXXX....
009 .........XXxx....
010 ..........XXXXX..
011 ...........XX.X..
012 ..............X..
013 ................
014 ................
Piece 2 2:
.*
.*
<got (O): [0, 0]


    < ... MOAR LATER ... >

Plateau 15 17:
    01234567890123456
000 OOOOOOOOOOOOOOOOO
001 OOOOOOOOOOOOOOOOO
002 OOOOOOOOOOOOOOOOO
003 OOOOOOOOOOOOOOOOO
004 OOOOOOOOOOOOXOOOO
005 OOOOOOXXXXXXXXXXO
006 .OOOOXXXXXXXXXXX
007 .OOOOXXXXXXXXXXX
008 OOOOXXXXXXXXXXXX
009 OOOOXXXXXXXXXXXX
010 .xxXXXXXXXXXXXXX
011 XXXXXXXXXXXXXXXX
012 .XXXXXXXXXXXXXXX
013 .XXXXXXXXXXXXXXX
014 .XXXXXXXXXXXXXX.
Piece 1 3:
***
<got (X): [0, 0]
== O fin: 70
== X fin: 97
$>
```

Si vous rencontrez un problème avec la VM qui ne vient pas de votre programme, veuillez créer un ticket sur l'intranet et assigner l'équipe pédago. Nous ne repondrons pas aux tickets qui n'expriment pas clairement et en détails la nature du bug et comment le reproduire.

Chapitre VI

Partie bonus

Les bonus ne seront évalués que si votre partie obligatoire est PARFAITE. Par PARFAITE, on entend bien évidemment qu'elle est entièrement réalisée, et qu'il n'est pas possible de mettre son comportement en défaut, même en cas d'erreur aussi vicieuse soit-elle, de mauvaise utilisation, etc. Concrètement, cela signifie que si votre partie obligatoire n'obtient pas TOUS les points à la notation, vos bonus seront intégralement IGNORÉS.

Pour ce projet, nous vous proposons les bonus suivants :

- Un visualisateur graphique.
- Tout autre bonus que vous jugerez utile et qui pourra l'être également aux yeux de vos pairs.

Chapitre VII

Rendu et peer-évaluation

Rendez-votre travail sur votre dépôt `GiT` comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance.