EPITECH.
L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

A-Maze-Ing

Partie 2: Cases hexagonales et OCamlSDL

Guillaume Collet

Abstract: La suite du projet consiste à générer un labyrinthe à cases hexagonales puis de l'afficher grâce à la bibliothèque OCamlSDL.

Table des matières

Chapitre I

Rappel des consignes

Vous **DEVEZ** impérativemment respecter les consignes suivantes :

- Tous les traits impératifs d'OCaml sont interdits, sauf concernant les exceptions, les séquences d'instructions (quand nécessaire) et les tableaux. Ceci est bien évidemment limité au strict nécessaire et devra faire l'objet d'une justification en soutenance.
- Il est interdit que votre programme se termine à cause d'une exception non-gérée.
- Vous avez le droit d'utiliser **TOUT** ce qu'OCaml met à votre disposition. La seule exception concerne la bibliothèque graphique qui **DOIT** être OCamlSDL (`https://github.com/ocamlsdl/OCamlSDL`).
- Votre rendu **DOIT** compiler avec `ocamlc` **ET** `ocamlopt`. Aucun script ne sera accepté en soutenance.
- Chaque étape de la partie obligatoire **DOIT** pouvoir être compilée et exécutée de manière indépendante des autres. Pour cela, vous **DEVEZ** créer deux dossiers `step3` et `step4`. L'organisation de votre code en conséquence est à votre discrétion.
- Pour chaque partie, l'utilisation d'un Makefile pour compiler votre projet est obligatoire. Il DOIT donc y avoir un Makefile à la racine du projet qui lance les Makefile présents dans les deux sous-dossiers.
- Les étapes seront corrigées dans l'ordre du sujet. Il n'est donc pas utile de présenter une étape 4 sans étape 3.
- Il n'existe pas de norme OCaml à Epitech. Toutefois, tout code illisible pourra être sanctionné. Une indentation claire, de l'espace entre les fonctions et des commentaires judicieux vous éviteront des malus.
- Une conception modulaire et générique sera très appréciée.
- La soutenance mettra l'accent sur vos connaissances personnelles en OCaml.

- Gardez un oeil sur ce sujet car il est susceptible d'être modifié.
- Les évolutions du sujet seront notifiées sur le forum du module. Vous devez lire ce forum régulièrement.
- Si vous constatez des erreurs dans ce sujet, merci d'en faire part sur le forum.
- Vous devez rendre votre projet sur le système de rendu mis à votre disposition par Epitech. Le nom du répertoire est `OCAML_2015_amazeing2`.
- Vos dépôts seront ramassés à l'heure exacte de la fin du projet, l'intranet Epitech faisant foi.
- Seul le code présent sur votre dépôt sera évalué lors de la soutenance.

Bon courage!!

Chapitre II

Partie obligatoire

II.1 Step3 : Cases hexagonales

La génération d'un labyrinthe à cases carrées ne vous pose plus de souci. Nous vous demandons donc de générer un labyrinthe à cases hexagonales (en nid d'abeille).

Consignes :

- Vous **DEVEZ** compiler un exécutable nommé `step3` prenant en argument la hauteur et la largeur du labyrinthe à générer ainsi qu'une option permettant de choisir le type de case :

  ```
  ./step3 -r 12 -c 34 --hexagonal
  ./step3 -r 6 -c 21 --square
  ```

- Le labyrinthe **DOIT** être aléatoire.
- En cas d'arguments invalides ou de nombres incorrects, vous **DEVEZ** afficher un *usage* et quitter proprement.
- Vous **DEVEZ** proposer une fonction `solve` qui permet de résoudre le labyrinthe.
- Vous **DEVEZ** afficher votre labyrinthe sur la sortie standard une fois celui-ci généré avec les cases de départ et d'arrivée ainsi que la solution.
- Les salles de départ (ST) et d'arrivée (ED) doivent être sur un bord.

```
Les cases hexagonales ont 6 portes qui mènent vers 6 cases
adjacentes. La conception de votre labyrinthe à cases carrées
doit pouvoir être étendue pour utiliser 2 portes supplémentaires.
L'utilisation de functors semble appropriée, non ?
```

Une fois le labyrinthe généré, vous devez l'afficher sur la sortie standard en suivant **EXACTEMENT** le format des exemples suivants :

```
$> ./step3 -r 3 -c 7 --hexagonal
 __    __    __    __
/  \__/  \__/..\__/  \
\  /.. __/.. __ .. __/
/.. __ .. __/      ED\
\  /  \  /  \__/  \  /
/ST __/   __/   __/  \
\__/   __   \__/   __/
   \__/  \__/  \__/

$> ./step3 -r 2 -c 8 --hexagonal
 __    __    __    __
/  \__/ED\__/  \__/  \__
\__ .. __/    __ ST   /  \
/.. __/..\__/..\  /   __/
\__ .. __ .. __ ..\__   \
   \__/  \__/  \__/  \__/

$> ./step3 -r 3 -c 4 --hexagonal
 __    __
/  \__/ED\__
\__ .. __   \
/.. __/  \__/
\   .. __/ST\
/  \  /..\  /
\__/.. __ ..\
   \__/  \__/

./step3 -r 4 -c 8 --square
+----------------------+
|        |ED ..        |
|  +--+  +--+  +  +--+--|
|  |.. .. .. ..|  |     |
|  +  +--+--+--+  +--+  |
|  |.. ..|.. ..|        |
|  +--+  +  +  +--+--+  |
|     |.. ..|.. .. ST|  |
+----------------------+
```

II.2 Step4 : Un vrai jeu en OCaml

Enfin les choses sérieuses ! Avec votre labyrinthe résolvable à cases carrées ou hexagonales, il est temps de montrer votre créativité et votre maîtrise d'OCaml !

Vous allez réaliser un petit jeu vidéo au gameplay de votre choix en vous basant sur votre labyrinthe. Vous êtes entièrement libres et la limite est votre imagination !
Consignes :

- Vous **DEVEZ** compiler un exécutable nommé `step4` prenant en argument la hauteur et la largeur du labyrinthe à générer ainsi que toute option qui vous paraîtra pertinente.
- En cas d'arguments invalides ou de nombres incorrects, vous **DEVEZ** afficher un *usage* et quitter proprement.
- les murs et les salles **DOIVENT** être clairement identifiables, ainsi que les cases de départ et d'arrivée (le net regorge de sprites en tout genre).
- Vous **DEVEZ** ajouter un bouton permettant d'afficher la solution du labyrinthe.
- Le labyrinthe **DOIT** être aléatoire.
- Votre boucle de jeu **DOIT** clairement distinguer 3 étapes qui **DOIVENT** être dans cet ordre :

  1. Gestion des événements.
  2. Logique du jeu et update du rendu.
  3. Affichage du rendu.

- Votre jeu doit être bien `:)`

Dans cette partie, avoir un personnage qui bouge ou des animations n'est **PAS** obligatoire. Il s'agit principalement d'afficher le labyrinthe à l'aide de la OCamlSDL et d'ajouter un bouton pour afficher la solution.

`L'utilisation d'un timer est toute indiquée si l'on souhaite s'abstraire de la vitesse du CPU...`

Chapitre III

Bonus

Une fois la partie obligatoire terminée et fonctionnelle, vous pouvez ajouter des bonus à votre projet, par exemple :

- Des animations
- Des menus
- Des sons, de la musique
- Sauvegarde / chargement
- Mouvement d'un joueur
- etc

Cette dernière partie ne sera pas passée à la moulinette, vous êtes libres de proposer ce que vous voulez, de la façon que vous voulez. Profitez-en !