Projet PFA : Tactical RPG

Sylvain Conchon Julien Lopez

21 janvier 2019

1 Introduction

1.1 SDL

*S*imple *D*irectMedia *L*ayer est une bibliothèque de développement multi-plateforme (Windows, Mac OS X, Linux, iOS et Android) conçue pour fournir un accès bas-niveau à l'audio, au clavier, à la souris, au joystick, et au matériel graphique via OpenGL et Direct3D. Il est utilisé par des lecteurs multimédia, des émulateurs, et des jeux populaires comme FTL : Faster Than Light, ou Amnesia : The dark descent.

SDL 2.0 est distribué sous la licence zlib. Cette licence vous permet d'utiliser SDL librement dans n'importe quel logiciel.

SDL est écrit en C, fonctionne nativement avec C++, et il existe des versions pour plusieurs autres langages, y compris pour OCaml.

1.2 Tsdl

Tsdl est la bibliothèque qui relie SDL à OCaml. Les fonctions de la bibliothèque sont pour la plupart les mêmes que SDL avec des noms dans le style OCaml.

Pour installer Tsdl sur votre machine, exécutez les commandes suivantes :

```
opam init
eval $(opam config env)
opam switch 4.07.0
eval $(opam config env)
export LIBFFI_CFLAGS="-I/opt/anaconda3/pkgs/libffi-3.2.1-1/include"
export LIBFFI_LIBS="-L/opt/anaconda3/pkgs/libffi-3.2.1-1/lib -lffi"
export SDL_LIB_PATH="/public/sdl/lib"
```

```
export PKG_CONFIG_PATH="$PKG_CONFIG_PATH:$SDL_LIB_PATH/pkgconfig"
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/opt/anaconda3/pkgs/libffi-3.2.1-1/lib"
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$SDL_LIB_PATH"
export LIBRARY_PATH="$LIBRARY_PATH:$SDL_LIB_PATH"
opam install tsdl tsdl-mixer tsdl-image tsdl-ttf
```

Pour compiler avec Tsdl, vous pouvez utiliser la ligne de compilation :

```
ocamlbuild -use-ocamlfind -package tsdl,tsdl_mixer,tsdl_image,tsdl_ttf trpg.byte
```

1.3 Documentation

Pour SDL 2 : https://wiki.libsdl.org/.

Pour Tsdl :

— http://erratique.ch/software/tsdl/doc/,

— https://github.com/tokenrove/tsdl-mixer,

— https://github.com/tokenrove/tsdl-image,

— https://github.com/tokenrove/tsdl-ttf

Introduction à SDL 2 : https://fr.wikibooks.org/wiki/Programmation_avec_la_SDL

Un tutoriel assez complet : http://lazyfoo.net/tutorials/SDL/ (attention aux fonctions SDL 1.2 utilisées dans la leçon 2 !)

Vous trouverez de nombreux tutoriels à SDL, mais **attention à ne suivre que les tutoriels pour SDL 2.0 !** La bibliothèque a beaucoup changé entre les versions 1.2 et 2.0. Notamment, n'utilisez pas les fonctions telles que `SDL_BlitSurface` ou `SDL_BlitScaled`, utilisez `SDL_RenderCopy` comme expliqué dans le tutoriel https://fr.wikibooks.org/wiki/Programmation_avec_la_SDL/Les_textures.

1.4 Aide

Tsdl définit des fonctions avec des noms similaires à SDL. Par exemple, l'équivalent de SDL_Init en Tsdl est Tsdl.Sdl.init. Une fois que vous avez compris comment SDL fonctionne, il vous suffit de trouver l'équivalent des fonctions en Tsdl et d'utiliser celles-ci à la place des fonctions SDL.

Vous verrez que beaucoup de fonctions dans Tsdl (comme Tsdl.Sdl.init) renvoient un type `'a Tsdl.Sdl.result = ('a, [ `Msg of string]) Result.result` où Result.result est un type défini dans OCaml (version $\geq$ 4.03.0) comme le type de base :

```
type ('a, 'b) result =
| Ok of 'a
| Error of 'b
```

où `Ok x` signifie que l'opération a réussi et a renvoyé une valeur `x`, et `Error e` signifie que l'opération a échoué. Dans le cas de `Tsdl.Sdl.result`, `e` est donc un constructeur `` `Msg s `` où `s` est un message d'erreur.

Donc, ces fonctions Tsdl renvoient `Ok x` ou `Error (`Msg s)` selon si la fonction a réussi ou échoué. Il vous faudra vérifier que la fonction a bien réussi et récupérer la valeur de retour en utilisant un pattern matching. Par exemple :

```
open Tsdl

match Sdl.init Sdl.Init.video with
| Error (`Msg e) -> Sdl.log "Init error: %s" e; exit 1
| Ok () -> (* Le code continue ici *)
```

On vous donne les fichiers `sdl_tools.ml` et `sdl_tools.mli` qui contiennent des fonctions utilitaires pour vous aider à factoriser votre code et comprendre comment utiliser Tsdl.

2 Modalités

Vous devrez coder un jeu vidéo en deux dimensions du type tactical RPG (https://fr.wikipedia.org/wiki/Tactical_RPG). Le projet se fera en binôme. Vous devrez coder en OCaml en utilisant la bibliothèque Tsdl. Les paliers du sujet décrivent les fonctionnalités attendues. L'ordre des paliers est un guide pour le développement du jeu que vous pouvez choisir de suivre ou pas. Vous êtes totalement libres de réaliser le jeu que vous souhaitez tant que vous respectez les règles ci-dessus et les instructions des paliers.

Vous pouvez organiser votre code de la manière que vous voulez, cependant vous devrez découper les différentes parties de votre code en modules, par exemple au moins un module pour représenter une scène du jeu, et un module pour représenter un objet se trouvant dans une scène. Vous devrez également coder votre jeu en utilisant *au maximum* la programmation fonctionnelle, incluant ce que vous avez appris en cours (listes, fonctions récursives, itérateurs, ...). Vous pouvez utiliser la programmation impérative (tableaux, références, ...) **uniquement dans des cas ponctuels justifiés**.

Vous devrez présenter votre programme pendant une soutenance. Vous serez notés sur les fonctionnalités de votre jeu, mais aussi sur la qualité de

votre code, et sur votre capacité à répondre aux questions sur le développement du projet.

Vous devrez aussi rendre un rapport dans lequel vous résumerez les fonctionnalités de votre jeu et décrirez le déroulement du développement, votre organisation, et les difficultés rencontrées.

3 Palier 1

3.1 Buts

— Fenêtre qui s'ouvre au lancement du jeu
— Image de fond
— Possibilité de quitter le jeu

3.2 Détails

Votre jeu doit avoir une fenêtre graphique avec une image de fond. Il doit être possible de quitter le jeu en appuyant sur une touche (Échap par exemple) ou en cliquant sur la croix en haut à droite de la fenêtre.

4 Palier 2

4.1 Buts

— Scènes et objets
— Grille
— Curseur

4.2 Détails

Le jeu sera composé d'*objets* : les personnages (alliés et ennemis), les obstacles, etc. Ces objets seront représentés dans des *scènes*. Le jeu montre la scène courante en affichant les objets présents dans cette scène dans leurs positions respectives. Pour les objets et pour les scènes, vous devrez définir un module contenant un type et des fonctions permettant de manipuler ce type. Le type pour les objets contiendra notamment sa position, ses points de vie et toutes autres données utiles à leur affichage où au calcul de leur

statut. Ce type ne devra pas être visible en dehors du module, vous devrez donc créer des fonctions pour le manipuler, notamment une fonction pour créer une instance de ce type.

Dans une scène, il y a une grille de cases dans lesquelles les objets peuvent se trouver. Par conséquent, vous pouvez considérer qu'un objet a une position (x, y) qui correspond à ses coordonnées dans la grille. La grille **doit** être représentée par un tableau car sa taille est connue à l'avance et donnée à la création de la scène. Il ne peut y avoir qu'un seul objet sur une case.

Une scène a un objet par défaut : un *curseur*. Ce curseur sera utilisé par le joueur pour sélectionner un objet dans la scène. Le joueur doit pouvoir déplacer le curseur case par case.

5 Palier 3

5.1 Buts

— Tour par tour
— Action : se déplacer
— Action : ramasser un objet
— Action : attaquer

5.2 Détails

Vous devez avoir un système de tours de jeu. Les alliés et les ennemis jouent à tour de rôle. Il doit être possible de terminer le tour en pressant une touche. Chaque joueur ne doit pouvoir interagir qu'avec ses personnages.

Un joueur doit pouvoir réaliser une action avec un de ses personnages en positionnant le curseur sur le personnage puis en appuyant sur une touche (par exemple "m" pour "mouvement", "a" pour "attaque"), le curseur ne doit alors pouvoir se déplacer que sur les cases atteignables par le personnage, et le personnage réalise l'action en ciblant la case indiquée par le curseur quand le joueur presse la touche "Entrée". Il doit aussi être possible d'annuler une action en pressant la touche d'action à nouveau et/ou en appuyant sur la touche "Échap".

Les personnages doivent pouvoir se déplacer sur la grille. Chaque objet doit avoir une "vitesse", c'est à dire la distance maximale en nombre de cases que le personnage peut couvrir en un tour. De plus, un personnage ne peut

se déplacer qu'une fois par tour. N'oubliez pas que deux objets ne peuvent pas se trouver sur la même case.

Certains objets doivent être inanimés et doivent pouvoir être ramassés par les personnages. Ramasser un objet doit avoir un effet sur le personnage qui l'a ramassé (une hache augmente ses dégâts, une potion augmente ses points de vie, ...).

Les personnages doivent pouvoir attaquer leurs ennemis (et autres objets si vous voulez) si ils sont suffisamment proches. Chaque objet doit avoir des points de vie, une portée d'attaque en nombre de cases, et des points de dégâts. Lors d'une attaque, l'objet subissant l'attaque perd des points de vie égaux aux points de dégâts de l'objet attaquant.

6 Palier 4

6.1 Buts

— Fin de partie
— Menu
— Objets en arrière plan
— Visualisation de portée d'action

6.2 Détails

Lorsque un objet est à 0 points de vie (ou moins), il est considéré comme "mort". Lorsque tous les ennemis sont morts, vous devez afficher une bannière de victoire sur la scène, et aucune interaction avec la scène doit être possible. Même chose si tous les alliés sont morts, vous devez afficher une bannière de "game over".

Le jeu doit commencer avec un menu basique (le joueur doit appuyer sur une touche pour accéder à une scène), et revenir sur ce menu lorsque la partie est terminée ou lorsque le joueur appuie sur une touche (par exemple la touche "q" pour "quitter").

Vous devrez avoir deux plans sur lesquels les objets peuvent exister : le *premier plan* et l'*arrière plan*. Les objets dans le premier plan seront les personnages et tout autres objets qui peuvent interagir entre eux, et les objets dans l'arrière plan seront du décor. Les objets en premier plan peuvent se trouver sur la même case visuelle que des objets en arrière plan, dans ce cas

l'image de l'objet de premier plan s'affiche au dessus de celle de l'objet en arrière plan.

Lorsque une action est sélectionnée (par exemple lorsque le joueur appuie sur "m" après avoir déplacé le curseur sur un de ses personnages), vous devez afficher les cases qui peuvent être sélectionnées comme cible de l'action (par exemple avec une couleur bleue transparente). N'oubliez pas que deux objets ne peuvent pas se trouver sur la même case.

7 Palier 5

7.1 Buts

— IA
— Animation
— Menu d'actions

7.2 Détails

Vous devez avoir une intelligence artificielle pour les ennemis. Cette intelligence artificielle devra être codée dans son (ses) propre(s) module(s). L'intelligence artificielle doit être capable de déplacer les ennemis vers les alliés (éviter les objets sur le chemin est un bonus), et de les faire attaquer si ils sont suffisamment proches.

Les objets mobiles doivent avoir une animation basique durant un mouvement. Vous pouvez lire la leçon 14 du tutoriel pour l'animation.

Lorsque le joueur presse la touche "Entrée" sur un de ses personnages, un menu des actions possible ce tour par le personnage doit s'afficher, et le joueur doit pouvoir interagir avec ce menu pour sélectionner une action. Le résultat final doit être le même que de presser la touche d'action correspondante. La touche "Échap" doit permettre d'annuler l'affichage du menu d'actions. Le menu d'action doit au moins contenir l'action globale de fin de tour.

8 Bonus

— Son
— Utilisation de la souris
— Gestion de la taille de la fenêtre

— Affichage des points de vie et de dégâts
— Effets des objets en arrière plan
— Plus d'actions
— Plus d'animations
— Animations avancées
— Menu avancé (options, sauvegarde)
— Codes de triche
— Différentes interactions (baliste, obstacles, ...)
— IA intelligente
— Équipement plus complet
— Différentes conditions de victoires
— ...