Le 23/11/2004 Version 3

MODENO

Alexandre NOUVEL
Guillaume MORIN
Jonathan DETCHART

Dernière révision : 16/01/2018

Table des matières

1.Préambule

«modeno» est un jeux de bataille au tour par tour. Il se joue contre l'ordinateur. Chaque adversaire a un certain nombre d'appareil de combat (tank, jeep, aerogliss ...). Ces appareils sont caractérisés par:

- des points de déplacement
- des points de résistance (armure)
- un certain nombre de tir
- une distance de tir (certains appareils pourront tirer plus loin que d'autres)
- des points de constructions (certains appareils pourront construire ou réparer d'autres unités)

Chaque tir diminue les points de résistance de l'adversaire d'un certains nombre d'unités. Ce nombre dépendra du type d'appareil, par exemple un char occasionnera plus de dégât qu'une jeep. Lorsqu'un appareil n'a plus de point de résistance, celui-ci est détruit.

2.Déroulement d'une partie

Une partie se compose de plusieurs tours. À chaque tour, on change de joueur. Pendant que l'un attaque (c'est-à dire qu'il peut se déplacer puis tirer), l'autre défend (il ne peut que tirer, les tirs se feront de manière automatique, le joueur reste donc passif et attendra la fin de l'attaque pour jouer de nouveau). Une partie est terminée lorsque au cours d'un tour, un des deux joueurs (l'ordinateur, ou le joueur) a perdu tous ses appareils.

Au début d'un tour, chaque joueur dispose d'un nombre d'appareil déterminé.
Lors d'un tour, celui qui joue peut déplacer ses appareils d'un certain nombre d'unités. Il peut ensuite tirer un certains nombre de tir sur les appareils de son adversaire. Si il lui reste après ces tirs des points de déplacement, l'attaquant pourra replier ses appareils pour anticiper la défense au prochain tour. Son adversaire pourra riposter automatiquement selon la configuration de l'attaquant. Par exemple, si le joueur attaque une dizaine de char avec une seule jeep, les chars riposteront à tour de rôle de telle manière que sa jeep soit détruite assez rapidement. Ainsi, on supprime certaines invraisemblances. L'attaquant sera donc tenu d'adopter une stratégie pour ne pas se faire éliminer ses appareils. La riposte d'un appareil pourra être prévu dans les cas suivants:

- lorsque le même appareil d'un attaquant tire plus de trois fois, alors l'adversaire riposte une fois
- lorsque l'attaquant vient affronter plus de deux ennemis avec un seul appareil alors l'adversaire riposte
- lorsque l'attaquant affronte l'ennemie avec des appareils de qualités inférieurs, alors l''adversaire riposte

3.Description du terrain de combat

Le terrain de combat est composé de rocher et d'arbres qui joueront le rôle d'obstacle. Les appareils attaquant seront obligés de les contourner. On pourra également se servir de ceux-ci comme moyen de défense. Une carte sera composé de prairie, de marais, de mer, de montagne et de rivière. Seulement quelques appareils pourront traverser les rivières (aerogliss par exemple). Les tanks, trop lourd ne pourront traverser ni les marais, ni les rivières. Certains engins de constructions pourront construire des ponts au dessus des rivières pour que d'autres unités puissent les traverser.

4.Caractéristiques des différents appareils

Aucun appareil ne pourra avoir un gros avantage par rapport aux autres. Par exemple, une jeep peut se déplacer plus loin qu'un char, elle aura un nombre de tir supérieur à un char mais ses tirs feront moins de dégât et son armure sera beaucoup plus faible.
Un lance missile tirera plus loin qu'un char, ses missiles pourront sauter les obstacles mais son armure sera faible et son déplacement limité.
Un aerogliss pourra se déplacer sur l'eau, il aura beaucoup de points de déplacement mais son armure sera faible et les dégâts qu'il infligera à ses adversaires seront limités
Etc ...

Certains appareils pourront construire des tourelles, des ponts et des murs de fortifications. Ils pourront également réparer d'autres unités.

5.Visualisation de la carte et agencement du jeux

Le jeux est en deux dimensions (vue de haut/profil). Ce que l'on appel communément du « faux 3D » (on a l'impression qu'il est en 3D mais la programmation est en 2D). Néanmoins avant de passer à la 2D isométriques, les premiers tests s'effectueront en vue de dessus avec des tuiles de forme carrés.
Le jeux se joue essentiellement à la souris. La carte sera composée du « terrain de combat » et d'une barre de configuration. Le « terrain de combat » correspond au jeux proprement dit, c'est l'endroit où évolue les appareils.
La barre de configuration sera composée des éléments suivants:

- un bouton pour quitter le jeux
- un bouton « déplacement »
- un bouton « tir »
- un bouton « construction » accessibles uniquement pour les appareils de construction
- une zone permettant de visualiser le nombre de points de déplacement, le nombre de points de tirs et le nombre de points de résistance de l'appareil sélectionné.

Pour déplacer un appareil, on le sélectionne avec un clic gauche de la souris. Un halo vert entourant l'appareil nous indique alors que celui-ci est sélectionné. Ensuite, on clique sur le bouton « déplacement » de la barre de configuration. Lorsqu'on clique une troisième fois sur l'endroit où l'on veut déplacer le véhicule, un trajet vert est matérialisé par une succession de polygone (dont le nombre correspond au point de déplacement du véhicule). Si le véhicule est dans l'incapacité d'aller à l'endroit indiqué, le trajet possible sera matérialisé en vert et le trajet impossible continuera en rouge. Enfin, si un quatrième clique est effectué sur un polygone vert, alors le véhicule se déplace à l'endroit indiqué.

Pour tirer sur un véhicule, on sélectionne l'appareil attaquant avec un clic gauche de la souris. On clique ensuite sur le bouton « tir » de la barre de configuration et on choisit enfin le véhicule à attaquer.

6.Extension prévu à long terme

Prise en charge du réseau. Moteur temps réels en vu de remplacer le « tour par tour ». Vrai 3D en utilisant OpenGL.

7.Langage et bibliothèque utilisé

Le langage utilisé est le C. La bibliothèque pour gérer l'affichage est SDL version 2. Le jeux est en licence GPL. La plate forme de développement est Linux. Des portages sous BSD et Windows sont envisageables (SDL est disponible sur la quasi-totalité des plates formes). Le but de ce projet est tout d'abord de se familiariser avec la bibliothèque SDL. On pourra envisager par la suite, lorsqu'on maîtrisera cette bibliothèque d'orienter le jeux vers le réseau et/ou une prise en charge totale par l'ordinateur en temps réels (du type Star Craft ou Total Annihilation). Comme SDL s' interface avec OpenGL, on pourra même envisager d'orienter le jeux vers une « vraie » 3D.
Puisque le jeux est en GPL, on utilisera comme logiciels de conception Blender pour concevoir les cartes, les chars etc.... Dans ce tutoriel et en guise d'exemples, les décors des cartes et les véhicules ont été crée à partir de KolourPaint pixel par pixel. On peut aussi utiliser le logiciel Tiled pour créer des Cartes.

8.Conception du jeux

1.Chargement d'une carte, principe de base et description des fichiers

Le programme lit deux fichiers par défaut. Le premier fichier (par exemple map/map.txt) permettra de charger les différents éléments composant une carte (terre, marais, montagne, arbre, rocher ...), et leurs positions. Ce fichier contiendra du code ASCII, chaque lettre correspondant à un type de terrain différent (principe des tuiles). Le deuxième fichier (par exemple unit/unit.txt) contiendra les caractéristiques des unités à savoir :
- fichier image
- fichier sons
- coordonnées de départ (x,y)
- type de terrain qui peut-être parcouru (Terre, Montagne, Marais, Rivière)
- point de déplacement
- nombre de tir
- distance de tir (un véhicule peut tirer plus ou moins loin)
- puissance de tir (un tank va davantage diminuer l'armure d'un ennemi que la mitraillette d'une jeep)
- points de résistances

Les différents objets possibles sont:
- des objets statiques (rochers, arbres, mur, tourelles)
- des véhicules
  Parmi les véhicules on peut distinguer:
  - le char
  - le soldat
  - la jeep
  - l'aerogliss
  - le lance missile
  - un véhicule de construction
- des objets particuliers comme des tourelles de défense (deplacement et mouvement à 0)

Les informations de la carte sont contenus dans un simple fichier texte ASCII. Il sera ainsi facile de créer de nouvelles cartes.

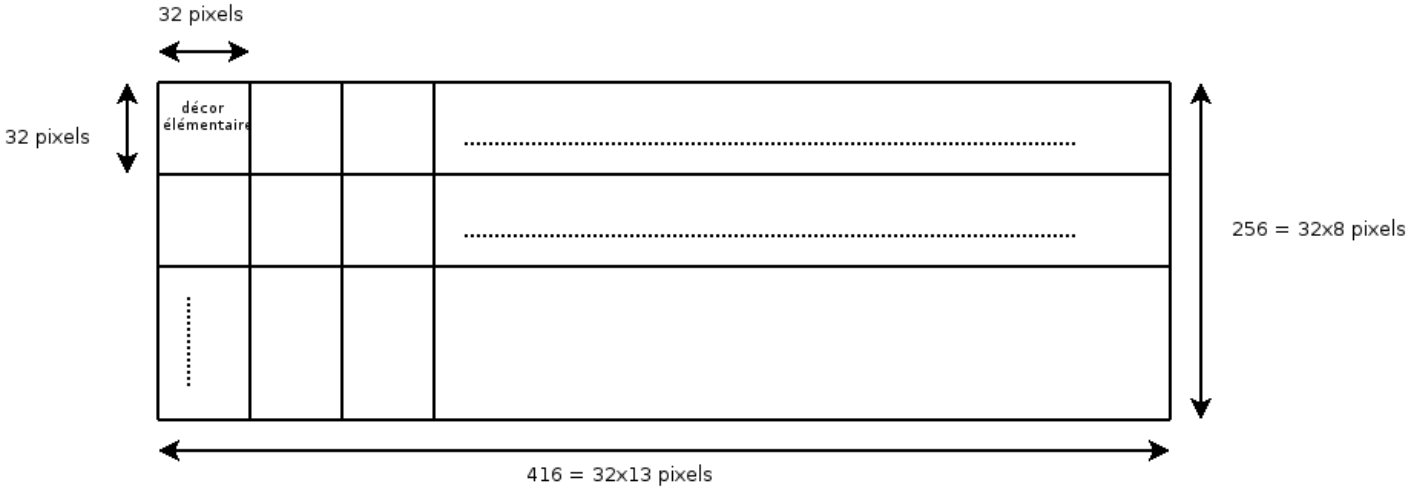
1 Description du premier fichier – Conception de la carte

On utilise le principe des tuiles. On s'appuie sur un fichier contenant les codes ASCII allant de 32 (correspondant à la touche « espace ») à 126 (correspondant à la touche « ~ »). Soit en tout 95 symboles de disponible pour la conception de la carte.
Exemple : supposons que le symbole «8» correspond à un type « prairie » et le symbole «V» à un type « marais ». On aura alors le fichier suivant, que l'on appellera « Map.txt »

8 8 8 8 8 8 8 8
8 8 8 8 V 8 8 8
8 8 8 8 V V 8 8
8 8 8 8 8 8 8 8

Qui donnera la carte suivante une fois le jeu lancé : 

Tous les décors (prairie, marais, montagne …) seront regroupés dans un seul fichier image au format BMP (format natif de la SDL2). Chaque décor élémentaire aura une taille de 32x32 pixels.
Le fichier image « Map.bmp » aura une taille de 416x256 pixels, ce qui permettra d'associer les 95 symboles ASCII à un décor élémentaire (le dernier emplacement restera libre).



Map.bmp avec des décors de 32x32

Il y aura donc en tout 8 lignes et chaque ligne pourra contenir jusqu'à 13 décors élémentaires (sauf la huitième qui en contiendra 4). Chaque décor élémentaire correspondra à un code ASCII (de 32 à 126).

Code 32 Espace	Code 33 !	Code 34 "	Code 35 #	Code 36 $	Code 37 %	Code 38 &	Code 39 ‘	Code 40 (	Code 41)	Code 42 *	Code 43 +	Code 44 ,
Code 45 -	Code 46 .	Code 47 /	Code 48 0	Code 49 1	Code 50 2	Code 51 3	Code 52 4	Code 53 5	Code 54 6	Code 55 7	Code 56 8	Code 57 9
Code 58 :	Code 59 ;	Code 60 <	Code 61 =	Code 62 >	Code 63 ?	Code 64 @	Code 65 A	Code 66 B	Code 67 C	Code 68 D	Code 69 E	Code 70 F
Code 71 G	Code 72 H	Code 73 I	Code 74 J	Code 75 K	Code 76 L	Code 77 M	Code 78 N	Code 79 O	Code 80 P	Code 81 Q	Code 82 R	Code 83 S
Code 84 T	Code 85 U	Code 86 V	Code 87 W	Code 88 X	Code 89 Y	Code 90 Z	Code 91 [	Code 92 \	Code 93]	Code 94 ^	Code 95 _	Code 96 `
Code 97 a	Code 98 b	Code 99 c	Code 100 d	Code 101 e	Code 102 f	Code 103 g	Code 104 h	Code 105 i	Code 106 j	Code 107 k	Code 108 l	Code 109 m

Code 110 n	Code 111 o	Code 112 p	Code 113 q	Code 114 r	Code 115 s	Code 116 t	Code 117 u	Code 118 v	Code 119 w	Code 120 x	Code 121 y	Code 122 z
Code 123 {	Code 124 |	Code 125 }	Code 126 ~									

On aura 4 types de terrains différents. À chaque type de terrains correspondra des types de véhicules qui pourront y évoluer :
- montagne (infanterie)
- terre/prairie (jeep, aerogliss, tank léger, tank lourd, lance missile, infanterie, engin de construction)
- marais/rivières (jeep, aerogliss, tank léger, infanterie)
- mer/océan (bateau, barge, aerogliss)

La première ligne du code ASCII (code 32 à 44) correspond aux obstacles (infranchissable quelque soit l'unité)
La deuxième et troisième ligne du code ASCII (code 45 à 70) contiendront les images liées aux montagnes.
La quatrième et cinquième ligne du code ASCII (code 71 à 96) contiendront les images liées aux terres/prairies/routes.
La sixième ligne du code ASCII (code 97 à 109) les décors liés aux marais/rivières.
La septième ligne du code ASCII (code 110 à 122) contiendra les décors liés aux mers/océans.
La neuvième ligne du code ASCII (code 123 à 126) à voir au cas où

La méthode affichant les décors dans la carte est la suivante :
Fichier « Map.c » :

```
void DisplayMapSDL(Map *map, SDL_Renderer *renderer) {
	SDL_Surface *map_image;
	SDL_Texture *texture;
	SDL_Rect Src;
	SDL_Rect Dest;
	char read='a';
	int i_read;
	int i=0, j;

	Src.x = 0;
	Src.y = 0;
	Src.w = SIZE_BMP;
	Src.h = SIZE_BMP;
	Dest.w = SIZE_BMP;
	Dest.h = SIZE_BMP;
	map_image = SDL_LoadBMP(map->file_image_tiles);
	if (map_image) {
		texture = SDL_CreateTextureFromSurface(renderer, map_image);
		for ( j = 0; j < map->nb_row; j++) {
		while ( read != '\n' ) {
		 read = map->array_map[i][j];
		 i_read = read - 0;
		 if ( (i_read > 31) && (i_read < 44) ) { i_read = read - 32; Src.y = 0; }
		 if ( (i_read > 43) && (i_read < 56) ) { i_read = read - 44; Src.y = SIZE_BMP; }
		 if ( (i_read > 55) && (i_read < 68) ) { i_read = read - 56; Src.y = 2*SIZE_BMP; }
		 if ( (i_read > 67) && (i_read < 80) ) { i_read = read - 68; Src.y = 3*SIZE_BMP; }
		 if ( (i_read > 79) && (i_read < 92) ) { i_read = read - 80; Src.y = 4*SIZE_BMP; }
		 if ( (i_read > 91) && (i_read < 104) ) { i_read = read - 92; Src.y = 5*SIZE_BMP; }
		 if ( (i_read > 103) && (i_read <116) ) { i_read = read - 104; Src.y = 6*SIZE_BMP; }
		 if ( (i_read > 115) && (i_read <128) ) { i_read = read - 116; Src.y = 7*SIZE_BMP; }

		 printf("i_read = %i\n", i_read);
		 Src.x = i_read*SIZE_BMP;
		 Dest.x = i*SIZE_BMP;
		 Dest.y = j*SIZE_BMP;

		 SDL_RenderCopy(renderer, texture, &Src, &Dest);
		 printf("%c", map->array_map[i][j]);
		 i++;
		}//end while
		i=0;
		read='a';
		}//end for

		SDL_RenderPresent(renderer);
		SDL_DestroyTexture(texture);
	} else {
		printf("Error Load tiles image!\n");
	}//end if map_image
}//end DisplayMapSDL
```

2 Objets animés

Exemple de fichier contenant des objets animés (ou tourelle), par exemple le fichier «unit.txt »:

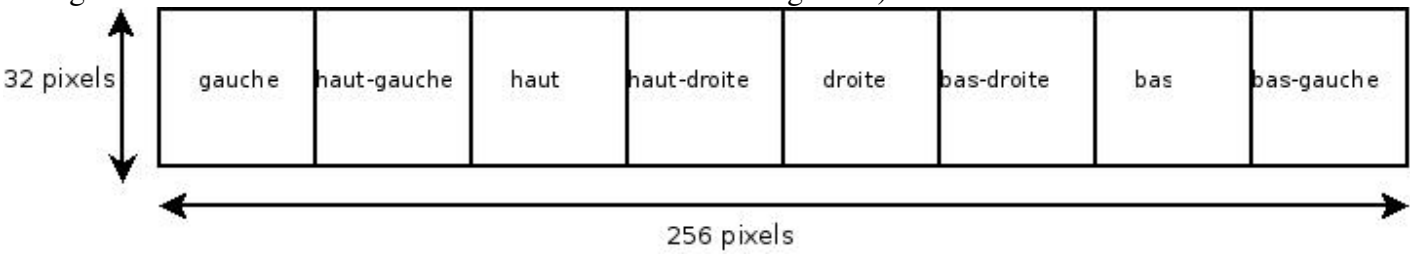
#image #	son	col	line	deplace- ment	résist- ance tir	dist- ance tir	puiss- ance tir	nombre tir
image/tank.bmp	sound/tank.wav	7	4	7	50	5	10	2
image/speeder.bmp	sound/speeder.wav	7	8	10	20	7	5	3
image/speeder.bmp	sound/speeder.wav	5	11	10	20	7	5	3
image/jeep.bmp	sound/jeep.wav	10	2	12	10	3	3	5

On remarquera que les tanks et les jeep sont les mêmes objets mais avec des images et des caractéristiques différentes (déplacement, résistance et tirs).
La structure d'un objet est donc de la forme suivante :

```
typedef struct {
    int column;
    int line;
    char *pathBMP;
    char *soundWAV;
    int movementPoint;      //point de mouvement
    int resistancePoint;
    int shootPoint;         //nombre de tir possible
    int shootDistance;      //distance de tir
    int shootPower;         //puissance des tirs
} Unit;
```

2. Gestion des objets animés (véhicules)

L'image des objets aura une longueur et une largeur multiple de 32 pixels. Elle sera découpée en plusieurs carrés ayant des tailles de 32x32. Ces carrés pourront représentés soit un tank, un aerogliss, un bateau etc ...
Les images de chaque objet de déplacement (jeep, speeder ...) font une taille de 256x32. La longueur de 256 pixels sera découpée en 8 parties de 32 pixels contenant chacun des images de direction différentes (gauche | haut-gauche | haut | haut-droite | droite | bas-droite | bas | bas-gauche).



Voici un exemple d'image de tank léger :

Il sera ainsi plus facile de créer de nouveaux objets de déplacements sans modifier le code principal.
Pour utiliser la portion de l'image qui nous intéresse, on utilisera le paramètres «capture» passé en deuxième argument dans la fonction SDL_RenderCopy().

3. Algorithme du plus court chemin

Comme chaque véhicule aura un certain nombre de points de déplacements, on va devoir calculer l'ensemble des chemins possibles autour du véhicule. Ceci en tenant compte des types de terrains. On utilisera pour cela l'algorithme de Dijsktra. Cet algorithme est implémenté dans les fichiers Tray.c et Tray.h. Les limitations autour du véhicule seront représentées par un effet de transparence sur les cases adjacentes au véhicule.
Les différents types de terrains sont :
- Les terres et prairies (EARTH)
- Les montagnes (MOUNTAIN)
- Les marais (SWAMP)
- Les mers, océans et rivière (WATER)

Le type de terrain sera donc décrit par un type énuméré :
```
typedef enum {OBSTACLE, EARTH, MOUNTAIN, SWAMP, WATER} Ground;
```

Un numéro associé au véhicule décrira les types de terrain sur lequel évoluera notre unité. On choisira les numéros dans cet ordre :

OBSTACLE	EARTH	MOUNTAIN	SWAMP	WATER
0	1	2	3	4

4. Gestion des menus

Un objet menu est composé de plusieurs objets boutons.
Un objet bouton est une image de taille 120x120 pixels séparée en trois parties de taille 120x40 pixels. On affiche une de ses trois parties selon les cas dans lesquels on se trouve.

- La première partie de l'image correspond au bouton lorsque celui-ci a été sélectionné (clique gauche).
- La deuxième partie correspond au bouton lorsque celui-ci n'est pas sélectionné et que la souris le survol.
- La troisième partie correspond au bouton lorsque celui-ci n'est pas sélectionné et que la souris est en dehors de la zone du bouton.

Un exemple de code est donné dans le chapitre « Description de la bibliothèque SDL »

5. Énumération des différents fichiers sources

- Map.c et Map.h : caractéristique de la carte, lit et charge la carte passé en argument
- main.c: lance le chargement de la carte (méthode contenu dans LoadMap.h) et rentre dans une boucle infinie en attente d'événement.
- Unit.c et Unit.h : caractéristique des véhicules terrestres (jeep, tank, lance missile, véhicule de construction)
- Scene.c et Scene.h
- Tray.c et Tray.h : algorithme calculant le chemin de coût minimal (Dijskstra)
- Button.c et Button.h : permet la création de bouton (Tirer, Déplacer, Quitter) afin de créer un menu interactif

9. Description de la bibliothèque SDL

Ce chapitre se contentera de donner plusieurs exemples simples sur l'utilisation de la bibliothèque SDL.

1. Initialisation SDL

Le programme suivant affiche une image en plein écran et se met en attente d'événement (boucle infinie). Pour quitter l'application, il suffit d'appuyer sur une touche du clavier.

```
#include <SDL/SDL.h>
#include <stdlib.h>

int main()
```

```
{
	SDL_Surface *screen;
	SDL_Surface *image;
	SDL_Event event;

	/* Initialisation de SDL */
	SDL_Init( SDL_INIT_VIDEO );
	atexit( SDL_Quit );

	/* On charge une image */
	image = SDL_LoadBMP( "../images/test3.bmp" );

	/* On ouvre une fenetre ayant les dimensions de l'image */
	screen = SDL_SetVideoMode(image->w, image->h , 16, SDL_FULLSCREEN);

	/* On inclu l'image dans la fenetre */
	SDL_BlitSurface( image, NULL, screen, NULL );

	/* On raffraichit le tout */
	SDL_UpdateRect( screen, 0, 0, image->w, image->h );

	/* Attente d'evenement */
	/* Il suffit d'appuyer sur une touche pour quitter l'application */
	int fin=0;
	while ( fin != 1 ){
		if ( SDL_PollEvent(&event) ){
			switch ( event.type ){
				case SDL_KEYDOWN:
					fin = 1;
					break;
			}
		}
	}

	SDL_FreeSurface( image );
	return 0;
}
```

2. Remplissage d'une carte avec des images de taille 32x32 de type prairie

```
#include <stdio.h>
#include <SDL/SDL.h>

#define TAILLE_ECRAN 640
#define HAUTEUR_ECRAN 480
#define TAILLE_BMP 40

int main(int argc, char *argv[]) {
	SDL_Surface *ecran;
	SDL_Surface *prairie;
	SDL_Rect position;
	SDL_Event event;

	int end = 0;
	int i, j, nb_horiz, nb_vert;
	// l'image "terre.bmp" fait 40x40
	nb_horiz=TAILLE_ECRAN/TAILLE_BMP;
	nb_vert=HAUTEUR_ECRAN/TAILLE_BMP;
	position.x=0;
	position.y=0;

	prairie = SDL_LoadBMP("./prairie.bmp");

	if (( SDL_Init(SDL_INIT_VIDEO  | SDL_INIT_AUDIO) == -1 )) {
		printf( "Ne peut pas initialiser SDL : %s\n", SDL_GetError() );
		exit(-1);
	}

	atexit( SDL_Quit );

	printf("Initialisation de SDL avec succes\n");
```

```
        ecran = SDL_SetVideoMode(TAILLE_ECRAN, HAUTEUR_ECRAN, 8, SDL_SWSURFACE);

        for (j=0;j<nb_vert;j++) {
                for (i=0;i<nb_horiz;i++) {
                        // On copie l'image source (prairie) dans l'ecran
                        SDL_BlitSurface(prairie, NULL, ecran, &position);
                        position.x=position.x+TAILLE_BMP;
                }// end for i
                position.y=position.y+TAILLE_BMP;
                position.x=0;
        }// end for j

        // On affiche l'ecran
        SDL_Flip(ecran);

        if ( ecran == NULL ) {
                printf( "Erreur : %s\n", SDL_GetError() );
        }

        // boucle principale
        while ( end != 1 ) {

                if ( SDL_PollEvent(&event) ) {
                        switch ( event.type ) {
                                case SDL_KEYDOWN:
                                                switch ( event.key.keysym.sym ) {
                                                        case SDLK_ESCAPE:
                                                                printf( "On quitte\n" );
                                                                end = 1;
                                                                break;
                                                }
                        }
                }
        }

        SDL_FreeSurface(prairie);
        SDL_FreeSurface(ecran);
        SDL_Quit();
        exit(0);
}
```

3. Code de l'objet Button

3 Fichier Button.h

```
#ifndef Button_H
#define Button_H

#include <SDL/SDL.h>

typedef struct {
    SDL_Surface *image;
    SDL_Rect capture;
    int selected;
    int x;
    int y;
} Button;

Button* new_button(char *image);
void Selected(Button *button);
void OnMouseOver(Button *button);
void NoSelected(Button *button);
void display(Button *button, int x_pixel, int y_pixel, SDL_Surface *screen);
int IsInside(Button *button, int event_motion_x, int event_motion_y);
void FreeButton(Button *button);


#endif //Button_H
```

4 Fichier Button.c

```
#include "Button.h"
```

```
Button* new_button(char *image) {
    Button* button;

    button = (Button*) malloc(sizeof(Button));

    button->image = SDL_LoadBMP(image);

    return button;
} // construct button

void Selected(Button *button) {
    button->capture.x=0;
    button->capture.y=0;
    button->capture.h=40;
    button->capture.w=120;
}

void OnMouseOver(Button *button) {
    button->capture.x=0;
    button->capture.y=40;
    button->capture.h=40;
    button->capture.w=120;
}

void NoSelected(Button *button) {
    button->capture.x=0;
    button->capture.y=80;
    button->capture.h=40;
    button->capture.w=120;
}

void display(Button *button, int x_pixel, int y_pixel, SDL_Surface *screen) {
    button->x = x_pixel;
    button->y = y_pixel;
    SDL_Rect position;
    position.x = x_pixel;
    position.y = y_pixel;
   // printf("pos x : %i\t pos y : %i\n", position.x, position.y);
    SDL_BlitSurface(button->image, &(button->capture), screen, &position);
    SDL_Flip(screen);
}

int IsInside(Button *button, int event_motion_x, int event_motion_y) {
    int result=0;
    if ( (event_motion_x > button->x) && (event_motion_x < button->x+120) &&
        (event_motion_y > button->y) && (event_motion_y < button->y+40)) {
            result=1;
    }//end if
    return result;
}

void FreeButton(Button *button) {
    SDL_FreeSurface(button→image);
}
```

5 Fichier main.c

```
#include <stdio.h>
#include <SDL/SDL.h>

#include "Button.h"

#define TAILLE_ECRAN 640
#define HAUTEUR_ECRAN 480
#define TAILLE_BMP 40
#define POS_X_BUTTON 200
#define POS_Y_BUTTON 200

int main(int argc, char *argv[]) {
        SDL_Surface *ecran;
        SDL_Surface *terre;
        SDL_Rect position;
    SDL_Rect capture;
        SDL_Event event;
```

```
int end = 0;
int i, j, nb_horiz, nb_vert;
int inside;
// l'image "terre.bmp" fait 40x40
nb_horiz=TAILLE_ECRAN/TAILLE_BMP;
nb_vert=HAUTEUR_ECRAN/TAILLE_BMP;
position.x=0;
position.y=0;

terre = SDL_LoadBMP("./terre.bmp");

if (( SDL_Init(SDL_INIT_VIDEO  | SDL_INIT_AUDIO) == -1 )) {
        printf( "Ne peut pas initialiser SDL : %s\n", SDL_GetError() );
        exit(-1);
}

atexit( SDL_Quit );

printf("Initialisation de SDL avec succes\n");

ecran = SDL_SetVideoMode(TAILLE_ECRAN, HAUTEUR_ECRAN, 8, SDL_SWSURFACE);

for (j=0;j<nb_vert;j++) {
        for (i=0;i<nb_horiz;i++) {
                // On copie l'image source (terre) dans l'ecran
                SDL_BlitSurface(terre, NULL, ecran, &position);
                position.x=position.x+TAILLE_BMP;
        }// end for i
        position.y=position.y+TAILLE_BMP;
        position.x=0;
}// end for j

// On affiche l'ecran
SDL_Flip(ecran);

        // Creation du bouton, il n'est pas selectionne au depart
Button *button = new_button("deplacement.bmp");
button->selected = 0;
NoSelected(button);
        display(button, POS_X_BUTTON, POS_Y_BUTTON, ecran);


if ( ecran == NULL ) {
        printf( "Erreur : %s\n", SDL_GetError() );
}

// boucle principale
while ( end != 1 ) {

        if ( SDL_PollEvent(&event) ) {
                switch ( event.type ) {
                        case SDL_KEYDOWN:
                                        switch ( event.key.keysym.sym ) {
                                                case SDLK_ESCAPE:
                                                        printf( "On quitte\n" );
                                                        end = 1;
                                                        break;
                                        }//end switch event.key

         case SDL_MOUSEMOTION:
                 inside = IsInside(button, event.motion.x, event.motion.y);
                 if (inside && (button->selected == 0)) {
                     OnMouseOver(button);
                     display(button, POS_X_BUTTON, POS_Y_BUTTON, ecran);
                 } else {
                     if (button->selected ==  0) {
                         NoSelected(button);
                         display(button, POS_X_BUTTON, POS_Y_BUTTON, ecran);
                         }
                     if (button->selected == 1) {
                         Selected(button);
                         display(button, POS_X_BUTTON, POS_Y_BUTTON, ecran);
                         }
```

```
                        }//end if
                    break;
                case SDL_MOUSEBUTTONDOWN:
                    switch (event.button.button) {
                         case SDL_BUTTON_LEFT:
                                                  if (inside) {
                                button->selected = 1;
                               Selected(button);
                               display(button, POS_X_BUTTON, POS_Y_BUTTON, ecran);
                                                  }
                              break;
                         case SDL_BUTTON_RIGHT:
                             button->selected = 0;
                             NoSelected(button);
                            display(button, POS_X_BUTTON, POS_Y_BUTTON, ecran);
                            break;
                    }//end switch event button

                      }// end swithc event.type
              }// end pollEvent
       }

       FreeButton(button);
       SDL_FreeSurface(terre);
       SDL_FreeSurface(ecran);
       SDL_Quit();
       exit(0);
}
```

6 Fichier Makefile

```
all: main

main: main.o Button.o
       gcc -o main main.o Button.o -lSDL
main.o: main.c Button.h
       gcc -c main.c -lSDL
Button.o: Button.c Button.h
       gcc -c Button.c -lSDL
```

6. Code de l'objet Map

1 Fichier Map.h

```
#ifndef MAP_H
#define MAP_H

#include <stdio.h>
#include <SDL2/SDL.h>

#include "Screen.h"

#define SIZE_MAP 128

typedef struct {
 char file_image_tiles[80];
 char file_map_txt[80];
 char array_map[SIZE_MAP][SIZE_MAP];
       int nb_column;
       int nb_row;
}Map;

Map *new_map(char file_image_tiles[80], char file_map_txt[80]);
int LoadMap(Map *map);
void DisplayMap(Map *map);
void DisplayMapSDL(Map *map, SDL_Renderer *renderer);

#endif // MAP_H
```

2 Fichier Map.c

```
#include "Map.h"
```

```
#include <string.h>


Map *new_map(char file_image_tiles[80], char file_map_txt[80]) {
 Map *map;

 map = (Map*) malloc( sizeof(Map) );
        strcpy(map->file_image_tiles, file_image_tiles);
        strcpy(map->file_map_txt, file_map_txt);
        map->nb_row = 0;
        map->nb_column = 0;
        return map;
}//end Map

int LoadMap(Map *map) {
        int i=0, j=0;
        char tile;
        printf("Fichier image : %s\n", map->file_image_tiles);
        printf("Fichier map : %s\n", map->file_map_txt);

    FILE *fileMap;
        fileMap = fopen(map->file_map_txt, "r");

        while ( (tile=getc(fileMap)) != EOF ) {
//              printf("Caractere %i %i: %c\n", j, i, tile);
                map->array_map[i][j] = tile;
                                i++;
                map->nb_column++;
                if ( tile == '\n' ) {
                        j++;
                        map->nb_row++;
                        map->nb_column--; //delete return => \n
                        i=0;
                }//end if \n
        }//end while !=EOF

    fclose(fileMap);
        map->nb_column = (int)  ( map->nb_column / map->nb_row);
        return 1;
} //end LoadMap

void DisplayMap(Map *map) {
   char read='a';
        int i=0, j;
        for ( j = 0; j < map->nb_row; j++) {
        while ( read != '\n' ) {
                read = map->array_map[i][j];
                printf("%c", map->array_map[i][j]);
                i++;
        }//end while
        i=0;
        read='a';
        }//end for
}//end DisplayMap

void DisplayMapSDL(Map *map, SDL_Renderer *renderer) {
        SDL_Surface *map_image;
        SDL_Texture *texture;
        SDL_Rect Src;
        SDL_Rect Dest;
        char read='a';
        int i_read;
        int i=0, j;

        Src.x = 0;
        Src.y = 0;
        Src.w = SIZE_BMP;
        Src.h = SIZE_BMP;
        Dest.w = SIZE_BMP;
        Dest.h = SIZE_BMP;
        map_image = SDL_LoadBMP(map->file_image_tiles);
        if (map_image) {
                texture = SDL_CreateTextureFromSurface(renderer, map_image);
                for ( j = 0; j < map->nb_row; j++) {
```

```
                while ( read != '\n' ) {
                        read = map->array_map[i][j];
                        i_read = read - 0;
                        if ( (i_read > 31) && (i_read < 44) ) { i_read = read - 32; Src.y = 0; }
                        if ( (i_read > 43) && (i_read < 56) ) { i_read = read - 44; Src.y = SIZE_BMP; }
                        if ( (i_read > 55) && (i_read < 68) ) { i_read = read - 56; Src.y = 2*SIZE_BMP; }
                        if ( (i_read > 67) && (i_read < 80) ) { i_read = read - 68; Src.y = 3*SIZE_BMP; }
                        if ( (i_read > 79) && (i_read < 92) ) { i_read = read - 80; Src.y = 4*SIZE_BMP; }
                        if ( (i_read > 91) && (i_read < 104) ) { i_read = read - 92; Src.y = 5*SIZE_BMP; }
                        if ( (i_read > 103) && (i_read < 116) ) { i_read = read - 104; Src.y = 6*SIZE_BMP; }
                        if ( (i_read > 115) && (i_read < 128) ) { i_read = read - 116; Src.y = 7*SIZE_BMP; }

        //              printf("i_read = %i\n", i_read);
                        Src.x = i_read*SIZE_BMP;
                        Dest.x = i*SIZE_BMP;
                        Dest.y = j*SIZE_BMP;

                        SDL_RenderCopy(renderer, texture, &Src, &Dest);
        //              printf("%c", map->array_map[i][j]);
                        i++;
                }//end while
                i=0;
                read='a';
                }//end for
                SDL_DestroyTexture(texture);
                SDL_FreeSurface(map_image);
        } else {
                printf("Error Load tiles image!\n");
        }//end if map_image
}//end DisplayMapSDL
```

3 Fichier main.c

```
#include <stdio.h>

#include "Screen.h"
#include "Map.h"


int main(int argc, char *argv[]) {
        SDL_Window *window;
    SDL_Renderer *renderer;
        SDL_Surface *sprite;
    SDL_Texture *texture;
    SDL_Rect position;
        SDL_Event event;
        Map *map;
        int quit=0;
    int i, j, nb_horiz, nb_vert;



        if (SDL_VideoInit(NULL) < 0) {
         SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't initialize SDL video: %s\n",
                 SDL_GetError());
         exit(1);
    }

        window = SDL_CreateWindow("Fenetre", SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED,
                                  WIDTH_SCREEN, HEIGHT_SCREEN,
                                SDL_WINDOW_SHOWN);

 renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED);

        map = new_map("images/Map.bmp", "map/Map.txt");
        LoadMap(map);
        DisplayMap(map);
        DisplayMapSDL(map, renderer);

 //Gestion des evenements
        while ( quit!=1 ) {
                if ( SDL_PollEvent(&event) ) {
                        switch (event.type) {
                                case SDL_QUIT:
```

```
                                    quit=1;
                                    break;
                        }//end switch event.type
                }//end if PollEvent
        }//end while quit


    SDL_DestroyRenderer(renderer);
        SDL_DestroyWindow(window);


        SDL_Quit();

        exit(0);
}//end main
```

Révision

Le 03/06/2016 : ajout du chapitre « Gestion des images », modification dans « Description du terrain de combat », ajout code exemple 2
Le 24/06/2016 : ajout du chapitre « Gestion des menus » et exemple de code de l'objet « Button »
Le 11/08/2016 : modification de Gestion de la carte
Le 18/08/2016 : modification de Gestion de la carte, ajout des chapitres « objets inanimés », « conception carte » et « objets animés »
Le 28/01/2018 : modification de Gestion de la carte, ajout du code map.h, map.c