Le jeu du Snake

Developpé par Evan Heindryckx et Danyil Garabajiu

SAE

Cette SAE consiste à developper un Snake en C89 en utilisant la librairie graphique de l'IUT. Nous avous décidé d'ajouter la règle supplémentaire des obstacles car c'est la règle la plus populaire dans le modding de Snake de base.

Fonctionalités

Menu et game loop

Le jeu commence par un menu. Nous avons choisi une approche assez simple :

Tant que tvb(tout va bien) est de 3, c'est qu'on est dans le menu. On l'assigne donc à la valeur renvoyée par AfficherMenu. Si 0 est renvoyé (Jouer appyué) alors on dessine la fênetre, on initialise les pommes et les murs (regle supplémentaire des obstacles). Si 1 est renvoyé (Quitter appyué) alors on quitte le graphique.

```
while (tvb == 3) {
    tvb = AfficherMenu();
    if (tvb == 0) {
        ChoisirCouleurDessin(green);
        RemplirRectangle(20, 20, 1200, 800);
        InitialiserTailleSerpent(segmentsX, segmentsY, &NOMBRE_SEGMENTS);
        Pomme(pomme_x, pomme_y, segmentsX, segmentsY, NOMBRE_SEGMENTS);
        mur(murx, mury, segmentsX, segmentsY, NOMBRE_SEGMENTS);
    } else {
        if (tvb == 1){
            FermerGraphique();
        }
    }
}
```

```
int AfficherMenu() {
    int selection = 1;
    int tailleTexte = 2;

    while (selection == 1) {
        if (SourisCliquee()) {
            int x_souris = _X;
            int y_souris = _Y;

            if (x_souris >= 400 && x_souris <= 600) {
                if (y_souris >= 300 && y_souris <= 325) {
                     return 0 ;

                } else if (y_souris >= 400 && y_souris <= 425) {
                    return 1;
                }
            }
        }

        ChoisirCouleurDessin(CouleurParComposante(163, 183, 99));
        RemplirRectangle(400, 300, 200, 25);
        RemplirRectangle(400, 400, 200, 25);

        ChoisirCouleurDessin(CouleurParComposante(0, 0, 0));
        EcrireTexte(450, 315, "Jouer", tailleTexte);
        EcrireTexte(450, 415, "Quitter", tailleTexte);

    }

    FermerGraphique();
}
```

Nous déssinons des boutons et vérifions si le joueur clique aux coordonnées qui leur corréspondent.

Le loop principal :

Consiste à tout mettre à jour, vérifier les collisions avec les murs et les pommes.

```
while(tvb == 0 && verifcollision != 2){
    verifcollision = Deplacement(segmentsX, segmentsY, NOMBRE_SEGMENTS, direction);
    Serpent(segmentsX,segmentsY,NOMBRE_SEGMENTS);
    Keys(&direction);
    Score(NOMBRE_SEGMENTS);
    Timer(temps,&minute, &seconde, &ancienne_seconde, &seconde_actuel);
    if (Collisions_Murs(murx, mury, segmentsX, segmentsY, NOMBRE_SEGMENTS) == 2){
        verifcollision = 2;
    }
    if (Collisions_Pommes(pomme_x, pomme_y, segmentsX, segmentsY) == 4 ){
        NOMBRE_SEGMENTS ++;
    }
}
```

Si la collision avec les murs est vérifée, alors on arrête la boucle.

Si la collision avec les pommes est vérifée, alors on augmente le nombre de ségments du serpent.

Si une collision est vérifiée, alors le joueur est mort. On fait donc un écran de mort, et tant que mort est à 1, on attend une action du joueur (appyuer sur rejouer).

```
if(verifcollision==2){
    mort=1;
    sprintf(score,"%d",((NOMBRE_SEGMENTS - 10) *5));
    EffacerEcran(black);
    EcrireTexte(570,300,"Perdu!",2);
    EcrireTexte(400,440,"Temps final :",2);
    EcrireTexte(400,470,temps,2);
    EcrireTexte(650,440,"Score final :",2);
    EcrireTexte(650,470,score,2);
    ChoisirCouleurDessin(blanc);
    RemplirRectangle(550, 700, 170, 25);
    ChoisirCouleurDessin(red);
    EcrireTexte(570,720,"Rejouer?",2);

    while(mort==1){
        if (SourisCliquee()) {
            x_souris = _X;
            y_souris = _Y;
            if ((x_souris>=550) && (x_souris<=720)){
                if ((y_souris>=700) && (y_souris<=725)){
                    FermerGraphique();
                    main();
                }
            }
        }
    }
}
```

Fermer le graphique ici est important car sinon on se fait engueuler par le terminal (donc probablement par Luc),

```
une seule fentre on a dit !!!!
```

et car sinon on ouvre plusieurs instances du graphique sans utilité. On call main pour redémarrer le jeu.

Serpent et son déplacement

Le serpent est initialisé avec une fonction InitialiserTailleSerpent (on dessine tous les ségments)

```
void InitialiserTailleSerpent(int *segmentsX, int *segmentsY, int *NOMBRE_SEGMENTS){
        int i;
        for (i = 0; i < 10; i++){
                segmentsX[i] = 600 - i * 20;
                segmentsY[i] = 400;
            *NOMBRE_SEGMENTS = 10;
        }
}
```

et son déplacement automatique se fait par un système de directions (up, down, left, right). Il avance dans sa direction (=orientation) actuelle.

```
int Deplacement(int *segmentsX, int *segmentsY, int NOMBRE_SEGMENTS, char direction){
        int i;
        for (i = NOMBRE_SEGMENTS -1; i>0; i--){
                segmentsX[i] = segmentsX[i - 1];
                segmentsY[i] = segmentsY[i -1];
        }

        if (direction=='u'){
                segmentsY[0] -= 20;
        } else if (direction=='d'){
                segmentsY[0] += 20;
        } else if (direction=='r'){
                segmentsX[0] += 20;
        } else if (direction=='l'){
                segmentsX[0] -= 20;
        }
        Attendre(100000);
        return Check_Collisions(segmentsX, segmentsY, NOMBRE_SEGMENTS);
}
```

Suite à quoi on vérifie les collisions => on vérifie qu'après le déplacement du serpent nous n'allons pas percuter le mur ou la queue.

```
int Check_Collisions(int *segmentsX, int *segmentsY, int NOMBRE_SEGMENTS){
	int i;
	if (segmentsX[0] < 40 || segmentsX[0] > 1180 || segmentsY[0] < 40 || segmentsY[0] > 780){
		return 2;
	}

	for (i = 1 ; i <NOMBRE_SEGMENTS ; i++) {
		if (segmentsX[0] == segmentsX[i] && segmentsY[0] == segmentsY[i]){
			return 2;
		}
	}
	return 0;
}
```

La direction est définie dans une fonction Keys

```
void Keys(char *direction){
	if(ToucheEnAttente()){
		int touche = Touche();
		switch(touche){
			case XK_Left :
				if(*direction!='r'){
					*direction='l';
				}
				break;
			case XK_Right:
				if(*direction!='l'){
					*direction='r';
				}
				break;
			case XK_Up:
				if(*direction!='d'){
					*direction='u';
				}
				break;
			case XK_Down:
				if(*direction!='u'){
					*direction='d';
				}
				break;
		}
	}
}
```

Le code vérifie si le changement de la direction est possible en fonction de la touche appyuée (si le serpent ne se rentrera pas dedans si on change de direction). Si le changement est possible, alors on le fait.

Après cela, la deplacement est traité dans la fonction précédement citée.

Le code qui sert à déssiner le serpent est le suivant ;

```
void Serpent(int *segmentsX, int *segmentsY, int NOMBRE_SEGMENTS){
        int i;
        couleur yellow = CouleurParComposante(255,255,204);
        couleur green = CouleurParComposante(163, 183, 99);
        ChoisirCouleurDessin(yellow);
        for (i = 0; i<NOMBRE_SEGMENTS; i++){
                RemplirRectangle(segmentsX[i],segmentsY[i],20,20);
        }
        ChoisirCouleurDessin(green);
        RemplirRectangle(segmentsX[NOMBRE_SEGMENTS - 1], segmentsY[NOMBRE_SEGMENTS - 1], 20 ,20);
}
```

On remplit chaque coordonnée des segments d'un carré jaune. La boucle tourne NOMBRE_SEGMENTS fois.

Les pommes

Sachant que time n'était pas une bibliothèque autorisée, nous avons opté pour un équivalent :

utiliser le nombre de microsecondes passées depuis le début du programme en tant que seed pour le srand.

Ensuite, 5 pommes sont générées aléatoirement en vérifiant bien que la pomme n'est pas générée dans le serpent, auquel cas on en regenere une autre.

Après ces vérifications, nous pouvons déssiner notre pomme.

```
void Pomme(int *pomme_x,int *pomme_y,int *segmentsX,int *segmentsY,int NOMBRE_SEGMENTS){
        int i, p;
        srand((Microsecondes())/1000000);
        for( p = 0; p < 5; p++){
                pomme_x[p] = ((rand() % (58)+1)*20);
                pomme_y[p] = ((rand() % (38)+1)*20);

                for(i=1;i<NOMBRE_SEGMENTS;i++){
                        if(&pomme_x[p]==&segmentsX[i] && pomme_y[p]==segmentsY[i]){
                                pomme_x[p] = ((rand() % (58)+1)*20);
                                pomme_y[p] = ((rand() % (38)+1)*20);
                                i = -1;
                        }
                }
                ChoisirCouleurDessin(CouleurParComposante(240,15,15));
                RemplirRectangle(pomme_x[p],pomme_y[p], 20 ,20);

        }
}
```

La vérification de la collision avec les pommes se fait dans le même fichier. On vérifie tout simplement si le ségment 0 donc la tête rentre en collision avec une pomme, auquel cas on en génere une autre et on return 4 (sert à augmenter le nombre de ségments).

```
int Collisions_Pommes(int *pomme_x, int *pomme_y, int *segmentsX, int *segmentsY){
        int i=0;

        for(i=0; i<5; i++){
                if((pomme_x[i]==segmentsX[0]) && (pomme_y[i]==segmentsY[0])){
                        pomme_x[i] = ((rand() % (58)+1)*20);
                        pomme_y[i] = ((rand() % (38)+1)*20);
                        ChoisirCouleurDessin(CouleurParComposante(240,15,15));
                        RemplirRectangle(pomme_x[i],pomme_y[i], 20 ,20);
                        return 4;

        }

        }
}
```

Score et Timer

Le score est défini simplement en fonction du nombre de ségments. Un ségment (sans les 10 premiers) est égal à 5 de score.

```
void Score(int NOMBRE_SEGMENTS){
    char score[4];
    int nombre = ((NOMBRE_SEGMENTS - 10) *50);
    snprintf(score,4,"%04d", nombre);
    ChoisirCouleurDessin(CouleurParComposante(0,0,0));
    RemplirRectangle(1100,830,1200,900);
    ChoisirCouleurDessin(CouleurParComposante(255,255,255));
    EcrireTexte(1100,850,"Score",2);
    EcrireTexte(1100,880,score,2);
}
```

Pas grande chose à dire sur le timer. Si la seconde a avancé, on met à jour la variable seconde. Si on est à 60 secondes, alors on réintialise les secondes et on incrémente les minutes.

```c
void Timer(char temps[6], int *minute, int *seconde, int *ancienne_seconde, int *seconde_actuel){
    *seconde_actuel = Microsecondes()/1000000;
    if (*seconde_actuel != *ancienne_seconde){
        *seconde = *seconde + 1;
        *ancienne_seconde = *seconde_actuel;
    }
    if (*seconde == 60){
        *seconde = 0;
        *minute = *minute + 1;
    }
    snprintf(temps,6,"%02d:%02d", *minute, *seconde);
    ChoisirCouleurDessin(CouleurParComposante(0,0,0));
    RemplirRectangle(0,830,900,900);
    ChoisirCouleurDessin(CouleurParComposante(255,255,255));
    EcrireTexte(10,880,"Temps :",2);
    EcrireTexte(130,880,temps,2);
}
```

Obstacles (règle supplémentaire)

Les obstacles sont éssentiellement des pommes avec l'effet de faire perdre à la place d'incrémenter le score et la taille.
Nous en initialisons 25 en faisant attention de ne pas en créer dans le serpent.

Nous vérifions la collision avec les murs de la même manière qu'avec les pommes :

si la tête, ségment 0, a les mêmes coordonnées qu'un mur, alors il y a collision.

```
void mur(int *murx, int *mury, int *segmentsX, int *segmentsY, int NOMBRE_SEGMENTS){
    srand((Microsecondes())/1000000);
    int i;
    for(i = 0; i < 25; i++)
    {
        murx[i] = ((rand() % (58) + 1) * 20);
        mury[i] = ((rand() % (38) + 1) * 20);
        ChoisirCouleurDessin(CouleurParComposante(50,50,50));
        RemplirRectangle(murx[i], mury[i], 20, 20);
    }
    for(i=1;i<NOMBRE_SEGMENTS;i++){
                        if(&murx[i]==&segmentsX[i] && mury[i]==segmentsY[i]){
                            murx[i] = ((rand() % (58)+1)*20);
                                mury[i] = ((rand() % (38)+1)*20);
                                i = -1;
                        }
}
}

int Collisions_Murs(int *murx, int *mury, int *segmentsX, int *segmentsY, int NOMBRE_SEGMENTS){
    int i;
    for(i = 0; i < 25; i++){
        if((murx[i] == segmentsX[0]) && (mury[i] == segmentsY[0])){
            printf("Collision mur\n");
            return 2;

        }
    }
}
```

Conclusion:

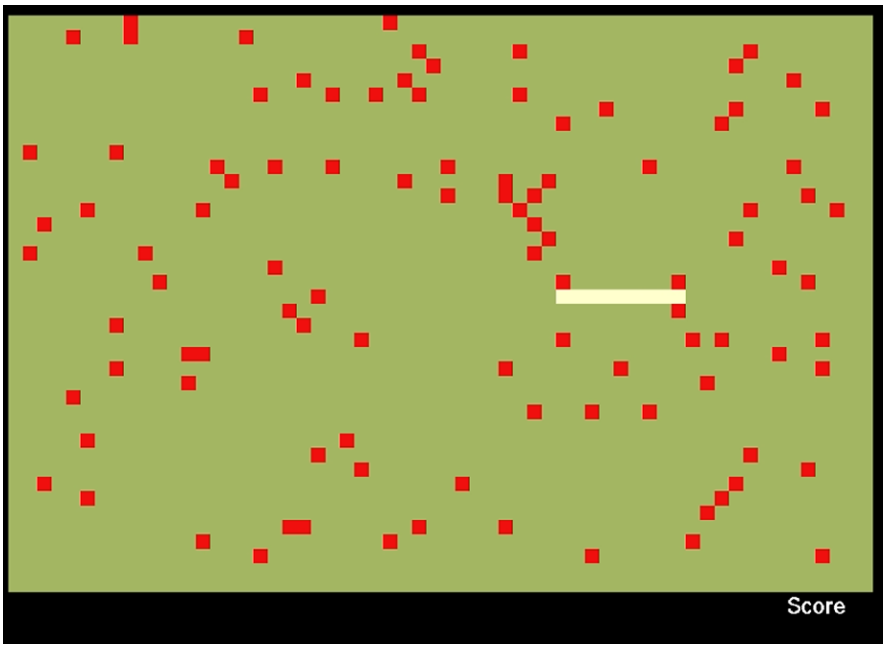
Evan :

Ce projet m'a permis d'apprendre à utiliser une bibliothèque graphique et à bien organiser mes fichiers en fonction de leur utilité. C'était intéressant d'apprendre cet aspect technique. Mais j'ai

eu du mal avec la partie visuelle pour rendre le jeu beau, et la gestion des fichiers pour éviter les erreurs a été difficile. Grâce à ce projet j'ai donc acquis beaucoup de compétences qui me semblent très utiles dans le développement d'un jeu.

Danyil :

J'ai beaucoup aimé cette SAE. Le sujet est cool et tout le monde connaît le principe du Snake donc on avait une idée génerale de notre code même avant le début de la programation. J'ai également pu comprendre l'utilité et utilisation des adresses (point faible personnel jusqu'ici). Le processus de la programation était également pas désagréable ; sujet un peu difficile sans être insurmontable. Deux petits bugs nous ont fait rire et on en a fait des captures d'écran :



La boucle pour initialiser les 5 pommes tournait à chaque update, créant un festin pour notre serpent

Le score se mettait à jour mais ça ne le rendait pas pour autant efficace. (pas de capture d'écran, mais à un moment il affichait les caractères ASCII en sautant 5 positions à chaque fois).