Durant les semaines précédents les vacances de Noël et durant cette dernière, il m'a été donné comme travail de programmer dans le langage C++ le jeu du Snake. Dans cette optique, voici le rendu du programme contenant les explications du programme :

1 – Initialisation du jeu

Avec la fonction « Initialiser », nous donnons les caractéristiques du jeu dans un premier temps. Après l'avoir lancé le jeu, on donne les coordonnés la tête du serpent, tout le temps au milieu du jeu avec le corps en dessous, et donnons une coordonnée générée aléatoirement dans le jeu au fruit. Par défaut, quand le jeu se lance, le serpent avancera vers le haut. Bien sûr, on initialise le score à 0. Avec cela, nous avons le nécessaire pour afficher le jeu.

```
jeu initialiser(){
    jeu j;
    //Lance le jeu
    j.jeuEnCours = true;
    //Met la tete du serpent au milieu
    j.teteX = LARGEUR / 2;
    j.teteY = HAUTEUR / 2;
    //Met le score a 0
    j.score = 0;
    //Defini la taille de depart du serpent
    j.tailleQueue = 2;
    //Direction initiale
    j.d = HAUT;
    // Positionne aleatoirement le fruit dans le jeu
    j.f.fruitX = rand() % (LARGEUR - 2) + 1;
    j.f.fruitY = rand() % (HAUTEUR - 2) + 1;
    // Positionne la queue
    for (int i = 0; i < j.tailleQueue; i++) {
        j.queueX[i] = j.teteX;
        j.queueY[i] = j.teteY + i + 1;
    }

    return j;
}
```

2 – L'affichage du jeu

Permettant d'avoir un visuel du jeu, la fonction « dessiner » est là justement pour permettre au joueurs de voir le contenu. Pour cela, il aura été nécessaire de créer une matrice de taille LARGEUR x HAUTEUR avant de le remplir avec des caractères en déclarant la variable i et k.
le caractère « » étant par défaut, on ajoute néanmoins plusieurs cas comme les bords du jeu, l'apparition du fruit dans des coordonnés aléatoires (générer la fonction initialiser) ainsi que la tête du serpent (la position aussi initialiser).
Aussi, pour créer la queue, on créer une variable « QueueBool » et « l » en faisant bien attention en ajoutant un if excluant le cas où les coordonées de la queue soit la même que celui de la tête pour que ce dernier ne soit pas remplacé.
Finalement, on affiche le score actuel, géré dans une autre fonction.

```
142     //Affiche le SNAKE
143     for (i = 0; i < HAUTEUR; i++) {
144         for (k = 0; k < LARGEUR; k++) {
145             //Caractere par defaut
146             c = ' ';
147             //Cree les bords
148             if (i == 0 || i == HAUTEUR - 1 || k == 0 || k == LARGEUR - 1) {
149                 c = '#';
150             }
151             //Cree le fruit
152             else if (k == j.f.fruitX && i == j.f.fruitY){
153                 c = 'X';
154             }
155             //Cree la tete du serpent
156             else if (k == j.teteX && i == j.teteY) {
157                 c = 'O';
158             }
159             //Cree la queue du serpent
160             else {
161                 for (l = 0; l < j.tailleQueue; l++) {
162                     if(j.queueX[i] == j.f.fruitX && j.queueY[i] == j.f.fruitY){
163                         QueueBool = false;
164                     }
165                     if (j.queueX[l] == k && j.queueY[l] == i) {
166                         c = 'o';
167                         break;
168                     }
169                 }
170             }
171             //Si aucun des cas ne sont respecte, alors c reste ' ' et affiche donc le vide.
172             cout << c;
173         }
174         //Pour chaque ligne
175         cout << "\n";
176     }
177 
178     //Affichage du score
179     cout << "Fruit collecté : " << j.score << "\n";
180     return 0;
```

3 – déplacer le serpent

Afin de déplacer le serpent, plusieurs commandes auront été nécessaires. Il faudra dans un premier temps récupérer les touches de clavier avec « userInput ».

```
int userInput(){
    struct termios oldt, newt;
    int ch;
    int oldf;

    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
    fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);

    ch = getchar();

    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    fcntl(STDIN_FILENO, F_SETFL, oldf);

    if (ch != EOF) {
        ungetc(ch, stdin);
        return 1;
    }

    return 0;
}
```

Par la suite, avec entrerDirection, associez 4 touches du clavier aux quatres directions que pouvait entreprendre le snake en veillant avec des informations de pouvoir prendre une direction opposé à celle de base.

```
direction entrerDirection(direction d){
    system("stty raw");
    char key;
    //Demande de definir la direction du serpent
    std::cin >> key;
    system("stty sane");

    // Differente direction
    switch (key){
        case 'z':
            // "If" permet d'eviter au serpent d'aller a l'opposee
            if (d != BAS) {
                return HAUT;
            }
            break;
        case 's':
            if (d != HAUT) {
                return BAS;
            }
            break;
        case 'q':
            if (d != DROITE) {
                return GAUCHE;
            }
            break;
        case 'd':
            if (d != GAUCHE) {
                return DROITE;
            }
            break;
        default:
            return d;
    }

    return d;
}
```

Puis enfin d'éxécuter la tâche qu'est de déplacer le corps et la tête dans les directions associées à la touche en veillant bien à ce qu'aucun élément ne disparaissent. Tout cela se retrouve dans la fonction « deplacer ».

```
jeu deplacer(jeu j, int queueX[], int queueY[]){

    // Déclaration variable

    int i;
    int tempX, tempY;
    int ancienneTeteX = j.teteX;
    int ancienneTeteY = j.teteY;
    int ancienneQueueX[MAX_QUEUE_SIZE];
    int ancienneQueueY[MAX_QUEUE_SIZE];
    bool QueueBool = true;

    for (i = 0; i < j.tailleQueue; ++i) {
        ancienneQueueX[i] = j.queueX[i];
        ancienneQueueY[i] = j.queueY[i];
    }

    switch (j.d){
        case HAUT:
            // Deplacement vers le HAUT
            j.teteY--;
            break;
        case BAS:
            // Deplacement vers le BAS
            j.teteY++;
            break;
        case GAUCHE:
            // Deplacement vers la GAUCHE
            j.teteX--;
            break;
        case DROITE:
            // Deplacement vers la DROITE
            j.teteX++;
            break;
        default:
            break;
    }
```

```
    // MAJ
    for (i = 0; i < j.tailleQueue; ++i){
        tempX = j.queueX[i];
        tempY = j.queueY[i];
        j.queueX[i] = ancienneTeteX;
        j.queueY[i] = ancienneTeteY;
        ancienneTeteX = tempX;
        ancienneTeteY = tempY;
    }

    return j;
}
```

4 – évolution de la queue.

Toujours dans la fonction « déplacer », lorsque la tête du serpent se retrouve aux mêmes coordonnées que le fruit, le score augmente tandis que l'on repositionne le fruit à une autre coordonnée, en veillant à ce que le fruit n'apparaisse pas malencontreusement sur la position de la tête le corps du serpent, ou bien même au bord voir en dehors.

```
    // Mange le fruit ?
if (j.teteX == j.f.fruitX && j.teteY == j.f.fruitY) {
    // Augmentation du score
    j.score = j.score + 1;
    // Comparaison
    if (j.tailleQueue < MAX_QUEUE_SIZE) {
        j.tailleQueue++;
    }
    // Nouvelle position fruit
    j.f.fruitX = rand() % (LARGEUR - 2) + 1;
    j.f.fruitY = rand() % (HAUTEUR - 2) + 1;
    while ((j.f.fruitX == j.teteX && j.f.fruitY == j.teteY) || QueueBool == false ){
        j.f.fruitX = rand() % (LARGEUR - 2) + 1;
        j.f.fruitY = rand() % (HAUTEUR - 2) + 1;
    }
}
```

5 – Cas de « Game over »

Il existe deux moyens de perdre dans ce jeu :
- en touchant le bord.
- en mordant sa propre queue.

Ces deux cas sont encore une fois gérer dans la fonction déplacer. Pour ça, il suffissait juste de comparer les coordonnées de la tête du serpent avec les coordonnées des bords du jeu ou bien du corps du serpent. Par ailleurs, dans le cas où cela arrivait, la déclaration « j.jeuEnCours » prenait la valeur booleennes « False », cessant le jeu.

```
// Collision bords
if (j.teteX <= 0 || j.teteX >= LARGEUR - 1 || j.teteY <= 0 || j.teteY >= HAUTEUR - 1) {
    j.jeuEnCours = false;
    return j;
}

// Collision corps
for (i = 0; i < j.tailleQueue; ++i) {
    if (j.teteX == j.queueX[i] && j.teteY == j.queueY[i]) {
        j.jeuEnCours = false;
        return j;
    }
}
```

6 – éxécution des fonctions

Ainsi, par le biais de « Main », toutes les fonctions cités plus haut sont éxécutés. Avec usleep, l'affichage du jeu est à chaque fois mis à jour, adaptant et affichant les nouvelles coordonnées des différents éléments du jeu. En rajoutant par ailleurs une soustraction « (TEMPS – (j.score*2500)), on permet d'accélérer les actualisation à mesure que l'on gagne des points, donnant alors l'impression que le jeu va plus vite.

```
int main() {
    jeu j;
    srand(time(NULL));
    j = initialiser();

    // Permet d'actualiser l'ecran
    while (j.jeuEnCours) {
        dessiner(j, j.queueX, j.queueY);
        if (userInput()) {
            j.d = entrerDirection(j.d);
        }
        j = deplacer(j, j.queueX, j.queueY);
        usleep(TEMPS-(j.score*2500));
    }

    return 0;
}
```