

Jeu Snake en C++

BOUCHE Hugo
BAGNEGUE-MVOUREBIA Christ-bruxie
KOFFI Carl-christopher

Sommaire:

- I. Introduction
- II. Technologies utilisées
- III. Base du jeu
 1. Le serpent
 2. Les obstacles
 3. La zone de jeu
 4. Les collisions
- IV. Personnalisation du jeu
 1. Les touches
 2. L'esthétique
 3. La vitesse du serpent
 4. Meilleur score de la session
 5. Options de contrôles

I. Introduction

Qu'est-ce que Snake ?

- Un jeu classique où un serpent se déplace pour attraper des fruits tout en évitant les obstacles et les murs.
- Le serpent grandit et sa vitesse augmente à chaque fruit mangé augmentant la difficulté du jeu.

Objectif du Jeu :

But du Jeu :

- Diriger le serpent pour attraper un maximum de fruits.
- Éviter les collisions avec les obstacles, les murs et la queue du serpent.

Condition de Fin :

- La partie se termine si le serpent touche un obstacle, un mur ou sa propre queue.

II. Technologies utilisées:

Langages de programmation : C++

Bibliothèques:

<iostream> : Entrée et sortie de texte

<conio.h> : Gestion des entrées clavier

<windows.h> : Fonctions Windows pour la gestion de l'affichage et des pauses

III. Base du jeu

1. Le serpent:

Créer le serpent et le faire grandir à chaque fruit manger

Placer la tête du serpent:

```
20 void Setup() {  
21     dir = STOP;  
22     x = width / 2;  
23     y = height / 2;  
24     score = 0;  
25     nTail = 0;  
26     gameOver = false;  
27     paused = false;  
28 }
```

La tête dirige le serpent:

```
167 void Algorithm() {  
168     int prevX = tailX[0];  
169     int prevY = tailY[0];  
170     int prev2X, prev2Y;  
171     tailX[0] = x;  
172     tailY[0] = y;  
173 }
```

Mise à jour des positions de chaque segment de la queue:

```
174  
175  
176  
177  
178  
179  
180  
181  
for (int i = 1; i < nTail; i++) {  
    prev2X = tailX[i];  
    prev2Y = tailY[i];  
    tailX[i] = prevX;  
    tailY[i] = prevY;  
    prevX = prev2X;  
    prevY = prev2Y;  
}
```

Augmenter la taille du serpent à chaque fruit manger:

```
199  
200  
201  
if (x == fruitX && y == fruitY) {  
    score += 10;  
    nTail++;  
}
```

III. Base du jeu

2. Les obstacles:

Placer des obstacles qui se déplacent aléatoirement dans la zone du jeu

```
14 int obstacleX, obstacleY, obstacleX2, obstacleY2, obstacleX3, obstacleY3;
```

```
30 obstacleX = rand() % width;
31 obstacleY = rand() % height;
32 do {
33     obstacleX2 = rand() % width;
34     obstacleY2 = rand() % height;
35 } while ((obstacleX2 == obstacleX && obstacleY2 == obstacleY) || (obstacleX2 == x && obstacleY2 == y));
36
37 do {
38     obstacleX3 = rand() % width;
39     obstacleY3 = rand() % height;
40 } while (((obstacleX3 == obstacleX && obstacleY3 == obstacleY) || (obstacleX3 == obstacleX2 && obstacleY3 == obstacleY2)) ||
41         (obstacleX3 == x && obstacleY3 == y));
```

III. Base du jeu

2. Les obstacles;

```
147 void moveObstacle(int& obsX, int& obsY) {
148     int direction = rand() % 4; // 0: UP, 1: DOWN, 2: LEFT, 3: RIGHT
149
150     switch (direction) {
151     case 0:
152         obsY--;
153         break;
154     case 1:
155         obsY++;
156         break;
157     case 2:
158         obsX--;
159         break;
160     case 3:
161         obsX++;
162         break;
163     default:
164         break;
165     }
166
167     // Garder les obstacles dans la zone de jeu
168     if (obsX < 0) obsX = 0;
169     if (obsX >= width) obsX = width - 1;
170     if (obsY < 0) obsY = 0;
171     if (obsY >= height) obsY = height - 1;
172 }
```

```
void MoveObstacles() {
    // déplacer chaque obstacle
    moveObstacle(obstacleX, obstacleY);
    moveObstacle(obstacleX2, obstacleY2);
    moveObstacle(obstacleX3, obstacleY3);
}
```


III. Base du jeu

3. La zone de jeu:

Délimiter les dimensions de la zone de jeu

```
9  const int width = 40;  
10 const int height = 20;
```

III. Base du jeu

4. Les collisions:

Collisions avec les murs de la zone de jeu:

```
191 | if (x < 0 || x >= width || y < 0 || y >= height)
192 |     gameOver = true;
```

Collisions avec le corps du serpent (auto-collision):

```
194 | for (int i = 0; i < nTail; i++) {
195 |     if (tailX[i] == x && tailY[i] == y)
196 |         gameOver = true;
197 | }
```

Collisions avec les obstacles:

```
212 | if ((x == obstacleX && y == obstacleY) ||
213 |     (x == obstacleX2 && y == obstacleY2) ||
214 |     (x == obstacleX3 && y == obstacleY3))
215 |     gameOver = true;
216 | }
```

IV. Personnalisation du jeu:

1. Les touches de contrôle:

Pour jouer, mettre en pause et quitter

```
108 void Input() {
109     if (_kbhit()) {
110         char key = _getch();
111         switch (key) {
112             case 'q':
113                 dir = LEFT;
114                 break;
115             case 'd':
116                 dir = RIGHT;
117                 break;
118             case 'z':
119                 dir = UP;
120                 break;
121             case 's':
122                 dir = DOWN;
123                 break;
124             case 'x':
125                 gameOver = true;
126                 break;
127             case 'p':
128                 paused = !paused;
129                 break;
130         }
131     }
132 }
```

```
231 int main() {
232     char ready;
233     cout << "Bienvenue dans le jeu Snake !" << endl;
234     cout << "Utilisez les touches Z, Q, S, D pour deplacer le serpent." << endl;
235     cout << "Appuyez sur X pour quitter." << endl;
236     cout << "Appuyez sur P pour mettre en pause ou reprendre le jeu." << endl;
237     cout << "Etes vous pret a jouer ? (o/n): ";
238     cin >> ready;
239
240     if (ready != 'o' && ready != 'O') {
241         cout << "Au revoir !" << endl;
242         return 0;
243     }
```

```
Bienvenue dans le jeu Snake !
Utilisez les touches Z, Q, S, D pour deplacer le serpent.
Appuyez sur X pour quitter.
Appuyez sur P pour mettre en pause ou reprendre le jeu.
Etes vous pret a jouer ? (o/n):
```

IV. Personnalisation du jeu:

2. L'esthétique:

Modifier la lettre qui compose le serpent, l'obstacle et les limitations de la zone de jeu:

```
51 void Draw() {
52     system("cls");
53
54     for (int i = 0; i < width + 2; i++)
55         cout << "#";
56     cout << endl;
57
58     for (int i = 0; i < height; i++) {
59         for (int j = 0; j < width; j++) {
60             if (j == 0)
61                 cout << "#";
62
63             if (i == y && j == x)
64                 cout << "O";
65             else if (i == fruitY && j == fruitX)
66                 cout << "o";
67             else if ((i == obstacleY && j == obstacleX) ||
68                    (i == obstacleY2 && j == obstacleX2) ||
69                    (i == obstacleY3 && j == obstacleX3)) |
70                 cout << "X";
71             else {
72                 bool printTail = false;
73                 for (int k = 0; k < nTail; k++) {
74                     if (tailX[k] == j && tailY[k] == i) {
75                         cout << "o";
76                         printTail = true;
77                     }
78                 }
79             }
80         }
81     }
82
83     if (j == width - 1)
84         cout << "#";
85     cout << endl;
86
87     for (int i = 0; i < width + 2; i++)
88         cout << "#";
89     cout << endl;
90
91     cout << "Score:" << score << endl;
92 }
```

```
79
80
81     if (!printTail)
82         cout << " ";
83
84     }
85
86     if (j == width - 1)
87         cout << "#";
88     cout << endl;
89
90     }
91
92     for (int i = 0; i < width + 2; i++)
93         cout << "#";
94     cout << endl;
95
96     cout << "Score:" << score << endl;
97 }
```

IV. Personnalisation du jeu

3. La vitesse du serpent:

Pour augmenter la difficulté du jeu, augmenter la vitesse du score en fonction du score en fixant une vitesse maximale pour que la vitesse du serpent ne soit pas infini.

```
18  const int baseSpeed = 800;
```

```
267  ┌─┐  
268  │  
269  │  
270  │  
271  └─┘  
  
if (baseSpeed - score * 10 < 100) {  
    Sleep(100);  
} else {  
    Sleep(baseSpeed - score * 10);  
}
```

IV. Personnalisation du jeu

4. Meilleur score de la session:

Pour savoir le meilleur score de la session, il se reset à chaque redémarrage du jeu:

```
17 int bestScore = 0;
```

Afficher le meilleure score de la session

```
94 | cout << "Score:" << score << " | Meilleur score de la session: " << bestScore << endl;
```

Mise à jour du meilleur score à la fin de chaque partie

```
251 |         if (score > bestScore) bestScore = score;
```

IV. Personnalisation du jeu

5. Options de contrôle:

Mettre la partie en pause:

```
8 bool paused;
```

```
134 void Pause() {  
135     while (paused) {  
136         if (_kbhit()) {  
137             char key = _getch();  
138             if (key == 'p') {  
139                 paused = false;  
140             }  
141         }  
142         Sleep(100);  
143     }  
144 }  
  
127 case 'p':  
128     paused = !paused;  
129     break;
```

Quitter le jeu:

```
7 bool gameOver;
```

```
124 |  
125 |  
126 |  
--- |  
    |  
    | case 'x':  
    |     gameOver = true;  
    |     break;  
    |  
    |
```

Pouvoir rejouer à chaque fin de partie:

```
97 bool AskToPlayAgain() {  
98     char response;  
99     while (true) {  
100         cout << "Voulez-vous rejouer ? (o/n): ";  
101         cin >> response;  
102         if (response == 'o' || response == 'O') return true;  
103         if (response == 'n' || response == 'N') return false;  
104         cout << "Réponse invalide. Veuillez entrer 'o' pour oui ou 'n' pour non." << endl;  
105     }  
106 }  
  
255 } while (AskToPlayAgain());  
256  
257 cout << "Au revoir et merci d'avoir joué !" << endl;  
258  
259 return 0;  
260 }
```