

SNAKE

BOUCHE Hugo

HERATH-MUDIYANSELAGE Shamal

MAMBOU-GEM Hervy

Sommaire:

1. Base du jeu
2. Personnalisation du jeu
 - I. Les touches
 - II. L'esthétique
 - III. La vitesse du serpent
 - IV. L'obstacle
 - V. Le meilleur Score

The screenshot displays the Dev-C++ IDE interface. The main editor window contains the source code for a file named `snake2.cpp`. The code defines a simple Snake game logic. It includes variables for width, height, score, and fruit positions. The game loop handles user input for moving the snake (up, down, left, right) and checks for collisions with boundaries or itself. When a collision occurs, it prints "Game Over". Otherwise, it moves the snake and checks if it has eaten a fruit, which would increase the score.

```

118     x--;
119     break;
120 case RIGHT:
121     x++;
122     break;
123 case UP:
124     y--;
125     break;
126 case DOWN:
127     y++;
128     break;
129 default:
130     break;
131 }
132
133 if (x >= width) x = 0;
134 else if (x < 0) x = width - 1;
135
136 if (y >= height) y = 0;
137 else if (y < 0) y = height - 1;
138
139 for (int i = 0; i < nTail; i++)
140     if (tail[i] == x && tail[i] == y)
141         gameover = true;
142
143 if (x == fruitX && y == fruitY)
144 {
145     score += 10;
146     fruitX = rand() % width;
147     fruitY = rand() % height;

```

A secondary window titled `C:\Users\Ugo Bouche Motto\Documents\sne.exe` shows the program's output. It consists of a large rectangular field filled with the character '#'. Within this field, there are some other characters: 'F' near the top left, 'O' in the middle, and '-' near the bottom center, likely representing the snake and fruit respectively in a visual representation of the game state.

Personnalisation du jeu:

I. Les touches

Pour jouer et mettre pause:

```
snake2.cpp  sn.cpp
97  }
98
99  void Input()
100 {
101     if (_kbhit())
102     {
103         char key = _getch();
104         switch (key)
105         {
106             case 'q':
107                 dir = LEFT;
108                 break;
109             case 'd':
110                 dir = RIGHT;
111                 break;
112             case 'z':
113                 dir = UP;
114                 break;
115             case 's':
116                 dir = DOWN;
117                 break;
118             case 'x':
119                 gameOver = true;
120                 break;
121             case 'p':
122                 paused = !paused;
123                 break;
124         }
125     }
126 }
```

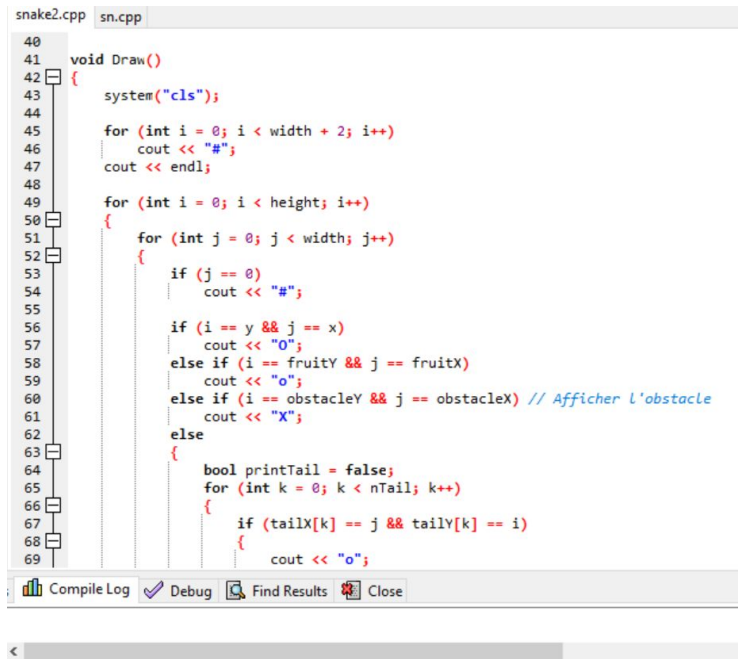
Les affichés dans le menus:

```
snake2.cpp  sn.cpp
238
239 int main()
240 {
241     char ready;
242     cout << "Bienvenue dans le jeu Snake !" << endl;
243     cout << "Meilleur score de la session : " << bestScore << endl;
244
245     cout << "Utilisez les touches Z, Q, S, D pour deplacer le serpent." << endl;
246     cout << "Appuyez sur X pour quitter." << endl;
247     cout << "Appuyez sur P pour mettre en pause ou reprendre le jeu." << endl;
248     cout << "Etes vous pret a jouer ? (o/n): ";
249     cin >> ready;
250
251     if (ready != 'o' && ready != 'O')
252     {
253         cout << "Au revoir !" << endl;
254         return 0;
255     }
256
257     do
258     {
259         Setup();
260         for (int i = 0; i < width + 2; i++)
261             cout << "#";
262         cout << endl;
263
264         Draw();
265
266         while (!gameOver)
267         {
```

Personnalisation du jeu:

II. L'Esthétique

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





```
snake2.cpp  sn.cpp
40
41 void Draw()
42 {
43     system("cls");
44
45     for (int i = 0; i < width + 2; i++)
46         cout << "#";
47     cout << endl;
48
49     for (int i = 0; i < height; i++)
50     {
51         for (int j = 0; j < width; j++)
52         {
53             if (j == 0)
54                 cout << "#";
55
56             if (i == y && j == x)
57                 cout << "O";
58             else if (i == fruitY && j == fruitX)
59                 cout << "o";
60             else if (i == obstacleY && j == obstacleX) // Afficher l'obstacle
61                 cout << "X";
62             else
63             {
64                 bool printTail = false;
65                 for (int k = 0; k < nTail; k++)
66                 {
67                     if (tailX[k] == j && tailY[k] == i)
68                     {
69                         cout << "o";
```

Personnalisation du jeu:

III. La vitesse du serpent

Pour augmenter la difficulté, augmenter la vitesse du serpent en fonction du score:

```
20
21 // Constante pour la vitesse de base
22 const int baseSpeed = 800;
23
```

```
272
273
274
275 
276
277
278
279 
280
281
282
```

```
// Ajustement de la vitesse en fonction du score
if (baseSpeed - score * 10 < 100)
{
    Sleep(100); // Vitesse minimale
}
else
{
    Sleep(baseSpeed - score * 10);
}
```

Personnalisation du jeu:

IV. L'obstacle

Pour augmenter la difficulté, placer un obstacle qui se déplace aléatoirement dans la zone de jeu:

```
14  int obstacleX, obstacleY; // New obstacle variables
```

```
36  // Initialiser la position de l'obstacle de manière aléatoire
```

```
37  obstacleX = rand() % width;
```

```
38  obstacleY = rand() % height;
```

```
60  else if (i == obstacleY && j == obstacleX) // Afficher l'obstacle
```

```
61  cout << "X";
```

```
215 // Check collision with obstacle
```

```
216 if (x == obstacleX && y == obstacleY)
```

```
217     gameOver = true;
```

```
128 void MoveObstacle()
129 {
130     // Move obstacle randomly
131     int direction = rand() % 4; // 0: UP, 1: DOWN, 2: LEFT, 3: RIGHT
132
133     switch (direction)
134     {
135     case 0:
136         obstacleY--;
137         break;
138     case 1:
139         obstacleY++;
140         break;
141     case 2:
142         obstacleX--;
143         break;
144     case 3:
145         obstacleX++;
146         break;
147     default:
148         break;
149     }
150
151     // Check if obstacle is within the game boundaries
152     if (obstacleX < 0)
153         obstacleX = 0;
154     if (obstacleX >= width)
155         obstacleX = width - 1;
156     if (obstacleY < 0)
```

Compile Log Debug Find Results Close

Personnalisation du jeu:

IV. Le Meilleur de score de la session

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

```
18 // Déclaration de la variable pour le meilleur score
19 int bestScore = 0;
20
286 // Mettre à jour le meilleur score à la fin du jeu
287 if (score > bestScore)
288 {
289     bestScore = score;
290     cout << "Nouveau meilleur score : " << bestScore << endl;
291 }
292 else
293 {
294     cout << "Meilleur score actuel : " << bestScore << endl;
295 }
296
```

Proposer de rejouer à chaque fin de partie:

```
91 bool AskToPlayAgain()
92 {
93     cout << "Voulez-vous rejouer ? (o/n): ";
94     char response;
95     cin >> response;
96     return (response == 'o' || response == 'O');
97 }
```