Memulai

Mari kita buat **project** baru di Visual Studio, dengan jenis **Win32 Console Application**. Pertama-tama, include dulu beberapa pustaka yang akan dipakai.

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
4. #include <sys\timeb.h>
5. #include <Windows.h>
```

Array of Struct

Array ini kita gunakan untuk menampung **ular**. Satu elemen pada *array* sama dengan satu segmen ular. Tiap elemen berisi posisi koordinat (x,y) segmen di layar. Berikut ini bentuk strukturnya. Kita beri nama **Segment**.

```
1. /** Struktur **********/
2.
3. /**
4.     Struktur untuk menampung data tiap segment dari
   snake
5.     (array)
6.  */
7. struct Segment {
8.     int x, y;
9. };
```

Kita tambahkan dua variabel global, yaitu *array* **snake**, dan **length** untuk menyimpan panjangnya.

```
1. /** Variabel global **********/
2.
3. // Array untuk menampung data ular
4. struct Segment snake[2000];
5.
6. // Variabel untuk menyimpan panjang ular (array snake)
7. int length = 0;
```

Untuk game ini, kita menggunakan konsep *queue*. Artinya, elemen pada *array* akan ditambahkan di awal (**head**), dan ketika dihapus, yang hilang adalah bagian akhir (**tail**). Istilahnya f*irst in first out*.
Berikut ini fungsi untuk melakukan penambahan **push()** dan penghapusan **pop()**.

```
1. /** Fungsi-fungsi **********/
2.
3. /**
4.     Push segment ke snake (pada bagian head).
5.  */
6. void push(int x, int y) {
7.     for(int i = length; i > 0; i--) {
8.         snake[i] = snake[i-1];
9.     }
10.         snake[0].x = x;
11.         snake[0].y = y;
12.         length++;
13.     }
14.
```

```
    /**
        Pop bagian ekor snake.
     */
    void pop() {
        length--;
    }
```

Ular 3 Segment

Sekarang mari kita coba buat ular sepanjang 3 segmen pada bagian **main()**. Oke, supaya mudah untuk mengubah-ubah pengaturan panjang awalnya, kita simpan nilai 3 tersebut di variabel global **snake_size**. Ketiga segmen ini kita tempatkan di baris pertama (y = 0), di kolom ke 1, 2, dan 3 (x = 0 s.d. 2).

```
/** Konfigurasi permainan **********/

// Panjang segment snake saat awal permainan
int snake_size = 3;

/**
    Program utama
 */
int main() {
        // Pertama-tama, push segment (node) ke kanan
        // sebanyak 3 segment (sesuai nilai variable snake_size)
        for (int i = 0; i < snake_size; i++) {
            push(i, 0);
        }

        return 0;
    }
```

Ayo kita tes sejenak untuk memastikan tidak ada *error* sejauh ini dengan F5.

Rendering

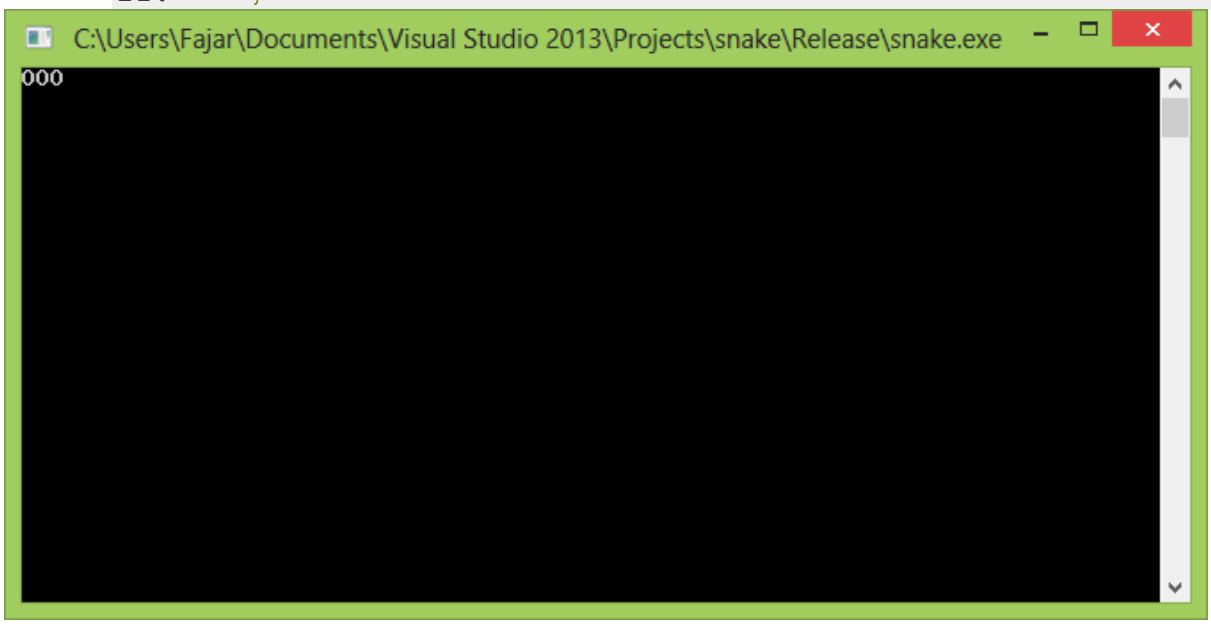
Setelah ular dibuat, kita akan mencetak ular tersebut di layar. Untuk mencetak, kita buat fungsi **display()**. Fungsi **display()** ini akan membaca nilai **x** dan **y** setiap *element* lalu mencetak satu karakter ‘O’ di posisi tersebut.
Untuk bisa mencetak di posisi (x,y), kita harus memindahkan kursor ke posisi tersebut. Untuk itu kita buat juga fungsi **gotoxy()**.

```
/**
    Pindahkan posisi kursor di layar
    Fungsi ini spesifik untuk OS windows.
*/
void gotoxy(int x, int y) {
    COORD pos;
    pos.X = x;
    pos.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), pos);
    }

    /**
```

```
13.         Gambar snake (array) di layar
14.      */
15.     void display() {
16.         for(int i = 0; i < length; i++) {
17.             // Cetak di posisi x,y
18.             gotoxy(snake[i].x, snake[i].y);
19.             printf("O");
20.         }
21.     }
```

Sekarang, mari panggil **display()** di **main()**, jalankan program dan lihat hasilnya.

```
1. /**
2.     Program utama
3.  */
4. int main() {
5.     // Pertama-tama, push segment (node) ke kanan
6.     // sebanyak 3 segment (sesuai nilai variable
   snake_size)
7.     for (int i = 0; i < snake_size; i++) {
8.         push(i, 0);
9.     }
10.
11.         // Tampilkan kondisi permainan saat ini di
   layar...
12.
13.         // Bersihkan layar
14.         system("cls");
15.
16.         // Cetak (render) snake di layar
17.         display();
18.
19.         getchar();
20.         return 0;
21.     }
```



Hore, ular sepanjang 3 segmen berhasil muncul di layar!

Game Loop

Bagaimana caranya agar ular bisa bergerak? Caranya, adalah dengan membuat ***infinite loop*** untuk me-*render* ulang layar setiap putarannya. Dengan demikian, setiap ada perubahan situasi (*state*) pada *array* **snake**, entah itu jumlah *element* (**length**) atau nilai x dan y nya, perubahan itu akan langsung tercermin di layar.
Mari kita taruh bagian rendering tadi ke dalam ***infinite loop***.

```
/**
    Program utama
 */
int main() {
    // Pertama-tama, push segment (node) ke kanan
    // sebanyak 3 segment (sesuai nilai variable snake_size)
    for (int i = 0; i < snake_size; i++) {
        push(i, 0);
    }

        // Game loop. Bagian di dalam while akan dieksekusi terus menerus
        while (true) {
            // Tampilkan kondisi permainan saat ini di layar...

            // Bersihkan layar
            system("cls");

            // Cetak (render) snake di layar
            display();
        }

        getchar();
        return 0;
    }
```

Untuk menggerakkan ular ke kanan setiap 200ms, pertama-tama, di dalam *game loop* kita menghitung berapa waktu yang sudah terlewati, jika waktu yang berlalu sudah lebih atau sama dengan 200ms, maka kita geser ular. Sama dengan sebelumnya, agar nilai 200 ini mudah diubah-ubah, kita simpan dalam variabel global **snake_speed**.

```
// Kecepatan gerakan snake dalam ms
int snake_speed = 200;
```

Untuk menghitung interval waktu yang berlalu, kita gunakan fungsi **ftime()** untuk mendapat kan penanda waktu.
Cara menggeser ular, adalah dengan melakukan **pop()**, lalu **push()** kembali di posisi koordinat **head** dengan nilai x ditambah 1 karena saat ini kepala ular mengarah ke kanan.

```
/**
    Program utama
 */
int main() {

    // Untuk menyimpan penanda waktu saat snake bergerak
    struct timeb last_timestamp;
    ftime(&last_timestamp); // Set nilai awal

```

```
10.         // Pertama-tama, push segment (node) ke kanan
11.         // sebanyak 3 segment (sesuai nilai variable
  snake_size)
12.         for (int i = 0; i < snake_size; i++) {
13.             push(i, 0);
14.         }
15.         // Game loop. Bagian di dalam while akan
  dieksekusi terus menerus
16.         while (true) {
17.             // Ambil penanda waktu saat ini
18.             struct timeb current_timestamp;
19.             ftime(&current_timestamp);
20.
21.             // Selisih waktu terakhir dengan waktu
  sekarang dalam ms
22.             int interval = 1000 *
  (current_timestamp.time - last_timestamp.time) +
  (current_timestamp.millitm - last_timestamp.millitm);
23.
24.             // Snake bergerak setiap 200 ms (sesuai
  nilai variable snake_speed)
25.             // Dihitung dengan membandingkan selisih
  waktu sekarang dengan waktu
26.             // terakhir kali snake bergerak.
27.             if (interval >= snake_speed) {
28.                 // Tentukan posisi x,y ke mana snake
  akan bergerak.
29.                 int x, y;
30.                 x = snake[0].x + 1;
31.                 y = snake[0].y;
32.
33.                 // Pop ekor, lalu push segment ke depan
  head sehingga
34.                 // snake tampak bergerak maju.
35.                 pop();
36.                 push(x, y);
37.
38.                 // Perbarui penanda waktu
39.                 last_timestamp = current_timestamp;
40.             }
41.
42.             // Tampilkan kondisi permainan saat ini di
  layar...
43.
44.             // Bersihkan layar
45.             system("cls");
46.
47.             // Cetak (render) snake di layar
48.             display();
49.         }
50.
51.         ...
52.     }
```

Coba jalankan lagi. Sekarang ular sudah bisa bergerak!

Tapi layar tampaknya berkedip-kedip. Hal ini terjadi karena program mencoba mengosongkan layar dengan **system("cls");** sebelum menggambar lagi. Umumnya pembuat game akan melakukan teknik *double buffering* untuk menghindari layar berkedip (*flicker*). Namun untuk menyederhanakan tutorial ini, kita akan lakukan pendekatan lain, yaitu dengan me-*render* ulang layar hanya ketika ular bergerak. Sehingga *rendering* hanya terjadi setiap 200ms sekali (5 FPS).
Caranya mudah, kita pindahkan baris-baris *rendering* ke dalam blok **if(interval >= snake_speed) { }**.

```
1. /**
2.     Program utama
3.  */
4. int main() {
5.     ...
6.
7.     // Game loop. Bagian di dalam while akan dieksekusi
   terus menerus
8.     while (true) {
9.         // Ambil penanda waktu saat ini
10.            struct timeb current_timestamp;
11.            ftime(&current_timestamp);
12.
13.            // Selisih waktu terakhir dengan waktu
   sekarang dalam ms
14.            int interval = 1000 *
   (current_timestamp.time - last_timestamp.time) +
   (current_timestamp.millitm - last_timestamp.millitm);
15.
16.            // Snake bergerak setiap 200 ms (sesuai
   nilai variable snake_speed)
17.            // Dihitung dengan membandingkan selisih
   waktu sekarang dengan waktu
18.            // terakhir kali snake bergerak.
19.            if (interval >= snake_speed) {
20.                // Tentukan posisi x,y ke mana snake
   akan bergerak.
21.                int x, y;
22.                x = snake[0].x + 1;
23.                y = snake[0].y;
24.
25.                // Pop ekor, lalu push segment ke depan
   head sehingga
26.                // snake tampak bergerak maju.
27.                pop();
28.                push(x, y);
29.
30.                // Tampilkan kondisi permainan saat ini
   di layar...
31.
32.                // Bersihkan layar
33.                system("cls");
```

```
34.
35.                 // Cetak (render) snake di layar
36.                 display();
37.
38.                 // Perbarui penanda waktu
39.                 last_timestamp = current_timestamp;
40.             }
41.         }
42.
43.         ...
44.     }
```

Mengontrol Arah Gerakan Ular

Untuk bisa mengontrol arah gerakan ular, kita membuat sebuah variabel global tambahan bernama **dir**. Variabel ini memberitahu arah **push()** berikutnya, apakah ke kanan, bawah, kiri, atau atas. Arah ini akan ditentukan berdasarkan input tombol panah yang ditekan.
Pertama-tama, buat variabel global **dir**, dengan nilai awal ke arah kanan. VK_RIGHT adalah konstanta berisi kode untuk tombol panah kanan.

```
1. // Arah kepala saat awal permainan
2. int dir = VK_RIGHT;
```

Sekarang kita modifikasi penentuan nilai x dan y untuk melakukan **push()** berdasarkan variabel **dir**. Lalu di dalam *game loop*, dilakukan juga pengecekan tombol yang sedang ditekan. Jika merupakan salah satu dari empat tombol panah di keyboard, maka ubah nilai **dir**.

```
1. /**
2.     Program utama
3.  */
4. int main() {
5.     ...
6.
7.     // Game loop. Bagian di dalam while akan dieksekusi
   terus menerus
8.     while (true) {
9.
10.             ...
11.
12.             // Snake bergerak setiap 200 ms (sesuai
   nilai variable snake_speed)
13.             // Dihitung dengan membandingkan selisih
   waktu sekarang dengan waktu
14.             // terakhir kali snake bergerak.
15.             if (interval >= snake_speed) {
16.                 // Tentukan posisi x,y ke mana snake
   akan bergerak.
17.                 // Posisi dilihat dari koordinat segment
   kepala (head)
18.                 // dan arah (variable dir)
19.                 int x, y;
20.                 switch (dir) {
21.                 case VK_LEFT:
22.                     x = snake[0].x - 1;
23.                     y = snake[0].y;
```

```
                    break;
                case VK_RIGHT:
                    x = snake[0].x + 1;
                    y = snake[0].y;
                    break;
                case VK_UP:
                    x = snake[0].x;
                    y = snake[0].y - 1;
                    break;
                case VK_DOWN:
                    x = snake[0].x;
                    y = snake[0].y + 1;
                    break;
                }

                // Pop ekor, lalu push segment ke depan head sehingga
                // snake tampak bergerak maju.
                pop();
                push(x, y);

                // Tampilkan kondisi permainan saat ini di layar...

                // Bersihkan layar
                system("cls");

                // Cetak (render) snake di layar
                display();

                // Perbarui penanda waktu
                last_timestamp = current_timestamp;
            }

            // Ubah arah jika tombol panah ditekan
            if (GetKeyState(VK_LEFT) < 0) {
                dir = VK_LEFT;
            }
            if (GetKeyState(VK_RIGHT) < 0) {
                dir = VK_RIGHT;
            }
            if (GetKeyState(VK_UP) < 0) {
                dir = VK_UP;
            }
            if (GetKeyState(VK_DOWN) < 0) {
                dir = VK_DOWN;
            }

            // Keluar dari program jika menekan tombol ESC
            if (GetKeyState(VK_ESCAPE) < 0) {
                return 0;
            }
```

```
        }

        ...
    }
```

Kita juga bisa menambahkan pengecekan untuk keluar dari program jika pemain menekan tombol ESC.

Coba jalankan lagi program, sekarang kita bisa menggerakan ular dengan bebas!

Collision Detection

Salah satu aspek yang penting dalam permainan ini adalah pengecekan apakah kepala ular bertabrakan dengan dinding atau dirinya sendiri. Di sini kita bisa melakukan pengecekan saat program memeroleh posisi x dan y yang baru, sebelum melakukan **pop()** dan **push()**.
Jika posisi x berada di luar batasan 0-79 (panjang console) atau posisi y berada diluar batasan 0-24 (tinggi console), maka ular telah menabrak dinding, dan permainan berakhir. Sama seperti sebelum-sebelumnya, untuk nilai panjang dan lebar console bisa kita simpan di variabel global **console_width** dan **console_height**.

```
// Panjang console
int console_width = 80;

// Tinggi console
int console_height = 25;
```

Pengecekan berikutnya yaitu mengecek apabila posisi x dan y sama dengan posisi salah satu *node*, yang artinya ular menabrak dirinya sendiri. Untuk mengeceknya, kita buat fungsi **check_collision()**.

```
/**
    Memeriksa apakah terdapat salah satu segment
    snake (array) di koordinat x,y.
    Return 0 artinya tidak bertumpuk, 1 artinya
  bertumpuk.
 */
int check_collision(int x, int y) {
    for(int i = 0; i < length; i++) {
        if (snake[i].x == x && snake[i].y == y) {
            return 1;
            }
        }
        return 0;
    }
```

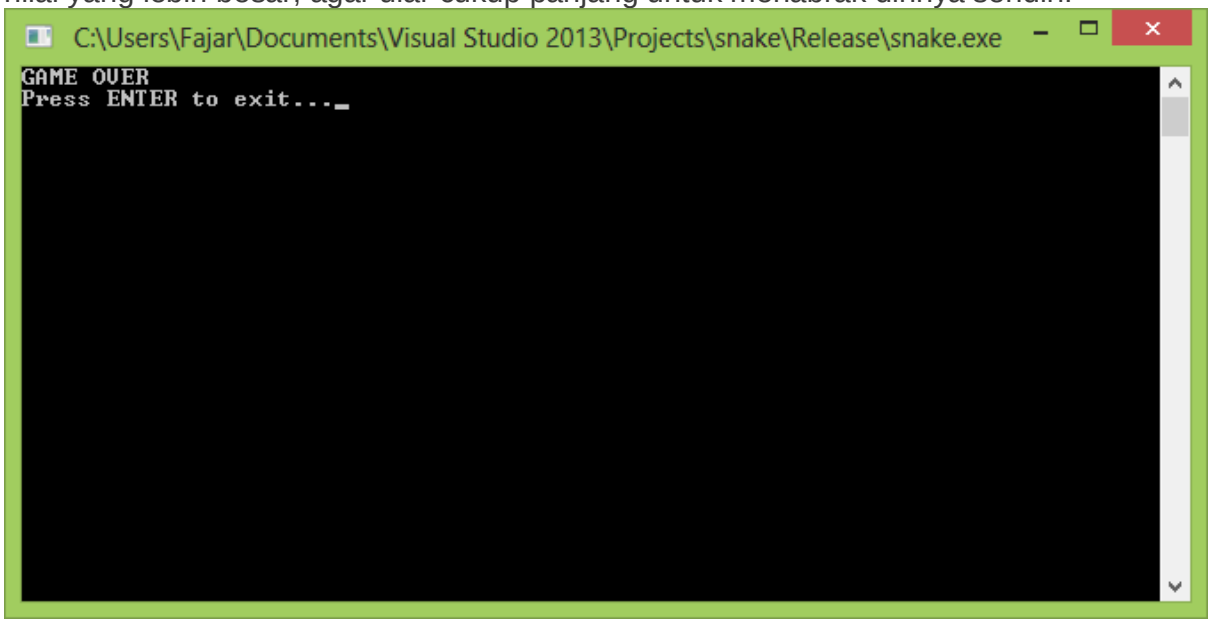
Berikut ini baris-baris yang ditambahkan di **main()** untuk melakukan pengecekan tadi, serta tambahan baris yang dilakukan di luar *game loop*, setelah permainan berakhir (*game over*).

```
/**
    Program utama
 */
int main() {
    ...

    // Game loop. Bagian di dalam while akan dieksekusi
  terus menerus
```

```
    while (true) {

            ...
            // Snake bergerak setiap 200 ms (sesuai nilai variable snake_speed)
            // Dihitung dengan membandingkan selisih waktu sekarang dengan waktu
            // terakhir kali snake bergerak.
            if (interval >= snake_speed) {

                ...

                // Jika posisi kepala (head) menabrak tembok pembatas,
                // maka permainan berakhir (keluar dari game loop)
                if (x < 0 || x >= console_width || y < 0 || y >= console_height) {
                    break;
                }

                // Jika posisi kepala (head) menabrak dirinya sendiri
                // (posisi sama dengan salah satu segment), maka permainan
                // berakhir (keluar dari game loop)
                if (check_collision(x, y) == 1) {
                    break;
                }

                // Jika tidak terjadi tabrakan (collision), maka snake
                // boleh bergerak maju..
                // Pop ekor, lalu push segment ke depan head sehingga
                // snake tampak bergerak maju.
                pop();
                push(x, y);

                // Tampilkan kondisi permainan saat ini di layar...
                ...
            }
            ...
        }

        // Setelah keluar dari game loop, berarti permainan berakhir (game over)
        system("cls");
        printf("GAME OVER\n");

        printf("Press ENTER to exit...");
        getchar();
```

```
50.         return 0;
51.     }
```

Jalankan program sekali lagi, lalu coba arahkan ular ke dinding. Untuk pengetesan tabrakan terhadap diri sendiri, bisa dilakukan dengan mengubah **snake_size** dengan nilai yang lebih besar, agar ular cukup panjang untuk menabrak dirinya sendiri.



Tampilan layar saat terjadi tabrakan. Permainan berakhir.

Makanan!

Ini adalah bagian terakhir dari tutorial ini, makanan! Ular perlu melahap makanan untuk menjadi lebih panjang. Untuk itu, kita perlu menempatkan makanan di koordinat acak. Untuk menaruh koordinat makanan, kita tambahkan dua variabel global **food_x** dan **food_y**.

```
1. // Posisi makanan
2. int food_x, food_y;
```

Meskipun makanan ditaruh secara acak, ada dua hal yang perlu diperhatikan:

1. Makanan harus berada di dalam layar *console* berukuran 80×25.
2. Makanan tidak boleh bertumpuk dengan ular saat ditempatkan.

Maka dari itu, kita buat sebuah fungsi **place_food()** untuk menaruh makanan dengan memerhatikan kedua syarat tersebut. Untuk syarat nomor 2, kita bisa memanfaatkan fungsi **check_collision()** yang baru saja dibuat.

```
1. /**
2.     Taruh makanan secara acak, namun memastikan
3.     makanan tidak bertumpuk dengan salah satu segment
4.     snake (array)
5.  */
6. void place_food() {
7.     // Jika makanan bertumpuk dengan salah satu segment
8.     // snake, ulangi penempatan makanan secara acak.
9.     do {
10.             food_x = rand() % console_width;
11.             food_y = rand() % console_height;
12.         } while (check_collision(food_x, food_y) == 1);
13.     }
```

Di awal program sebelum memasuki *game loop*, kita menempatkan makanan pertama. Berikutnya, makanan akan ditempatkan ulang jika posisi x dan y baru dari ular sama dengan koordinat makanan, yang artinya ular memakan makanan. Dalam hal ini, kita hanya melakukan **push()** tanpa melakukan **pop()**, sehingga jumlah elemen bertambah. Jangan lupa pula untuk melakukan *rendering* makanan di layar.
Di samping itu, kita juga bisa menerapkan sistem penilaian, misalnya nilai bertambah 100 jika ular memakan makanan. Lalu pada akhir permainan (saat *game over*), nilai yang sudah terkumpul ditampilkan kepada pemain.

```
1. /**
2.     Program utama
3.  */
4. int main() {
5.     // Randomize
6.     srand(time(NULL));
7.
8.     // Untuk menyimpan penanda waktu saat snake bergerak
9.     struct timeb last_timestamp;
10.         ftime(&last_timestamp); // Set nilai awal
11.
12.         // Untuk menyimpan nilai
13.         int score = 0;
14.
15.         // Pertama-tama, push segment (node) ke kanan
16.         // sebanyak 3 segment (sesuai nilai variable
  snake_size)
17.         for (int i = 0; i < snake_size; i++) {
18.             push(i, 0);
19.         }
20.
21.         // Tempatkan makanan secara acak
22.         place_food();
23.
24.         // Game loop. Bagian di dalam while akan
  dieksekusi terus menerus
25.         while (true) {
26.
27.             ...
28.
29.             // Snake bergerak setiap 500 ms (sesuai
  nilai variable snake_speed)
30.             // Dihitung dengan membandingkan selisih
  waktu sekarang dengan waktu
31.             // terakhir kali snake bergerak.
32.             if (interval >= snake_speed) {
33.
34.                 ...
35.
36.                 // Jika tidak terjadi tabrakan
  (collision), maka snake
37.                 // boleh bergerak maju..
38.
39.                 // Pop ekor, lalu push segment ke depan
  head sehingga
```

```
                // snake tampak bergerak maju.
                // Namun jika posisi x,y ke mana kepala (head) snake akan
                // bergerak berada di posisi makanan, tidak perlu pop
                // sehingga segment bertambah panjang.
                if (x == food_x && y == food_y) {
                    // Dalam hal snake memakan makanan, maka nilai bertambah
                    score += 100;

                    // Lalu makanan ditempatkan ulang secara acak
                    place_food();
                }
                else {
                    pop();
                }
                push(x, y);

                // Tampilkan kondisi permainan saat ini di layar...

                // Bersihkan layar
                system("cls");

                // Cetak (render) snake di layar
                display();

                // Cetak (render) makanan di layar
                gotoxy(food_x, food_y);
                printf("X");

                // Perbarui penanda waktu
                last_timestamp = current_timestamp;
            }

            ...

        }

        // Setelah keluar dari game loop, berarti permainan berakhir (game over)
        // Tampilkan nilai yang diraih pemain
        system("cls");
        printf("GAME OVER\n");
        printf("Your score : %d\n\n", score);

        printf("Press ENTER to exit...");
        getchar();

        ...

```

```
87.     }
```