

Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра системного аналізу та теорії прийняття рішень

Звіт
з лабораторної роботи №3
**Діаграма Вороного, тріангуляція Делоне
і побудова лінійної опуклої оболонки
на площині**

Виконав
студент(ка) групи К-23

Флакей Р.Р.

1. Реалізувати алгоритм Форчуна побудови діаграми Вороного (fortune.doc) і його поточну візуалізацію (5 балів).

Збалансоване дерево [ред | ред код]

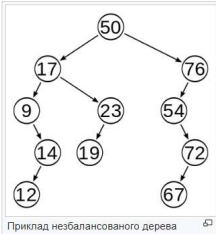
Матеріал з Вікіпедії — вільної енциклопедії

В програмуванні **збалансоване дерево** в загальному розумінні цього слова — це такий різновид бінарного дерева пошуку, яке автоматично підтримує свою висоту, тобто кількість рівнів вершин під коренем є мінімальною. Ця властивість є важливою тому, що час виконання більшості алгоритмів на бінарних деревах пошуку пропорційний до їхньої висоти, і звичайні бінарні дерева пошуку можуть мати досить велику висоту в тривіальних ситуаціях. Процедура зменшення (балансування) висоти дерева виконується за допомогою трансформацій, відомих як **обернення дерева**, в певні моменти часу (переважно при видаленні або додаванні нових елементів).

Більш точне визначення збалансованих дерев було дане Г.Адельсон-Вельським та Є.Ландісом.

Ідеально збалансоване дерево, за Адельсон-Вельським та Ландісом — це дерево, у якого для кожної вершини різниця між висотами лівого та правого піддерев не перевищує одиниці. Однак, така умова доволі складна для виконання на практиці і може вимагати значної перебудови дерева при додаванні або видаленні елементів.

Тому було запропоноване менш строгі визначення, яке отримало назву умови **ABП**(AVL)-**збалансованості** і говорить, що бінарне дерево є збалансованим, якщо висоти лівого та правого піддерев різняться не більше ніж на одиницю. Деревя, що задовольняють таким умовам, називаються **AVL-деревями**. Зрозуміло, що кожне ідеально збалансоване дерево є також AVL-збалансованим, але не навпаки.



Приклад незбалансованого дерева



Те саме дерево після балансування

Різновиди двійкових дерев [ред | ред код]

- **Двійкове дерево** — таке кореневе дерево, в якому кожна вершина має не більше двох дітей.
- **Повне (закінчене) двійкове дерево** — таке двійкове дерево, в якому кожна вершина має нуль або двох дітей.
- **Ідеальне двійкове дерево** — це таке повне двійкове дерево, в якому листя (вершини без дітей) лежать на однаковій глибині (відстані від кореня).

Двійкове дерево на кожному n -му рівні має від 1 до 2^n вершин.

Структура данных [править | править код]

Двухсвязный список рёбер состоит из таких типов объектов как: вершина, ребро и грань. Эти объекты содержат указатели на другие объекты. Это могут быть как указатели C/C++, содержащие адрес в памяти, так и индексы этих объектов в массиве или любой другой тип адресации. Непременным свойством является возможность прямого доступа к объекту на который ссылаются, без его поиска. Каждый из объектов может также содержать дополнительную информацию, например грань может содержать информацию о цвете или имени.

Вершина [править | править код]

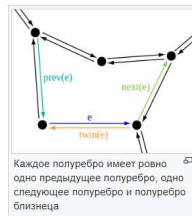
Вершина содержит единственный указатель на любое полуребро, исходящее из этой вершины. Также содержит информацию о своих координатах.

Полуребро [править | править код]

Полуребро содержит указатель на вершину, являющуюся его началом, указатель на грань, лежащую слева от ребра, а также указатели на следующее полуребро, предыдущее полуребро и полуребро близнеца. Грань лежит слева, потому полуребро близнец описывает правую сторону ребра, дополняя тем самым его до целого.

Грань [править | править код]

Грань содержит указатель на любое полуребро, составляющее его границу. Также может содержать список полуребер всех своих отверстий по одному полуребру на отверстие.



Каждое полуребро имеет ровно одно предыдущее полуребро, одно следующее полуребро и полуребро близнеца

Монотонный многоугольник [ред | ред код]

Матеріал з Вікіпедії — вільної енциклопедії

У геометрії, **многокутник** P на площині називають **монотонним** щодо прямої L , якщо кожна лінія ортогональна до L перетинає P щонайбільше двічі.^[1]

Подібно, ламану S зовуть **монотонною** щодо прямої L , якщо кожна лінія ортогональна з L перетинає S щонайбільше раз.

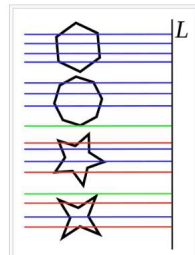
Для багатьох практичних цілей це визначення можна розширити, щоб дозволити випадки коли деякі ребра P ортогональні з L , і простий многокутник можна назвати монотонним якщо відрізок прямої, що поєднує дві точки в P і є ортогональним з L повністю належить P .

Властивості [ред | ред код]

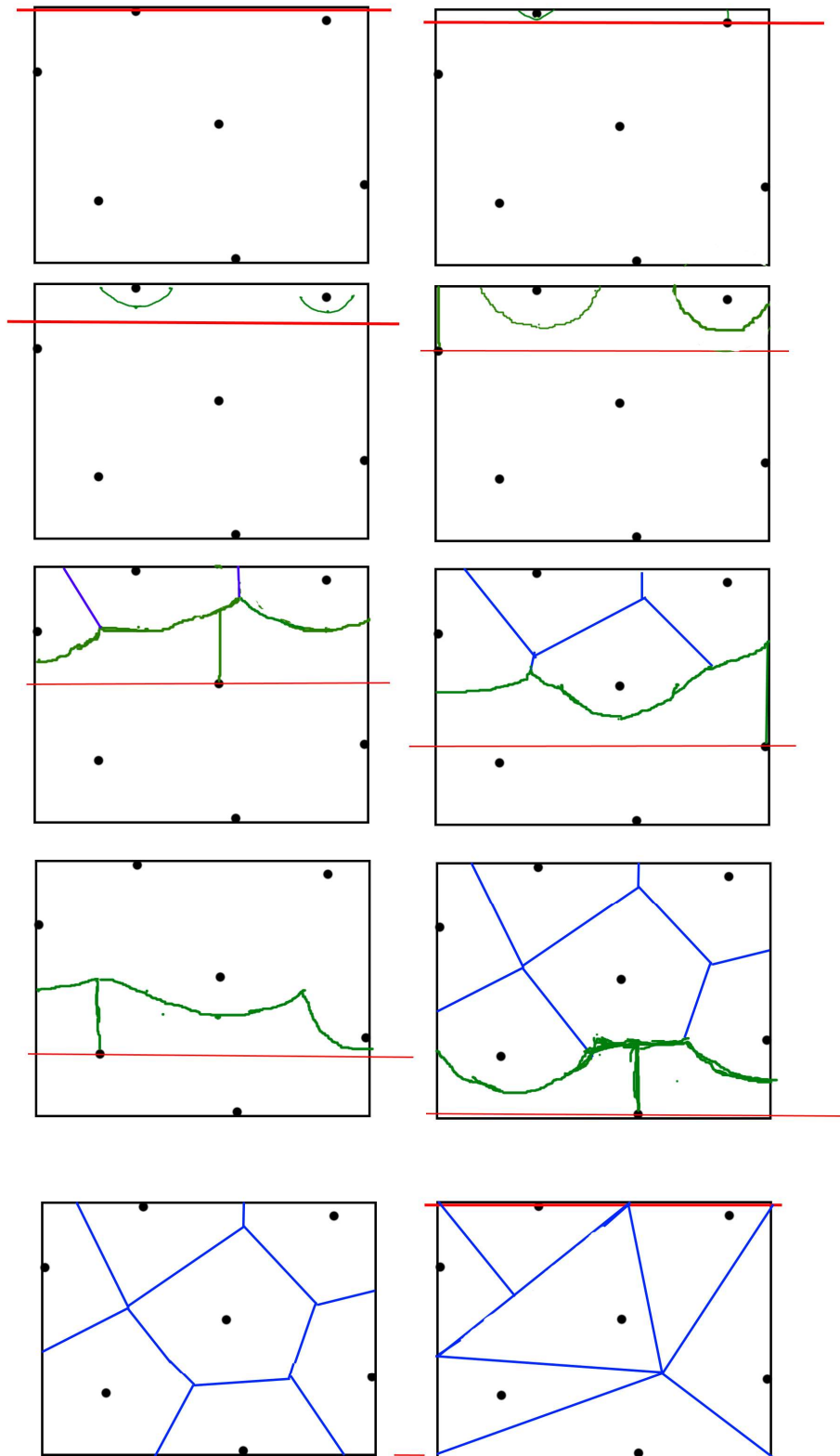
Якщо припустити, що L збігається з віссю x . Тоді найлівіша і найправіша вершини монотонного многокутника розбивають його границю на дві монотонні ламані, такі що коли вершини будь-якої ламаної перебирати в їхньому природному порядку, то їхні x -координати монотонно зростають або спадають.

Насправді, що властивість можна взяти за визначення монотонного многокутника і вона дає йому свої ім'я.

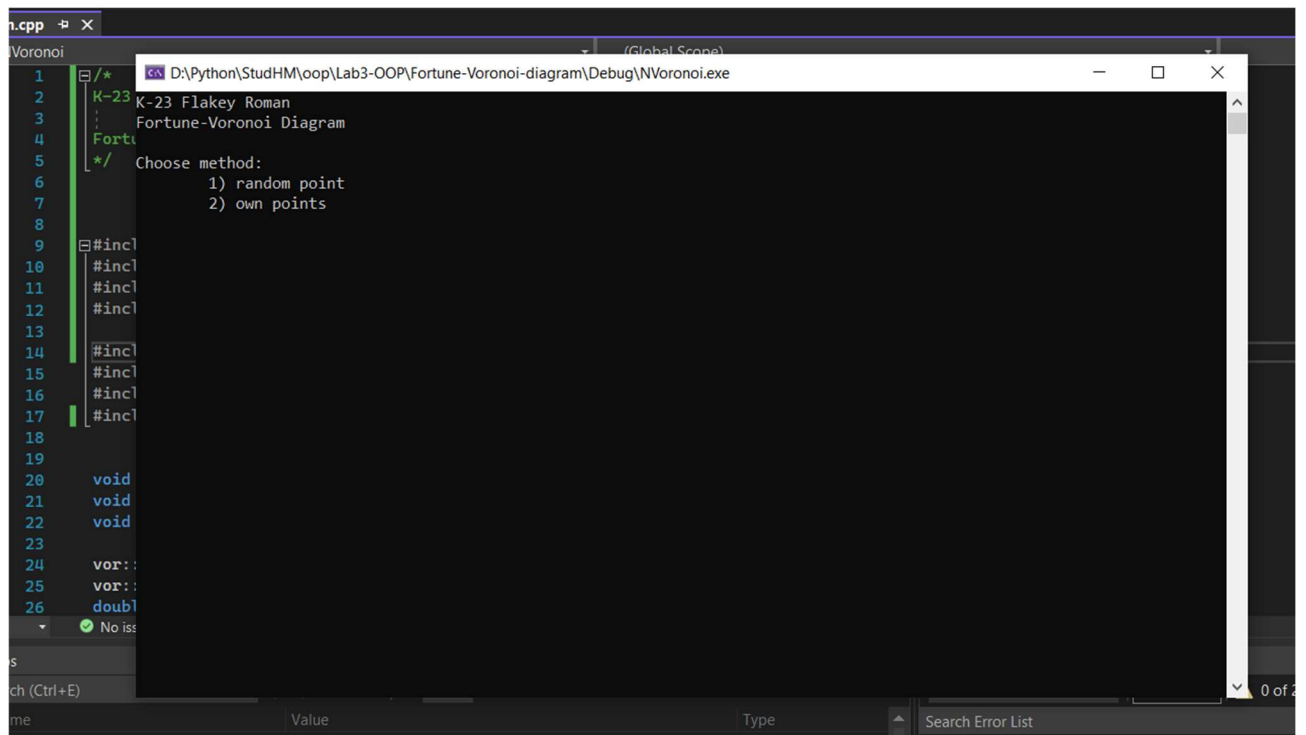
Опуклий многокутник є монотонним щодо будь-якої прямої і многокутник монотонний щодо будь-якої прямої є опуклим.



Зелене позначає один перетин, синє позначає два перетини і червоне позначає три або більше. Два горішні многокутники монотонні щодо L , тоді як інші — ні.

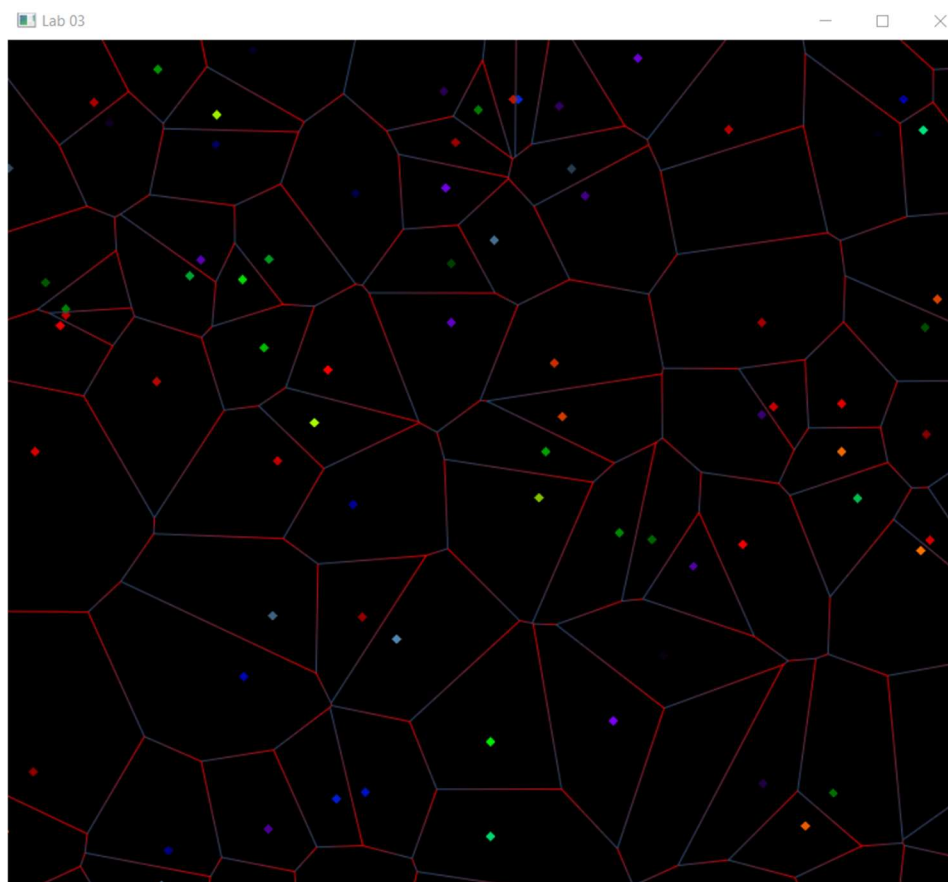


1. Реалізовано вибір вхідних даних, а саме Рандомні точки/власні (з файлу)



```
1  // ...
2  K-23 Flakey Roman
3  : Fortune-Voronoi Diagram
4  Fortu
5  */ Choose method:
6      1) random point
7      2) own points
8
9  #incl
10 #incl
11 #incl
12 #incl
13
14 #incl
15 #incl
16 #incl
17 #incl
18
19
20 void
21 void
22 void
23
24 vor::
25 vor::
26 doubl
27
28 No iss
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

1.1 Демонстрація 1 типу (випадкові точки)

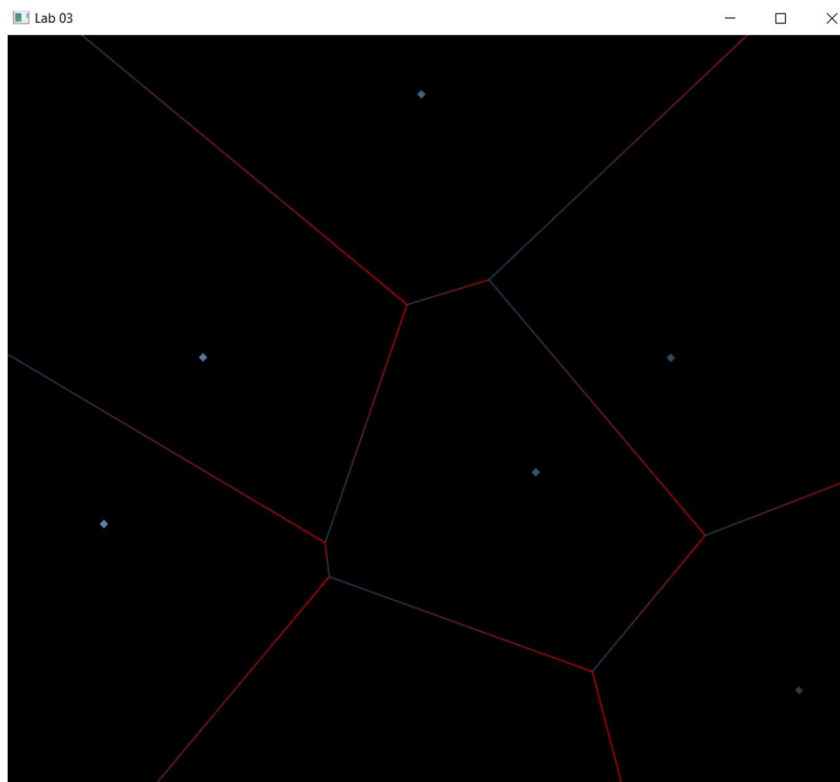


1.2 Демонстрація 2 типу (власні точки)

points.txt

```
1.0 6.0  
2.0 3.0  
3.0 4.0  
4.0 1.0  
4.0 7.0  
6.0 3.0  
7.0 5.0
```

```
D:\Python\StudHM\oop\Lab3-OOP\Fortune-Voronoi-diagram\Debug\NVoronoi.exe  
K-23 Flakey Roman  
Fortune-Voronoi Diagram  
  
Choose method:  
    1) random point  
    2) own points  
2  
Input name of your file with points:  
points.txt
```



Частина альтернатив

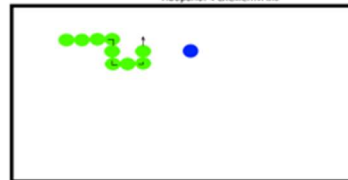
Змійка

0. Умова

4) Змійка - рухається по полю, генерується "їжа", при поглинанні якої змійка росте. Користувач може керувати поворотами з клавіатури. Кількість поглиненої їжі видно зверху текстом.

Змійка складається як наприклад, набір куль.

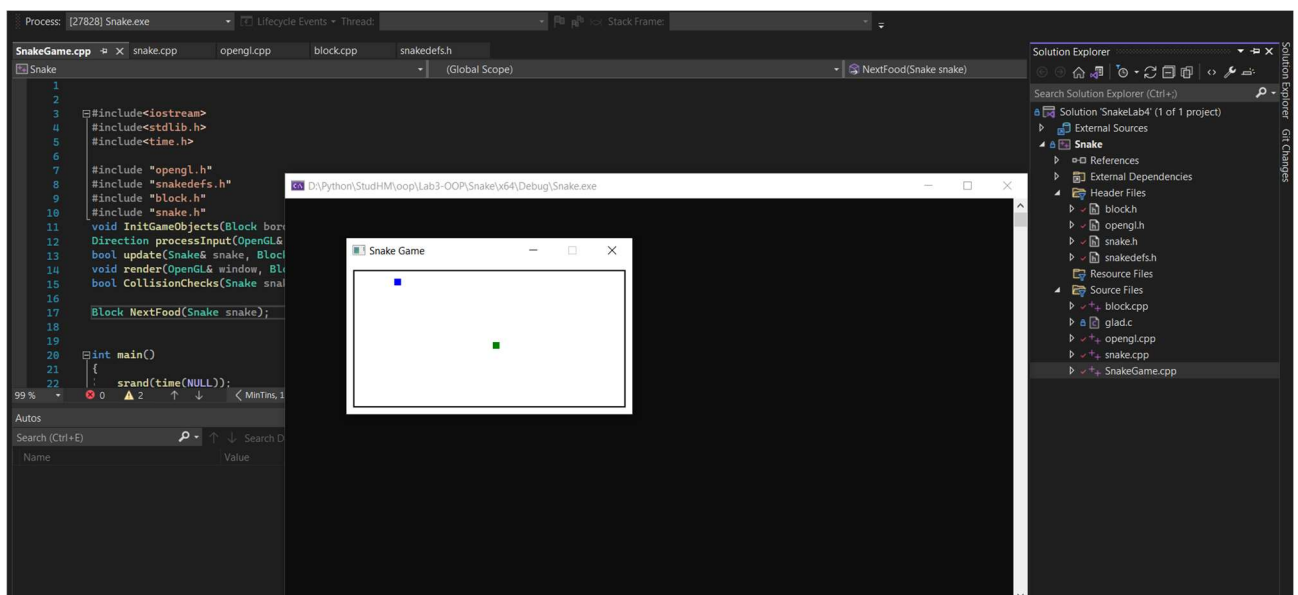
Вартість до 5 балів (якщо все красиво, кольорове..)



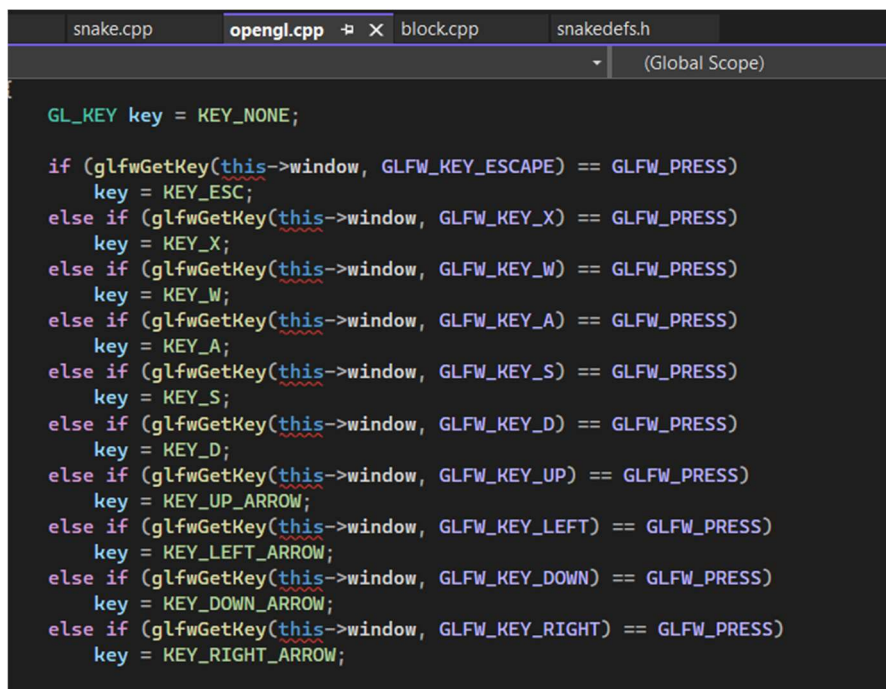
Після з'їдання генерується нова їжа (рандомно або з заданою позицією), змійка стає довшою на 1 елемент

WASD / стрілки

1. Реалізація



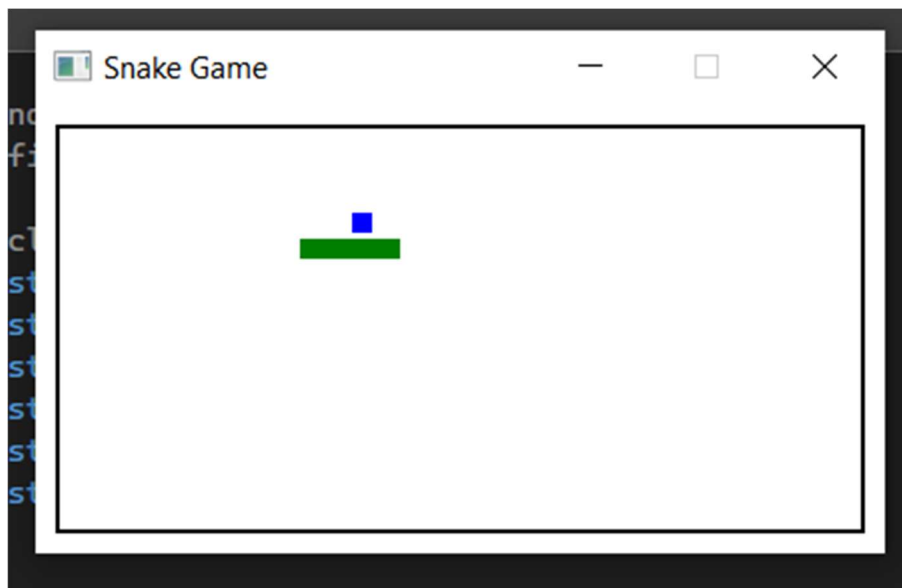
1.1. Розроблена наявність керування, за допомогою WASD/стрілок



1.2. Сама по собі змійка складається з квадратиків - блоків

```
block.h  SnakeGame.cpp  snake.cpp  opengl.cpp  block.cpp  sna
(Global Scope)
public:
    Block();
    Block(float x, float y, float width, float height, Color color,
          float velocityX = 0, float velocityY = 0,
          Block* before = nullptr, Block* after = nullptr);
    Block(Block* block);
    float getX();
    void setX(float value);
    float getY();
    void setY(float value);
    int getWidth();
    void setWidth(int value);
    int getHeight();
    void setHeight(int value);
    Color getColor();
    void setColor(Color value);
    float getVelocityX();
    void setVelocityX(float value);
    float getVelocityY();
    void setVelocityY(float value);
    Block* getBefore();
    void setBefore(Block* ptrBlock);
    Block* getAfter();
    void setAfter(Block* ptrBlock);
    void draw(OpenGL& window);
    bool intersects(Block* other);
    bool isMoving();
    void move(float distance);
    void append(Block* other);
```

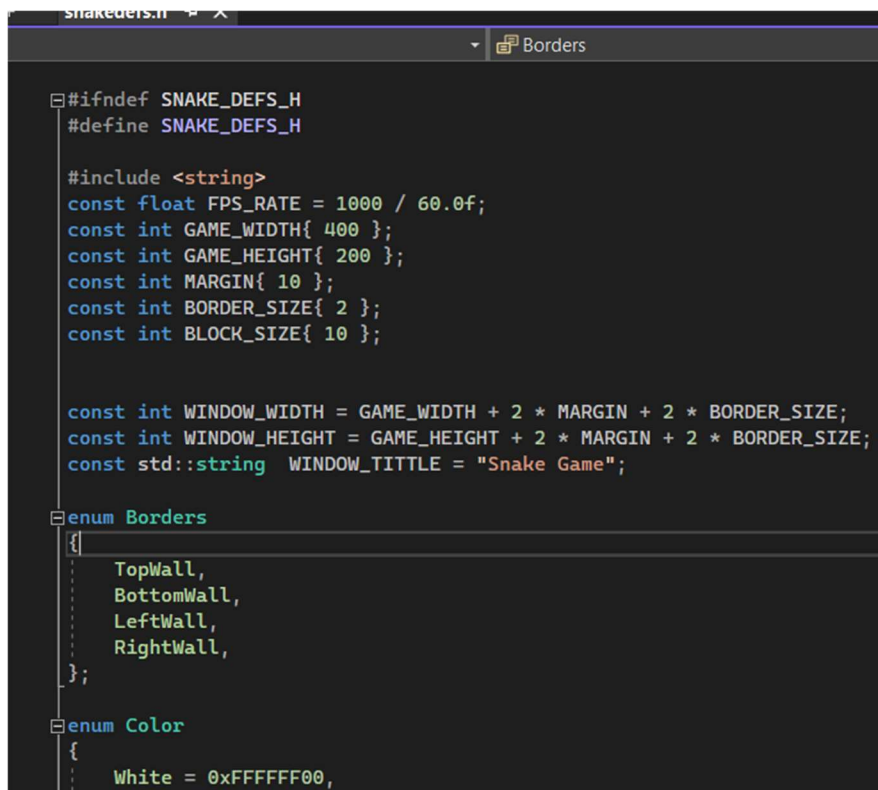
1.3. Поїдаючи випадково виставлені сині квадратики “їжі”, змійка росте



1.4. Присутня обробка ситуації, якщо змійка врізається в саму себе/в границю, або ж натискається хрестик

```
D:\Python\StudHM\oop\Lab3-OOP\Snake\x64\Debug\Snake.exe
Game Over!!
press any key and enter to close...
```

1.5. Для зручної зміни певних характеристик, все легко змінюється через `snakedefs.h`



```
snakedefs.h  Borders

#ifndef SNAKE_DEFS_H
#define SNAKE_DEFS_H

#include <string>
const float FPS_RATE = 1000 / 60.0f;
const int GAME_WIDTH{ 400 };
const int GAME_HEIGHT{ 200 };
const int MARGIN{ 10 };
const int BORDER_SIZE{ 2 };
const int BLOCK_SIZE{ 10 };

const int WINDOW_WIDTH = GAME_WIDTH + 2 * MARGIN + 2 * BORDER_SIZE;
const int WINDOW_HEIGHT = GAME_HEIGHT + 2 * MARGIN + 2 * BORDER_SIZE;
const std::string WINDOW_TITLE = "Snake Game";

enum Borders
{
    TopWall,
    BottomWall,
    LeftWall,
    RightWall,
};

enum Color
{
    White = 0xFFFFFFFF00,
```