Laporan Tugas 3
Struktur Data dan Analisa Algoritma

Mochammad Hilmi Rusydiansyah
5024211008

October 5, 2022

Source Code Linked List Application with Stack

```cpp
#include <iostream>
#include <unistd.h>

struct iniNode{
    float data;
    iniNode* below;
};

class Stack{
private:
    iniNode* downNode;
    iniNode* topNode;
    int len;
public:
    Stack(){
        downNode = new iniNode;
        topNode = new iniNode;
        downNode->below = NULL;
        topNode->below = NULL;
        len = 0;
    }

    void push(float inp){
        iniNode* temp = new iniNode;
        if(!len){  // input saat stack kosong
            temp->data = inp;
            temp->below = NULL;

            downNode = temp;
            topNode = temp;
        } else{  // input saat stack ada isinya
            temp = topNode;

            topNode = new iniNode;
            topNode->data = inp;
            topNode->below = temp;
        }
        len++;
    }

    void display(){
        iniNode* temp = topNode;
        std::cout << "ini top ygy" << std::endl;
        while(temp){
            std::cout << temp << " ==> " <<  temp->data << " -- " << temp
->below << std::endl;
            temp = temp->below;
        }
        std::cout << "ini down ygy" << std::endl;
    }

    void pop(){
        topNode = topNode->below;
        len--;
    }

    bool isEmpty(){
        if(!len){
            return 1;
        } return 0;
    }

    int size(){
        return len;
    }

    iniNode* top(){
        return topNode;
    }
};

int main(){
    Stack stackKuhh;
    unsigned int sec_to_microsec = 1000000;

    std::cout << "making a stack, please wait..." << std::endl;
    usleep(2*sec_to_microsec);
    std::cout << "completed !!!" << std::endl;
    usleep(1*sec_to_microsec);

    // isEmpty() checking
    std::cout << std::endl << "check dengan isEmpty() : ";
    if(stackKuhh.isEmpty()){
        std::cout << stackKuhh.isEmpty() << " ---> ini stack khosong" <<
    std::endl << std::endl;
    }

    std::cout << "Add 5 elements" << std::endl;
    usleep(1*sec_to_microsec);
    std::cout << "please wait..." << std::endl;
    usleep(2*sec_to_microsec);
    stackKuhh.push(5.0);
    stackKuhh.push(6.0);
    stackKuhh.push(7.0);
    stackKuhh.push(8.0);
    stackKuhh.push(9.0);
    std::cout << "adding process completed" << std::endl << std::endl;
    usleep(1*sec_to_microsec);

    // displaying linked list
    std::cout << "size() : " << stackKuhh.size() << std::endl;
    stackKuhh.display();
    usleep(2*sec_to_microsec);

    // isEmpty() checking
    std::cout << std::endl << "check dengan isEmpty() : ";
    if(!stackKuhh.isEmpty()){
        std::cout << stackKuhh.isEmpty() << " ---> sudah ada isinya" <<
    std::endl;
    }

    // remove
    std::cout << "\nPOP OUT 1 element" << std::endl;
    usleep(1*sec_to_microsec);
    stackKuhh.pop();
    std::cout << "delete process completed" << std::endl;
    usleep(1*sec_to_microsec);
    std::cout << std::endl;

    // displaying linked list
    std::cout << "size() : " << stackKuhh.size() << std::endl;
    stackKuhh.display();
    usleep(2*sec_to_microsec);

    std::cout << std::endl << "Add 7 elements" << std::endl;
    usleep(1*sec_to_microsec);
    std::cout << "please wait..." << std::endl;
    usleep(2*sec_to_microsec);
    stackKuhh.push(21);
    stackKuhh.push(22);
    stackKuhh.push(23);
    stackKuhh.push(24);
    stackKuhh.push(25);
    stackKuhh.push(26);
    stackKuhh.push(27);
    std::cout << "adding process completed" << std::endl << std::endl;
    usleep(1*sec_to_microsec);

    // displaying linked list
    std::cout << "size() : " << stackKuhh.size() << std::endl;
    stackKuhh.display();
    usleep(2*sec_to_microsec);

    // remove
    std::cout << "\nPOP OUT 4 element" << std::endl;
    usleep(1*sec_to_microsec);
    stackKuhh.pop();
    stackKuhh.pop();
    stackKuhh.pop();
    stackKuhh.pop();
    std::cout << "delete process completed" << std::endl;
    usleep(1*sec_to_microsec);
    std::cout << std::endl;

    // displaying linked list
    std::cout << "size() : " << stackKuhh.size() << std::endl;
    stackKuhh.display();
```

```cpp
    usleep(2*sec_to_microsec);


    // top
    std::cout << std::endl << "check the element on the top" << std::endl
    ;
    usleep(1*sec_to_microsec);
    iniNode* nodeBaru = stackKuhh.top();
    std::cout << "top -> data : " << nodeBaru->data << std::endl;
    std::cout << "top -> address : " << nodeBaru << std::endl;

    std::cout << "\nthank you" << std::endl;
    usleep(1*sec_to_microsec);

    return 0;
}
```

Source Code for Game Pesawat with SFML

```
#include <SFML/Graphics.hpp>
#include <cmath>
#include <unistd.h>

#define PI 3.14159265359

// global var/////////////////////////
unsigned int width_screen = 1000;
unsigned int height_screen = 1000;
//////////////////////////////////////////////////

class PesawatLakon{
private :
    float pos_x;
    float pos_y;
    float scale;
    float body_points[13][2] = {{1,-3}, {0.25,-3}, {0.25,-4.5}, {0,-5},
    {-0.25,-4.5}, {-0.25,-3}, {-1,-3}, {-1,1.5}, {-1.5,1.75}, {-1.25,2},
    {1.25,2}, {1.5,1.75}, {1,1.5}};  // for drawing body_points plane
    float wing_points[6][2] = {{1,-2}, {5,0}, {4,1}, {-4,1}, {-5,0},
    {-1,-2}};  // for drawing wing_points plane
    float tole_box;
    float health;
    float randValue_forBody[13];
    float randValue_forWing[6];

public:
    PesawatLakon(){
        scale = 15;
        pos_x = width_screen * 0.5;
        pos_y = height_screen * 0.75;
        tole_box = 2.5;
        health = 100;

        for(int i=0; i<13; i++){
            randValue_forBody[i] = (rand()%3) * scale;
        }
        for(int i=0; i<6; i++){
            randValue_forWing[i] = (rand()%3) * scale;
        }
    }

    void moveHor(int speed){
        pos_x = pos_x + speed;
    }

    void moveVer(int speed){
        pos_y = pos_y + speed;
    }

    void drawPesawat(sf::RenderWindow& window){
```

```
    sf::Vertex draw_body[] = {
        sf::Vertex(sf::Vector2f(pos_x + body_points[0][0]*scale,
pos_y + body_points[0][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + body_points[1][0]*scale,
pos_y + body_points[1][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + body_points[2][0]*scale,
pos_y + body_points[2][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + body_points[3][0]*scale,
pos_y + body_points[3][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + body_points[4][0]*scale,
pos_y + body_points[4][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + body_points[5][0]*scale,
pos_y + body_points[5][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + body_points[6][0]*scale,
pos_y + body_points[6][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + body_points[7][0]*scale,
pos_y + body_points[7][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + body_points[8][0]*scale,
pos_y + body_points[8][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + body_points[9][0]*scale,
pos_y + body_points[9][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + body_points[10][0]*scale,
pos_y + body_points[10][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + body_points[11][0]*scale,
pos_y + body_points[11][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + body_points[12][0]*scale,
pos_y + body_points[12][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + body_points[0][0]*scale,
pos_y + body_points[0][1]*scale))
    };
    sf::Vertex draw_wing[] = {
        sf::Vertex(sf::Vector2f(pos_x + wing_points[0][0]*scale,
pos_y + wing_points[0][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + wing_points[1][0]*scale,
pos_y + wing_points[1][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + wing_points[2][0]*scale,
pos_y + wing_points[2][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + wing_points[3][0]*scale,
pos_y + wing_points[3][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + wing_points[4][0]*scale,
pos_y + wing_points[4][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + wing_points[5][0]*scale,
pos_y + wing_points[5][1]*scale)),
        sf::Vertex(sf::Vector2f(pos_x + wing_points[0][0]*scale,
pos_y + wing_points[0][1]*scale))
    };

    window.draw(draw_body, 13+1, sf::LineStrip);
    window.draw(draw_wing, 6+1, sf::LineStrip);
}

void drawPesawatHancur(sf::RenderWindow& window){
    // ini asalnya body
    sf::Vertex draw_line1[] = {
        sf::Vertex(sf::Vector2f((pos_x + body_points[0][0]*scale) + randValue_forBody[0],                     (pos_y + body_points[0][1]*scale) + randValue_forBody[0])),
        sf::Vertex(sf::Vector2f((pos_x + body_points[1][0]*scale) + randValue_forBody[0],                     (pos_y + body_points[1][1]*scale) + randValue_forBody[0]))
    };
    sf::Vertex draw_line2[] = {
        sf::Vertex(sf::Vector2f((pos_x + body_points[1][0]*scale) + randValue_forBody[1],                     (pos_y + body_points[1][1]*scale) + randValue_forBody[1])),
        sf::Vertex(sf::Vector2f((pos_x + body_points[2][0]*scale) + randValue_forBody[1],                     (pos_y + body_points[2][1]*scale) + randValue_forBody[1]))
    };
    sf::Vertex draw_line3[] = {
        sf::Vertex(sf::Vector2f((pos_x + body_points[2][0]*scale) - randValue_forBody[2],                     (pos_y + body_points[2][1]*scale) + randValue_forBody[2])),
        sf::Vertex(sf::Vector2f((pos_x + body_points[3][0]*scale) - randValue_forBody[2],                     (pos_y + body_points[3][1]*scale) + randValue_forBody[2]))
    };
    sf::Vertex draw_line4[] = {
        sf::Vertex(sf::Vector2f((pos_x + body_points[3][0]*scale) + randValue_forBody[3],                     (pos_y + body_points[3][1]*scale) - randValue_forBody[3])),
        sf::Vertex(sf::Vector2f((pos_x + body_points[4][0]*scale) + randValue_forBody[3],                     (pos_y + body_points[4][1]*scale) - randValue_forBody[3]))
    };
    sf::Vertex draw_line5[] = {
        sf::Vertex(sf::Vector2f((pos_x + body_points[4][0]*scale) - randValue_forBody[4],                     (pos_y + body_points[4][1]*scale) - randValue_forBody[4])),
        sf::Vertex(sf::Vector2f((pos_x + body_points[5][0]*scale) - randValue_forBody[4],                     (pos_y + body_points[5][1]*scale) - randValue_forBody[4]))
    };
    sf::Vertex draw_line6[] = {
        sf::Vertex(sf::Vector2f((pos_x + body_points[5][0]*scale) + randValue_forBody[5],                     (pos_y + body_points[5][1]*scale) - randValue_forBody[5])),
        sf::Vertex(sf::Vector2f((pos_x + body_points[6][0]*scale) + randValue_forBody[5],                     (pos_y + body_points[6][1]*scale) - randValue_forBody[5]))
    };
    sf::Vertex draw_line7[] = {
        sf::Vertex(sf::Vector2f((pos_x + body_points[6][0]*scale) + randValue_forBody[6],                     (pos_y + body_points[6][1]*scale) + randValue_forBody[6])),
        sf::Vertex(sf::Vector2f((pos_x + body_points[7][0]*scale) +
randValue_forBody[6],                              (pos_y + body_points[7][1]*
scale) + randValue_forBody[6]))
        };
        sf::Vertex draw_line8[] = {
            sf::Vertex(sf::Vector2f((pos_x + body_points[7][0]*scale) -
randValue_forBody[7],                              (pos_y + body_points[7][1]*
scale) + randValue_forBody[7])),
            sf::Vertex(sf::Vector2f((pos_x + body_points[8][0]*scale) -
randValue_forBody[7],                              (pos_y + body_points[8][1]*
scale) + randValue_forBody[7]))
        };
        sf::Vertex draw_line9[] = {
            sf::Vertex(sf::Vector2f((pos_x + body_points[8][0]*scale) -
randValue_forBody[8],                              (pos_y + body_points[8][1]*
scale) - randValue_forBody[8])),
            sf::Vertex(sf::Vector2f((pos_x + body_points[9][0]*scale) -
randValue_forBody[8],                              (pos_y + body_points[9][1]*
scale) - randValue_forBody[8]))
        };
        sf::Vertex draw_line10[] = {
            sf::Vertex(sf::Vector2f((pos_x + body_points[9][0]*scale) +
randValue_forBody[9],                              (pos_y + body_points[9][1]*
scale) + randValue_forBody[9])),
            sf::Vertex(sf::Vector2f((pos_x + body_points[10][0]*scale) +
randValue_forBody[9],                             (pos_y + body_points[10][1]*
scale) + randValue_forBody[9]))
        };
        sf::Vertex draw_line11[] = {
            sf::Vertex(sf::Vector2f((pos_x + body_points[10][0]*scale) +
randValue_forBody[10],                             ( pos_y + body_points
[10][1]*scale) - randValue_forBody[10])),
            sf::Vertex(sf::Vector2f((pos_x + body_points[11][0]*scale) +
randValue_forBody[10],                             ( pos_y + body_points
[11][1]*scale) - randValue_forBody[10]))
        };
        sf::Vertex draw_line12[] = {
            sf::Vertex(sf::Vector2f((pos_x + body_points[11][0]*scale) -
randValue_forBody[11],                             ( pos_y + body_points
[11][1]*scale) - randValue_forBody[11])),
            sf::Vertex(sf::Vector2f((pos_x + body_points[12][0]*scale) -
randValue_forBody[11],                             ( pos_y + body_points
[12][1]*scale) - randValue_forBody[11]))
        };
        sf::Vertex draw_line13[] = {
            sf::Vertex(sf::Vector2f((pos_x + body_points[12][0]*scale) -
randValue_forBody[12],                             ( pos_y + body_points
[12][1]*scale) + randValue_forBody[12])),
            sf::Vertex(sf::Vector2f((pos_x + body_points[0][0]*scale) -
randValue_forBody[12],                              (pos_y + body_points
[0][1]*scale) + randValue_forBody[12]))
        };
        //ini asalnya wing
        sf::Vertex draw_line14[] = {
            sf::Vertex(sf::Vector2f((pos_x + wing_points[0][0]*scale) + randValue_forWing[0],                              (pos_y + wing_points[0][1]*scale) + randValue_forWing[0])),
            sf::Vertex(sf::Vector2f((pos_x + wing_points[1][0]*scale) + randValue_forWing[0],                              (pos_y + wing_points[1][1]*scale) + randValue_forWing[0]))
        };
        sf::Vertex draw_line15[] = {
            sf::Vertex(sf::Vector2f((pos_x + wing_points[1][0]*scale) + randValue_forWing[1],                              (pos_y + wing_points[1][1]*scale) + randValue_forWing[1])),
            sf::Vertex(sf::Vector2f((pos_x + wing_points[2][0]*scale) + randValue_forWing[1],                              (pos_y + wing_points[2][1]*scale) + randValue_forWing[1]))
        };
        sf::Vertex draw_line16[] = {
            sf::Vertex(sf::Vector2f((pos_x + wing_points[2][0]*scale) - randValue_forWing[2],                              (pos_y + wing_points[2][1]*scale) + randValue_forWing[2])),
            sf::Vertex(sf::Vector2f((pos_x + wing_points[3][0]*scale) - randValue_forWing[2],                              (pos_y + wing_points[3][1]*scale) + randValue_forWing[2]))
        };
        sf::Vertex draw_line17[] = {
            sf::Vertex(sf::Vector2f((pos_x + wing_points[3][0]*scale) + randValue_forWing[3],                              (pos_y + wing_points[3][1]*scale) - randValue_forWing[3])),
            sf::Vertex(sf::Vector2f((pos_x + wing_points[4][0]*scale) + randValue_forWing[3],                              (pos_y + wing_points[4][1]*scale) - randValue_forWing[3]))
        };
        sf::Vertex draw_line18[] = {
            sf::Vertex(sf::Vector2f((pos_x + wing_points[4][0]*scale) - randValue_forWing[4],                              (pos_y + wing_points[4][1]*scale) - randValue_forWing[4])),
            sf::Vertex(sf::Vector2f((pos_x + wing_points[5][0]*scale) - randValue_forWing[4],                              (pos_y + wing_points[5][1]*scale) - randValue_forWing[4]))
        };
        sf::Vertex draw_line19[] = {
            sf::Vertex(sf::Vector2f((pos_x + wing_points[5][0]*scale) + randValue_forWing[5],                              (pos_y + wing_points[5][1]*scale) - randValue_forWing[5])),
            sf::Vertex(sf::Vector2f((pos_x + wing_points[0][0]*scale) + randValue_forWing[5],                              (pos_y + wing_points[0][1]*scale) - randValue_forWing[5]))
        };
        // sf::Vertex draw_wing[] = {
            // sf::Vertex(sf::Vector2f(pos_x + wing_points[0][0]*scale, pos_y + wing_points[0][1]*scale)),
            // sf::Vertex(sf::Vector2f(pos_x + wing_points[1][0]*scale, pos_y + wing_points[1][1]*scale)),
            // sf::Vertex(sf::Vector2f(pos_x + wing_points[2][0]*scale,
```

```cpp
    pos_y + wing_points[2][1]*scale)),
            // sf::Vertex(sf::Vector2f(pos_x + wing_points[3][0]*scale,
    pos_y + wing_points[3][1]*scale)),
            // sf::Vertex(sf::Vector2f(pos_x + wing_points[4][0]*scale,
    pos_y + wing_points[4][1]*scale)),
            // sf::Vertex(sf::Vector2f(pos_x + wing_points[5][0]*scale,
    pos_y + wing_points[5][1]*scale)),
            // sf::Vertex(sf::Vector2f(pos_x + wing_points[0][0]*scale,
    pos_y + wing_points[0][1]*scale))
        // };

        window.draw(draw_line1, 2, sf::Lines);
        window.draw(draw_line2, 2, sf::Lines);
        window.draw(draw_line3, 2, sf::Lines);
        window.draw(draw_line4, 2, sf::Lines);
        window.draw(draw_line5, 2, sf::Lines);
        window.draw(draw_line6, 2, sf::Lines);
        window.draw(draw_line7, 2, sf::Lines);
        window.draw(draw_line8, 2, sf::Lines);
        window.draw(draw_line9, 2, sf::Lines);
        window.draw(draw_line10, 2, sf::Lines);
        window.draw(draw_line11, 2, sf::Lines);
        window.draw(draw_line12, 2, sf::Lines);
        window.draw(draw_line13, 2, sf::Lines);
        window.draw(draw_line14, 2, sf::Lines);
        window.draw(draw_line15, 2, sf::Lines);
        window.draw(draw_line16, 2, sf::Lines);
        window.draw(draw_line17, 2, sf::Lines);
        window.draw(draw_line18, 2, sf::Lines);
        window.draw(draw_line19, 2, sf::Lines);
        // window.draw(draw_wing, 6+1, sf::LineStrip);

        for(int i=0; i<13; i++){
            if(i%2==0){
                randValue_forBody[i] += 1*0.025*scale;
            } else{
                randValue_forBody[i] -= 1*0.025*scale;
            }
        }
        for(int i=0; i<6; i++){
            if(i%2==0){
                randValue_forWing[i] += 1*0.025*scale;
            } else{
                randValue_forWing[i] -= 1*0.025*scale;
            }
        }
    }

    float batasBox_kiri(){
        return pos_x - tole_box * scale;
    }
    float batasBox_kanan(){
        return pos_x + tole_box * scale;
    }
    float batasBox_atas(){
        return pos_y - tole_box * scale;
    }
    float batasBox_bawah(){
        return pos_y + tole_box * scale;
    }

    void reduceHealth(float damage){
        if(health <= 0){
            health = 0;
        } else{
            health = health + damage;
        }
    }

    void displayHealth(sf::RenderWindow& window){
        sf::RectangleShape rectangle(sf::Vector2f((health/100) * 220.f,
    30.f));
        rectangle.setPosition(sf::Vector2f(30.f, 940.f));

        window.draw(rectangle);
    }

    float getHealth(){
        return health;
    }
};

class PesawatMusuh{
private :
    float pos_x;
    float pos_y;
    float scale;
    // int speed_y;  // determine plane speed y
    int speed_x;  // determine plane speed x
    float body_points[9][2] = {{0.5,0}, {0.5,-1.25}, {0,-1.5},
    {-0.5,-1.25}, {-0.5,0}, {-0.25,2.75}, {-1,3}, {1,3}, {0.25,2.75}};  //
     for drawing body_points plane
    float wing_points[6][2] = {{2.5,0.5}, {3,1.75}, {2.5,-0.5},
    {-2.5,-0.5}, {-3,1.75}, {-2.5,0.5}};  // for drawing wing_points plane

    bool bullet_exist;
    float pos_bullet_x;
    float pos_bullet_y;
    float rad_bullet;
    int speed_bullet_y;

public:
    PesawatMusuh(){
        scale = rand()%15 + 5;
        // speed_y = rand()%5 + 1;
        speed_x = rand()%5 + 1;
        pos_x = rand() % width_screen + 10;
        pos_y = rand() % int(height_screen*0.25) + 10;

        bullet_exist = false;
        rad_bullet = 5;
        speed_bullet_y = 5;
    }

    void moveRight(){
        pos_x = pos_x + speed_x;
        if(pos_x > width_screen){
            pos_x = pos_x - width_screen;
        }
    }

    void moveLeft(){
        pos_x = pos_x - speed_x;
        if(pos_x < 0){
            pos_x = pos_x + width_screen;
        }
    }

    void rotatePesawat(float degree){
        int num_bpoints = 9;
        int num_wpoints = 6;
        // convert degree to radian
        float rad = degree * (PI/180);

        // rotating body
        for(int i=0; i<num_bpoints; i++){
            float temp[2];  // for storing one-point coordinate
            // assign to temp
            for(int j=0; j<2; j++){
                temp[j] = body_points[i][j];
            }
            // convert cartesian to polar (r, theta)
            float r = sqrt( pow(temp[0],2) + pow(temp[1],2) );
            float theta = atan2(temp[1], temp[0]);

            // adding theta by degree(radian) inputted
            theta = theta + rad;

            // convert polar to cartesian again and store it to temp
            temp[0] = r * cos(theta);  // as x
            temp[1] = r * sin(theta);  // as y

            // return temp to body_array
            for(int j=0; j<2; j++){
                body_points[i][j] = temp[j];
            }
        }

        // rotating wing
        for(int i=0; i<num_wpoints; i++){
            float temp[2];  // for storing one-point coordinate
            // assign to temp
            for(int j=0; j<2; j++){
                temp[j] = wing_points[i][j];
            }
            // convert cartesian to polar (r, theta)
            float r = sqrt( pow(temp[0],2) + pow(temp[1],2) );
            float theta = atan2(temp[1], temp[0]);

            // adding theta by degree(radian) inputted
            theta = theta + rad;

            // convert polar to cartesian again and store it to temp
            temp[0] = r * cos(theta);  // as x
            temp[1] = r * sin(theta);  // as y

            // return temp to wing_array
            for(int j=0; j<2; j++){
                wing_points[i][j] = temp[j];
            }
        }
    }

    void drawPesawat(sf::RenderWindow& window){
        sf::Vertex draw_body[] = {
            sf::Vertex(sf::Vector2f(pos_x + body_points[0][0]*scale, pos_y + body_points[0][1]*scale)),
            sf::Vertex(sf::Vector2f(pos_x + body_points[1][0]*scale, pos_y + body_points[1][1]*scale)),
            sf::Vertex(sf::Vector2f(pos_x + body_points[2][0]*scale, pos_y + body_points[2][1]*scale)),
            sf::Vertex(sf::Vector2f(pos_x + body_points[3][0]*scale, pos_y + body_points[3][1]*scale)),
            sf::Vertex(sf::Vector2f(pos_x + body_points[4][0]*scale, pos_y + body_points[4][1]*scale)),
            sf::Vertex(sf::Vector2f(pos_x + body_points[5][0]*scale, pos_y + body_points[5][1]*scale)),
            sf::Vertex(sf::Vector2f(pos_x + body_points[6][0]*scale, pos_y + body_points[6][1]*scale)),
            sf::Vertex(sf::Vector2f(pos_x + body_points[7][0]*scale, pos_y + body_points[7][1]*scale)),
            sf::Vertex(sf::Vector2f(pos_x + body_points[8][0]*scale, pos_y + body_points[8][1]*scale)),
            sf::Vertex(sf::Vector2f(pos_x + body_points[0][0]*scale, pos_y + body_points[0][1]*scale))
        };
        sf::Vertex draw_wing[] = {
            sf::Vertex(sf::Vector2f(pos_x + wing_points[0][0]*scale, pos_y + wing_points[0][1]*scale)),
            sf::Vertex(sf::Vector2f(pos_x + wing_points[1][0]*scale, pos_y + wing_points[1][1]*scale)),
            sf::Vertex(sf::Vector2f(pos_x + wing_points[2][0]*scale,
```

```cpp
    pos_y + wing_points[2][1]*scale)),
            sf::Vertex(sf::Vector2f(pos_x + wing_points[3][0]*scale, pos_y + wing_points[3][1]*scale)),
            sf::Vertex(sf::Vector2f(pos_x + wing_points[4][0]*scale, pos_y + wing_points[4][1]*scale)),
            sf::Vertex(sf::Vector2f(pos_x + wing_points[5][0]*scale, pos_y + wing_points[5][1]*scale)),
            sf::Vertex(sf::Vector2f(pos_x + wing_points[0][0]*scale, pos_y + wing_points[0][1]*scale))
        };

        window.draw(draw_body, 9+1, sf::LineStrip);
        window.draw(draw_wing, 6+1, sf::LineStrip);
    }

    void saveBulletPos(){
        pos_bullet_x = pos_x;
        pos_bullet_y = pos_y;
    }

    void moveBulletDown(){
        pos_bullet_y = pos_bullet_y + speed_bullet_y;
    }

    void drawBullet(sf::RenderWindow& window){
        sf::CircleShape circle(rad_bullet);
        circle.setPosition(sf::Vector2f(pos_bullet_x, pos_bullet_y));
        window.draw(circle);
    }

    float posBulletX(){
        return pos_bullet_x;
    }

    float posBulletY(){
        return pos_bullet_y;
    }

    bool isBulletExist(){
        return bullet_exist;
    }

    void setBulletExistance(bool existance){
        bullet_exist = existance;
    }
};


int main(){
    sf::RenderWindow window(sf::VideoMode(width_screen, height_screen), "Pesawat-Pesawatan");

    // create pesawat lakon
    PesawatLakon jatayu;

    // create pesawat musuh
    int num_opponents = 6;
    PesawatMusuh opponents[num_opponents];
    // PesawatMusuh opponents_toLeft[num_opponents];
    for(int i=0; i<num_opponents; i++){
        if(i<(num_opponents/2)){
            opponents[i].rotatePesawat(90);
        } else{
            opponents[i].rotatePesawat(-90);
        }
    }

    while(window.isOpen()){
        sf::Event event;
        while(window.pollEvent(event)){
            if(event.type == sf::Event::Closed){
                window.close();
            }
            else if(event.type == sf::Event::KeyPressed){
                if(event.key.code == sf::Keyboard::W){
                    // printf("W\n");
                    jatayu.moveVer(-10);
                } if(event.key.code == sf::Keyboard::S){
                    // printf("S\n");
                    jatayu.moveVer(10);
                } if(event.key.code == sf::Keyboard::A){
                    // printf("A\n");
                    jatayu.moveHor(-10);
                } if(event.key.code == sf::Keyboard::D){
                    // printf("D\n");
                    jatayu.moveHor(10);
                }
            }
            else if(event.type == sf::Event::MouseButtonPressed){
                if(event.mouseButton.button == sf::Mouse::Left){
                    // printf("klik kiri\n");
                }
            }
        }

        // kalo peluru kena ke kita
        for(int i=0; i<num_opponents; i++){
            if(opponents[i].posBulletX() >= jatayu.batasBox_kiri() &&
                opponents[i].posBulletX() <= jatayu.batasBox_kanan() &&
                opponents[i].posBulletY() >= jatayu.batasBox_atas() &&
                opponents[i].posBulletY() <= jatayu.batasBox_bawah()){
                    opponents[i].setBulletExistance(false);
                    jatayu.reduceHealth(-25);
            }
        }

        window.clear();

        jatayu.displayHealth(window);
        if(jatayu.getHealth() == 0){
            jatayu.drawPesawatHancur(window);
        } else{
            jatayu.drawPesawat(window);
        }

        for(int i=0; i<num_opponents; i++){
            // for draw pesawat musuh
            if(i<(num_opponents/2)){
                opponents[i].moveRight();
            } else{
                opponents[i].moveLeft();
            }
            opponents[i].drawPesawat(window);

            // for draw bullet musuh
            if(!(opponents[i].isBulletExist())){  // checking bullet existance
                opponents[i].saveBulletPos();
                opponents[i].setBulletExistance(true);
            }
            opponents[i].moveBulletDown();
            if(opponents[i].posBulletY() > width_screen){
                opponents[i].saveBulletPos();  // reset bullet position
            }
            opponents[i].drawBullet(window);
        }

        window.display();
        usleep(20*1000);
    }

    return 0;
}
```