```cpp
///////////////////////////////////////////////////////////////////////////////
//
// Author:           Sharome Burton
// Email:            koulkoudakis@gmail.com
// Label:            P01
// Title:            Game: Game
// Course:           CMPS 2143
// Semester:         Spring 2021
//
// Description:
//       This program implements a simple game where a green player ball aims
//       to dodge oncoming debris
// Usage:
//      - $ ./main filename
//      - Use up/down arrow keys to move green player ball
//
// Files:
//      P01.cpp    : driver program
///////////////////////////////////////////////////////////////////////////////

#include <SFML/Graphics.hpp>
#include <iostream>
#include <vector>
#include <time.h>
#include <string.h>

class Player : public sf::Drawable {
public:
    Player(int w, int h) {

        width = w;          // window size
        height = h;
        dx = -1;            // direction
        dy = -1;
        x = (rand() % int(width / 4));  // location (Left quarter of screen)
        y = int(height / 2);
        d = .1;             // distance

        // define a circle with radius = 200
        circle = new sf::CircleShape(20.f);

        circle->setFillColor(sf::Color::Green);

        // set the radius to whatever
        circle->setRadius(20.f);

        // change the number of sides (points) to 100
        circle->setPointCount(100);

        circle->setPosition(sf::Vector2f(x, y));

    }

    // Updates Player values
    void updateP() {

        position = circle->getPosition();
        bool hit = false;

        //circle->move(sf::Vector2f(x, y));
        circle->setPosition(sf::Vector2f(x, y));
    }

    // Allows player ball to translate up or down after key input
    void moveVert(int dir) {

        switch (dir) {
        case 0:     // Move player up
            y = position.y - 10;
            break;
        case 1:     // Move player down
            y = position.y + 10;
            break;
        }
    }

    // Checks bounding box for debris for collision-checking purposes
    sf::FloatRect getBounds() {
        return circle->getGlobalBounds();
    }

private:
    sf::CircleShape *circle;    // reference to our "ball"
    float x;                      // x location
    float y;                      // y location
    float dx;                     // "change" in x
    float dy;                     // "change" in y
    float width;
    float height;
    float d;                    // distance to move
    sf::Vector2f position;

    /**
     * virtual = A virtual function a member function which is
     * declared within base class and is re-defined (Overriden)
     *  by derived class.
     * function draw:
     *     draw an SFML object to some window
     */
    virtual void draw(sf::RenderTarget &target, sf::RenderStates states) const
      {
        //states.transform *= getTransform();
        target.draw(*circle, states);
    }
};

class Debris : public sf::Drawable {
public:
    Debris(int w, int h) {

        width = w;            // window size
        height = h;
        dx = -1;              // direction
        dy = 0;
        x = width - (rand() % 200);  // location (Far right of screen)
        y = (rand() % int(height));
        d = 0.03*(rand() % 10 + 1);            // speed
        rwmin = 350;    // minimum width
        rhmin = 350;    // minimum height

        srand(time(NULL));      // randomize size
        rheight = rand() % rhmin + 150;      // rect height
        rwidth = rand() % rwmin + 150;       // and width
        counter = 0;          // number of debris passed

        // define a rectangle
        rectangle = new sf::RectangleShape(sf::Vector2f(rwidth, rheight));

        rectangle->setFillColor(sf::Color::Red);

        rectangle->setPosition(sf::Vector2f(x, y));
    }

    // Update Debris position
    void updateD() {

        position = rectangle->getPosition();
        bool hit = false;

        if (position.x < -1 * (rwmin)) {
            srand(time(NULL));
            //position.x = 0;
            //dx = 0;
            //hit = true;
            x = width - (rand() % 200);  // location (Far right of screen)
            position.x = x;
            y = (rand() % int(height));
            position.y = y;
            d = 0.03*(counter + rand() % 10 + 1);             // speed
            rheight = rhmin + rand() % 50;      // rect height
            rwidth = rwmin + rand() % 50;       // and width
            counter += 1;

            rectangle->setSize(sf::Vector2f(rwidth, rheight));
            // std::cout << counter << '\n';
        }


        x = position.x + (d * dx);
        y = position.y + (d * dy);

        //circle->move(sf::Vector2f(x, y));
        rectangle->setPosition(sf::Vector2f(x, y));

    }

    // "Destroys" one piece of debris (basically sends it to right side of
      screen
    //with new starting position)
    void destroy() {
        rectangle->setPosition(sf::Vector2f(-1 * (rwmin), y));
    }

    // Checks bounding box for debris for collision-checking purposes
    sf::FloatRect getBounds() {
        return rectangle->getGlobalBounds();
    }

    // Returns number of debris that have been 'generated'
    int getCounter() {
        return counter;
    }

private:
    sf::RectangleShape *rectangle;    // reference to debris
    float x;                         // x location
    float y;                         // y location
    float dx;                        // "change" in x
    float dy;                        // "change" in y
    float width;
    float height;
    int rwmin;
    int rhmin;
    float rheight;
```

```cpp
    float rwidth;
    float d;                        // distance to move
    int counter;
    sf::Vector2f position;

    /**
     * virtual = A virtual function a member function which
     * is declared within base class and is re-defined (Overriden)
     * by derived class.
     * function draw:
     * draw an SFML object to some window
     */
    virtual void draw(sf::RenderTarget &target, sf::RenderStates states) const
       {
        //states.transform *= getTransform();
        target.draw(*rectangle, states);
    }
};

int main() {
    int window_width = 600;
    int window_height = 600;

    sf::RenderWindow window(sf::VideoMode(window_width, window_height),
      "P01");

    Player B(window_width, window_height);
    Debris D(window_width, window_height);

    sf::Font font;
    if (!font.loadFromFile("arial.ttf"))
    {
        std::cout << "Error: no font file!\n";
    }
    sf::Text text;
    text.setFont(font); // font is a sf::Font
    text.setFillColor(sf::Color::Red);
    text.setString("Game Score: 0");
    text.setCharacterSize(30);
    text.setPosition(10, 10);

    int score = 0;      // Incrememts when debris is successfully avoided
    int penalty = 0;    // Decrements when debris is collided with
    std::string scoreString;    // Score is stored as string for display


    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.text.unicode < 128)
                    if (event.text.unicode == 73) {
                        std::cout << "int(" << event.key.code << ") letter(" << char(event.key.code + 65) << ")" << std::endl;
                        std::cout << "UP\n";
                        B.moveVert(0);
                    }

                    else if (event.text.unicode == 74) {
                        std::cout << "int(" << event.key.code << ") letter(" << char(event.key.code + 65) << ")" << std::endl;
                        std::cout << "DOWN\n";
                        B.moveVert(1);
                    }
        }

        B.updateP();
        D.updateD();

        sf::FloatRect recB = B.getBounds();
        sf::FloatRect recD = D.getBounds();

        // Checks for collision and updates score
        if (recB.intersects(recD)) {
            std::cout << "Minus one point!\n";
            penalty += 2;   // Penalty is incremented by 2 because counter increments
                            // by 1 for each debris 'created' so subtracting 2 when
                            // a collision takes place is the practical way of lowering
                            // the score by 1 (counter - penalty = score)
            D.destroy();
        }

        score = D.getCounter() - penalty; // if collision
        // std::cout<<"Score: "<< score << '\n';


        scoreString = "Game Score: " + std::to_string(score);
        std::cout << scoreString << '\n';
        text.setString(scoreString);

        window.clear();
        window.draw(B);
        window.draw(D);
        window.draw(text);
        window.display();
    }

    return 0;
}}
```