Component Code Documentation

> This document outlines the development and testing process for the game component designed for the "Warrior Game". The component includes classes representing game entities such as `Soldier`, `Bullet`, `Enemy`, and `Wall`. Additional game logic for movement, collision detection, and rendering is also encapsulated within the main game loop.

Implementation

write by Kaixing Song and Tunan Zhao

```
class Bullet {
public:
    int x, y, radius;
    double angle;

    Bullet(int x, int y, int radius, double angle) : x(x), y(y), radius(radius),
angle(angle) {}

    void draw() {
        glBegin(GL_POLYGON);
        for(int i = 0; i < 360; i++) {
            #..............
        }
        glEnd();
    }

    void move() {
        y -= BULLET_SPEED * sin(angle);
        x += BULLET_SPEED * cos(angle);
    }
};
```

write by Tianqi Yu

```
class Soldier {
public:
    int x, y, size;
    Soldier(int x, int y, int size) : x(x), y(y), size(size) {}
    void draw() {
        glBegin(GL_QUADS);
        glVertex2i(x, y);
        glVertex2i(x + size, y);
        glVertex2i(x + size, y + size);
        glVertex2i(x, y + size);
        glEnd();
    }

    void move(int dx, int windowWidth) {
        int newX = x + dx * MOVE_STEP; // Calculate the new x coordinate
        if (newX >= 0 && newX <= windowWidth - size) { // Check if the new x
coordinate is within the window
            x = newX; // Update the x coordinate
        }
        y -= 1; // Move the soldier upwards
    }
```

```
    std::vector<Bullet> shoot(int numBullets) {
        std::vector<Bullet> bullets;
        for(int i = 0; i < numBullets; i++) {
            double angle = (180.0 / (numBullets + 1) * (i + 1)) * 3.14159 / 180;
            bullets.push_back(Bullet(x, y, BULLET_RADIUS, angle));
        }
        return bullets;
    }
};
```

Write by Liao Qu

```
class Enemy {
public:
    int x, y, radius;
    bool isMoving;

    Enemy(int x, int y, int radius) : x(x), y(y), radius(radius),
isMoving(false) {}

    void draw() {
        glBegin(GL_POLYGON);
        for(int i = 0; i < 360; i++) {
            double angle = i * 3.14159 / 180;
            double fx = x + cos(angle) * radius;
            double fy = y + sin(angle) * radius;
            glVertex2d(fx, fy);
        }
        glEnd();
    }

    void move(int soldierX, int soldierY) {
        if (isMoving) {
            if (x < soldierX) {
                x += 1;
            } else if (x > soldierX) {
                x -= 1;
            }
            if (y < soldierY) {
                y += 1;
            } else if (y > soldierY) {
                y -= 1;
            }
        }
    }
};
```

Write by Zhihao Zhang

```
class Wall {
public:
    int x1, y1, x2, y2;
    int operation; // 0: add, 1: subtract, 2: multiply, 3: divide
    bool isPassed; // flag to track if a soldier has passed through the wall

    Wall(int x1, int y1, int x2, int y2, int operation) : x1(x1), y1(y1),
x2(x2), y2(y2), operation(operation), isPassed(false) {}

    void draw() {
```

```
        glBegin(GL_LINES);
        glVertex2i(x1, y1);
        glVertex2i(x2, y2);
        glEnd();
    }

    int performOperation(int numSoldiers) {
        switch(operation) {
            case 0: return numSoldiers + 2; // add 2 soldiers
            case 1: return numSoldiers > 2 ? numSoldiers - 2 : 1; // subtract 2
soldiers, but ensure there's at least 1 soldier
            case 2: return numSoldiers * 2; // multiply by 2
            case 3: return numSoldiers > 1 ? numSoldiers / 2 : 1; // divide by
2, but ensure there's at least 1 soldier
            default: return numSoldiers;
        }
    }
};
```

write by Xianwei Zou and Yichen Zheng

```
int main() {
    int windowWidth = 800;
    int windowHeight = 600;
    FsOpenWindow(0, 0, windowWidth, windowHeight, 1);

    std::vector<Soldier> soldiers;
    soldiers.push_back(Soldier(windowWidth / 2, windowHeight - SOLDIER_SIZE,
SOLDIER_SIZE));


    std::vector<Enemy> enemies;
    enemies.push_back(Enemy(400, 300, ENEMY_RADIUS));

    std::vector<Wall> walls;
    walls.push_back(Wall(100, 400, 390, 400, 0)); // Wall with operation 0 (add)
    walls.push_back(Wall(410, 400, 700, 400, 1)); // Wall with operation 1
(subtract)

    std::vector<Bullet> bullets;
    int lastShotTime = FsSubSecondTimer();

    // In the main loop
    while(FsInkey() != FSKEY_ESC) {
        glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

        // Draw road
        glBegin(GL_LINES);
        glVertex2i(100, 0);
        glVertex2i(100, 600);
        glVertex2i(700, 0);
        glVertex2i(700, 600);
        glEnd();

        FsPollDevice();  // Check for user input
        auto key = FsInkey();

        if(FSKEY_ESC == key) // if the user press ESC key
        {
            break;  // Exit the game
        }
```

```
        // Create bullets every SHOOTING_FREQUENCY milliseconds
        if (FsSubSecondTimer() - lastShotTime >= SHOOTING_FREQUENCY) {
            for(auto& soldier : soldiers) {
                auto newBullets = soldier.shoot(soldiers.size());
                bullets.insert(bullets.end(), newBullets.begin(),
newBullets.end());
            }
            lastShotTime = FsSubSecondTimer();
        }

        // Draw and move bullets
        for(auto& bullet : bullets) {
            bullet.draw();
            bullet.move();
        }

        // Draw and move soldiers
        for(auto& soldier : soldiers) {
            soldier.draw();
            if(key == FSKEY_LEFT) {
                soldier.move(-1, windowWidth); // Move the soldier to the left
            } else if(key == FSKEY_RIGHT) {
                soldier.move(1, windowWidth); // Move the soldier to the right
            } else {
                soldier.move(0, windowWidth); // Keep moving upwards
            }
        }

        // Draw and move enemies
        for(auto& enemy : enemies) {
            enemy.draw();
            if (soldiers.size() > 0) {
                enemy.move(soldiers[0].x, soldiers[0].y);
            }
        }

        // Check for collisions between soldiers and enemies
        for (auto it = soldiers.begin(); it != soldiers.end(); ) {
            bool isHit = false;
            for (auto& enemy : enemies) {
                if (abs(it->x - enemy.x) < it->size && abs(it->y - enemy.y) <
enemy.radius) {
                    isHit = true;
                    break;
                }

            }
            if (isHit) {
                it = soldiers.erase(it);
            } else {
                ++it;
            }
        }

        // Check for collisions between bullets and enemies
        for (auto it = bullets.begin(); it != bullets.end(); ) {
            bool isHit = false;
            for (auto jt = enemies.begin(); jt != enemies.end(); ) {
                if (abs(it->x - jt->x) < it->radius + jt->radius && abs(it->y -
jt->y) < it->radius + jt->radius) {
                    isHit = true;
                    jt = enemies.erase(jt);
```

```
                } else {
                    ++jt;
                }
            }
            if (isHit) {
                it = bullets.erase(it);
            } else {
                ++it;
            }
        }

        // Draw walls
        for(auto& wall : walls) {
            wall.draw();
        }

        // Check if soldier passed a wall
        for(auto& wall : walls) {
            if(!wall.isPassed && soldiers.size() > 0 && soldiers[0].y < wall.y1
&& soldiers[0].x >= wall.x1 && soldiers[0].x <= wall.x2) {
                wall.isPassed = true; // set the flag to true
                for(auto& enemy : enemies) {
                    enemy.isMoving = true;
                }
                int numSoldiers = soldiers.size();
                numSoldiers = wall.performOperation(numSoldiers);
                while(soldiers.size() < numSoldiers) {
                    int startX = soldiers[0].x + (soldiers.size() % 2 == 0 ? -
SOLDIER_SIZE - 1 : SOLDIER_SIZE + 1) * ((soldiers.size() + 1) / 2);
                    soldiers.push_back(Soldier(startX, soldiers[0].y,
SOLDIER_SIZE)); // add new soldiers at the same y position as the existing
soldier
                }
                while(soldiers.size() > numSoldiers && soldiers.size() > 1) {
                    soldiers.pop_back(); // remove soldiers from the end
                }
            }
        }

        FsSwapBuffers();
        FsSleep(25);
    }

    return 0;
}
```

Result

The implementation includes the following key classes:

- `Bullet` : Defines the bullet's properties and movement logic.
- `Soldier` : Manages the soldier's state, rendering, movement, and shooting actions.
- `Enemy` : Controls enemy behavior, rendering, and movement towards the soldier after passing a wall.
- `Wall` : Represents a wall that can perform operations on the soldier count when passed.

The game loop handles rendering, input processing, collision detection, and maintains the game state.