Project 9 - David Duffrin

```
/**
 * The Main class initializes and starts a new SpaceWars game.
 */
class Main {

    /** Initializes a new game and starts it. */
    function void main() {
        var SpaceWars game;
        var int difficulty;
        let difficulty = 0;

        do Output.printString("Welcome to SpaceWars!");
        do Output.println();
        do Output.printString("The controls are up and down arrow to move");
        do Output.println();
        do Output.printString("and left arrow to shoot");
        do Output.println();
        do Output.println();
        do Output.printString("Please select your difficulty!");
        do Output.println();
        do Output.printString("1: Easy, 2: Medium, 3: Hard");
        while ((difficulty < 49) | (difficulty > 51)) {
            let difficulty = Keyboard.keyPressed();
        }
        do Screen.clearScreen();
        let game = SpaceWars.new(difficulty);
        do game.run();
        do game.dispose();

        return;
    }
}

/**
 * SpaceWars game.
 * Your player can only move up (arrow up), down (arrow down),
 * shoot (arrow left), or give up (q).
 * Dodge the Enemy's bullets while shooting the Enemy to weaken it (decreases in size)
 * until it dies (you win). If shot, you lose. The Enemy can shoot multiple
 * bullets at a time, however the Player can only shoot one at a time.
 */
class SpaceWars {
```

```
// Variables are self defining
field Player player;
field Bullet bullet;
field Enemy enemy;
field EnemyBullet enemyBullet;
field bool bulletAlive, enemyBulletAlive, exit;
field Array enemyBulletArray;
field int direction, count, bulletCount;

/** Constructs a new Game. */
constructor SpaceWars new(int difficulty) {
    let enemy = Enemy.new();
    let player = Player.new(480, 125, 10);
    if (difficulty = 49) {
        do player.decSize();
    }
    if (difficulty = 51) {
        do player.incSize();
    }

    let direction = 0;
    let bulletAlive = false;
    let enemyBulletAlive = false;
    let exit = false;
    let count = 0;
    let bulletCount = 0;
    let enemyBulletArray = Array.new(12);

    let enemyBulletArray[0] = 0;
    let enemyBulletArray[1] = 0;
    let enemyBulletArray[2] = 0;
    let enemyBulletArray[3] = 0;
    let enemyBulletArray[4] = 0;
    let enemyBulletArray[5] = 0;
    let enemyBulletArray[6] = 0;
    let enemyBulletArray[7] = 0;
    let enemyBulletArray[7] = 0;
    let enemyBulletArray[8] = 0;
    let enemyBulletArray[9] = 0;
    let enemyBulletArray[10] = 0;
    let enemyBulletArray[11] = 0;
```

```
    return this;
}

/** Deallocates the object's memory. */
method void dispose() {
    do player.dispose();
    do enemyBulletArray.dispose();
    do Memory.deAlloc(this);
    return;
}

/** Constantly pull the keyboard memory for updates. */
method void run() {
    var char key;
    while (~exit) {

        let direction = 0;
        let key = Keyboard.keyPressed();

        if (key = 81) { // q for quit
            let exit = true;
        }
        if (key = 131) { // up arrow for up
            let direction = 1;
        }
        if (key = 133) { // down arrow for up
            let direction = 2;
        }
        if (key = 130) { // left arrow for shoot
            let direction = 3;
        }
        do update();
    }
    return;
}

/** Moves the player while updating all bullets and enemy. */
method void update() {
    var int i;
    // first, move the player or shoot
    if (direction = 1) {
        do player.moveUp();
    }
```

```
        if (direction = 2) {
            do player.moveDown();
        }
        if (direction = 3) {
            if (bulletAlive = false) {
                let bullet = Bullet.new((player.getY() + (player.getSize() / 2)));
                let bulletAlive = true;
            }
        }
        // update the enemy
        do enemy.update();
        // if the player's bullet is alive, update the bullet and then check for collisions with the
enemy
        if (bulletAlive = true) {
            do bullet.update();
            if (bullet.getX() < 5) {
                do bullet.erase();
                do bullet.dispose();
                let bulletAlive = false;
            }
            if (bullet.getX() < (enemy.getX() + (enemy.getSize() + 1))) {
                if (bullet.getX() > enemy.getX()) {
                    if (bullet.getY() > (enemy.getY() - 1)) {
                        if (bullet.getY() < (enemy.getY() + (enemy.getSize() + 1))) {
                            // the enemy starts at size 25 with speed 1, and then for each shot becomes
                            // size 19, speed 2 -> size 13, speed 4 -> size 7, speed 8 -> dead
                            if (enemy.getSize() > 6) {
                                do enemy.decSize();
                            }
                            if (enemy.getSize() < 7) {
                                do enemy.dispose();
                                do enemyBullet.dispose();
                                do Screen.clearScreen();
                                do Output.printString("You won!");
                                let exit = true;
                            }
                            let bulletAlive = false;
                            do bullet.erase();
                            do bullet.dispose();
                        }
                    }
                }
            }
```

```
        }
        // have the enemy shoot another bullet every 20 iterations
        let count = count + 1;
        if ((count > 19)  & (exit = false)) {
            let enemyBulletArray[bulletCount] = EnemyBullet.new((enemy.getX() + enemy.getSize()
+ 1), (enemy.getY() + (enemy.getSize() / 2)), (enemy.getSpeed() * 2));
            let bulletCount = bulletCount + 1;
            if (bulletCount > 11) {
                let bulletCount = 0;
            }
            let count = 0;
        }
        // loop through all of the enemy's bullets and update them (move and check for collision)
        let i = 0;
        while ((i < 12) & (exit = false)) {
            if (~(enemyBulletArray[i] = 0)) {
                let enemyBullet = enemyBulletArray[i];
                do enemyBullet.update();
                if (enemyBullet.getX() > 500) {
                    do enemyBullet.erase();
                    do enemyBullet.dispose();
                    let enemyBulletArray[i] = 0;
                }
                if (enemyBullet.getX() > (player.getX() - enemyBullet.getSpeed())) {
                    if (enemyBullet.getX() < (player.getX() + player.getSize())) {
                        if (enemyBullet.getY() > player.getY()) {
                            if (enemyBullet.getY() < (player.getY() + player.getSize())) {
                                let exit = true;
                                let enemyBulletArray[i] = 0;
                                do enemyBullet.dispose();
                                do enemyBulletArray.dispose();
                                do Screen.clearScreen();
                                do Output.printString("You Lost!");
                            }
                        }
                    }
                }
            }
            let i = i + 1;
        }
        do Sys.wait(25); // Delays the next movement.
        return;
    }
```

```
}

/**
 * Creates a player. A player has x and y coordinate location and a size.
 * There are methods for changing the size, drawing, erasing, moving,
 * and returning the size and coordinates.
 */

class Player {

   // Location on the screen
   field int x, y;

   // The size of the player
   field int size;

   /** Constructs a new player with a given location and size. */
   constructor Player new(int Ax, int Ay, int Asize) {
      let x = Ax;
      let y = Ay;
      let size = Asize;
      do draw();
      return this;
   }

   /** Deallocates the object's memory. */
   method void dispose() {
      do Memory.deAlloc(this);
      return;
   }

   /** Draws the player on the screen. */
   method void draw() {
      do Screen.setColor(true);
      do Screen.drawRectangle(x, y, x + size, y + size);
      return;
   }

   /** Erases the player from the screen. */
   method void erase() {
      do Screen.setColor(false);
      do Screen.drawRectangle(x, y, x + size, y + size);
      return;
```

```
}

/** Increments the size by 5 pixels. */
method void incSize() {
   if (((y + size) < 254) & ((x + size) < 510)) {
      do erase();
      let size = size + 5;
      do draw();
   }
   return;
}

/** Decrements the size by 5 pixels. */
method void decSize() {
   if (size > 2) {
      do erase();
      let size = size - 5;
      do draw();
   }
   return;
}

/** Moves up by 2 pixels. */
method void moveUp() {
   if (y > 1) {
      do Screen.setColor(false);
      do Screen.drawRectangle(x, (y + size) - 1, x + size, y + size);
      let y = y - 2;
      do Screen.setColor(true);
      do Screen.drawRectangle(x, y, x + size, y + 1);
   }
   return;
}

/** Moves down by 2 pixel. */
method void moveDown() {
   if ((y + size) < 253) {
      do Screen.setColor(false);
      do Screen.drawRectangle(x, y, x + size, y + 1);
      let y = y + 2;
      do Screen.setColor(true);
      do Screen.drawRectangle(x, (y + size) - 1, x + size, y + size);
   }
```

```
        return;
    }

    /** Returns the current height of the player */
    method int getY() {
        return y;
    }

    /** Returns the current width of the player */
    method int getX() {
        return x;
    }

    /** Returns the current size of the player */
    method int getSize() {
        return size;
    }
}

/**
 * Implements an Enemy. The Enemy moves up and down on the screen and
 * shoots incessantly. When shot, the Enemy decreases in size but
 * doubles its speed.
 */

class Enemy {

    // Location and speed on the screen
    field int x, y, speed;

    // Boolean of if the Enemy will move up or down
    field bool directionUp;

    // The size of the Enemy
    field int size;

    /** Constructs a new Enemy with a given location and size. */
    constructor Enemy new() {
        let x = 10;
        let y = 10;
        let size = 25;
        let directionUp = false;
        let speed = 1;
```

```
    do draw();
    return this;
}

/** Deallocates the object's memory. */
method void dispose() {
    do Memory.deAlloc(this);
    return;
}

/** Draws the Enemy on the screen. */
method void draw() {
    do Screen.setColor(true);
    do Screen.drawRectangle(x, y, x + size, y + size);
    return;
}

/** Erases the Enemy from the screen. */
method void erase() {
    do Screen.setColor(false);
    do Screen.drawRectangle(x, y, x + size, y + size);
    return;
}

/** Updates the Enemy. Moves (speed) spaces in (directionUp's) direction. */
method void update() {
    if (y < (2 + speed)) {
        let directionUp = false;
    }
    if (y > (253 - (size + speed))) {
        let directionUp = true;
    }
    do Screen.setColor(false);
    do Screen.drawRectangle(x, y, x + size, y + size);
    if (directionUp = true) {
        let y = y - speed;
    }
    if (directionUp = false) {
        let y = y + speed;
    }
    do Screen.setColor(true);
    do Screen.drawRectangle(x, y, x + size, y + size);
    return;
```

```
    }

    /** Decrements the size by 6 pixels and doubles the speed. */
    method void decSize() {
        if (size > 6) {
            do erase();
            let size = size - 6;
            do draw();
            let speed = speed * 2;
        }
        return;
    }

    /** Returns the x coordinate of the upper left pixel of the Enemy. */
    method int getX() {
        return x;
    }

    /** Returns the y coordinate of the upper left pixel of the Enemy. */
    method int getY() {
        return y;
    }

    /** Returns the size of the Enemy. */
    method int getSize() {
        return size;
    }

    /** Returns the speed of the Enemy. */
    method int getSpeed() {
        return speed;
    }
}

/**
 * Creates an EnemyBullet. This has x and y coordinate location.
 * There are methods for drawing, erasing, update movement,
 * and returning the coordinates.
 */

class Bullet {

    // Location on the screen
```

```
field int x, y;

// The size of the bullet
field int size;

/** Constructs a new bullet with a given location and size. */
constructor Bullet new(int Ay) {
   let x = 478;
   let y = Ay;
   let size = 1;
   do draw();
   return this;
}

/** Deallocates the object's memory. */
method void dispose() {
   do Memory.deAlloc(this);
   return;
}

/** Draws the bullet on the screen. */
method void draw() {
   do Screen.setColor(true);
   do Screen.drawRectangle(x, y, x + size, y + size);
   return;
}

/** Erases the bullet from the screen. */
method void erase() {
   do Screen.setColor(false);
   do Screen.drawRectangle(x, y, x + size, y + size);
   return;
}

/** Updates the bullet. Moves left 4 spaces. */
method void update() {
   if (x > 1) {
      do Screen.setColor(false);
      do Screen.drawRectangle((x + size) - 1, y, x + size, y + size);
      let x = x - 4;
      do Screen.setColor(true);
      do Screen.drawRectangle(x, y, x + 1, y + size);
   }
```

```
        return;
    }

    /** Returns the current x coordinate of the bullet */
    method int getX() {
        return x;
    }

    /** Returns the current x coordinate of the bullet */
    method int getY() {
        return y;
    }
}

/**
 * Creates an EnemyBullet. This has x and y coordinate location,
 * and speed. There are methods for drawing, erasing, update movement,
 * and returning the speed and coordinates.
 */

class EnemyBullet {

    // Location and speed on the screen
    field int x, y, speed;

    // The size of the EnemyBullet
    field int size;

    /** Constructs a new EnemyBullet with a given location and size. */
    constructor EnemyBullet new(int Ax, int Ay, int Aspeed) {
        let x = Ax;
        let y = Ay;
        let size = 1;
        let speed = Aspeed;
        do draw();
        return this;
    }

    /** Deallocates the object's memory. */
    method void dispose() {
        do Memory.deAlloc(this);
        return;
    }
```

```
    /** Draws the EnemyBullet on the screen. */
    method void draw() {
        do Screen.setColor(true);
        do Screen.drawRectangle(x, y, x + size, y + size);
        return;
    }

    /** Erases the EnemyBullet from the screen. */
    method void erase() {
        do Screen.setColor(false);
        do Screen.drawRectangle(x, y, x + size, y + size);
        return;
    }

    /** Updates the bullet. Moves right (speed) spaces. */
    method void update() {
        if (x < 510) {
            do Screen.setColor(false);
            do Screen.drawRectangle((x + size) - 1, y, x + size, y + size);
            let x = x + speed;
            do Screen.setColor(true);
            do Screen.drawRectangle(x, y, x + 1, y + size);
        }
        return;
    }

    /** Returns the current x coordinate of the EnemyBullet */
    method int getX() {
        return x;
    }

    /** Returns the current y coordinate of the EnemyBullet */
    method int getY() {
        return y;
    }

    /** Returns the current speed of the EnemyBullet */
    method int getSpeed() {
        return speed;
    }
}
```