Урок 3

Разработка каркаса игры

Создание каркаса проекта: игрок, астероиды, пули и их взаимодействие.

Задний фон

Background представляет собой класс, отвечающий за отображение заднего фона. Он содержит в себе внутренний класс Star для описания звезд. При выполнении методов render() и update() происходит просчет состояния заднего фона.

```
public class Background {
    class Star {
        Vector2 position;
        Vector2 velocity;
        float scl;

        public Star() {
            position = new Vector2((float) Math.random() * 1280, (float)
Math.random() * 720);
            velocity = new Vector2((float) (Math.random() - 0.5) * 5f, (float)
(Math.random() - 0.5) * 5f);
            scl = 0.5f + (float) Math.random() / 4.0f;
        }

        public void update(Hero hero, float dt) {
            position.mulAdd(velocity, dt);
            position.mulAdd(hero.velocity, -0.001f);
            float half = textureStar.getWidth() * scl;
            if (position.x < -half) position.x = 1280 + half;
            if (position.x > 1280 + half) position.x = -half;
            if (position.y < -half) position.y = 720 + half;
            if (position.y > 720 + half) position.y = -half;
        }
    }

    Texture texture;
    Texture textureStar;
    Star[] stars;

    public Background() {
        texture = new Texture("bg.png");
        textureStar = new Texture("star16.png");
        stars = new Star[250];
        for (int i = 0; i < stars.length; i++) {
            stars[i] = new Star();
        }
    }

    public void render(SpriteBatch batch) {
        batch.draw(texture, 0, 0);
        for (Star s : stars) {
            batch.draw(textureStar, s.position.x - 8, s.position.y - 8, 8, 8, 16,
```

```
16, s.scl, s.scl, 0, 0, 0, 16, 16, false, false);
        }
    }

    public void update(Hero hero, float dt) {
        for (Star s : stars) {
            s.update(hero, dt);
        }
    }

    public void dispose() {
        texture.dispose();
        textureStar.dispose();
    }
}
```

Игрок

Класс Hero описывает космический корабль игрока, который имеет следующие свойства: координаты, скорость, мощность двигателя, угол и скорость поворота, текстуру, область поражения (hit box), частоту стрельбы.

Ниже представлен полный листинг данного класса. В конструкторе задаем начальное состояние корабля. Метод render() занимается отрисовкой игрока, при этом текстура центрируется по отношению к Vector2 position, и учитывается его угол поворота. При смене кадров происходит обновление логики корабля с помощью метода update(). К вектору положения прибавляется вектор скорости, после чего вектор скорости умножается на скаляр 0.97 для постепенного затухания движения. При нажатии на экран корабль старается развернуться в сторону нажатия с включенными двигателями. Угол поворота всегда находится в пределах от -pi до pi радиан. При изменении координат корабля меняем и координаты области поражения. Если игра запущена на Android, работаем с тачскрином, в противном случае с клавиатурой.

```
// Если угол до точки отличается от текущего угла корабля, стараемся
развернуться в нужную сторону
                if (angle > ang) {
                    if (angle - ang < PI) {
                        angle -= rotationSpeed * dt;
                    } else {
                        angle += rotationSpeed * dt;
                    }
                }
                if (angle < ang) {
                    if (ang - angle < PI) {
                        angle += rotationSpeed * dt;
                    } else {
                        angle -= rotationSpeed * dt;
                    }
                }
```

```
// Увеличиваем мощность двигателя
                currentEnginePower += 100 * dt;
                if (currentEnginePower > maxEnginePower) currentEnginePower =
maxEnginePower;
                velocity.add((float) (currentEnginePower * cos(angle) * dt),
(float) (currentEnginePower * sin(angle) * dt));
            }
        }
// Если игра запущена на десктопе,
        if (!StarGame.isAndroid) {
// управление реализуется на клавиатуре
            if (Gdx.input.isKeyJustPressed(Input.Keys.W)) {
                currentEnginePower = lowEnginePower;
            }
            if (Gdx.input.isKeyPressed(Input.Keys.W)) {
                currentEnginePower += 100 * dt;
                if (currentEnginePower > maxEnginePower) currentEnginePower =
maxEnginePower;
                velocity.add((float) (currentEnginePower * cos(angle) * dt),
(float) (currentEnginePower * sin(angle) * dt));
            }
            if (Gdx.input.isKeyPressed(Input.Keys.A)) {
                angle += rotationSpeed * dt;
            }
            if (Gdx.input.isKeyPressed(Input.Keys.D)) {
                angle -= rotationSpeed * dt;
            }
            if (Gdx.input.isKeyPressed(Input.Keys.L)) {
                fireCounter += dt;
                if (fireCounter > fireRate) {
                    fireCounter = 0;
                    fire();
                }
            }
        }
// Угол корабля держим в пределах от -PI до PI
        if (angle < -PI) angle += 2 * PI;
        if (angle > PI) angle -= 2 * PI;
// Если корабль улетел за экран, перебрасываем его на другую сторону
        if (position.y > 752) position.y = -32;
        if (position.y < -32) position.y = 752;
        if (position.x > 1312) position.x = -32;
        if (position.x < -32) position.x = 1312;
// Перемещаем хитбокс за кораблем
        hitArea.x = position.x;
        hitArea.y = position.y;
    }
// Метод, который занимается выстреливанием пули
    public void fire() {
```

```
        Bullet[] bl = BulletEmitter.getInstance().bullets;
        for (Bullet o : bl) {
            if (!o.active) {
                o.setup(position.x, position.y, 400 * (float) cos(angle), 400 *
(float) sin(angle));
                break;
            }
        }
    }
}
```

Астероиды

Астероиды представлены в игре в качестве «врагов» игрока. Для перехода на новый уровень необходимо уничтожить все астероиды. Когда астероид уничтожается, он создает четыре астероида меньшего размера, самые маленькие астероиды не распадаются на части. У астероидов есть уровень здоровья, при падении которого до нуля происходит его уничтожение. Основная часть логики очень схожа с работой игрового корабля, за тем лишь исключением, что астероид сам управляет собой.

```
public class Asteroid {
    static Texture texture;
    Vector2 position;
    Vector2 velocity;
    float scl;
    float angle;
    int hp;
    int hpMax;
    Circle hitArea;

    public Asteroid(Vector2 position, Vector2 velocity, float scl, int hpMax) {
        if (texture == null) {
            texture = new Texture("asteroid.png");
        }
        this.position = position;
        this.velocity = velocity;
        this.scl = scl;
        this.hpMax = hpMax;
        this.hp = hpMax;
        this.angle = 0.0f;
        this.hitArea = new Circle(position.x, position.y, 120 * scl);
    }

    public void render(SpriteBatch batch) {
        batch.draw(texture, position.x - 128, position.y - 128, 128, 128, 256,
256, scl, scl, angle, 0, 0, 256, 256, false, false);
    }
```

```
// При получении урона метод takeDamage вернет boolean, который показывает,
уничтожен ли астероид или нет
    public boolean takeDamage(int dmg) {
        hp -= dmg;
        if (hp <= 0) {
            return true;
        }
        return false;
    }
// Как и игрок, астероид на каждом кадре пролетает какое-то расстояние,
производит проверку вылета за экран, двигает за собой область поражения
    public void update(float dt) {
        position.mulAdd(velocity, dt);
        if (position.x < -128 * scl) position.x = 1280 + 128 * scl;
        if (position.x > 1280 + 128 * scl) position.x = -128 * scl;
        if (position.y < -128 * scl) position.y = 720 + 128 * scl;
        if (position.y > 720 + 128 * scl) position.y = -128 * scl;
        hitArea.x = position.x;
        hitArea.y = position.y;
    }
}
```

Пули

Класс Bullet отвечает за работу с пулями. Они могут быть активными (лететь по экрану) либо неактивными (просто лежать в запасе). При выполнении метода setup() пуля активируется и вылетает из указанной точки. Метод destroy() выполняется для деактивации пули (например, в случае, когда она вылетела за экран).

```
public class Bullet {
    Vector2 position;
    Vector2 velocity;
    boolean active;

    public Bullet() {
        this.position = new Vector2(0, 0);
        this.velocity = new Vector2(0, 0);
        this.active = false;
    }

    public void setup(float x, float y, float vx, float vy) {
        position.set(x, y);
        velocity.set(vx, vy);
        active = true;
    }

    public void destroy() {
        active = false;
    }
```

```
    public void update(float dt) {
        position.mulAdd(velocity, dt);
        if (position.x < -20 || position.x > 1300 || position.y < -20 ||
position.y > 740) {
            destroy();
        }
    }
}
```

Управляем пулями (BulletEmitter)

Класс BulletEmitter занимается управлением пулями и является синглтоном. Пока что этот класс представляет собой просто хранилище для массива из 200 пуль. При выполнении методов update() и render() он обновляет и отрисовывает только активные пули.

```
public class BulletEmitter {
    private static final BulletEmitter ourInstance = new BulletEmitter();

    public static BulletEmitter getInstance() {
        return ourInstance;
    }

    Texture texture;
    Bullet[] bullets;

    private BulletEmitter() {
        texture = new Texture("bullet.png");
        bullets = new Bullet[200];
        for (int i = 0; i < bullets.length; i++) {
            bullets[i] = new Bullet();
        }
    }

    public void update(float dt) {
        for (Bullet o : bullets) {
            if (o.active) {
                o.update(dt);
            }
        }
    }

    public void render(SpriteBatch batch) {
        for (Bullet o : bullets) {
            if (o.active) {
                batch.draw(texture, o.position.x - 16, o.position.y - 16);
            }
        }
```

```
    }
}
```

Простое управление

Класс InputHandler занимается работой с модулем ввода Gdx.input.

```
public class InputHandler {
    public static boolean isTouched() {
        return Gdx.input.isTouched();
    }

    public static boolean isJustTouched() {
        return Gdx.input.justTouched();
    }

    public static float getX() {
        return Gdx.input.getX();
    }

    public static float getY() {
        return Gdx.graphics.getHeight() - Gdx.input.getY();
    }
}
```

Основной класс

Класс StarGame собирает все классы воедино.

```
public class StarGame extends ApplicationAdapter {
    public static boolean isAndroid = false;
    SpriteBatch batch;
    Background background;
    Hero hero;

    @Override
    public void create() {
        batch = new SpriteBatch();
        background = new Background();
        hero = new Hero();
        for (int i = 0; i < 4; i++) {
            AsteroidEmitter.getInstance().addAsteroid(new Vector2((float)
Math.random() * 1280, (float) Math.random() * 720), new Vector2((float)
(Math.random() - 0.5) * 200, (float) (Math.random() - 0.5) * 200), 1.0f, 100);
```

```
        }
    }

    @Override
    public void render() {
        float dt = Gdx.graphics.getDeltaTime();
        update(dt);
        Gdx.gl.glClearColor(1, 1, 1, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
        batch.begin();
        background.render(batch);
        hero.render(batch);
        AsteroidEmitter.getInstance().render(batch);
        BulletEmitter.getInstance().render(batch);
        batch.end();
    }

    public void update(float dt) {
        background.update(hero, dt);
        hero.update(dt);
        AsteroidEmitter.getInstance().update(dt);
        BulletEmitter.getInstance().update(dt);
        checkCollision();
    }

    @Override
    public void dispose() {
        batch.dispose();
    }
// Метод checkCollision занимается проверкой столкновений. Первый цикл за
столкновения игрока с астероидами, второй - пуль с астероидами. Проверка
осуществляется за счет сравнения окружностей (построенных вокруг объектов), если
две окружности пересекаются, значит столкновение есть.
    public void checkCollision() {
        for (Asteroid o : AsteroidEmitter.getInstance().asteroids) {
            if (hero.hitArea.overlaps(o.hitArea)) {
                Vector2 acc = hero.position.cpy().sub(o.position).nor();
                hero.velocity.mulAdd(acc, 20);
                o.velocity.mulAdd(acc, -20);
            }
        }
        for (Bullet b : BulletEmitter.getInstance().bullets) {
            if (b.active) {
                for (Asteroid a : AsteroidEmitter.getInstance().asteroids) {
                    if (a.hitArea.contains(b.position)) {
                        a.takeDamage(50);
                        b.destroy();
                    }
                }
```

```
                }
            }
        }
    }
```

Домашнее задание

С домашним заданием вы можете ознакомиться на странице урока.

Дополнительные материалы

Нет рекомендаций по дополнительным материалам для данного занятия, так как оно имеет чисто практический характер.

Используемая литература

При подготовке данного методического пособия литературные источники не использовались.