GameObjects

Understanding the GameObject system

Overview

In Origami Engine, all game entities (players, enemies, walls, items, etc.) are **GameObjects**. Every GameObject is a TypeScript class that extends the base `GameObject` class.

Basic GameObject Structure

```
import { GameObject } from '../lib/origami-runtime.js';

export class obj_enemy extends GameObject {
  private health: number = 100;

  create(): void {
    // Called when instance is created
    this.sprite_index = 'spr_enemy';
    this.speed = 2;
  }

  step(): void {
    // Called every frame (60 FPS)
    // Game logic here
  }

  draw(): void {
    // Called for rendering
    draw_self.call(this);
  }

  roomStart(): void {
    // Called when room loads
  }

  roomEnd(): void {
    // Called when leaving room
  }
}
```

Event Methods

`create()`

Called **once** when an instance is created.

Use for:

- Setting initial properties
- Assigning sprites
- Initializing variables

```
create(): void {
  this.sprite_index = 'spr_player';
  this.x = 100;
  this.y = 100;
  this.speed = 4;
}
```

`step()`

Called **every frame** (60 times per second).

Use for:

- Movement logic
- Collision detection
- Game state updates
- Input handling

```
step(): void {
  // Movement
  if (keyboard_check(vk_right)) this.x += 4;
  if (keyboard_check(vk_left)) this.x -= 4;

  // Collision
  if (place_meeting.call(this, this.x, this.y, 'obj_wall')) {
    // Handle collision
  }
}
```

`draw()`

Called **every frame** during the rendering phase.

Use for:

- Custom drawing
- Visual effects
- UI elements

```
draw(): void {
  // Draw the sprite
  draw_self.call(this);

  // Draw health bar above object
  const barWidth = 50;
  const healthPercent = this.health / this.maxHealth;

  draw_set_color('#FF0000');
  draw_rectangle(
    this.x - 25,
    this.y - 35,
    this.x - 25 + (barWidth * healthPercent),
    this.y - 30,
    false
  );
  draw_set_color('#FFFFFF');
}
```

`roomStart()`

Called when the room first loads.

Use for:

- Room-specific initialization
- Setting up level state

```
roomStart(): void {
  // Reset player health at start of level
  this.health = 100;
}
```

`roomEnd()`

Called when transitioning to a new room.

Use for:

- Cleanup
- Saving state
- Stopping sounds/animations

```
roomEnd(): void {
  // Save player state before leaving
  localStorage.setItem('playerHealth', this.health.toString());
}
```

Built-in Properties

Every GameObject automatically has these properties:

Position

```
x, y              // Current position
xprevious, yprevious  // Position last frame
xstart, ystart    // Initial creation position
```

Motion

```
speed             // Movement speed (pixels per frame)
direction         // Movement direction (degrees, 0-360)
hspeed            // Horizontal speed component
vspeed            // Vertical speed component
```

Speed vs hspeed/vspeed:

- `speed` and `direction` work together (polar coordinates)
- `hspeed` and `vspeed` work independently (cartesian coordinates)

- Setting `speed` updates `hspeed` / `vspeed` automatically
- Setting `hspeed` / `vspeed` updates `speed` / `direction` automatically

Sprite & Animation

```
sprite_index      // Current sprite name ('spr_player')
image_index       // Current animation frame (0, 1, 2...)
image_speed       // Animation speed (1.0 = normal)
image_alpha       // Opacity (0.0 = invisible, 1.0 = solid)
image_angle       // Rotation (degrees)
image_xscale      // Horizontal scale (1.0 = normal)
image_yscale      // Vertical scale (1.0 = normal)
```

Rendering

```
visible           // Whether to draw (true/false)
depth             // Draw order (lower = in front)
order             // Update order within same depth
```

Other

```
persistent        // Stays between rooms (true/false)
```

Instance Management

Creating Instances

```
// Create new instance at position
await instance_create(100, 200, 'obj_bullet');

// In your object's step():
step(): void {
  if (keyboard_check_pressed(vk_space)) {
    await instance_create(this.x, this.y, 'obj_bullet');
  }
}
```

Destroying Instances

```
// Destroy this instance
instance_destroy.call(this);

// Destroy specific instance
if (this.health <= 0) {
  instance_destroy.call(this);
}
```

Finding Instances

```
// Check if any instance exists
if (instance_exists('obj_player')) {
  // Player exists
}

// Find specific instance
const enemy = instance_find('obj_enemy', 0); // Get first enemy
if (enemy) {
  console.log(`Enemy at: ${enemy.x}, ${enemy.y}`);
}
```

Common Patterns

Private Variables

```
export class obj_player extends GameObject {
  private health: number = 100;
  private maxHealth: number = 100;
  private invincible: boolean = false;

  step(): void {
    if (!this.invincible) {
      // Take damage logic
    }
  }
}
```

State Machines

```
enum PlayerState {
  Idle,
  Running,
  Jumping,
  Falling
}

export class obj_player extends GameObject {
  private state: PlayerState = PlayerState.Idle;

  step(): void {
    switch (this.state) {
      case PlayerState.Idle:
        this.handleIdle();
        break;
      case PlayerState.Running:
        this.handleRunning();
        break;
      case PlayerState.Jumping:
        this.handleJumping();
        break;
      case PlayerState.Falling:
        this.handleFalling();
        break;
    }
  }

  private handleIdle(): void {
    if (keyboard_check(vk_d) || keyboard_check(vk_a)) {
      this.state = PlayerState.Running;
    }
  }

  private handleRunning(): void {
    // Running logic
  }
```

```
  // ... other states
}
```

Timers

```
export class obj_spawner extends GameObject {
  private spawnTimer: number = 0;
  private readonly SPAWN_DELAY = 120; // 2 seconds at 60 FPS

  step(): void {
    this.spawnTimer++;

    if (this.spawnTimer >= this.SPAWN_DELAY) {
      await instance_create(this.x, this.y, 'obj_enemy');
      this.spawnTimer = 0;
    }
  }
}
```

Registering Objects

All GameObjects must be registered in `src/main.ts`:

```
import { obj_player } from '../objects/obj_player.js';
import { obj_wall } from '../objects/obj_wall.js';
import { obj_enemy } from '../objects/obj_enemy.js';

// After creating engine:
engine.registerObject(obj_player);
engine.registerObject(obj_wall);
engine.registerObject(obj_enemy);
```

Important: Objects must be registered before `engine.start()`.

Next Steps

- **05-sprites.md** - Working with sprites and animation
- **06-collision.md** - Collision detection
- **08-input.md** - Keyboard and mouse input
- **40-common-patterns.md** - More advanced patterns

← Back to Index