Motion API

Movement and physics functions

Overview

Motion functions help with movement, angles, and physics calculations. These are commonly used in `step()` events.

Direction Functions

`point_direction()`

Gets angle between two points.

Syntax: `point_direction(x1, y1, x2, y2)`

Arguments:

- `x1`, `y1` (number) - First point
- `x2`, `y2` (number) - Second point

Returns: `number` - Angle in degrees (0-360, GMS-style)

Angles:

- `0` = Right (→)
- `90` = Up (↑)
- `180` = Left (←)
- `270` = Down (↓)

Example:

```
step(): void {
  // Point at player
  const player = instance_find('obj_player', 0);
  if (player) {
    this.direction = point_direction(this.x, this.y, player.x, player.y);
  }

  // Point at mouse
  this.image_angle = point_direction(this.x, this.y, mouse_x, mouse_y);

  // Fire bullet towards target
  if (keyboard_check_pressed(vk_space)) {
    const bullet = await instance_create(this.x, this.y, 'obj_bullet');
    bullet.direction = point_direction(this.x, this.y, this.targetX, this.targe
    bullet.speed = 10;
  }
}
```

`point_distance()`

Gets distance between two points.

Syntax: `point_distance(x1, y1, x2, y2)`

Arguments:

- `x1` , `y1` (number) - First point
- `x2` , `y2` (number) - Second point

Returns: `number` - Distance in pixels

Example:

```
step(): void {
  const player = instance_find('obj_player', 0);
  if (player) {
    const dist = point_distance(this.x, this.y, player.x, player.y);

    if (dist < 100) {
      // Player is close, chase
      this.state = 'chase';
    } else if (dist > 300) {
      // Player is far, stop chasing
      this.state = 'patrol';
    }
  }

  // Check if reached target
  const dist = point_distance(this.x, this.y, this.targetX, this.targetY);
  if (dist < 5) {
    this.reachedTarget = true;
  }
}
```

Vector Functions

`lengthdir_x()`

Gets X component of a vector.

Syntax: `lengthdir_x(length, direction)`

Arguments:

- `length` (number) - Vector length (magnitude)
- `direction` (number) - Angle in degrees (0-360)

Returns: `number` - X component

Example:

```
step(): void {
  // Move in a direction
  this.x += lengthdir_x(5, this.direction);
  this.y += lengthdir_y(5, this.direction);

  // Move towards point
  const dir = point_direction(this.x, this.y, mouse_x, mouse_y);
  this.hspeed = lengthdir_x(4, dir);
  this.vspeed = lengthdir_y(4, dir);

  // Orbit around point
  this.angle += 2;
  this.x = this.centerX + lengthdir_x(100, this.angle);
  this.y = this.centerY + lengthdir_y(100, this.angle);
}
```

`lengthdir_y()`

Gets Y component of a vector.

Syntax: `lengthdir_y(length, direction)`

Arguments:

- `length` (number) - Vector length
- `direction` (number) - Angle in degrees

Returns: `number` - Y component

Example:

```
step(): void {
  // Apply force in direction
  const pushForce = 10;
  const pushDir = 45; // Diagonal
  this.hspeed += lengthdir_x(pushForce, pushDir);
  this.vspeed += lengthdir_y(pushForce, pushDir);

  // Recoil effect
  if (this.shooting) {
    const recoilDir = this.image_angle + 180; // Opposite direction
    this.x += lengthdir_x(2, recoilDir);
    this.y += lengthdir_y(2, recoilDir);
  }
}
```

Movement Functions

`move_towards_point()`

Moves instance towards a point.

Syntax: `move_towards_point.call(this, x, y, speed)`

Arguments:

- `x` , `y` (number) - Target point
- `speed` (number) - Movement speed

Returns: `void`

Description: Sets the instance's `speed` and `direction` to move towards the target point.

Example:

```
step(): void {
  // Chase player
  const player = instance_find('obj_player', 0);
  if (player) {
    move_towards_point.call(this, player.x, player.y, 3);
  }

  // Move to waypoint
  if (this.currentWaypoint < this.waypoints.length) {
    const waypoint = this.waypoints[this.currentWaypoint];
    move_towards_point.call(this, waypoint.x, waypoint.y, 2);

    // Check if reached
    const dist = point_distance(this.x, this.y, waypoint.x, waypoint.y);
    if (dist < 5) {
      this.currentWaypoint++;
    }
  }
}
```

Motion Patterns

Chase Player

```
step(): void {
  const player = instance_find('obj_player', 0);
  if (!player) return;

  const dist = point_distance(this.x, this.y, player.x, player.y);

  if (dist < 200 && dist > 30) {
    // Chase if within range but not too close
    const dir = point_direction(this.x, this.y, player.x, player.y);
    this.x += lengthdir_x(2, dir);
    this.y += lengthdir_y(2, dir);
  }
}
```

Flee from Player

```
step(): void {
  const player = instance_find('obj_player', 0);
  if (!player) return;

  const dist = point_distance(this.x, this.y, player.x, player.y);

  if (dist < 150) {
    // Run away if player is close
    const dir = point_direction(player.x, player.y, this.x, this.y); // Opposit
    this.x += lengthdir_x(3, dir);
    this.y += lengthdir_y(3, dir);
  }
}
```

Orbit Point

```
private angle: number = 0;
private readonly ORBIT_RADIUS = 100;
private readonly ORBIT_SPEED = 2;

step(): void {
  this.angle += this.ORBIT_SPEED;

  this.x = this.centerX + lengthdir_x(this.ORBIT_RADIUS, this.angle);
  this.y = this.centerY + lengthdir_y(this.ORBIT_RADIUS, this.angle);
}
```

Figure-8 Pattern

```
private t: number = 0;

step(): void {
  this.t += 0.05;

  this.x = this.centerX + Math.sin(this.t) * 100;
  this.y = this.centerY + Math.sin(this.t * 2) * 50;
}
```

Bounce Pattern

```
step(): void {
  // Move
  this.x += this.hspeed;
  this.y += this.vspeed;

  // Bounce off walls
  if (this.x <= 0 || this.x >= room_width) {
    this.hspeed *= -1;
  }
  if (this.y <= 0 || this.y >= room_height) {
    this.vspeed *= -1;
  }
}
```

Homing Missile

```
step(): void {
  const player = instance_find('obj_player', 0);
  if (!player) return;

  // Current direction
  const currentDir = this.direction;

  // Desired direction
  const targetDir = point_direction(this.x, this.y, player.x, player.y);

  // Turn towards target gradually
  const turnSpeed = 3;
  let diff = targetDir - currentDir;

  // Normalize angle difference
  if (diff > 180) diff -= 360;
  if (diff < -180) diff += 360;

  if (Math.abs(diff) < turnSpeed) {
    this.direction = targetDir;
  } else {
    this.direction += Math.sign(diff) * turnSpeed;
  }

  // Maintain speed
  this.speed = 5;
}
```

Wave Movement

```
private waveTime: number = 0;

step(): void {
  this.waveTime += 0.1;

  // Move forward
  this.x += 3;

  // Wave up and down
  this.y += Math.sin(this.waveTime) * 2;
}
```

Zigzag Movement

```
private zigzagTimer: number = 0;
private zigzagDirection: number = 1;

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

  // Move forward
  this.x += 2;

  // Zigzag
  this.y += this.zigzagDirection * 3;

  // Change direction every 30 frames
  if (this.zigzagTimer >= 30) {
    this.zigzagDirection *= -1;
    this.zigzagTimer = 0;
  }
}
```

Spiral Movement

```
private angle: number = 0;
private radius: number = 0;

step(): void {
  this.angle += 5;
  this.radius += 0.5;

  this.x = this.centerX + lengthdir_x(this.radius, this.angle);
  this.y = this.centerY + lengthdir_y(this.radius, this.angle);
}
```

Physics Patterns

Gravity

```
private readonly GRAVITY = 0.5;
private readonly MAX_FALL_SPEED = 10;

step(): void {
  // Apply gravity
  this.vspeed += this.GRAVITY;

  // Cap fall speed
  if (this.vspeed > this.MAX_FALL_SPEED) {
    this.vspeed = this.MAX_FALL_SPEED;
  }

  // Apply vertical movement
  this.y += this.vspeed;
}
```

Friction

```
private readonly FRICTION = 0.9;

step(): void {
  // Apply friction
  this.hspeed *= this.FRICTION;
  this.vspeed *= this.FRICTION;

  // Stop if very slow
  if (Math.abs(this.hspeed) < 0.1) this.hspeed = 0;
  if (Math.abs(this.vspeed) < 0.1) this.vspeed = 0;
}
```

Knockback

```
takeDamage(amount: number, fromX: number, fromY: number): void {
  this.health -= amount;

  // Knockback away from damage source
  const knockbackForce = 8;
  const dir = point_direction(fromX, fromY, this.x, this.y);
  this.hspeed = lengthdir_x(knockbackForce, dir);
  this.vspeed = lengthdir_y(knockbackForce, dir);
}
```

Acceleration

```
private readonly ACCELERATION = 0.5;
private readonly MAX_SPEED = 5;

step(): void {
  if (keyboard_check(vk_d)) {
    this.hspeed += this.ACCELERATION;
  }
  if (keyboard_check(vk_a)) {
    this.hspeed -= this.ACCELERATION;
  }

  // Cap speed
  if (this.hspeed > this.MAX_SPEED) this.hspeed = this.MAX_SPEED;
  if (this.hspeed < -this.MAX_SPEED) this.hspeed = -this.MAX_SPEED;

  // Apply movement
  this.x += this.hspeed;
}
```

Useful Calculations

Smooth Following

```
step(): void {
  const player = instance_find('obj_player', 0);
  if (player) {
    // Smooth lerp (20% of distance per frame)
    this.x += (player.x - this.x) * 0.2;
    this.y += (player.y - this.y) * 0.2;
  }
}
```

Prediction

```
step(): void {
  const player = instance_find('obj_player', 0);
  if (player) {
    // Predict future position
    const bulletSpeed = 8;
    const dist = point_distance(this.x, this.y, player.x, player.y);
    const timeToReach = dist / bulletSpeed;

    const predictedX = player.x + player.hspeed * timeToReach;
    const predictedY = player.y + player.vspeed * timeToReach;

    // Aim at predicted position
    this.direction = point_direction(this.x, this.y, predictedX, predictedY);
  }
}
```

Next Steps

- **26-api-math.md** - Math utilities
- **24-api-input.md** - Input functions
- **40-common-patterns.md** - Movement patterns

← Back to Index