# Input

**Keyboard and mouse handling**

## Overview

Origami Engine provides GameMaker-style input functions for keyboard and mouse. All input functions are global and can be called from any object's `step()` event.

## Keyboard Functions

### `keyboard_check()`

Checks if a key is currently held down.

**Syntax**: `keyboard_check(key)`

**Arguments**:

- `key` (number) - Virtual key constant

**Returns**: `boolean` - True if key is down

**Example**:

```
step(): void {
  // Continuous movement while holding key
  if (keyboard_check(vk_right)) {
    this.x += 4;
  }
  if (keyboard_check(vk_left)) {
    this.x -= 4;
  }
  if (keyboard_check(vk_up)) {
    this.y -= 4;
  }
  if (keyboard_check(vk_down)) {
    this.y += 4;
  }
}
```

**Use case**: Actions that should happen continuously while key is held (movement, shooting, etc.).

---

### `keyboard_check_pressed()`

Checks if a key was **just pressed** this frame.

**Syntax**: `keyboard_check_pressed(key)`

**Arguments**:

- `key` (number) - Virtual key constant

**Returns**: `boolean` - True if just pressed

**Example**:

```
step(): void {
  // Jump only on initial press
  if (keyboard_check_pressed(vk_space)) {
    if (this.onGround) {
      this.vspeed = -10;
    }
  }

  // Toggle debug mode
  if (keyboard_check_pressed(vk_f3)) {
    this.debugMode = !this.debugMode;
  }
}
```

**Use case**: Single-action events (jumping, firing, toggling, menu navigation).

---

### `keyboard_check_released()`

Checks if a key was **just released** this frame.

**Syntax**: `keyboard_check_released(key)`

**Arguments**:

- `key` (number) - Virtual key constant

**Returns**: `boolean` - True if just released

**Example**:

```
step(): void {
  // Charge jump while holding, release to jump
  if (keyboard_check(vk_space)) {
    this.jumpCharge++;
  }
  if (keyboard_check_released(vk_space)) {
    this.vspeed = -this.jumpCharge;
    this.jumpCharge = 0;
  }
}
```

**Use case**: Charge attacks, hold-to-aim mechanics.

---

# Keyboard Constants

## Letter Keys

```
vk_a, vk_b, vk_c, vk_d, vk_e, vk_f, vk_g, vk_h, vk_i, vk_j,
vk_k, vk_l, vk_m, vk_n, vk_o, vk_p, vk_q, vk_r, vk_s, vk_t,
vk_u, vk_v, vk_w, vk_x, vk_y, vk_z
```

**Example**:

```
if (keyboard_check(vk_w)) this.y -= 4;
if (keyboard_check(vk_a)) this.x -= 4;
if (keyboard_check(vk_s)) this.y += 4;
if (keyboard_check(vk_d)) this.x += 4;
```

## Number Keys

```
vk_0, vk_1, vk_2, vk_3, vk_4, vk_5, vk_6, vk_7, vk_8, vk_9
```

## Arrow Keys

```
vk_left, vk_right, vk_up, vk_down
```

**Example**:

```
if (keyboard_check(vk_right)) this.image_xscale = 1;
if (keyboard_check(vk_left)) this.image_xscale = -1;
```

### Special Keys

```
vk_space     // Spacebar
vk_enter     // Enter/Return
vk_shift     // Shift
vk_control   // Ctrl
vk_alt       // Alt
vk_escape    // Esc
vk_backspace // Backspace
vk_tab       // Tab
```

### Function Keys

```
vk_f1, vk_f2, vk_f3, vk_f4, vk_f5, vk_f6,
vk_f7, vk_f8, vk_f9, vk_f10, vk_f11, vk_f12
```

**Example**:

```
if (keyboard_check_pressed(vk_f3)) {
  globalThis.debugMode = !globalThis.debugMode;
}
```

---

## Mouse Functions

`mouse_check_button()`

Checks if mouse button is currently held.

**Syntax**: `mouse_check_button(button)`

**Arguments**:

- `button` (number) - Button constant (mb_left, mb_right, mb_middle)

**Returns**: `boolean` - True if button is down

**Example**:

```
step(): void {
  // Fire weapon while holding
  if (mouse_check_button(mb_left)) {
    this.shootTimer--;
    if (this.shootTimer <= 0) {
      await instance_create(this.x, this.y, 'obj_bullet');
      this.shootTimer = 10;
    }
  }
}
```

---

### `mouse_check_button_pressed()`

Checks if mouse button was **just pressed**.

**Syntax**: `mouse_check_button_pressed(button)`

**Arguments**:

- `button` (number) - Button constant

**Returns**: `boolean` - True if just pressed

**Example**:

```
step(): void {
  // Single shot on click
  if (mouse_check_button_pressed(mb_left)) {
    await instance_create(this.x, this.y, 'obj_bullet');
  }

  // Right-click for special ability
  if (mouse_check_button_pressed(mb_right)) {
    this.useSpecialAbility();
  }
}
```

---

### `mouse_check_button_released()`

Checks if mouse button was **just released**.

**Syntax**: `mouse_check_button_released(button)`

**Arguments**:

- `button` (number) - Button constant

**Returns**: `boolean` - True if just released

**Example**:

```
step(): void {
  // Charge shot
  if (mouse_check_button(mb_left)) {
    this.chargeAmount++;
  }
  if (mouse_check_button_released(mb_left)) {
    this.fireChargedShot(this.chargeAmount);
    this.chargeAmount = 0;
  }
}
```

---

## Mouse Constants

```
mb_left    // Left mouse button
mb_right   // Right mouse button
mb_middle  // Middle mouse button (scroll wheel click)
```

---

# Mouse Position

## Global Variables

```
mouse_x  // Mouse X coordinate in room
mouse_y  // Mouse Y coordinate in room
```

**Example**:

```
step(): void {
  // Point at mouse
  this.image_angle = point_direction(this.x, this.y, mouse_x, mouse_y);

  // Move towards mouse
  const dir = point_direction(this.x, this.y, mouse_x, mouse_y);
  this.x += lengthdir_x(this.speed, dir);
  this.y += lengthdir_y(this.speed, dir);
}
```

---

# Common Input Patterns

## WASD Movement

```
step(): void {
  const speed = 4;

  if (keyboard_check(vk_w)) this.y -= speed;
  if (keyboard_check(vk_s)) this.y += speed;
  if (keyboard_check(vk_a)) this.x -= speed;
  if (keyboard_check(vk_d)) this.x += speed;
}
```

## 8-Direction Movement

```
step(): void {
  let xdir = 0;
  let ydir = 0;

  if (keyboard_check(vk_right)) xdir += 1;
  if (keyboard_check(vk_left)) xdir -= 1;
  if (keyboard_check(vk_down)) ydir += 1;
  if (keyboard_check(vk_up)) ydir -= 1;

  if (xdir !== 0 || ydir !== 0) {
    const dir = point_direction(0, 0, xdir, ydir);
    this.x += lengthdir_x(4, dir);
    this.y += lengthdir_y(4, dir);
  }
}
```

## Platformer Controls

```
step(): void {
  const moveSpeed = 4;
  const jumpPower = -10;

  // Horizontal movement
  if (keyboard_check(vk_d)) {
    this.hspeed = moveSpeed;
    this.image_xscale = 1; // Face right
  } else if (keyboard_check(vk_a)) {
    this.hspeed = -moveSpeed;
    this.image_xscale = -1; // Face left
  } else {
    this.hspeed = 0;
  }

  // Jump
  if (keyboard_check_pressed(vk_space) && this.onGround) {
    this.vspeed = jumpPower;
  }
}
```

## Mouse Aiming

```
step(): void {
  // Aim at mouse
  this.image_angle = point_direction(this.x, this.y, mouse_x, mouse_y);

  // Shoot on click
  if (mouse_check_button_pressed(mb_left)) {
    const bullet = await instance_create(this.x, this.y, 'obj_bullet');
    if (bullet) {
      bullet.direction = this.image_angle;
      bullet.speed = 10;
    }
  }
}
```

## Click to Move

```
private targetX: number = 0;
private targetY: number = 0;
private moving: boolean = false;

step(): void {
  // Set target on click
  if (mouse_check_button_pressed(mb_left)) {
    this.targetX = mouse_x;
    this.targetY = mouse_y;
    this.moving = true;
  }

  // Move towards target
  if (this.moving) {
    const dist = point_distance(this.x, this.y, this.targetX, this.targetY);
    if (dist > 2) {
      const dir = point_direction(this.x, this.y, this.targetX, this.targetY);
      this.x += lengthdir_x(4, dir);
      this.y += lengthdir_y(4, dir);
    } else {
      this.moving = false;
    }
  }
}
```

## Drag Objects

```
private dragging: boolean = false;
private dragOffsetX: number = 0;
private dragOffsetY: number = 0;

step(): void {
  // Start dragging
  if (mouse_check_button_pressed(mb_left)) {
    const dist = point_distance(this.x, this.y, mouse_x, mouse_y);
    if (dist < 20) {
      this.dragging = true;
      this.dragOffsetX = this.x - mouse_x;
      this.dragOffsetY = this.y - mouse_y;
    }
  }

  // While dragging
  if (this.dragging) {
    this.x = mouse_x + this.dragOffsetX;
    this.y = mouse_y + this.dragOffsetY;
  }

  // Stop dragging
  if (mouse_check_button_released(mb_left)) {
    this.dragging = false;
  }
}
```

### Shoot Cooldown

```
private shootCooldown: number = 0;
private readonly SHOOT_DELAY = 10; // 10 frames

step(): void {
  // Decrease cooldown
  if (this.shootCooldown > 0) {
    this.shootCooldown--;
  }

  // Shoot if cooldown ready
  if (keyboard_check(vk_space) && this.shootCooldown === 0) {
    await instance_create(this.x, this.y, 'obj_bullet');
    this.shootCooldown = this.SHOOT_DELAY;
  }
}
```

---

## Input Buffering

Handle input more reliably:

```
private jumpBuffer: number = 0;
private readonly BUFFER_FRAMES = 5;

step(): void {
  // Buffer jump input
  if (keyboard_check_pressed(vk_space)) {
    this.jumpBuffer = this.BUFFER_FRAMES;
  }

  // Decrease buffer
  if (this.jumpBuffer > 0) {
    this.jumpBuffer--;
  }

  // Jump if buffered and on ground
  if (this.jumpBuffer > 0 && this.onGround) {
    this.vspeed = -10;
    this.jumpBuffer = 0;
  }
}
```

---

## Combo System

Detect button sequences:

```
private comboKeys: number[] = [];
private comboTimer: number = 0;
private readonly COMBO_WINDOW = 30; // 0.5 seconds

step(): void {
  // Decrease combo timer
  if (this.comboTimer > 0) {
    this.comboTimer--;
  } else {
    this.comboKeys = []; // Reset if too slow
  }

  // Record keypresses
  if (keyboard_check_pressed(vk_a)) {
    this.comboKeys.push(vk_a);
    this.comboTimer = this.COMBO_WINDOW;
    this.checkCombos();
  }
  if (keyboard_check_pressed(vk_s)) {
    this.comboKeys.push(vk_s);
    this.comboTimer = this.COMBO_WINDOW;
    this.checkCombos();
  }
  if (keyboard_check_pressed(vk_d)) {
    this.comboKeys.push(vk_d);
    this.comboTimer = this.COMBO_WINDOW;
    this.checkCombos();
  }
}

private checkCombos(): void {
  // Check for A-S-D combo
  if (this.comboKeys.length === 3 &&
      this.comboKeys[0] === vk_a &&
      this.comboKeys[1] === vk_s &&
      this.comboKeys[2] === vk_d) {
    this.specialMove();
    this.comboKeys = [];
  }
}
```

## Common Issues

### Input Not Working

**Checklist**:

- ✅ Using correct key constants? ( `vk_space` , not `'space'` )
- ✅ Browser tab has focus?
- ✅ Input check is in `step()` method?
- ✅ Check console for JavaScript errors?

### Keys Stick

**Problem**: Key stays "pressed" after releasing

**Cause**: Browser loses focus while key is down

**Solution**: Reset input state on window blur

```
window.addEventListener('blur', () => {
  // Reset all input when window loses focus
});
```

### Diagonal Movement Too Fast

**Problem**: Moving diagonally is faster than cardinal directions

**Cause**: Adding both X and Y speeds without normalization

**Solution**: Use `point_direction()` for normalized movement

```
let xdir = 0;
let ydir = 0;

if (keyboard_check(vk_right)) xdir += 1;
if (keyboard_check(vk_left)) xdir -= 1;
if (keyboard_check(vk_down)) ydir += 1;
if (keyboard_check(vk_up)) ydir -= 1;

if (xdir !== 0 || ydir !== 0) {
  const dir = point_direction(0, 0, xdir, ydir);
  this.x += lengthdir_x(4, dir);  // Normalized to 4 pixels
  this.y += lengthdir_y(4, dir);
}
```

---

## Next Steps

- **04-gameobjects.md** - Using input in GameObjects
- **40-common-patterns.md** - More input patterns
- **24-api-input.md** - Complete input API

---

← Back to Index