Input API

Complete keyboard and mouse reference

Overview

Input functions check keyboard keys and mouse buttons. All input functions are global and can be called from any `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
  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;

  // Hold to charge
  if (keyboard_check(vk_space)) {
    this.chargeAmount++;
  }
}
```

`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 {
  // Single action on press
  if (keyboard_check_pressed(vk_space)) {
    if (this.onGround) {
      this.vspeed = -10;
    }
  }

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

  // Menu navigation
  if (keyboard_check_pressed(vk_down)) {
    this.menuIndex++;
  }
  if (keyboard_check_pressed(vk_up)) {
    this.menuIndex--;
  }
}
```

`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
  if (keyboard_check(vk_space)) {
    this.jumpCharge++;
  }
  if (keyboard_check_released(vk_space)) {
    this.vspeed = -this.jumpCharge;
    this.jumpCharge = 0;
  }

  // Stop running animation
  if (keyboard_check_released(vk_shift)) {
    this.sprite_index = 'spr_player_walk';
  }
}
```

Keyboard Constants

Arrow Keys

- `vk_left` = 37
- `vk_right` = 39
- `vk_up` = 38
- `vk_down` = 40

Example:

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

Letter Keys (A-Z)

- `vk_a` through `vk_z` = 65-90

Example:

```
step(): void {
  // WASD movement
  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 (0-9)

- `vk_0` through `vk_9` = 48-57

Example:

```
step(): void {
  // Quick weapon select
  if (keyboard_check_pressed(vk_1)) this.weapon = 0;
  if (keyboard_check_pressed(vk_2)) this.weapon = 1;
  if (keyboard_check_pressed(vk_3)) this.weapon = 2;
}
```

Special Keys

- `vk_space` = 32
- `vk_enter` = 13
- `vk_escape` = 27
- `vk_shift` = 16
- `vk_control` = 17
- `vk_alt` = 18
- `vk_backspace` = 8
- `vk_tab` = 9

Example:

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

  // Sprint
  if (keyboard_check(vk_shift)) {
    this.speed = 8;
  } else {
    this.speed = 4;
  }

  // Pause
  if (keyboard_check_pressed(vk_escape)) {
    this.paused = !this.paused;
  }
}
```

Function Keys

- `vk_f1` through `vk_f12` = 112-123

Example:

```
step(): void {
  if (keyboard_check_pressed(vk_f1)) this.showHelp = true;
  if (keyboard_check_pressed(vk_f3)) this.debugMode = !this.debugMode;
  if (keyboard_check_pressed(vk_f5)) this.quickSave();
}
```

Numpad Keys

- `vk_numpad0` through `vk_numpad9` = 96-105
- `vk_multiply` = 106
- `vk_add` = 107

- `vk_subtract` = 109
- `vk_decimal` = 110
- `vk_divide` = 111

Other Keys

- `vk_home` = 36
- `vk_end` = 35
- `vk_pageup` = 33
- `vk_pagedown` = 34
- `vk_delete` = 46
- `vk_insert` = 45

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 {
  // Continuous shooting 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;
    }
  }

  // Drag object
  if (mouse_check_button(mb_left) && this.grabbed) {
    this.x = mouse_x;
    this.y = mouse_y;
  }
}
```

`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 menu
  if (mouse_check_button_pressed(mb_right)) {
    this.showContextMenu = true;
    this.menuX = mouse_x;
    this.menuY = mouse_y;
  }

  // Pick up object
  if (mouse_check_button_pressed(mb_left)) {
    const item = instance_position(mouse_x, mouse_y, 'obj_item');
    if (item) {
      this.inventory.push(item);
      instance_destroy.call(item);
    }
  }
}
```

`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;
  }

  // Drop object
  if (mouse_check_button_released(mb_left) && this.grabbed) {
    this.grabbed = false;
  }
}
```

Mouse Constants

- `mb_left` = 0 - Left mouse button
- `mb_right` = 2 - Right mouse button
- `mb_middle` = 1 - Middle mouse button (scroll wheel click)

Example:

```
step(): void {
  if (mouse_check_button_pressed(mb_left)) {
    // Primary action
  }
  if (mouse_check_button_pressed(mb_right)) {
    // Secondary action
  }
  if (mouse_check_button_pressed(mb_middle)) {
    // Special action
  }
}
```

Mouse Position

Global Variables

- `mouse_x` (number) - Mouse X position in room coordinates
- `mouse_y` (number) - Mouse Y position in room coordinates

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);

  // Check distance to mouse
  const dist = point_distance(this.x, this.y, mouse_x, mouse_y);
  if (dist < 50) {
    // Mouse is close
  }
}

draw(): void {
  // Draw line to mouse
  draw_set_color('#FF0000');
  draw_line(this.x, this.y, mouse_x, mouse_y);
}
```

Note: Coordinates account for camera/view position automatically.

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;
    this.image_xscale = -1; // Face left
  }
  if (keyboard_check(vk_d)) {
    this.x += speed;
    this.image_xscale = 1; // Face right
  }
}
```

8-Direction Movement (Normalized)

```
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);
  }
}
```

Mouse Aiming

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

  // Shoot at mouse
  if (mouse_check_button_pressed(mb_left)) {
    const bullet = await instance_create(this.x, this.y, 'obj_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 and Drop

```
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;

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

  // Shoot if 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

```
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;
  }
}
```

Key Combos

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

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

  // 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 browser console for errors

Keys Stick

Problem: Key stays "pressed" after releasing

Cause: Browser loses focus while key is down

Solution: Reset input on window blur

```
window.addEventListener('blur', () => {
  // Reset input state
});
```

Diagonal Movement Too Fast

Problem: Moving diagonally is faster than cardinal directions

Solution: Use normalized direction

```
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

- **08-input.md** - Input handling guide
- **25-api-motion.md** - Motion functions
- **40-common-patterns.md** - Input patterns

← Back to Index