JGame.js Documentation

INTRODUCTION GETTING STARTED BEGINNERS
CLASSES AND METHODS EXAMPLES FAQ INTERMEDIATE
ADVANCED GAME EXAMPLES

1. Introduction

Welcome to the JGame.js documentation! This guide will help you understand how to use the JGame.js game engine to create your own 2D games.

2. Getting Started

To get started with JGame.js, include the following scripts in your HTML file:

```
<script src="/jgme/jgame.js"></script>
<script src="/jgme/jgame_more.js"></script>
```

3. Classes and Methods

Display Class

The `Display` class is responsible for creating and managing the game canvas.

Methods:

- `start(width, height)`: Initializes the canvas with the specified width and height.

- `addEventListeners()`: Adds event listeners for keyboard and mouse input.
- `clear()`: Clears the canvas.
- `borderStyle(borderStyle)`: Sets the canvas border style.
- `stop()`: Stops the game loop.
- `borderSize(borderSize)`: Sets the canvas border size.
- `backgroundColor(color)`: Sets the canvas background color.
- `borderColor(color)`: Sets the canvas border color.
- `fontColor(color)`: Sets the canvas font color.
- `scale(width, height)`: Scales the canvas dimensions.
- `add(component)`: Adds a component to the display.
- `update()`: Updates the display and components.

Component Class

The `Component` class represents game objects.

Methods:

- `update(ctx)`: Updates the component's appearance.
- `move()`: Moves the component based on its speed and physics.
- `hitBottom()`: Checks if the component hits the bottom of the canvas.
- `stopMove()`: Stops the component's movement.
- `clicked()`: Checks if the component is clicked.
- `crashWith(otherobj)`: Checks if the component crashes with another object.

Sound Class

The `Sound` class handles audio playback.

Methods:

- `play()`: Plays the sound.
- `stop()`: Stops the sound.

Move Class

The `Move` class contains various movement and transformation methods.

Methods:

- `backward(id, steps)`: Moves the component backward.
- `teleport(id, x, y)`: Teleports the component to the specified coordinates.
- `setX(id, x)`: Sets the component's x-coordinate.
- `setY(id, y)`: Sets the component's y-coordinate.
- `stamp(id)`: Creates a stamped copy of the component.
- `circle(id, speed)`: Moves the component in a circular path.
- `dot(id)`: Draws a dot at the component's position.
- `clearStamp(id)`: Clears the stamped component.
- `turnLeft(id, steps)`: Rotates the component to the left.
- `turnRight(id, steps)`: Rotates the component to the right.
- `bound(id)`: Keeps the component within the canvas bounds.
- `hitObject(id, otherid)`: Checks if the component hits another object.
- `glideX(id, t, x)`: Glides the component horizontally.
- `glideY(id, t, y)`: Glides the component vertically.
- `glideTo(id, t, x, y)`: Glides the component to the specified coordinates.
- `project(id, initialVelocity, angle, gravity)`: Projects the component into the air.
- `pointTo(id, targetX, targetY)`: Points the component towards the specified coordinates.

State Class

The `State` class contains methods for retrieving the state of components.

Methods:

- `distance(id, otherid)`: Calculates the distance between two components.
- `rect(id)`: Returns the component's rectangle (x, y, width, height).
- `physics(id)`: Returns whether the component has physics enabled.
- `changeAngle(id)`: Returns whether the component's angle changes.
- `angle(id)`: Returns the component's angle.
- `pos(id)`: Returns the component's position as a string.

4. Examples

Here are some examples to help you get started:

Creating a Display:

```
const display = new Display();
display.start(800, 600);
display.borderStyle("solid");
display.borderColor("black");
```

Adding a Component:

```
const component = new Component(50, 50, "red", 100, 100, "rectangle");
display.add(component);
```

Moving a Component:

```
move.glideTo(component, 3, 400, 300);
```

Playing a Sound:

```
const sound = new Sound("path/to/sound.mp3");
sound.play();
```

Checking for Collision:

```
const component1 = new Component(50, 50, "blue", 200, 200, "rectangle");
const component2 = new Component(50, 50, "green", 250, 250, "rectangle");
display.add(component1);
display.add(component2);

if (move.checkCollision(component1, component2)) {
    console.log("Collision detected!");
}
```

Animating a Sprite:

```
const spriteImage = new Image();
spriteImage.src = "path/to/sprite.png";
const sprite = new Sprite(spriteImage, 64, 64, 4, 10);

function update() {
    sprite.update();
    sprite.draw(display.context, 100, 100);
}

display.add({ update });
```

3. Beginners

If you're new to game development, this section will guide you through the basics of using JGame.js.

Setting Up Your Project

First, create a new HTML file and include the JGame.js scripts:

```
<!DOCTYPE html>
        <html lang="en">
        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.
            <title>My First Game</title>
            <script src="/jgme/jgame.js"></script>
            <script src="/jgme/jgame_more.js"></script>
        </head>
        <body>
            <script>
                // Your game code will go here
            </script>
        </body>
        </html>
```

Creating a Display

Next, create a display for your game:

```
const display = new Display();
        display.start(800, 600);
        display.borderStyle("solid");
        display.borderColor("black");
```

Adding a Component

Now, add a component to the display:

```
const component = new Component(50, 50, "red", 100, 100, "rectangle");
        display.add(component);
```

Moving a Component

To move the component, use the following code:

```
move.glideTo(component, 3, 400, 300);
```

Running Your Game

Finally, create an update function to run your game:

```
function update() {
            // Update your game logic here
        }

        display.update = update;
```

Example 1: Changing Background Color

Change the background color of the canvas:

```
display.backgroundColor("lightblue");
```

Example 2: Adding Multiple Components

Add multiple components to the display:

```
const component1 = new Component(50, 50, "blue", 200, 200, "rectangle");
        const component2 = new Component(50, 50, "green", 300, 300, "rectangle"
```

```
        display.add(component1);
        display.add(component2);
```

Example 3: Moving Components Independently

Move multiple components independently:

```
move.glideTo(component1, 3, 400, 300);
        move.glideTo(component2, 3, 100, 100);
```

Example 4: Rotating a Component

Rotate a component to a specific angle:

```
move.turnRight(component, 45);
```

Example 5: Adding a Click Event

Add a click event to a component:

```
component.clicked = function() {
            alert("Component clicked!");
        };
```

Example 6: Playing a Sound

Play a sound when a component is clicked:

```
const sound = new Sound("path/to/sound.mp3");
        component.clicked = function() {
            sound.play();
        };
```

Example 7: Checking for Collisions

Check for collisions between two components:

```
if (move.checkCollision(component1, component2)) {
            console.log("Collision detected!");
        }
```

4. Intermediate

This section is for those who have a basic understanding of JGame.js and want to explore more advanced features and techniques.

Example 1: Creating a Custom Component

Create a custom component with additional properties and methods:

```
class CustomComponent extends Component {
            constructor(width, height, color, x, y, type) {
                super(width, height, color, x, y, type);
                this.customProperty = "value";
            }

            customMethod() {
                console.log("Custom method called!");
            }

            update(ctx) {
                super.update(ctx);
                // Add custom rendering logic here
            }
        }

        const customComponent = new CustomComponent(50, 50, "purple", 150, 150,
        display.add(customComponent);
        customComponent.customMethod();
```

Example 2: Implementing Gravity

Add gravity to a component to simulate falling:

```
component.gravity = 0.1;
        component.gravitySpeed = 0;
```

```
        function update() {
            component.gravitySpeed += component.gravity;
            component.y += component.gravitySpeed;
            component.hitBottom();
        }

        display.update = update;
```

Example 3: Creating a Simple Animation

Animate a component by changing its properties over time:

```
let angle = 0;

        function update() {
            angle += 1;
            component.x = 100 + 50 * Math.cos(angle * Math.PI / 180);
            component.y = 100 + 50 * Math.sin(angle * Math.PI / 180);
        }

        display.update = update;
```

Example 4: Handling Keyboard Input

Move a component using keyboard input:

```
window.addEventListener("keydown", function(e) {
            switch (e.key) {
                case "ArrowUp":
                    component.y -= 5;
                    break;
                case "ArrowDown":
                    component.y += 5;
                    break;
                case "ArrowLeft":
                    component.x -= 5;
                    break;
                case "ArrowRight":
                    component.x += 5;
                    break;
            }
        });
```

Example 5: Creating a Particle System

Create a simple particle system for effects like explosions:

```
class Particle extends Component {
            constructor(x, y) {
                super(5, 5, "yellow", x, y, "rectangle");
                this.speedX = Math.random() * 2 - 1;
                this.speedY = Math.random() * 2 - 1;
                this.life = 100;
            }

            update(ctx) {
                this.x += this.speedX;
                this.y += this.speedY;
                this.life -= 1;
                if (this.life <= 0) {
                    display.comm.splice(display.comm.indexOf(this), 1);
                }
                super.update(ctx);
            }
        }

        function createExplosion(x, y) {
            for (let i = 0; i < 50; i++) {
                const particle = new Particle(x, y);
                display.add(particle);
            }
        }

        createExplosion(200, 200);
```

Example 6: Implementing Collision Detection

Detect and respond to collisions between components:

```
function update() {
            if (move.checkCollision(component1, component2)) {
                component1.color = "red";
                component2.color = "red";
            } else {
                component1.color = "blue";
                component2.color = "green";
            }
        }
```

```
        display.update = update;
```

Example 7: Creating a Simple Game Loop

Implement a basic game loop to update and render your game:

```
function gameLoop() {
            display.clear();
            update();
            display.comm.forEach(component => component.update(display.context)
            requestAnimationFrame(gameLoop);
        }

        gameLoop();
```

5. Advanced

This section is for those who have a solid understanding of JGame.js and want to explore more advanced features and techniques.

Example 1: Implementing a Physics Engine

Integrate a basic physics engine for realistic movement and collisions:

```
class PhysicsComponent extends Component {
            constructor(width, height, color, x, y, type) {
                super(width, height, color, x, y, type);
                this.velocityX = 0;
                this.velocityY = 0;
                this.accelerationX = 0;
                this.accelerationY = 0;
            }

            update(ctx) {
                this.velocityX += this.accelerationX;
                this.velocityY += this.accelerationY;
                this.x += this.velocityX;
                this.y += this.velocityY;
```

```
                super.update(ctx);
            }
        }

        const physicsComponent = new PhysicsComponent(50, 50, "orange", 100, 10
        physicsComponent.accelerationY = 0.1; // Gravity
        display.add(physicsComponent);
```

Example 2: Creating a Tile-Based Map

Implement a tile-based map for your game:

```
const tileSize = 32;
        const map = [
            [0, 1, 0, 0, 0],
            [0, 1, 0, 1, 0],
            [0, 0, 0, 1, 0],
            [1, 1, 0, 1, 0],
            [0, 0, 0, 0, 0]
        ];

        function drawMap() {
            for (let row = 0; row < map.length; row++) {
                for (let col = 0; col < map[row].length; col++) {
                    if (map[row][col] === 1) {
                        const tile = new Component(tileSize, tileSize, "gray",
                        display.add(tile);
                    }
                }
            }
        }

        drawMap();
```

Example 3: Implementing Pathfinding

Add pathfinding to navigate complex environments:

```
function findPath(start, goal) {
            const openSet = [start];
            const cameFrom = new Map();
            const gScore = new Map();
            const fScore = new Map();
```

```
    gScore.set(start, 0);
    fScore.set(start, heuristic(start, goal));

    while (openSet.length > 0) {
        let current = openSet.reduce((a, b) => fScore.get(a) < fScore.g

        if (current === goal) {
            return reconstructPath(cameFrom, current);
        }

        openSet.splice(openSet.indexOf(current), 1);

        for (let neighbor of getNeighbors(current)) {
            let tentativeGScore = gScore.get(current) + distance(curren

            if (tentativeGScore < (gScore.get(neighbor) || Infinity)) {
                cameFrom.set(neighbor, current);
                gScore.set(neighbor, tentativeGScore);
                fScore.set(neighbor, gScore.get(neighbor) + heuristic(n

                if (!openSet.includes(neighbor)) {
                    openSet.push(neighbor);
                }
            }
        }
    }

    return null;
}

function heuristic(a, b) {
    return Math.abs(a.x - b.x) + Math.abs(a.y - b.y);
}

function reconstructPath(cameFrom, current) {
    const path = [current];
    while (cameFrom.has(current)) {
        current = cameFrom.get(current);
        path.unshift(current);
    }
    return path;
}

function getNeighbors(node) {
    // Implement this function to return the neighbors of a node
}
```

```
        function distance(a, b) {
            return Math.sqrt(Math.pow(a.x - b.x, 2) + Math.pow(a.y - b.y, 2));
        }
```

Example 4: Creating a Game Menu

Implement a simple game menu with buttons:

```
class Button extends Component {
            constructor(width, height, color, x, y, text) {
                super(width, height, color, x, y, "rectangle");
                this.text = text;
            }

            update(ctx) {
                super.update(ctx);
                ctx.fillStyle = "black";
                ctx.font = "20px Arial";
                ctx.fillText(this.text, this.x + 10, this.y + 30);
            }

            clicked() {
                alert(this.text + " button clicked!");
            }
        }

        const startButton = new Button(100, 50, "lightgreen", 150, 200, "Start"
        const optionsButton = new Button(100, 50, "lightblue", 150, 300, "Optio
        display.add(startButton);
        display.add(optionsButton);

        window.addEventListener("click", function(e) {
            if (startButton.clicked()) {
                startButton.clicked();
            } else if (optionsButton.clicked()) {
                optionsButton.clicked();
            }
        });
```

Example 5: Implementing a Health Bar

Add a health bar to a component:

```
class HealthComponent extends Component {
            constructor(width, height, color, x, y, type) {
                super(width, height, color, x, y, type);
                this.health = 100;
            }

            update(ctx) {
                super.update(ctx);
                ctx.fillStyle = "red";
                ctx.fillRect(this.x, this.y - 10, this.width * (this.health / 1
            }
        }

        const healthComponent = new HealthComponent(50, 50, "blue", 200, 200, "
        display.add(healthComponent);

        function update() {
            healthComponent.health -= 0.1; // Decrease health over time
        }

        display.update = update;
```

Example 6: Creating a Multiplayer Game

Implement basic multiplayer functionality using WebSockets:

```
const socket = new WebSocket("ws://yourserver.com");

        socket.onopen = function() {
            console.log("Connected to server");
        };

        socket.onmessage = function(event) {
            const data = JSON.parse(event.data);
            // Update game state based on received data
        };

        function sendUpdate() {
            const data = {
                x: component.x,
                y: component.y
            };
            socket.send(JSON.stringify(data));
        }
```

```
        function update() {
            // Update game logic
            sendUpdate();
        }

        display.update = update;
```

Example 7: Creating a Level Editor

Build a simple level editor to design game levels:

```
class LevelEditor {
            constructor() {
                this.tiles = [];
            }

            addTile(x, y) {
                const tile = new Component(tileSize, tileSize, "gray", x, y, "r
                this.tiles.push(tile);
                display.add(tile);
            }

            saveLevel() {
                const levelData = this.tiles.map(tile => ({ x: tile.x, y: tile.
                localStorage.setItem("level", JSON.stringify(levelData));
            }

            loadLevel() {
                const levelData = JSON.parse(localStorage.getItem("level"));
                if (levelData) {
                    levelData.forEach(data => this.addTile(data.x, data.y));
                }
            }
        }

        const levelEditor = new LevelEditor();
        levelEditor.loadLevel();

        window.addEventListener("click", function(e) {
            const x = Math.floor(e.pageX / tileSize) * tileSize;
            const y = Math.floor(e.pageY / tileSize) * tileSize;
            levelEditor.addTile(x, y);
        });

        document.getElementById("saveButton").addEventListener("click", functio
```

```
            levelEditor.saveLevel();
        });
```

6. Game Examples

This section provides examples of classic games implemented using JGame.js.

Example 1: Dino Game

Recreate the classic Dino Game where the player controls a dinosaur that jumps over obstacles:

```
class Dino extends Component {
            constructor(width, height, color, x, y) {
                super(width, height, color, x, y, "rectangle");
                this.gravity = 0.5;
                this.gravitySpeed = 0;
                this.jumpPower = -10;
            }

            jump() {
                this.gravitySpeed = this.jumpPower;
            }

            update(ctx) {
                this.gravitySpeed += this.gravity;
                this.y += this.gravitySpeed;
                if (this.y > display.canvas.height - this.height) {
                    this.y = display.canvas.height - this.height;
                    this.gravitySpeed = 0;
                }
                super.update(ctx);
            }
        }

        class Obstacle extends Component {
            constructor(width, height, color, x, y) {
                super(width, height, color, x, y, "rectangle");
                this.speedX = -5;
```

```
            }

            update(ctx) {
                this.x += this.speedX;
                if (this.x < 0) {
                    this.x = display.canvas.width;
                }
                super.update(ctx);
            }
        }

        const dino = new Dino(50, 50, "green", 50, display.canvas.height - 50);
        const obstacles = [new Obstacle(20, 50, "red", 300, display.canvas.heig

        display.add(dino);
        obstacles.forEach(obstacle => display.add(obstacle));

        window.addEventListener("keydown", function(e) {
            if (e.key === " ") {
                dino.jump();
            }
        });

        function update() {
            obstacles.forEach(obstacle => {
                if (move.checkCollision(dino, obstacle)) {
                    alert("Game Over!");
                    window.location.reload();
                }
            });
        }

        display.update = update;
```

Example 2: Snake Game

Implement the classic Snake Game where the player controls a snake that grows longer as it eats food:

```
class Snake {
            constructor() {
                this.body = [{ x: 10, y: 10 }];
                this.direction = "right";
                this.food = this.generateFood();
            }
```

```
generateFood() {
    return { x: Math.floor(Math.random() * 20), y: Math.floor(Math.
}

move() {
    const head = { ...this.body[0] };
    switch (this.direction) {
        case "right":
            head.x += 1;
            break;
        case "left":
            head.x -= 1;
            break;
        case "up":
            head.y -= 1;
            break;
        case "down":
            head.y += 1;
            break;
    }
    this.body.unshift(head);
    if (head.x === this.food.x && head.y === this.food.y) {
        this.food = this.generateFood();
    } else {
        this.body.pop();
    }
}

changeDirection(newDirection) {
    this.direction = newDirection;
}

checkCollision() {
    const head = this.body[0];
    for (let i = 1; i < this.body.length; i++) {
        if (head.x === this.body[i].x && head.y === this.body[i].y)
            return true;
        }
    }
    return false;
}

draw(ctx) {
    ctx.fillStyle = "green";
    this.body.forEach(segment => {
        ctx.fillRect(segment.x * 20, segment.y * 20, 20, 20);
```

```
        });
        ctx.fillStyle = "red";
        ctx.fillRect(this.food.x * 20, this.food.y * 20, 20, 20);
    }
}

const snake = new Snake();

window.addEventListener("keydown", function(e) {
    switch (e.key) {
        case "ArrowUp":
            snake.changeDirection("up");
            break;
        case "ArrowDown":
            snake.changeDirection("down");
            break;
        case "ArrowLeft":
            snake.changeDirection("left");
            break;
        case "ArrowRight":
            snake.changeDirection("right");
            break;
    }
});

function update() {
    snake.move();
    if (snake.checkCollision()) {
        alert("Game Over!");
        window.location.reload();
    }
}

function draw() {
    display.clear();
    snake.draw(display.context);
}

function gameLoop() {
    update();
    draw();
    requestAnimationFrame(gameLoop);
}

gameLoop();
```

Example 3: Tic-Tac-Toe

Create a simple Tic-Tac-Toe game where two players take turns marking X and O on a 3x3 grid:

```
class TicTacToe {
            constructor() {
                this.board = [
                    ["", "", ""],
                    ["", "", ""],
                    ["", "", ""]
                ];
                this.currentPlayer = "X";
            }

            makeMove(x, y) {
                if (this.board[y][x] === "") {
                    this.board[y][x] = this.currentPlayer;
                    this.currentPlayer = this.currentPlayer === "X" ? "O" : "X"
                }
            }

            checkWinner() {
                const winningCombinations = [
                    [[0, 0], [0, 1], [0, 2]],
                    [[1, 0], [1, 1], [1, 2]],
                    [[2, 0], [2, 1], [2, 2]],
                    [[0, 0], [1, 0], [2, 0]],
                    [[0, 1], [1, 1], [2, 1]],
                    [[0, 2], [1, 2], [2, 2]],
                    [[0, 0], [1, 1], [2, 2]],
                    [[0, 2], [1, 1], [2, 0]]
                ];

                for (let combination of winningCombinations) {
                    const [a, b, c] = combination;
                    if (this.board[a[1]][a[0]] && this.board[a[1]][a[0]] === th
                        return this.board[a[1]][a[0]];
                    }
                }

                return null;
            }

            draw(ctx) {
                ctx.clearRect(0, 0, display.canvas.width, display.canvas.height
```

```
        ctx.strokeStyle = "black";
        ctx.lineWidth = 2;

        for (let i = 1; i < 3; i++) {
            ctx.beginPath();
            ctx.moveTo(i * 100, 0);
            ctx.lineTo(i * 100, 300);
            ctx.stroke();
            ctx.beginPath();
            ctx.moveTo(0, i * 100);
            ctx.lineTo(300, i * 100);
            ctx.stroke();
        }

        for (let y = 0; y < 3; y++) {
            for (let x = 0; x < 3; x++) {
                if (this.board[y][x]) {
                    ctx.font = "80px Arial";
                    ctx.fillText(this.board[y][x], x * 100 + 20, y * 10
                }
            }
        }
    }
}

const ticTacToe = new TicTacToe();

display.canvas.addEventListener("click", function(e) {
    const x = Math.floor(e.offsetX / 100);
    const y = Math.floor(e.offsetY / 100);
    ticTacToe.makeMove(x, y);
    const winner = ticTacToe.checkWinner();
    if (winner) {
        alert(winner + " wins!");
        window.location.reload();
    }
});

function gameLoop() {
    ticTacToe.draw(display.context);
    requestAnimationFrame(gameLoop);
}

gameLoop();
```

5. FAQ

Q: How do I add a new component to the display?

A: Use the `display.add(component)` method to add a new component.

Q: How do I play a sound?

A: Create a new `Sound` object and call the `play()` method.

Q: How do I make a component move in a circular path?

A: Use the `move.circle(id, speed)` method to move the component in a circular path.

Q: How do I check for collisions between components?

A: Use the `move.checkCollision(id1, id2)` method to check for collisions between two components.

Q: How do I animate a sprite?

A: Create a `Sprite` object and use the `update()` and `draw(ctx, x, y)` methods to animate it.