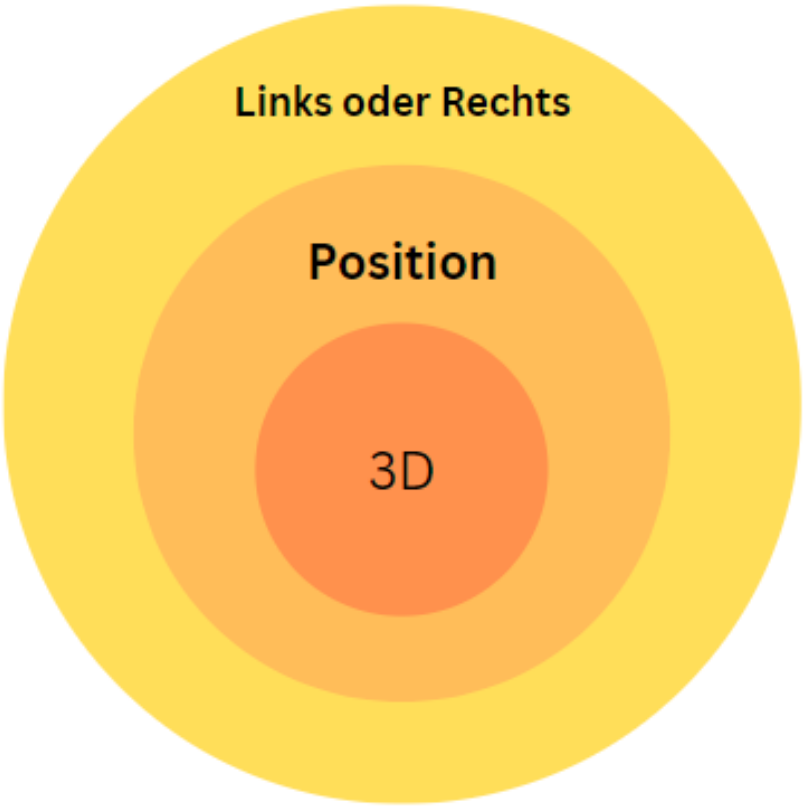
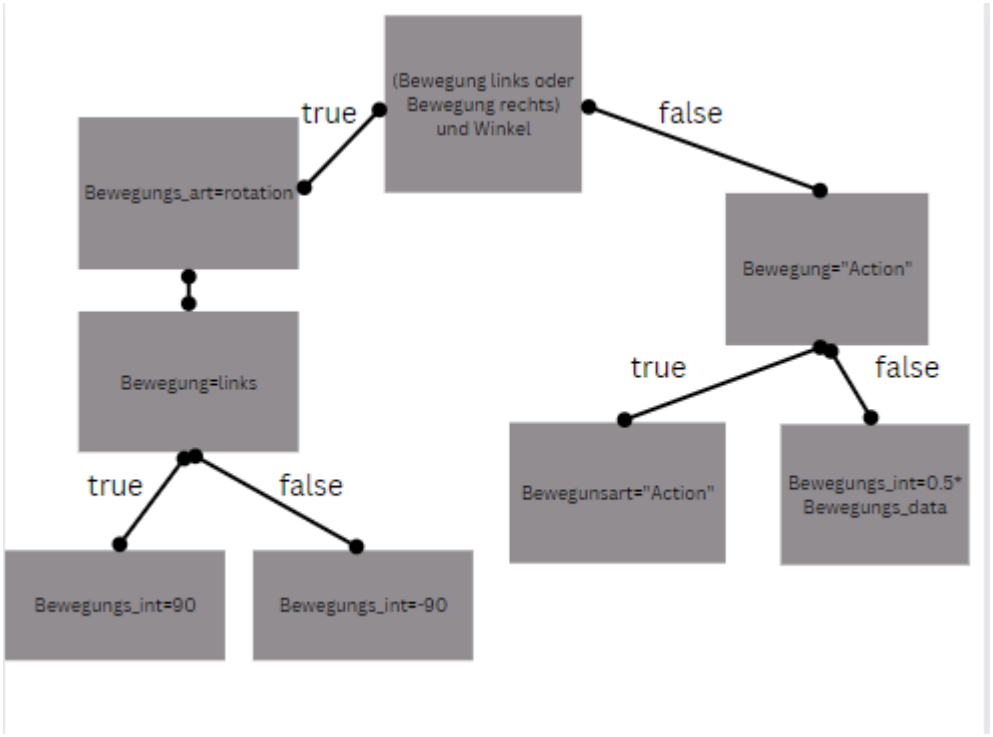
Überlegungen zu drone_actions

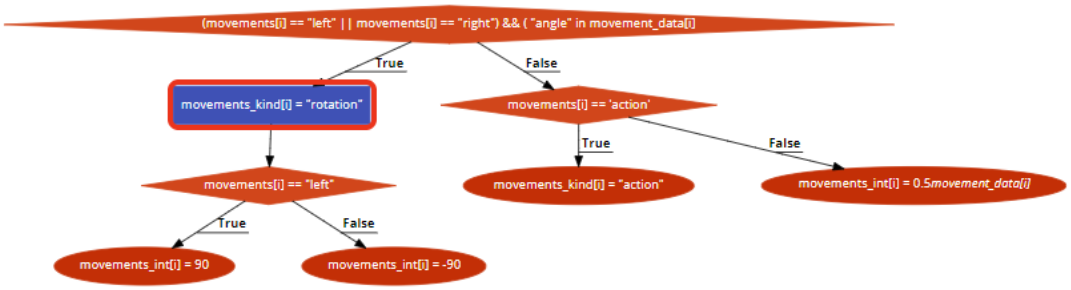
1. Grafiken:

Allgemeiner Aufbau:

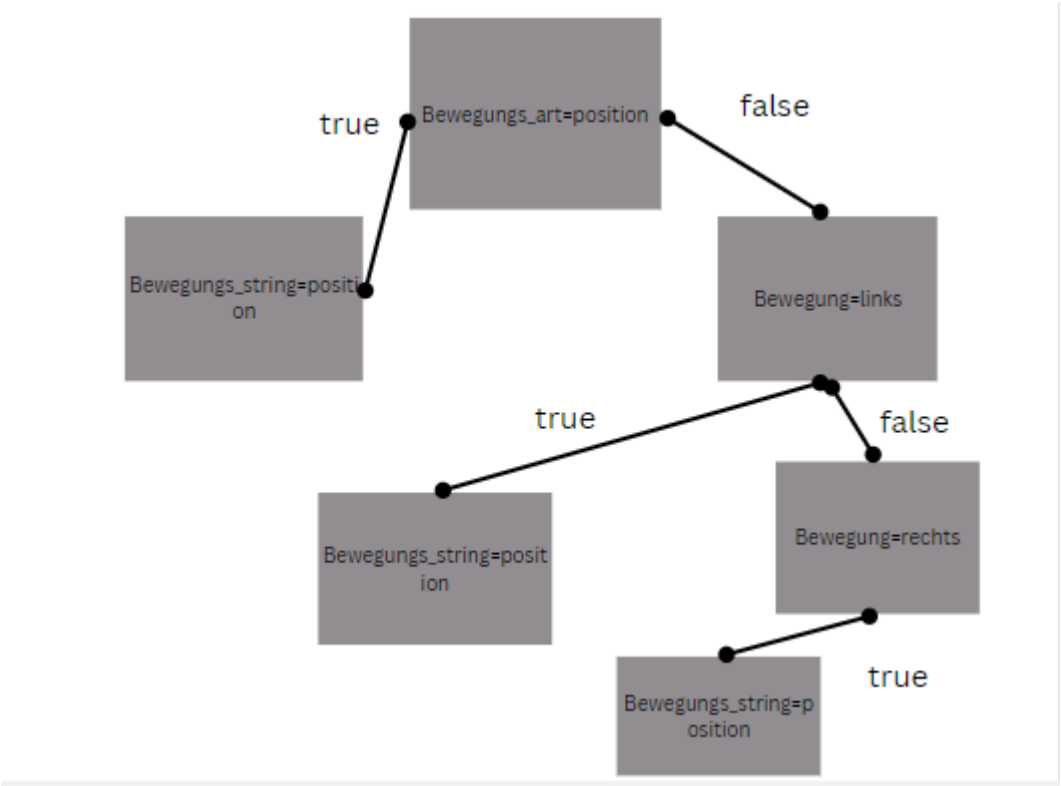


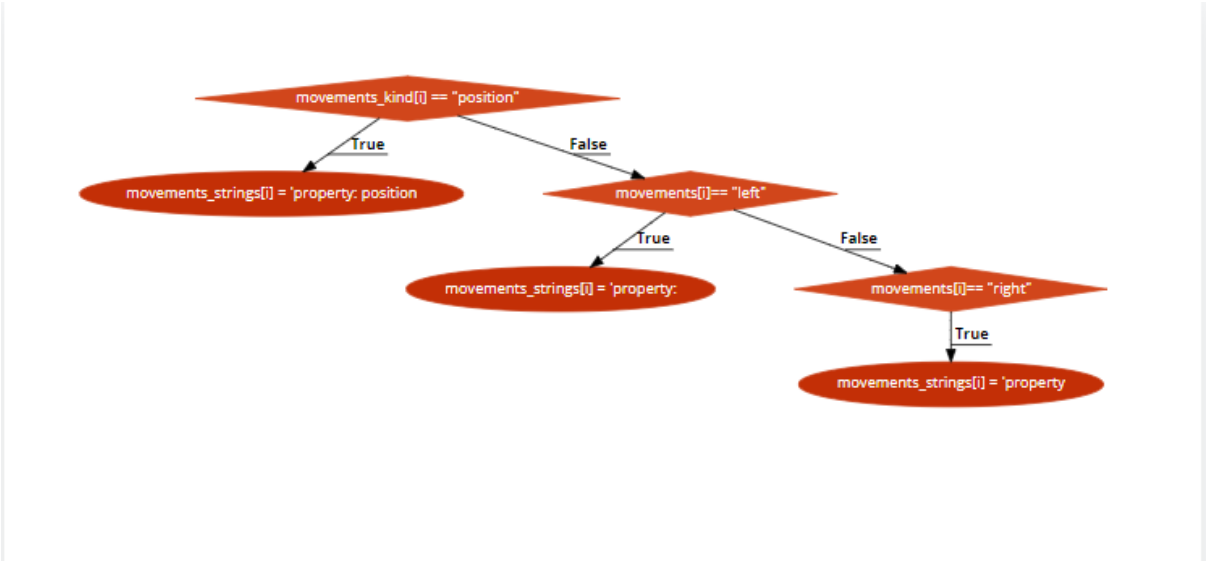
Links oder Rechts:



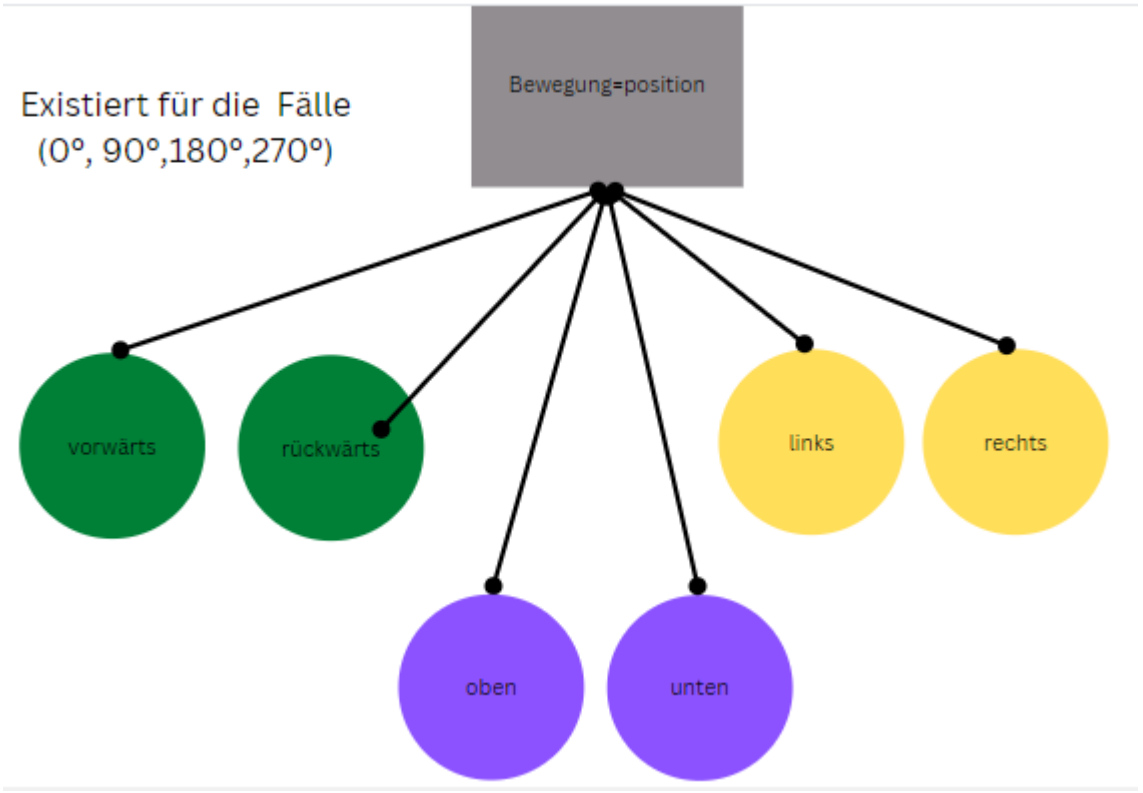


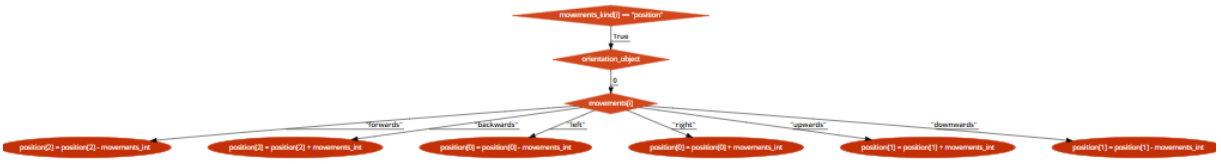
Position:


Bewegungs_art=position
true
false
Bewegungs_string=positi
on
Bewegung=links
true
false
Bewegungs_string=posit
ion
Bewegung=rechts
Bewegungs_string=p
osition
true

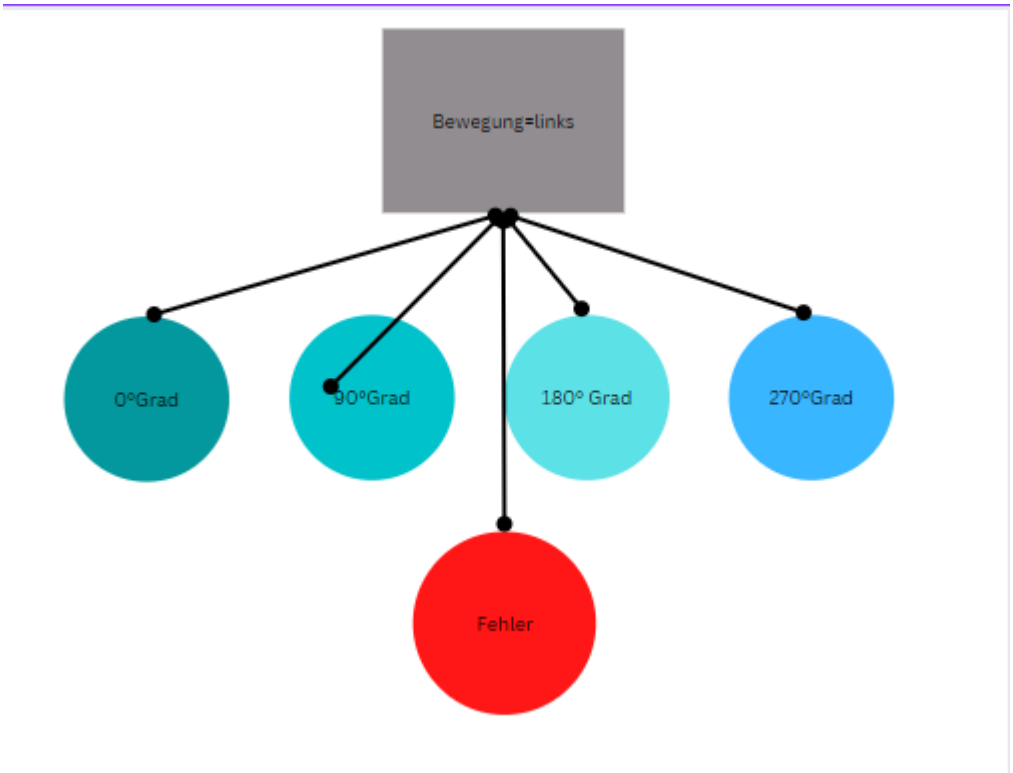

movements_kind[i] == "position"
True
False
movements_strings[i] = 'property: position
movements[i]== "left"
True
False
movements_strings[i] = 'property:
movements[i]== "right"
True
movements_strings[i] = 'property

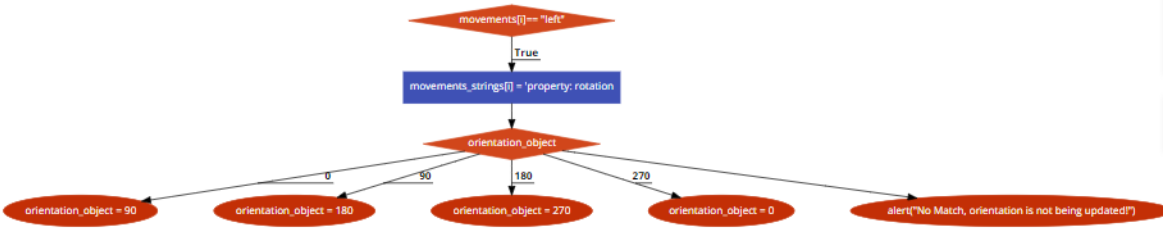
3D:


Existiert für die Fälle
(0°, 90°,180°,270°)
Bewegung=position
vorwärts
rückwärts
links
rechts
oben
unten



Einrichten Position:



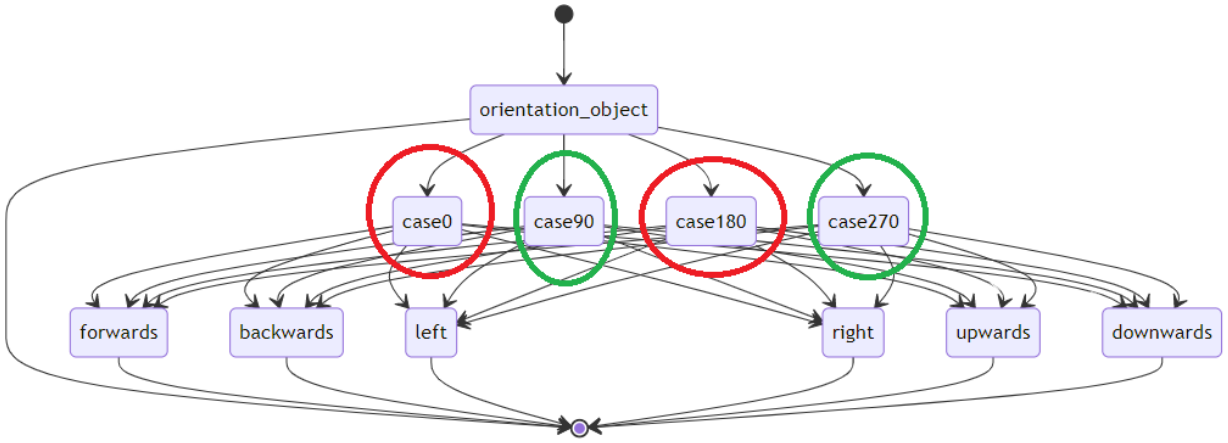


Auffälligkeiten bei den Arrayplätzen:

```
Case 0:
forward, backward= position[2]
left,right=position[0]
upwards,downwards=position[1]

Case 90:
forward, backward= position[0]
left,right=position[2]
upwards,downwards=position[1]

Case 180:
forward, backward= position[2]
left,right=position[0]
upwards,downwards=position[1]

Case 270:
forward, backward= position[0]
left,right=position[2]
upwards,downwards=position[1]
```

sehr elegant gelöst !

Als Zustandsgraph:



2. Anmerkungen und Code

Variablen, deren Gebrauch und Tauglichkeit

Legende:
rot = wichtig
gelb= unwichtig

```
var movements_length = movement_data.length;
var movements = new Array( movements_length );
var movements_kind = new Array( movements_length );
var movements_int = new Array( movements_length );
var movements_strings = new Array( movements_length );
var positions = new Array( movements_length );
var orientations = new Array( movements_length );
var lines = new Array( movements_length );

for(var i = 0, length = movements_length; i < length; i++) [in Z133]
for (var i = 0; i < movements_length; i++)  [in Z176]

movements[i] = movement_data[i]["direction"] [Z138]
if( (movements[i] == "left" || movements[i] == "right") && ( "angle" in movement_data[i] ) ) {
        movements_kind[i] = "rotation";

        if(movements[i] == "left") {
            movements_int[i] = 90;
        } else {
            movements_int[i] = -90;
        }


var position = [0,0.1,0]; [Z163]
var position_old = [0,0.1,0];
position_old[0] = position[0];[Z310]


for(var i = 0, length = indizes_action.length; i < length; i++){  [Z528]
index_position = indizes_action[i];
```

Überlegungen zu den Variablen, deren Gebrauch und Tauglichkeit

- movements_length redundant, kann weg.
- Loop durch JSON array um Bewegungsablauf zu erhalten (movement_data.length)
- Prüfen ob sich die Position oder Orientierung geändert hat (Richtung [position] oder Rotation [orientations])
- Orientation: 0 - "forward", 90 - "left", 180 - "back", 270 - "right"
- entsprechende Bewegung wird in [movements_kind] gespeichert
- ⇒ movements_kind, positions, orientations dürfen nicht ersetzt werden
- movements kann ersetzt werden durch den Arrayaufruf movement_data[i]["direction"]
- movements_kind und movements_int sind abhängig von movements, dürfen nicht ersetzt werden
- position und position_old haben die selben Ursprungskoordinaten.
- am Ende einer Iteration werden position und position_old einfach gleichgesetzt.
- position_old ist somit redundant und kann entfernt werden.
- length überflüssig
- index_position wird mit indizes_action gleichgesetzt und ansonsten passiert nichts
- index_position somit überflüssig
- für mehr Klarheit wird die Variable position in position_fest umbenannt.

Funktionen, Anmerkungen und Code

Um die Wartbarkeit und Übersichtlichkeit des Codes zu erhöhen, werden die zentralen Bestandteile thematisch strukturiert und in Funktionen gepackt. Die Funktionen haben die sprechenden Namen: *move*, *left_right_or_action* und *position_and_3D*.
Alle Variablen, die im Entferntesten mit *movements* etwas zu tun haben, werden der Funktion *move* zugeordnet. So ändert sich im Quellcode *lines* zu *move.lines*. Der scope der Variable *movement_data* ist global, so muss der Funktion kein Parameter übergeben werden.

```
function move(){

        var movements_kind = new Array( movement_data.length );
        var movements_int = new Array( movement_data.length );
        var movements_strings = new Array( movement_data.length );
        var positions = new Array( movement_data.length );
        var orientations = new Array( movement_data.length );
        var indizes_action = [];
        var lines = new Array( movement_data.length );
        var duration_movement = 2000;
        var delay = 2000;
        var orientation_object = 0;
        return {
            movements_kind: kind,
            movements_int: int,
```

```
            movements_strings: strings,
            positions: positions,
            orientations: orientations,
            indizes_action: indizes_action,
            lines: lines,
            duration_movement: duration,
            delay: delay,
            orientation_object: orientation_object
        }

        }
```

Der Schleifenablauf, der bestimmt ob die Drone nach links oder nach rechts steuert oder eine Action ausführt wurde in folgende Funktion gepackt:

```
function left_right_or_action(){
        for(var i = 0; i < movement_data.length; i++){


            //eventuell movements ersetzen
            //siehe google docs dokument
            //movements[i] = movement_data[i]["direction"];

            if( (movement_data[i]["direction"] == "left" ||
movement_data[i]["direction"] == "right") && ( "angle" in
movement_data[i] ) ) {
                return move.kind[i] = "rotation";

                if(movement_data[i]["direction"] == "left") {
                    return move.int[i] = 90;
                } else {
                    return move.int[i] = -90;
                }

            } else if ( movement_data[i]["direction"] == 'action' ) {
                return [ move.kind[i] = "action", move.int[i] =
movement_data[i]["type"],move.indizes_action.push(i) ];


            } else {
                return [move.kind[i] = "position", move.int[i] =
0.5*movement_data[i]["distance"] ];
                }
```

```
        }
    }
```

Der Schleifenablauf, der die Position bestimmt und das Verhalten im 3D deklariert, wird in die Funktion *position_and_3D* gelagert. Als Funktion ist *position_and_ 3D* sehr umständlich und klobig, aber wie oben ab- und dargelegt, ist das Verhalten der Schleife ausgesprochen elegant gelöst und das Aufbrechen der Schleife würde den Code eher verkomplizieren. Die function *position_and_3D* muss in Unterfunktionen zerlegt werden, damit eine Wartbarkeit, die wenig Rückfragen benötigt, gewährleistet ist.

```
function position_and_3D(){
        for (var i = 0; i < movement_data.length; i++) {

            if( move.kind[i] == "position") {

                switch( move.orientation_object ) {

                    case 0:

                        switch(  movement_data[i]["direction"] ) {

                            case "forwards":
                                position_fest[2] = position_fest[2] -
move.int[i];
                                return position_fest[2];
                                break;

                            case "backwards":
                                position_fest[2] = position_fest[2] +
move.int[i];
                                return position_fest[2];
                                break;

                            case "left":
                                position_fest[0] = position_fest[0] -
move.int[i];
                                return position_fest[0];
                                break;

                            case "right":
                                position_fest[0] = position_fest[0] +
move.int[i];
                                return position_fest[0];
                                break;
```

```
                            case "upwards":
                                position_fest[1] = position_fest[1] +
move.int[i];
                                return position_fest[1];
                                break;

                            case "downwards":
                                position_fest[1] = position_fest[1] -
move.int[i];
                                return position_fest[1];
                                break;
                        }
                        break;

                    case 90:

                        switch(  movement_data[i]["direction"] ) {

                            case "forwards":
                                position_fest[0] = position_fest[0] -
move.int[i];
                                return position_fest[0];
                                break;

                            case "backwards":
                                position_fest[0] = position_fest[0] +
move.int[i];
                                return position_fest[0];
                                break;

                            case "left":
                                position_fest[2] = position_fest[2] +
move.int[i];
                                return position_fest[2];
                                break;

                            case "right":
                                position_fest[2] = position_fest[2] -
move.int[i];
                                return position_fest[2];
                                break;

                            case "upwards":
```

```
                                position_fest[1] = position_fest[1] +
move.int[i];
                                return position_fest[1];
                                break;

                            case "downwards":
                                position_fest[1] = position_fest[1] -
move.int[i];
                                return position_fest[1];
                                break;
                        }
                        break;

                    case 180:

                        switch(  movement_data[i]["direction"] ) {

                            case "forwards":
                                position_fest[2] = position_fest[2] +
move.int[i];
                                return position_fest[2];
                                break;

                            case "backwards":
                                position_fest[2] = position_fest[2] -
move.int[i];
                                return position_fest[2];
                                break;

                            case "left":
                                position_fest[0] = position_fest[0] +
move.int[i];
                                return position_fest[0];
                                break;

                            case "right":
                                position_fest[0] = position_fest[0] -
move.int[i];
                                return position_fest[0];
                                break;

                            case "upwards":
```

```
                                position_fest[1] = position_fest[1] +
move.int[i];
                                return position_fest[1];
                                break;

                            case "downwards":
                                position_fest[1] = position_fest[1] -
move.int[i];
                                return position_fest[1];
                                break;
                        }
                        break;

                    case 270:

                        switch(  movement_data[i]["direction"] ) {

                            case "forwards":
                                position_fest[0] = position_fest[0] +
move.int[i];
                                return position_fest[0];
                                break;

                            case "backwards":
                                position_fest[0] = position_fest[0] -
move.int[i];
                                return position_fest[0];
                                break;

                            case "left":
                                position_fest[2] = position_fest[2] -
move.int[i];
                                return position_fest[2];
                                break;

                            case "right":
                                position_fest[2] = position_fest[2] +
move.int[i];
                                return position_fest[2];
                                break;

                            case "upwards":
```

```
                                position_fest[1] = position_fest[1] +
move.int[i];
                                return position_fest[1];
                                break;

                            case "downwards":
                                position_fest[1] = position_fest[1] -
move.int[i];
                                return position_fest[1];
                                break;
                        }
                        break;

                }

                move.strings[i] = 'property: position; from: '+
position_fest[0] + ' ' + position_fest[1] + ' ' + position_fest[2] +';
to: '+ position_fest[0] + ' ' + position_fest[1] + ' ' +
position_fest[2] +'; dur: '+ move.duration +'; easing: linear; loop: 1;
delay: '+ (i+1)*move.delay  +'; startEvents:first';
                move.lines[i] = 'start:' + position_fest[0] + ' ' +
position_fest[1] + ' ' + position_fest[2] +'; end: '+ position_fest[0]
+ ' ' + position_fest[1] + ' ' + position_fest[2] + '; color: red';

                //position[0] = position[0];
                //position[1] = position[1];
                //position[2] = position[2];

            } else {


                if( movement_data[i]["direction"]== "left" ) {

                    // alert("1: " + movements[i]);

                    return move.strings[i] = 'property: rotation; from:
'+ 0 + ' ' + move.orientation_object + ' ' + 0 +' ;to: '+ 0 + ' ' +
(move.orientation_object + move.int[i]) + ' ' + 0 +'; dur: '+
move.duration +'; easing: linear; loop: 1; delay: '+ (i+1)*move.delay
+'; startEvents:first';


                    // alert("2: " + orientation);
```

```
                    switch( move.orientation_object ) {

                        case 0:
                            //alert("case 0" + orientation);
                            return move.orientation_object = 90;
                            //alert("case 0" + orientation);
                            break;
                        case 90:
                            return move.orientation_object = 180;
                            break;
                        case 180:
                            return move.orientation_object = 270;
                            break;
                        case 270:
                            return move.orientation_object = 0;
                            break;
                        default:
                            return alert("No Match, orientation is not
being updated!");
                            break;
                    }
                    // alert("3: " + orientation);

                } else if( movement_data[i]["direction"]== "right" ) {

                    return move.strings[i] = 'property: rotation; from:
'+ 0 + ' ' + move.orientation_object + ' ' + 0 +' ;to: '+ 0 + ' ' +
(move.orientation_object + move.int[i]) + ' ' + 0 +'; dur: '+
move.duration +'; easing: linear; loop: 1; move.delay: '+
(i+1)*move.delay  +'; startEvents:first';


                    switch( move.orientation_object ) {

                        case 0:
                            return move.orientation_object = 270;
                            break;
                        case 90:
                            return move.orientation_object = 0;
                            break;
                        case 180:
                            return move.orientation_object = 90;
```

```
                            break;
                        case 270:
                            return move.orientation_object = 180;
                            break;
                        default:
                            return alert("No Match, orientation is not
being updated!");
                            break;
                    }

                } else {

                    // movement is action!



                }

                move.lines[i] = 'start:' + position_fest[0] + ' ' +
position_fest[1] + ' ' + position_fest[2] +'; end: '+ position_fest[0]
+ ' ' + position_fest[1] + ' ' + position_fest[2] + '; color: red';

            }

            return [move.positions[i] = [position_fest[0],
position_fest[1], position_fest[2]],
            move.orientations[i] = move.orientation_object];

        }}
```