

Übungsblatt 4

Aufgabenlösung

Abgabe: 19.06.2017

4.1 *Curried in Space*

Server

Zunächst wurden alle Nachrichtentypen erstellt und ihre entsprechende Kodierung als JSON-Objekt implementiert.

```

22 data Message = Ping | Pong | Init | Update | Seek |
23   CreateLaser Int (Double, Double) (Double, Double) Double |
24   CreateAsteroid (Double, Double) ASize AColor |
25   Destroy { destroy :: Int } |
26   SetLifes { sld :: Int, lifes :: Int } |
27   Cmd { sld :: Int, cmd :: Command } |
28   ClientId { clientId :: Int } |
29   Zorg { common :: CommonState } |
30   Asteroid {
31     common :: CommonState,
32     size :: ASize,
33     color :: AColor } |
34   Spaceship { common :: CommonState } |
35   Laser { common :: CommonState, shooter :: Int } deriving (Generic, Show)
36
37 data Command = F | B | L | R | S deriving (Generic, Show)
38
39 data CommonState = CommonState {
40   ident :: Int,
41   pos :: (Double, Double),
42   velo :: (Double, Double),
43   acc :: Double,
44   omega :: Double,
45   phi :: Double
46 } deriving (Generic, Show)
47
48 data ASize = Tiny | Small | Medium | Big deriving (Generic, Show, Eq)
49 data AColor = Brown | Gray deriving (Generic, Show, Eq)
50
51 currentTimeMillis :: IO Integer
52 currentTimeMillis = (round ∘ (1000 *)) <$> getPOSIXTime
53
54 timestamped :: Message → IO TimestampedMessage
55 timestamped msg = currentTimeMillis >>= \time → return $ TimestampedMessage time msg
56
57 data TimestampedMessage = TimestampedMessage {
58   timestamp :: Integer,
59   payload :: Message
60 } deriving (Generic, Show)
61
62 data ServerMessage =

```

```

63     Connect {
64         socketOut :: (TimestampedMessage → IO ()),
65         connectionActor :: MVar (ActorRef ServerMessage)
66     }
67 | Disconnect Int
68 | Msg TimestampedMessage
69 | Ok
70
71 instance Show ServerMessage where
72     show (Connect _ _) = "Connect_<.>"
73     show (Disconnect _) = "Disconnect"
74     show (Msg x) = "Msg_" ++ show x
75     show Ok = "Ok"
76
77 —————
78 — JSON En/Decoding —
79
80 jsonOptions :: Options
81 jsonOptions = defaultOptions {
82     sumEncoding = ObjectWithSingleField,
83     allNullaryToStringTag = False
84 }
85
86 instance ToJSON Message where
87     toEncoding = genericToEncoding jsonOptions
88
89 instance FromJSON Message where
90     parseJSON = genericParseJSON jsonOptions
91
92 instance ToJSON TimestampedMessage where
93     toEncoding = genericToEncoding jsonOptions
94
95 instance FromJSON TimestampedMessage where
96     parseJSON = genericParseJSON jsonOptions
97
98 instance ToJSON CommonState where
99     toEncoding = genericToEncoding jsonOptions
100
101 instance FromJSON CommonState where
102     parseJSON = genericParseJSON jsonOptions
103
104 instance ToJSON ASize where
105     toEncoding = genericToEncoding jsonOptions
106
107 instance FromJSON ASize where
108     parseJSON = genericParseJSON jsonOptions
109
110 instance ToJSON AColor where
111     toEncoding = genericToEncoding jsonOptions
112
113 instance FromJSON AColor where
114     parseJSON = genericParseJSON jsonOptions
115
116 instance ToJSON Command where
117     toEncoding = genericToEncoding jsonOptions
118

```

instance FromJSON Command **where**

parseJSON = genericParseJSON jsonOptions

Danach wurde auf dem Server ein Connection-Aktor erstellt. Dieser repräsentiert jeweils eine Verbindung mit einem Client.

```

11 connection :: Int → ActorRef ServerMessage → ActorIO ServerMessage (ActorRef ServerMessage)
12 connection n univ = actor ("connection" ++ show n) receive where
13   receive = λcase
14     Connect out res → do
15       log Info "new_client_connected"
16       self >>= liftIO ∘ putMVar res
17       become $ connected out
18     Disconnect _ → do
19       univ ! Disconnect n
20       log Info "connection_closed"
21       stop
22   connected send = λcase
23     Msg msg@(TimestampedMessage t m) → do
24       case m of
25         Ping → univ ! (Msg (TimestampedMessage t (ClientId n)))
26         Cmd _ cmd → univ ! (Msg (TimestampedMessage t (Cmd n cmd)))
27         _ → liftIO $ send msg
28       become $ connected send
29     Disconnect _ → do
30       univ ! Disconnect n
31       log Info "connection_closed"
32       stop

```

Anschließend wurde ein Universums-Aktor erstellt. Dieser verwaltet alle anderen Objekte in der Simulation.

```

15 universe :: ActorIO ServerMessage (ActorRef ServerMessage)
16 universe = actor "universe" $ initialize 0 where
17   initialize idCount = λcase
18     Msg (TimestampedMessage t Init) → do
19       num ← liftIO $ randomRIO (8,20)
20       asteroids ← forM [idCount..idCount+num] $ λindex → do
21         rnds ← liftIO ∘ replicateM 8 $ randomRIO (0,1.0)
22         let pos = (500 - (rnds !! 0) * 1000, 500 - (rnds !! 1) * 1000)
23         let omega = (rnds !! 2) * pi * 2
24         let phi = (rnds !! 3) - 0.5
25         let velo = (50 - (rnds !! 4) * 100, 50 - (rnds !! 5) * 100)
26         let size = case round ((rnds !! 6) * 4) of
27           0 → Tiny
28           1 → Small
29           2 → Medium
30           _ → Big
31         let color = case round ((rnds !! 7) * 2) of
32           0 → Brown
33           _ → Gray
34         asteroid index pos velo 0 omega phi size color
35         become $ receive (idCount + length asteroids) Map.empty (Map.fromList $ zip [idCount..idCount+num]
36           forM_ asteroids (! (Msg (TimestampedMessage t Update))))
37   receive idCount connections asteroids spaceships lasers zorgs = λcase
38     Connect out res → do
39       self >>= λs → do
40         connection ← connection idCount s
41         forward connection (Connect out res)

```

```

42     become $ receive (idCount+1) (Map.insert idCount connection connections) asteroids spaceships
43 Disconnect cld → do
44     sender >>= λcon → do
45         let connections' = Map.delete cld connections
46         become $ receive idCount connections' asteroids spaceships lasers zorgs
47 destroy@(Msg (TimestampedMessage t (Destroy id))) → do
48     let new = map (Map.delete id) [asteroids, spaceships, lasers, zorgs]
49     forM_ connections (! destroy)
50     become $ receive idCount connections (new !! 0) (new !! 1) (new !! 2) (new !! 3)
51 msg@(Msg (TimestampedMessage t (SetLives id lifes))) → do
52     let con = Map.lookup id connections
53     when (isJust con) $ fromJust con ! msg
54     become $ receive idCount connections asteroids spaceships lasers zorgs
55 Msg (TimestampedMessage t (ClientId cld)) →
56     case Map.lookup cld connections of
57         Just connection → do
58             sp ← spaceship cld (0,0) (0,0) 0 0 0
59             num ← liftIO $ randomRIO (2,5)
60             newZorgs ← forM [idCount..idCount+num] $ λindex → do
61                 [xR,yR] ← liftIO ∘ replicateM 2 $ randomRIO (0,1.0)
62                 let pos = (500 - xR * 1000, 500 - yR * 1000)
63                 let velo = (0,0)
64                 let a = 0
65                 let omega = 0
66                 let phi = 0
67                 zorg index pos velo a omega phi sp
68                 (liftIO $ timestamped (ClientId cld)) >>= λmsg → connection ! Msg msg
69                 (liftIO $ timestamped Update) >>= λmsg → (sp ! Msg msg)
70                 forM_ newZorgs $ λz → (liftIO $ timestamped Update) >>= λmsg → (z ! Msg msg)
71                 forM_ newZorgs $ λz → (liftIO $ timestamped Seek) >>= λmsg → scheduleOnce 1500 z $ Msg msg
72                 let zorgs' = Map.union (Map.fromList $ zip [idCount..idCount+num] newZorgs) zorgs
73                 become $ receive (idCount+num+1) connections asteroids (Map.insert cld sp spaceships)
74                     lasers zorgs'
75         Nothing → do
76             liftIO $ print "Unexpected: connection not available"
77             become $ receive idCount connections asteroids spaceships lasers zorgs
78 update@(Msg (TimestampedMessage t (Asteroid _ _ _))) → do
79     become $ receive idCount connections asteroids spaceships lasers zorgs
80     forM_ zorgs (! update)
81     forM_ spaceships (! update)
82     forM_ connections (! update)
83 update@(Msg (TimestampedMessage t (Zorg _))) → do
84     become $ receive idCount connections asteroids spaceships lasers zorgs
85     forM_ connections (! update)
86 Msg (TimestampedMessage t (CreateAsteroid pos size color)) → do
87     rnds ← liftIO ∘ replicateM 4 $ randomRIO (0,1.0)
88     let omega = (rnds !! 0) * pi * 2
89     let phi = (rnds !! 1) - 0.5
90     let velo = (50 - (rnds !! 2) * 100, 50 - (rnds !! 3) * 100)
91     asteroid ← asteroid idCount pos velo 0 omega phi size color
92     let asteroids' = Map.insert idCount asteroid asteroids
93     asteroid ! Msg (TimestampedMessage t Update)
94     become $ receive (idCount+1) connections asteroids' spaceships lasers zorgs
95 update@(Msg (TimestampedMessage t (Spaceship _))) → do
96     become $ receive idCount connections asteroids spaceships lasers zorgs
97     forM_ connections (! update)

```

```

98   update@(Msg (TimestampedMessage t (Laser st sld))) → do
99     become $ receive idCount connections asteroids spaceships lasers zorgs
100   forM_ asteroids (! update)
101   forM_ spaceships (! update)
102   forM_ zorgs (! update)
103   forM_ connections (! update)
104   msg@(Msg (TimestampedMessage t (Cmd sld cmd))) → do
105     case Map.lookup sld spaceships of
106       Just sp → sp ! msg
107       _ → liftIO $ print "spaceship_is_dead"
108   become $ receive idCount connections asteroids spaceships lasers zorgs
109   msg@(Msg (TimestampedMessage t (CreateLaser sld pos v@(vx,vy) omega))) → do
110     let velo' = add v $ scalarMult (rotate (0,1) omega) 1000
111     let a = 0
112     let phi = 0
113     laser ← laser idCount sld pos velo' a omega phi
114     laser ! Msg (TimestampedMessage t Update)
115     become $ receive (idCount+1) connections asteroids spaceships (Map.insert idCount laser lasers) zo

```

Alle weiteren Objekte werden ebenfalls durch Aktoren dargestellt, die jeweils bei einer Update-Nachricht ihre neue Position im Raum berechnen und an das Universum senden.

```

118 asteroid :: Int → (Double,Double) → (Double, Double) → Double → Double → Double → ASize →
119 AColor → ActorIO ServerMessage (ActorRef ServerMessage)
120 asteroid id p@(px,py) v@(vx,vy) a omega phi size color = actor (show id) (receive p v omega) where
121   receive (px,py) (vx,vy) omega = λcase
122     Msg (TimestampedMessage prevTime Update) → do
123       time ← liftIO $ currentTimeMillis
124       let δS = (fromIntegral $ time - prevTime) / 1000
125       let p' = (px+vx*δS, py+vy*δS)
126       let v' = (vx - δS * sin omega * a, vy + δS * cos omega * a)
127       let omega' = omega + δS * phi
128       parent >>= (! (Msg (TimestampedMessage time $ Asteroid (CommonState id p' v' a omega' phi) size col
129 self >>= λs → scheduleOnce 300 s (Msg (TimestampedMessage time Update))
130       become $ receive p' v' omega'
131   Msg (TimestampedMessage t (Laser st sld)) → do
132     if (intersects (px,py) (sizeToRadius size) $ pos st) then do
133       when (size ≠ Tiny) $ do
134         let size' = case size of
135           Small → Tiny
136           Medium → Small
137           Big → Medium
138         numAst ← liftIO $ randomRIO (2,4)
139         replicateM_ numAst $ do
140           parent >>= (! Msg (TimestampedMessage t $ CreateAsteroid (px,py) size' color))
141           parent >>= (! Msg (TimestampedMessage t $ Destroy id))
142         stop
143     else
144       become $ receive (px,py) (vx,vy) omega
145
146
147 spaceship :: Int → (Double, Double) → (Double, Double) → Double → Double → Double → ActorIO ServerM
148 spaceship id p@(px,py) v@(vx,vy) alnit omega philnit = actor (show id) (receive p v alnit omega philnit
149   receive (px,py) (vx,vy) a omega phi lifes lastCollisionTime = λcase
150     Msg (TimestampedMessage prevTime Update) → do
151       time ← liftIO $ currentTimeMillis

```

```

152   let  $\delta S = (\text{fromIntegral } \$ \text{ time} - \text{prevTime}) / 1000$ 
153   let  $p' = (px + vx * \delta S, py + vy * \delta S)$ 
154   let  $v' = (vx - \delta S * \sin \text{ omega} * a, vy + \delta S * \cos \text{ omega} * a)$ 
155   let  $\text{omega}' = \text{omega} + \delta S * \text{phi}$ 
156   parent >>= (! (Msg (TimestampedMessage time $ Spaceship (CommonState id p' v' a omega' phi))))
157   self >>=  $\lambda s \rightarrow \text{scheduleOnce } 100 \text{ s } (\text{Msg (TimestampedMessage time Update)})$ 
158   become $ receive p' v' alnit omega' phi nit lifes lastCollisionTime
159   Msg (TimestampedMessage time (Cmd - S))  $\rightarrow$  do
160     parent >>= (! Msg (TimestampedMessage time (CreateLaser id (px,py) (vx,vy) omega)))
161     become $ receive (px,py) (vx,vy) a omega phi lifes lastCollisionTime
162   Msg (TimestampedMessage - (Cmd - cmd))  $\rightarrow$  do
163     let (a', phi') = case cmd of
164       F  $\rightarrow$  (500, phi)
165       B  $\rightarrow$  (-200, phi)
166       L  $\rightarrow$  (a, -3)
167       R  $\rightarrow$  (a, 3)
168       _  $\rightarrow$  (a, phi)
169     become $ receive (px,py) (vx,vy) a' omega phi' lifes lastCollisionTime
170   Msg (TimestampedMessage t (Asteroid st size -))  $\rightarrow$  do
171     now  $\leftarrow$  liftIO $ currentTimeMillis
172     if now - lastCollisionTime  $\geq$  3000 then
173       if intersectsC (px,py) spaceshipSize (pos st) (sizeToRadius size) then do
174         parent >>= (! Msg (TimestampedMessage now (SetLifes id $ lifes - 1)))
175         become $ receive (px,py) (vx,vy) a omega phi (lifes - 1) now
176         when (lifes == 1) $ do
177           parent >>= (! Msg (TimestampedMessage now (Destroy id)))
178           stop
179       else
180         become $ receive (px,py) (vx,vy) a omega phi lifes lastCollisionTime
181     else
182       become $ receive (px,py) (vx,vy) a omega phi lifes lastCollisionTime
183   Msg (TimestampedMessage t Seek)  $\rightarrow$  do
184     sender >>= (! Msg (TimestampedMessage t $ Spaceship (CommonState id (px,py) (vx,vy) a omega phi)))
185     become $ receive (px,py) (vx,vy) a omega phi lifes lastCollisionTime
186   Msg (TimestampedMessage t (Laser st sld))  $\rightarrow$  do
187     if (intersects (px,py) spaceshipSize $ pos st) && sld  $\neq$  id then do
188       parent >>= (! Msg (TimestampedMessage t (SetLifes id $ lifes - 1)))
189       become $ receive (px,py) (vx,vy) a omega phi (lifes - 1) t
190       when (lifes == 1) $ do
191         parent >>= (! Msg (TimestampedMessage t (Destroy id)))
192         stop
193     else
194       become $ receive (px,py) (vx,vy) a omega phi lifes lastCollisionTime
195
196
197
198   laser :: Int  $\rightarrow$  Int  $\rightarrow$  (Double, Double)  $\rightarrow$  (Double, Double)  $\rightarrow$  Double  $\rightarrow$  Double  $\rightarrow$  Double  $\rightarrow$  ActorIO ()
199   laser id shooterId pos velo alnit omegalnit phi nit = do
200     time  $\leftarrow$  liftIO $ currentTimeMillis
201     actor (show id) (receive pos velo alnit omegalnit phi nit time) where
202       receive (px,py) (vx,vy) a omega phi t =  $\lambda$  case
203         Msg (TimestampedMessage prevTime Update)  $\rightarrow$  do
204           if prevTime - t > 1000 then do
205             msg  $\leftarrow$  liftIO $ timestamped $ Destroy id
206             parent >>= (! Msg msg)
207             stop

```

```

208     else do
209         time ← liftIO $ currentTimeMillis
210         let  $\delta S = (\text{fromIntegral } \$ \text{ time} - \text{prevTime}) / 1000$ 
211         let  $p' = (px + vx * \delta S, py + vy * \delta S)$ 
212         let  $v' = (vx - \delta S * \sin \text{ omega} * a, vy + \delta S * \cos \text{ omega} * a)$ 
213         let  $\text{omega}' = \text{omega} + \delta S * \text{phi}$ 
214         parent >>= (! (Msg (TimestampedMessage time $ Laser (CommonState id p' v' a omega' phi) shooter
215         self >>=  $\lambda s \rightarrow \text{scheduleOnce } 100 \text{ s } (\text{Msg } (\text{TimestampedMessage time Update}))$ 
216         become $ receive p' v' a omega' phi t
217
218
219 zorg :: Int → (Double, Double) → (Double, Double) → Double → Double → Double →
220 ActorRef ServerMessage → ActorIO ServerMessage (ActorRef ServerMessage)
221 zorg zld plnit vlnit alnit omegalnit philnit sp = actor (show zld) (receive plnit vlnit alnit omegalnit
222 receive p@(px,py) v@(vx,vy) a omega phi =  $\lambda \text{case}$ 
223   Msg (TimestampedMessage prevTime Update) → do
224     time ← liftIO $ currentTimeMillis
225     let  $\delta S = (\text{fromIntegral } \$ \text{ time} - \text{prevTime}) / 1000$ 
226     let  $p' = (px + vx * \delta S, py + vy * \delta S)$ 
227     let  $v' = (\min 300 (vx - \delta S * \sin \text{ omega} * a), \min 300 (vy + \delta S * \cos \text{ omega} * a))$ 
228     let  $\text{omega}' = \text{omega} + \delta S * \text{phi}$ 
229     parent >>= (! (Msg (TimestampedMessage time $ Zorg (CommonState zld p' v' a omega' phi))))
230     self >>=  $\lambda s \rightarrow \text{scheduleOnce } 100 \text{ s } (\text{Msg } (\text{TimestampedMessage time Update}))$ 
231     prob ← liftIO $ randomRIO (0, 1.0)
232     when (prob ≤ (0.05 :: Double)) $ parent >>= (! Msg (TimestampedMessage time (CreateLaser zld p' v'
233     become $ receive p' v' alnit omega' philnit
234 msg@(Msg (TimestampedMessage t Seek)) → do
235   sp ! msg
236   become $ receive p v a omega phi
237 Msg (TimestampedMessage t (Spaceship st)) → do
238   let distance = add (pos st) (−px, −py)
239   let normalizedDistance = unit distance
240   let  $\text{phi}' = (((\text{atan2 } (\text{snd distance}) (\text{fst distance})) - (1/2 * \pi)) - (\text{omega})) * 3$ 
241   let  $a' = (((\text{dist } (\text{pos st}) p) - 100) - 200) / 1$ 
242   time ← liftIO $ currentTimeMillis
243   self >>=  $\lambda s \rightarrow \text{scheduleOnce } 250 \text{ s } (\text{Msg } (\text{TimestampedMessage time Seek}))$ 
244   become $ receive p v a' omega phi'
245 Msg (TimestampedMessage t (Laser st sld)) → do
246   if (intersects (px,py) spaceshipSize $ pos st) && sld ≠ zld then do
247     parent >>= (! Msg (TimestampedMessage t (Destroy zld)))
248     stop
249 else
250   become $ receive p v a omega phi
251 Msg (TimestampedMessage t (Asteroid st size _)) → do
252   if intersectsC (px,py) spaceshipSize (pos st) (sizeToRadius size) then do
253     parent >>= (! Msg (TimestampedMessage t (Destroy zld)))
254     stop
255 else
256   become $ receive p v a omega phi
257
258 rotate :: (Double, Double) → Double → (Double, Double)
259 rotate (x,y) rad = let  $\cos' = \cos \text{ rad}$ ;  $\sin' = \sin \text{ rad}$  in  $(x * \cos' - y * \sin', x * \sin' + y * \cos')$ 
260
261 add :: (Double, Double) → (Double, Double) → (Double, Double)
262 add (x, y) (x', y') = (x+x', y+y')

```

```

263
264 scalarMult :: (Double, Double) → Double → (Double, Double)
265 scalarMult (x,y) k = (x*k, y*k)
266
267 divide :: (Double, Double) → Double → (Double, Double)
268 divide _ 0 = (0,0)
269 divide (x,y) k = (x/k, y/k)
270
271 unit :: (Double, Double) → (Double, Double)
272 unit p = p 'divide' sizeVec p
273
274 dist :: (Double, Double) → (Double, Double) → Double
275 dist (px,py) (px', py') = sqrt $ (px-px')^2 + (py-py')^2
276
277 intersects :: (Double, Double) → Double → (Double, Double) → Bool
278 intersects c r p = dist c p ≤ r
279
280 intersectsC :: (Double, Double) → Double → (Double, Double) → Double → Bool
281 intersectsC p r p' r' = dist p p' < r + r'
282
283 sizeVec :: (Double, Double) → Double
284 sizeVec (x,y) = sqrt $ x^2 + y^2
285
286 sizeToRadius :: ASize → Double
287 sizeToRadius Tiny = 16
288 sizeToRadius Small = 27
289 sizeToRadius Medium = 44
290 sizeToRadius Big = 95
291
292 spaceshipSize :: Double
293 spaceshipSize = 60

```

Client

Im Client wurden zuerst Objekte um einen Identifikation erweitert, sodass die Objekte einfacher wiedergefunden werden können.

```

7  trait SpaceObject {
8      val sprite: Sprite
9
10     var zorg : Boolean = false
11     var ident : Int = -1
12
13     def pos = Vector2d(sprite.position.x, sprite.position.y)
14     def pos_=(value: Vector2d) = sprite.position.set(value.x, value.y)
15
16     def orientation = sprite.rotation - Math.PI
17     def orientation_=(value: Double) = sprite.rotation = (value + Math.PI)
18
19     var angularVelocity = 0.0          // rad / s
20     var velocity = new Vector2d(0,0)  // px / s
21
22     var acceleration = 0.0             // px / s ^ 2
23
24     var exists: Boolean = true
25
26     def α = sprite.α
27     def α_=(value: Double) = sprite.α = value

```



```

28
29
30 def remove() = {
31     exists = false
32 }
33
34 def update( $\delta$ : Double) = { //  $\delta$ : s
35     velocity += Vector2d.unit.rotate(orientation) * ( $\delta$  * acceleration)
36     orientation +=  $\delta$  * angularVelocity
37     pos += velocity *  $\delta$ 
38 }
39 }
40
41 class SimpleSpaceObject(val id: Int, val sprite: Sprite, initialPos: Vector2d = Vector2d.zero, initialOr
0.0) extends SpaceObject {
42     ident = id
43     pos = initialPos
44     orientation = initialOrientation
45
46     def canEqual(other : Any) = other.isInstanceOf[SimpleSpaceObject]
47     override def equals(other : Any) : Boolean =
48         other match {
49             case other : SimpleSpaceObject  $\Rightarrow$  other.canEqual(this) && this.hashCode == other.hashCode
50             case _  $\Rightarrow$  false
51         }
52     override def hashCode : Int = {
53         return ident
54     }
55 }
56
57 /**
58  * Creates Space Objects
59  */
60 class SpaceObjectFactory(val textures: Resources.SpaceTextures) {
61     def player(id : Int, isZorg : Boolean, color: Color.Player, variant: Int = 0): PlayerShip =
62         new PlayerShip(id, isZorg, color, variant)(textures)
63
64     def meteor(id : Int, size: Size.Meteor, color: Color.Meteor): SpaceObject =
65         new SimpleSpaceObject(id, new Sprite(Util.chooseFrom(textures.meteors(color)(size)))) {
66             sprite.anchor.set(0.5,0.5)
67             ident = id
68         }
69
70     def laser(id : Int, rel: SpaceObject, color: Color.Laser = Color.Laser.Red, variant: Int = 0) =
71     {
72         val sprite = new Sprite(textures.lasers(color)(variant))
73         sprite.anchor.x = 0.5
74         sprite.anchor.y = 0.0
75         val laser = new SimpleSpaceObject(id, sprite, rel.pos, rel.orientation) {
76             override def update( $\delta$ : Double) = {
77                  $\alpha$  -=  $\delta$  / 2
78                 //if ( $\alpha \leq 0.0$ ) this.remove()
79                 super.update( $\delta$ )
80             }
81         }
82         laser.velocity = rel.velocity + Vector2d.unit.rotate(rel.orientation) * 1000

```

```

82     laser
83 }
84 }
85
86 case class PlayerShip(id : Int, isZorg : Boolean, color: Color.Player, variant: Int)(textures: SpaceTextures) {
87     val sprite = new Sprite(textures.players.ships(color)(variant))
88     ident = id
89     zorg = isZorg
90     sprite.anchor.set(0.5,0.5)
91
92     val thrustContainer = new Container()
93     sprite.addChild(thrustContainer)
94
95     val (thrustBL,thrustBR,thrustFL,thrustFR) = {
96         import js.Dynamic.literal
97         def thrustEmitterSettings(back: Boolean,left: Boolean) = literal (
98             "α" → literal ( "start" → 0.8, "end" → 0.0 ),
99             "scale" → literal ( "start" → 0.1, "end" → 2 ),
100            "color" → literal ( "start" → "ffffff", "end" → "331100" ),
101            "speed" → literal ( "start" → (if (back) 300 else 250), "end" → (if (back) 300 else 250) ),
102            "startRotation" → (if (back) literal ( "min" → 89, "max" → 91 ) else literal ( "min" →
265, "max" → 275)),
103            "lifetime" → literal ( "min" → 0.5, "max" → 1 ),
104            "frequency" → (if (back) 0.003 else 0.003),
105            "maxParticles" → 256,
106            "pos" → literal ( "x" → -45, "y" → 25 ),
107            "spawnType" → "circle",
108            "spawnCircle" → new Circle(0,0,if(back)10 else 5)
109        )
110
111        def thrustEmitter(back: Boolean, left: Boolean) =
112            new Emitter(thrustContainer,js.Array(textures.stars(2)),thrustEmitterSettings(back, left))
113
114        (
115            thrustEmitter(true,true),
116            thrustEmitter(true,false),
117            thrustEmitter(false,true),
118            thrustEmitter(false,false)
119        )
120    }
121
122    def updateThrust(δ: Double) = {
123        thrustContainer.rotation -= sprite.rotation
124        thrustBL.emit = acceleration > 0 || angularVelocity > 0 && acceleration == 0
125        thrustBR.emit = acceleration > 0 || angularVelocity < 0 && acceleration == 0
126        thrustFL.emit = acceleration < 0 || angularVelocity < 0 && acceleration == 0
127        thrustFR.emit = acceleration < 0 || angularVelocity > 0 && acceleration == 0
128        val rot = sprite.rotation * PIXI.RAD_TO_DEG
129        thrustBL.rotate(rot)
130        thrustBR.rotate(rot)
131        thrustFL.rotate(rot)
132        thrustFR.rotate(rot)
133        val spawnBL = Vector2d(-45,25).rotate(sprite.rotation)
134        val spawnBR = Vector2d(45,25).rotate(sprite.rotation)
135        val spawnFL = Vector2d(-47,-8).rotate(sprite.rotation)
136        val spawnFR = Vector2d(47,-8).rotate(sprite.rotation)

```

```

137     thrustBL.updateSpawnPos(spawnBL.x,spawnBL.y)
138     thrustBR.updateSpawnPos(spawnBR.x,spawnBR.y)
139     thrustFL.updateSpawnPos(spawnFL.x,spawnFL.y)
140     thrustFR.updateSpawnPos(spawnFR.x,spawnFR.y)
141     thrustBL.update( $\delta$ )
142     thrustBR.update( $\delta$ )
143     thrustFL.update( $\delta$ )
144     thrustFR.update( $\delta$ )
145 }
146
147
148 private val damages = textures.players.damage(variant).map { t =>
149     val ds = new Sprite(t)
150     ds.blendMode = PIXI.BLEND.MODES.MULTIPLY
151     ds.anchor.set(0.5,0.5)
152     ds
153 }
154
155 private var damageLevel_ = 0
156 def damageLevel = damageLevel_
157 def damageLevel_=(level: Int) = {
158     damages.lift(damageLevel_ - 1).foreach(sprite.removeChild(_))
159     damages.lift(level - 1).foreach(sprite.addChild(_))
160     if (level > damages.size) damageLevel_ = 0 else damageLevel_ = level
161 }
162
163 private var damaged = 0.0
164 private var damageFilter = new filters.TwistFilter()
165 damageFilter.radius = sprite.width / 2
166 damageFilter.offset = new Point(sprite.width / 2, sprite.height / 2)
167
168 def hit() = damaged = 1.0
169 sprite.filters = js.Array(damageFilter)
170
171 override def update( $\delta$ : Double) = {
172     super.update( $\delta$ )
173     if (damaged > 0) {
174         damaged = Math.max(0,damaged -  $\delta$ )
175         sprite.tint = 0xffffffff - 0x000101 * (damaged * 255).toInt
176         damageFilter.angle = damaged * 360
177     } else if (damaged == 0) {
178         damaged = -1
179         sprite.tint = 0xffffffff
180         damageFilter.angle = 0
181     }
182     updateThrust( $\delta$ )
183 }
184 }

```

Zum Wiederauffinden wurde eine Look-up-Methode implementiert.

```

66     def lookup(id : Int): Option[SpaceObject] = {
67         objects_.find(x => x.ident == id)
68     }

```

Ebenso wurde auch im Client die Nachrichten implementiert, die bereits auf dem Server implementiert wurden.

```

9  sealed trait Message
10 case object Ping extends Message
11 case object Pong extends Message
12 case class Asteroid(common : CommonState, size : ASize, color : AColor) extends Message
13 case class Spaceship(common : CommonState) extends Message
14 case class Laser(common : CommonState, shooter : Int) extends Message
15 case class Cmd(sld : Int, cmd : Command) extends Message
16 case class ClientId(clientId : Int) extends Message
17 case class Destroy(destroy : Int) extends Message
18 case class SetLives(sld : Int, lives : Int) extends Message
19 case class Zorg(common : CommonState) extends Message
20
21 case class CommonState(ident : Int, pos : (Double, Double), velo : (Double, Double), acc : Double,
22
23 sealed trait AColor
24 case object Brown extends AColor
25 case object Gray extends AColor
26
27 sealed trait ASize
28 case object Tiny extends ASize
29 case object Small extends ASize
30 case object Medium extends ASize
31 case object Big extends ASize
32
33 sealed trait Command
34 case object F extends Command
35 case object B extends Command
36 case object L extends Command
37 case object R extends Command
38 case object S extends Command
39
40 //////////////////////////////////////
41 // JSON De/serialization //
42 //////////////////////////////////////
43
44 sealed trait JSONConfig {
45   import io.circe.Encoder
46   import shapeless.{ Generic, HNil }
47
48   implicit def encodeCaseObject[A <: Product](implicit
49     gen: Generic.Aux[A, HNil]
50   ): Encoder[A] = Encoder.instance(_ => Json.arr())
51 }
52
53 object TimestampedMessage extends JSONConfig {
54   def apply(payload: Message): TimestampedMessage = TimestampedMessage(System.currentTimeMillis(), payload)
55
56   implicit val encodeTimestampedMessage: Encoder[TimestampedMessage] =
57     deriveEncoder[TimestampedMessage]
58
59   implicit val decodeTimestampedMessage: Decoder[TimestampedMessage] =
60     deriveDecoder[TimestampedMessage]
61 }
62
63 object Message extends JSONConfig {
64   implicit val encodeMessage: Encoder[Message] =

```

```

65     deriveEncoder[Message]
66
67     implicit val decodeMessage: Decoder[Message] =
68         deriveDecoder[Message]
69 }
70
71 object CommonState extends JSONConfig {
72     implicit val encodeMessage: Encoder[CommonState] =
73         deriveEncoder[CommonState]
74
75     implicit val decodeMessage: Decoder[CommonState] =
76         deriveDecoder[CommonState]
77 }
78
79 object AColor extends JSONConfig {
80     implicit val encodeMessage: Encoder[AColor] =
81         deriveEncoder[AColor]
82
83     implicit val decodeMessage: Decoder[AColor] =
84         deriveDecoder[AColor]
85 }
86
87 object ASize extends JSONConfig {
88     implicit val encodeMessage: Encoder[ASize] =
89         deriveEncoder[ASize]
90
91     implicit val decodeMessage: Decoder[ASize] =
92         deriveDecoder[ASize]
93 }
94
95 object Command extends JSONConfig {
96     implicit val encodeMessage: Encoder[Command] =
97         deriveEncoder[Command]
98
99     implicit val decodeMessage: Decoder[Command] =
100         deriveDecoder[Command]
101 }

```

Die eintreffenden Objekte erneuern die Werte für ihre entsprechende Repräsentation auf dem Client. Sollte keine entsprechende Repräsentation vorhanden sein, wird eine neue Repräsentation erzeugt.

```

41     socket.onMessage { msg =>
42         console.log(msg.toString)
43         msg match {
44             case TimestampedMessage (t, Zorg (state)) =>
45                 val m = view.lookup(state.ident)
46                 m match {
47                     case Some(v) =>
48                         v.pos = Vector2d(state.pos._1, state.pos._2)
49                         v.orientation = state.omega
50                         v.angularVelocity = state.phi
51                         v.velocity = Vector2d(state.velo._1, state.velo._2)
52                         val  $\delta$  = (System.currentTimeMillis() - t) / 1000
53                         v.update( $\delta$ )
54                     case None =>
55                         val zorg = view.factory.player(state.ident, true, Color.Green, 2)
56                         view.spawn(zorg)

```

```

57         val  $\delta$  = (System.currentTimeMillis() - t) / 1000
58         zorg.update( $\delta$ )
59         view.addAmination(Animation.fade(zorg.sprite, 0.0, 1.0, 0.5 seconds))
60     }
61     case TimestampedMessage (t, SetLifes (sld, lifes)) =>
62         view.setLifes(lifes)
63     case TimestampedMessage (t, Destroy (id)) =>
64         var e = view.lookup(id)
65         e match {
66             case Some(v) =>
67                 console.log("Destroying_object_with_id:" + id)
68                 view.remove(v)
69                 v.remove()
70             case None => console.log("Deleting_non_existant_object_with_id:" + id)
71         }
72     case TimestampedMessage (t, ClientId (cld)) =>
73         val player = view.factory.player(cld, false, Color.Blue)
74         view.spawn(player)
75         view.focus(player)
76         val  $\delta$  = (System.currentTimeMillis() - t) / 1000
77         player.update( $\delta$ )
78         Control.bindKeyboard(socket, player, view)
79         view.addAmination(Animation.fade(player.sprite, 0.0, 1.0, 0.5 seconds))
80     case TimestampedMessage (t, Spaceship (state)) =>
81         val m = view.lookup(state.ident)
82         m match {
83             case Some(v) =>
84                 v.pos = Vector2d(state.pos._1, state.pos._2)
85                 v.orientation = state.omega
86                 v.angularVelocity = state.phi
87                 v.velocity = Vector2d(state.velo._1, state.velo._2)
88                 val  $\delta$  = (System.currentTimeMillis() - t) / 1000
89                 v.update( $\delta$ )
90             case None =>
91                 val spaceship = view.factory.player(state.ident, false, Util.choose(Color.Blue, Col
92                 view.spawn(spaceship)
93                 val  $\delta$  = (System.currentTimeMillis() - t) / 1000
94                 spaceship.update( $\delta$ )
95                 view.addAmination(Animation.fade(spaceship.sprite, 0.0, 1.0, 0.5 seconds))
96         }
97     case TimestampedMessage (t, Laser (state, sld)) =>
98         val m = view.lookup(state.ident)
99         m match {
100             case Some(v) =>
101                 console.log("Lazorz_coordz:_x:" + state.pos._1 + "y:" + state.pos._2)
102                 v.pos = Vector2d(state.pos._1, state.pos._2)
103                 v.orientation = state.omega
104                 v.angularVelocity = state.phi
105                 v.velocity = Vector2d(state.velo._1, state.velo._2)
106                 val  $\delta$  = (System.currentTimeMillis() - t) / 1000
107                 v.update( $\delta$ )
108             case None =>
109                 val p = view.lookup(sld)
110                 p match {
111                     case Some (p) =>
112                         val o = view.lookup(sld)

```

```

113         o match {
114             case Some (PlayerShip(_, isZ, _, _)) =>
115                 if (!isZ) {
116                     val laser = view.factory.laser(state.ident, p)
117                     view.spawnBelow(laser, p)
118                     laser.velocity = Vector2d(state.velo._1, state.velo._2)
119                     val  $\delta$  = (System.currentTimeMillis() - t) / 1000
120                     laser.update( $\delta$ )
121                 } else {
122                     val laser = view.factory.laser(state.ident, p, Color.Green, 1)
123                     view.spawnBelow(laser, p)
124                     laser.velocity = Vector2d(state.velo._1, state.velo._2)
125                     val  $\delta$  = (System.currentTimeMillis() - t) / 1000
126                     laser.update( $\delta$ )
127                 }
128             case _ => Unit
129         }
130     case None => Unit
131 }
132 }
133 case TimestampedMessage (t, Asteroid (state, size, color)) =>
134     val m = view.lookup(state.ident)
135     m match {
136         case Some (v) =>
137             v.pos = Vector2d(state.pos._1, state.pos._2)
138             v.orientation = state.omega
139             v.angularVelocity = state.phi
140             v.velocity = Vector2d(state.velo._1, state.velo._2)
141             val  $\delta$  = (System.currentTimeMillis() - t) / 1000
142             v.update( $\delta$ )
143         case None =>
144             val s = size match {
145                 case Tiny => Size.Tiny
146                 case Small => Size.Small
147                 case Medium => Size.Medium
148                 case Big => Size.Big
149             }
150             val c = color match {
151                 case Brown => Color.Brown
152                 case Gray => Color.Grey
153             }
154             val meteor = view.factory.meteor(state.ident, s, c)
155             meteor.pos = Vector2d(state.pos._1, state.pos._2)
156             meteor.orientation = state.omega
157             meteor.angularVelocity = state.phi
158             meteor.velocity = Vector2d(state.velo._1, state.velo._2)
159             view.spawn(meteor)
160             val  $\delta$  = (window.performance.now() - t) / 1000
161             meteor.update( $\delta$ )
162         }
163     case _ => Unit
164 }
165 }

```

Tests

Getestet wurden beide Aufgaben, indem die Simulation funktional ausprobiert wurde. Die Tests verliefen

erfolgreich.