Source code sample AS3

Extract from the main game loop and some more core code from a puzzler with a character that influences its environment via a ring menu.

```
/*
 * [...] marks repetitive parts or long sections with non-interesting material removed for demo-
 * purposes.
 * This code was written as part of a flash game at the University of Applied Sciences in Turku
 * Finland by Marc A. Modrow in 2009.
 * Copyright (c) 2009 Marc A. Modrow (mmodrow@uni-bremen.de)
 */

//[...]
/*
 * What has to be done every time is done here.
 */
function everyFrame(e:Event):void {
  // Fps Counter for Debug purposes
  var now:uint = getTimer();
  now -= fpsTimer;
  if (now != 0) {
    fps = (1000/now).toFixed(1) + " fps";
  }
  if (!fpsUpdateTimer.running) {
    fpsText.text = fps;
    fpsUpdateTimer.reset();
    fpsUpdateTimer.start();
  }
  fpsTimer = getTimer();
  //tidy up laser beam container
  deleteAllChildren(laserBeams);
  for (var i:int = 0; i < emitters.length; i++) {
    if (emitters[i] != null && emitters[i].alpha > 0.2) {
      emitters[i].beamLength = 0;
      //extend beam until it hits something and create a new one if it is a mirror or prism
      advance(new Point(emitters[i].x, emitters[i].y), emitters[i].rotation, emitters[i].colour,
          emitters[i], emitters[i]);
    }
  }
  //Win condition and check
  var openExit:Boolean = true
  for (i = 0; i < sensors.length; i++) {
    if (sensors[i]!= null && !sensors[i].activated) {
      openExit = false
    }
  }
  if (openExit) {
    exit.toggleMe(openExit);
  }
  mousePos.text = "mouseX " + mouseX + " mouseY " + mouseY
  if (player.hitTestObject(exit)) {
    if (!feedback.visible) {
      feedback.visible = true;
    }
    if (openExit) {
      if (!nextCode.visible && !cookie.visible) {
        playLevelCompleteSound();
      }
      if(levelCodes.indexOf(levelName.text)+1 == levelCodes.length){
        cookie.visible = true;
        level.text = winCode;
      } else {
        nextCode.text = levelCodes[levelCodes.indexOf(levelName.text)+1];
        feedback.text = "You completed this Level.\n Press enter or the code below to continue:";
        tips.text = "If you want come back later, write this code down somewhere. You can enter this
            text to the box right of me and press the arrow to come back later. \n\n\n\n" + nextCode.
            text;
        level.text = nextCode.text;
        nextCode.visible = true;
      }
    } else if (!nextCode.visible && !cookie.visible) {
      feedback.text = "First activate all sensors.\n Come back after you did this.";
    }
  //lose condition
  } else if (!player.alive){
    feedback.visible = true;
    feedback.text = "You died.\n Press enter or click the level load icon to restart the experiment or
        press space to revive and relocate the spider."; //Press 'scroll' to revive and/or 'pause' to
        reset.
  //play state
  } else if (feedback.text != ""){
    if (feedback.visible) {
      feedback.visible = false;
```

```
75|     }
76|     feedback.text = "";
77|   }
78| }
79|
80| /*
81|  * Load a level from an external xml file
82|  */
83| function loadLevel():void {
84|   // reset everything
85|   removeEventListener(Event.ENTER_FRAME, everyFrame)
86|   emitters = []; // contains all emitters, sending out laser beams.
87|   sensors = []; // contains all sensors receiving laser beams.
88|   buttons = [];
89|   var i:int = 0;
90|   for (;worldContainer.numChildren > i;) {
91|     if (worldContainer.getChildAt(i) is Border || worldContainer.getChildAt(i) is Player ||
          worldContainer.getChildAt(i) is Exit) {
92|       i++;
93|     } else {
94|       worldContainer.removeChildAt(i);
95|     }
96|   }
97|   i = 1;
98|   for (;pickList.numChildren > i;) {
99|     pickList.removeChildAt(i);
100|  }
101|  i = 0;
102|  for (;emitterContainer.numChildren > i;) {
103|    emitterContainer.removeChildAt(i);
104|  }
105|  deleteAllChildren(laserBeams);
106|  levelName.text = "";
107|  levelNumber.text = "";
108|  tips.text = "";
109|  nextCode.visible = false;
110|  player.visible = false;
111|  player.x = 0;
112|  player.y = 0;
113|  reach.visible = false;
114|  exit.visible = false;
115|  exit.x = 100;
116|  exit.y = 0;
117|  tip1.visible = false;
118|  tip2.visible = false;
119|  tip3.visible = false;
120|  tip4.visible = false;
121|  tip5.visible = false;
122|  tip6.visible = false;
123|  tip7.visible = false;
124|  tip1.removeEventListener(MouseEvent.CLICK, showTip1);
125|  tip2.removeEventListener(MouseEvent.CLICK, showTip2);
126|  tip3.removeEventListener(MouseEvent.CLICK, showTip3);
127|  tip1.removeEventListener(MouseEvent.CLICK, showTip4);
128|  tip2.removeEventListener(MouseEvent.CLICK, showTip5);
129|  tip3.removeEventListener(MouseEvent.CLICK, showTip6);
130|  // Initiate file call
131|  if(level.text != winCode){
132|    urlReq = new URLRequest(levelCodes.indexOf(level.text) + ".xml")
133|  } else {
134|    cookie.visible = true;
135|  }
136|  xmlLoader = new URLLoader(urlReq)
137|  xmlLoader.addEventListener(Event.COMPLETE, xmlLoaded)
138|  xmlLoader.addEventListener(IOErrorEvent.IO_ERROR, xmlError)
139| }
140|
141| /*
142|  * If something went wrong on loading the xml file
143|  */
144| function xmlError(e:ErrorEvent):void {
145|   if(level.text != "" && levelCodes.indexOf(level.text) == -1 && (level.text.length > 1 || !(int(level
        .text) < levelCodes.length && int(level.text) > 0))) {
146|     urlReq = new URLRequest(level.text + ".xml")
147|     xmlLoader = new URLLoader(urlReq)
148|     xmlLoader.addEventListener(Event.COMPLETE, xmlLoaded)
149|     xmlLoader.addEventListener(IOErrorEvent.IO_ERROR, badXmlError)
150|   } else {
151|     tips.text = "Something strange has happened. I don't think that we have such an experiment."; //
          Error loading XML data.\n\nCheck if the file you called exists.
152|   }
153| }
154|
155| /*
```

```
156|  * When inputting an invalid load codes
157|  */
158| function badXmlError(e:ErrorEvent):void {
159|   tips.text = "Something strange has happened. I don't think that we have such an experiment."; //
          Error loading XML data.\n\nCheck if the file you called exists.
160| }
161|
162| /*
163|  * Successfully loading xml file
164|  */
165| function xmlLoaded(e:Event):void {
166|   xml = XML(xmlLoader.data)
167|   levelName.text = xml.general.levelName;
168|   levelNumber.text = xml.general.levelNumber;
169|   if (unlockedLevels.text.indexOf(levelName.text) == -1) {
170|     unlockedLevels.appendText("\n" + levelNumber.text + " " + levelName.text);
171|   }
172|   var i:int = 0;
173|   // Remove everything from the world container that is not a border
174|   for (;worldContainer.numChildren > i;) {
175|     if (worldContainer.getChildAt(i) is Border) {
176|       i++;
177|     } else {
178|       worldContainer.removeChildAt(i);
179|     }
180|   }
181|   // Reset the laser emitters, player & exit
182|   for (;emitterContainer.numChildren > 0;) {
183|     emitterContainer.removeChildAt(0);
184|   }
185|   player.resetPlayer();
186|   player.x = Number(xml.world.player.xLoc) + worldContainer.x;
187|   player.y = Number(xml.world.player.yLoc) + worldContainer.y;
188|   player.visible = true;
189|   reach.x = player.x;
190|   reach.y = player.y;
191|   exit.x = Number(xml.world.exit.xLoc) + worldContainer.x;
192|   exit.y = Number(xml.world.exit.yLoc) + 2*worldContainer.y;
193|   exit.visible = true;
194|   emitters = [];
195|   i = 0;
196|   // Create laser emitters from xml
197|   for each (var p in xml.world.emitters.children()) {
198|     if(p.@exists == "true") {
199|       emitters[i] = new Emitter;
200|       emitters[i].rotatable = (p.rotatable == "true");
201|       emitters[i].colour = hex2dec(p.colour);
202|       emitters[i].activated = (p.activated == "true");
203|       if (!emitters[i].activated) {
204|         emitters[i].toggleMe(false);
205|       }
206|       emitters[i].x = Number(p.xLoc) + worldContainer.x;
207|       emitters[i].y = Number(p.yLoc) + worldContainer.y;
208|       emitters[i].rotation = int(p.rotation);
209|       emitterContainer.addChild(emitters[i]);
210|       registerHovers(emitters[i]);
211|     }
212|     i++
213|   }
214|
215|   // Create buttons from xml
216|   i = 0;
217|   for each (p in xml.world.buttons.children()) {
218|     if(p.@exists == "true") {
219|       buttons[i] = new GameButton;
220|       buttons[i].toggleMe(p.activated == "true");
221|       if (stripNumbers(p.linkedObject) == "emitter") {
222|         buttons[i].linkedObject = emitters[stripButNumbers(p.linkedObject)];
223|       } else if (stripNumbers(p.linkedObject) == "button") {
224|         buttons[i].linkedObject = buttons[stripButNumbers(p.linkedObject)];
225|       }
226|       buttons[i].pressed = buttons[i].linkedObject.activated;
227|       buttons[i].x = Number(p.xLoc);
228|       buttons[i].y = Number(p.yLoc);
229|       worldContainer.addChild(buttons[i]);
230|       registerHovers(buttons[i]);
231|     }
232|     i++
233|   }
234|
235|   // Create laser sensors from xml
236|   sensors = [];
237|   i = 0;
238|   for each (p in xml.world.sensors.children()) {
```

```
239|     if(p.@exists == "true") {
240|       sensors[i] = new Sensor;
241|       sensors[i].activated = false;
242|       sensors[i].alpha = 0.2
243|       sensors[i].colour = hex2dec(p.colour);
244|       if (stripNumbers(p.linkedObject) == "emitter") {
245|         sensors[i].linkedObject = emitters[stripButNumbers(p.linkedObject)];
246|       } else if (stripNumbers(p.linkedObject) == "button") {
247|         sensors[i].linkedObject = buttons[stripButNumbers(p.linkedObject)];
248|       }
249|       sensors[i].x = Number(p.xLoc);
250|       sensors[i].y = Number(p.yLoc);
251|       worldContainer.addChild(sensors[i]);
252|     }
253|     i++
254|   }
255|
256|   // Create mirrors from xml
257|   i = 0;
258|   for each (p in xml.world.mirrors.children()) {
259|     if(p.@exists == "true") {
260|       var mc = new Mirror;
261|       mc.programmable = p.programmable == "true";
262|       mc.rotatable = p.rotatable == "true";
263|       mc.nonDefault = p.nonDefault == "true";
264|       mc.autoRotSpeed = Number(p.autoRotSpeed);
265|       mc.x = Number(p.xLoc);
266|       mc.y = Number(p.yLoc);
267|       mc.rotation = int(p.rotation);
268|       worldContainer.addChild(mc);
269|       registerHovers(mc);
270|     }
271|     i++
272|   }
273|
274|   // Create coloured glass from xml to refract laser beams
275|   i = 0;
276|   for each (p in xml.world.colouredGlasses.children()) {
277|     if(p.@exists == "true") {
278|       mc = new ColouredGlass;
279|       mc.rotatable = p.rotatable == "true";
280|       mc.nonDefault = p.nonDefault == "true";
281|       mc.colour = hex2dec(p.colour);
282|       mc.x = Number(p.xLoc);
283|       mc.y = Number(p.yLoc);
284|       mc.rotation = int(p.rotation);
285|       worldContainer.addChild(mc);
286|       registerHovers(mc);
287|     }
288|     i++
289|   }
290|
291|   // Create prisms from xml to refract laser beams
292|   i = 0;
293|   for each (p in xml.world.prisms.children()) {
294|     if(p.@exists == "true") {
295|       mc = new Prism;
296|       mc.sides = p.sides;
297|       mc.rotatable = p.rotatable == "true";
298|       mc.nonDefault = p.nonDefault == "true";
299|       mc.colours = [];
300|       for (var j:int = 0; j < p.colours.length() && j < int(p.sides); j++) {
301|         mc.colours[j] = hex2dec(p.colours[j]);
302|       }
303|       mc.x = Number(p.xLoc);
304|       mc.y = Number(p.yLoc);
305|       mc.rotation = int(p.rotation);
306|       worldContainer.addChild(mc);
307|       registerHovers(mc);
308|     }
309|     i++
310|   }
311|
312|   // Create shutter doors from xml, that open and close periodically
313|   i = 0;
314|   for each (p in xml.world.shutters.children()) {
315|     if(p.@exists == "true") {
316|       mc = new Shutter;
317|       mc.programmable = p.programmable == "true";
318|       mc.setOffset(p.offsetTime, p.frequency);
319|       //mc.automatedTimer.delay = int(p.frequency);
320|       mc.x = Number(p.xLoc);
321|       mc.y = Number(p.yLoc);
322|       worldContainer.addChild(mc);
```

```
323|       registerHovers(mc);
324|     }
325|     i++
326|   }
327|
328|   // Create walls from xml
329|   i = 0;
330|   for each (p in xml.world.walls.children()) {
331|     if(p.@exists == "true") {
332|       mc = new Wall;
333|       mc.x = Number(p.xLoc);
334|       mc.y = Number(p.yLoc);
335|       mc.width = int(p.width);
336|       mc.height = int(p.height);
337|       worldContainer.addChild(mc);
338|     }
339|     i++
340|   }
341|
342|   // Create objects from xml for the inventory list the player can place
343|   i = 0;
344|   pickList.pickList = [];
345|   for (;pickList.numChildren > i;) {
346|     if (pickList.getChildAt(i) is PickListFilling) {
347|       i++;
348|     } else {
349|       pickList.removeChildAt(i);
350|     }
351|   }
352|   i = 0;
353|   for each (p in xml.pickList.children()) {
354|     var xOffset:int = Math.floor(i/3)*40 + 15;
355|     var yOffset:int = (i%3)*20 + 15;
356|     if (p.@type == "Mirror") {
357|       mc = new Mirror;
358|       mc.programmable = p.programmable == "true";
359|       mc.rotatable = p.rotatable == "true";
360|       mc.nonDefault = p.nonDefault == "true";
361|       mc.autoRotSpeed = Number(p.autoRotSpeed);
362|       mc.x = xOffset;
363|       mc.y = yOffset;
364|       mc.rotation = int(p.rotation);
365|     } else if (p.@type == "ColouredGlass") {
366|       mc = new ColouredGlass;
367|       mc.rotatable = p.rotatable == "true";
368|       mc.nonDefault = p.nonDefault == "true";
369|       mc.colour = hex2dec(p.colour);
370|       mc.x = xOffset;
371|       mc.y = yOffset;
372|       mc.rotation = int(p.rotation);
373|     } else if (p.@type == "Prism") {
374|       mc = new Prism;
375|       mc.nonDefault = p.nonDefault == "true";
376|       mc.sides = p.sides;
377|       mc.rotatable = p.rotatable == "true";
378|       mc.colours = [];
379|       for (j = 0; j < p.colours.length() && j < int(p.sides); j++) {
380|         mc.colours[j] = hex2dec(p.colours[j]);
381|       }
382|       mc.rotation = int(p.rotation);
383|       mc.x = xOffset;
384|       mc.y = yOffset;
385|     }
386|     mc.scaleX = 0.75;
387|     mc.scaleY = 0.75;
388|     pickList.pickList[i] = mc;
389|     pickList.pickList[i].addEventListener(MouseEvent.CLICK, pickList.pickFromList)
390|     pickList.addChild(mc);
391|     var indexObject = new Amount
392|     indexObject.setText(String(p.amount));
393|     mc.addChildAt(indexObject, 0)
394|     i++
395|   }
396|   // Set level and hint texts from xml
397|   if(xml.general.tip.length() > 0) {
398|     tips.text = stripDoubleLinebreaks(xml.general.tip[0]);
399|   } else {
400|   }
401|   if(xml.general.tip.length() > 1) {
402|     tip1.visible = true;
403|     tip2.visible = true;
404|     tip1.addEventListener(MouseEvent.CLICK, showTip1);
405|     tip2.addEventListener(MouseEvent.CLICK, showTip2);
406|   } else {
```

```
407|     tip1.visible = false;
408|     tip2.visible = false;
409|   }
410|   if(xml.general.tip.length() > 2) {
411|     tip3.visible = true;
412|     tip3.addEventListener(MouseEvent.CLICK, showTip3);
413|   } else {
414|     tip3.visible = false;
415|   }
416|   if(xml.general.tip.length() > 3) {
417|     tip4.visible = true;
418|     tip4.addEventListener(MouseEvent.CLICK, showTip4);
419|   } else {
420|     tip4.visible = false;
421|   }
422|   if(xml.general.tip.length() > 4) {
423|     tip5.visible = true;
424|     tip5.addEventListener(MouseEvent.CLICK, showTip5);
425|   } else {
426|     tip5.visible = false;
427|   }
428|   if(xml.general.tip.length() > 5) {
429|     tip6.visible = true;
430|     tip6.addEventListener(MouseEvent.CLICK, showTip6);
431|   } else {
432|     tip6.visible = false;
433|   }
434|   if(xml.general.tip.length() > 6) {
435|     tip7.visible = true;
436|     tip7.addEventListener(MouseEvent.CLICK, showTip7);
437|   } else {
438|     tip7.visible = false;
439|   }
440|   addEventListener(Event.ENTER_FRAME, everyFrame)
441| }
442|
443| //[...]
444|
445| /*
446|  * This one initiates the movement of the player's avatar on stage.
447|  */
448| function startPlayerMovement (e:MouseEvent):void {
449|   if(bGround.hitTestPoint(mouseX, mouseY) && !(ringMenu.visible && ringMenu.hitTestPoint(mouseX,
        mouseY))) {
450|     stage.addEventListener(Event.ENTER_FRAME, updatePlayerPosition)
451|     stage.addEventListener(MouseEvent.MOUSE_UP, unregisterPlayerMovementEvents)
452|   }
453| }
454|
455| //[...]
456|
457| /*
458|  * This manages the actual movement action.
459|  */
460| function updatePlayerPosition (e:Event):void {
461|   // calculate the distance between the mouse and the avatar.
462|   var dy:Number = player.y - mouseY;
463|   var dx:Number = player.x - mouseX;
464|   // verification: Is the player moving for a noticable distance??
465|   if ((Math.abs(dx) > 10 || Math.abs(dy) > 10) && player.alive) {
466|     // rotate - no matter if you hit something or not.
467|     player.rotation = Math.atan2(dy, dx)*180/Math.PI - 90
468|     // but only move around if you don't hit anything on your way.
469|     if (!playerCollusion(dx/-20, dy/-20)) {
470|       player.y -= (dy)/20
471|       player.x -= (dx)/20
472|       reach.x = player.x;
473|       reach.y = player.y;
474|     }
475|     // animate the player while walking/rotating
476|     player.walk(true);
477|
478|   } else { // stop animation if not moving at all.
479|     player.walk(false);
480|   }
481| }
482|
483| //[...]
484|
485| /*
486|  * Find out if the player WILL hit anything when moving any further.
487|  * Returns true if it will hit anything.
488|  */
489| function playerCollusion(offsetX:Number, offsetY:Number):Boolean {
```

```
490|   var incX:Number = 0;
491|   // The dummy simulates the player moving.
492|   var hitBox:MovieClip = new Player;
493|   hitBox.rotation = player.rotation;
494|   hitBox.x = player.x;
495|   hitBox.y = player.y;
496|   hitBox.alpha = 0.2
497|   // make the hitbox slightly smaller than the player itself.
498|   hitBox.scaleX = 0.75
499|   hitBox.scaleY = 0.75
500|   stage.addChild(hitBox)
501|   // if the dummy is stuck in an obstacle on creation it is bumped forward.
502|   for (var i:int = 0; i < worldContainer.numChildren; i++){
503|     if (hitBox.hitTestObject(worldContainer.getChildAt(i))) {
504|       hitBox.x += offsetX/5
505|       hitBox.y += offsetY/5
506|       incX += offsetX/5
507|     }
508|   }
509|   var hit:Boolean = false
510|   // otherwise it will step forward in small steps.
511|   while (Math.abs(incX) <= Math.abs(offsetX)){
512|     for (i = 0; i < worldContainer.numChildren; i++){
513|       var child = worldContainer.getChildAt(i);
514|       if (hitBox.hitTestObject(child)) {
515|
516|         if (child is Mirror) { // Mirrors should not be judged by their bounding box
517|           for (var j:int = -1/2*MovieClip(child).getChildAt(0).height; j < 1/2*MovieClip(child).
                    getChildAt(0).height; j += 1/10*MovieClip(child).getChildAt(0).height) {
518|             if(hitBox.hitTestPoint(child.localToGlobal(new Point(0, j) ).x, child.localToGlobal(new
                      Point(0, j) ).y) ) {
519|               if (!hit) {
520|                 hit = true;
521|               }
522|             }
523|           }
524|         } else if (child is ColouredGlass) { // ColouredGlass should not be judged by its bounding box
525|           for (j = -1/2*MovieClip(child).getChildAt(0).width; j < 1/2*MovieClip(child).getChildAt(0).
                    width; j += 1/10*MovieClip(child).getChildAt(0).width) {
526|             if(hitBox.hitTestPoint(child.localToGlobal(new Point(j, 0) ).x, child.localToGlobal(new
                      Point(j, 0) ).y) ) {
527|               if (!hit) {
528|                 hit = true;
529|               }
530|             }
531|           }
532|         } else if (child is Wall || child is Sensor || child is GameButton || child is Border || child
                is Prism) { // Sensor & Box bounding circle instead of bounding box?
533|               if (!hit) {
534|                 hit = true;
535|               }
536|         } else if (child is Shutter) {
537|           if (Shutter(child).activated) {
538|           } else {
539|               if (!hit) {
540|                 hit = true;
541|               }
542|           }
543|         }
544|       }
545|     }
546|     hitBox.x += offsetX/5
547|     hitBox.y += offsetY/5
548|     incX += Math.abs(offsetX/5)
549|   }
550|   stage.removeChild(hitBox)
551|   return hit; // if you have not hit anything by now you won't later :p
552| }
553|
554| //[...]
555|
556| /*
557|  * When clicking to the world container
558|  */
559| function clickWorld(e:MouseEvent):void {
560|   if (ringMenuTimer.running) {
561|     ringMenuTimer.stop();
562|     var dblClickedObject:DisplayObject = underCursor();
563|     // Only show the ring menu if the object is close to the player and no wall
564|     if (!(dblClickedObject is Wall) && distanceToPlayer(dblClickedObject) < 50) {
565|       // Set individual values as editable or not.
566|       if (!Object(dblClickedObject).rotatable) {
567|         ringMenu.rotateLeft.alpha = 0.2;
568|         ringMenu.rotateRight.alpha = 0.2;
569|       } else {
```

```
570|         ringMenu.rotateLeft.alpha = 1;
571|         ringMenu.rotateRight.alpha = 1;
572|       }
573|       if (!Object(dblClickedObject).programmable) {
574|         ringMenu.plus.alpha = 0.2;
575|         ringMenu.minus.alpha = 0.2;
576|       } else {
577|         ringMenu.plus.alpha = 1;
578|         ringMenu.minus.alpha = 1;
579|       }
580|       if (Object(dblClickedObject).pickable) {
581|         ringMenu.pick.visible = true;
582|       } else {
583|         ringMenu.pick.visible = false;
584|       }
585|       ringMenu.x = dblClickedObject.parent.localToGlobal(new Point(dblClickedObject.x,
               dblClickedObject.y)).x;
586|       ringMenu.y = dblClickedObject.parent.localToGlobal(new Point(dblClickedObject.x,
               dblClickedObject.y)).y;
587|       ringMenu.targetObject = dblClickedObject;
588|       ringMenu.visible = true;
589|       playRingOpenSound();
590|     }
591|   } else if(player.alive) {
592|     ringMenuTimer.start();
593|     if (/*stage.contains(ringMenu)*/ringMenu.visible) {
594|       ringMenu.visible = false;
595|       playRingCloseSound();
596|     }
597|   }
598| }
599|
600| /*
601|  * When clicking on a laser emitter
602|  */
603| function clickEmitter(e:MouseEvent):void {
604|   if (ringMenuTimer.running) {
605|     ringMenuTimer.stop();
606|     ringMenu.pick.visible = false;
607|     var dblClickedObject:DisplayObject = underCursor();
608|     // Only show the ring menu if the emitter is close to the player
609|     if (distanceToPlayer(dblClickedObject) < 50) {
610|       // Set individual values as editable or not.
611|       if (!Emitter(dblClickedObject).rotatable) {
612|         ringMenu.rotateLeft.alpha = 0.2;
613|         ringMenu.rotateRight.alpha = 0.2;
614|       } else {
615|         ringMenu.rotateLeft.alpha = 1;
616|         ringMenu.rotateRight.alpha = 1;
617|       }
618|       if (!Emitter(dblClickedObject).programmable) {
619|         ringMenu.plus.alpha = 0.2;
620|         ringMenu.minus.alpha = 0.2;
621|       } else {
622|         ringMenu.plus.alpha = 1;
623|         ringMenu.minus.alpha = 1;
624|       }
625|       // Translate container-coordinates to stage coordinates
626|       ringMenu.x = dblClickedObject.parent.localToGlobal(new Point(dblClickedObject.x,
               dblClickedObject.y)).x;
627|       ringMenu.y = dblClickedObject.parent.localToGlobal(new Point(dblClickedObject.x,
               dblClickedObject.y)).y;
628|       ringMenu.targetObject = dblClickedObject;
629|       ringMenu.visible = true;
630|       playRingOpenSound();
631|     }
632|   } else if(player.alive) {
633|     ringMenuTimer.start();
634|     if (/*stage.contains(ringMenu)*/ringMenu.visible) {
635|       ringMenu.visible = false;
636|       playRingCloseSound();
637|     }
638|   }
639| }
640|
641| /*
642|  * Remove the ring men from the interface
643|  */
644| function removeRingMenu (e:MouseEvent):void {
645|   if (ringMenu.visible && !ringMenu.hitTestPoint(mouseX, mouseY)) {
646|     playRingCloseSound();
647|     ringMenuTimer.stop();
648|     ringMenu.visible = false;
649|   } else if (e.currentTarget == ringMenu.pick) {
```

```
650|     playRingCloseSound();
651|     ringMenuTimer.stop();
652|     ringMenu.visible = false;
653|   }
654| }
655|
656| /*
657|  * Changing a value of a world object
658|  */
659| function changeValue(mc:MovieClip, to:String):void {
660|   if (mc.programmable) {
661|     if (mc is Shutter) {
662|       if (to == "up" && Shutter(mc).automatedTimer.delay+250 <= 5000) {
663|         playProgUpSound();
664|         var by:Number = 250;
665|       } else if (to == "down" && Shutter(mc).automatedTimer.delay-250 > 0){
666|         playProgDownSound();
667|         by = -250
668|       } else {
669|         by = 0;
670|       }
671|       Shutter(mc).automatedTimer.delay += by;
672|     } else if (mc is Mirror) {
673|       if (Mirror(mc).programmable) {
674|         if (to == "up") {
675|         playProgUpSound();
676|           /*if (Mirror(mc).autoRotSpeed == -0.5) {
677|             by = 1
678|           } else */{
679|             by = 0.5;
680|           }
681|         } else {
682|           playProgDownSound();
683|           /*if (Mirror(mc).autoRotSpeed == 0.5) {
684|             by = -1
685|           } else */{
686|             by = -0.5;
687|           }
688|         }
689|       }
690|       Mirror(mc).autoRotSpeed += by;
691|     } else if (mc is GameButton && GameButton(mc).activated) {
692|       if (to == "up"){
693|         playProgUpSound();
694|         GameButton(mc).pressed = true;
695|       } else {
696|         playProgDownSound();
697|         GameButton(mc).pressed = false;
698|       }
699|     }
700|   }
701| }
702|
703| //[...]
704|
705| /*
706|  * Starts a laser beam with the given properties and continues it until it hits anything.
707|  * grow the beam in length. Returns true if another increase is possible and false if not.
708|  * May be used for by tick growth.
709|  */
710| function advance(sourcePoint:Point, sourceRotation:int, myColour:int, emitter:Emitter, hitLast:
      DisplayObject):Boolean {
711|   // Introduce laser beam container
712|   var minLength:int = 5;
713|   var mc:Sprite = new Sprite();
714|   mc.x = sourcePoint.x
715|   mc.y = sourcePoint.y
716|   mc.rotation = sourceRotation
717|   laserBeams.addChild(mc)
718|   var beamEnd:Point = new Point (0,-minLength)
719|   emitter.beamLength += 4;
720|   // Set visual appearance of the laser beam
721|   mc.addChildAt(new Laser(), 0);
722|   mc.getChildAt(0).y = -minLength
723|   mc.getChildAt(0).height = minLength
724|   var newColorTransform:ColorTransform = mc.getChildAt(0).transform.colorTransform;
725|   newColorTransform.redOffset = (myColour)>>16;
726|   newColorTransform.greenOffset = (myColour%0x010000) >> 8;
727|   newColorTransform.blueOffset = myColour% 0x000100;
728|   mc.getChildAt(0).transform.colorTransform = newColorTransform;
729|   // Extend beam until it hits something - not elegant, but it works
730|   var globalBeamEnd:Point = mc.localToGlobal(beamEnd);
731|   while (emitter.beamLength < 2000 && !hitsObject(globalBeamEnd.x,globalBeamEnd.y, worldContainer) &&
        globalBeamEnd.x < 500 && globalBeamEnd.x > 0 && globalBeamEnd.y < 500 && globalBeamEnd.y > 0){
```

```
732|     emitters.beamLength ++
733|     beamEnd.y--
734|     if(player.alive && player.hitTestPoint(mc.localToGlobal(beamEnd).x, mc.localToGlobal(beamEnd).y,
          true)) {
735|       player.die();
736|     }
737|     globalBeamEnd = mc.localToGlobal(beamEnd);
738|   }
739|
740|   mc.getChildAt(0).y = beamEnd.y;
741|   mc.getChildAt(0).height = -beamEnd.y;
742|   // Find hit object and move the end of the beam to its edge
743|   if (hitsObject(globalBeamEnd.x, globalBeamEnd.y, worldContainer) && emitter.beamLength < 2000 &&
        hitObject != emitter) {
744|     var hitObject = whatIsHit(mc);
745|     if (mc.getChildAt(0).height == minLength) {
746|       while (whatIsHit(mc) == hitObject) {
747|         mc.getChildAt(0).y -= 1
748|         mc.getChildAt(0).height += 1
749|         emitters.beamLength += 1;
750|         beamEnd.y  -= 1
751|       }
752|       mc.getChildAt(0).y -= 1
753|       mc.getChildAt(0).height += 1
754|       emitters.beamLength += 1;
755|       beamEnd.y  -= 1
756|     }
757|     laserBeams.addChild(new LaserEnding)
758|     laserBeams.getChildAt(laserBeams.numChildren-1).rotation = mc.rotation
759|     laserBeams.getChildAt(laserBeams.numChildren-1).x = laserBeams.globalToLocal(mc.localToGlobal(
          beamEnd)).x;
760|     laserBeams.getChildAt(laserBeams.numChildren-1).y = laserBeams.globalToLocal(mc.localToGlobal(
          beamEnd)).y;
761|     laserBeams.addChild(new LaserEnding)
762|     laserBeams.getChildAt(laserBeams.numChildren-1).rotation = mc.rotation+180;
763|     laserBeams.getChildAt(laserBeams.numChildren-1).x = mc.x;
764|     laserBeams.getChildAt(laserBeams.numChildren-1).y = mc.y;
765|
766|     newColorTransform = laserBeams.getChildAt(laserBeams.numChildren-1).transform.colorTransform;
767|     newColorTransform.redOffset = (myColour)>>16;
768|     newColorTransform.greenOffset = (myColour%0x010000) >> 8;
769|     newColorTransform.blueOffset = myColour% 0x000100;
770|     laserBeams.getChildAt(laserBeams.numChildren-1).transform.colorTransform = newColorTransform;
771|     laserBeams.getChildAt(laserBeams.numChildren-2).transform.colorTransform = newColorTransform;
772|     // React depending on the type of object hit
773|     if (hitObject is Mirror) {
774|       reflect (mc, beamEnd, hitObject, myColour, emitter, hitLast);
775|     } else if (hitObject is Prism) {
776|       refract (mc, beamEnd, hitObject, myColour, emitter, hitLast);
777|     } else if (hitObject is Wall) {
778|     } else if (hitObject is ColouredGlass) {
779|       if (isRecolourAllowed(myColour, ColouredGlass(hitObject).colour)) {
780|         advance(mc.localToGlobal(beamEnd), mc.rotation, ColouredGlass(hitObject).colour, emitter,
              hitObject);
781|       } else {
782|         advance(mc.localToGlobal(beamEnd), mc.rotation, 0x000000, emitter, hitObject);
783|       }
784|     } else if (hitObject is Sensor) {
785|       // how to reset?
786|       if (myColour == hitObject.colour) {
787|         hitObject.toggleMe(true);
788|       }
789|     } else if (hitObject is Shutter) {
790|       if (hitObject.activated) {
791|         advance(mc.localToGlobal(beamEnd), mc.rotation, myColour, emitter, hitObject);
792|       }
793|     }
794|   }
795|   return true;
796| }
797|
798| /*
799|  * reflects a laser beam when it hit a mirror.
800|  * reflects the laser by from the MovieClip it hit.
801|  * Therefore it uses mcs rotation and creates a new Laser object.
802|  */
803| function reflect(mc:Sprite, beamEnd:Point, hitObject:DisplayObject, myColour:int, emitter:Emitter,
      hitLast:DisplayObject):void {
804|       var sourceRotation:int = mc.rotation;
805|       var sprite = new Sprite();
806|       sprite.y = mc.parent.globalToLocal(laserPos(mc)).y
807|       sprite.x = mc.parent.globalToLocal(laserPos(mc)).x
808|       var nativeSourceRot:int = -(sourceRotation - 180);
809|       var nativeHitRot:int = -(hitObject.rotation - 180);
810|       laserBeams.addChild(sprite)
```

```
811|       var newRotation:int = -sourceRotation + (2*hitObject.rotation);
812|       if (Math.abs(hitObject.globalToLocal(mc.localToGlobal(beamEnd)).y)< MovieClip(hitObject).
               getChildAt(0).height/2.05) {
813|         advance(mc.localToGlobal(beamEnd), newRotation, myColour, emitter, hitObject);
814|       }
815|
816| }
817|
818| /*
819|  * Refracts the Laser beam into mc.getAmount() number of beams, each mc.getAngle() degrees apart
820|  * from each other while mc.getColours() holds each individual new beams colourcode as hex int.
821|  * Uses recolour(int, int) to check if a refraction is allowed. mcs orientation does not alter
822|  * the Lasers orientation. What to do when the colour does not match the allowed ones?
823|  */
824| function refract(mc:Sprite, beamEnd:Point, hitObject:DisplayObject, myColour:int, emitter:Emitter,
       hitLast:DisplayObject):void {
825|       var beamEndCopy:Point = new Point(beamEnd.x, beamEnd.y-1);
826|       var sourceRotation:int = mc.rotation;
827|       var sprite:Sprite;
828|       var points:Array = []
829|       var angularSum:Number = (Prism(hitObject).sides-2)*180;
830|       // A prism has multiple sides and each creates an individual beam
831|       for (var i:int = 0; i < Prism(hitObject).sides-1; i++) {
832|         sprite = new Sprite();
833|         sprite.x = mc.localToGlobal(beamEndCopy).x;
834|         sprite.y = mc.localToGlobal(beamEndCopy).y;
835|         sprite.rotation = sourceRotation - 90 + (180/(Prism(hitObject).sides-2))*i;
836|         laserBeams.addChild(sprite);
837|         points[i] = new Point(0,0)
838|         while (hitsObject(sprite.localToGlobal(points[i]).x, sprite.localToGlobal(points[i]).y,
               hitObject)) {
839|           points[i].y--
840|         }
841|         if (hitLast != hitObject){
842|           if (isRecolourAllowed(myColour, Prism(hitObject).colours[i])){
843|             advance(sprite.localToGlobal(points[i]), sprite.rotation, Prism(hitObject).colours[i],
                     emitter, hitObject);
844|           } else {
845|             advance(sprite.localToGlobal(points[i]), sprite.rotation, 0x000000, emitter, hitObject);
846|           }
847|         } else {
848|           advance(sprite.localToGlobal(points[i]), sourceRotation, myColour, emitter, hitObject);
849|         }
850|       }
851| }
852|
853| //[...]
```

"./Marc A Modrow source code sample AS3.txt"