```
 1 let memoryRequests = [];
 2 let currentIndex = 0;
 3 let currentRequest = {}; // Request broken into format {'
   process': 'P1', 'page': '000011'}
 4 let pcbs = {}; // PCB in format {'P1': 1} with 1 being the
    table #
 5 let numberOfPcbTables = 0;
 6 let pageTables = {};
 7 let stats = {};
 8 let currentStep = "updateOrder";
 9 let freeFrameList = [15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0
   ];
10 let usedFrameList = [];
11 let currentFrame = 0;
12 let highlightedPageTable = 0;
13 let numberOfFaults = 0;
14 let numberOfRefs = 0;
15 let numberOfPages = 0;
16 let physicalMemory = ['','','','','','','','','','','','',
   '','','',''];
17 // Lists the processes in order from least used to most.
   With every new request,
18 // if it is in the table, remove it and append it to the
   end. If it is not in the table,
19 // append it to the end. The first process was used least
   recently and will be the victim
20 // if it is not already in the victims table.
21 let processesInOrder = [];
22 let victimProcess = '';
23 let victimPage = '';
24 let victimFrame = '';
25 let rawRequest = '';
26 let splitRequest = '';
27 let givenProcess = '';
28 let givenPage = '';
29
30 /**
31  * A setter for the memory requests pulled from the file.
32  * Needed to be in the model as 404 errors are only
   checked after
33  *
34  * @param requests The list of requests that were
   retrieved from the file
35  */
36 function setMemoryRequests(requests) {
```

```
37      memoryRequests = requests;
38  }
39
40  /**
41   * Moves the simulation one step forward.
42   * @returns {number} -1 if error, 0 if success, 1 if page
    fault
43   */
44  function moveOneStep() {
45      rawRequest = memoryRequests[currentIndex];
46      splitRequest = rawRequest.split(':');
47      givenProcess = splitRequest[0].trim();
48      givenPage = splitRequest[1].trim();
49
50      let processValid = validateAndUpdateProcess(
    givenProcess);
51      if (!processValid) {
52          return -1;
53      }
54      let pageValid = validateAndUpdateBinaryPage(givenPage)
    ;
55      if (!pageValid) {
56          return -1;
57      }
58
59      let decimalPage = parseInt(givenPage, 2);
60
61      // Only moving forward in memory table do nothing for
    this step
62      if (currentStep === "incrementMemory") {
63          currentStep = 'updateOrder';
64          return 0;
65      }
66
67      // Update the order of processes with this as the most
     recently used
68      if (currentStep === 'updateOrder') {
69          let formattedEntry = "Process: " + givenProcess +
    " Page: " + decimalPage;
70          if (processesInOrder.includes(formattedEntry)) {
71              let index = processesInOrder.indexOf(
    formattedEntry);
72              processesInOrder.splice(index, 1);
73          }
74          processesInOrder.push(formattedEntry);
```

```
 75         updateStatistics(0,1,0);
 76         numberOfRefs++;
 77         ViewModel.updateOrderedProcesses(processesInOrder
    );
 78         ViewModel.colorCell("currentOrder", 0, 'yellow',
    processesInOrder.length-1);
 79         currentStep = 'checkPCB';
 80         return 0;
 81     }
 82
 83     // Create a PCB if one does not exist already
 84     if (currentStep === 'checkPCB' && !pcbs.
    hasOwnProperty(givenProcess)){
 85         ViewModel.colorCell("currentOrder", 0, 'clear',
    processesInOrder.length-1);
 86         pcbs[givenProcess] = numberOfPcbTables;
 87         ViewModel.addPCB(givenProcess);
 88         numberOfPcbTables++;
 89         currentStep = 'checkLogicalPage';
 90         return 0;
 91     }
 92
 93     if (currentStep === 'checkPCB' && pcbs.hasOwnProperty
    (givenProcess)) {
 94         ViewModel.colorCell("currentOrder", 0, 'clear',
    processesInOrder.length-1);
 95         ViewModel.highlightPCB(givenProcess);
 96         currentStep = 'checkLogicalPage';
 97         return 0;
 98     }
 99
100     // Create the page table if one does not exist
    already
101     if (currentStep === 'checkLogicalPage' && !pageTables
    .hasOwnProperty(givenProcess)) {
102         // Clear yellow from pcb
103         ViewModel.colorCell('pcb', 0, 'clear', 0);
104
105         pageTables[givenProcess] = [{"va": decimalPage, "
    pa": ''}];
106         updateStatistics(1, 0, 0);
107         numberOfPages++;
108
109         let keys = Object.keys(pageTables);
110         for (let i=0; i<keys.length; i++) {
```

```
111             let key = keys[i];
112             if (key === givenProcess) {
113                 ViewModel.updatePageTable(givenProcess,
    pageTables[key]);
114                 ViewModel.updateStats(stats);
115                 highlightedPageTable = Object.keys(
    pageTables).indexOf(key);
116                 ViewModel.colorCell('pageTableLogical',
    highlightedPageTable, 'yellow', decimalPage);
117             }
118         }
119         numberOfFaults++;
120         updateStatistics(0,0,1);
121         currentStep = 'checkFreeFrames';
122         return 1;
123     }
124 
125     // Add to the page table if one exists already
126     if (currentStep === 'checkLogicalPage' && pageTables.
    hasOwnProperty(givenProcess)) {
127         // Clear yellow from pcb
128         ViewModel.colorCell('pcb', 0, 'clear', 0);
129 
130         //Check if the page is already mapped
131         if (pageTables[givenProcess][decimalPage] ===
    undefined) {
132             pageTables[givenProcess].push({"va":
    decimalPage, "pa": ''});
133             updateStatistics(1, 0, 0);
134             numberOfPages++;
135             let keys = Object.keys(pageTables);
136             for (let i = 0; i < keys.length; i++) {
137                 let key = keys[i];
138                 if (key === givenProcess) {
139                     ViewModel.updatePageTable(
    givenProcess, pageTables[key]);
140                     highlightedPageTable = Object.keys(
    pageTables).indexOf(key);
141                     ViewModel.colorCell('pageTableLogical
    ', highlightedPageTable, 'yellow', decimalPage);
142                 }
143             }
144             numberOfFaults++;
145             updateStatistics(0,0,1);
146             currentStep = 'checkFreeFrames';
```

```
147             return 1;
148         }
149         else if (pageTables[givenProcess][decimalPage]['
    pa'] === '') {
150             let keys = Object.keys(pageTables);
151             for (let i = 0; i < keys.length; i++) {
152                 let key = keys[i];
153                 if (key === givenProcess) {
154                     ViewModel.updatePageTable(
    givenProcess, pageTables[key]);
155                     highlightedPageTable = Object.keys(
    pageTables).indexOf(key);
156                     ViewModel.colorCell('pageTableLogical
    ', highlightedPageTable, 'yellow', decimalPage);
157                 }
158             }
159             numberOfFaults++;
160             updateStatistics(0,0,1);
161             currentStep = 'checkFreeFrames';
162             return 1;
163         }
164         else {
165             // Page is already mapped highlight it and
    move on
166             let keys = Object.keys(pageTables);
167             for (let i=0; i<keys.length; i++) {
168                 let key = keys[i];
169                 if (key === givenProcess) {
170                     highlightedPageTable = Object.keys(
    pageTables).indexOf(key);
171                     ViewModel.colorCell('pageTableLogical
    ', highlightedPageTable, 'yellow', decimalPage);
172                     currentStep = 'showPhysicalFrame';
173                     return 0;
174                 }
175             }
176         }
177     }
178
179     // Get the physical frame from the free frame list or
     page fault
180     if (currentStep === 'checkFreeFrames') {
181         // Clear yellow from logical page
182         ViewModel.colorCell('pageTableLogical',
    highlightedPageTable, 'clear', decimalPage);
```

```
183
184         if (freeFrameList.length > 0) {
185             // Get frame from frame list
186             currentFrame = freeFrameList.pop();
187             pageTables[givenProcess][decimalPage]['pa'] =
     currentFrame;
188             usedFrameList.push(currentFrame);
189             ViewModel.updateFreeFrameList(usedFrameList,
    currentFrame);
190             currentStep = '
    showPhysicalMemoryFrameFromFreeFrame';
191             return 0;
192
193         } else {
194             ViewModel.colorCell("currentOrder", 0, '
    yellow', 0);
195             currentStep = 'locateVictimsPCB';
196             return 0
197         }
198     }
199
200     // A victim needs to be removed
201     if (currentStep === 'locateVictimsPCB') {
202         let victim = processesInOrder[0];
203         victimProcess = victim.substring(9,11);
204         victimPage = victim.substring(18);
205         ViewModel.colorCell("currentOrder", 0, 'clear', 0
    );
206         ViewModel.highlightPCB(victimProcess);
207         processesInOrder.shift();
208         ViewModel.updateOrderedProcesses(processesInOrder
    );
209         currentStep = 'locateVictimsPageTableEntryLogical
    ';
210         return 0;
211     }
212
213     // Highlight the victim's page table entry
214     if (currentStep === '
    locateVictimsPageTableEntryLogical') {
215         ViewModel.colorCell('pcb', 0, 'clear', 0);
216         let keys = Object.keys(pageTables);
217         for (let i=0; i<keys.length; i++) {
218             let key = keys[i];
219             if (key === victimProcess) {
```

```
220                 highlightedPageTable = Object.keys(
    pageTables).indexOf(key);
221                 ViewModel.colorCell('pageTableLogical',
    highlightedPageTable, 'yellow', victimPage);
222             }
223         }
224         currentStep = '
    locateVictimsPageTableEntryPhysical';
225         return 0;
226     }
227 
228     // Delete the victim's page table entry
229     if (currentStep === '
    locateVictimsPageTableEntryPhysical') {
230         let keys = Object.keys(pageTables);
231         for (let i=0; i<keys.length; i++) {
232             let key = keys[i];
233             if (key === victimProcess) {
234                 highlightedPageTable = Object.keys(
    pageTables).indexOf(key);
235                 ViewModel.colorCell('pageTableLogical',
    highlightedPageTable, 'clear', victimPage);
236                 ViewModel.colorCell('pageTablePhysical',
    highlightedPageTable, 'yellow', victimPage);
237             }
238         }
239         currentStep = 'locateVictimMemoryFrame';
240         return 0;
241     }
242 
243     // Locate the frame number in memory for deletion
244     if (currentStep === 'locateVictimMemoryFrame') {
245         ViewModel.showPhysMemory(true);
246         victimFrame = pageTables[victimProcess][
    victimPage]['pa'];
247         ViewModel.colorCell('pageTablePhysical',
    highlightedPageTable, 'clear', victimPage);
248         pageTables[victimProcess][victimPage]["pa"]='';
249         ViewModel.updatePageTable(victimProcess,
    pageTables[victimProcess]);
250 
251         ViewModel.colorCell('physicalMemoryFrameNumber',
    0, 'yellow', victimFrame);
252         currentStep = 'showVictimPhysicalMemory';
253         return 0;
```

```
254     }
255
256     // Highlight the memory to be cleared
257     if (currentStep === 'showVictimPhysicalMemory') {
258         physicalMemory[victimFrame] = "";
259         ViewModel.updatePhysicalMemory(physicalMemory);
260         ViewModel.colorCell('physicalMemoryFrameNumber',
    0, 'clear', victimFrame);
261         ViewModel.colorCell('physicalMemoryInUseBy', 0, '
    yellow', victimFrame);
262         currentStep = 'updateVictimMemoryWithNewData';
263         return 0;
264     }
265
266     // Place the new data in the cleared frame
267     if (currentStep === 'updateVictimMemoryWithNewData')
    {
268         physicalMemory[victimFrame] = "Process: " +
    givenProcess + " Page: " + decimalPage;
269         ViewModel.updatePhysicalMemory(physicalMemory);
270         ViewModel.colorCell('physicalMemoryInUseBy', 0, '
    yellow', victimFrame);
271         currentFrame = victimFrame;
272         currentStep = 'updatePageTableWithVictimValue';
273         return 0;
274     }
275
276     // Puts the obtained frame in the new page table
277     if (currentStep === 'updatePageTableWithVictimValue')
     {
278         ViewModel.colorCell('physicalMemoryFrameNumber',
    0, 'clear', victimFrame);
279         highlightedPageTable = Object.keys(pageTables).
    indexOf(givenProcess);
280         ViewModel.colorCell('pageTablePhysical',
    highlightedPageTable, 'yellow', decimalPage);
281         pageTables[givenProcess][decimalPage]['pa'] =
    currentFrame;
282         ViewModel.updatePageTable(givenProcess,
    pageTables[givenProcess]);
283         highlightedPageTable = Object.keys(pageTables).
    indexOf(givenProcess);
284         ViewModel.colorCell('pageTablePhysical',
    highlightedPageTable, 'yellow', decimalPage);
285         currentStep = 'stepComplete';
```

```
286         return 0;
287     }
288 
289     // Frame was already mapped so highlight it
290     if (currentStep === 'showPhysicalFrame') {
291         // Clear yellow from logical page
292         ViewModel.colorCell('pageTableLogical',
    highlightedPageTable, 'clear', decimalPage);
293         //Set physical to yellow
294         ViewModel.colorCell('pageTablePhysical',
    highlightedPageTable, 'yellow', decimalPage);
295         currentStep = '
    showPhysicalMemoryFrameFromPageTable';
296         return 0
297     }
298 
299     // Move on to highlighting the frame number found
300     if (currentStep === '
    showPhysicalMemoryFrameFromFreeFrame') {
301         ViewModel.updatePhysicalMemory(physicalMemory);
302         ViewModel.showPhysMemory(true);
303         ViewModel.colorCell("freeFrameList", 0, "clear",
    currentFrame);
304         ViewModel.colorCell('physicalMemoryFrameNumber',
    0, 'yellow', currentFrame);
305         currentStep = 'fillInPhysicalInUseByReturnFrame';
306         return 0;
307     }
308 
309     // Move on to highlighting the frame number found
310     if (currentStep === '
    showPhysicalMemoryFrameFromPageTable') {
311         ViewModel.showPhysMemory(true);
312         currentFrame = pageTables[givenProcess][
    decimalPage]['pa'];
313         ViewModel.colorCell('pageTablePhysical',
    highlightedPageTable, 'clear', decimalPage);
314         ViewModel.colorCell('physicalMemoryFrameNumber',
    0, 'yellow', currentFrame);
315         currentStep = '
    fillInPhysicalInUseByReturnComplete';
316         return 0;
317     }
318 
319     // Move on to highlighting the in use by physical
```

```
319 memory cell and go back to free frames
320     if (currentStep === 'fillInPhysicalInUseByReturnFrame
    ') {
321         physicalMemory[currentFrame] = "Process: " +
    givenProcess + " Page: " + decimalPage;
322         ViewModel.updatePhysicalMemory(physicalMemory);
323         if (currentFrame !== 15) {
324             ViewModel.colorCell('
    physicalMemoryFrameNumber', 0, 'clear', currentFrame+1);
325         }
326         ViewModel.colorCell('physicalMemoryInUseBy', 0, '
    yellow', currentFrame);
327         currentStep = 'showFreeFrameListAgain';
328         return 0;
329     }
330
331     // Move on to highlighting the in use by physical
    memory cell and finish
332     if (currentStep === '
    fillInPhysicalInUseByReturnComplete') {
333         physicalMemory[currentFrame] = "Process: " +
    givenProcess + " Page: " + decimalPage;
334         ViewModel.updatePhysicalMemory(physicalMemory);
335         ViewModel.colorCell('physicalMemoryFrameNumber',
    0, 'clear', currentFrame);
336         ViewModel.colorCell('physicalMemoryInUseBy', 0, '
    yellow', currentFrame);
337         currentStep = 'stepComplete';
338         return 0;
339     }
340
341     // Highlight free frame list a second time
342     if (currentStep === 'showFreeFrameListAgain') {
343         ViewModel.colorCell("freeFrameList", 0, "yellow",
     currentFrame);
344         ViewModel.colorCell('physicalMemoryInUseBy', 0, '
    clear', currentFrame);
345         currentStep = 'updatePhysicalFrame';
346         return 0;
347     }
348
349     // Update the page table with the physical frame
350     if (currentStep === 'updatePhysicalFrame') {
351         ViewModel.colorCell("freeFrameList", 0, "yellow",
     currentFrame);
```

```
352         ViewModel.colorCell('physicalMemoryFrameNumber',
    0, 'clear', currentFrame);
353         let keys = Object.keys(pageTables);
354         for (let i=0; i<keys.length; i++) {
355             let key = keys[i];
356             if (givenProcess === key) {
357                 ViewModel.updatePageTable(givenProcess,
    pageTables[key]);
358                 highlightedPageTable = Object.keys(
    pageTables).indexOf(key);
359                 ViewModel.colorCell('pageTablePhysical',
    highlightedPageTable, 'yellow', decimalPage);
360                 ViewModel.colorCell("freeFrameList", 0, "
    gray", currentFrame);
361             }
362         }
363         currentStep = 'stepComplete';
364         return 0;
365     }
366
367     // There is nothing left to do for this memory
    request so move on
368     if (currentStep === 'stepComplete') {
369         // Clear yellow from logical page
370         ViewModel.colorCell('pageTablePhysical',
    highlightedPageTable, 'clear', decimalPage);
371         ViewModel.colorCell('physicalMemoryInUseBy', 0, '
    clear', currentFrame);
372         currentIndex++;
373         currentStep = 'incrementMemory';
374         ViewModel.updateIndex(currentIndex);
375         return 0;
376     }
377
378     return -1;
379 }
380
381 /**
382  * Validates a process name.
383  * @param process the process name to be validated
384  * @returns {boolean} true on success false on error
385  */
386 function validateAndUpdateProcess(process) {
387     let processPattern = new RegExp("P\\d");
388
```

```
389     if (processPattern.test(process)) {
390         currentRequest['process'] = process;
391         return true;
392     } else {
393         alertError("Bad File. Invalid process name: " +
    process);
394         return false;
395     }
396 }
397 
398 /**
399  * Validates a page passed in binary
400  * @param page The binary page to be validated
401  * @returns {boolean} true on success false on error
402  */
403 function validateAndUpdateBinaryPage(page){
404     let pagePattern = new RegExp("(0|1){6}");
405 
406     if (page.length !== 6) {
407         alertError("Bad File. Invalid length of page
    number: " + page);
408         return false;
409     }
410     else if (pagePattern.test(page)) {
411         currentRequest['page'] = page;
412         return true;
413     } else {
414         alertError("Bad File. Invalid page number: " +
    page);
415         return false;
416     }
417 }
418 
419 /**
420  * Updates the statistics for the overall simulation.
421  * @param size amount to increase the size by
422  * @param references amount to increase the memory
    references by
423  * @param faults increase the page faults by
424  */
425 function updateStatistics(size, references, faults) {
426     if (!stats.hasOwnProperty(givenProcess)) {
427         stats[givenProcess] = [{"size": '0', "references"
    : '0', "faults": '0'}];
428     }
```

```
429     if (stats[givenProcess][0] === undefined) {
430         stats[givenProcess].push({"size": '0', "
    references": '0', "faults": '0'});
431     }
432     let newSize =  parseInt(stats[givenProcess][0]['size'
    ]) + size;
433     let newRefs =  parseInt(stats[givenProcess][0]['
    references']) + references;
434     let newFaults =  parseInt(stats[givenProcess][0]['
    faults']) + faults;
435     stats[givenProcess] = [{"size": newSize.toString(), "
    references": newRefs.toString(), "faults": newFaults.
    toString()}];
436
437     let keys = Object.keys(stats);
438     for (let i=0; i<keys.length; i++) {
439         let key = keys[i];
440         if (key === givenProcess) {
441             ViewModel.updateStats(givenProcess, stats[key
    ]);
442         }
443     }
444
445     ViewModel.updatePageFaults(numberOfFaults);
446     ViewModel.updateMemRefs(numberOfRefs);
447     ViewModel.updatePagesNumber(numberOfPages);
448 }
449
450 /**
451  * Clears all fields in the controller for a restart of
    the simulation.
452  */
453 function clearController() {
454     memoryRequests = [];
455     currentIndex = 0;
456     currentRequest = {};
457     pcbs = {};
458     numberOfPcbTables = 0;
459     pageTables = {};
460     stats = {};
461     currentStep = "updateOrder";
462     freeFrameList = [15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,
    0];
463     usedFrameList = [];
464     currentFrame = 0;
```

```
465     highlightedPageTable = 0;
466     physicalMemory = ['','','','','','','','','','','','','','','','','','','',''];
467     processesInOrder = [];
468     numberOfFaults = 0;
469     numberOfRefs = 0;
470     numberOfPages = 0;
471     processesInOrder = [];
472     victimProcess = '';
473     victimPage = '';
474     victimFrame = '';
475     rawRequest = '';
476     splitRequest = '';
477     givenProcess = '';
478     givenPage = '';
479 }
480
481 /**
482  * Sets the error message and clears the page.
483  * @param message The error message to display
484  */
485 function alertError(message) {
486     ViewModel.setError(message);
487 }
488
```