```
  1 import components.program.Program;
  9
 10 /**
 11  * Program to test method to interpret a BugsWorld virtual machine program.
 12  *
 13  * @author Gage Farmer
 14  *
 15  */
 16 public final class BugsWorldVMInterpreter {
 17
 18     /*
 19      * Private members --------------------------------------------------------
 20      */
 21
 22     /**
 23      * BugsWorld possible cell states.
 24      */
 25     enum CellState {
 26         EMPTY, WALL, FRIEND, ENEMY;
 27     }
 28
 29     /**
 30      * Private constructor so this utility class cannot be instantiated.
 31      */
 32     private BugsWorldVMInterpreter() {
 33     }
 34
 36      * Gets a file name from the user and loads a BL compiled program from the
 53     private static int[] loadProgram(SimpleReader in, SimpleWriter out) {
 66
 68      * Returns whether the given integer is the byte code of a BugsWorld virtual
 81     private static boolean isPrimitiveInstructionByteCode(int byteCode) {
 82         return (byteCode == Instruction.MOVE.byteCode())
 83                 || (byteCode == Instruction.TURNLEFT.byteCode())
 84                 || (byteCode == Instruction.TURNRIGHT.byteCode())
 85                 || (byteCode == Instruction.INFECT.byteCode())
 86                 || (byteCode == Instruction.SKIP.byteCode())
 87                 || (byteCode == Instruction.HALT.byteCode());
 88     }
 89
 91      * Returns the value of the condition in the given conditional jump
108     private static boolean conditionalJumpCondition(CellState wbs,
144
146      * Checks whether the given location {@code loc} is the location of an
160     private static boolean isValidInstructionLocation(int[] cp, int loc) {
179
180     /*
181      * Public members --------------------------------------------------------
182      */
183
184     /**
185      * Returns the location of the next primitive instruction to execute in
186      * compiled program {@code cp} given what the bug sees {@code wbs} and
187      * starting from location {@code pc}.
188      *
189      * @param cp
190      *            the compiled program
191      * @param wbs
192      *            the {@code CellState} indicating what the bug sees
```

```
193      * @param pc
194      *            the program counter
195      * @return the location of the next primitive instruction to execute
196      * @requires <pre>
197      * [cp is a valid compiled BL program]  and
198      * 0 <= pc < cp.length  and
199      * [pc is the location of an instruction byte code in cp, that is, pc
200      *  cannot be the location of an address]
201      * </pre>
202      * @ensures <pre>
203      * [return the address of the next primitive instruction that
204      *  should be executed in program cp given what the bug sees wbs and
205      *  starting execution at address pc in program cp]
206      * </pre>
207      */
208     public static int nextPrimitiveInstructionAddress(int[] cp, CellState wbs,
209             int pc) {
210         assert cp != null : "Violation of: cp is not null";
211         assert wbs != null : "Violation of: wbs is not null";
212         assert cp.length > 0 : "Violation of: cp is a valid compiled BL program";
213         assert 0 <= pc : "Violation of: 0 <= pc";
214         assert pc < cp.length : "Violation of: pc < cp.length";
215         assert isValidInstructionLocation(cp, pc) : ""
216                 + "Violation of: pc is the location of an instruction byte code in
    cp";
217
218         while (!isPrimitiveInstructionByteCode(cp[pc])) {
219             switch (cp[pc]) {
220                 case 6:
221                     if (isValidInstructionLocation(cp, cp[pc + 1])) {
222                         pc = cp[pc + 1];
223                     }
224                     break;
225
226                 case 7:
227                     if (wbs != CellState.EMPTY) {
228                         if (isValidInstructionLocation(cp, cp[pc + 1])) {
229                             pc = cp[pc + 1];
230                         }
231                     } else {
232                         pc++;
233                     }
234                     break;
235
236                 case 8:
237                     if (wbs == CellState.EMPTY) {
238                         if (isValidInstructionLocation(cp, cp[pc + 1])) {
239                             pc = cp[pc + 1];
240                         }
241                     } else {
242                         pc++;
243                     }
244                     break;
245
246                 case 9:
247                     if (wbs != CellState.WALL) {
248                         if (isValidInstructionLocation(cp, cp[pc + 1])) {
249                             pc = cp[pc + 1];
250                         }
```

```
251                     } else {
252                         pc++;
253                     }
254                     break;
255
256                 case 10:
257                     if (wbs == CellState.WALL) {
258                         if (isValidInstructionLocation(cp, cp[pc + 1])) {
259                             pc = cp[pc + 1];
260                         }
261                     } else {
262                         pc++;
263                     }
264                     break;
265
266                 case 11:
267                     if (wbs != CellState.FRIEND) {
268                         if (isValidInstructionLocation(cp, cp[pc + 1])) {
269                             pc = cp[pc + 1];
270                         }
271                     } else {
272                         pc++;
273                     }
274                     break;
275
276                 case 12:
277                     if (wbs == CellState.FRIEND) {
278                         if (isValidInstructionLocation(cp, cp[pc + 1])) {
279                             pc = cp[pc + 1];
280                         }
281                     } else {
282                         pc++;
283                     }
284                     break;
285
286                 case 13:
287                     if (wbs != CellState.ENEMY) {
288                         if (isValidInstructionLocation(cp, cp[pc + 1])) {
289                             pc = cp[pc + 1];
290                         }
291                     } else {
292                         pc++;
293                     }
294                     break;
295
296                 case 14:
297                     if (wbs == CellState.ENEMY) {
298                         if (isValidInstructionLocation(cp, cp[pc + 1])) {
299                             pc = cp[pc + 1];
300                         }
301                     } else {
302                         pc++;
303                     }
304                     break;
305
306                 case 15:
307                     if (Math.random() >= .5) {
308                         if (isValidInstructionLocation(cp, cp[pc + 1])) {
309                             pc = cp[pc + 1];
```

```
310                         }
311                     } else {
312                         pc++;
313                     }
314                     break;
315 
316                 case 16:
317                     if (!conditionalJumpCondition(wbs, cp[pc])) {
318                         if (isValidInstructionLocation(cp, cp[pc + 1])) {
319                             pc = cp[pc + 1];
320                         }
321                     } else {
322                         pc++;
323                     }
324                     break;
325 
326                 default:
327                     pc++;
328                     break;
329             }
330         }
331 
332         // This line added just to make the program compilable.
333         return pc;
334     }
335 
337      * Main method.
342     public static void main(String[] args) {
436 
437 }
438 
```