Tinco Andringa (s0159786) & Daniel Moscoviter (s0140155)

System Validation Assignment 2

1. Lock requirements

Both doors are not open at the same time

We assume the doors are modeled as modules with an open property that tells wether they are open. The specification in LTLSpec:

```
G !(door1.open & door2.open)
```

Allow at least one ship to pass eventually

We assume a ship is modeled as a module, with a property position that is defined by the enumeration `{before, in, after}` that are set to the position the ship is in relative to the lock and the direction it is going in.

```
G (ship.before -> F ship.after)
```

This specification is more strict than 'at least one' in that it guarantees that every ship entering the lock will eventually pass it instead of just one of them. In my opinion that makes more sense as a lock requirement.

In CTLSpec making sure it is possible at least one ship to pass would be:

```
EF (ship.after)
```

Only open the door if the water levels are equal

We assume the doors are modeled as modules with a can*open that guards the opening of that door. The lock also has three parts, left, middle and right, that have a water*level property that can have values from the enum `{low, high}`.

```
G ((door1.can_open -> left.water_level = middle.water_level) &
   (door2.can_open -> middle.water_level = right.water_level))
```

The doors should not open or close without any ships waiting

We assume that the parts of the lock also have an occupied property that indicates wether a ship is waiting there. The doors also have properties opening and closing.

```
G ((door1.opening | door1.closing -> left.occupied | middle.occupied) &
   (door2.opening | door2.closing -> right.occupied | middle.occupied))
```

Boats should pass in order of arrival

We assume that every ship has a number that is assigned to it when it enters the queue for the lock. The number is equal to the amount of ships that have entered the queue for the lock thus far.

We also assume the lock has a list of all boats currently waiting in or before the lock called `waitingShips`.

The lock has the following specification for the pass method that gets called to remove a ship from the lock when it has passed in JML.

```
//@ requires \forall Ship otherShip; waitingShips.has(otherShip); otherShip.number > ship.number
```

It should be possible to adjust the waterlevel

We assume that there is a property water*level*adjustable. In CTLSpec:

```
G (water_level_adjustable)
```

This is hardly a useful property for the system. At the very least it is a dangerous property. A better property would be, the water level should only be adjustable when both doors are closed, like so:

```
G (water_level_adjustable -> door1.closed & door2.closed)
```

This property protects the doors and the ships against water damage.

Do not close the door while a ship is leaving

We assume that there is a property on the lock called direction that denotes wether the ships currently being served are going from left to right or from left to right.

```
G (( direction = leftToRight & middle.occupied -> door2.can_close = false) &
   ( direction = rightToLeft & middle.occupied -> door1.can_close = false))
```

This assumes that with leaving it is meant that the middle, the lock part, is being left.

2. JML Annotations

Board.java

Added an invariant that makes sure xSize and ySize are greater than 0 to give the board meaningful dimensions. Though a 1x1 board still isn't very meaningful.

```
//@ public invariant xSize > 0 && ySize > 0;
```

Added invariants that make sure the board is of the size xSize and ySize say it is.

```
//@ public invariant items.length == xSize;
//@ public invariant (\forall int i; 0 <= i && i < xSize; items[i].length == ySize);
```

Added a requires spec that demands the parameters are bigger than 0.

```
//@ requires xSize > 0 && ySize > 0;
```

Added an ensures spec that says the method populates the items array with Ground objects.

//@ ensures (\forall int x,y; 0 <= x && 0 <= y && x < xSize && y < ySize; items[x][y] instanceof Ground);

This spec revealed an error in the constructor. The two for loops went from 1 to Size instead of from 0 to Size. We fixed this and the specs all passed again.

Added requires specs that make sure no items are put on board on invalid positions.

```
//@ requires item.position().x >= 0 && item.position().x < xSize;
//@ requires item.position().y >= 0 && item.position().y < ySize;
```

BoardItem.java

Added pure attribute to pure functions

```
//@ public invariant this instanceof Ground ==> !isMovable();
```

Added invariants that make sure the ground is not moveable, that something that is moveable can't be stepped upon and that only ground and crates can be marked.

```
//@ public invariant this instanceof Ground ==> !isMovable();
//@ public invariant isMovable() ==> !isCanStepOn();
//@ public invariant !(this instanceof Ground) && !(this instanceof Crate)  ==> !isMarked();
```

Added requirements that make sure only a moveable item can be moved to a legal position. And if the requirements are not correct an exception is thrown.

```
//@ requires isMovable();
//@ requires position().isValidNextPosition(newPosition);
//@ ensures  position() == newPosition;
//@ also
//@ requires !isMovable();
//@ requires !position().isValidNextPosition(newPosition);
//@ signals_only IllegalStateException;
//@ ensures position() == \old(position());
```

Crate.java

Added an ensures that makes sure the position is set.

```
//@ ensures position == p;
```

Added also ensures that give information about the properties of this crate.

For the isCanStepOn()

```
 //@ also ensures \result == false;
```

For the isMovable()

```
 //@ also ensures \result == false;
```

For the isMarked() we also check if this is a regular instance of Crate.

//@ also ensures getClass().getName() == "Crate" ==> \result == false;

And the position()

```
//@ also ensures \result == position;
```

Game.java

Added some consistency properties for the player.

```
//@ public invariant player.position().x >= 0 && player.position().x < board.xSize;
//@ public invariant player.position().y >= 0 && player.position().y < board.ySize;
//@ public invariant board.items[player.position().x][player.position().y].isCanStepOn();
```

Added an ensures to the constructor to check initialization.

```
//@ ensures this.board == board && this.player == player;
```

Implemented the ensures for the wonGame method.

```
@     (\forall int x; 0 <= x && x < board.xSize;
@         (\forall int y; 0 <= y && y < board.ySize;
@             (board.items[x][y].isMarked () && !(board.items[x][y] instanceof Crate)) ==> !\result));
```

We added a requirement for the movePlayer that the newPosition should be valid, along with the assertion that the position is on the board.

```
//@ requires player.position().isValidNextPosition(newPosition);
boolean movePlayer (Position newPosition) {
  //@ assert newPosition.x >= 0 && newPosition.x < board.xSize;
  //@ assert newPosition.y >= 0 && newPosition.y < board.ySize;
```

We discovered that the mouseClicked function of GameGUI breaches this contract. Since GameGUI is in charge of validating input, we left this bug in.

We then added a check before the moving of an object that the object is movable.

```
  //@ assert board.items[newPosition.x][newPosition.y].isMovable();
```

Ground.java

Like with the Crate.java we added requirements and ensures for the basic properties and the constructor.

```
//@ ensures position == p;
```

For the isCanStepOn()

```
 //@ also ensures \result == true;
```

For the isMovable()

```
 //@ also ensures \result == false;
```

For the isMarked(), checking that the class is just Ground.

//@ also ensures getClass().getName() == "Ground" ==> \result == false;

And the position()

```
//@ also ensures \result == position;
```

MarkedCrate.java

Here we added an ensures to isMarked() that checks that it always returns true.

```
 //@ also ensures \result == true;
```

MarkedGround.java

Here we also added an ensures to isMarked() that checks that it always returns true.

```
 //@ also ensures \result == true;
```

Player.java

We added a spec that ensures the player is constructed properly.

```
//@ ensures position == p;
```

We added a spec that ensures the position is correctly returned.

```
//@ ensures \result == position;
```

We added specs to the SetPosition method that guard against invalid next positions and ensure the position gets updated.

```
//@ requires position().isValidNextPosition(newPosition);
//@ ensures position == newPosition;
```

Position.java

We added an invariant that says that the position is never negative.

```
//@ public invariant x > -1 && y > -1;
```

Then added specs for the constructor, making sure it gets constructed properly.

```
//@ requires x > -1 && y > -1;
//@ ensures this.x == x && this.y == y;
```

And specs to validate the correct function of the equals method.

```
//@ requires o instanceof Position;
//@ ensures \result == ((Position)o).x == x && ((Position)o).y) == y;
```

Then we added a spec to see if the important isValidNextPosition method works correctly.

```
//@ ensures \result ==> (newPosition.x == x && (newPosition.y == y + 1 || newPosition.y == y-1)) ||
//@                     (newPosition.y == y && (newPosition.x == x + 1 || newPosition.x == x-1));
```

Wall.java

We added the standard boardItem property specifications. For the constructor.

```
//@ ensures position == p;
```

For isCanStepOn()

```
 //@ also ensures \result == false;
```

For isMovable()

```
 //@ also ensures \result == false;
```

For isMarked()

//@ also ensures getClass().getName() == "Crate" ==> \result == false;

And the position()

```
//@ also ensures \result == position;
```

3. Static Checking

We ran the escjava2 tool on checkWonRow, and the first warning it gave was that y could be negative. To solve this we added a loop_invariant that stated that y is always greater than -1.

```
//@ loop_invariant y >= 0;
```

In addition, y could be too large. We solved this with a requires.

```
//@ requires row.length == board.ySize;
```

Finally, there was the problem of rows possibly being null. We added a forall check to catch this error.

```
//@ requires (\forall int y; 0 <= y && y < board.ySize; row[y] != null);
```

Then, we had to run the tool on gameWon. The first error and its fix were similar to the ones in checkWonRow.

```
//@ loop_invariant x >= 0;
```

Due to time constraints we were not able to fix the remaining warnings.

4. Abstract Specifications

We added a model field called gameWon with a rewritten gameWon representation.

```
/*@ public model boolean gameWon;
  @ private represents gameWon <-
  @    (\forall int x; 0 <= x && x < board.xSize;
  @        (\forall int y; 0 <= y && y < board.ySize;
  @            (board.items[x][y].isMarked () && (board.items[x][y] instanceof Crate)) ||
  @             !board.items[x][y].isMarked ()
  @              ));
*/
```

And then rewrote the ensures for the wonGame method to read:

```
//@ ensures \result == gameWon;
```

For the gameStuck field we defined a representation that checks for each crate if there's a free space next to it, and if the opposite space is also free then it is not stuck otherwise it is.

```
/*@ public model boolean gameStuck;
  @ private represents gameStuck <-
  @   !gameWon && (\forall int x; 0 <= x && x < board.xSize;
  @        (\forall int y; 0 <= y && y < board.ySize;
  @          (board.items[x][y] instanceof Crate && !(
               (board.items[x-1][y] instanceof Ground && board.items[x+1][y] instanceof Ground) ||
               (board.items[x][y-1] instanceof Ground && board.items[x][y+1] instanceof Ground)
              )) || !(board.items[x][y] instanceof Crate) ));
*/
```

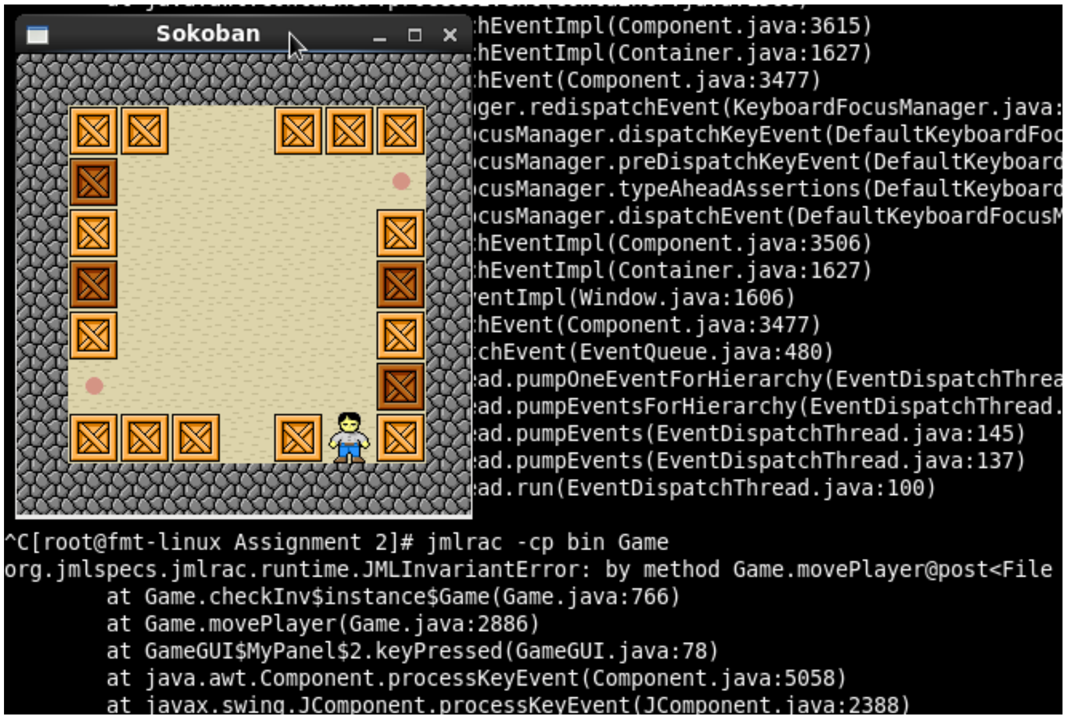
We then added the invariant that the game should never be stuck unless it is won.

```
//@ public invariant !gameWon ==> !gameStuck;
```

In jmlrac, when all crates are pushed to the walls, it gives the following error:

```
org.jmlspecs.jmlrac.runtime.JMLInvariantError: by method Game.movePlayer@post<File "src/Game.java", line 101, character 15>
```

Indicating that the invariant has been breached. A screenshot has been included:



```
hEventImpl(Component.java:3615)
hEventImpl(Container.java:1627)
hEvent(Component.java:3477)
ger.redispatchEvent(KeyboardFocusManager.java:
cusManager.dispatchKeyEvent(DefaultKeyboardFoc
cusManager.preDispatchKeyEvent(DefaultKeyboard
cusManager.typeAheadAssertions(DefaultKeyboard
cusManager.dispatchEvent(DefaultKeyboardFocusM
hEventImpl(Component.java:3506)
hEventImpl(Container.java:1627)
entImpl(Window.java:1606)
hEvent(Component.java:3477)
chEvent(EventQueue.java:480)
ad.pumpOneEventForHierarchy(EventDispatchThrea
ad.pumpEventsForHierarchy(EventDispatchThread.
ad.pumpEvents(EventDispatchThread.java:145)
ad.pumpEvents(EventDispatchThread.java:137)
ad.run(EventDispatchThread.java:100)
^C[root@fmt-linux Assignment 2]# jmlrac -cp bin Game
org.jmlspecs.jmlrac.runtime.JMLInvariantError: by method Game.movePlayer@post<File
        at Game.checkInv$instance$Game(Game.java:766)
        at Game.movePlayer(Game.java:2886)
        at GameGUI$MyPanel$2.keyPressed(GameGUI.java:78)
        at java.awt.Component.processKeyEvent(Component.java:5058)
        at javax.swing.JComponent.processKeyEvent(JComponent.java:2388)
```

To see wether a crate can't be moved anymore is easy to check like we did in the gameStuck representation. You loop over all crates, and see if there's ground left to it, if there is there should either be ground to the right of it or there should be ground above it. If there is ground above it and not left or right, then there should be ground below it. If this doesn't hold, then the crate is stuck.

5. Test Generation

We ran out of time for test generation.