**IMPORTANT**:

* A precondition for all these movement functions is that a function has been implemented to handle wall collisions.
* Each robot would have its velocity set during initialization.
* There should also be a function to perform the check to see if the robot is still moving.
* Robot should have local distance and rotation variables to be set as movement calculations are being performed.

**Total movement behaviors**: 6

1. Still – The robot doesn’t move.
2. BulletAvoid – robot only moves when hit.
3. Random – at random intervals, the robot moves and rotates by a random amount in a random direction.
4. Linear – the robot moves back and by a set amount.
5. RandomLinear – at random intervals, the robot moves back and forth by a random amount.
6. Circular – the robot moves along a continuous circular path.

**Additional genetic parameters**:

1. minDistance – the minimum distance the robot will move in “one” go.
2. maxDistance – the maximum distance the robot will move in “one” go.
3. minRotation – the minimum amount the robot will rotate in “one” go.
4. maxRotation – the maximum amount the robot will rotate in “one” go.
5. moveDelay – the maximum amount of time to wait before executing the next movement.
6. moveDir – Determines whether the robot will move forward (+1) or backward (-1).
7. velocity – how quickly the robot should move (0-MAXVELOCITY)

**Still()**

Narrative: Robot doesn’t move. Not even sure we will need a function call.

Pre-conditions: None

Post-conditions: Robot hasn’t moved from its spawn location.

**BulletAvoid(minDist, maxDist, rotateDir, moveDir, HitByBulletEvent)**

Narrative: Robot only moves when hit by a bullet, functions similar to the “Fire” robot in sample.

Pre-conditions: rotateDir & moveDir are either +/- 1. HitbyBulletEvent object is still defined. This is called when hit by a bullet.

Post-conditions: Robot moves away perpendicular to the bullet.

Rotate to be perpendicular to the direction the bullet came from

//which way it rotates, clockwise vs. counterclockwise, is determined by rotateDir

Distance = random value between min- and maxDistance \* moveDir

Flip “directions” of rotateDir and moveDir

Move distance

Examine surroundings

**Random(minDistance, maxDistance, minRotation, maxRotation, moveDelay)**

Narrative: Randomly moves the robot about the map.

Pre-condition: Potentially none (depends on how moveDelay is implemented).

Post-condition: Robot moves to a new location.

If not moving

Delay by a random value between 0 and moveDelay

//Not sure if we would want to use a java function for this or look at one of Robocode’s

// methods (i.e. waitFor).

Distance = random value between min- and maxDistance

Distance = distance \* Randomly selected direction (forward, +, or backward, -)

Move distance

Rotation = random value between min- and maxRotation

Rotate by rotation

Else

Finish movement

**Linear(minDistance, maxDistance, moveDir)**

Narrative: Moves the robot back and forth.

Pre-condition: None

Post-condition: The robot moves.

If not moving

Distance = random # between min- and maxDistance \* moveDir

//It might be better to have the distance set during initialization and use the same

// amount each time.

Move distance

moveDir = moveDir \* -1

else

finish movement

**RandomLinear(minDistance, maxDistance, moveDir, moveDelay)**

Narrative: The robot moves back and forth at random intervals.

Pre-conditions: None

Post-condition: Robot moves

If not moving

Delay by a random value between 0 and moveDelay

//Not sure if we would want to use a java function for this or look at one of Robocode’s

// methods (i.e. waitFor).

Distance = random value between min- and maxDistance

Distance = distance \* Randomly selected direction (forward, +, or backward, -)

Move distance

Else

Finish movement

**Circular(minDistance, maxDistance, minRotation, maxRotation)**

OR **Circular()**

Narrative: Robot moves around in a circular path.

Pre-condition: May want to assign distance and rotation during initialization rather than calculating each time movement is called. If that is the case, then use the second function header.

Post-condition: Robot moves.

If not moving

Move distance

Rotate by rotation

Else

Finish movement

**WallCollision(HitWallEvent e)**

Narrative: Called whenever the robot collides with a wall. Will have the robot move in reverse of what led to the collision.

Pre-condition: None

Post-condition: The robot moves back the way it came by it’s remaining movement.

Direction = direction \* -1

Rotation = 180 – rotation

Move direction

Rotate by rotation