```
/* Shared plans (see also [8]) */
def crowdTracking(p: Double, r: Double, w: Double, t: Double,
                  thCrowd: Double, thDanger: Double): Crowding = {
  val localDensity = localDensityEstimation(p, r)
  mux(recentlyTrue(localDensity > thCrowd, t)){
    collectiveDensityEstimation(p, r, thDanger)
  } { NoRisk }
}

def crowdDispersal(c: Crowding, r: Double): TargetPosition =
    // if a device is closer than r to any device with crowding level = Risk or higher
    // then compute a movement vector to move away from the Overcrowded area, or do not move
    branch(distanceTo(c >= Risk) < r){ vectorFrom(c == Overcrowded) }{ currentPosition() }

/* Shared beliefs: see [8] for motivation of the specific values */
val p = 0.005 // proportion of people with a corresponding device (agent)
val r  = 30   // range in metres for local crowding estimation
val w  = 0.25 // fraction of walkable space
val t = 60.0  // timeframe (in seconds) for risk monitoring
val thCrowd  = 1.08 // relevant crowding threshold
val thDanger = 2.17 // dangerous crowding threshold
val riskRange = 50  // distance to risk for triggering alert

/* Aggregate program: main logic */
val crowdingLevel = crowdTracking(p, r, w, thCrowd, thDanger, t)
crowdDispersal(crowdingLevel, riskRange) // dispersal advice
```