```
/*
this is a test implemetation of the morality action pillar system.
 this first test only uses two MAPs, The propensity for murder and thievery.
 uses the value to check whether an ai agent will commit a murder, steal or both,
 whether they get caught and whether an observer agent in the room will report the
action
 based on their values.

 this test implements different scenarios where different agents have differing
behaviours,
 depending on who is the the room with them.

 in this scenario there are 4 agents in a room who are potential murderers. Agents
interact with the rest of the room,
 and depending on their respective propensities ( which accounts for hw many people
are in the room at the time,
 how many kills they've already committed.


 */

NPC a, b, c, d;

void setup() {
  a = new NPC("Player", 50);   //give your character's whatever propensity fror murder
you want and press play
  b = new NPC("Bob", 0);
  c = new NPC("James", 0);
  d = new NPC("Kid", 0);
  NPC[] CharID = {a, b, c, d};
  Room test;
  test = new Room("Bar", CharID.length, 0);
  test.populate(CharID);
  test.population();
  Room[] world = {test};

  //println(shareRoom(a,b));
  println(test.numOfOccupants() + " Occupant(s)");
```

```
  for (int i = 0; i < CharID.length; i++) {
    test.occupants.get(i).murderPropensity = murPropensity(world);
    println(test.occupants.get(i).murderPropensity);
        //println(test.occupants.get(i).murTol);
            //println(test.occupants.get(i).mapMurd);
  }
}

//void draw() {
////FURTHER WORK
////make a bigger world
////allow agents around the world(in different rooms, of different sizes to interact simultaneously, and to move between rooms;
////create types of NPC's with different behaviours, like NPC's that prefer less crowded rooms or NPC's that only attack,
////certain types of people
////implement function which automatically cause agents in the same room to pick a random agent to engage with every so often;
////check who they shareRooms with a choose from that?;
////below implemented for simplicity
//  while (a.isAlive) {

//    if (a.isAlive) {
//      a.interact(b);
//      a.interact(c);
//      a.interact(d);
//      if(b.isAlive != true && c.isAlive != true && d.isAlive != true ){
//       println(a.name + " wins");
//       exit();
//      }
//    }
//    if (b.isAlive) {
//      b.interact(a);
//      b.interact(c);
//      b.interact(d);
//       if(a.isAlive != true && c.isAlive != true && d.isAlive != true ){
//       println(b.name + " wins");
//       exit();
//      }
```

```
//    }
//    if (c.isAlive) {
//      c.interact(a);
//      c.interact(b);
//      c.interact(d);
//       if(a.isAlive != true && b.isAlive != true && d.isAlive != true ){
//       println(c.name + " wins");
//       exit();
//      }
//    }
//    if (d.isAlive) {
//      d.interact(a);
//      d.interact(b);
//      d.interact(c);
//       if(a.isAlive != true && b.isAlive != true && c.isAlive != true ){
//       println(d.name + " wins");
//       exit();
//      }
//    }
//  }
//  exit();
//}

String shareRoom(NPC a, NPC b) {
  String agentALoc = a.roomNameIn;
  String agentBLoc = b.roomNameIn;
  String reply;
  if (agentALoc == agentBLoc) {
    reply = a.name + " is in the same room as " + b.name;
  } else {
    reply = "They aren't in the same room";
  }

  return reply;
}

static int murPropensity(Room[] World) {
  Room currRoom;
  int numOfObservers = 0;
```

```
  int murderMAP = 0;
  int murderTolerance = 0;
  int murderPropensity = 0;
  for (int i = 0; i < World.length; i++) {
    currRoom = World[i];
    println(currRoom);
    for (int j = 0; j < currRoom.occupants.size(); j++) {
      if (currRoom.occupants.get(i).roomNameIn == currRoom.roomName) {
        murderMAP = currRoom.occupants.get(i).mapMurd;
        murderTolerance = currRoom.occupants.get(i).murTol;
        numOfObservers = currRoom.numOfOccupants() - 1 ;
       murderPropensity = ((murderMAP * 50) + (murderTolerance/2)) / (1 +
numOfObservers);
         // formula for how likely an agent is to commit a murder, based on their murders
already committed, tolerance to murder and how many observers there are
        // print(Characters[j].name + "'s murder propensity is " + " " +
Characters[j].murderPropensity);

      }

    }

  }
         return murderPropensity;
}



//MURDER PROPENSITY TESTSrst
////////////////////////////////////////////////////////////////////////////

//int murderTolerance = 100;
//int murdersCommitted = 0;

//if (murdersCommitted > 0) {
//  murderPropensity = (murdersCommitted * 10);
//} else {
//  murderPropensity = PI * (murderTolerance/ 5);
//}
```

```
////////////////////////////////////////////////////////////////////////////
//int numOfObservers = 0;

//int murdersCommitted = 1;
//int murderTolerance = 50;
//murderPropensity = ((murdersCommitted * 50) + (murderTolerance/2)) / (1 +
(numOfObservers));
//print(murderPropensity);

////////////////////////////////////////////////////////////////////////////
// COPY OF ORIGINAL MURPROPENSITY INCASE IT GOES AWRY
////////////////////////////////////////////////////////////////////////////
//int murPropensity(ArrayList<NPC> roomIn) {

//  int numOfObservers = 0;
//  String roomAgentIn;
//  int murdersCommitted = kills;
//  int murderTolerance = murTol;
////    boolean isObserved = beingObserved; //for implementing periodic checking of
interactions, rather than a constant watch

//  if (roomNameIn != null) {
//    roomAgentIn = roomNameIn;
//    for (int i = 0; i < roomIn.size(); i++) {
//      if (roomAgentIn == roomIn.get(i).roomNameIn) {
//        numOfObservers++;
//      }
//    }
//  }
//  murderPropensity = ((murdersCommitted * 50) + (murderTolerance/2)) / (1 +
(numOfObservers)); // formula for how likely an agent is to commit a murder, based on
their murders already committed, tolerance to murder and how many observers there
are

//  return murderPropensity;
//}
////////////////////////////////////////////////////////////////////////////
```