Bonks and Zaps are Beings that live in Grid World. Objects, put them in the Array somewhere (Could be in the object that’s stored in the array.

• Grid World is flat and consists of a grid of 20 rows and 20 columns. Perhaps a 2D array similar to that used for the noughts and crosses game would work (see tutorial 4). Use a 2D array full of something with an X and Y int field. I used Rooms.

• Each position in Grid World is indexed by row and column integer values, and is represented by a Room. Pretty much above.

• Each Room is connected by doors to all surrounding rooms, including those on the diagonal (in the corners). However, those on the edge of Grid World do not have doors into outer space. Rooms connect to adjacent and diagonals, I encoded this as an ArrayList of “doors” (Position objects (Just an X and Y value))

• Each room may contain Bonks and Zaps (their behaviour is described below). (ArrayList of each)

• There must be a Game Engine that drives activity in Grid World (there are no human players). This Game Engine must have the following characteristics: Game Engine class, similar to how you (probably) used Map in the other assignment. Runs most stuff.

o It creates the Grid World and populates it in random positions with Bonks and Zaps. The starting population is 20 Bonks and 5 Zaps. Put them in random rooms in the World (random indexes in your 2D array).

o A game consists of N turns (you can decide what value to use for N as part of your testing). On each turn, the Grid World’s Cycle is incremented by one (a Cycle is similar to a day on Earth). The world ends when N Cycles have completed. After running a whole turn’s worth of functions (E.g. Move(), reproduce()), increment int cycle. Do this until cycle = int cycleMax.

o On each Cycle every room in Grid World is visited and every Bonk and Zap in that room is commanded to act (do its stuff, as described below). Iterate through every room, from there iterate through every creature and call an act() method.

o On each Cycle the state of Grid World is displayed as text on the screen showing the Bonks and Zaps (their names, whether a Bonk or Zap, and for Bonks whether dead or alive). If possible, display as a grid, rather than the simple coordinates I used in my Panopto demonstration. Output a good toString() for every creature at turn’s end.

o Each Cycle is delayed by one second. You may use Thread.sleep(1000) to achieve this. Use thread.sleep(1000).

• Zaps are immortal (cannot be killed). On each Cycle they kill all the Bonks in the same room and then move randomly to an adjacent room, or stay where they are. Each Zap has a unique name: Zn, where n is a unique integer. Zaps have no isAlive bool, Bonks do. Zaps iterate through all Bonks in the same room and set the isAlive bool on them to false. Zap Constructor takes a number (how many have been created), name is “z” + number.

• Bonks are Mortals, i.e. Beings that reproduce and can die. Each Bonk has a unique name: Bn, where n is a unique integer. On each Cycle a live Bonk will start by reproducing with every other live Bonk in the same room that has a different gender (they are male and female). However, live male and female Bonks can only reproduce once every Cycle. Bonk babies are also born immediately (during the same Cycle) but cannot reproduce until the next Cycle. Finally, on each Cycle live Bonks move randomly to an adjacent room, or stay where they are. Dead Bonks remain where they are for all time. Bonks have isAlive, reproduce by iterating through all Bonks in same room until one is mateable, set hasReproduced on both to false, create a new Bonk in the same room with isChild set to true and hasReproduced to false. Only alive bonks get a call to their act() method.

• Note that other kinds mortal are planned for future versions of the game. Make a Mortal class, make Bonks a subclass of it, put most stuff in Mortal.