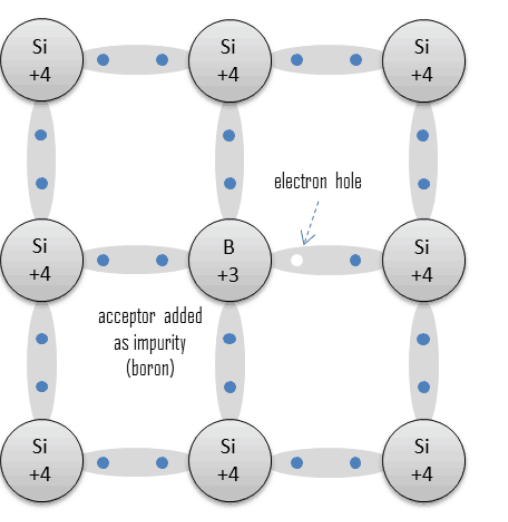
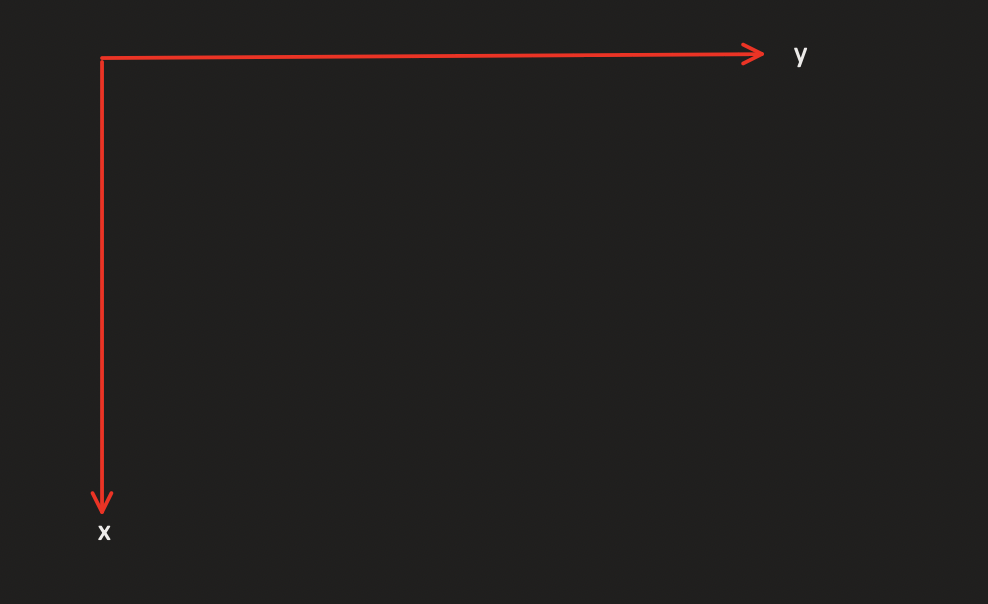
**Sample drawing**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | # | **Desired behaviors** | model | Animation |
| No reaction/ 1 charge | 1 | Conduction-band e: translational movement, exiting the semiconductor at one end & entering at the other | Done (except customized field direction and strength) |  |
| 2 | Valence-band e: might jump into valance-band hole if it is chosen | Done (except customized field direction and strength) |  |
| 3 | Valence-band hole: disappear when a Valence-band e jump into its place, & reappear in the e’s old position | Done (except customized field direction and strength) |  |
| Reaction of charges: | 4 | Valence-band e ---> Valence-band hole + Condution-band e (under heat) (explosion) | Not started |  |
|  | 5 | Valence-band hole + Condution-band e ---> Valence-band e (when a hole and a free e happen to meet) (spiral transition) | Not started |  |

**TO-DO:**

1. Visual representations
2. A timeline with trigger events & event handle functions etc.(already in the works with the invokemove() function)

Note: there are 2 Types of events:

User input is updated

Desired behaviors -- aka, the natural progression of the semiconductor in time //maybe event isn’t the right word here lol

1. Work on implementing the not started behaviors //or ignore them

**Classes**

0. Settings:

* crystalLength
* crystalWidth
* voltage
* externalField //might not be implemented
* Temperature
* Scalar (to scale speed of conduction band electron according to force fied) //will come up with a less vague variable name

1. Crystal:

* Atom[Settings.crystalHeight][Settings.crystalWidth] atoms;

Methods:

void initCrystal(String type) [\\type](file://type) as in: pure, N-type doping, or P-type doping

{

//if doping type, set an impurity atom in the middle position

//else set all other atoms to silic

}

void progress(){

[\\loop](file://loop) through every atoms by indexes, if one has moved, break

}

Temporary:

void displayCrystal()

void displayHolePosition()

1. (abstract) Atom:

* int indexX;
* int indexY;
* HashMap<String, ValenceBandCharge> valenceCharges
* protected ConductionBandElectron conductingE = null;

constr:

Atom(int indexX, int indexY)

Getters:

int getIndexX()

int getIndexY()

ValenceBandCharge getValenceCharge(String position)

Methods:

Atom getAdjacentAtom(String position)

String checkForHole()

boolean checkForConductingE()

void exchangeHoleWithElectron()

void passOnConductingE()

* + - Case1: |V| > 0

//chose new atom using min angle

* + - Case2: V = 0

// chosse new atom randomly

Temporary: String toString()

* invokeMove()

2a. SiliconAtom extends Atom:

Constr:

SiliconAtom(int indexX, int indexY){

//init 4 valence as valence e

}

Temporary: String toString()

2b. PhosphorousAtom extends Atom:

Constr:

PhosphorousAtom(int indexX, int indexY){

//init 4 valence as 3 valence e & 1 hole at up position

}

Temporary: String toString()

2c. AluminumAtom extends Atom:

AluminumAtom(int indexX, int indexY){

//init 4 valence as valence e & conducting charge as conducting e

}

Temporary: String toString()

1. (abstract) Charge:

* (abstract) invokemove()
* (abstract) move(newCoordinate)
* Vector coordinate
* checkContainer(Atom)
* checkAdjacentInvoke(Atom)

+ Vector getCoordinate()

3a. (abstract) ValenceBandCharge extends Charge

* Bond bond
* getJuxtapositionwithAtom() returns left, right, up, or down using this.bond.getAtom(this).getValenceBandCharges() and then compare the charge with each of the charges returned.

3a1.ValenceBandElectron extends ValenceBandCharge

@Override

* invokemove() //nothing

@Override

* move(newCoordinate) //move in an arc orbit from the current position to newCoordinate

3a2.ValenceBandHole extends ValenceBandCharge

@Override

* invokemove(): // get list of 4 nearby ValenceBandElectron using getNearbyElectron()

// choose the e that is most aligned with force field

//update the bonds involved

Method 1: bind the content of the Bond class with its visual representation: how to bind the content of the Bond class with its visual representation without making the transition immediately (the goal is to stretch the transition gradually over time, not jump right after the calculation)

Method 2: no binding, the content will be updated immediately, but the visual representation will only change in the event a triggerfunction() is called

// call move() of the chosen electron: move(newCoordinate = coordinate of this)

// call the move() of itself: move(newCoordinate = chosen e’s old coordinate)

# (if hole is visible, may need to have the calls to move() run simultaneously, perhaps using threads?)

@Override

* move(newCoordinate): // nothing/fade-out transition at start, then fade-in at end at newCoordinate
* getNearbyElectron() returns ArrayList<ValenceBandElectron>:

First, get nearby electron from the same bond: this.bond.getOtherCharge()

Second, get nearby electrons from from different bonds, but in adjacent atoms:

* Get list of Adjacent Atoms by this.bond.getContainerAtom(this).getNearbyAtoms()
* For each atoms in list, iterate

3b. ConductionBandElectron extends Charge

@Override

* move(newCoordinate):
  + //calculate v = scalar \* forcefield
  + //move in an straight-line orbit from the current position to newCoordinate

1. Controls:

4a. Start // button, to start and stop simulation

4b. Voltage // input or slider, to change voltage value

4c. ExternalField // input or slider, to change external field

4d. Temperature // input or slider, to change external temperature

4e. Zoom // button or slider, to change view

1. Information

5a. (static) Notes // to note presentations of figures in simulation

5b. Graph // draw graph, maybe have to use java.swing

5b. (static) Circuit // equivalent circuit

5c. Energy bands: components

1. Draw

drawElectron();

drawHole();

drawRecombination();

drawBackground();

drawGraph();