Cory Buck

Operating Systems

Lab 3

A-Option

Contents

[C Option – Runtime HUD 3](#_Toc417882905)

[C Option – Storage 6](#_Toc417882906)

[B Option – Runtime HUD 8](#_Toc417882907)

[B Options – Storage 11](#_Toc417882908)

[A Option – Runtime HUD 13](#_Toc417882909)

[A Option – Discussion 18](#_Toc417882910)

[A Option – Discussion ProbeTest 20](#_Toc417882911)

[A Option – Discussion ProbeTest Output 23](#_Toc417882912)

[A Option – Storage 24](#_Toc417882913)

[A Option – Storage Interface 26](#_Toc417882914)

[A Option – Poisson Distribution 27](#_Toc417882915)

[A Option –Task Force Command 29](#_Toc417882916)

[A Option – Scout Probe 33](#_Toc417882917)

[A Option – Photon Torpedo Probe 35](#_Toc417882918)

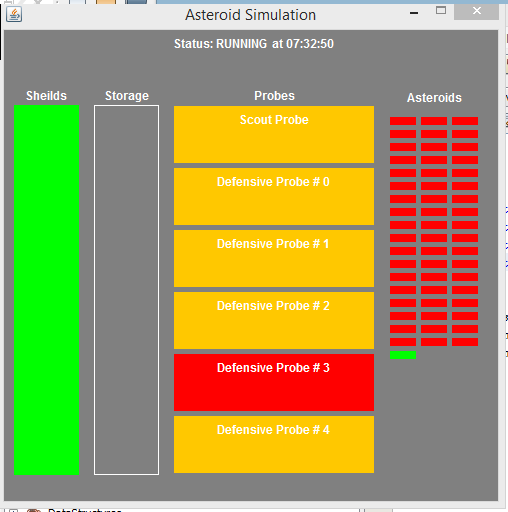
[A Option – Phaser Probe 37](#_Toc417882919)

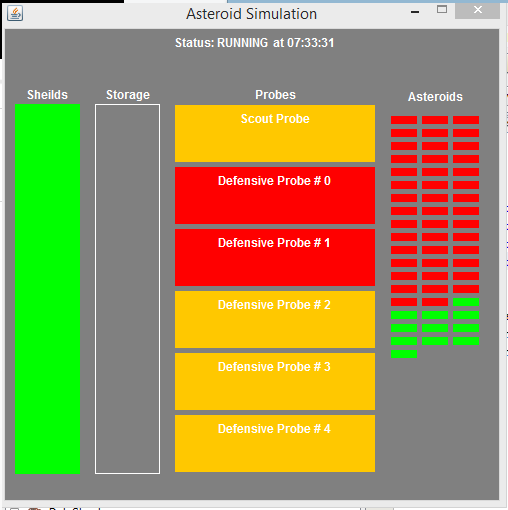
[A Option – Asteroid 39](#_Toc417882920)

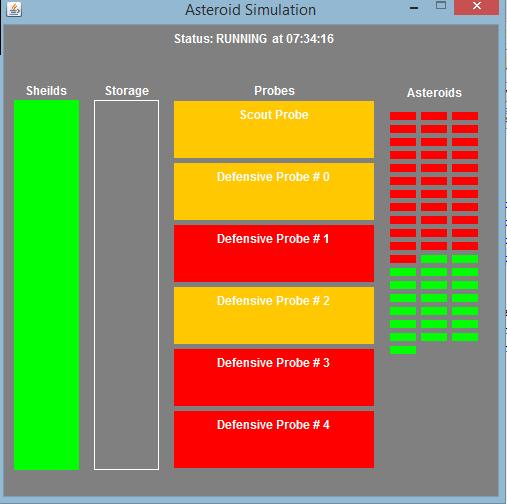
[A Option – Asteroid Simulation GUI 41](#_Toc417882921)

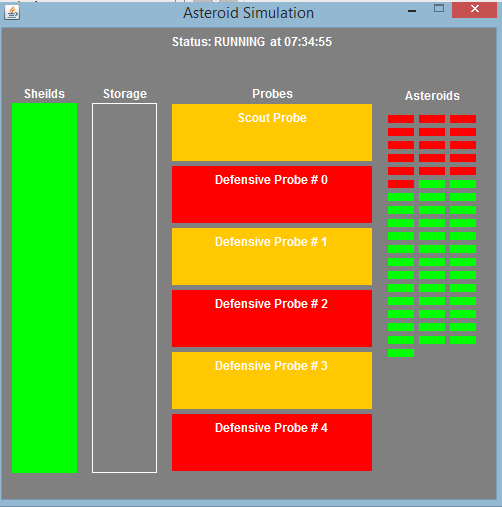
[A Option – Main 47](#_Toc417882922)

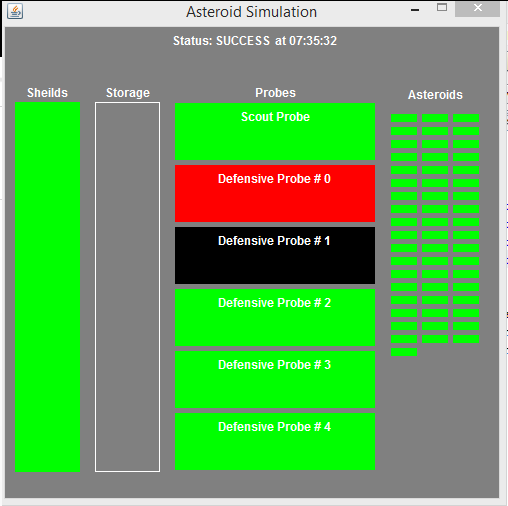
# C Option – Runtime HUD

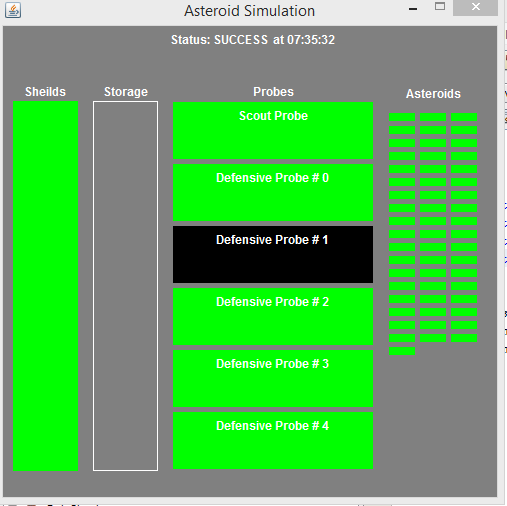












# C Option – Storage

package asteroidfield;

/\*\*

\*

\* @author cory

\*/

public class AsteroidStack implements AsteroidStorage{

private Asteroid[] storage;

private int num\_items;

private final int MAX\_SIZE;

AsteroidStack(int m){

num\_items = 0;

MAX\_SIZE = m;

storage = new Asteroid[MAX\_SIZE];

}

@Override

public void insert(Asteroid a){

if(num\_items < MAX\_SIZE){

storage[num\_items] = a;

num\_items++;

}

}

@Override

public Asteroid remove(){

if(num\_items > 0){

num\_items--;

return storage[num\_items];

}

return null;

}

@Override

public int getSize(){

return num\_items;

}

@Override

public Boolean isFull(){

return num\_items == MAX\_SIZE;

}

@Override

public Boolean isEmpty(){

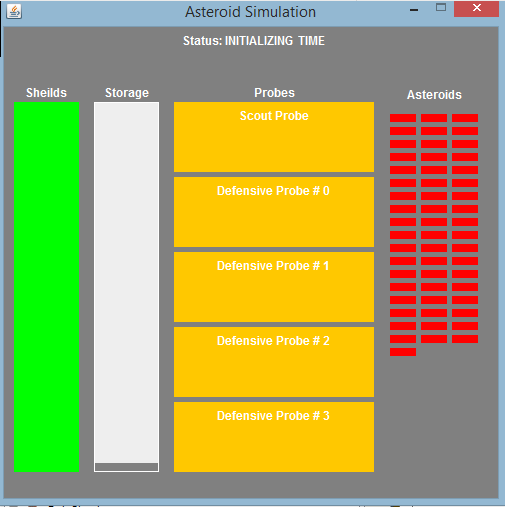
return num\_items == 0;

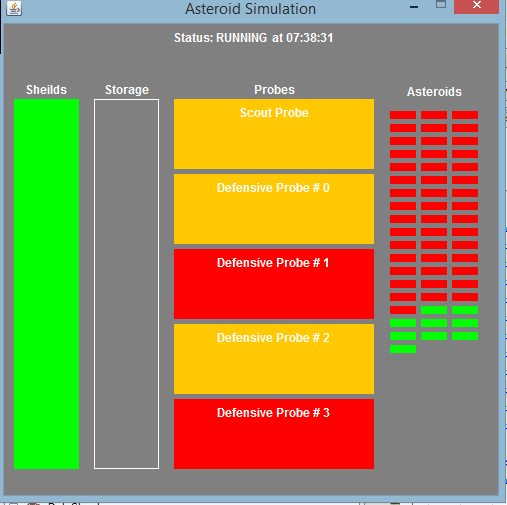
}

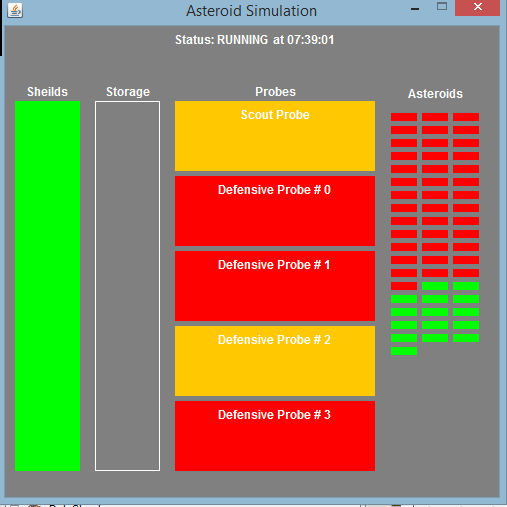
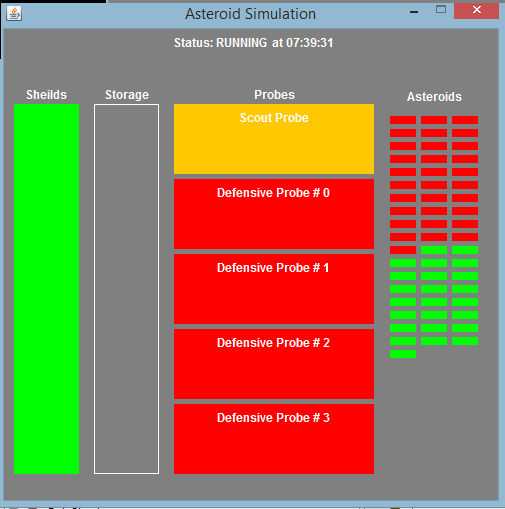
}

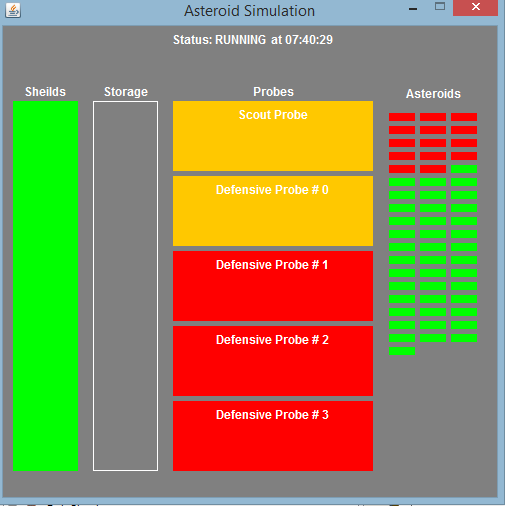
}

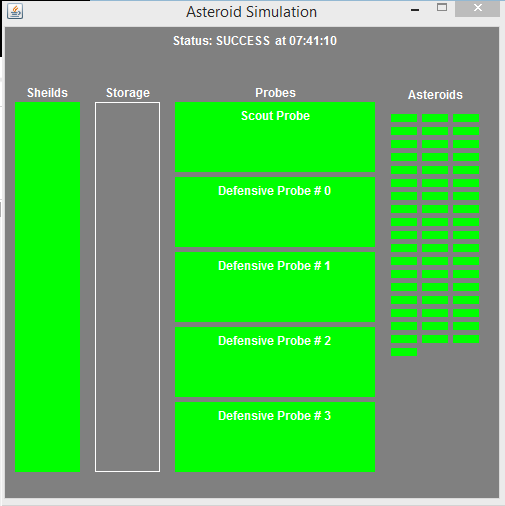
# B Option – Runtime HUD











# B Options – Storage

package asteroidfield;

/\*\*

\*

\* @author cBeezy

\*/

public class AsteroidQueue implements AsteroidStorage{

final private Asteroid[] storage;

private int num\_items;

private int front;

private int back;

public AsteroidQueue(int MAX){

storage = new Asteroid[MAX];

num\_items = 0;

front = 0;

back = 0;

}

@Override

public void insert(Asteroid a){

num\_items++;

storage[back] = a;

back = (back + 1) % storage.length;

}

@Override

public Asteroid remove(){

num\_items--;

Asteroid temp = storage[front];

front = (front + 1) % storage.length;

return temp;

}

@Override

public int getSize(){

return num\_items;

}

@Override

public Boolean isFull(){

return num\_items == storage.length;

}

@Override

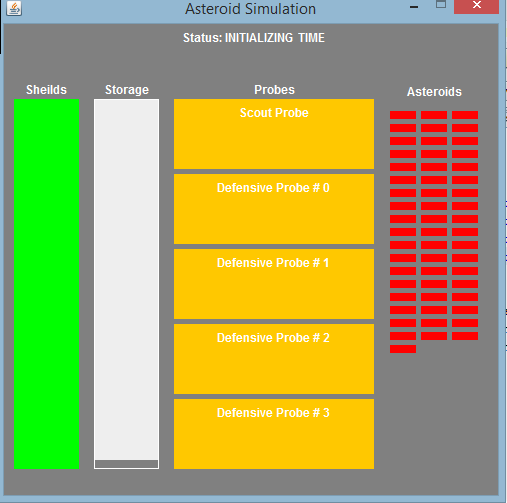
public Boolean isEmpty(){

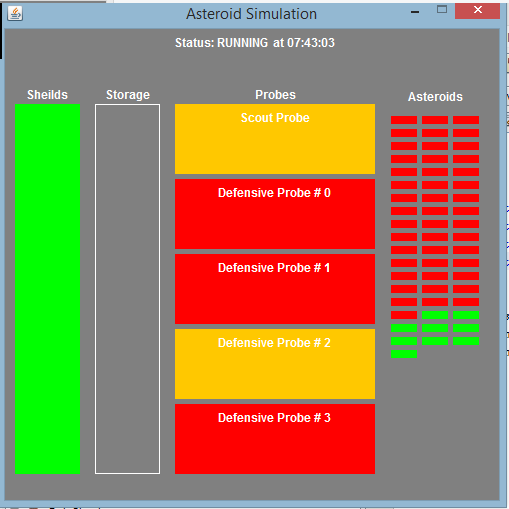
return num\_items == 0;

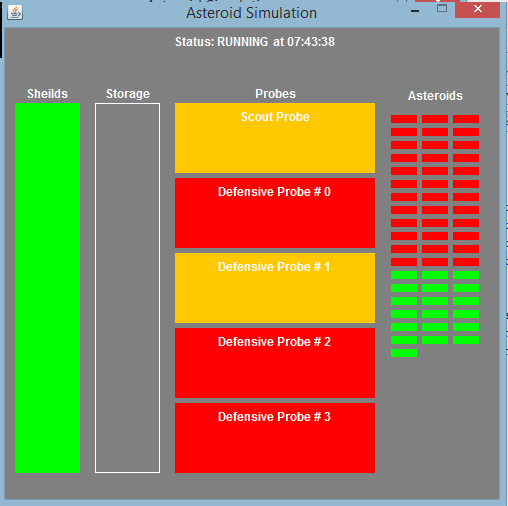
}

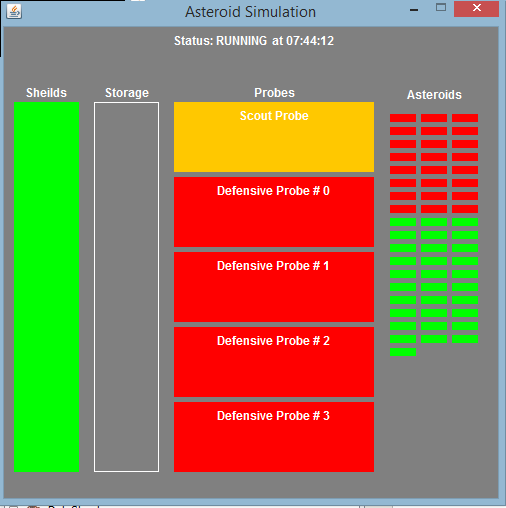
}

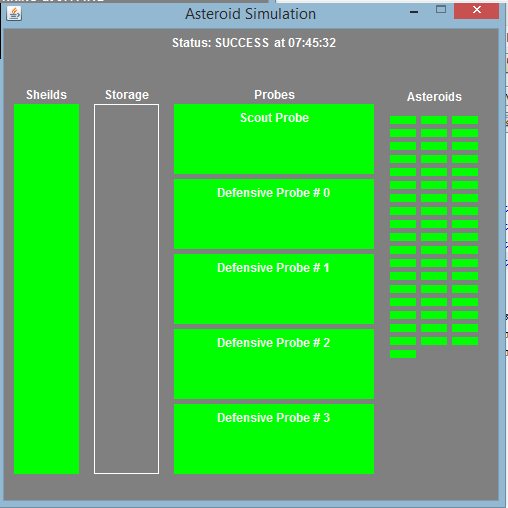
# A Option – Runtime HUD

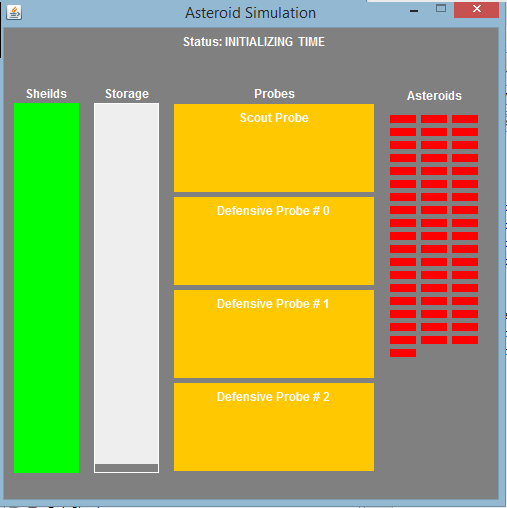


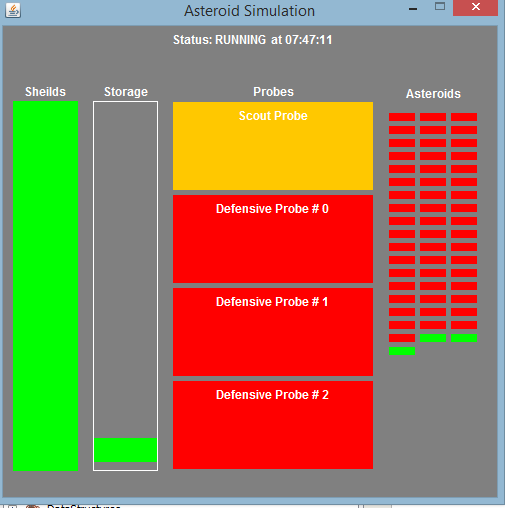


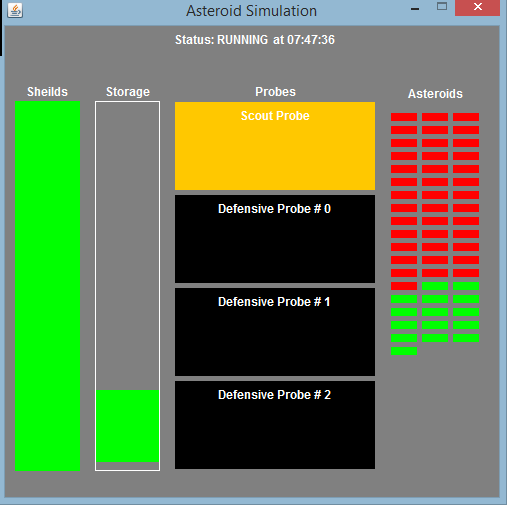


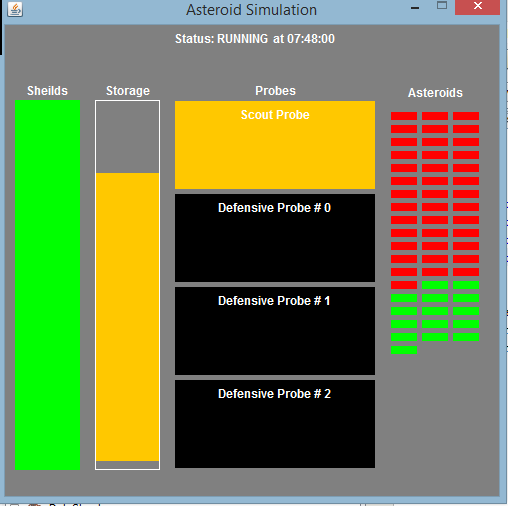


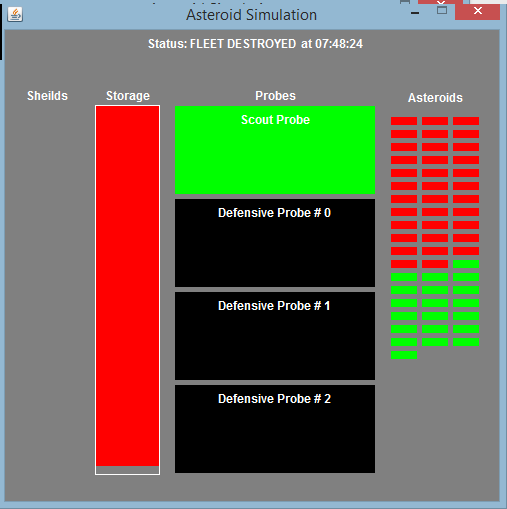












# A Option – Discussion

Storage Performance

The only difference in the “C”, “B”, and “A” options is the manner in which the Asteroids are stored, therefore performance improvement or degradation relies on which data structure was used. The “C” option stack places items on top of one another, increasing the time before the lowest item is serviced by a defensive probe and the chance of a shield hit. The queue in option “B” is an upgrade to the stack in option “C” because it ensures that asteroids are serviced in order, therefore no additional time is added to the servicing of each asteroid before it is removed. However, the “B” option queue, as well as the “C” option stack, does not consider priority and an asteroid with a longer time before impact might be serviced before an asteroid with a shorter time before impact . The “A” option stack is another improvement because it gives priority to those asteroids that will hit soonest, placing them at the top of the stack. The only disadvantage to the “A” option stack is that it requires more overhead during insertion, blocking defensive probes from being able to retrieve an Asteroid. I saw increased performance from the “C” option to the “B” option, as well as from the “B” option to the “A” option.

Probe Performance

The largest thief in asteroid destruction performance is the weapons used by the probes. The minimal destruction time for an asteroid can be calculated by the formula:

Destruction Time = ( CEILING( Asteroid\_Size / Weapon\_Damage) – 1 ) \* Weapon\_Recharge\_Rate

Utilizing the specification of the photon torpedo probe:

An asteroid of size 3 would require one torpedo and the destruction time would be 0 seconds. An asteroid of size 6 or 9 would require two torpedoes and the destruction time would be equal to one weapon recharge, 3 seconds. An asteroid of size 11 would require three torpedoes and the destruction time would be equal to two recharge weapon recharge rates, 6 seconds.

Utilizing the specification of the phaser probe :

An asteroid of size 3 would require one phaser and the destruction time would be 0 seconds.

An asteroid of size 6 would require two phaser hits and the destruction time would be equal to one weapon recharge, 3 seconds. An asteroid of size 9 would require three phaser hits and the destruction time would be equal to two weapon recharge rates, 4 seconds. An asteroid of size 11 would require four phaser hits and the destruction time would be equal to three weapon recharge rates, 6 seconds.

In comparison to the performance of the asteroid storage, the capabilities of the probes to destroy the asteroids in a timely manner is by far the biggest performance fault in this simulation. Significant gains in performance could be gained by reducing the weapon recharge rates or increasing the amount of damage the weapons could deliver.

Number of Probe Self-Destructs

When calculating the minimum number of probes needed to safely traverse the asteroid field, the probe performance was the biggest factor taken into consideration. By deriving the amount of time it would take a probe to destroy an asteroid it was possible to create a test in which destruction times were compared to asteroid impact times. In this test the number of maximum and minimum probe self-destructs were increased if neither probe was capable of destroying the given asteroid before its impact time would run out. However, if only one probe was not capable of destroying the given asteroid then the maximum probe self-destructs was incremented.

The final test consisted of one-million trials of fifty-five asteroids being destroyed. On average, the asteroids were discovered to have an average size of 7 units and an average impact time of 7.5 seconds. Utilizing the predefined destruction time formula, a photon torpedo probe would destroy this average asteroid with two photon torpedoes in three seconds. The phaser probe would destroy this average asteroid with three phaser hits in four seconds. The lowest number of possible probe self-destructs was determined to be 0. On the other hand, the highest number of possible probe self-destructs was determined to be 31. On average, the minimum amount of self-destructs that would occur was determined to be 11 and the maximum number of self-destructs was determined to be 12.

Minimum Number of Probes

In order to traverse the asteroid field safely a lack of resources can be a significant disadvantage. The average maximum of 12 probe self-destructs indicates that the minimum number of probes they should deploy to pass the “average” scenario would be 13, 12 to cover the self-destructs and one to take care of any additional asteroids. However, if the fleet is not scarce on resources and can manage 32 probes it would be in their best interest to manufacture them to cover the 31 probe self-destructs and still have one to cover any additional asteroids. The data structure used would have little effect on performance with such high quantities of probes being utilized for defense, however it would be best to use the priority stack from Option ‘A’ to ensure that whichever asteroid will impact next is serviced first.

# A Option – Discussion ProbeTest

package asteroidfield;

import static java.lang.Thread.sleep;

/\*\*

\*

\* @author cory

\*/

public class ProbeTest extends Thread implements Runnable{

static PoissonDistribution pd = new PoissonDistribution();

static int num\_asteroids\_to\_destroy;

static int num\_tests = 1000000; //one million tests

int lowest\_self\_destructs = 10;

int highest\_self\_destructs = 0;

long min\_self\_destructs = 0;

long max\_self\_destructs = 0;

int num\_tests\_ran = 0;

long total\_size = 0;

double total\_impact\_time = 0;

public static void main(String[] args){

num\_asteroids\_to\_destroy = 55;

num\_tests = 1000000;

Thread p\_test = new ProbeTest();

System.out.println("Running tests");

p\_test.start();

}

@Override

public void run(){

for(int test\_count = 0; test\_count < num\_tests; test\_count++){

int min = 0;

int max = 0;

for(int i = 0; i < num\_asteroids\_to\_destroy; i++){

int d\_status = nextProbeSelfDestructStatus();

if(d\_status == 0){ //neither asteroid can destroy, so both will have to self-destruct

min++;

max++;

}else if(d\_status == 1){

max++; //one can not destroy, so if it is the one that obtains the target it will have to self-destruct

}

}

if(min < lowest\_self\_destructs) lowest\_self\_destructs = min;

if(max > highest\_self\_destructs) highest\_self\_destructs = max;

min\_self\_destructs += min;

max\_self\_destructs += max;

num\_tests\_ran++;

try{sleep(0);}catch(InterruptedException e){}

}

System.out.println("Tests Complete!\n");

long average\_self\_destructs = (max\_self\_destructs + min\_self\_destructs) / (2 \* num\_tests);

int median\_self\_destructs = (lowest\_self\_destructs + highest\_self\_destructs) / 2;

System.out.println("Size average: " + (total\_size/(num\_tests \* 55)));

System.out.println("Impact time average: " + (total\_impact\_time/(num\_tests \* 55)));

System.out.println("Min number of self-destructs: " + lowest\_self\_destructs);

System.out.println("Median number of self destructs: " + median\_self\_destructs);

System.out.println("Max number of self-destructs: " + highest\_self\_destructs);

System.out.println("Averge min number of self-destructs: " + (min\_self\_destructs / num\_tests));

System.out.println("Average number of self-destructs: " + average\_self\_destructs);

System.out.println("Average max number of self-destructs: " + (max\_self\_destructs / num\_tests));

System.out.println("\nNumber of probes needed to ensure safe passage: " + highest\_self\_destructs);

}

public int nextProbeSelfDestructStatus(){

double size, impact; //asteroid vars

boolean destroyed\_by\_photon = false, destroyed\_by\_phaser = false; //asteroid destroyed vars

size = pd.nextSize();

impact = pd.nextImpact();

total\_size += size;

total\_impact\_time += impact;

if(timeToDestroyWithPhotonTorpedo(size) < impact) destroyed\_by\_photon = true;

if(timeToDestroyWithPhaser(size) < impact) destroyed\_by\_phaser = true;

if(destroyed\_by\_photon && destroyed\_by\_phaser) return 2;

if(destroyed\_by\_photon || destroyed\_by\_phaser) return 1;

return 0;

}

public double timeToDestroyWithPhotonTorpedo(double size){

return (Math.ceil(size / 5) - 1 ) \* 3;

}

public double timeToDestroyWithPhaser(double size){

return (Math.ceil(size / 3) - 1 ) \* 2;

}

}

# A Option – Discussion ProbeTest Output

run:

Running tests

Tests Complete!

Min number of self-destructs: 0

Median number of self destructs: 15

Max number of self-destructs: 31

Averge min number of self-destructs: 11

Average number of self-destructs: 11

Average max number of self-destructs: 12

Size average: 7

Impact time average: 7.5000358049287925

Number of probes needed to ensure safe passage: 31

BUILD SUCCESSFUL (total time: 3 seconds)

# A Option – Storage

package asteroidfield;

/\*\*

\*

\* @author cBeezy

\*/

public class SortedAsteroidStack implements AsteroidStorage{

final private Asteroid[] storage;

private int num\_items;

public SortedAsteroidStack(int MAX){

storage = new Asteroid[MAX];

num\_items = 0;

}

@Override

public void insert(Asteroid a){

Asteroid temp = a;

Asteroid temp2;

for(int i = 0; i < num\_items; i++){

temp2 = storage[i];

if(storage[i] != null){

if(temp2.getTimeOfImpact() < temp.getTimeOfImpact()){

storage[i] = temp;

temp = temp2;

}

}

}

storage[num\_items] = temp;

num\_items++;

}

@Override

public Asteroid remove(){

num\_items--;

return storage[num\_items];

}

@Override

public int getSize(){

return num\_items;

}

@Override

public Boolean isFull(){

return num\_items == storage.length;

}

@Override

public Boolean isEmpty(){

return num\_items == 0;

}

}

# A Option – Storage Interface

package asteroidfield;

/\*\*

\*

\* @author cBeezy

\*/

public interface AsteroidStorage {

public void insert(Asteroid a);

public Asteroid remove();

public int getSize();

public Boolean isFull();

public Boolean isEmpty();

}

# A Option – Poisson Distribution

package asteroidfield;

import java.util.Random;

/\*\*

\*

\* @author cBeezy

\*/

public class PoissonDistribution {

Random rand = new Random();

public double nextPoisson(double mean){

double input = rand.nextDouble();

double output;

if(0 <= input && 0.1 > input){

output = 0;

}else if(0.1 <= input && 0.2 > input){

output = 0.104;

}else if(0.2 <= input && 0.3 > input){

output = 0.222;

}else if(0.3 <= input && 0.4 > input){

output = 0.355;

}else if(0.4 <= input && 0.5 > input){

output = 0.509;

}else if(0.5 <= input && 0.6 > input){

output = 0.690;

}else if(0.6 <= input && 0.7 > input){

output = 0.915;

}else if(0.7 <= input && 0.75 > input){

output = 1.20;

}else if(0.75 <= input && 0.80 > input){

output = 1.38;

}else if(0.80 <= input && 0.84 > input){

output = 1.60;

}else if(0.84 <= input && 0.88 > input){

output = 1.83;

}else if(0.88 <= input && 0.995 > input){

output = 2.12;

}else if(0.995 <= input && 0.998 > input){

output = 5.30;

}else if(0.998 <= input && 0.999 > input){

output = 6.20;

}else if(0.999 <= input && 0.9997 > input){

output = 7.0;

}else{

output = 8.0;

}

return output \* mean;

}

}

# A Option –Task Force Command

package asteroidfield;

/\*\*

\*

\* @author cory

\*/

public class TaskForceCommand extends Thread{

private AsteroidSimulationGUI gui;

private SortedAsteroidStack asteroids;

private int max\_shield\_hits;

private int num\_shield\_hits;

private int num\_asteroids\_destroyed;

private int num\_asteroids\_left;

private int num\_probes;

private double last\_time\_update;

private Boolean destroyed;

private Boolean mission\_complete;

private Thread[] probes;

public void run(){

while(!destroyed && !mission\_complete){

try{sleep(0);}catch(Exception e){}

gui.setTime(System.currentTimeMillis());

}

if(destroyed)gui.setStatus("FLEET DESTROYED");

else gui.setStatus("SUCCESS");

}

public void Init(int MAX\_STORAGE, int SHIELD\_DEFENSE, int ASTEROIDS\_TO\_DESTROY, int PROBE\_COUNT){

gui = new AsteroidSimulationGUI(SHIELD\_DEFENSE,ASTEROIDS\_TO\_DESTROY, PROBE\_COUNT, MAX\_STORAGE);

gui.setStatus("INITIALIZING");

asteroids = new SortedAsteroidStack(MAX\_STORAGE);

max\_shield\_hits = SHIELD\_DEFENSE;

num\_shield\_hits = 0;

num\_asteroids\_destroyed = 0;

num\_asteroids\_left = ASTEROIDS\_TO\_DESTROY;

probes = new Thread[PROBE\_COUNT];

num\_probes = PROBE\_COUNT;

last\_time\_update = 0.0;

destroyed = false;

mission\_complete = false;

gui.setNumAsteroids(num\_asteroids\_left);

for(int i = 0; i < num\_probes; i++){

switch(i){

case 0:

probes[i] = new ScoutProbe(i,this);

break;

case 1:

probes[i] = new PhotonTorpedoProbe(i,this);

break;

case 2:

probes[i] = new PhotonTorpedoProbe(i,this);

break;

default:

probes[i] = new PhaserProbe(i,this);

break;

}

probes[i].start();

}

try{Thread.sleep(2000);}

catch(InterruptedException e){}

gui.setStatus("RUNNING");

System.out.println("Command\t\t->\tInitialized");

}

public synchronized void reportAsteroid(Asteroid a){

gui.setProbeStatus(0, "REPORTING\_TARGET");

if(asteroids.isFull()){

System.out.println("Command\t\t->\tScout, Asteroid Was Not Accepted");

System.out.println("Command\t\t->\tFleet Incurred One Hit");

num\_shield\_hits++;

num\_asteroids\_left--;

gui.setShields(max\_shield\_hits - num\_shield\_hits);

if(num\_shield\_hits == max\_shield\_hits){

gui.setStatus("SIMULATION FAILED");

destroyed = true;

}

}else{

if(num\_asteroids\_left <= asteroids.getSize()) gui.setProbeStatus(0, "RETURNING\_TO\_BASE");

else{

asteroids.insert(a);

gui.setStorage(asteroids.getSize());

System.out.println("Command\t\t->\tScout, Asteroid Accepted");

}

}

notify();

gui.setNumAsteroids(num\_asteroids\_left);

}

public synchronized Asteroid getTarget(int i){

gui.setProbeStatus(i, "ACQUIRING\_TARGET");

while(asteroids.getSize() == 0){

if(mission\_complete || destroyed){

notify();

return null;

}

try{wait();}

catch(InterruptedException e){}

}

notify();

gui.setProbeStatus(i, "DESTROYING\_TARGET");

gui.setStorage(asteroids.getSize() - 1);

return asteroids.remove();

}

public void targetDestroyed(int i){

num\_asteroids\_left--;

num\_asteroids\_destroyed++;

gui.setNumAsteroids(num\_asteroids\_left);

if(num\_asteroids\_left <= 0)mission\_complete = true;

}

public void probeSelfDestruct(int i){

num\_probes--;

num\_asteroids\_left--;

num\_asteroids\_destroyed++;

gui.setNumAsteroids(num\_asteroids\_left);

gui.setProbeStatus(i, "DESTROYED");

}

public String giveOrder(int i, String curr\_order){

if(destroyed || mission\_complete){

gui.setProbeStatus(i, "RETURNING\_TO\_BASE");

return "RETURN\_TO\_BASE";

}else{

if(i == 0){

if(curr\_order.equals("ACQUIRE\_NEXT\_TARGET")){

if(destroyed || mission\_complete){

gui.setProbeStatus(i, "RETURNING\_TO\_BASE");

return "RETURN\_TO\_BASE";

}

gui.setProbeStatus(i, "REPORTING\_TARGET");

return "REPORT\_TARGET";

}else{

gui.setProbeStatus(i, "ACQUIRING\_TARGET");

return "ACQUIRE\_NEXT\_TARGET";

}

}else{

if(curr\_order.equals("ACQUIRE\_NEXT\_TARGET")){

if(destroyed || mission\_complete){

gui.setProbeStatus(i, "RETURNING\_TO\_BASE");

return "RETURN\_TO\_BASE";

}

gui.setProbeStatus(i, "DESTROYING\_TARGET");

return "DESTROY\_TARGET";

}else{

gui.setProbeStatus(i, "ACQUIRING\_TARGET");

return "ACQUIRE\_NEXT\_TARGET";

}

}

}

}

}

# A Option – Scout Probe

package asteroidfield;

import static java.lang.Math.log;

import java.util.Random;

/\*\*

\*

\* @author cory

\*/

public class ScoutProbe extends Thread{

private int id;

private Asteroid curr\_asteroid;

private int next\_asteroid\_id;

private int next\_asteroid\_mass;

private double next\_asteroid\_discovered;

private double next\_asteroid\_detection;

private boolean fleet\_destroyed;

private String curr\_order;

private Random rand = new Random();

private PoissonDistribution p\_dist = new PoissonDistribution();

private TaskForceCommand command;

ScoutProbe(int i , TaskForceCommand t){

System.out.println("Scout probe id: " + i);

id = i;

curr\_order = "";

curr\_asteroid = null;

next\_asteroid\_id = 0;

command = t;

}

public void run(){

curr\_order = command.giveOrder(id,curr\_order);

while(curr\_order.equals("ACQUIRE\_NEXT\_TARGET")){

next\_asteroid\_id++;

next\_asteroid\_detection = p\_dist.nextPoisson(4.0) \* 1000;

try{

Thread.sleep(((int)next\_asteroid\_detection));

}catch(InterruptedException e){}

curr\_asteroid = new Asteroid(next\_asteroid\_id, System.currentTimeMillis());

System.out.println("Scout\t\t->\tfound a new asteroid");

curr\_order = command.giveOrder(id,curr\_order);

if(curr\_order.equals("REPORT\_TARGET")){

command.reportAsteroid(curr\_asteroid);

}else break;

curr\_order = command.giveOrder(id, curr\_order);

}

}

}

# A Option – Photon Torpedo Probe

package asteroidfield;

/\*\*

\*

\* @author cory

\*/

public class PhotonTorpedoProbe extends Thread{

private int id;

private int weapon\_fire\_rate;

private int weapon\_damage;

private Asteroid curr\_asteroid;

private double curr\_asteroid\_impact\_time;

private double curr\_asteroid\_destruction\_time;

private String curr\_order;

private TaskForceCommand command;

PhotonTorpedoProbe(int i, TaskForceCommand t){

id = i;

command = t;

weapon\_fire\_rate = 3;

weapon\_damage = 5;

curr\_order = "";

}

public void run(){

curr\_order = command.giveOrder(id, curr\_order);

while(curr\_order.equals("ACQUIRE\_NEXT\_TARGET")){

System.out.println("Probe " + id + "\t\t->\tAcquiring Target");

curr\_asteroid = command.getTarget(id);

if(curr\_asteroid != null){

curr\_asteroid\_destruction\_time = System.currentTimeMillis() + (curr\_asteroid.getMass() / weapon\_damage) \* weapon\_fire\_rate;

System.out.println("Asteroid Impact Time" + ( curr\_asteroid.getTimeOfImpact() - curr\_asteroid\_destruction\_time));

if(curr\_asteroid\_destruction\_time < curr\_asteroid.getTimeOfImpact()){

curr\_order = command.giveOrder(id, curr\_order);

if(curr\_order.equals("RETURN\_TO\_BASE")) break;

while(curr\_asteroid.getMass() > 0){

System.out.println("Probe " + id + "\t\t->\tFiring at Asteroid " + curr\_asteroid.getId());

curr\_asteroid.setMass(curr\_asteroid.getMass() - weapon\_damage);

try{

sleep(weapon\_fire\_rate \* 1000);

}catch(InterruptedException e){

}

}

System.out.println("Probe " + id + "\t\t->\tDestroyed Asteroid " + curr\_asteroid.getId());

command.targetDestroyed(id);

}else{

System.out.println("Probe " + id + "\t\t->\tSelf Destructing");

command.probeSelfDestruct(id);

break;

}

}

curr\_order = command.giveOrder(id, curr\_order);

}

}

}

# A Option – Phaser Probe

package asteroidfield;

import static java.lang.Thread.sleep;

/\*\*

\*

\* @author cory

\*/

public class PhaserProbe extends Thread {

private int id;

private int weapon\_fire\_rate;

private int weapon\_damage;

private Asteroid curr\_asteroid;

private double curr\_asteroid\_impact\_time;

private double curr\_asteroid\_destruction\_time;

private String curr\_order;

private TaskForceCommand command;

PhaserProbe(int i, TaskForceCommand t){

id = i;

command = t;

weapon\_fire\_rate = 2;

weapon\_damage = 3;

curr\_order = "";

}

public void run(){

curr\_order = command.giveOrder(id, curr\_order);

while(curr\_order.equals("ACQUIRE\_NEXT\_TARGET")){

System.out.println("Probe " + id + "\t\t->\tAcquiring Target");

curr\_asteroid = command.getTarget(id);

if(curr\_asteroid != null){

curr\_asteroid\_impact\_time = curr\_asteroid.getTimeDiscovered() + curr\_asteroid.getTimeToImpact();

curr\_asteroid\_destruction\_time = System.currentTimeMillis() + (curr\_asteroid.getMass() / weapon\_damage) \* weapon\_fire\_rate;

if(curr\_asteroid\_destruction\_time < curr\_asteroid\_impact\_time){

curr\_order = command.giveOrder(id, curr\_order);

if(curr\_order.equals("RETURN\_TO\_BASE")) break;

while(curr\_asteroid.getMass() > 0){

System.out.println("Probe " + id + "\t\t->\tFiring at Asteroid " + curr\_asteroid.getId());

curr\_asteroid.setMass(curr\_asteroid.getMass() - weapon\_damage);

try{

sleep(weapon\_fire\_rate \* 1000);

}catch(InterruptedException e){

}

}

System.out.println("Probe " + id + "\t\t->\tDestroyed Asteroid " + curr\_asteroid.getId());

command.targetDestroyed(id);

}else{

System.out.println("Probe " + id + "\t\t->\tSelf Destructing");

command.probeSelfDestruct(id);

break;

}

}

curr\_order = command.giveOrder(id, curr\_order);

}

}

}

# A Option – Asteroid

package asteroidfield;

import java.util.Random;

/\*\*

\*

\* @author cory

\*/

public class Asteroid {

private Random rand;

private int id;

private int mass;

private float mass\_variant;

private double time\_discovered;

private double time\_to\_impact;

private double time\_of\_impact;

Asteroid(int i,double d){

rand = new Random();

id = i;

mass\_variant = rand.nextFloat();

if(mass\_variant <= 0.2){

mass = 3;

}else if(mass\_variant > 0.2 && mass\_variant <= 0.5){

mass = 6;

}else if(mass\_variant > 5 && mass\_variant <= 0.7){

mass = 9;

}else{

mass = 11;

}

time\_discovered = d;

time\_to\_impact = rand.nextDouble() \* 15.0 \* 1000;

time\_of\_impact = time\_discovered + time\_to\_impact;

}

public int getId(){

return id;

}

public void setMass(int m){

mass = m;

}

public int getMass(){

return mass;

}

public double getTimeDiscovered(){

return time\_discovered;

}

public double getTimeToImpact(){

return time\_to\_impact;

}

public double getTimeOfImpact(){

return time\_of\_impact;

}

}

# A Option – Asteroid Simulation GUI

package asteroidfield;

import java.awt.Color;

import java.awt.GridLayout;

import static java.lang.Thread.sleep;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.TimeZone;

import javax.swing.\*;

/\*\*

\*

\* @author cBeezy

\*/

public class AsteroidSimulationGUI extends JFrame{

private int shields;

private int num\_asteroids;

private int num\_probes;

private int storage\_size;

private JPanel header;

private JPanel body;

private JPanel shields\_panel;

private JPanel storage\_panel;

private JPanel probes\_panel;

private JPanel asteroids\_panel;

private JPanel shield\_contents\_panel;

private JPanel storage\_contents\_panel;

private JPanel probe\_contents\_panel;

private JPanel asteroid\_contents\_panel;

private JPanel[] shield;

private JPanel[] storage;

private JPanel[] probes;

private JPanel[] asteroids;

private JLabel sim\_status\_label;

private JLabel sim\_time\_label;

private JLabel shield\_contents\_label;

private JLabel storage\_contents\_label;

private JLabel probe\_contents\_label;

private JLabel asteroid\_contents\_label;

private JLabel[] probe\_labels;

public AsteroidSimulationGUI(int s, int a, int p, int c){

shields = s;

num\_asteroids = a;

num\_probes = p;

storage\_size = c;

setTitle("Asteroid Simulation");

setLayout(null);

setSize(500,500);

setResizable(false);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setBackground(Color.BLACK);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

header = new JPanel();

header.setBounds(0,0,500,50);

header.setBackground(Color.GRAY);

body = new JPanel();

body.setBounds(0,50,500,450);

body.setBackground(Color.GRAY);

add(header);

add(body);

sim\_status\_label = new JLabel("SIMULATION STATUS");

sim\_status\_label.setBackground(Color.BLACK);

sim\_status\_label.setForeground(Color.WHITE);

sim\_time\_label = new JLabel("\tTIME");

sim\_time\_label.setBackground(Color.BLACK);

sim\_time\_label.setForeground(Color.WHITE);

header.add(sim\_status\_label);

header.add(sim\_time\_label);

shields\_panel = new JPanel();

shields\_panel.setBorder(BorderFactory.createLineBorder(Color.GRAY));

shields\_panel.setBounds(5,0,75,400);

shields\_panel.setBackground(Color.GRAY);

storage\_panel = new JPanel();

storage\_panel.setBorder(BorderFactory.createLineBorder(Color.GRAY));

storage\_panel.setBounds(85,0,75,400);

storage\_panel.setBackground(Color.GRAY);

probes\_panel = new JPanel();

probes\_panel.setBorder(BorderFactory.createLineBorder(Color.GRAY));

probes\_panel.setBounds(165,0,210,400);

probes\_panel.setBackground(Color.GRAY);

asteroids\_panel = new JPanel();

asteroids\_panel.setBorder(BorderFactory.createLineBorder(Color.GRAY));

asteroids\_panel.setBounds(380,0,100,400);

asteroids\_panel.setBackground(Color.GRAY);

body.setLayout(null);

body.add(shields\_panel);

body.add(storage\_panel);

body.add(probes\_panel);

body.add(asteroids\_panel);

shields\_panel.setLayout(null);

shield\_contents\_label = new JLabel("Sheilds");

shield\_contents\_label.setBounds(5,5,65,20);

shield\_contents\_label.setHorizontalAlignment(JLabel.CENTER);

shield\_contents\_label.setForeground(Color.WHITE);

shield\_contents\_panel = new JPanel();

shield\_contents\_panel.setBounds(5,25,65,370);

shield\_contents\_panel.setBackground(Color.GRAY);

shields\_panel.add(shield\_contents\_label);

shields\_panel.add(shield\_contents\_panel);

storage\_panel.setLayout(null);

storage\_contents\_label = new JLabel("Storage");

storage\_contents\_label.setBounds(5,5,65,20);

storage\_contents\_label.setHorizontalAlignment(JLabel.CENTER);

storage\_contents\_label.setForeground(Color.WHITE);

storage\_contents\_panel = new JPanel();

storage\_contents\_panel.setBounds(5,25,65,370);

storage\_contents\_panel.setBorder(BorderFactory.createLineBorder(Color.WHITE));

storage\_contents\_panel.setBackground(Color.GRAY);

storage\_panel.add(storage\_contents\_label);

storage\_panel.add(storage\_contents\_panel);

probes\_panel.setLayout(null);

probe\_contents\_label= new JLabel("Probes");

probe\_contents\_label.setBounds(5,5,200,20);

probe\_contents\_label.setHorizontalAlignment(JLabel.CENTER);

probe\_contents\_label.setForeground(Color.WHITE);

probe\_contents\_panel = new JPanel();

probe\_contents\_panel.setBounds(5,25,200,370);

probe\_contents\_panel.setBackground(Color.GRAY);

probes\_panel.add(probe\_contents\_label);

probes\_panel.add(probe\_contents\_panel);

asteroids\_panel.setLayout(null);

asteroid\_contents\_label = new JLabel("Asteroids");

asteroid\_contents\_label.setBounds(5,5,90,25);

asteroid\_contents\_label.setHorizontalAlignment(JLabel.CENTER);

asteroid\_contents\_label.setForeground(Color.WHITE);

asteroid\_contents\_panel = new JPanel();

asteroid\_contents\_panel.setBounds(5,25,90,370);

asteroid\_contents\_panel.setBackground(Color.GRAY);

asteroids\_panel.add(asteroid\_contents\_label);

asteroids\_panel.add(asteroid\_contents\_panel);

shield\_contents\_panel.setLayout(new BoxLayout(shield\_contents\_panel, BoxLayout.Y\_AXIS));

shield = new JPanel[shields];

for(int i = 0; i < shields; i++){

shield[shields - 1 - i] = new JPanel();

shield\_contents\_panel.add(shield[shields - 1 - i]);

}

storage\_contents\_panel.setLayout(new BoxLayout(storage\_contents\_panel,BoxLayout.Y\_AXIS));

storage = new JPanel[storage\_size];

for(int i = 0; i < storage\_size; i++){

storage[storage\_size - 1 - i] = new JPanel();

storage\_contents\_panel.add(storage[storage\_size - 1 - i]);

}

probe\_contents\_panel.setLayout(new GridLayout(num\_probes,1,0,5));

probes = new JPanel[num\_probes];

probe\_labels = new JLabel[num\_probes];

for(int i = 0; i < num\_probes; i++){

probes[i] = new JPanel();

if(i == 0)probe\_labels[i] = new JLabel("Scout Probe");

else probe\_labels[i] = new JLabel("Defensive Probe # " + (i - 1));

probe\_labels[i].setForeground(Color.WHITE);

probe\_contents\_panel.add(probes[i]);

probes[i].add(probe\_labels[i]);

}

asteroid\_contents\_panel.setLayout(new GridLayout(num\_asteroids/2, 2, 5, 5));

asteroids = new JPanel[num\_asteroids];

for(int i = 0; i < num\_asteroids; i++){

asteroids[i] = new JPanel();

asteroid\_contents\_panel.add(asteroids[i]);

}

setShields(shields);

setNumAsteroids(num\_asteroids);

for(int i = 0; i < num\_probes; i++)setProbeStatus(i, "AWAITING\_ORDER");

setVisible(true);

}

public void setTime(long t){

String time = "";

DateFormat df = new SimpleDateFormat("HH:mm:ss");

df.setTimeZone(TimeZone.getTimeZone("CST"));

sim\_time\_label.setText("at " + df.format(new Date(t)));

}

public void setStatus(String s){

sim\_status\_label.setText("Status: " + s);

}

public void setShields(int s){

Color c;

double shield\_amount = (double) s / (double) shields;

if(shield\_amount >= 1.0) c = Color.GREEN;

else if(shield\_amount < 1.0 && shield\_amount >= 0.66) c = Color.YELLOW;

else if(shield\_amount < 0.66 && shield\_amount >= 0.33) c = Color.ORANGE;

else c = Color.RED;

for(int i = 0; i < shields; i++){

if(i < s) shield[i].setBackground(c);

else shield[i].setBackground(null);

}

}

public void setStorage(int num\_items){

Color c;

double fill\_amount = (double)num\_items / (double)storage\_size;

if(fill\_amount >= 1.0) c = Color.RED;

else if(fill\_amount < 1.0 && fill\_amount >= 0.66) c = Color.ORANGE;

else if(fill\_amount < 0.66 && fill\_amount >= 0.33) c = Color.YELLOW;

else c = Color.GREEN;

for(int i = 0; i < storage\_size; i++){

if( i < num\_items) storage[i].setBackground(c);

else storage[i].setBackground(null);

}

}

public void setNumAsteroids(int num\_left){

for(int i = 0; i < num\_asteroids; i++){

if( i < num\_left)asteroids[i].setBackground(Color.RED);

else asteroids[i].setBackground(Color.GREEN);

}

}

public void setProbeStatus(int i, String s){

if(s.equals("AWAITING\_ORDER"))probes[i].setBackground(Color.YELLOW);

else if(s.equals("ACQUIRING\_TARGET"))probes[i].setBackground(Color.ORANGE);

else if(s.equals("REPORTING\_TARGET"))probes[i].setBackground(Color.BLUE);

else if(s.equals("DESTROYING\_TARGET"))probes[i].setBackground(Color.RED);

else if(s.equals("DESTROYED"))probes[i].setBackground(Color.BLACK);

else if(s.equals("ABANDONING\_MISSION"))probes[i].setBackground(Color.MAGENTA);

else if(s.equals("RETURNING\_TO\_BASE"))probes[i].setBackground(Color.GREEN);

}

}

# A Option – Main

package asteroidfield;

import static java.lang.Thread.sleep;

/\*\*

\*

\* @author cory

\*/

public class AsteroidField {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

// TODO code application logic here

int max\_storage = 15;

int max\_shields = 5;

int num\_asteroids = 55;

int num\_probes = 4;

TaskForceCommand command = new TaskForceCommand();

command.Init(max\_storage, max\_shields, num\_asteroids, num\_probes);

command.start();

}

}