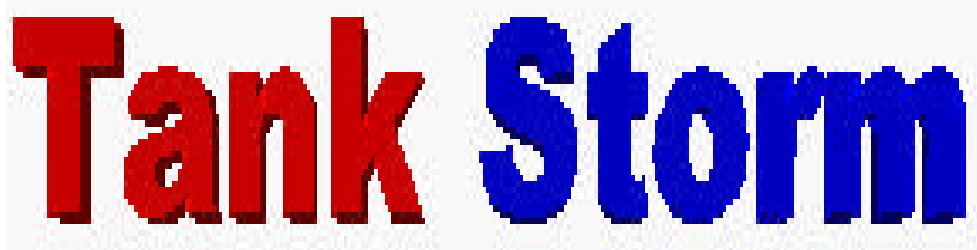


Design Manual

*Computer Applications 3rd Year Project by
Mark Cunningham and Mark Greene*



Tank Storm

A Computer Game

Project Supervisor: Howard Ducan
Project Assessor: Alex Monaghan

(Please Refer to User Manual)

Table of Contents

1. Overall Design
2. Graphics
3. Collision and Motion
4. Artificial Intelligence
5. User Input
6. Performance
7. Miscellaneous
8. Code Listings and Notes
9. References

Overall Design

In this section:

A brief look at Java programming language.
Overall Design and Structure of our Project.

A brief look at Java programming language

(For more information on Java, please refer to any book or web resource on Java, the purpose of this is to give a brief outline of the Java as a programming language.)

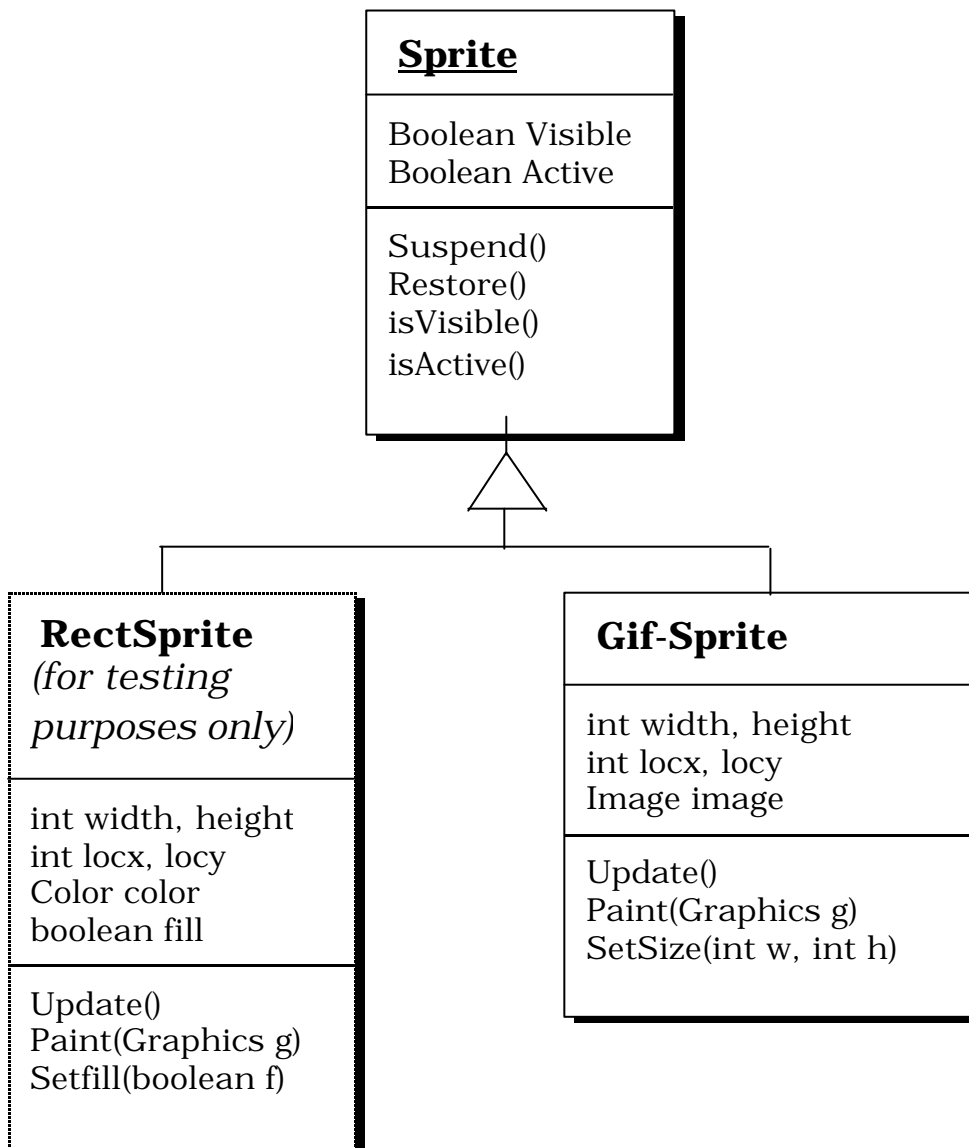
Java is the latest fashionable programming language to come out in recent years. One of the main reasons for this popularity is that it can be used over the web and is fully portable over all operating systems. Meaning that the Internet is no longer a place of still graphics and text but interaction and usefulness and also it doesn't matter what Computer or Operating System you have just as long as you can access the web you can use Java.

On a technical side, Java uses the standard syntax of C and C++ but properly implements classes. Instead of main() and loads of functions, Java code is split into separate distinct classes that other classes can use without knowing the implementation of those classes. Java programs are either Applets or Applications. Applets are held in Web Pages (using HTML tags to call them) and Applications are ran by using Java State Machines.

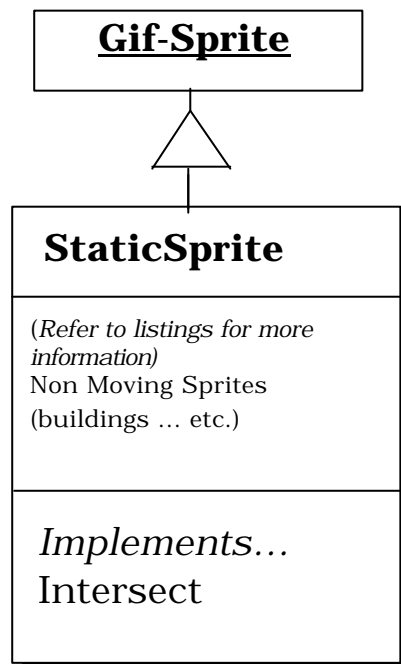
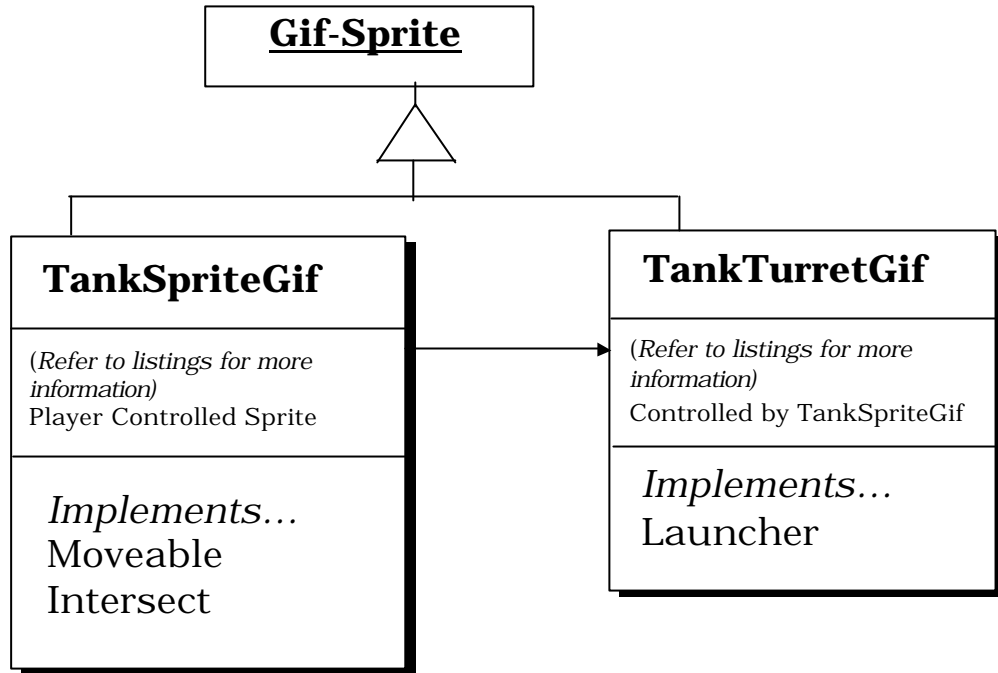
On the downside Java is slow. It is compiled into byte streams which is recompiled by Internet Browsers (such as **Microsoft Internet Explorer** or **Netscape**) into native code and ran or by Java State Machines (such as **JDK** development tools). This is a major factor influencing our project. Also there are inconsistencies between Operation Systems and Browsers, such as the implementation of threads as each Operation System has it's own time slicing for processes. The *work arounds* will be discussed as we come across them.

Overall Design and Structure of our Project.

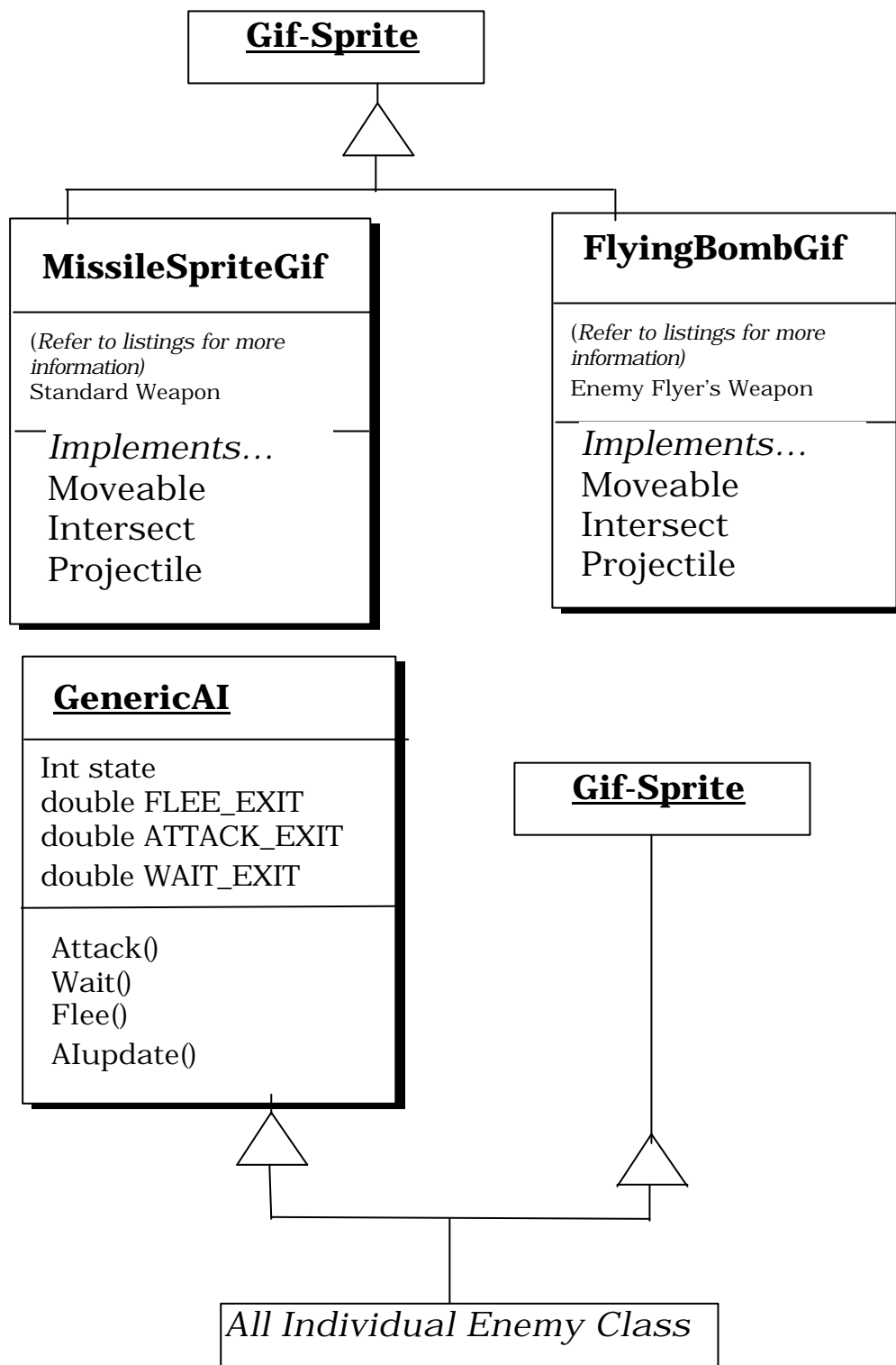
Class Hierarchy



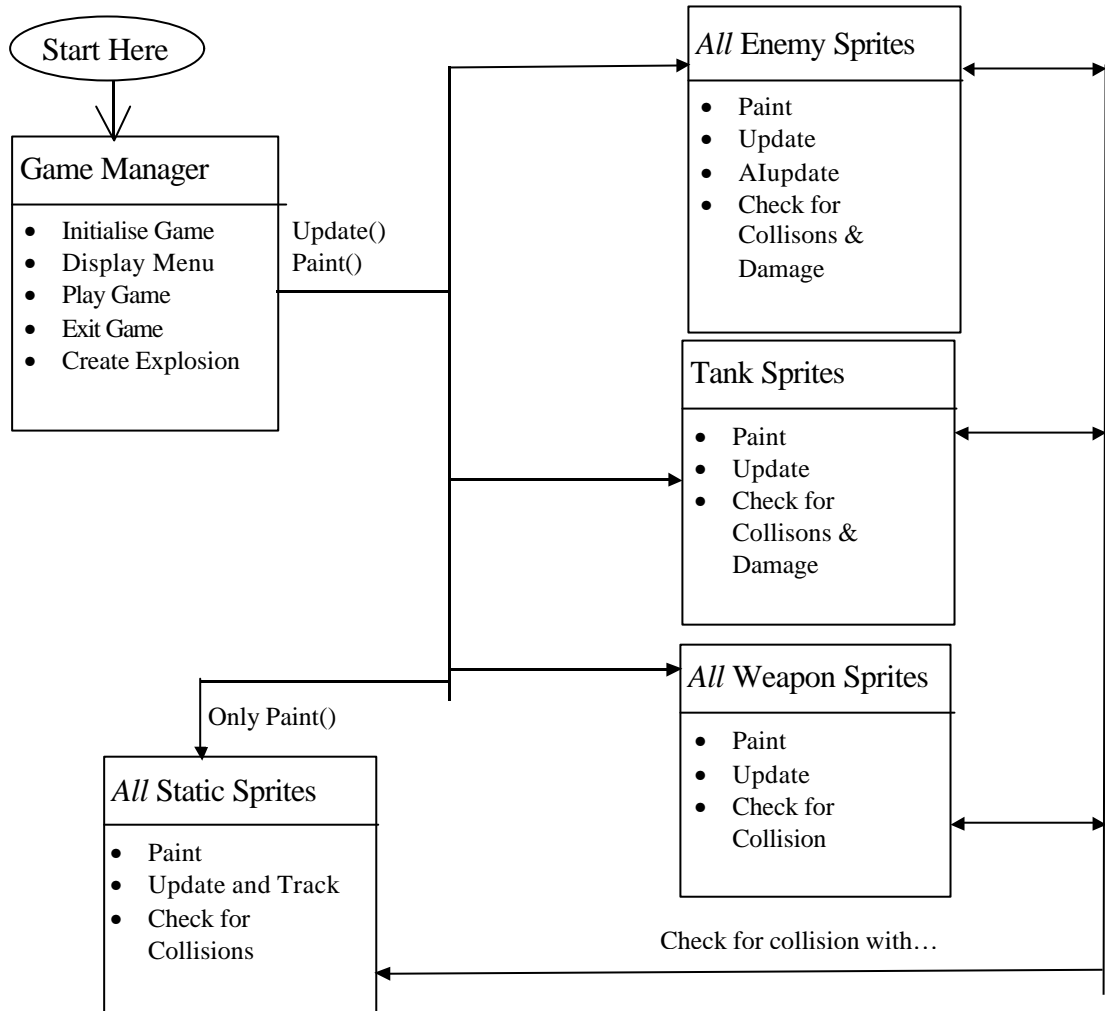
Continued on next page...



Continued on next page...



Brief Outline of Class interaction...



Complete Class List:

Abstract Classes:

- * Sprite (basic sprite class)
- * GifSprite (for picture (.gif) support)
- * GenericAI (for that important intelligence)

Instances of classes

- * TankSpriteGif (tank sprite)
- * TankTurretGif (tank's turret)
- * StaticSprite (buildings)
- * EnemyHover (enemy hover tank)
- * EnemyProducer (enemy producer)
- * EnemyFlyer (enemy flyer)
- * PBot (enemy producer bots)
- * MissileSpriteGif (standard weapon)
- * FlyingBombGif (enemy flyer's weapon)
- * EnemyTurret (enemy turret)

Interfaces (see Section below)

- * Moveable (for sprites that move)
- * Intersect (for collisions)
- * Projectile (for objects that are shot by other objects)
- * Launcher (for objects that shot other objects)

Other Classes

- * GameManager (applet implements thread: manages game)

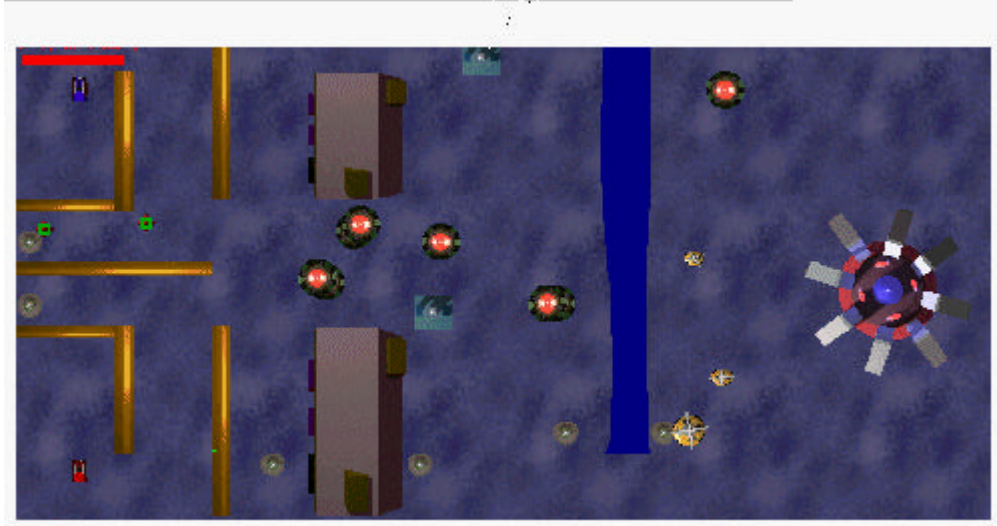
For information on each class refer to relevant sections or code listings.

A note on interfaces: Java does not support multiple class inheritance, so there is interfaces in Java. While we didn't necessarily use them for multiple inheritance they provide a certain useful abstractness. An object could hold a pointer to another object of type interface no matter what class it inherited. So for example an EnemyHover could hold an array of interface projectile without ever worrying what type of projectile it is. This gave the game design added flexibility.

| | |
|-------------------|---------------------------|
| Compiler: | JDK1.1.1 |
| Operating System: | Windows NT and Windows 95 |
| Viewer: | Internet Explorer 3 |

Other Software used: Lightwave 3D, Paint Shop Pro, GifConverter, Piclab, Giftrans.

Graphics



The one of the most important parts of a game is the graphics. How smooth is it? is it fast enough? Does it look good?

Java provides usefully libraries that deal with images well unfortunately because Java is slow it provides additional problems.

One of the first problems is flicker. An animation is a sequence of images that change slightly one of the other and then show in sequence very quickly. In Java, to draw an image is very simple:
`g.drawImage(image,x,y,this);`

To do animation then would be to clear the screen and redraw the next image. But this doesn't work well, it creates flicker, the screen being redrawn then the image is still noticeable by the eye. Part of the reason is that when update is called in the applet it clears the screen. So just override update():

```
public void update(){
    paint(g);
}
```

Also a more efficient improvement is double-buffering. A separate graphics context is used, everything is drawn to it and then it is drawn to the screen.

A sample of code from the GameManager Applet...

```

public class GameManager extends Applet implements Runnable{

    //A Single thread for animation and game
    Thread animation;
    //for double buffering & background
    Graphics offscreen;
    Image image, background;

    public void init() {

        showStatus("Initilising Applet");
        setBackground(Color.black);
        width = bounds().width;
        height = bounds().height;

        //create double buffer
        image = createImage(width,height);
        offscreen = image.getGraphics();

        //load title screen and related images
        loading = true;
        loadTitle();
        loading = false;
    }

    //repaint()
    public void paint(Graphics g) {

        //offscreen is the double buffer
        //graphics context

        //if game paint level
        if(playing){
            paintGame(offscreen);
        }
        else if(endgame){
            paintEndGame(offscreen);
        }
        else { //else display option and title
            //screens
            paintIntro(offscreen);
        }

        //paint double buffer to screen
        g.drawImage(image,0,0,this);
    }
}

```

Another point worth noting is that for an animation to run smoothly, a simple while loop in run won't really be affective. The Applet must be implemented as a thread. From the sample code above the class definition of GameManager *implements Runnable*, which is allowing GameManager to be used as a thread.

To animate the sprites themselves, they have to hold an array of images and every time update is called, increment the counter and next paint, paint the next image.

Sample Code from EnemyHover...

```
public class EnemyHover extends GenericAI implements Moveable, Intersect, Launcher{

    protected Image images[ ];
    protected int currentImage; //keeps track of images

    public void paint(Graphics g){
        for(int i = 0; i < ammo.length; i++){
            ((Sprite)ammo[i]).paint(g);
        }
        if(visible){
            g.drawImage(images[currentImage],locx,locy,applet);
            /* -- for testing and design
            g.setColor(Color.black);
            g.drawRect(patrolAera.x,patrolAera.y,patrolAera.width,patrolAera.height);
            */
        }
    }

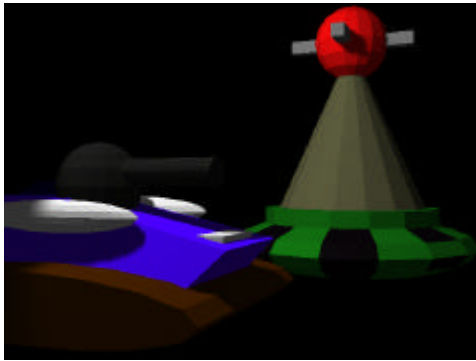
    public void update(){

        //see class file for full listing...

        //update images for animation
        if(currentImage == 19){
            currentImage = 0;
        }
        else {
            currentImage++;
        }
    }
}
```

An interesting advantage of this is that in the TankSpriteGif class to make sure the sprite appears to be pointed in the right direction to which it is moving, when the user's input turns the Tank Sprite, only then is currentImage increment or decrement (depending on whether it's left or right).

It is also important that the sprites look good. So we used a 3D editor called **Lightwave 3D** to create the sprites as simple objects and then animate them moving or turning or firing. This allowed us a lot of freedom creating sprites and taught us some thing about 3D design.



It's also nice to have the game running on a nice background. One way and the way we did it was to have a tiled background. Take a single image and draw it as many times as will fit onto the screen. If you look at the sample code from GameManager above, you will see an image defined called background. On this image is the tiled background. And when paintGame is called, instead of clearing the buffer, it's covered with the tiled background in the background image.

The code below is how the tiled background is created. The createImage method is used to create an image of the right width and height while getGraphics() is used to allow us to draw directly to the image.

```
//creates a tiled image the size of the applet
private Image createTiled(Image tile){

    Image base = createImage(width,height);
    for(int i = 0; i < width; i += tile.getWidth(this)){
        for(int j = 0; j < height; j += tile.getHeight(this)){
            base.getGraphics().drawImage(tile, i, j, this);
        }
    }
    return base;
}
```

One last thing is that Java likes to load images on demand only. This unfortunately is not an advantage for the game, it's not great when your fighting a boss that's only half there. So the only alternative is to load all the images initially with MediaTracker (A Java class that watches the loading of images). This creates a performance overhead but one that improves the look of the game overall.

More Sample code from GameManager showing MediaTracker in action.

```
//loads initial game screens, including title and options and explosions
private void loadTitle(){
```

```
    Image exp[] = new Image[9];
    Image lexp[] = new Image[9];
```

```
    MediaTracker t = new MediaTracker(this);
    title = getImage(getCodeBase(),"intro/title.gif");
    t.addImage(title,0,width,height);
    for(int i = 0; i < 9; i++){
        exp[i] = getImage(getCodeBase(),"images/Texp"+i+".gif");
        t.addImage(exp[i],0);
        lexp[i] = getImage(getCodeBase(),"images/TLexp"+i+".gif");
        t.addImage(lexp[i],0);
    }
    opt = getImage(getCodeBase(),"intro/options.gif");
    t.addImage(opt,0);
    icon = getImage(getCodeBase(),"intro/icon.gif");
    t.addImage(icon,0);
    onewin = getImage(getCodeBase(),"intro/P1Win.gif");
    t.addImage(onewin,0);
    twowin = getImage(getCodeBase(),"intro/P2Win.gif");
    t.addImage(twowin,0);
    lose = getImage(getCodeBase(),"intro/PLose.gif");
    t.addImage(lose,0);
    allwin = getImage(getCodeBase(),"intro/PWin.gif");
    t.addImage(allwin,0);
    endScrn = getImage(getCodeBase(),"intro/end.gif");
    t.addImage(endScrn,0);
```

```
    showStatus("Loading Images -- Please Wait!");
```

```
    try { //waits for all images to load and signals
        // if they were loaded with/with out errors.
        t.waitForAll();
    } catch (InterruptedException e){ return; }
```

```
    if (t.isErrorAny()){
        showStatus("Error Loading Images");
    }
    else if (t.checkAll()){
        showStatus("Successfully loaded.");
    }
}
```

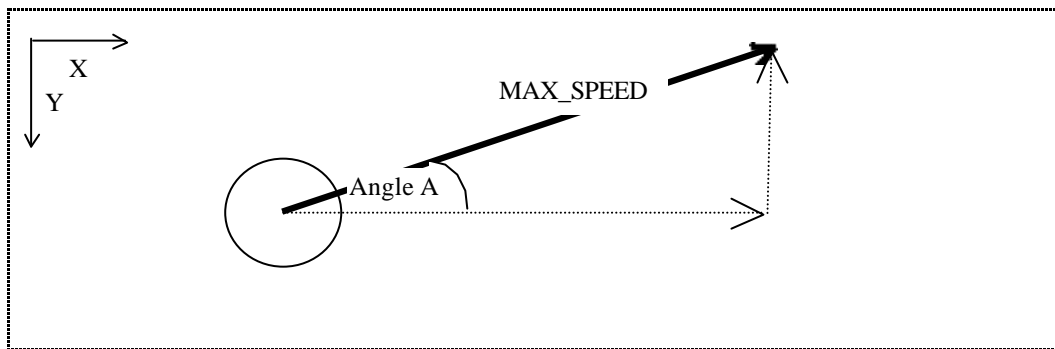
```
//etc.
```

Collision and Motion

Another important part of our game is how sprites move and collide. Enemies bounce into buildings and other enemies, missiles collide with targets, tanks move forward. Movement and from that collision is necessary and complicated.

A little Maths...

If a sprite has motion it has direction. Unfortunately it's not simple to just say move there, it's direction and speed must be calculated in terms of vectors in the x and y direction. I.e. it has a relative speed in the x direction and another relative speed in the y direction.



From the diagram about- using some trigonometry it's x speed would be the $\text{Cos}(\text{angle } A) * \text{MAX_SPEED}$ and the y speed would be the $\text{Sin}(\text{angle } A) * \text{MAX_SPEED}$. This vector idea of different speeds in x and y is important.

Lets look at one of interfaces: Moveable

```
interface Moveable {
    public abstract void setPosition(int x, int y);
    public abstract void setVelocity(int x, int y);
    public abstract void updatePosition();
    public abstract void setVx(int x);
    public abstract void setVy(int y);
}
```

(this code was taken and modified, from the Black Art Of Java Game Programming: See references at end of manual).

And here a sample implementation of it, from EnemyHover

```

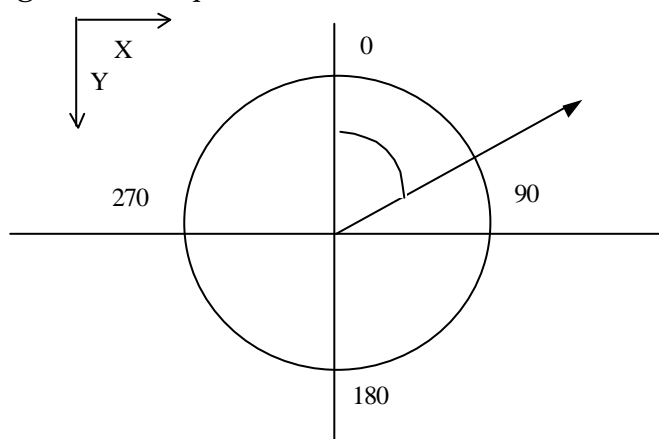
//implement Moveable interface (setPosition, setVelocity and updatePosition)
int vx; //stores velocity in x direction
int vy; //stores velocity in y direction
public void setPosition(int x, int y){
    locx = x; //set x position
    locy = y; //set y position
}
public void setVelocity(int x, int y){
    vx = x; //set velocities
    vy = y;
}
public void updatePosition(){
    locx += vx; //move the sprite
    locy += vy;
}
//set velocities individually
public void setVx(int x){
    vx = x;
}
public void setVy(int y){
    vy = y;
}

```

Fairly simple really. We set the initial vx and vy and then in update call update position, all we then need is too make sure it doesn't go off the boundaries. But what if we want to give it a direction and a speed, not the x-speed or y-speed? As in TankSpriteGif which is controlled by the player or the MissileSpriteGif.

The launcher interface allows a assessor method called getAngle, which returns a int from 0 to 360. This is what MissileSpriteGif uses to calculate it's direction and speed.

First it calculates the angle in radians and then using some Trigonometry from the leaving cert., calculates the x and y speeds depending on what quadrant it's in.



Sample code from MissileSpriteGif

```

//calculate realtive velocity to the angle of the turret
protected void setVel() {

```

```

//calculates correct angle (getAngle supplied by interface launcher)
float angle = launcher.getAngle();
//angle in radions
float rad_angle = (float)((angle/180) * Math.PI);

//calculate velocity
if(angle == 0){
    setVelocity(0,-MAX_SPEED);
}
else if(angle == 90){
    setVelocity(MAX_SPEED,0);
}
else if(angle == 180){
    setVelocity(0,MAX_SPEED);
}
else if(angle == 270){
    setVelocity(-MAX_SPEED,0);
}
else if(angle>0 && angle<90){
    setVx((int)(Math.sin(rad_angle)*MAX_SPEED));
    setVy((int)(-(Math.cos(rad_angle)*MAX_SPEED)));
}
else if(angle>90 && angle<180){
    setVx((int)(Math.cos(rad_angle-(Math.PI/2))*MAX_SPEED));
    setVy((int)(Math.sin(rad_angle-(Math.PI/2))*MAX_SPEED));
}
else if(angle>180 && angle<270){
    setVx((int)(-(Math.sin(rad_angle-Math.PI)*MAX_SPEED));
    setVy((int)(Math.cos(rad_angle-Math.PI)*MAX_SPEED));
}
else if(angle>270 && angle<360) {
    setVx((int)(-(Math.cos(rad_angle-(1.5*Math.PI))*MAX_SPEED));
    setVy((int)(-(Math.sin(rad_angle-(1.5*Math.PI))*MAX_SPEED));
}
}
}

```

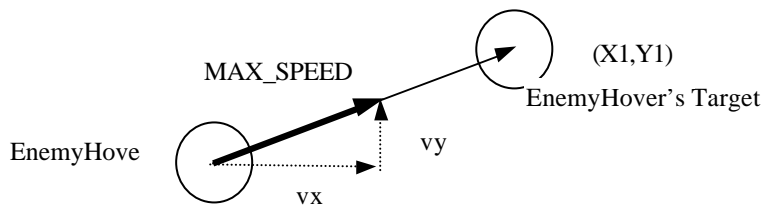
There is a slight complication in TankSpriteGif but it's easily resolved. The tank must be able to rotate, so it's direction must be calculated to it's relative angle. But the rotating depends on the number of frames of animation, of course the animation must be symmetrically, if it's 20 frames, then at 0 it's angle 0 and at frame 10 it's angle is 180.

```

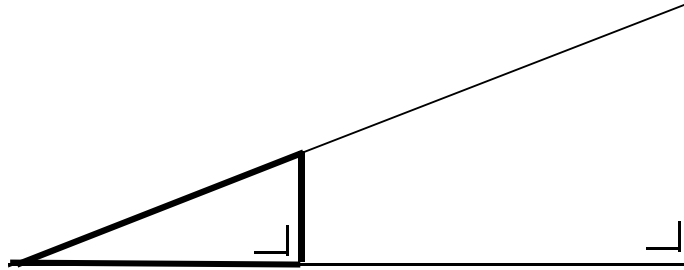
//calculates correct angle
float temp = (360/images.length);
angle = currentImage * temp;
//angle in radions
float rad_angle = (float)((angle/180) * Math.PI);

```

There is one other motion calculation that should be mentioned before we move on to collisions. This is really part of Artificial Intelligence section, but EnemyHover when in attack state, moves towards it's target. To calculate the correct direction and then relative speeds it uses the intersect interface which contains two methods getLocx() and getLocy() which returns the x and y position of the targets.



So how do we calculate EnemyHover's vx and vy. Using this formula of ratios for triangles...



The Smaller triangle will have all it's sides in the same ratio to the larger triangle.

This means that if we can calculate the x distance, the y distance and then the ratio of max_speed to actually distance, we can work out the x speed and the y speed.

We have the x distance and y distance from getLocx and getLocy, we have EnemyHover's max_speed.

To find the actual distance, use pythagorous... $x^2 + y^2 = (\text{actually distance})^2$.

Then the actual distance = $\sqrt{x^2 + y^2}$.

The ratio is max_speed/actual distance.

And therefore the velocity in the x direction is equal to x * ratio and the y velocity is y * ratio.

```
//get x and y distances
double diffx = (double)(targets[target_attack].getLocx() - locx);
double diffy = (double)(targets[target_attack].getLocy() - locy);
//square them and then square root them to find the actual distance
double xSq = Math.pow(diffx,2.0);
double ySq = Math.pow(diffy,2.0);
double dist = Math.pow(ySq+xSq,0.5);
//calculate the ration
double ratio = (double)speed/dist;
//calculate x and y velocities
setVx((int)(diffx*ratio));
setVy((int)(diffy*ratio));
```

Collisions are dealt with by using the Intersect interface.

```

interface Intersect {
    public boolean intersect(int x1, int y1, int x2, int y2);
    public void hit(int Damage);
    public int getDamage();
    public int getLocx();
    public int getLocy();
    public int getHeight();
    public int getWidth();
}

```

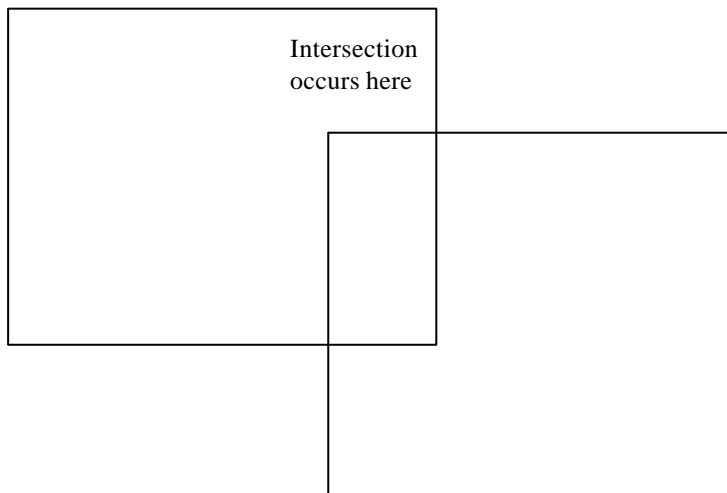
Sample implementation...

```

//implement Intersect interface (intersect and hit)
public boolean intersect(int x1,int y1,int x2, int y2){
return visible && (x2 >= locx) && (locx+width >= x1) && (y2 >= locy) && (locy+height >= y1);
}
public void hit(int Damage){
    energy -= Damage;
    flee = true;
}
public int getDamage(){ return damage;}
public int getLocx(){ return locx; }
public int getLocy(){ return locy; }
public int getWidth(){ return width; }
public int getHeight(){ return height; }

```

The most important function from this is the boolean intersect(int x1, int y1, int x2, int y2). It checks for the intersection of the given co-ordinates and it's own bounding box.



If we wanted to be advanced we could have once a collision is detected check for pixel collision, because what if the sprite or object isn't square? We decided against that due to the performance overhead of the calculation.

This detect collision is grand for missiles hitting targets but what about bouncing off? First step is how to make an object or sprite bounce realistically off the boundaries of the screen or it's area?

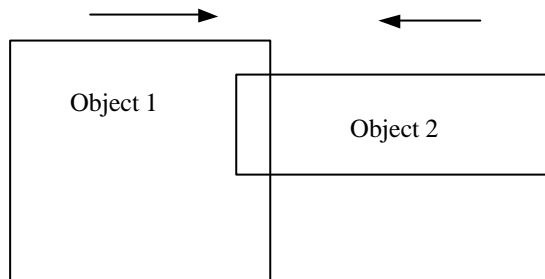
```

//bounce off boundaries
if((locx+width+6 > (patrolAera.x+patrolAera.width) && vx > 0) || (locx-6 < patrolAera.x && vx < 0)){
    vx = -vx;
}
if((locy+height+6 > (patrolAera.y+patrolAera.height) && vy > 0) || (locy-6 < patrolAera.y && vy < 0)){
    vy = -vy;
}

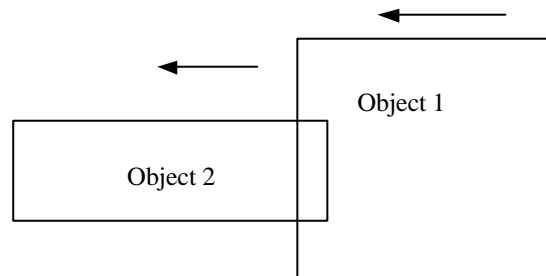
```

Reverse the direction once it crosses the boundary, but you have to make sure it's head to cross the boundary. Otherwise if it is AT the boundary and tries to reverse away the speed would be reversed and it would actually be pulled outside of it's boundaries.

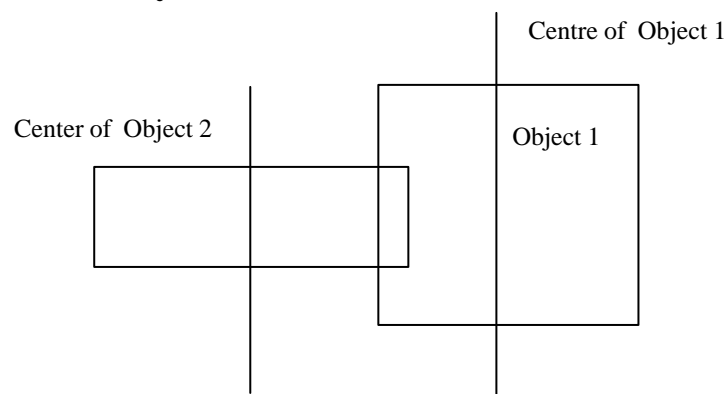
Another complication exists if two objects collide, such as TankSpriteGif and EnemyHover?? If we try to use the code above just modified a strange bug occurs.



Two objects collide. We test to see if object 2's front passes object 1's front, if so reverse direction. That makes sense but what happens if this occurs?



We test to see if the front of object 2 passes object 1's front and then reverse it's direction... bug... object 2 is then forced into object 1 even though they aren't meant to be allowed to move into each other. There is a way around this.



Object 2

We test to see if the right end of object 2 is inside the area bounded by object 1's right side and centre line. The other condition is to test if it's left end of object 2 is in the area bounded by object 1's centre and it's right side. And only if those conditions are satisfied do we reverse direction.

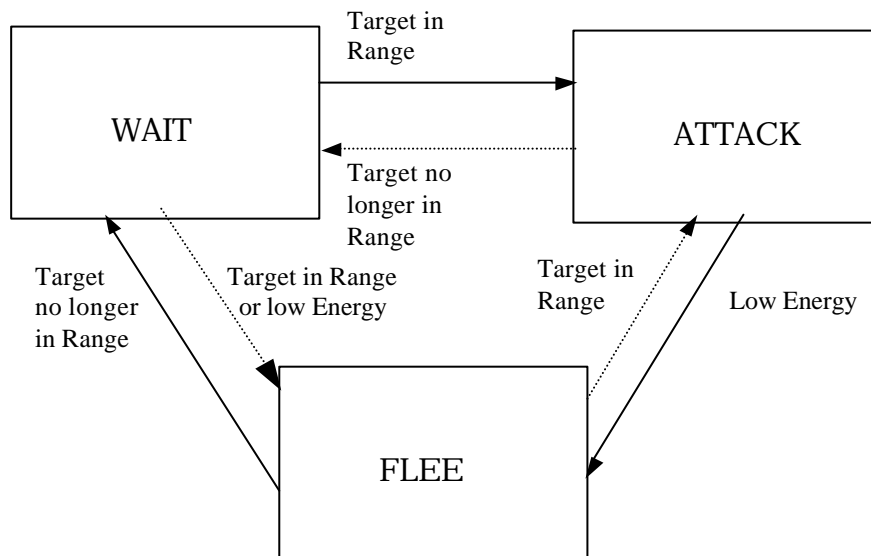
```
for(int i = 0; i < statics.length; i++){  
  
    if(statics[i] != null){  
        if(((Sprite)statics[i]).isVisible()){  
            if(statics[i].intersect(locx,locy,locx+width,locy+height)){  
  
                statics[i].hit(damage);  
                energy = energy - statics[i].getDamage();  
  
                //get Obstacle x info  
  
                int tempPos, tempWidth, tempCentre;  
                tempPos = statics[i].getLocx();  
                tempWidth = statics[i].getWidth();  
                tempCentre = tempPos+(tempWidth/2);  
  
                //testing in the x direction  
                if(((locx+width)>tempPos && (locx+width)<tempCentre && vx > 0)  
                || (locx <(tempPos+tempWidth) && locx>tempCentre && vx < 0)){  
                    vx = -vx;  
                }  
  
                //get Obstacle y info  
                tempPos = statics[i].getLocy();  
                tempWidth = statics[i].getHeight();  
                tempCentre = tempPos+(tempWidth/2);  
  
                //testing in the y direction  
                if(((locy+height)>tempPos && (locy+height)<tempCentre && vy > 0)  
                || (locy <(tempPos+tempWidth) && locy>tempCentre && vy < 0)){  
                    vy = -vy;  
                }  
            }  
        }  
    }  
}
```

Artificial Intelligence

Artificial intelligence in relation to a game is how well sprites react to the player, where they attack, ignore or run away from the player.

Looking at other games of the same type, the enemy sprites behaviour can be described as existed in one of three states, wait (or patrol), attack and flee (or retreat).

The change in states can then be conditions, enemy in range, move to attack, too little energy, flee (or attack), enemy has moved out of range, wait etc.



After developing this model, we tried to create a generic class for the AI. With reference to *The Black Art Of Java Game Programming*, we came up with this `AIupdate()` method.

```
public void AIupdate(){  
  
    double r1 = Math.random();  
    double r2 = Math.random();  
    switch (state){  
  
        case WAIT:  
            if(r1 > WAIT_EXIT){  
                if(r2 > 0.5 && FleeCondition()){  
                    Flee();  
                    state = FLEE;  
                }  
            }  
        }  
    }  
}
```

```

        }
        else if (r2 < 0.5 && AttackCondition()){
            Attack();
            state = ATTACK;
        }
    }
    else {
        Wait();
    }
break;
case ATTACK:
    if(r1 > ATTACK_EXIT){
        if(r2 > 0.5 && FleeCondition()){
            Flee();
            state = FLEE;
        }
        else if (r2 < 0.5 && WaitCondition()){
            Wait();
            state = WAIT;
        }
    }
    else{
        Attack();
    }
break;
case FLEE:
    if(r1 > FLEE_EXIT){
        if(r2 > 0.5 && WaitCondition()){
            Wait();
            state = WAIT;
        }
        else if (r2 < 0.5 && AttackCondition()){
            Attack();
            state = ATTACK;
        }
    }
    else {
        Flee();
    }
break;
}
}

```

The Basic idea is that an enemy is in one of the particular states and depending on random variable and conditions it will leave that state depending on a probability that can be set. Lets take a closer look at one of the case statements.

```

case ATTACK:
    if(r1 > ATTACK_EXIT){
        if(r2 > 0.5 && FleeCondition()){
            Flee();
            state = FLEE;
        }
        else if (r2 < 0.5 && WaitCondition()){
            Wait();
            state = WAIT;
        }
    }
    else{
        Attack();
    }

```

```

    }
    break;

```

r1 and r2 are two integer values containing random value between 0 and 1 (Math.random()). ATTACK_EXIT is a value between 0 and 1 as well, basically the probability it will leave this state. If it will leave the state, it tests if it should FLEE or WAIT. FleeCondition and WaitCondition are two abstract methods that the enemy implements, so for example, I have too little energy make FleeCondition true. It can now move to flee condition. Attack(), Flee() and Wait() are implementations of the 3 states.

EnemyHover implements all the states...

```

//implement methods from generic AI
boolean attack = false;
boolean wait = false;
boolean flee = false;
public boolean AttackCondition() {return attack;}
public boolean WaitCondition() {return wait;}
public boolean FleeCondition() {return flee;}

public void Attack() {
    //shoot at target
    int cnt = 0;
    for(;(cnt < ammo.length) && ((Sprite)ammo[cnt]).isVisible(); cnt++){ }
    if(cnt < ammo.length && Math.random() > 0.5){
        ammo[cnt].fire();
    }

    speed = MX_speed;//speed up
    fixSpeed(); //move towards target
}

public void Wait() {
    //slow down and do nothing
    speed = NM_speed;
}

public void Flee() {
    //speed up and move away from target
    speed = MX_speed;
    fixSpeed();
    vx = -vx;
    vy = -vy;
}

public void update(){

//set class file for full listing

//set states accordingly
if(target_attack != -1){
    //if target in range
    attack = true;
    flee = true;
    wait = false;
}
else {
    //else no target in range

```

```
        attack = false;  
        flee = false;  
        wait = true;  
    }  
    if(energy < 10){  
        //if energy low  
        flee = true;  
    }  
  
    AIupdate();  
  
}
```

Due to the fact that Java has no multiply inheritance, we made GenericAI extend GifSprite so that enemy sprites could inherit from both classes.

User Input

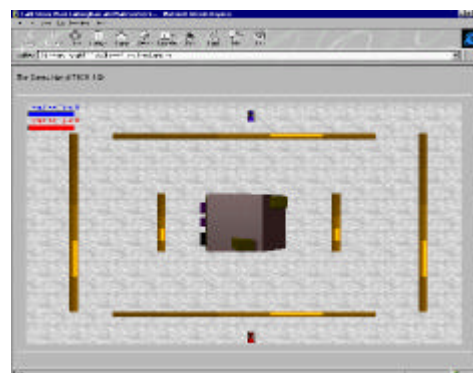
Here we come across another hurdle of the Java language. There is unfortunately no really good viewer for Applets or applications. Hopefully this situation will be remedied with the release of JavaSoft's runtime performance pack (Just In Time Compilers) for Win32 and NT and a Solaris Pack. Unfortunately they won't be available for our project demonstration.

The reason I mention this is because of the way different viewers seem to handle input. Appletviewer, the viewer that comes with JDK, Javasoft's Java development tools is probably the best for running Applets, except it only accepts input from the mouse, no keyboard input. Netscape 3 suffers from being too slow all round in Java while Microsoft Internet Explorer 3 manages adequately and handles keyboard input.

Applet Viewer



Internet Explorer 3



Keyboard input is handled through event handlers, `keyDown(Event e, key)` or `handleEvent(Event e)`. The first problem occurred when trying to input simultaneously to the applet as in cursor key up and cursor key left. The way around was to use the `keyDown` and `KeyUp` events, allow the applet to detect the key press for as long as it is pressed.

The sprite the player controls is `TankSpriteGif`, which has several methods for handling control. `Turn(int)`, `turret.turn(int)`, `setSpeed(boolean)`, `setSpeed(int)`. While a key is pressed the value is set to true or greater than 0 and when a key is released the value is set to false or 0 again. This allows us to have two tanks controlled by the keyboard on the screen at one time.

The game also allows several options the title screen ask you to click on the screen to start, this gives the screen the focus for the keyboard input, if you alt-tab or click somewhere else besides the screen, the applet can stop receiving the input. To remedy this just re-click on the screen.



There are several options on the next screen, the blue icon is controlled by pressing cursor up and cursor down and enter will run that option.



This is all handled by mouseDown, keyDown and keyUp events, and by having the GameManager in different states (intro, playing, endgame).

```
//if mouse is clicked on screen
public boolean mouseDown(Event e,int x, int y){

    if(!playing && !options && !endgame){
        options = true;
    }
    return true;
}
```

```

//key pressed
public boolean keyDown(Event e, int key) {

    //if playing do this
    if(playing){
    switch(key) {
        case Event.RIGHT:
            player1.turn(1);
            break;
        case Event.LEFT:
            player1.turn(-1);
            break;
        case Event.UP:
            player1.setSpeed(true);
            break;
        case Event.DOWN:
            player1.setReverse(true);
            break;
        case Event.ENTER:
            int cnt;
            for(cnt = 0; player1Missiles[cnt].isVisible(); cnt++){ }
            if(player1.isVisible())
                player1Missiles[cnt].fire();
            break;
        case 'o':
            player1.turret.turn(-1);
            break;
        case 'p':
            player1.turret.turn(1);
            break;
        case 'a':
            player2.turn(-1);
            break;
        case 'd':
            player2.turn(1);
            break;
        case 'w':
            player2.setSpeed(true);
            break;
        case 's':
            player2.setReverse(true);
            break;
        case 'q':
            player2.turret.turn(-1);
            break;
        case 'e':
            player2.turret.turn(1);
            break;
        case ' ':
            for(cnt = 0; player2Missiles[cnt].isVisible(); cnt++){ }
            if(player2.isVisible())
                player2Missiles[cnt].fire();
            break;
        default:
            break;
    }
    }
    else if(options){
    switch(key) {
        case Event.UP:
            optionsChoice--;
            break;
        case Event.DOWN:
            optionsChoice++;
    }
}

```

```

        break;
    case Event.ENTER:
        playing = true;
        options = false;
        if(optionsChoice == 0){
            //Start Single Player Level
            startLevel(false);
        }
        else if(optionsChoice == 1){
            //Start Two Player Level
            startLevel(true);
        }
        else if(optionsChoice == 2){
            //Start Two Player Head To Head
            startBattle();
        }
        else if(optionsChoice == 3){
            //Exit Game
            options = false;
            playing = false;
            endgame = true; //makes endgamepaint run
            quit = true; //makes endgamepaint end screen
            repaint();//paint end screen!
            //stop applet
            stop();
        }
        break;
    default:
        break;
    }
}

return true;
}

//if key released
public boolean keyUp(Event e, int key){
    if(playing){
        switch(key) {
            case Event.RIGHT:
                player1.turn(0);
                break;
            case Event.LEFT:
                player1.turn(0);
                break;
            case Event.UP:
                player1.setSpeed(false);
                break;
            case Event.DOWN:
                player1.setReverse(false);
                break;
            case 'o':
                player1.turret.turn(0);
                break;
            case 'p':
                player1.turret.turn(0);
                break;
            case 'q':
                player2.turret.turn(0);
                break;
            case 'e':

```

```
        player2.turret.turn(0);
    break;
    case 'w':
        player2.setSpeed(false);
    break;
    case 's':
        player2.setReverse(false);
    break;
    case 'a':
        player2.turn(0);
    break;
    case 'd':
        player2.turn(0);
    break;
    default:
        break;
    }
    return true;
}
```

Performance

Performance is one of the main concerns with Java, but already mentioned this problem may become unimportant with the eminent release of Just In Time Compilers. Unfortunately it doesn't help us. Minor improvements that were done was making sure work was done only when necessary, for example

```
if(statics!=null)
    for(int i = 0; i < statics.length; i++){
        if(statics[i] != null){
            if(((Sprite)statics[i]).isVisible()){
                f(statics[i].intersect(locx,locy,locx+width,locy+height)){
```

instead of sticking all the if statement together with and (&&), it'll only test the ones with higher priority, for example, the intersect function calls another object and does a calculate: slow and labour intensive, while isVisible() is just a call to see if visible = true, and statics[i]!=null tests if it even exist. This sort of minor performance improvement is everywhere.

Another way to increase performance is to use the -O command line with the JDK javac compiler. This makes the byte streams bigger but the code is optimised quicker.

We spent a lot of time looking into performance and investigated threads. Our function specification gave that GameManager would update and run LevelManager and EnemyManager and HumanManger. After developing the sprite hierarchy we looked into creating these Managers. First LevelManager. Unfortunately the game's speed decreased under Internet Explorer 3 and Netscape (while remaining at a good speed under Appletviewer which was completely useless for us). So LevelManager was implemented as a thread, so that the applet would run along side with LevelManager and while LevelManager was running with MAX_PRIORITY thread. This improved the performance under Appletviewer, but unfortunately no operating system deals with threads the same way, Internet explorer crawled with multiply threads as did Netscape.

Until the Just-in-time compilers are released, our game will have to run without threads.

Another performance increase is the graphics, to increase performance we tried creating a background image with all the

static sprites so that there would be no need to call the paint methods of each static sprite more than once ever while running the game. This worked under the Appletviewer but Internet Explorer 3 couldn't scale properly and only drew a portion of the StaticSprites, though we kept the background image for the tiled background and used it to clear the screen instead of blanking the double buffer in the paint method of GameManager.

We didn't use clipping rectangles either due to the increase in calculating all the separate rectangles for each sprite. Clipping rectangles, limit the drawing to a specific region on the screen.

Miscellaneous

A couple of decisions that were made that didn't fit into any of the other sections. The first was to do the project as an applet. One of the main advantages of Java is that it's portable, but most end users won't have a Java state machine for their operation system but more then lightly will have a Java enabled browser. So if they can download the project they will be able to use it. For the same reason the *User Manual is done in HTML format*. This may not matter once the Just-in-time compilers are release for home users. Also there is only ONE level for the game, unfortunately due to time constraints other levels couldn't be designed and implemented.

Things we would have like to have done, would be storing levels as separate files, *though the level could easily coded into GameManager and a user would only need to get the latest version of this file*, keeping the levels as separate files would allow other people to create their own levels. Which means we could have developed level and sprite editors.

Through doing this project we both learned about Java as a programming languages, it's faults and advantages, about Graphics and animation. We learned about simple 3D design and animation, some maths about motion and speed and implementing them in Java. We learned about simple and technical ways of improving performance and speed and about the disadvantages and advantages of the different Internet Browsers. We also learned about planning and design, how better planning might have led to better design but how our initial well formed design helped us throughout our project.

Code listings and Notes

The first file is **GameManager**, the applet class, the class that runs everything else. It's quite large due to the fact that LevelManager had to be abandoned and incorporated into GameManager. It runs as a thread and runs through three states till it is stopped, introduction (intro), playing and endgame. In introduction; displays the intro and options screens, in playing; runs the game either single player or double, battle or level and in endgame; says who wins and who loses.

It also loads up all the images so that they aren't loaded at a latter date and also initialises and updates all the sprites, a task that would have be delegated to LevelManager if LevelManager had been viable.

```
import java.applet.*;
import java.awt.*;
import java.lang.*;

public class GameManager extends Applet implements Runnable{

    //A Single thread for animation and game
    Thread animation;
    //for double buffering & background
    Graphics offscreen;
    Image image, background;
    //title screens
    Image title, opt, icon;
    //win game status screens
    Image onewin, twowin, lose, allwin;
    //final screen
    Image endScrn;

    //refresh the screen every 80 microseconds
    static final int REFRESH_RATE = 80;

    //playing state (is 2 player? is level?)
    boolean playing = false; //playing a game
    boolean twoPlayer = false; //twoplayer?
    boolean level = true; //battle or level

    //intro state
    boolean options = false;
    int optionsChoice = 0;

    //end game state
    boolean endgame = false;
    int endGameCnt = 0;
    static int endMax = 30;

    //exit state
```

```

boolean quit = false;

//applet dimensions
int width,height;

//weather loading images or not
boolean loading= false;

//-- sprites in the game --

//user tanks (player 1 and player 2)
TankSpriteGif player1, player2;
//player missiles
MissileSpriteGif player1Missiles[], player2Missiles[];
//player start points in game (spawing points)
int spawnx1, spawny1, spawnx2, spawny2;
//player lives
int player1lives;
int player2lives;
//enemies
GenericAI enemies[];
//explosions when sprites die
Explosion boom[], Lboom;
//for testing purposes
StaticSprite buildings[];

////////////////////////////////////
//
//      GameManager methods and control - called by the applet running thread
//
////////////////////////////////////

//loads initial game screens, including title and options and explosions
private void loadTitle(){

    Image exp[] = new Image[9];
    Image lexp[] = new Image[9];

    MediaTracker t = new MediaTracker(this);
    title = getImage(getCodeBase(),"intro/title.gif");
    t.addImage(title,0,width,height);
    for(int i = 0; i < 9; i++){
        exp[i] = getImage(getCodeBase(),"images/Texp"+i+".gif");
        t.addImage(exp[i],0);
        lexp[i] = getImage(getCodeBase(),"images/TLexp"+i+".gif");
        t.addImage(lexp[i],0);
    }
    opt = getImage(getCodeBase(),"intro/options.gif");
    t.addImage(opt,0);
    icon = getImage(getCodeBase(),"intro/icon.gif");
    t.addImage(icon,0);
    onewin = getImage(getCodeBase(),"intro/P1Win.gif");
    t.addImage(onewin,0);
    twowin = getImage(getCodeBase(),"intro/P2Win.gif");
    t.addImage(twowin,0);
    lose = getImage(getCodeBase(),"intro/PLose.gif");
    t.addImage(lose,0);
}

```

```

allwin = getImage(getCodeBase(),"intro/PWin.gif");
t.addImage(allwin,0);
endScrn = getImage(getCodeBase(),"intro/end.gif");
t.addImage(endScrn,0);

showStatus("Loading Images -- Please Wait!");

try {
    t.waitForAll();
} catch (InterruptedException e){ return; }

if (t.isErrorAny()){
    showStatus("Error Loading Images");
}
else if (t.checkAll()){
    showStatus("Successfully loaded.");
}

//creating explosion option
boom = new Explosion[5];
for(int i = 0; i < 5 ; i++){
    boom[i] = new Explosion(exp,this);
}
Lboom = new Explosion(lexp,this);
}

//to start a game
public void startLevel(boolean p){

    level = true;           //playing level
    twoPlayer = p;         //is it 2 player?
    player1lives = 2;      //set lives for players
    player2lives = 2;

    loading = true;
    initBot();             //loading and init level
    loading = false;

}

//to start 2player battle
public void startBattle(){

    twoPlayer = true; //if it's battle it's 2 player
    level = false;    //it's not a level
    player1lives = 0; //setting lives to one (only 1 chance)
    player2lives = 0;

    loading = true;
    initPlayers();     //loading and init battle aera
    loading = false;

}

```

```

//loads images and initalises all bots
private void initBot(){

    //loads images
    Image Tank1[] = new Image[20];
    Image Tank2[] = new Image[20];
    Image turret[] = new Image[20];
    Image flying[] = new Image[20];
    Image hover[] = new Image[20];
    Image prod[] = new Image[20];
    Image boss1[] = new Image[20];
    Image pbot, missile, spike, bomb, hwall, vwall;
    Image b1, r1, br1, star, turret2, tile;

    MediaTracker t = new MediaTracker(this);

    for (int i = 0; i < 20; i++){
        Tank1[i] = getImage(getCodeBase(),"images/TANKR"+i+".GIF");
        t.addImage(Tank1[i],0);
        if(twoPlayer){
            Tank2[i] = getImage(getCodeBase(),"images/2tank"+i+".gif");
            t.addImage(Tank2[i],0);
        }
        turret[i] = getImage(getCodeBase(),"images/TUR"+i+".GIF");
        t.addImage(turret[i],0);

        hover[i] = getImage(getCodeBase(),"images/hover"+i+".gif");
        t.addImage(hover[i],0);
        flying[i] = getImage(getCodeBase(),"images/flyer"+i+".gif");
        t.addImage(flying[i],0);
        prod[i] = getImage(getCodeBase(),"images/prod"+i+".gif");
        t.addImage(prod[i],0);
        boss1[i] = getImage(getCodeBase(),"images/boss1"+i+".gif");
        t.addImage(boss1[i],0);
    }

    showStatus("Loading Some Images For Level 1 -- Please Wait");

    try {
        t.waitForAll();
    } catch (InterruptedException e){ return; }

    if (t.isErrorAny()){
        showStatus("Error Loading Images");
    }

    if (t.checkAll()){
        showStatus("Successfully loaded.");
    }

    missile = getImage(getCodeBase(),"images/ROCKET.GIF");
    t.addImage(missile,0);
    spike = getImage(getCodeBase(),"images/spike.gif");
    t.addImage(spike,0);
    bomb = getImage(getCodeBase(),"images/TBomb.gif");
    t.addImage(bomb,0);

```

```

star = getImage(getCodeBase(),"images/Tstar.gif");
t.addImage(star,0);

pbot = getImage(getCodeBase(),"images/pbot.gif");
t.addImage(pbot,0);
b1 = getImage(getCodeBase(),"images/house.gif");
t.addImage(b1,0);
br1 = getImage(getCodeBase(),"images/bridge.gif");
t.addImage(br1,0);
hwall = getImage(getCodeBase(),"images/HWall.gif");
t.addImage(hwall,0);
vwall = getImage(getCodeBase(),"images/VWall.gif");
t.addImage(vwall,0);
r1 = getImage(getCodeBase(),"images/river.gif");
t.addImage(r1,0);
turret2 = getImage(getCodeBase(),"images/turret.gif");
t.addImage(turret2,0);
tile = getImage(getCodeBase(),"images/back1.gif");
t.addImage(tile,0);

showStatus("Loading Rest Of Images For Level 1 -- Please Wait");

try {
    t.waitForAll();
} catch (InterruptedException e){ return; }

if (t.isErrorAny()){
    showStatus("Error Loading Images");
}

if (t.checkAll()){
    showStatus("Successfully loaded.");
}

showStatus("Initilising Level -- Please Wait");

//creating tiled background to game
background = createTiled(tile);

//to create a game- create all sprites and the game
//will run it's self
//note: code below is complicated but only needed before the game
//is run

//applet is divided into segments (10 * 8) to
//allow placement of objects
int xsegment = width/10;
int ysegment = height/8;

//initilising level aeras
//areas in which enemy (and players) can move
//full applet aera
Rectangle fullRange = new Rectangle(0,0,width,height);
//middle applet aera
Rectangle middleRange = new Rectangle(xsegment*2,0,xsegment*4,height);
//complete range of boss

```

```

        Rectangle bossRange = new Rectangle(xsegment *5, 0, xsegment*5, height-
ysegment);
        //little range of bridge
        Rectangle bridgeRange = new Rectangle(xsegment*4,
ysegment*6,xsegment*4,ysegment*2);

        //initilise buildings
        //StaticSprite(int x, int y, int width,int height, Image i, int Damage, Applet a)
        //StaticSprite buildings[] = new StaticSprite[10];
        buildings = new StaticSprite[10];
        buildings[0] =
new StaticSprite(0,ysegment*4, xsegment *2, ysegment/5,vwall, this);
        buildings[1] =
new StaticSprite(xsegment*2,0,xsegment/5,ysegment*3, hwall, this);
        buildings[2] =
new StaticSprite(xsegment*2,ysegment*5,xsegment/5,ysegment*3,hwall,this);
        buildings[3] =
new StaticSprite(xsegment*3,ysegment, xsegment, ysegment*2, b1, this);
        buildings[4] =
new StaticSprite(xsegment*3,ysegment*5, xsegment, ysegment*3,b1, this);
        buildings[5] =
new StaticSprite(xsegment*6,0,xsegment/2,ysegment*7,r1,this);
        buildings[6] =
new StaticSprite(xsegment,ysegment,xsegment/5,(ysegment*2)+(ysegment/5), hwall,
this);
        buildings[7] =
new StaticSprite(xsegment,ysegment*5,xsegment/5,ysegment*2, hwall, this);
        buildings[8] =
new StaticSprite(0,ysegment*3,xsegment,ysegment/5,vwall,this);
        buildings[9] =
new StaticSprite(0,ysegment*5,xsegment,ysegment/5,vwall,this);

        //initilise players next
        //TankSpriteGif(int x, int y, Image f[], Image t[], Rectangle patrolAera,
//Intersect Statics[], Applet a)
        spawnx1 = xsegment/2;
        spawny1 = ysegment;
        player1 = new TankSpriteGif(spawnx1,spawny1 , Tank1, turret, fullRange,
(Intersect[])buildings, this);
        if(twoPlayer){
            spawnx2 = xsegment/2;
            spawny2 = ysegment * 7;
            player2 = new TankSpriteGif(spawnx2, spawny2, Tank2, turret,
fullRange, (Intersect[])buildings, this);
            player1.addObstucle(player2);
            player2.addObstucle(player1);
        }
        //note players- hold no static pointers to enemies

        //creating target array
        Intersect tempTargets[];
        if(twoPlayer){
            tempTargets = new Intersect[2];
            tempTargets[0] = player1;
            tempTargets[1] = player2;
        }
        else {

```

```

        tempTargets = new Intersect[1];
        tempTargets[0] = player1;
    }

    //creating player missiles
    player1Missiles = new MissileSpriteGif[5];
    if(twoPlayer){
        player2Missiles = new MissileSpriteGif[5];
    }
    //creating enemy missiles
    Projectile ammo[][] = new Projectile[16][5];
    for(int i =0; i < 16; i++){
        for(int j = 0; j < 5; j++){
            if(i == 0){ //boss ammo (diffrent image)
                ammo[i][j] = new
MissileSpriteGif(width,height,star,buildings,null,this);
            }
            else if(i > 11){ //flyer ammo (diffrent object)
                ammo[i][j] = new
FlyingBombGif(width,height,bomb,buildings,null,this);
            }
            else { //gun and hover ammo
                ammo[i][j] = new
MissileSpriteGif(width,height,spike,buildings,null,this);
            }
            ammo[i][j].addTarget((Intersect)player1);
            if(twoPlayer){
                ammo[i][j].addTarget((Intersect)player2);
            }
        }
    }
    for(int i = 0; i < 5; i++){
        player1Missiles[i] = new
MissileSpriteGif(width,height,missile,buildings,player1.turret,this);
        if(twoPlayer){
            player2Missiles[i] = new
MissileSpriteGif(width,height,missile,buildings,player2.turret,this);
            player1Missiles[i].addTarget(player2);
            player2Missiles[i].addTarget(player1);
        }
    }
    //missiles hold no enemy targets

    //initilising enemies
    enemies = new GenericAI[18];

    //enemy 0 is the main Boss that has to be killed
    enemies[0] = new
EnemyHover(xsegment*8,ysegment*3,boss1,bossRange,tempTargets,buildings,
        ammo[0], this);
    enemies[0].setEnergy(200); //increase it's stamina
    enemies[0].setRange(150); //increase it's visibility
range
    enemies[0].setDamage(3); //set it's collision damage
    ((Moveable)enemies[0]).setVelocity(0,0); //freeze it to one point

    //next create hover enemies

```

```

//these 4 enemies will be used by a producer so they are not visible
//straight away
for(int i = 1; i < 5; i++){
    enemies[i] = new EnemyHover(0,0,hover,middleRange,tempTargets,
buildings,ammo[i], this);
    enemies[i].suspend();
}
//create a single standing hover inside bossrange
enemies[5] = new
EnemyHover(xsegment*7,ysegment*1,hover,bossRange,tempTargets, buildings,
ammo[5], this);
((Moveable)enemies[5]).setVelocity(0,0);

//create flyer sprites
//they start at beginning of level and move towards specific areas
enemies[14] = new EnemyFlyer(width,height,flying,bridgeRange,tempTargets,
buildings,
ammo[12], this);
((Moveable)enemies[14]).setPosition(0,ysegment*2);
enemies[15] = new EnemyFlyer(width,height,flying,middleRange,tempTargets,
buildings,
ammo[13], this);
((Moveable)enemies[15]).setPosition(0,ysegment*4);
for(int i = 16; i < 18; i++){
    enemies[i] = new
EnemyFlyer(width,height,flying,bossRange,tempTargets, buildings,
ammo[i-2], this);
}
((Moveable)enemies[16]).setPosition(0,ysegment*5);
((Moveable)enemies[17]).setPosition(0,(ysegment*5)+ysegment/2);

//Producer enemies (enemys that produce other enemies)
//EnemyProducer(int x, int y, Image i[],Image PBotImage, Intersect targets[],
// Intersect statics[], Applet a)
enemies[6] = new
EnemyProducer((xsegment*4),(ysegment*3)+((ysegment/2)*3),prod,pbot,tempTargets,buildings,th
is);

//changing producer to produce hover enemies
Sprite tempSprite[] = new Sprite[4];
for(int i = 0; i < 4; i++){
    tempSprite[i] = enemies[i+1];
}
((EnemyProducer)enemies[6]).changeAmmo(tempSprite);
enemies[7] = new
EnemyProducer((xsegment*4)+(xsegment/2),ysegment/2,prod,pbot,tempTargets,buildings,this);
for(int i = 0; i < ((EnemyProducer)enemies[7]).ammo.length; i++){

((PBot)((EnemyProducer)enemies[7]).ammo[i]).changePatrolAera(middleRange);
}

//basicially gun turrets
//EnemyTurret(int x, int y, Image i, Intersect targets[],
// Projectile ammo[], Applet a)
enemies[8] = new EnemyTurret(0,(ysegment*3)+(ysegment/2),turret2,
tempTargets, ammo[6], this);
enemies[8].setRange(150);

```

```

        enemies[9] = new EnemyTurret(0,(ysegment*4)+(ysegment/2),turret2,
tempTargets, ammo[7], this);
        enemies[9].setRange(150);
        enemies[10] = new
EnemyTurret((xsegment*2)+(xsegment/2),ysegment*7,turret2,tempTargets, ammo[8],this);
        enemies[10].setRange(150);
        enemies[11] = new EnemyTurret(xsegment*4,ysegment*7,turret2,tempTargets,
ammo[9],this);
        enemies[11].setRange(150);
        enemies[12] = new
EnemyTurret((xsegment*5)+(xsegment/2),(ysegment*6)+(ysegment/2),turret2,tempTargets,
ammo[10],this);
        enemies[12].setRange(150);
        enemies[13] = new
EnemyTurret((xsegment*6)+(xsegment/2),(ysegment*6)+(ysegment/2),turret2,tempTargets,
ammo[11],this);
        enemies[13].setRange(150);

//update all sprites and enemies static and target lists

//update player obstucle and missile target lists with enemies
for(int i = 0; i < 18; i++){
    player1.addObstucle((Intersect)enemies[i]);
    if(twoPlayer){
        player2.addObstucle((Intersect)enemies[i]);
    }
    for(int j = 0; j < 5; j++){
        player1Missiles[j].addTarget((Intersect)enemies[i]);
        if(twoPlayer){
            player2Missiles[j].addTarget((Intersect)enemies[i]);
        }
        if(i == 7){
            for(int p = 0; p <
((EnemyProducer)enemies[i]).ammo.length; p++){
                player1Missiles[j].addTarget(((Intersect)((EnemyProducer)enemies[i]).ammo[p]));
                if(twoPlayer){
                    player2Missiles[j].addTarget(((Intersect)((EnemyProducer)enemies[i]).ammo[p]));
                }
            }
        }
    }
}

//update enemies with static pointers to other enemies
//and target to players
for(int i = 1; i < 18; i++){
    for(int j = 0; j < 18; j++){
        if(i!=j){
            enemies[i].addStatic((Intersect)enemies[j]);
        }
        enemies[i].addStatic((Intersect)player1);
        if(twoPlayer){
            enemies[i].addStatic((Intersect)player2);
        }
    }
}

```

```

        if(i == 7){
            for(int j = 0; j < ((EnemyProducer)enemies[i]).ammo.length;
j++){
                for(int p = 0; p < 12; p++){
                    if(i!=p){
                        ((PBot)((EnemyProducer)enemies[i]).ammo[j]).addStatic((Intersect)enemies[p]);
                    }
                }
            }
        }

        showStatus("Kill The Boss!!");
    }

//load images for 2 player battle
private void initPlayers(){

    //loads images
    Image Tank1[] = new Image[20];
    Image Tank2[] = new Image[20];
    Image turret[] = new Image[20];
    Image missile, hwall,vwall,b1,tile;

    MediaTracker t = new MediaTracker(this);

    for (int i = 0; i < 20; i++){
        Tank1[i] = getImage(getCodeBase(),"images/TANKR"+i+".GIF");
        t.addImage(Tank1[i],0);
        Tank2[i] = getImage(getCodeBase(),"images/2tank"+i+".gif");
        t.addImage(Tank2[i],0);
        turret[i] = getImage(getCodeBase(),"images/TUR"+i+".GIF");
        t.addImage(turret[i],0);
    }

    missile = getImage(getCodeBase(),"images/ROCKET.GIF");
    t.addImage(missile,0);
    b1 = getImage(getCodeBase(),"images/house.gif");
    t.addImage(b1,0);
    hwall = getImage(getCodeBase(),"images/Hwall.gif");
    t.addImage(hwall,0);
    vwall = getImage(getCodeBase(),"images/Vwall.gif");
    t.addImage(vwall,0);
    tile = getImage(getCodeBase(),"images/back2.gif");
    t.addImage(tile,0);

    showStatus("Loading Images For Battle Mode -- Please Wait");

    try {
        t.waitForAll();
    } catch (InterruptedException e){ return; }

    if (t.isErrorAny()){

```

```

        showStatus("Error Loading Images");
    }
    else if (t.checkAll()){
        showStatus("Successfully loaded.");
    }

    showStatus("Initilising Battle Arena");

    //create a tiled background image
    background = createTiled(tile);

    //applet is divided into segments (10 * 8) to
    //allow placement of objects
    int xsegment = width/10;
    int ysegment = height/8;

    //StaticSprite(int x, int y, int width,int height, Image i, Applet a)
    //StaticSprite buildings[] = new StaticSprite[7];
    buildings = new StaticSprite[7];
    buildings[0] = new
StaticSprite(xsegment*4,ysegment*3,xsegment*2,ysegment*2, b1, this);
    buildings[1] = new
StaticSprite(xsegment*3,ysegment*3,xsegment/5,ysegment*2, vwall, this);
    buildings[2] = new
StaticSprite(xsegment*7,ysegment*3,xsegment/5,ysegment*2, vwall, this);
    buildings[3] = new
StaticSprite(xsegment,ysegment,xsegment/5,ysegment*6,vwall,this);
    buildings[4] = new
StaticSprite(xsegment*9,ysegment,xsegment/5,ysegment*6,vwall,this);
    buildings[5] = new
StaticSprite(xsegment*2,ysegment,xsegment*6,ysegment/5,hwall,this);
    buildings[6] = new
StaticSprite(xsegment*2,ysegment*7,xsegment*6,ysegment/5,hwall,this);

    //initilising level aeras
    Rectangle fullRange = new Rectangle(0,0,width,height);

    //initilise players next
    //TankSpriteGif(int x, int y, Image f[], Image t[], Rectangle patrolAera,
    //Intersect Statics[], Applet a)
    spawnx1 = width/2;
    spawny1 = 10;
    player1 = new TankSpriteGif(width/2, 10, Tank1, turret, fullRange,
        buildings, this);
    spawnx2 = width/2;
    spawny2 = height-30;
    player2 = new TankSpriteGif(width/2, height-30, Tank2, turret,
        fullRange, buildings, this);
    player1.addObstucle(player2);
    player2.addObstucle(player1);

    //creating player missiles
    player1Missiles = new MissileSpriteGif[5];
    player2Missiles = new MissileSpriteGif[5];
    //creating enemy missiles
    for(int i = 0; i < 5; i++){

```

```

        player1Missiles[i] = new
MissileSpriteGif(width,height,missile,buildings,player1.turret,this);
        player2Missiles[i] = new
MissileSpriteGif(width,height,missile,buildings,player2.turret,this);
        player1Missiles[i].addTarget(player2);
        player2Missiles[i].addTarget(player1);
    }

    showStatus("Go get 'em!");

}

public void updateGame(){

    checkPlayers(); //check to see if players are alive
    //update explosions
    for(int i = 0; i < boom.length;i++){
        boom[i].update();
    }
    Lboom.update();
    //update players and missiles
    player1.update();
    for(int i = 0; i < player1Missiles.length; i++){
        player1Missiles[i].update();
    }
    if(twoPlayer){
        player2.update();
        for(int i = 0; i < player2Missiles.length; i++){
            player2Missiles[i].update();
        }
    }
    //if playing the level- update enemies
    if(level){
        for(int i = 0; i < enemies.length; i++){
            if(enemies[i] != null){
                enemies[i].update();
            }
        }
    }
    checkGameStatus();
}

//paint the game
public void paintGame(Graphics g){

    //draw background
    g.drawImage(background,0,0,width,height,this);
    for(int i = 0; i < buildings.length; i++){
        buildings[i].paintScaled(g);
    }
    //draw players and missiles
    player1.paint(g);
    for(int i = 0; i < player1Missiles.length; i++){
        player1Missiles[i].paint(g);
    }
    if(twoPlayer){
        player2.paint(g);
    }
}

```

```

        for(int i = 0; i < player2Missiles.length; i++){
            player2Missiles[i].paint(g);
        }
    }
    //draw explosions
    for(int i = 0; i < boom.length;i++){
        boom[i].paint(g);
    }
    Lboom.paint(g);
    //and if level draw enemies
    if(level){
        for(int i = 0; i < enemies.length; i++){
            if(enemies[i] != null){
                enemies[i].paint(g);
            }
        }
    }
    paintPlayerScores(g);
}

public void paintIntro(Graphics g){

    if(!options){
        g.drawImage(title,0,0,width,height,this);
        showStatus("Click on screen to start.");
    }
    else {
        g.drawImage(opt,0,0,width,height,this);
        if(optionsChoice < 0){
            optionsChoice = 0;
        }
        else if(optionsChoice > 3){
            optionsChoice = 3;
        }
        g.drawImage(icon,0,(height/4)*optionsChoice,width/6,height/4,this);
        showStatus("Chose an option- use UP and DOWN arrow keys to select
option and ENTER key to start.");
    }

}

//handles all end of game sequences even EXIT option
public void updateEndGame(){

    if(endGameCnt <= 0){
        //if finished showing endgame message
        endgame = false;
    }
    else{
        //decrement counter
        endGameCnt--;
    }
}

```

```

//want game running in background-
//but players unable to interact with it

//update player missiles
for(int i = 0; i < player1Missiles.length; i++){
    player1Missiles[i].update();
}
if(twoPlayer){
    for(int i = 0; i < player2Missiles.length; i++){
        player2Missiles[i].update();
    }
}
//update explosions
for(int i = 0; i < boom.length;i++){
    boom[i].update();
}
Lboom.update();
//if playing the level- update enemies
if(level){
    for(int i = 0; i < enemies.length; i++){
        if(enemies[i] != null){
            enemies[i].update();
        }
    }
}
}

//paints end of game message (either 1 player or 2 player wins or all win or all lose)
public void paintEndGame(Graphics g){

    if(quit){
        g.drawImage(endScrn,0,0,width,height,this);
    }
    else if(level){
        //want the game to happen in background- even though nothing
        //is happening..
        paintGame(g);

        if(((Sprite)player1).isVisible()){
            if(twoPlayer){
                if(((Sprite)player2).isVisible()){
                    g.drawImage(allwin,0,height/3,width,(height/3),this);
                }
            }
            else {
                g.drawImage(allwin,0,height/3,width,(height/3),this);
            }
        }
        else {
            g.drawImage(lose,0,height/3,width,(height/3),this);
        }
    }
    else {
        paintGame(g);
    }
}

```

```

        if(player1.getEnergy() > player2.getEnergy()){
            g.drawImage(onewin,0,height/3,width,(height/3),this);
        }
        else {
            g.drawImage(twowin,0,height/3,width,(height/3),this);
        }
    }
}

private void paintPlayerScores(Graphics g){

    g.setColor(Color.blue);
    g.drawString("Energy Player 1 (lives "+player1lives+)", 3, 13);
    g.fillRect(5, 17, player1.getEnergy(), 10);
    g.setColor(Color.black);
    g.drawRect(5, 17, 100, 10);
    if(twoPlayer){
        g.setColor(Color.red);
        g.drawString("Energy Player 2 (lives "+player2lives+)", 3, 40);
        g.fillRect(5, 45, player2.getEnergy(), 10);
        g.setColor(Color.black);
        g.drawRect(5, 45, 100, 10);
    }
}

//creates a tiled image the size of the applet
private Image createTiled(Image tile){

    Image base = createImage(width,height);
    for(int i = 0; i < width; i += tile.getWidth(this)){
        for(int j = 0; j < height; j += tile.getHeight(this)){
            base.getGraphics().drawImage(tile, i, j, this);
        }
    }
    return base;
}

//check to see if players are alive and respawn them
private void checkPlayers(){

    if(player1lives > 0){
        if(!player1.isVisible()){
            ((Moveable)player1).setPosition(spawnx1,spawny1);
            player1.restore();
            player1lives--;
        }
    }

    if(twoPlayer){
        if(player2lives > 0){
            if(!player2.isVisible()){
                ((Moveable)player2).setPosition(spawnx2,spawny2);
                player2.restore();
            }
        }
    }
}

```

```

        player2lives--;
    }
}

//is game finished?
private void checkGameStatus(){
    //if it is the game
    if(level){
        //if players are dead level is finished
        if(player1lives <= 0){
            if(!((Sprite)player1).isVisible()){
                if(twoPlayer){
                    if(player2lives <= 0){
                        if(!((Sprite)player2).isVisible()){
                            playing = false;
                            endGameCnt = endMax;
                            endgame = true;
                        }
                    }
                }
            }
            else {
                playing = false;
                endGameCnt = endMax;
                endgame = true;
            }
        }
    }
    //if boss(always stored in position 0 of enemies array)
    if(!((Sprite)enemies[0]).isVisible()){
        playing = false;
        endGameCnt = endMax;
        endgame = true;
    }
}
else { //battle mode
    //if one of the players is dead- game over
    if(player1lives <= 0){
        if(!((Sprite)player1).isVisible()){
            playing = false;
            endGameCnt = endMax;
            endgame = true;
        }
    }
    if(player2lives <= 0){
        if(!((Sprite)player2).isVisible()){
            playing = false;
            endGameCnt = endMax;
            endgame = true;
        }
    }
}
}
}

```

```

//explosion called by sprites when they die
//can be a small explosion or big explosion
public void explosion(int x, int y, boolean big){
    if(big){
        Lboom.start(x,y);
    }
    else {
        int cnt;
        for(cnt = 0; cnt < boom.length && boom[cnt].isVisible(); cnt++){ }
        boom[cnt].start(x,y);
    }
}

////////////////////////////////////
//
//      abstract thread and applet methods
//
////////////////////////////////////

public void init() {

    showStatus("Initalising Applet");
    setBackground(Color.black);
    width = bounds().width;
    height = bounds().height;

    //create double buffer
    image = createImage(width,height);
    offscreen = image.getGraphics();

    //load title screen and related images
    loading = true;
    loadTitle();
    loading = false;
}

//start applet
public void start() {

    showStatus("Applet start");
    animation = new Thread(this);
    if(animation != null) {
        animation.start();
    }
}

//update applet
public void update(Graphics g){
    paint(g);
}

//repaint()
public void paint(Graphics g) {

    //offscreen is the double buffer
    //graphics context

```

```

//if game paint level
if(playing){
    paintGame(offscreen);
}
else if(endgame){
    paintEndGame(offscreen);
}
else { //else display option and title
    //screens
    paintIntro(offscreen);
}

//paint double buffer to screen
g.drawImage(image,0,0,this);
}

//while running
public void run(){
    while(true) {
        //as long as not loading images
        if(!loading){
            //refresh screen
            repaint();
            if(playing){
                //if game update the game
                updateGame();
            }
            else if(endgame){
                //if the game has ended
                updateEndGame();
            }
        }
        try {
            Thread.sleep (REFRESH_RATE);
        } catch (Exception exc){};
    }
}

//if applet is stopped
public void stop() {

    showStatus("Applet stopped");
    if (animation != null) {
        animation.stop();
        animation = null;
    }
}

////////////////////////////////////
//
//      keyboard handling (mouse and keyboard)
//
////////////////////////////////////

//if mouse is clicked on screen

```

```

public boolean mouseDown(Event e,int x, int y){

    if(!playing && !options && !endgame){
        options = true;
    }
    return true;
}

//key pressed
public boolean keyDown(Event e, int key) {

    //if playing do this
    if(playing){
        switch(key) {
            case Event.RIGHT:
                player1.turn(1);
                break;
            case Event.LEFT:
                player1.turn(-1);
                break;
            case Event.UP:
                player1.setSpeed(true);
                break;
            case Event.DOWN:
                player1.setReverse(true);
                break;
            case Event.ENTER:
                int cnt;
                for(cnt = 0; player1Missiles[cnt].isVisible(); cnt++){ }
                if(player1.isVisible())
                    player1Missiles[cnt].fire();
                break;
            case 'o':
                player1.turret.turn(-1);
                break;
            case 'p':
                player1.turret.turn(1);
                break;
            case 'a':
                player2.turn(-1);
                break;
            case 'd':
                player2.turn(1);
                break;
            case 'w':
                player2.setSpeed(true);
                break;
            case 's':
                player2.setReverse(true);
                break;
            case 'q':
                player2.turret.turn(-1);
                break;
            case 'e':
                player2.turret.turn(1);
                break;
            case ' ':

```

```

        for(cnt = 0; player2Missiles[cnt].isVisible(); cnt++){ }
        if(player2.isVisible())
            player2Missiles[cnt].fire();
        break;
    default:
        break;
    }
}
else if(options){
    switch(key) {
        case Event.UP:
            optionsChoice--;
            break;
        case Event.DOWN:
            optionsChoice++;
            break;
        case Event.ENTER:
            playing = true;
            options = false;
            if(optionsChoice == 0){
                //Start Single Player Level
                startLevel(false);
            }
            else if(optionsChoice == 1){
                //Start Two Player Level
                startLevel(true);
            }
            else if(optionsChoice == 2){
                //Start Two Player Head To Head
                startBattle();
            }
            else if(optionsChoice == 3){
                //Exit Game
                options = false;
                playing = false;
                endgame = true; //makes endgamepaint run
                quit = true; //makes endgamepaint end screen
                repaint(); //paint end screen!
                //stop applet
                stop();
            }
        break;
    default:
        break;
    }
}
}

return true;
}

//if key released
public boolean keyUp(Event e, int key){

    if(playing){
        switch(key) {
            case Event.RIGHT:

```

```

        player1.turn(0);
    break;
    case Event.LEFT:
        player1.turn(0);
    break;
    case Event.UP:
        player1.setSpeed(false);
    break;
    case Event.DOWN:
        player1.setReverse(false);
    break;
    case 'o':
        player1.turret.turn(0);
    break;
    case 'p':
        player1.turret.turn(0);
    break;
    case 'q':
        player2.turret.turn(0);

    break;
    case 'e':
        player2.turret.turn(0);
    break;
    case 'w':
        player2.setSpeed(false);
    break;
    case 's':
        player2.setReverse(false);
    break;
    case 'a':
        player2.turn(0);
    break;
    case 'd':
        player2.turn(0);
    break;
    default:
    break;
}
}
return true;
}
}

```

This next class is an abstract class called Sprite. This is taken from the Black Art Of Java Game Programming, though all classes derived from it are our own work.

```
// Sprite class
//
// Sprites are objects which can be displayed
//

import java.awt.*;

////////////////////////////////////
abstract class Sprite {
    protected boolean visible;    // is sprite visible
    protected boolean active;    // is sprite updatable

    // abstract methods:
    abstract void paint (Graphics g);
    abstract void update();

    // accessor methods:
    public boolean isVisible() {
        return visible;
    }

    public void setVisible(boolean b) {
        visible = b;
    }

    public boolean isActive() {
        return active;
    }

    public void setActive(boolean b) {
        active = b;
    }

    // suspend the sprite
    public void suspend() {
        setVisible(false);
        setActive(false);
    }

    // restore the sprite
    public void restore() {
        setVisible(true);
        setActive(true);
    }
}
```

```
}
```

This is the first class to extend Sprite class, it handles images, draws a single image to the screen.

```
import java.applet.*;
import java.awt.*;

public class GifSprite extends Sprite {

    //image position and width and height
    protected int width, height;
    protected int locx, locy;

    //image and applet
    Image image;
    Applet applet;

    //constructor
    GifSprite(int x, int y, Image i, Applet a){

        locx = x;
        locy = y;
        image = i;
        applet = a;
        restore();

    }

    //asscesor method
    public void setSize(int w, int h){

        width = w;
        height = h;

    }

    //implements abstract methods:
    //from Sprite
    public void update(){ }

    public void paint(Graphics g){
        if (visible)
            g.drawImage(image,locx,locy,applet);
    }

}
```

Here are the interfaces that were used. Intersect and Moveable are modified code from The Black Art of Java Game Programming.

```
//deals with intersect and collision of sprites
//for sprites that can collide
interface Intersect {
    public boolean intersect(int x1, int y1, int x2, int y2);
    public void hit(int Damage);
    public int getDamage();
    public int getLocx();
    public int getLocy();
    public int getHeight();
    public int getWidth();
}

//deals with motion: speed and direction and position
//for sprites that move
interface Moveable {
    public abstract void setPosition(int x, int y);
    public abstract void setVelocity(int x, int y);
    public abstract void updatePosition();
    public abstract void setVx(int x);
    public abstract void setVy (int y);
}

interface Launcher {
//launcher interface allows sprites to get information from
//other sprites that control them
//such as turret Sprite and Missile Sprite
//launcher: gun or cannon
    public abstract float getAngle();
    public abstract int getFireLocx();
    public abstract int getFireLocy();
}

interface Projectile {
/* Projectile interface is used by objects that can be fired
to allow ai and human sprites to hold many different types without worrying
about what they are */
    public abstract void fire();
    public abstract void changeLauncher(Launcher newLauncher);
    public abstract void addTarget(Intersect newTarget);
}
```

This class GenericAI extends GifSprite but is in it's self a separate abstract class. Please refer to the section Artificial Intelligence for more.

```
import java.awt.*;
import java.applet.*;
import java.lang.*;

abstract class GenericAI extends GifSprite{

    //states an intelligent enemy can be in
    static final byte FLEE = 0;
    static final byte WAIT = 1;
    static final byte ATTACK = 2;
    byte state;

    //probabilities of changing state
    double FLEE_EXIT;
    double WAIT_EXIT;
    double ATTACK_EXIT;

    //constructor
    GenericAI(int x, int y, Image f, Applet a){

        super(x,y,f,a);

        FLEE_EXIT = 0.95;
        WAIT_EXIT = 0.95;
        ATTACK_EXIT = 0.95;

        state = WAIT;

    }

    //assessor methods
    public void setFlee(double value){ FLEE_EXIT = value; }
    public void setWait(double value){ WAIT_EXIT = value; }
    public void setAttack(double value){ ATTACK_EXIT = value; }

    //abstract methods: when are you able to enter states
    public boolean AttackCondition(){return true;}
    public boolean WaitCondition(){return true;}
    public boolean FleeCondition(){return true;}
    //abstract holders for implementation of those states
    public void Attack(){ }
    public void Wait(){ }
    public void Flee(){ }

    //the ai update
    public void AIupdate(){

        double r1 = Math.random();
        double r2 = Math.random();
        switch (state){
```

```

case WAIT:
    if(r1 > WAIT_EXIT){
        if(r2 > 0.5 && FleeCondition()){
            Flee();
            state = FLEE;
        }
        else if (r2 < 0.5 && AttackCondition()){
            Attack();
            state = ATTACK;
        }
    }
    else {
        Wait();
    }
break;
case ATTACK:
    if(r1 > ATTACK_EXIT){
        if(r2 > 0.5 && FleeCondition()){
            Flee();
            state = FLEE;
        }
        else if (r2 < 0.5 && WaitCondition()){
            Wait();
            state = WAIT;
        }
    }
    else{
        Attack();
    }
break;
case FLEE:
    if(r1 > FLEE_EXIT){
        if(r2 > 0.5 && WaitCondition()){
            Wait();
            state = WAIT;
        }
        else if (r2 < 0.5 && AttackCondition()){
            Attack();
            state = ATTACK;
        }
    }
    else {
        Flee();
    }
break;
}

```

```

//some abstract methods for enemy sprites
public void setSpeed(int newSpeed){ }
public void setDamage(int newDamage){ }
public void setRange(int newRange){ }
public void changePatrolAera(Rectangle newPatrolAera){ }
public void setEnergy(int newEnergy){ }
public int getEnergy(){return 0;}
public void addStatic(Intersect newStatic){ }
public void addTarget(Intersect newTarget){ }

```

```

}

```

This class StaticSprite, is an extension of GifSprite and is meant to be non-moving sprites, something like walls or buildings in the game.

```
import java.awt.*;
import java.applet.*;

public class StaticSprite extends GifSprite implements Intersect{

    //if it does damage to objects that collide with it
    int Damage;

    StaticSprite(int x, int y, Image i, int Damage, Applet a){
        super(x,y,i,a);
        this.Damage = Damage;
        setSize(image.getWidth(applet),image.getHeight(applet));
    }

    StaticSprite(int x, int y,int width,int height,Image i, Applet a){
        super(x,y,i,a);
        Damage = 0;
        setSize(width,height);
    }

    //implement Intersect
    public boolean intersect(int x1,int y1,int x2, int y2){
return visible && (x2 >= locx) && (locx+width >= x1) && (y2 >= locy) && (locy+height >= y1);}
    public void hit(int Damage){/*DO NOTHING*/}
    public int getLocx(){ return locx; }
    public int getLocy(){ return locy; }
    public int getWidth(){ return width; }
    public int getHeight(){ return height; }
    public int getDamage(){ return Damage;}

    //implements abstract method from Sprite.class
    public void update(){/*does nothing*/}

    public void paint(Graphics g){
        g.drawImage(image,locx,locy,applet);
    }

    public void paintScaled(Graphics g){
        g.drawImage(image,locx,locy,width,height,applet);
    }

    public Image getImage(){return image;}

}
```

TankSpriteGif is the player's sprite that they control and move. It also holds the object for the TankTurretGif, the moving turret on top of the tank.

```
import java.awt.*;
import java.applet.*;
import java.lang.*;

public class TankSpriteGif extends GifSprite implements Moveable, Intersect{

    protected Image images[];
    protected int currentImage;

    Rectangle patrolAera;

    protected boolean speed;
    protected boolean reverse;
    protected int imageinc = 0;

    protected float angle = 0;

    protected TankTurretGif turret;
    private final int max_speed = 5;
    private final int rev_speed = 3;

    private Intersect Obstucles[];

    private int energy = 100;

    //constructor 1
    TankSpriteGif(int x, int y, Image f[] , Image t[],Rectangle patrolAera, Intersect
    Obstucles[], Applet a){

        super(x,y,f[0],a);

        images = f;
        currentImage = 0;

        fixSize(0);
        setSpeed(false);
        setReverse(false);

        this.patrolAera = patrolAera;

        turret = new TankTurretGif(x,y,t,a,this);

        this.Obstucles = Obstucles;
    }

    //constructor2
    TankSpriteGif(int x, int y, Image f[] , Image t[],Rectangle patrolAera, Applet a){

        super(x,y,f[0],a);

        images = f;
        currentImage = 0;
```

```

        fixSize(0);
        setSpeed(false);
        setReverse(false);

        this.patrolAera = patrolAera;

        turret = new TankTurretGif(x,y,t,a,this);

        Obstucles = new Intersect[1];
    }

    public int getEnergy (){ return energy; }

    public void addObstucle(Intersect ObjectToAdd){

        int oldLength = Obstucles.length;
        Intersect temp[] = new Intersect[oldLength+1];

        for(int i = 0; i < oldLength; i++){
            if(Obstucles[i]!=null){
                temp[i] = Obstucles[i];
            }
        }
        temp[oldLength] = ObjectToAdd;

        Obstucles = temp;
    }

    public void paint(Graphics g){

        if(visible){
            g.drawImage(images[currentImage],locx,locy,applet);
            turret.paint(g);

            /* --- for testing bounding boxes
            g.setColor(Color.black);
            g.drawRect(locx,locy,width,height);
            */
        }
    }

    //implement Moveable interface (setPosition, setVelocity and updatePosition)
    int vx;int vy;
    public void setPosition(int x, int y){locx = x;locy = y;}
    public void setVelocity(int x, int y){vx = x;vy = y;}
    public void updatePosition(){ locx += vx; locy += vy;        }

    //implement Intersect interface (intersect and hit)
    public boolean intersect(int x1,int y1,int x2, int y2){
        return visible && (x2 >= locx) && (locx+width >= x1) && (y2 >= locy) &&
        (locy+height >= y1);
    }

    public void hit(int Damage){
        energy -= Damage;
    }

```

```

}
public int getDamage(){ return 0;}

//implements abstract method from Sprite class:
public void update(){

if(visible){

    if(energy <= 0){
        suspend();
        ((GameManager)applet).explosion(locx,locy,false);
    }

    //tracks current image
    if((currentImage+imageinc)<0){
        currentImage = images.length-1;
    }
    else if((currentImage+imageinc)==images.length){
        currentImage = 0;
    }
    else {
        currentImage = currentImage+imageinc;
    }

    setVel(); // works out velocity and sets it up

    if(Obstucles!=null)
    for(int i = 0; i < Obstucles.length; i++){

        if(Obstucles[i] != null){
            if(((Sprite)Obstucles[i]).isVisible()){
                if(Obstucles[i].intersect(locx,locy,locx+width,locy+height)){

                    Obstucles[i].hit(0);
                    energy = energy - Obstucles[i].getDamage();

                    //get Obstucle x info

                    int tempPos, tempWidth, tempCentre;
                    tempPos = Obstucles[i].getLocx();
                    tempWidth = Obstucles[i].getWidth();
                    tempCentre = tempPos+(tempWidth/2);

                    //testing in the x direction
                    if(((locx+width)>tempPos && (locx+width)<tempCentre && vx > 0)
                    || (locx <(tempPos+tempWidth) && locx>tempCentre && vx < 0)){
                        vx = -vx;
                    }

                    //get Obstucle y info
                    tempPos = Obstucles[i].getLocy();
                    tempWidth = Obstucles[i].getHeight();
                    tempCentre = tempPos+(tempWidth/2);

                    //testing in the y direction
                    if(((locy+height)>tempPos && (locy+height)<tempCentre && vy > 0)
                    || (locy <(tempPos+tempWidth) && locy>tempCentre && vy < 0)){
                        vy = -vy;
                    }
                }
            }
        }
    }
}
}

```

```

        }
    }
}

//see if tank is at edge of screen...
if((locx+width+6 > (patrolAera.x+patrolAera.width) && vx > 0) || (locx-6 <
patrolAera.x && vx < 0)){

    vx = -vx;
}
if((locy+height+6 > (patrolAera.y+patrolAera.height) && vy > 0) || (locy-6 <
patrolAera.y && vy < 0)){

    vy = -vy;
}

updatePosition();

//update turret
turret.update();
}
}

protected void setVel() {

    //calculates correct angle
    float temp = (360/images.length);
    angle = currentImage * temp;
    //angle in radions
    float rad_angle = (float)((angle/180) * Math.PI);

    //if in motion
    if(speed || reverse){

        int temp_speed;
        if(speed){
            temp_speed = max_speed;
        }
        else {
            temp_speed = rev_speed;
        }

        //calculate velocity
        if(angle == 0){
            setVelocity(0,-temp_speed);
        }
        else if(angle == 90){
            setVelocity(temp_speed,0);
        }
        else if(angle == 180){
            setVelocity(0,temp_speed);
        }
    }
}

```

```

else if(angle == 270){
    setVelocity(-temp_speed,0);
}
else if(angle>0 && angle<90){

    setVx((int)(Math.sin(rad_angle)*temp_speed));
    setVy((int)(-(Math.cos(rad_angle)*temp_speed)));

}
else if(angle>90 && angle<180){

    setVx((int)(Math.cos(rad_angle-(Math.PI/2))*temp_speed));
    setVy((int)(Math.sin(rad_angle-(Math.PI/2))*temp_speed));

}
else if(angle>180 && angle<270){

    //these set work
    setVx((int)(-(Math.sin(rad_angle-Math.PI)*temp_speed)));
    setVy((int)(Math.cos(rad_angle-Math.PI)*temp_speed));

}
else if(angle>270 && angle<360) {

    //these set work
    setVx((int)(-(Math.cos(rad_angle-
(1.5*Math.PI))*temp_speed)));
    setVy((int)(-(Math.sin(rad_angle-
(1.5*Math.PI))*temp_speed)));
}
//it its in reverse... reverse velocity
if(reverse){
    vx=-vx;
    vy=-vy;
}

}
else {
    setVelocity(0,0);
}

}

//this little funtion sets the width and height to the image
//pointed to by imageVal
protected void fixSize(int imageVal){

setSize(images[imageVal].getWidth(applet),images[imageVal].getHeight(applet));
}

public void setSpeed(boolean s){ speed = s; }
public void setReverse(boolean r){ reverse = r; }

//set turning motion
public void turn(int t){ imageinc = t; }

//allows person to set x y velocities seperatily
public void setVx(int x){ vx = x; }

```

```

public void setVy(int y){ vy = y; }

//assessor methods
public int getVx(){ return vx; }
public int getVy(){ return vy; }

public int getLocx(){ return locx; }
public int getLocy(){ return locy; }

public int getWidth(){ return width; }
public int getHeight(){ return height; }

public void setEnergy(int newEnergy){energy = newEnergy;}

public void restore(){
    energy = 100;

    setVisible(true);
    setActive(true);

    currentImage = 0;

    setSpeed(false);
    setReverse(false);
}

public void changePatrolAera(Rectangle newPatrolAera){
    patrolAera = newPatrolAera;
}
}

```

This class TankTurretGif is merely the sprite class for the revolving turret on the player tank.

```

import java.awt.*;
import java.applet.*;
import java.lang.*;

public class TankTurretGif extends GifSprite implements Launcher{

    protected Image images[];
    protected int currentImage;

    protected int max_width, max_height;
}

```

```

protected int imageinc = 0;

protected TankSpriteGif parentTank;

TankTurretGif(int x, int y, Image f[], Applet a, TankSpriteGif t){

    super(x,y,f[0],a);

    images = f;
    currentImage = 0;
    parentTank = t;
    setSize(images[0].getWidth(applet),images[0].getHeight(applet));

}

public void paint(Graphics g){

    g.drawImage(images[currentImage],locx,locy,applet);

}

public void setPosition(int x, int y){

    locx = x;
    locy = y;

}

//implements abstract method from Sprite class

public void update(){

    //tracks current image
    if((currentImage+imageinc)<0){
        currentImage = images.length-1;
    }
    else if((currentImage+imageinc)==images.length){
        currentImage = 0;
    }
    else {
        currentImage = currentImage+imageinc;
    }

    //fix size
    setSize(images[0].getWidth(applet),images[0].getHeight(applet));

    //fix position
    setPosition(parentTank.getLocx(),parentTank.getLocy());

}

//allows player to turn turret
public void turn(int t){ imageinc = t; }

//assesor method: Launcher interface
public float getAngle(){

    return (currentImage * (360/images.length));

}

public int getFireLocx(){
    return (locx + (width/2));
}

```

```

    }
    public int getFireLocy(){
        return (locy + (height/2));
    }
}

```

This class MissileSpriteGif is the standard weapon for most of the sprites in the game.

```

import java.awt.*;
import java.applet.*;
import java.lang.*;

public class MissileSpriteGif extends GifSprite implements Moveable, Intersect, Projectile{

    protected int max_width, max_height;
    protected Launcher launcher;

    protected Intersect targets[];

    private int MAX_SPEED = 20;

    //constructor
    MissileSpriteGif(int mx, int my, Image f, Intersect targets[], Launcher parentTurret, Applet a){

        super(0,0,f,a);
    }
}

```

```

        max_width = mx;
        max_height = my;

        this.targets = targets;

        launcher = parentTurret;

        suspend();

        setSize(image.getWidth(applet),image.getHeight(applet));
    }

    public void addTarget(Intersect ObjectToAdd){

        int oldLength = targets.length;
        Intersect temp[] = new Intersect[oldLength+1];

        for(int i = 0; i < oldLength; i++){
            if(targets[i]!=null){
                temp[i] = targets[i];
            }
        }
        temp[oldLength] = ObjectToAdd;

        targets = temp;
    }

    //implement Moveable interface (setPosition, setVelocity and updatePosition)
    int vx;
    int vy;
    public void setPosition(int x, int y){ locx = x; locy = y; }
    public void setVelocity(int x, int y){ vx = x; vy = y; }
    public void updatePosition(){ locx += vx; locy += vy; }
    public void setVx(int x){ vx = x; }
    public void setVy(int y){ vy = y; }

    //implement Intersect interface (intersect and hit)
    public boolean intersect(int x1,int y1,int x2, int y2){
        return visible && (x2 >= locx) && (locx+width >= x1) && (y2 >= locy) &&
        (locy+height >= y1);
    }
    public void hit(int Damage){ suspend(); }
    public int getLocx(){ return locx; }
    public int getLocy(){ return locy; }
    public int getWidth(){ return width; }
    public int getHeight(){ return height; }
    public int getDamage(){ return 0; }

    //Projectile Abstract Methods:
    public void changeLauncher(Launcher newLauncher){
        launcher = newLauncher;
    }

    //when it's fired
    public void fire(){

        //only works if not already in use
        if(!visible){

            setPos();//sets position

```

```

        setVel(); //sets velocity and direction

        restore();//makes visible
    }

}

public void update(){

    if(visible){

        //check if still on screen
        if((locx+width > max_width) || (locx < 0) || (locy+height > max_height) || (locy
< 0)){

            suspend();

        }

        if(targets!=null) //this line is here for when doing testing
            //see if it hit anything
            for(int i = 0; i < targets.length; i++){
                if(targets[i] != null){
                    if(((Sprite)targets[i]).isVisible()){
                        if(targets[i].intersect(locx,locy,locx+width,locy+height)){
                            targets[i].hit(1);
                            suspend();

                        }
                    }
                }
            }

            updatePosition();

        }

    }

//sets up initil firing position of missile
protected void setPos(){
    //uses the launcher interface
    //calcuates it's relative x,y in realltion to firing position
    setPosition(launcher.getFireLocx()-(width/2),launcher.getFireLocy()-(height/2));
}

//calculate realtive velocity to the angle of the turret
protected void setVel() {

    //calculates correct angle (getAngle supplied by interface launcher)
    float angle = launcher.getAngle();
    //angle in radions
    float rad_angle = (float)((angle/180) * Math.PI);

    //calculate velocity
    if(angle == 0){
        setVelocity(0,-MAX_SPEED);
    }
    else if(angle == 90){
        setVelocity(MAX_SPEED,0);
    }
    else if(angle == 180){
        setVelocity(0,MAX_SPEED);
    }
}

```

```

    }
    else if(angle == 270){
        setVelocity(-MAX_SPEED,0);
    }
    else if(angle>0 && angle<90){
        setVx((int)(Math.sin(rad_angle)*MAX_SPEED));
        setVy((int)(-(Math.cos(rad_angle)*MAX_SPEED)));
    }
    else if(angle>90 && angle<180){
        setVx((int)(Math.cos(rad_angle-(Math.PI/2))*MAX_SPEED));
        setVy((int)(Math.sin(rad_angle-(Math.PI/2))*MAX_SPEED));
    }
    else if(angle>180 && angle<270){
        setVx((int)(-(Math.sin(rad_angle-Math.PI)*MAX_SPEED));
        setVy((int)(Math.cos(rad_angle-Math.PI)*MAX_SPEED));
    }
    else if(angle>270 && angle<360) {
        setVx((int)(-(Math.cos(rad_angle-(1.5*Math.PI))*MAX_SPEED));
        setVy((int)(-(Math.sin(rad_angle-(1.5*Math.PI))*MAX_SPEED));
    }
}
}
}

```

This is the toughest AI enemy. This class is also used for the main boss.

```

import java.awt.*;
import java.applet.*;
import java.lang.*;

public class EnemyHover extends GenericAI implements Moveable, Intersect, Launcher{

    protected Intersect targets[];//array of viable targest
    protected int target_attack; //which target to attack
    protected Intersect statics[];//array of objects that are in way

    protected Projectile ammo[]; //what it fires

    protected Rectangle patrolAera; //max boundries of hover

    protected int NM_speed = 3; //normal speed
    protected int MX_speed = 5; //fast speed
    protected int energy = 30; //it's energy
    protected int range = 75; //range of sight
    protected int speed; //it's speed
    protected int damage = 0; //damage it does it if touches
    protected Image images[];

    protected int currentImage; //keeps track of images

    EnemyHover(int x, int y, Image i[], Rectangle r, Intersect targets[],

```

```

        Intersect statics[], Projectile ammo[], Applet a){
super(x,y,i[0],a);

//setups up it's view of the world
this.targets = targets;
this.statics = statics;
patrolAera = r;

this.ammo = ammo;
for(int j = 0; j < ammo.length; j++){
    ammo[j].changeLauncher(this);
}

images = i;

//set size
setSize(image.getWidth(applet),image.getHeight(applet));

//set chances of action
setFlee(0.3);
setAttack(0.85);
setWait(0.3);

//set speed
speed = NM_speed;

if(Math.random() > 0.5){
    setVx(-speed);
}
else{
    setVx(speed);
}
if(Math.random() > 0.5){
    setVy(speed);
}
else{
    setVy(-speed);
}
}

public float getAngle(){
    float angle = 0;

    if(target_attack == -1){
        target_attack = 0;
    }

    int targetx = targets[target_attack].getLocx();
    int targety = targets[target_attack].getLocy();

    float rand = (float)(Math.random() * 90);

    if(locx <= targetx && locy >= targety){
        angle = rand;
    }
    else if(locx <= targetx && locy < targety){
        angle = 90+rand;
    }
    else if(locx > targetx && locy < targety){
        angle = 180+rand;
    }
}

```

```

        else if(locx > targetx && locy >= targety){
            angle = 270+rand;
        }

        return angle;
    }
    public int getFireLocx(){ return locx+(width/2);}
    public int getFireLocy(){ return locy+(height/2);}

    //implement Moveable interface (setPosition, setVelocity and updatePosition)
    int vx,int vy;
    public void setPosition(int x, int y){locx = x;locy = y;}
    public void setVelocity(int x, int y){ vx = x;vy = y;}
    public void updatePosition(){ locx += vx; locy += vy; }
    public void setVx(int x){ vx = x; }
    public void setVy(int y){ vy = y; }

    //implement Intersect interface (intersect and hit)
    public boolean intersect(int x1,int y1,int x2, int y2){
        return visible && (x2 >= locx) && (locx+width >= x1) && (y2 >= locy) &&
        (locy+height >= y1);
    }
    public void hit(int Damage){
        energy -= Damage;
        flee = true;
    }
    public int getDamage(){ return damage;}
    public int getLocx(){ return locx; }
    public int getLocy(){ return locy; }
    public int getWidth(){ return width; }
    public int getHeight(){ return height; }

    //implement methods from generic AI
    boolean attack = false;
    boolean wait = false;
    boolean flee = false;
    public boolean AttackCondition(){return attack;}
    public boolean WaitCondition(){return wait;}
    public boolean FleeCondition(){return flee;}

    public void Attack(){
        int cnt = 0;
        for(;(cnt < ammo.length) && ((Sprite)ammo[cnt]).isVisible(); cnt++){
            if(cnt < ammo.length && Math.random() > 0.5){
                ammo[cnt].fire();
            }

            speed = MX_speed;
            fixSpeed();
        }

    public void Wait(){
        speed = NM_speed;
    }

    public void Flee(){
        speed = MX_speed;
        fixSpeed();
        vx = -vx;
        vy = -vy;
    }

    public void fixSpeed(){

```

```

        if(target_attack == -1){
            target_attack = 0;
        }

        double diffx = (double)(targets[target_attack].getLocx() - locx);
        double diffy = (double)(targets[target_attack].getLocy() - locy);

        double xSq = Math.pow(diffx,2.0);
        double ySq = Math.pow(diffy,2.0);
        double dist = Math.pow(ySq+xSq,0.5);

        double ratio = (double)speed/dist;

        setVx((int)(diffx*ratio));
        setVy((int)(diffy*ratio));
    }

    public void paint(Graphics g){
        for(int i = 0; i < ammo.length; i++){
            ((Sprite)ammo[i]).paint(g);
        }
        if(visible){
            g.drawImage(images[currentImage],locx,locy,applet);
            /* -- for testing and design
            g.setColor(Color.black);
            g.drawRect(patrolAera.x,patrolAera.y,patrolAera.width,patrolAera.height);
            */
        }
    }

    public void update(){

        for(int i = 0; i < ammo.length; i++){
            ((Sprite)ammo[i]).update();
        }

        if(visible){

            //check to see if sprite is dead if not do update
            if(energy <= 0){
                suspend();
                if(width < 100){
                    ((GameManager)applet).explosion(locx,locy,false);
                }
                else {
                    ((GameManager)applet).explosion(locx,locy,true);
                }
            }

            //check if target nearby and chose target randomly...
            double ran = Math.random();
            target_attack = -1;
            for(int i = 0; i < targets.length; i++){
                if(targets[i] != null){
                    if(((Sprite)targets[i]).isVisible()){
                        if(targets[i].intersect(locx-range,locy-range,locx+width+range,locy+height+range)){
                            if(target_attack != -1 && ran > 0.5){/* do nothing */}
                            else{
                                target_attack = i;
                            }
                        }
                    }
                }
            }
        }
    }

```

```

    }
        }
    }
    //set states accordingly
    if(target_attack != -1){
        attack = true;
        flee = true;
        wait = false;
    }
    else {
        attack = false;
        flee = false;
        wait = true;
    }
    if(energy < 10){
        flee = true;
    }

    AIupdate();

    //bounce of obstacles...
    if(statics!=null)
    for(int i = 0; i < statics.length; i++){

        if(statics[i] != null){
            if(((Sprite)statics[i]).isVisible()){
                if(statics[i].intersect(locx,locy,locx+width,locy+height)){

                    statics[i].hit(damage);
                    energy = energy - statics[i].getDamage();

                    //get Obstacle x info

                    int tempPos, tempWidth, tempCentre;
                    tempPos = statics[i].getLocx();
                    tempWidth = statics[i].getWidth();
                    tempCentre = tempPos+(tempWidth/2);
                    //testing in the x direction
                    if(((locx+width)>tempPos && (locx+width)<tempCentre && vx > 0)
                    || (locx <(tempPos+tempWidth) && locx>tempCentre && vx < 0)){
                        vx = -vx;
                    }

                    //get Obstacle y info
                    tempPos = statics[i].getLocy();
                    tempWidth = statics[i].getHeight();
                    tempCentre = tempPos+(tempWidth/2);
                    //testing in the y direction
                    if(((locy+height)>tempPos && (locy+height)<tempCentre && vy > 0)
                    || (locy <(tempPos+tempWidth) && locy>tempCentre && vy < 0)){
                        vy = -vy;
                    }
                }
            }
        }
    }

    //bounce off boundaries
    if((locx+width+6 > (patrolAera.x+patrolAera.width) && vx > 0) || (locx-6<patrolAera.x
    && vx < 0)){

        vx = -vx;
    }

```

```

    }
    if((locy+height+6 > (patrolAera.y+patrolAera.height) && vy > 0) || (locy-6<patrolAera.x
&& vy < 0)){

        vy = -vy;
    }

    updatePosition();

    //update images for animation
    if(currentImage == 19){
        currentImage = 0;
    }
    else {
        currentImage++;
    }

} //end if visible
}

//allow controlling class to check energy
public int getEnergy(){return energy;}

//allows user to add another obstacle to list
public void addStatic(Intersect ObjectToAdd){

    int oldLength = statics.length;
    Intersect temp[] = new Intersect[oldLength+1];

    for(int i = 0; i < oldLength; i++){
        if(statics[i]!=null){
            temp[i] = statics[i];
        }
    }
    temp[oldLength] = ObjectToAdd;

    statics = temp;
}

//allows user to add another target
public void addTarget(Intersect ObjectToAdd){

    int oldLength = targets.length;
    Intersect temp[] = new Intersect[oldLength+1];

    for(int i = 0; i < oldLength; i++){
        if(targets[i]!=null){
            temp[i] = targets[i];
        }
    }
    temp[oldLength] = ObjectToAdd;

    targets = temp;
}

public void restore() {
    setVisible(true);
    setActive(true);

    energy = 30;
    speed = NM_speed;

    if(Math.random() > 0.5){

```

```
        setVx(-speed);
    }
    else{
        setVx(speed);
    }
    if(Math.random() > 0.5){
        setVy(speed);
    }
    else{
        setVy(-speed);
    }
}

public void setDamage(int newDamage){ damage = newDamage;}
public void setRange(int newRange){ range = newRange;}
public void changePatrolAera(Rectangle newPatrolAera){ patrolAera = newPatrolAera;}
public void setEnergy(int newEnergy){ energy = newEnergy;}
}
```

This is another enemy, but this one “flies” around dropping bombs on the passing tank.

```
import java.awt.*;
import java.applet.*;
import java.lang.*;

public class EnemyFlyer extends GenericAI implements Moveable, Intersect, Launcher{

    protected Intersect targets[];//array of viable targest
    protected int target_attack;//which target to attack
    protected Intersect statics[];//array of objects that are in way

    protected Projectile ammo[]; //what it fires

    protected Rectangle patrolAera; //max boundries of hover

    protected int speed = 5; //fast speed
    protected int energy = 10; //it's energy
    protected int range = 10; //range of sight

    protected Image images[];

    protected int currentImage; //keeps track of images

    protected boolean landing = false;
    protected boolean landed = false;
    protected boolean takeoff = false;

    protected int landing_cnt = 0;
    protected int takeoff_cnt = 0;
    protected int landed_cnt = 0;
    protected int flying_cnt = 0;

    EnemyFlyer(int x, int y, Image i[], Rectangle patrolAera, Intersect targets[],
               Intersect statics[], Projectile ammo[], Applet a){

        super(x,y,i[0],a);

        //setups up it's view of the world
        this.targets = targets;
        this.statics = statics;
        this.patrolAera = patrolAera;

        this.ammo = ammo;
        for(int j = 0; j < ammo.length; j++){
            ammo[j].changeLauncher(this);
        }
    }
}
```

```

        images = i;

        //set size
        setSize(image.getWidth(applet),image.getHeight(applet));

        //set chances of action
        //always in attack mode
        setFlee(0);
        setAttack(1);
        setWait(0);

        if(Math.random() > 0.5){
            setVx(-speed);
        }
        else{
            setVx(speed);
        }
        if(Math.random() > 0.5){
            setVy(speed);
        }
        else{
            setVy(-speed);
        }
    }

    public float getAngle(){

        //projectile must drop behind- no angle needed
        return 0;
    }
    public int getFireLocx(){ return locx+(width/2);}
    public int getFireLocy(){ return locy+(height/2);}

    //implement Moveable interface (setPosition, setVelocity and updatePosition)
    int vx;int vy;
    public void setPosition(int x, int y){locx = x;locy = y;}
    public void setVelocity(int x, int y){vx = x;vy = y;}
    public void updatePosition(){ locx += vx; locy += vy; }
    public void setVx(int x){ vx = x; }
    public void setVy(int y){ vy = y; }

    //implement Intersect interface (intersect and hit)
    public boolean intersect(int x1,int y1,int x2, int y2){
        //can only be hit if it is landed
        if(landed){
            return visible && (x2 >= locx) && (locx+width >= x1) && (y2 >= locy) &&
(locy+height >= y1);
        }
        else {
            return false;
        }
    }
    public void hit(int Damage){
        energy -= Damage;
    }
    public int getDamage(){ return 0;}
    public int getLocx(){ return locx; }
    public int getLocy(){ return locy; }
    public int getWidth(){ return width; }
    public int getHeight(){ return height; }

    //implement methods from generic AI
    public boolean AttackCondition(){return true;}

```

```

public boolean WaitCondition(){return false;}
public boolean FleeCondition(){return false;}

public void Attack(){
    if(!landed && !takeoff && !landing){

        boolean fire = false;
        //if targets in range - flyer must fire
        for(int i = 0; i < targets.length; i++){
            if(targets[i] != null){
                if(((Sprite)targets[i]).isVisible()){
                    if(targets[i].intersect(locx-range,locy-
range,locx+width+range,locy+height+range)){
                        fire = true;
                    }
                }
            }
        }

        if(fire){
            int cnt = 0;
            for(;(cnt < ammo.length) && ((Sprite)ammo[cnt]).isVisible(); cnt++){ }
            if(cnt < ammo.length){
                ammo[cnt].fire();
            }
        }
    }
}

public void Wait(){/*do nothing*/}
public void Flee(){/*do nothing*/}

public void paint(Graphics g){
    if(visible){
        for(int i = 0; i < ammo.length; i++){
            ((Sprite)ammo[i]).paint(g);
        }
        g.drawImage(images[currentImage],locx,locy,width,height,applet);
        /* -- for testing patrol aera
        g.setColor(Color.black);
        g.drawRect(patrolAera.x,patrolAera.y,patrolAera.width,patrolAera.height);
        */
    }
}

public void update(){

    for(int i = 0; i < ammo.length; i++){
        ((Sprite)ammo[i]).update();
    }

    if(visible){
        //check to see if sprite is dead if not do update
        if(energy <= 0){
            suspend();
            if(width < 100){
                ((GameManager)applet).explosion(locx,locy,false);
            }
            else {
                ((GameManager)applet).explosion(locx,locy,true);
            }
        }
    }
}

```

```

    }

    AIupdate();

    if(flying_cnt >= 95){
        //bounce of obstucles...
        for(int i = 0; i < statics.length; i++){
            if(statics[i] != null){
                if(((Sprite)statics[i]).isVisible()){
                    if(statics[i].intersect(locx,locy,locx+width,locy+height)){
                        flying_cnt = 70;
                    }
                }
            }
        }
    }

    //bounce off boundaries
    if(!landed && !landing && !takeoff){
        if((locx+width+6 > (patrolAera.x+patrolAera.width) && vx > 0) || (locx-6 < patrolAera.x && vx < 0)){
            vx = -vx;
        }
        if((locy+height+6 > (patrolAera.y+patrolAera.height) && vy > 0) || (locy-6 < patrolAera.y && vy < 0)){
            vy = -vy;
        }
    }
    updatePosition();

    //increment counters to change state
    //depending on if its landed, landing, taking off or flying
    //increase and decrease it's height and width accordingly
    if(!landed && !landing && !takeoff){
        flying_cnt++;
        if(flying_cnt++ > 100){
            landing = true;
            landed = false;
            takeoff = false;
            landing_cnt = 0;
            setVelocity(0,0);
        }
    }
    else if(landing){
        landing_cnt++;
        if(landing_cnt > 20){
            landed = true;
            landing = false;
            takeoff = false;
            landed_cnt = 0;
        }
        else {
            height-=1;
            width-=1;
        }
    }
    else if(landed){
        landed_cnt++;
        if(landed_cnt > 20){
            landed = false;
            landing = false;
            takeoff = true;
            takeoff_cnt = 0;
        }
    }

```

```

    }
    else if(takeoff){
        takeoff_cnt++;
        if(takeoff_cnt > 20){
            landed = false;
            landing = false;
            takeoff = false;
            flying_cnt = 0;
            if(Math.random() > 0.5){
                setVx(-speed);
            }
            else{
                setVx(speed);
            }

            if(Math.random() > 0.5){
                setVy(speed);
            }
            else{
                setVy(-speed);
            }
        }
        else {
            height+=1;
            width+=1;
        }
    }

    if(!landed){
        //update images for animation
        if(currentImage == 19){
            currentImage = 0;
        }
        else {
            currentImage++;
        }
    }

}

//end if visible
}

//allow controlling class to check energy
public int getEnergy(){return energy;}

//allows user to add another obstacle to list
public void addStatic(Intersect ObjectToAdd){

    int oldLength = statics.length;
    Intersect temp[] = new Intersect[oldLength+1];

    for(int i = 0; i < oldLength; i++){
        if(statics[i]!=null){
            temp[i] = statics[i];
        }
    }
    temp[oldLength] = ObjectToAdd;

    statics = temp;
}

//allows user to add another target
public void addTarget(Intersect ObjectToAdd){

```

```

        int oldLength = targets.length;
        Intersect temp[] = new Intersect[oldLength+1];

        for(int i = 0; i < oldLength; i++){
            if(targets[i]!=null){
                temp[i] = targets[i];
            }
        }
        temp[oldLength] = ObjectToAdd;

        targets = temp;
    }

    public void restore() {
        setVisible(true);
        setActive(true);

        energy = 10;
        currentImage = 0;

        landing = false;
        landed = false;
        takeoff = false;

        landing_cnt = 0;
        takeoff_cnt = 0;
        landed_cnt = 0;
        flying_cnt = 0;

        if(Math.random() > 0.5){
            setVx(-speed);
        }
        else{
            setVx(speed);
        }
        if(Math.random() > 0.5){
            setVy(speed);
        }
        else{
            setVy(-speed);
        }

        setSize(image.getWidth(applet),image.getHeight(applet));
    }

    public void setRange(int newRange){ range = newRange;}
    public void changePatrolAera(Rectangle newPatrolAera){ patrolAera = newPatrolAera;}
    public void setEnergy(int newEnergy){ energy = newEnergy;}
}

```

This is the weapon that EnemyFlyer uses.

```

import java.awt.*;
import java.applet.*;
import java.lang.*;

```

```

public class FlyingBombGif extends GifSprite implements Moveable, Intersect, Projectile{

    protected int max_width, max_height;
    protected Launcher launcher;

    protected Intersect targets[];

    private int MAX_SPEED = 20;
    protected int origWidth=0,origHeight;
    public boolean hasLanded=false;
    protected int landed_count=0;

    //constructor
    FlyingBombGif(int mx, int my, Image f, Intersect targets[], Launcher parentTurret, Applet a){

        super(0,0,f,a);

        max_width = mx;
        max_height = my;

        this.targets = targets;

        launcher = parentTurret;

        suspend();

        setSize(image.getWidth(applet),image.getHeight(applet));

        origWidth=width;
        origHeight=height;
    }

    public void addTarget(Intersect ObjectToAdd){

        int oldLength = targets.length;
        Intersect temp[] = new Intersect[oldLength+1];

        for(int i = 0; i < oldLength; i++){
            if(targets[i]!=null){
                temp[i] = targets[i];
            }
        }
        temp[oldLength] = ObjectToAdd;

        targets = temp;
    }

    //implement Moveable interface (setPosition, setVelocity and updatePosition)
    int vx;
    int vy;
    public void setPosition(int x, int y){ locx = x; locy = y; }
    public void setVelocity(int x, int y){ vx = x; vy = y; }
    public void updatePosition(){ locx += vx; locy += vy; }
    public void setVx(int x){ vx = x; }
    public void setVy(int y){ vy = y; }

    //implement Intersect interface (intersect and hit)
    public boolean intersect(int x1,int y1,int x2, int y2){
        return visible && (x2 >= locx) && (locx+width >= x1) && (y2 >= locy) &&
        (locy+height >= y1);
    }
}

```

```

public void hit(int Damage){ suspend(); }
public int getLocx(){ return locx; }
public int getLocy(){ return locy; }
public int getWidth(){ return width; }
public int getHeight(){ return height; }
public int getDamage(){ return 0;}

//Projectile Abstract Methods:
public void changeLauncher(Launcher newLauncher){
    launcher = newLauncher;
}

//when it's fired
public void fire(){

    //only works if not already in use
    if(!visible){

        setPos();//sets position

        restore();//makes visible
    }

}

public void update(){

    if(visible){

        hasLanded = false;
        width -=3;
        height -=3;
        landed_count++;

        //check if still on screen
        if((locx+width > max_width) || (locx < 0) || (locy+height > max_height) || (locy
< 0)){

            suspend();

        }

        if (landed_count == 10) { hasLanded = true; }

        if(targets!=null) //this line is here for when doing testing
            //see if it hit anything

if(hasLanded)
    {
        landed_count=0;
        //reset width and height to original values.
        width=origWidth;
        height=origHeight;
        visible=false;

        for(int i = 0; i < targets.length; i++){
            if(targets[i]!=null){
                if(((Sprite)targets[i]).isVisible()){
                    if(targets[i].intersect(locx,locy,locx+width,locy+height)){
                        targets[i].hit(5);
                        suspend();
                    }
                }
            }
        }
    }
}

```

```

        }
        }
    }
    } //end if
    updatePosition();
}

}

//sets up initial firing position of missile
protected void setPos(){
    //uses the launcher interface
    //calculates it's relative x,y in relation to firing position
    setPosition(launcher.getFireLocx()-(width/2),launcher.getFireLocy()-(height/2));
}

//Override Paint method ...
public void paint(Graphics g){
    if (visible)
        g.drawImage(image,locx,locy,width,height,applet);
}
}

```

Another enemy sprite, a gun turret that fires at the player.

```

import java.awt.*;
import java.applet.*;
import java.lang.*;

public class EnemyTurret extends GenericAI implements Intersect, Launcher{

    protected Intersect targets[];    //array of viable targest
    protected int target_attack;      //which target to attack
}

```

```

protected Projectile ammo[]; //what it fires

protected int energy = 30; //it's energy
protected int range = 75; //range of sight

EnemyTurret(int x, int y, Image i, Intersect targets[],
             Projectile ammo[], Applet a){

    super(x,y,i,a);

    //setups up it's view of the world
    this.targets = targets;

    this.ammo = ammo;
    for(int j = 0; j < ammo.length; j++){
        ammo[j].changeLauncher(this);
    }

    //set size
    setSize(image.getWidth(applet),image.getHeight(applet));

    //set chances of action
    setFlee(1);
    setAttack(0.5);
    setWait(0.5);
}

public float getAngle(){

    float angle = 0;

    if(target_attack == -1){
        target_attack = 0;
    }

    int targetx = targets[target_attack].getLocx();
    int targety = targets[target_attack].getLocy();

    float rand = (float)(Math.random() * 90);

    if(locx <= targetx && locy >= targety){
        angle = rand;
    }
    else if(locx <= targetx && locy < targety){
        angle = 90+rand;
    }
    else if(locx > targetx && locy < targety){
        angle = 180+rand;
    }
    else if(locx > targetx && locy >= targety){
        angle = 270+rand;
    }
}

```

```

        return angle;
    }
    public int getFireLocx(){ return locx+(width/2);}
    public int getFireLocy(){ return locy+(height/2);}

    //implement Intersect interface (intersect and hit)
    public boolean intersect(int x1,int y1,int x2, int y2){
        return visible && (x2 >= locx) && (locx+width >= x1) && (y2 >= locy) &&
        (locy+height >= y1);
    }
    public void hit(int Damage){ energy -= Damage;}
    public int getDamage(){ return 0;}
    public int getLocx(){ return locx; }
    public int getLocy(){ return locy; }
    public int getWidth(){ return width; }
    public int getHeight(){ return height; }

    //implement methods from generic AI
    boolean attack = false;
    boolean wait = true;
    public boolean AttackCondition(){return attack;}
    public boolean WaitCondition(){return wait;}
    public boolean FleeCondition(){return false;}

    public void Attack(){
        int cnt = 0;
        for(;cnt < ammo.length) && ((Sprite)ammo[cnt]).isVisible(); cnt++){ }
        if(cnt < ammo.length && Math.random() > 0.5){
            ammo[cnt].fire();
        }
    }

    public void Wait(){/*do nothing*/}
    public void Flee(){/*not used*/}

    public void paint(Graphics g){
        for(int i = 0; i < ammo.length; i++){
            ((Sprite)ammo[i]).paint(g);
        }
        if(visible){
            g.drawImage(image,locx,locy,applet);
            /* -- for testing and design
            g.setColor(Color.black);
            g.drawRect(locx-range,locy-range,width+(2*range),height+(2*range));
            */
        }
    }

    public void update(){

        for(int i = 0; i < ammo.length; i++){
            ((Sprite)ammo[i]).update();
        }
    }

```

```

if(visible){

    //check to see if sprite is dead if not do update
    if(energy <= 0){
        suspend();
        if(width < 100){
            ((GameManager)applet).explosion(locx,locy,false);
        }
        else {
            ((GameManager)applet).explosion(locx,locy,true);
        }
    }

    //check if target nearby and chose target randomly...
    double ran = Math.random();
    target_attack = -1;
    for(int i = 0; i < targets.length; i++){
        if(targets[i] != null){
            if(((Sprite)targets[i]).isVisible()){
if(targets[i].intersect(locx-range,locy-range,locx+width+range,locy+height+range)){
                if(target_attack != -1 && ran > 0.5){/* do nothing */}
                else{
                    target_attack = i;
                }
            }
        }
    }

    //set states accordingly
    if(target_attack != -1){
        attack = true;
        wait = false;
    }
    else {
        attack = false;
        wait = true;
    }

    AIupdate();

} //end if visible
}

```

```

//allow controlling class to check energy
public int getEnergy(){return energy;}

```

```

//allows user to add another target
public void addTarget(Intersect ObjectToAdd){

```

```

    int oldLength = targets.length;
    Intersect temp[] = new Intersect[oldLength+1];

    for(int i = 0; i < oldLength; i++){
        if(targets[i]!=null){
            temp[i] = targets[i];
        }
    }

```

```

        }
        temp[oldLength] = ObjectToAdd;

        targets = temp;
    }

    public void restore() {
        setVisible(true);
        setActive(true);

        energy = 30;
    }

    public void setRange(int newRange){ range = newRange;}
    public void setEnergy(int newEnergy){ energy = newEnergy;}
}

```

Another Enemy- this one produces other enemy sprites.

```

import java.awt.*;
import java.applet.*;
import java.lang.*;

public class EnemyProducer extends GenericAI implements Intersect{

    protected Intersect targets[]; //array of viable target
    //protected int target_attack; //which target to attack

    public Sprite ammo[]; //what it fires

    protected int energy = 100; //it's energy
    protected int range = 100; //range of sight
    protected Image images[];

    protected int currentImage; //keeps track of images

    protected boolean inRange;
}

```

```

EnemyProducer(int x, int y, Image i[],Image PBotImage, Intersect targets[],
              Intersect statics[], Applet a){

    super(x,y,i[0],a);

    //setups up it's view of the world
    this.targets = targets;

    Rectangle ammoR= new Rectangle(locx-range,locy-
range,width+(2*range),height+(2*range));
    ammo = new PBot[10];
    for(int j = 0; j < 10; j++){
        ammo[j] = new
PBot(locx+width/2,locy+height/2,PBotImage,ammoR,targets,statics,applet);
    }

    images = i;

    //set size
    setSize(image.getWidth(applet),image.getHeight(applet));

    //set chances of action
    setFlee(0.0);
    setAttack(1);
    setWait(0.0);

    inRange = false;

}

public void changeAmmo(Sprite changeTo[]){
    ammo = changeTo;
}

//implement Intersect interface (intersect and hit)
public boolean intersect(int x1,int y1,int x2, int y2){
    return visible && (x2 >= locx) && (locx+width >= x1) && (y2 >= locy) &&
(locy+height >= y1);
}
public void hit(int Damage){ energy -= Damage;}
public int getDamage(){ return 0;}
public int getLocx(){ return locx; }
public int getLocy(){ return locy; }
public int getWidth(){ return width; }
public int getHeight(){ return height; }

//implement methods from generic AI
public boolean AttackCondition(){return true;}
public boolean WaitCondition(){return false;}
public boolean FleeCondition(){return false;}

public void Attack(){
    if(inRange){
        produce();
    }
}
public void Wait(){/*do nothing*/}
public void Flee(){/*do nothing*/}

public void paint(Graphics g){
    if(visible){

```

```

        g.drawImage(images[currentImage],locx,locy,applet);
        /*
        g.setColor(Color.black);
        g.drawRect(locx - range,locy - range,width+(range*2),height+(range*2));
        */
    }
    //always paint..
    for(int i = 0; i < ammo.length; i++){
        ammo[i].paint(g);
    }
}

public void update(){

    //update ammo even if dead
    for(int i = 0; i < ammo.length; i++){
        ammo[i].update();
    }

    if(visible){
        //check to see if sprite is dead if not do update
        if(energy <= 0){
            suspend();
            if(width < 100){
                ((GameManager)applet).explosion(locx,locy,false);
            }
            else {
                ((GameManager)applet).explosion(locx,locy,true);
            }
        }
        //store old state;
        boolean temp = inRange;
        //check if target nearby and chose target randomly...
        double ran = Math.random();
        inRange = false;
        //target_attack = -1;
        for(int i = 0; i < targets.length; i++){
            if(targets[i] != null){
                if(((Sprite)targets[i]).isVisible()){
                    if(targets[i].intersect(locx-range,locy-range,locx+width+range,locy+height+range)){

                        //if(target_attack != -1 && ran > 0.5){/* do nothing */}
                        //else{
                        //    target_attack = i;
                        //    inRange = true;
                        //}
                    }
                }
            }
        }

        if(inRange){
            //update images for animation
            if(currentImage == 19){
                currentImage = 0;
            }
            else {
                currentImage++;
            }
        }
        else{
            currentImage = 0;
        }
    }
}

```

```

    }

    AIupdate();

} //end if visible
}

//allow controlling class to check energy
public int getEnergy(){return energy;}

//allows user to add another target
public void addTarget(Intersect ObjectToAdd){

    int oldLength = targets.length;
    Intersect temp[] = new Intersect[oldLength+1];

    for(int i = 0; i < oldLength; i++){
        if(targets[i]!=null){
            temp[i] = targets[i];
        }
    }
    temp[oldLength] = ObjectToAdd;

    targets = temp;
}

protected void produce(){

    if(currentImage >= 19){
        int cnt = 0;
        for(;cnt < ammo.length && ammo[cnt].isVisible();cnt++){/*do nothing*/}
        if(cnt < ammo.length && Math.random() > 0.3){
            ((Moveable)ammo[cnt]).setPosition(locx,locy);
            ammo[cnt].restore();
        }
    }
}

public void setRange(int newRange){ range = newRange;}
public void setEnergy(int newEnergy){ energy = newEnergy;}
public void restore(){
    setVisible(true);
    setActive(true);
    inRange = false;
}

}
}

```

This is the standard sprite that the above producer enemy sprite produces...

```
import java.awt.*;
import java.applet.*;
import java.lang.*;

public class PBot extends GenericAI implements Moveable, Intersect{

    protected Intersect targets[];//array of viable targest
    protected int target_attack; //which target to attack
    protected Intersect statics[];//array of objects that are in way

    protected Rectangle patrolAera; //boundry area

    protected int energy = 1; //it's energy
    protected int speed = 3; //it's speed
    protected int range = 100;
    protected Image images[];

    PBot(int x, int y, Image i, Rectangle r, Intersect targets[],
        Intersect statics[], Applet a){

        super(x,y,i,a);

        //setups up it's view of the world
        this.targets = targets;
        this.statics = statics;
        patrolAera = r;

        for(int j = 0; j < targets.length; j++){
            addStatic(targets[j]);
        }
    }
}
```

```

        //set size
        setSize(image.getWidth(applet),image.getHeight(applet));

        //set chances of action
        setFlee(0);
        setAttack(1);
        setWait(0);

        suspend();
    }

    //implement Moveable interface (setPosition, setVelocity and updatePosition)
    int vx,int vy;
    public void setPosition(int x, int y){locx = x;locy = y;}
    public void setVelocity(int x, int y){ vx = x;vy = y;}
    public void updatePosition(){ locx += vx; locy += vy; }
    public void setVx(int x){ vx = x; }
    public void setVy(int y){ vy = y; }

    //implement Intersect interface (intersect and hit)
    public boolean intersect(int x1,int y1,int x2, int y2){
        return visible && (x2 >= locx) && (locx+width >= x1) && (y2 >= locy) &&
        (locy+height >= y1);
    }
    public void hit(int Damage){
        suspend();
    }
    public int getDamage(){ return 5;}
    public int getLocx(){ return locx; }
    public int getLocy(){ return locy; }
    public int getWidth(){ return width; }
    public int getHeight(){ return height; }

    public boolean AttackCondition(){return true;}
    public boolean WaitCondition(){return false;}
    public boolean FleeCondition(){return false;}

    public void Attack(){ fixSpeed(); }

    public void Wait(){/*do nothing*/}
    public void Flee(){/*do nothing*/}

    public void fixSpeed(){

        if(target_attack != -1){

            double diffx = (double)(targets[target_attack].getLocx() - locx);
            double diffy = (double)(targets[target_attack].getLocy() - locy);

            double xSq = Math.pow(diffx,2.0);
            double ySq = Math.pow(diffy,2.0);
            double dist = Math.pow(ySq+xSq,0.5);

            double ratio = (double)speed/dist;

            setVx((int)(diffx*ratio));
            setVy((int)(diffy*ratio));
        }
    }

    public void update(){

```

```

//check to see if sprite is dead if not do update
if(energy <= 0){
    suspend();
}
if(visible){

    AIupdate();

    //check if target nearby and chose target randomly...
    double ran = Math.random();
    target_attack = -1;
    for(int i = 0 ; i < targets.length; i++){
        if(targets[i] != null){
            if(((Sprite)targets[i]).isVisible()){
                if(targets[i].intersect(locx-range, locy-
range, locx+width+range, locy+height+range)){
                    if((target_attack == -1) || (target_attack != -1 && ran > 0.5)){
                        target_attack = i;
                    }
                }
            }
        }
    }

    for(int i = 0; i < statics.length; i++){
        if(statics[i] != null){
            if(((Sprite)statics[i]).isVisible()){
                if(statics[i].intersect(locx, locy, locx+width, locy+height)){
                    statics[i].hit(3);
                    suspend();
                }
            }
        }
    }

    //bounce off boundaries
    if((locx+width+6 > (patrolAera.x+patrolAera.width) && vx > 0) || (locx-6 < patrolAera.x
&& vx < 0)){

        vx = -vx;
    }
    if((locy+height+6 > (patrolAera.y+patrolAera.height) && vy > 0) || (locy-6 < patrolAera.y
&& vy < 0)){

        vy = -vy;
    }

    updatePosition();

} //end if visible
}

//allow controlling class to check energy
public int getEnergy(){return energy;}

//allows user to add another obstacle to list
public void addStatic(Intersect ObjectToAdd){

    int oldLength = statics.length;

```

```

        Intersect temp[] = new Intersect[oldLength+1];

        for(int i = 0; i < oldLength; i++){
            if(statics[i]!=null){
                temp[i] = statics[i];
            }
        }
        temp[oldLength] = ObjectToAdd;

        statics = temp;
    }

    //allows user to add another target
    public void addTarget(Intersect ObjectToAdd){

        int oldLength = targets.length;
        Intersect temp[] = new Intersect[oldLength+1];

        for(int i = 0; i < oldLength; i++){
            if(targets[i]!=null){
                temp[i] = targets[i];
            }
        }
        temp[oldLength] = ObjectToAdd;

        targets = temp;
    }

    public void restore() {
        setVisible(true);
        setActive(true);

        energy = 1;
    }

    public void setEnergy(int newEnergy){ energy = newEnergy; }
    public void setRange(int newRange){ range = newRange; }
    public void changePatrolAera(Rectangle newPatrolAera){ patrolAera = newPatrolAera;}
}

```

This is the final class, an explosion class. This just displays an explosion animation at the desired location.

```
import java.applet.*;
import java.awt.*;

class Explosion extends GifSprite {

    Thread animate;
    Image expImages[];
    int currentImage;

    Explosion(Image EImage[] , Applet a) {

        super(0,0,EImage[0],a);

        expImages = EImage;
        suspend();
    }

    public void start(int x, int y){
        currentImage = 0;
        locx = x;
        locy = y;
        restore();
    }

    public void paint(Graphics g){
        if(visible){
            g.drawImage(expImages[currentImage],locx,locy,applet);
        }
    }

    public void update(){
        currentImage++;
        if(currentImage >= expImages.length){
            suspend();
        }
    }
}
```

```
}  
} //end class
```