CT255 / NGT2 / Digital Media

Week 5

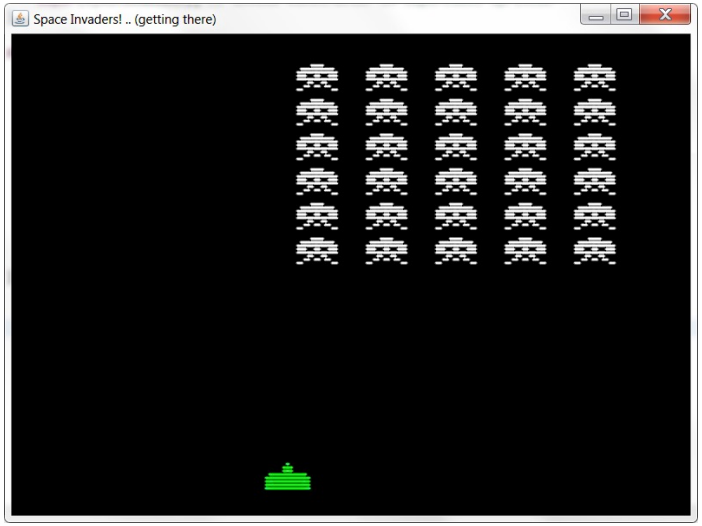
[2D Games in Java]

Dr. Sam Redfern

sam.redfern@universityofgalway.ie

Last week's assignment

- We're moving closer to a finished game!
- Refactor the game:
  - Make the application window larger (800x600)?
  - Create an **Alien** class and a **Spaceship** class, both subclasses of **Sprite2D**. Move functionality from Sprite2D to the new classes as appropriate.
  - Modify the member variables of the **InvadersApplication** class so that it stores an array of **Alien** objects and a single **Spaceship** object (previously all were Sprite2D objects)
  - Make sure all of the above is working before moving on!

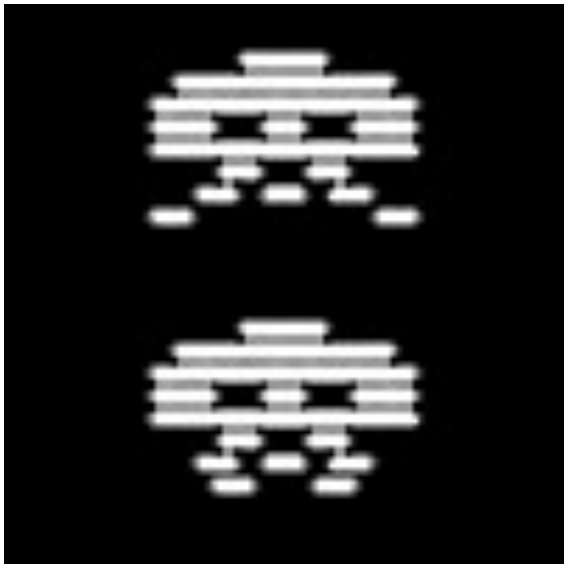


- Implement double buffering to get rid of flickering
- Initialise the aliens in a grid formation rather than randomly positioned
- Modify alien movement so that they all move left or right together (i.e. aliens should use the xSpeed variable similar to how the spaceship does)
- Make all the aliens reverse their movement direction and move down a bit when *any* of them hits the edge of the screen.. somehow

Topics this week

- Animated 2D sprites
- Collision detection in 2D raster games
- ArrayLists
- Game States

Animated 2D sprites

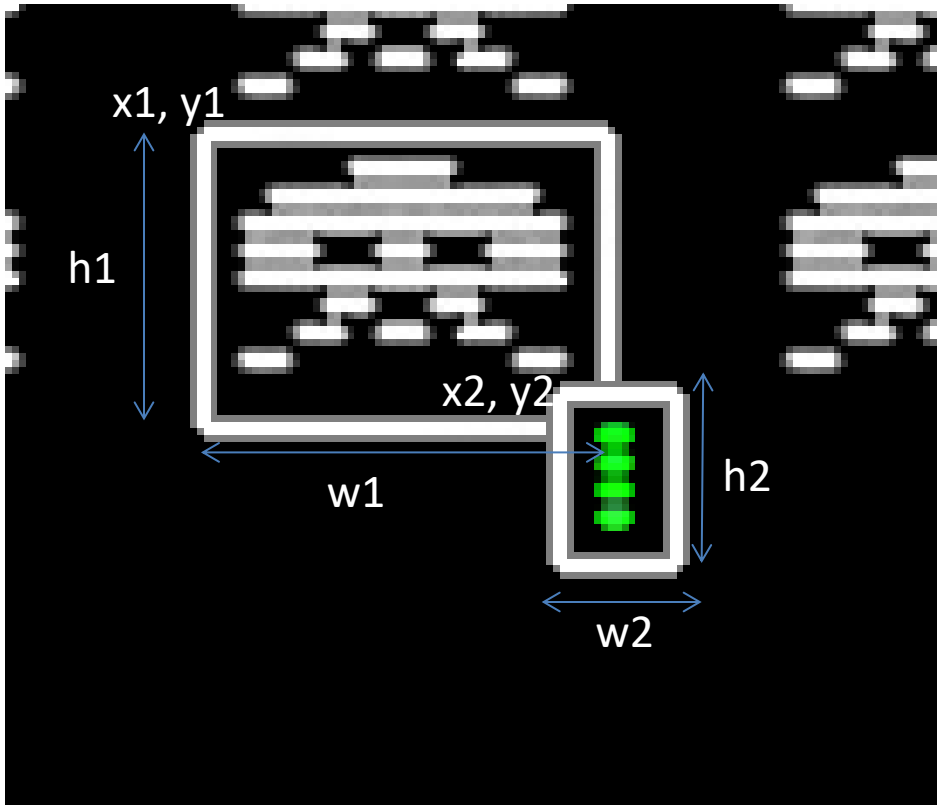


- Simply load two or more images, rather than just one
- Alternate between, or cycle through, the images
- For our game, switching image once per second (i.e. every 50th re-draw) is about right

e.g. use this in a modified Sprite2D class:

```
public void paint(Graphics g) {
   framesDrawn++;
   if ( framesDrawn%100<50 )
      g.drawImage(myImage, (int)x, (int)y, null);
   else
      g.drawImage(myImage2, (int)x, (int)y, null);
}
```

Collision Detection



- Check for overlapping rectangles..

```
if (
  (  (x1<x2 && x1+w1>x2) ||
       (x2<x1 && x2+w2>x1)  )
&&
   (  (y1<y2 && y1+h1>y2) ||
        (y2<y1 && y2+h2>y1)  )
   )
```

Game States

- Games normally have at least two high-level ‘states’ – i.e. is the game in progress or are we currently displaying a menu before the game starts (or after it finishes)?
- We can simply add a boolean member to the application class: isGameInProgress
- Depending on the value of this, we can handle various things differently:
  - Keypresses
  - The paint method
  - The thread’s game loop

This week's assignment – Finishing off the Invaders game!

- Modify the Sprite2D class so that it has two separate member Images, rather than one. When painting, it should alternate between these images every 50 frames
- For the Alien class constructor, receive two animation frames. For the Spaceship class, one frame will do.
- Create a new class, **public class PlayerBullet extends Sprite2D**, and program it to fire when the spacebar is pressed. It should be initialised to the player ship's x/y position, and move upwards (negative y direction) every frame
- While calling the move method on the bullet, check for collision with each Alien object.
- You'll need some way of knowing whether the bullet and each Alien are alive, in order to decide whether to paint them
- If possible, enable several bullets at a time rather than just one
- Add game states (in progress or in menus): you will need to modify various methods of the application class based on the state it's in
- Add scoring
- Add collision detection between aliens and the player spaceship (game over when this happens => switch back to menus state)
- When all aliens are killed, re-create a new, faster-moving wave of them
- I would suggest you carefully assess where certain code is running (e.g. setting the initial positions of aliens and player ship) – the constructor is no longer the best place for this – you will need to create new methods such as startNewWave() and startNewGame()


Space Invaders!
GAME OVER
Press any key to play
[Arrow keys to move, space to fire]


Space Invaders!
Score: 120 Best: 1040

Suggested class interfaces

InvadersApplication.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.image.*;
import java.util.ArrayList;
import java.util.Iterator;

public class InvadersApplication extends JFrame implements Runnable, KeyListener {

    // member data
    private static final Dimension WindowSize = new Dimension(800,600);
    private BufferStrategy strategy;
    private static final int NUMALIENS = 30;
    private Alien[] AliensArray = new Alien[NUMALIENS];
    private Spaceship PlayerShip;
    private Image bulletImage;
    private ArrayList bulletsList = new ArrayList();

    // constructor
    public InvadersApplication() {

    // thread's entry point
    public void run() {

    // Three Keyboard Event-Handler functions
    public void keyPressed(KeyEvent e) {

    public void keyReleased(KeyEvent e) {

    public void keyTyped(KeyEvent e) {
    //

    // method to handle shooting
    public void shootBullet() {

    // application's paint method
    public void paint(Graphics g) {

    // application entry point
    public static void main(String[] args) {

}
```

Sprite2D.java

```
import java.awt.*;

public class Sprite2D {

    // member data
    protected double x,y;
    protected double xSpeed=0;
    protected Image myImage, myImage2;
    int framesDrawn=0;
    int winWidth;

    // constructor
    public Sprite2D(Image i, Image i2, int windowWidth) {

    public void setPosition(double xx, double yy) {

    public void setXSpeed(double dx) {

    public void paint(Graphics g) {

}
```

Spaceship.java

```
import java.awt.Image;

public class Spaceship extends Sprite2D {

    public Spaceship(Image i, int windowWidth) {

    public void move() {

}
```

Alien.java

```
import java.awt.Graphics;

public class Alien extends Sprite2D {

    public boolean isAlive = true;

    public Alien(Image i, Image i2, int windowWidth) {

    public void paint(Graphics g) {

    // public interface
    public boolean move() {

    public void reverseDirection() {

}
```

PlayerBullet.java

```
import java.awt.Image;

public class PlayerBullet extends Sprite2D {

    public PlayerBullet(Image i, int windowWidth) {

    public boolean move() {

}
```

A note on Java Collection Classes

- Java provides a number of useful classes for dealing with collections of objects
- More advanced/flexible than arrays
- To allow a number of bullets rather than just one at a time, consider using an ArrayList
- See: http://tutorials.jenkov.com/java-collections/list.html [discuss in class]
- E.g. code to add a PlayerBullet object, and to draw all PlayerBullet objects:

```
public void shootBullet() {
   // add a new bullet to our list
   PlayerBullet b = new PlayerBullet(bulletImage,WindowSize.width);
   b.setPosition(PlayerShip.x+54/2, PlayerShip.y);
   bulletsList.add(b);
}
```

To remove an element *while iterating the list*:
`iterator.remove();`
(a 'for' loop is not safe for this)

```
Iterator iterator = bulletsList.iterator();
while(iterator.hasNext()){
   PlayerBullet b = (PlayerBullet) iterator.next();
   b.paint(g);
}
```