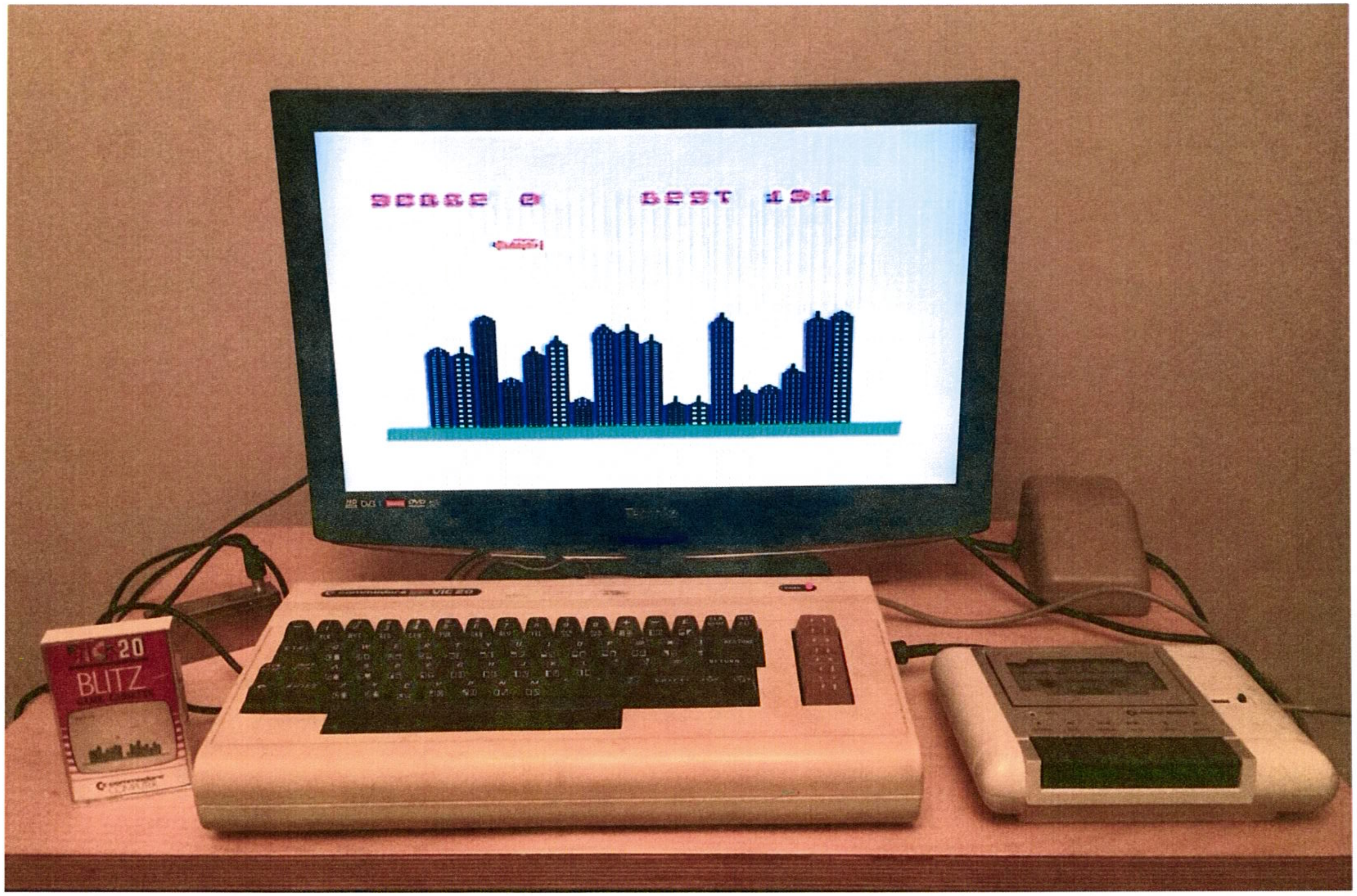
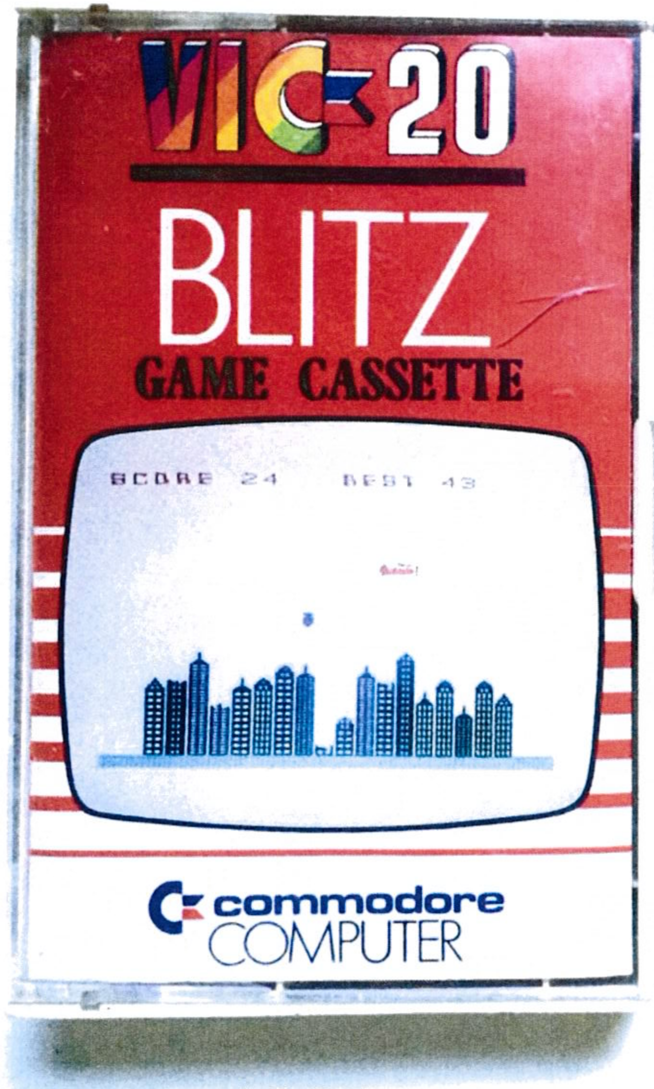
Blitz Resurrection: Re-creating a classic 80's video game in **Processing 2**

@stevebattle
http://blog.stevebattle.me



"Why the plane can't land at an airport or fly around the buildings, rather than destroying a city is beyond me. Why didn't I question ridiculous plots when I was young?"

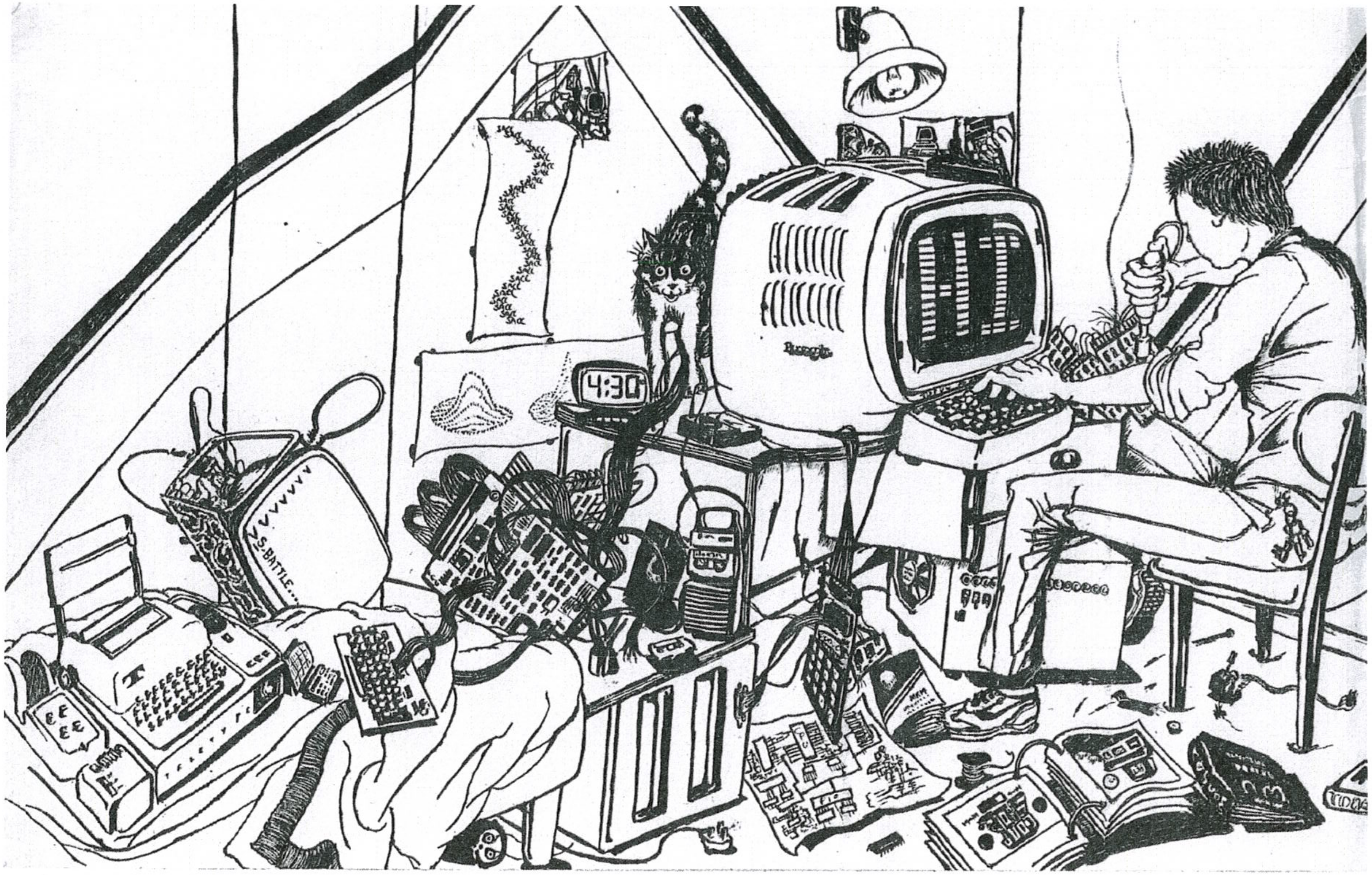
`http://bestretrogames.blogspot.co.uk/2012/01/blitz-commodore-vic-20-1981.html`



1981

1983

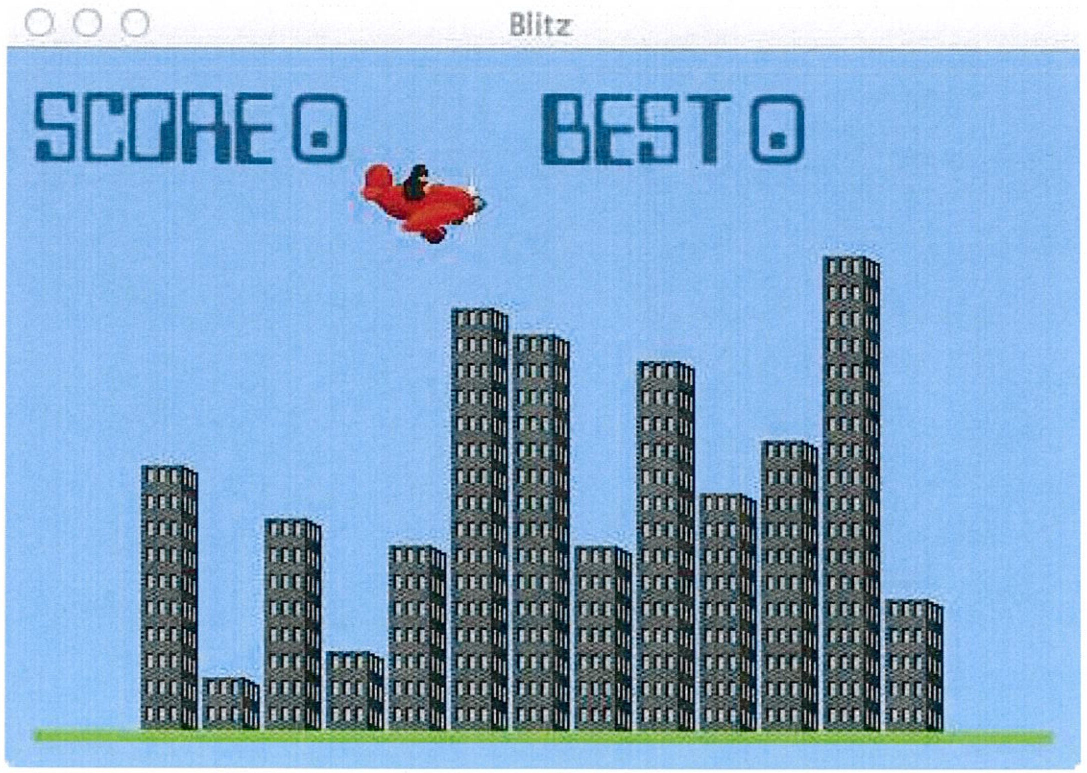
http://en.wikipedia.org/wiki/Blitz_(video_game)



http://battle-bot.blogspot.co.uk/2013/09/retrogeek.html

Workshop goals

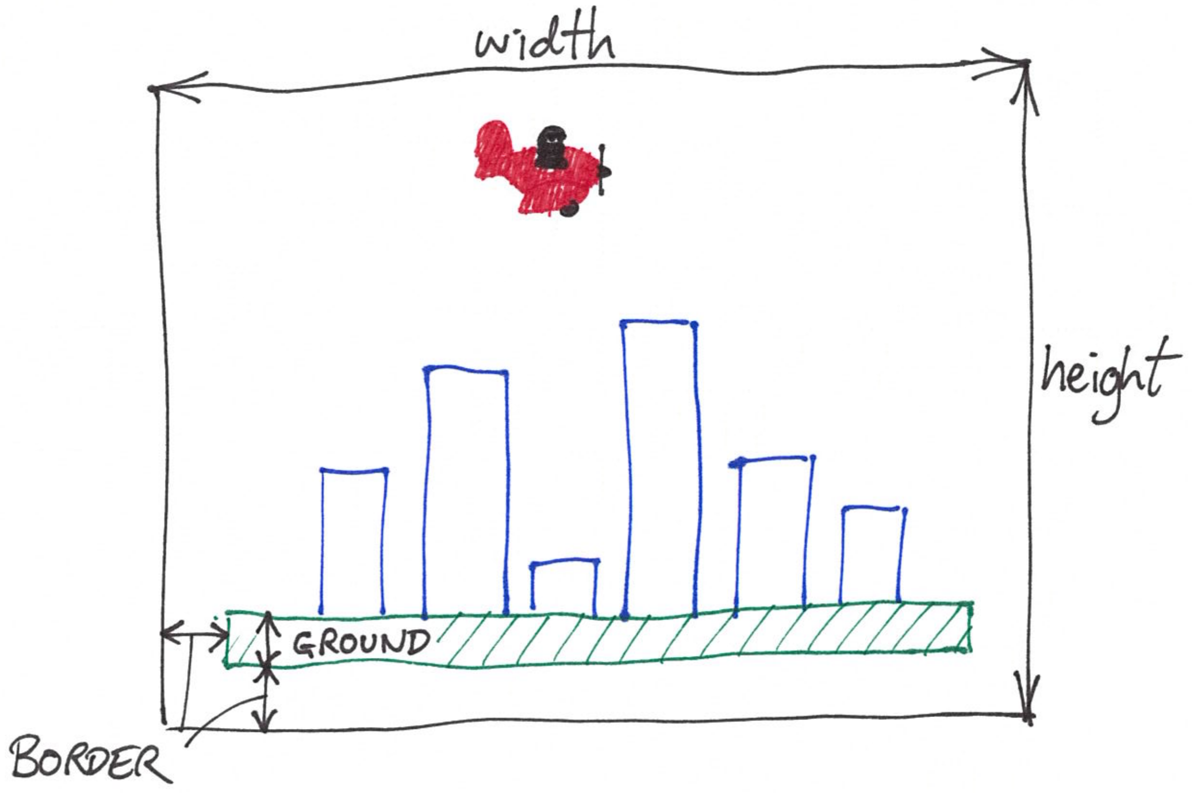
- Introduce programming in Processing 2.0
- Re-create a simple version of Blitz



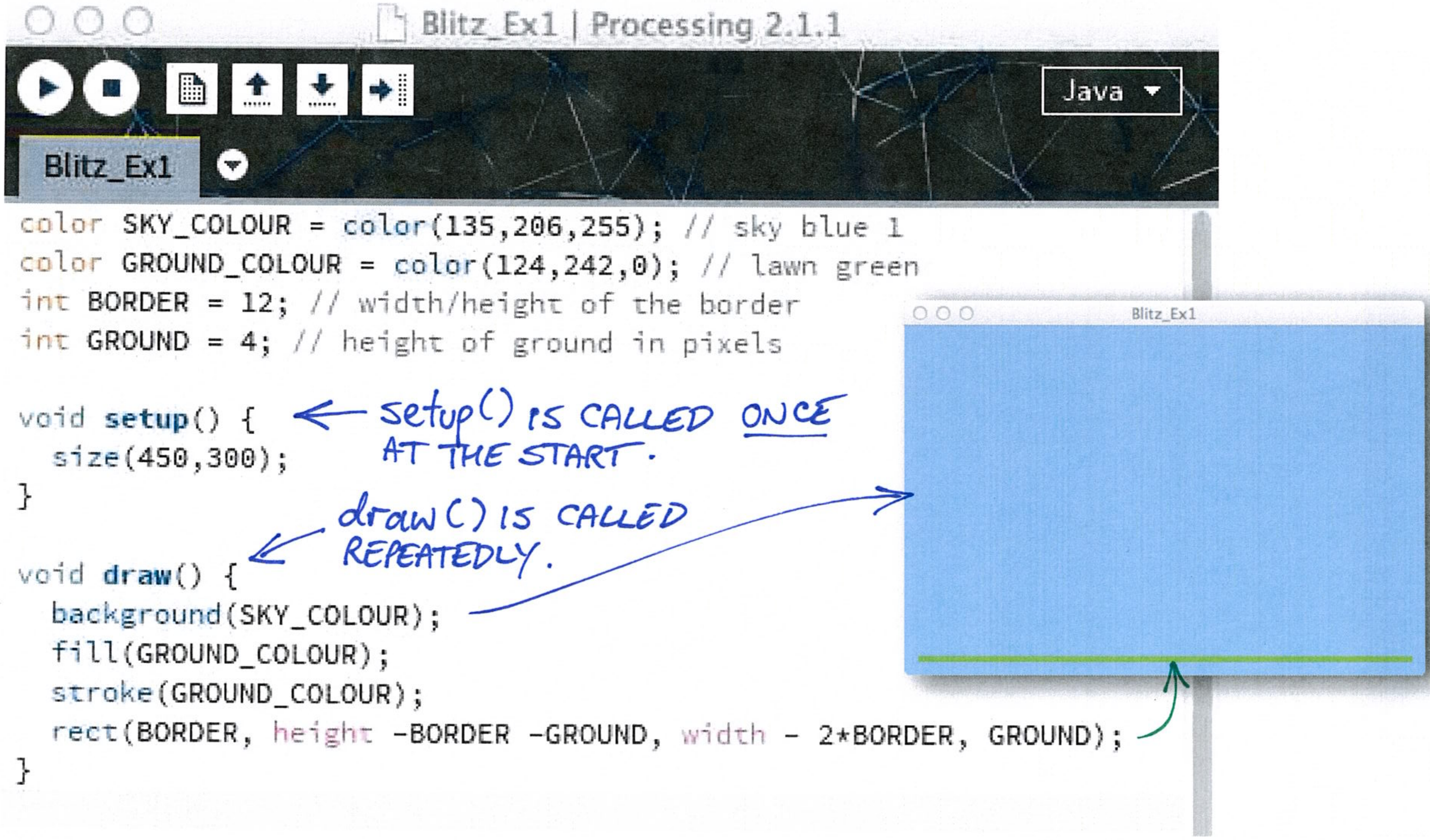
Getting started

- Download Processing:
  http://www.processing.org
- Installation instructions on the forum:
  http://processing.freeforums.org/app-programming-with-processing-f3.html
- Start Processing
- Create a new project:
  File > New
- Save the project as 'Blitz_Ex1':
  File > Save

Sketch out your ideas



Basics: Sky & Ground



```
color SKY_COLOUR = color(135,206,255); // sky blue 1
color GROUND_COLOUR = color(124,242,0); // lawn green
int BORDER = 12; // width/height of the border
int GROUND = 4; // height of ground in pixels

void setup() {
  size(450,300);
}

void draw() {
  background(SKY_COLOUR);
  fill(GROUND_COLOUR);
  stroke(GROUND_COLOUR);
  rect(BORDER, height -BORDER -GROUND, width - 2*BORDER, GROUND);
}
```

setup() IS CALLED ONCE AT THE START.

draw() IS CALLED REPEATEDLY.

Variables

*NOTE THE AMERICAN SPELLING.

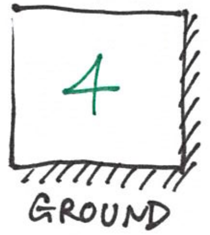
* NO SPACES ALLOWED

DECLARATION AND INITIALIZATION

CAPITALS TYPICALLY INDICATE CONSTANTS

```
color SKY_COLOUR = color(135,206,255); // sky blue 1
color GROUND_COLOUR = color(124,242,0); // lawn green
int BORDER = 12; // width/height of the border
int GROUND = 4; // height of ground in pixels
```

VARIABLES HAVE A TYPE.



VARIABLES ARE LIKE BOXES THAT CAN BE EMPTY, OR CONTAIN A VALUE.

```
PImage image;
int x, y;
boolean falling = false;
```

*SHORT FOR INTEGER (A WHOLE NUMBER)

*GIVE VARIABLES MEANINGFUL NAMES

Expressions

$$\texttt{width} - \underbrace{\texttt{2*BORDER}}_{}$$

* MULTIPLICATION (AND DIVISION) BEFORE SUBTRACTION (AND ADDITION).

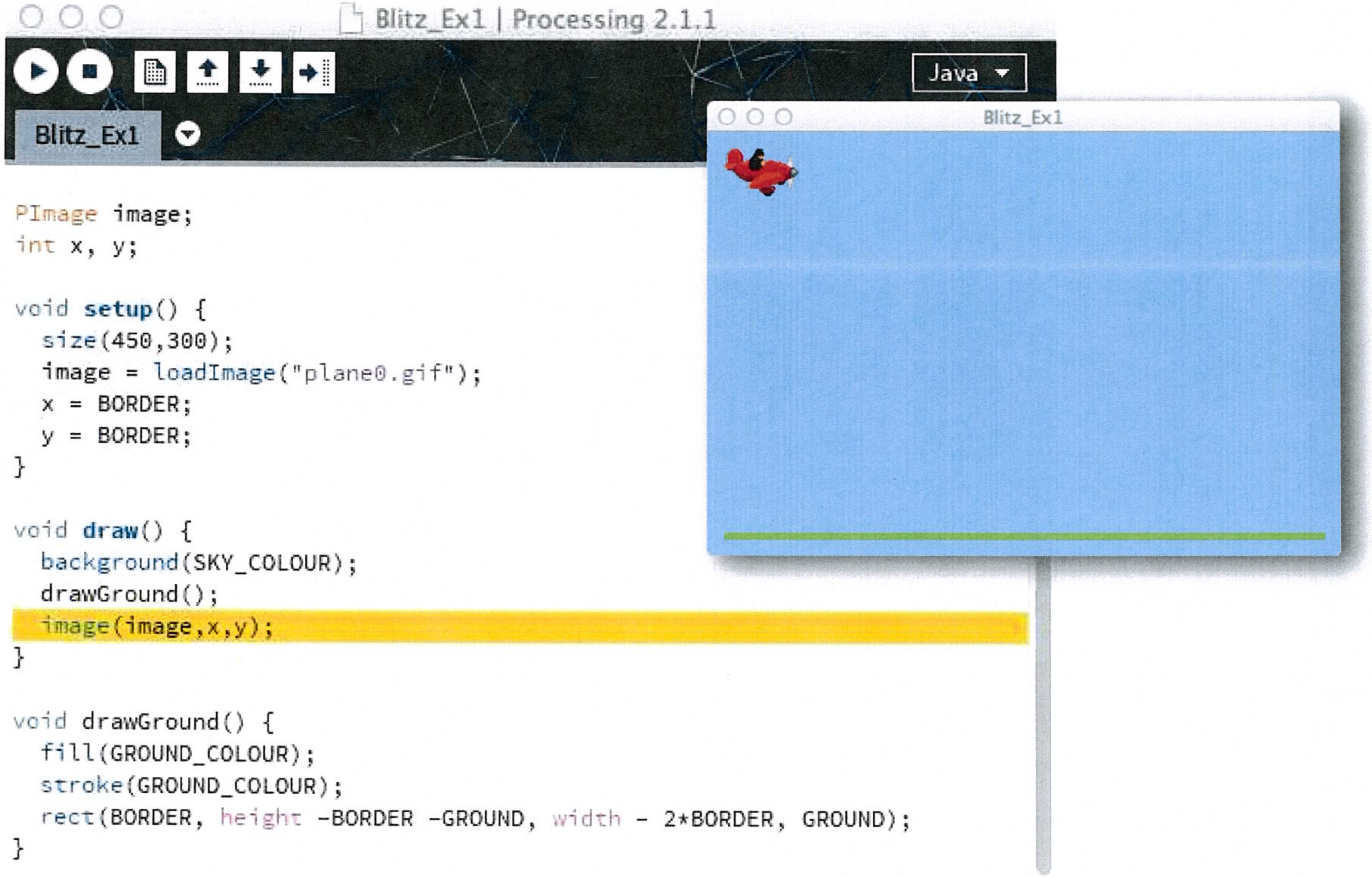
+	add	LOW PRECEDENCE (DO LAST)
-	subtract	
*	times	HIGH PRECEDENCE (DO FIRST)
/	divide	
%	modulo (remainder)	

Graphics files

Download the graphics files

- Go to http://github.com/stevebattle/Blitz
- Click on 'Download ZIP' (bottom righthand corner)
- Extract the ZIP and copy into your Processing folder.
- **Copy** the 'data' folder from Blitz to 'Blitz_Ex1'. This contains the graphics.

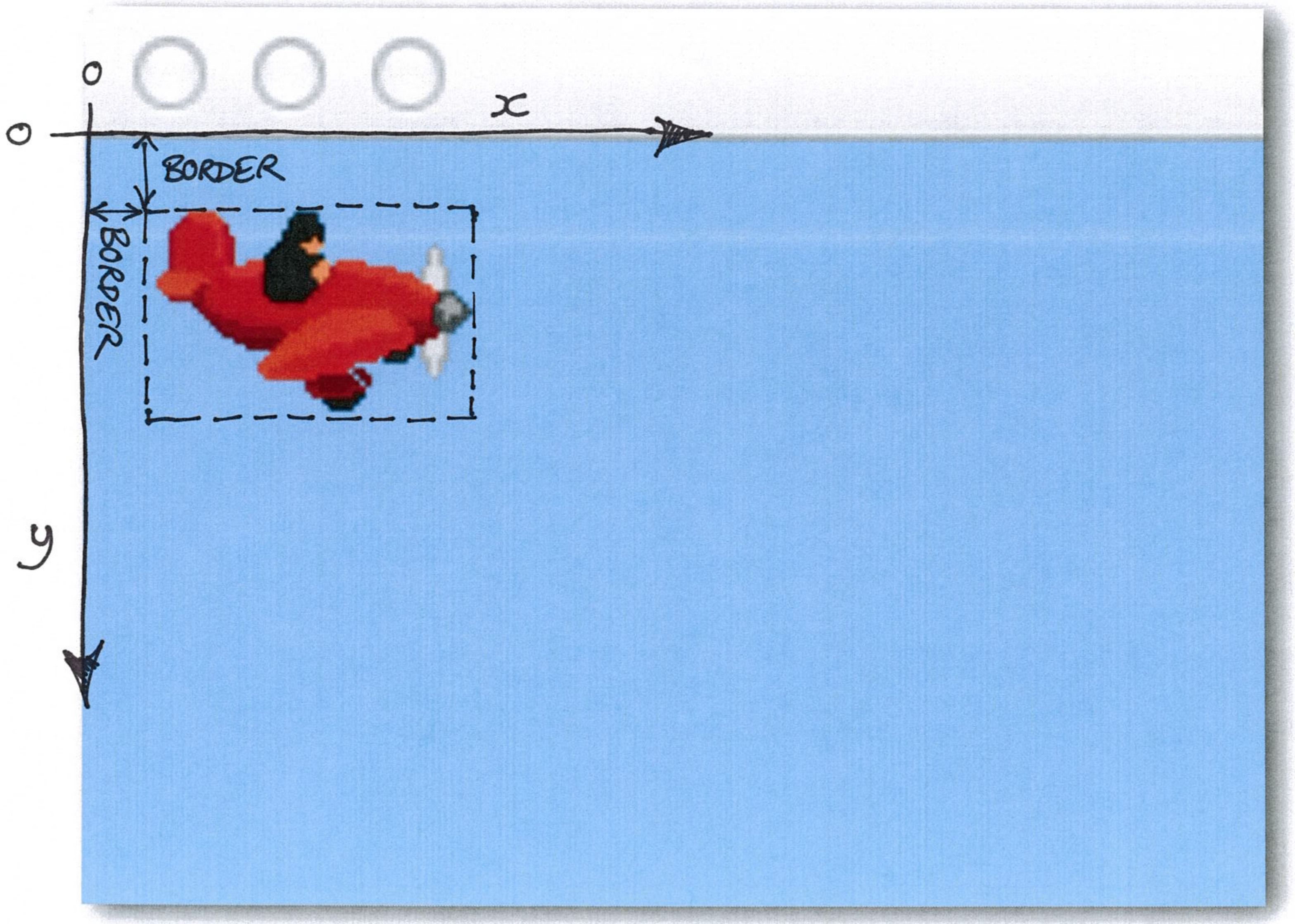
Draw the plane



```
PImage image;
int x, y;

void setup() {
  size(450,300);
  image = loadImage("plane0.gif");
  x = BORDER;
  y = BORDER;
}

void draw() {
  background(SKY_COLOUR);
  drawGround();
  image(image,x,y);
}

void drawGround() {
  fill(GROUND_COLOUR);
  stroke(GROUND_COLOUR);
  rect(BORDER, height -BORDER -GROUND, width - 2*BORDER, GROUND);
}
```

https://gist.github.com/stevebattle/8634913

Co-ordinates



Functions

* FUNCTIONS CAN RETURN A VALUE. THIS ONE DOESN'T

camelCase

```
void drawGround() {
  fill(GROUND_COLOUR);
  stroke(GROUND_COLOUR);
  rect(BORDER, height -BORDER -GROUND, width - 2*BORDER, GROUND);
}
```

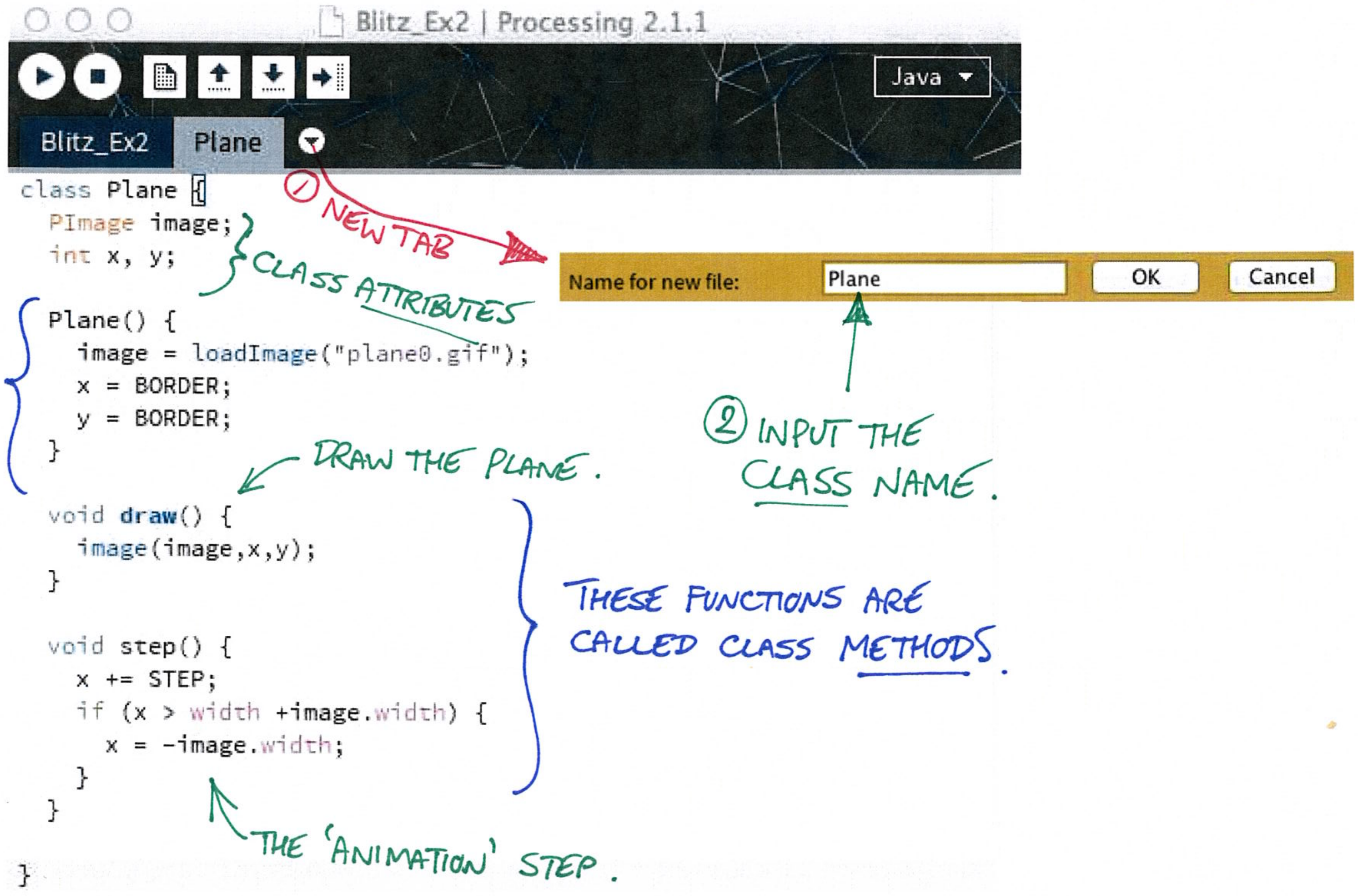
FUNCTION CALLS

THESE ARE FUNCTION ARGUMENTS.

- FUNCTIONS GROUP TOGETHER CODE THAT DOES A PARTICULAR JOB.
- THIS IS A FUNCTION DEFINITION.

Classes

"RECIPIES FOR CREATING NEW OBJECTS."



```
class Plane {
  PImage image;
  int x, y;

  Plane() {
    image = loadImage("plane0.gif");
    x = BORDER;
    y = BORDER;
  }

  void draw() {
    image(image,x,y);
  }

  void step() {
    x += STEP;
    if (x > width +image.width) {
      x = -image.width;
    }
  }

}
```

Name for new file: Plane

① NEW TAB

② INPUT THE CLASS NAME.

CLASS ATTRIBUTES

THIS IS CALLED THE CLASS CONSTRUCTOR
(IT HAS NO RETURN TYPE, NOT EVEN VOID)
(IT HAS THE SAME NAME AS THE CLASS)

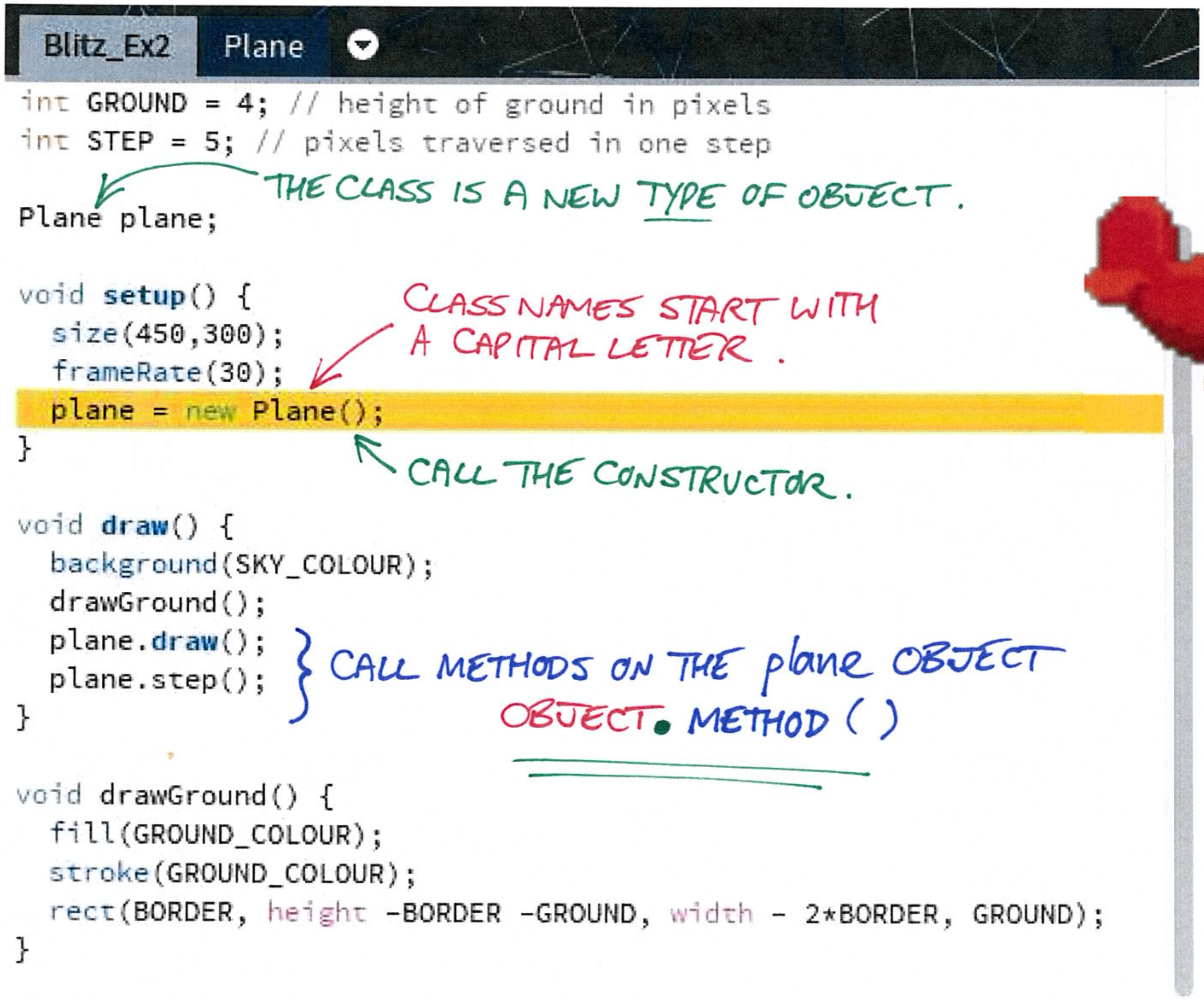
DRAW THE PLANE.

THESE FUNCTIONS ARE CALLED CLASS METHODS.

THE 'ANIMATION' STEP.

https://gist.github.com/stevebattle/8637954#file-plane

Create a plane **object**



```
int GROUND = 4; // height of ground in pixels
int STEP = 5; // pixels traversed in one step

Plane plane;

void setup() {
  size(450,300);
  frameRate(30);
  plane = new Plane();
}

void draw() {
  background(SKY_COLOUR);
  drawGround();
  plane.draw();
  plane.step();
}

void drawGround() {
  fill(GROUND_COLOUR);
  stroke(GROUND_COLOUR);
  rect(BORDER, height -BORDER -GROUND, width - 2*BORDER, GROUND);
}
```

THE CLASS IS A NEW TYPE OF OBJECT.

CLASS NAMES START WITH A CAPITAL LETTER.

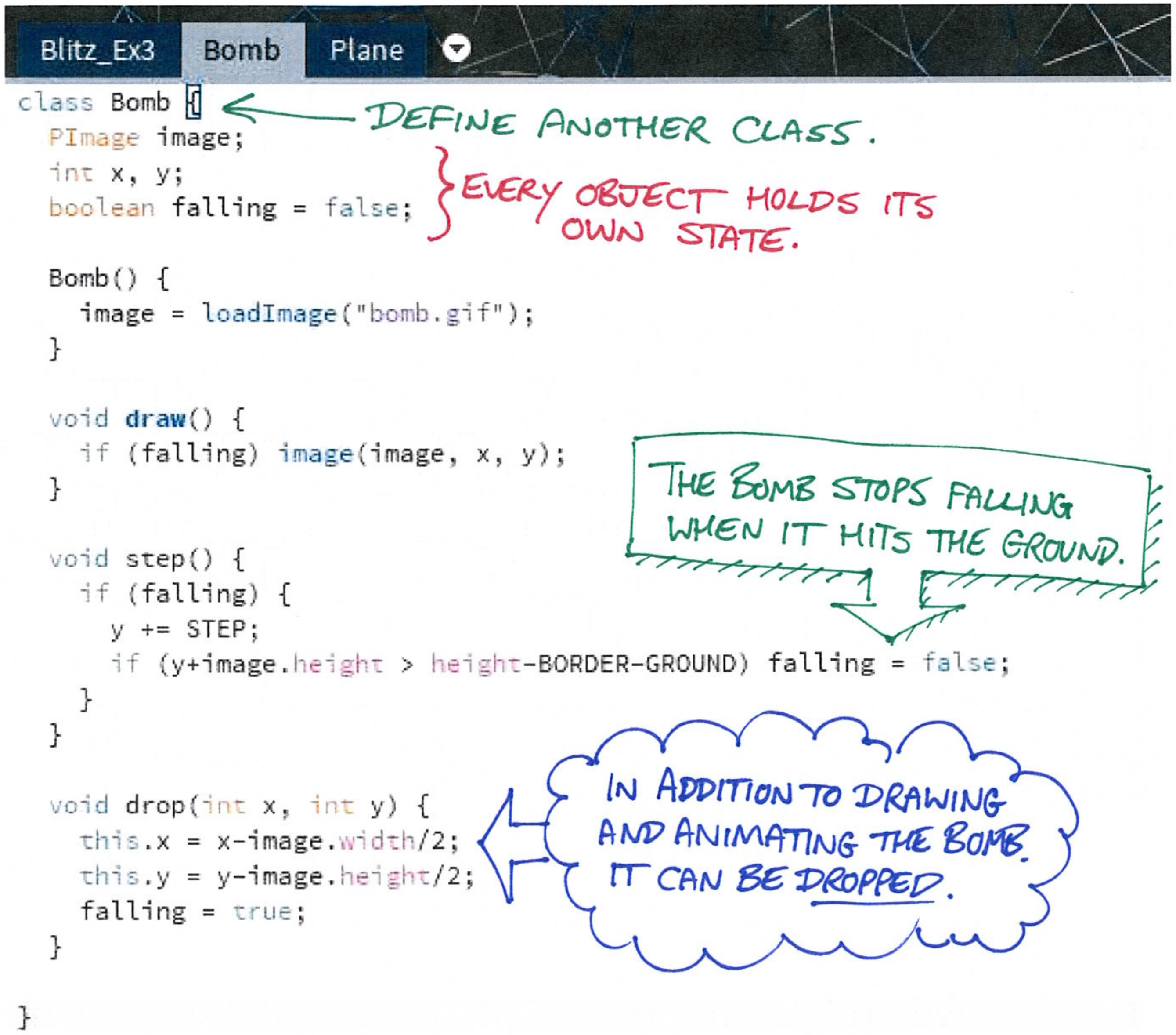
CALL THE CONSTRUCTOR.

CALL METHODS ON THE plane OBJECT

OBJECT. METHOD ( )

https://gist.github.com/stevebattle/8637954

Bombs



```
class Bomb {
  PImage image;
  int x, y;
  boolean falling = false;

  Bomb() {
    image = loadImage("bomb.gif");
  }

  void draw() {
    if (falling) image(image, x, y);
  }

  void step() {
    if (falling) {
      y += STEP;
      if (y+image.height > height-BORDER-GROUND) falling = false;
    }
  }

  void drop(int x, int y) {
    this.x = x-image.width/2;
    this.y = y-image.height/2;
    falling = true;
  }

}
```

DEFINE ANOTHER CLASS.

EVERY OBJECT HOLDS ITS OWN STATE.

THE BOMB STOPS FALLING WHEN IT HITS THE GROUND.

IN ADDITION TO DRAWING AND ANIMATING THE BOMB, IT CAN BE DROPPED.

https://gist.github.com/stevebattle/8638453#file-bomb

if statement

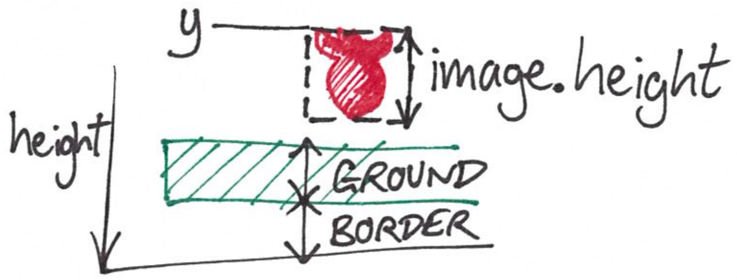
THIS MUST BE TRUE OR FALSE

```
if (CONDITION) ...
else ...
```

SOME CODE. EXECUTED IF TRUE. IF THERE'S MORE THAN ONE INSTRUCTION, IT MUST BE SURROUNDED BY CURLY BRACES {}

OPTIONAL. CODE EXECUTED IF THE CONDITION IS FALSE.

```
if (y+image.height > height-BORDER-GROUND) falling = false;
```



AN ATTRIBUTE OF image. (NO BRACKETS)

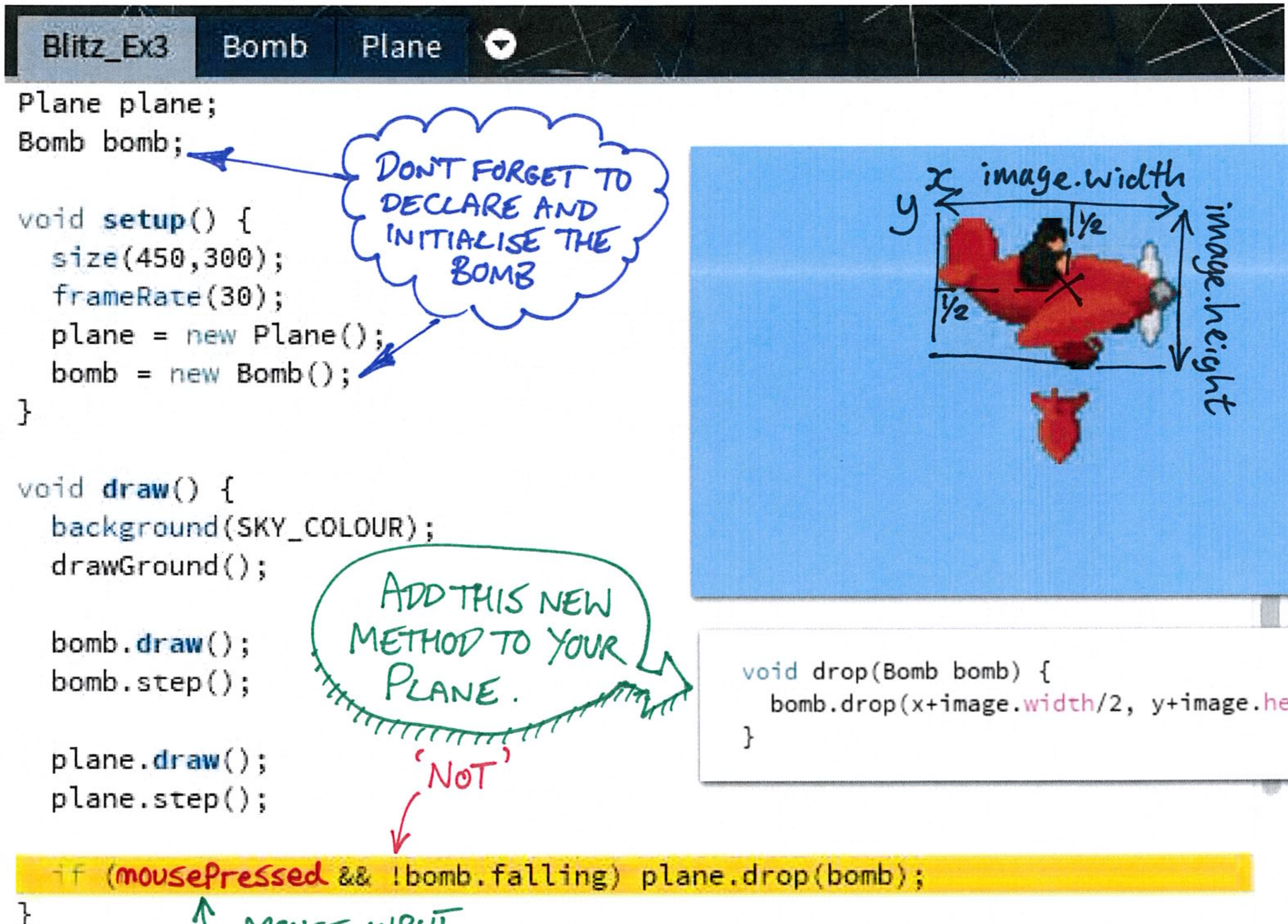
Relational Operators

```
y+image.height > height-BORDER-GROUND
```

THE RESULT OF A RELATIONAL OPERATOR IS TRUE OR FALSE (BOOLEAN).

`<`	*less than*
`>`	*greater than*
`<=`	*less than or equals*
`>=`	*greater than or equals*
`!=`	*not equals*
`==`	*equals*

Drop the bomb



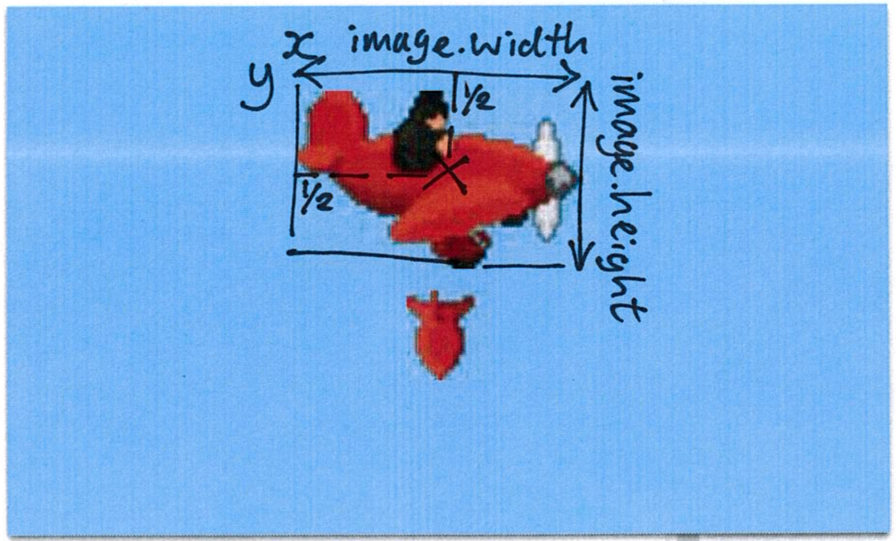
```
Plane plane;
Bomb bomb;

void setup() {
  size(450,300);
  frameRate(30);
  plane = new Plane();
  bomb = new Bomb();
}

void draw() {
  background(SKY_COLOUR);
  drawGround();

  bomb.draw();
  bomb.step();

  plane.draw();
  plane.step();

  if (mousePressed && !bomb.falling) plane.drop(bomb);
}
```

DON'T FORGET TO DECLARE AND INITIALISE THE BOMB

ADD THIS NEW METHOD TO YOUR PLANE.

'NOT'

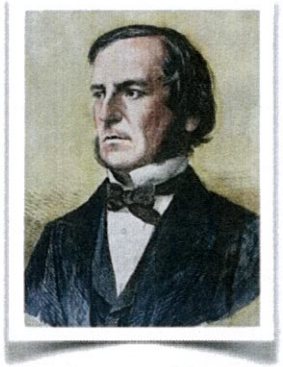
MOUSE INPUT



```
void drop(Bomb bomb) {
  bomb.drop(x+image.width/2, y+image.height/2);
}
```

https://gist.github.com/stevebattle/8638453

Boolean Operators



George Boole
1815-1864

```
mousePressed && !bomb.falling
```

BOOLEAN 'and' 'not' BOOLEAN

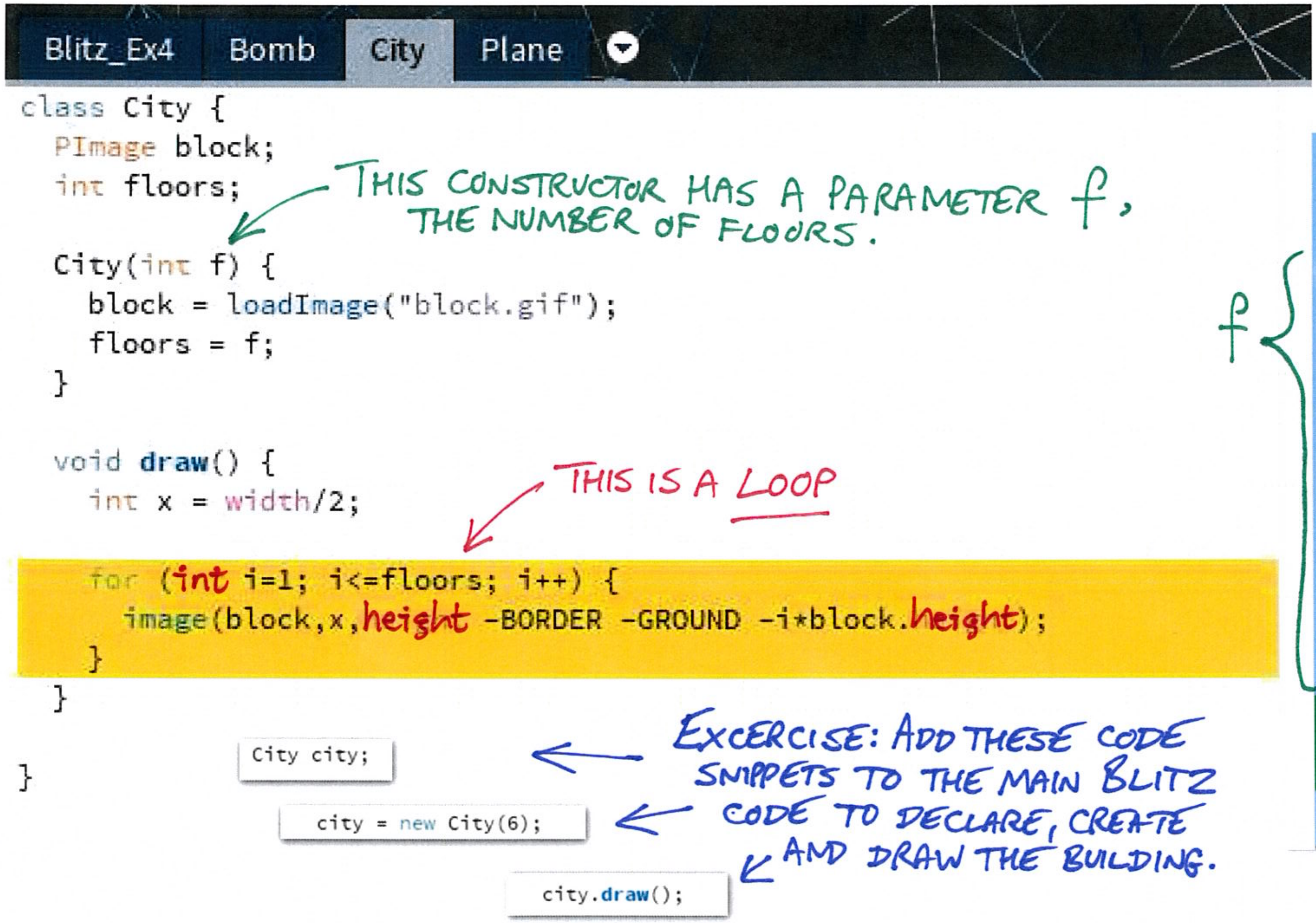
● THE INPUTS TO A BOOLEAN OPERATOR MUST BE BOOLEAN.

● THE OUTPUT OF A BOOLEAN OPERATOR IS A BOOLEAN.

&&	*and*
||	*or*
!	*not*

'NOT' IS A UNARY OPERATOR, AS IT ONLY HAS ONE ARGUMENT.

Draw a building



```
class City {
  PImage block;
  int floors;

  City(int f) {
    block = loadImage("block.gif");
    floors = f;
  }

  void draw() {
    int x = width/2;

    for (int i=1; i<=floors; i++) {
      image(block,x,height -BORDER -GROUND -i*block.height);
    }
  }

}
```

THIS CONSTRUCTOR HAS A PARAMETER f, THE NUMBER OF FLOORS.

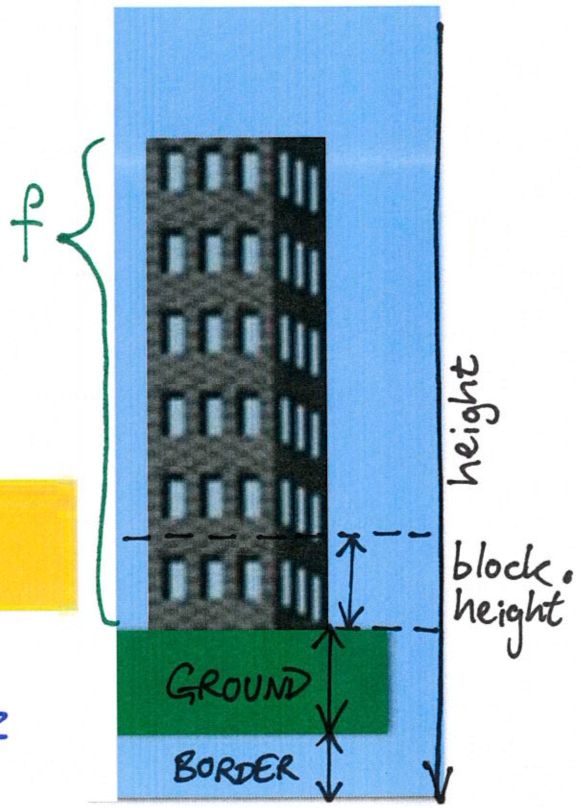
THIS IS A LOOP

```
City city;
```

```
city = new City(6);
```

```
city.draw();
```

EXCERCISE: ADD THESE CODE SNIPPETS TO THE MAIN BLITZ CODE TO DECLARE, CREATE AND DRAW THE BUILDING.



https://gist.github.com/stevebattle/8639123

The *for* loop

INITIALIZATION HAPPENS ONCE AT THE START OF THE LOOP.

THE TEST IS EVALUATED AT THE START OF EACH ITERATION.

```
for (INITIALIZE; TEST; INCREMENT) {
    ...
}
```

THE INCREMENT OCCURS AT THE END OF EACH ITERATION.

DECLARE AND INITIALIZE THE LOOP VARIABLE

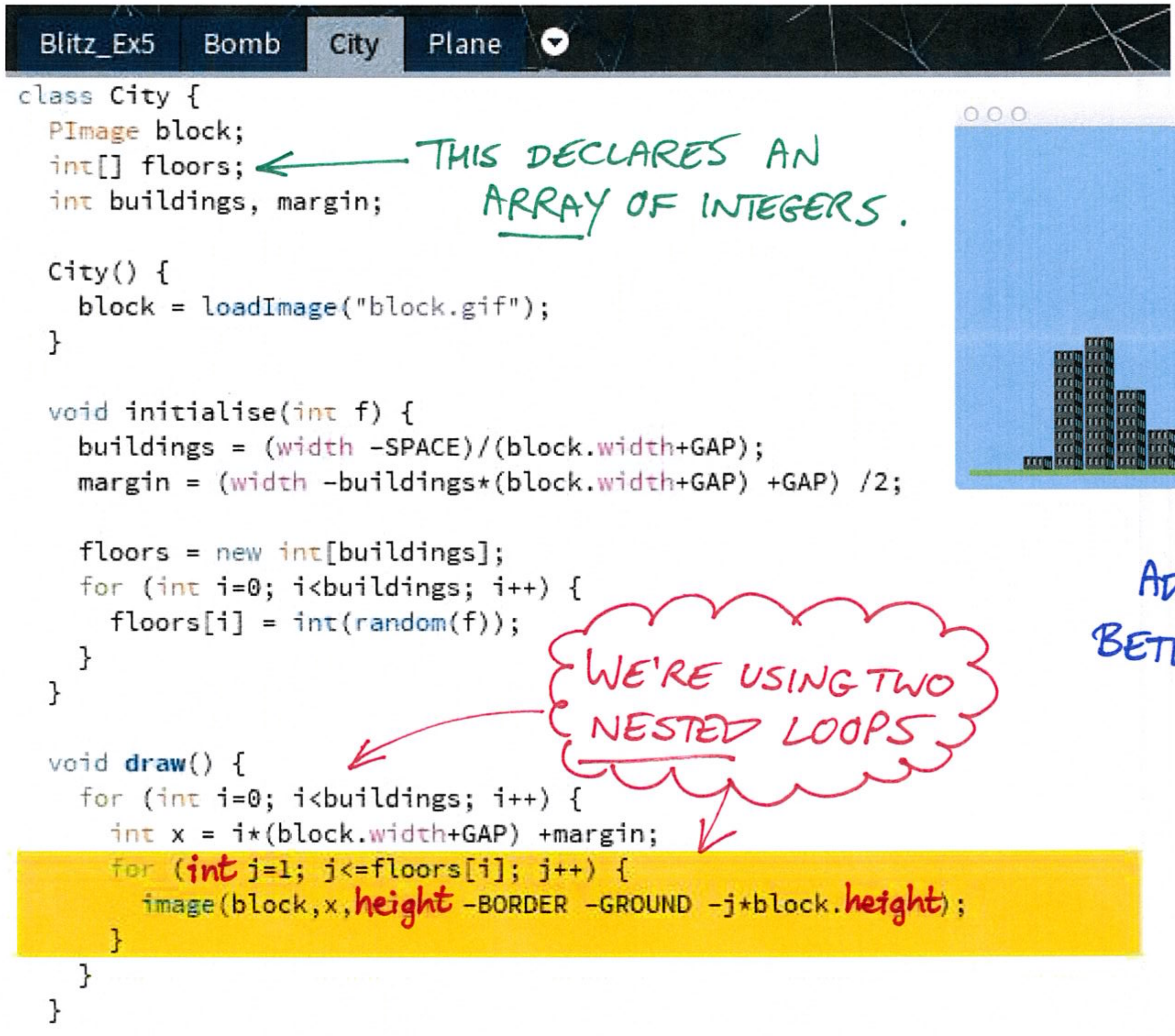
AS THIS LOOP STARTS AT 1, TEST FOR '≤' TO INCLUDE ALL FLOORS.

```
for (int i=1; i<=floors; i++) {
  image(block,x,height -BORDER -GROUND -i*block.height);
}
```

ADD ONE. SAME AS $i = i + 1$

∗ WE EXIT THE LOOP WHEN THE TEST IS FALSE.

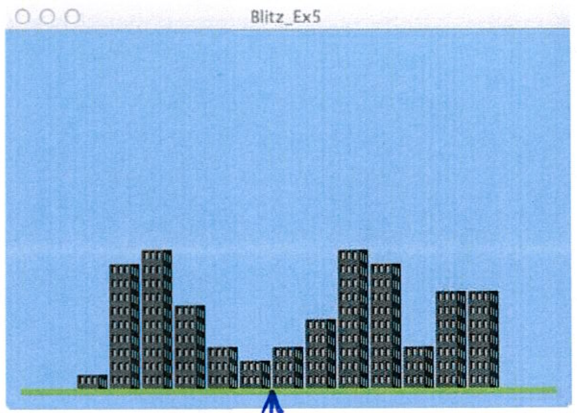
Draw the city



```
class City {
  PImage block;
  int[] floors;
  int buildings, margin;

  City() {
    block = loadImage("block.gif");
  }

  void initialise(int f) {
    buildings = (width -SPACE)/(block.width+GAP);
    margin = (width -buildings*(block.width+GAP) +GAP) /2;

    floors = new int[buildings];
    for (int i=0; i<buildings; i++) {
      floors[i] = int(random(f));
    }
  }

  void draw() {
    for (int i=0; i<buildings; i++) {
      int x = i*(block.width+GAP) +margin;
      for (int j=1; j<=floors[i]; j++) {
        image(block,x,height -BORDER -GROUND -j*block.height);
      }
    }
  }

}
```

THIS DECLARES AN ARRAY OF INTEGERS.

WE'RE USING TWO NESTED LOOPS

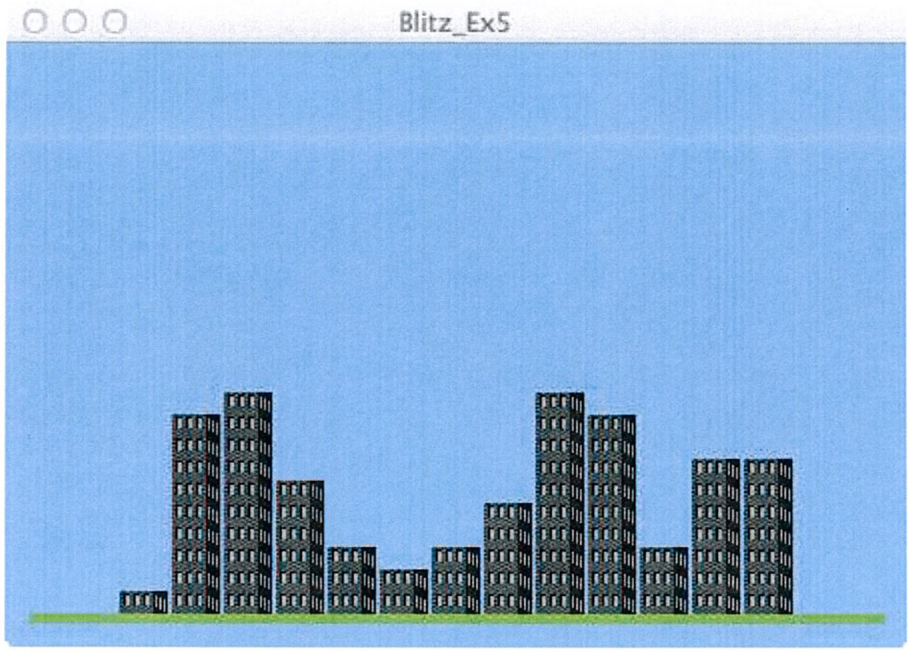


ADD A GAP BETWEEN BUILDINGS.

https://gist.github.com/stevebattle/8639461#file-city

Arrays

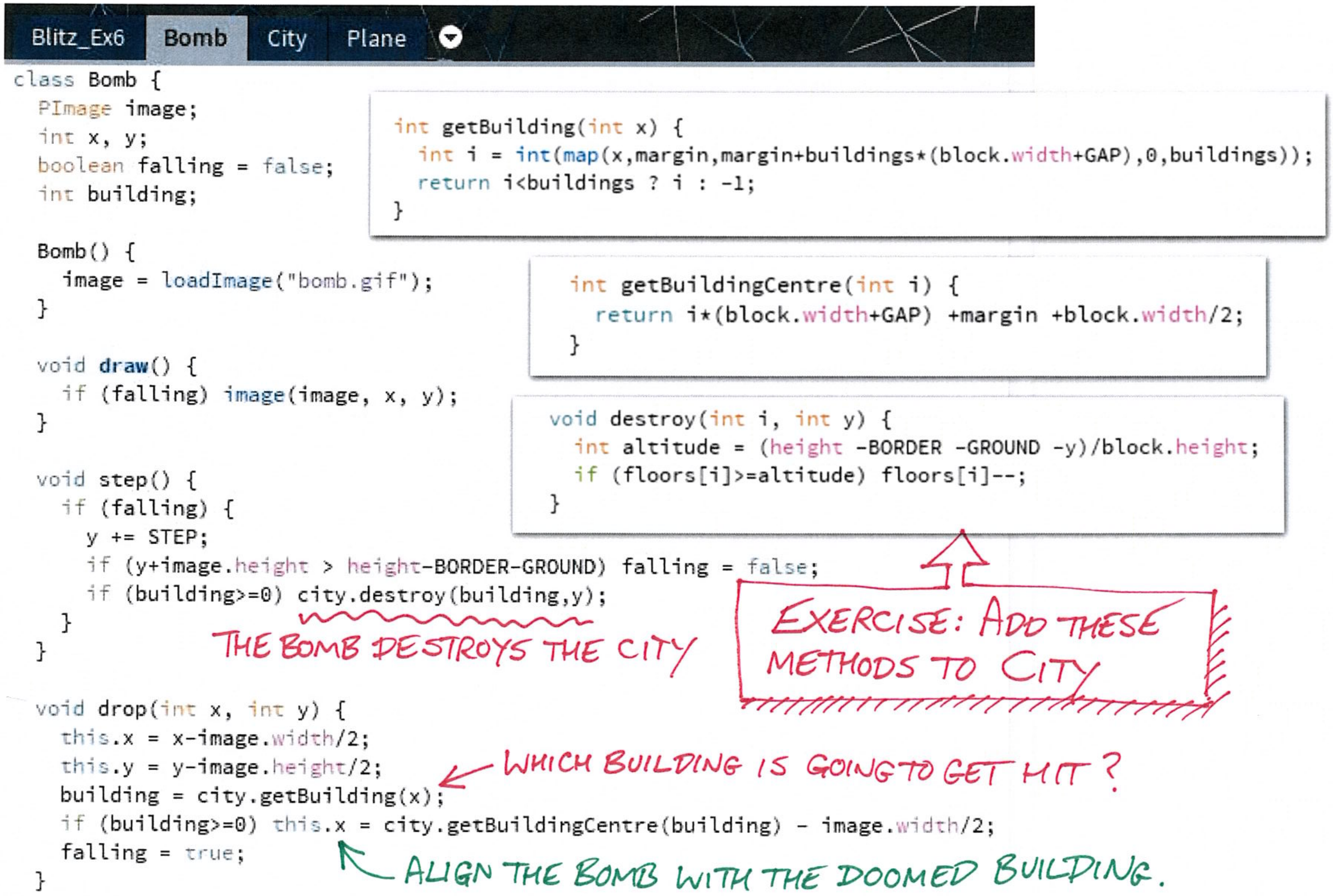
```
int[] floors;
```



floors	1	9	10	6	3	2	3	5	10	9	3	7	7
	0	1	2	3	4	5	6	7	8	9	10	11	12

* THE ARRAY INDEX.

Destroy the city



```
class Bomb {
  PImage image;
  int x, y;
  boolean falling = false;
  int building;

  Bomb() {
    image = loadImage("bomb.gif");
  }

  void draw() {
    if (falling) image(image, x, y);
  }

  void step() {
    if (falling) {
      y += STEP;
      if (y+image.height > height-BORDER-GROUND) falling = false;
      if (building>=0) city.destroy(building,y);
    }
  }

  void drop(int x, int y) {
    this.x = x-image.width/2;
    this.y = y-image.height/2;
    building = city.getBuilding(x);
    if (building>=0) this.x = city.getBuildingCentre(building) - image.width/2;
    falling = true;
  }
}
```

```
int getBuilding(int x) {
  int i = int(map(x,margin,margin+buildings*(block.width+GAP),0,buildings));
  return i<buildings ? i : -1;
}
```

```
int getBuildingCentre(int i) {
  return i*(block.width+GAP) +margin +block.width/2;
}
```

```
void destroy(int i, int y) {
  int altitude = (height -BORDER -GROUND -y)/block.height;
  if (floors[i]>=altitude) floors[i]--;
}
```

THE BOMB DESTROYS THE CITY

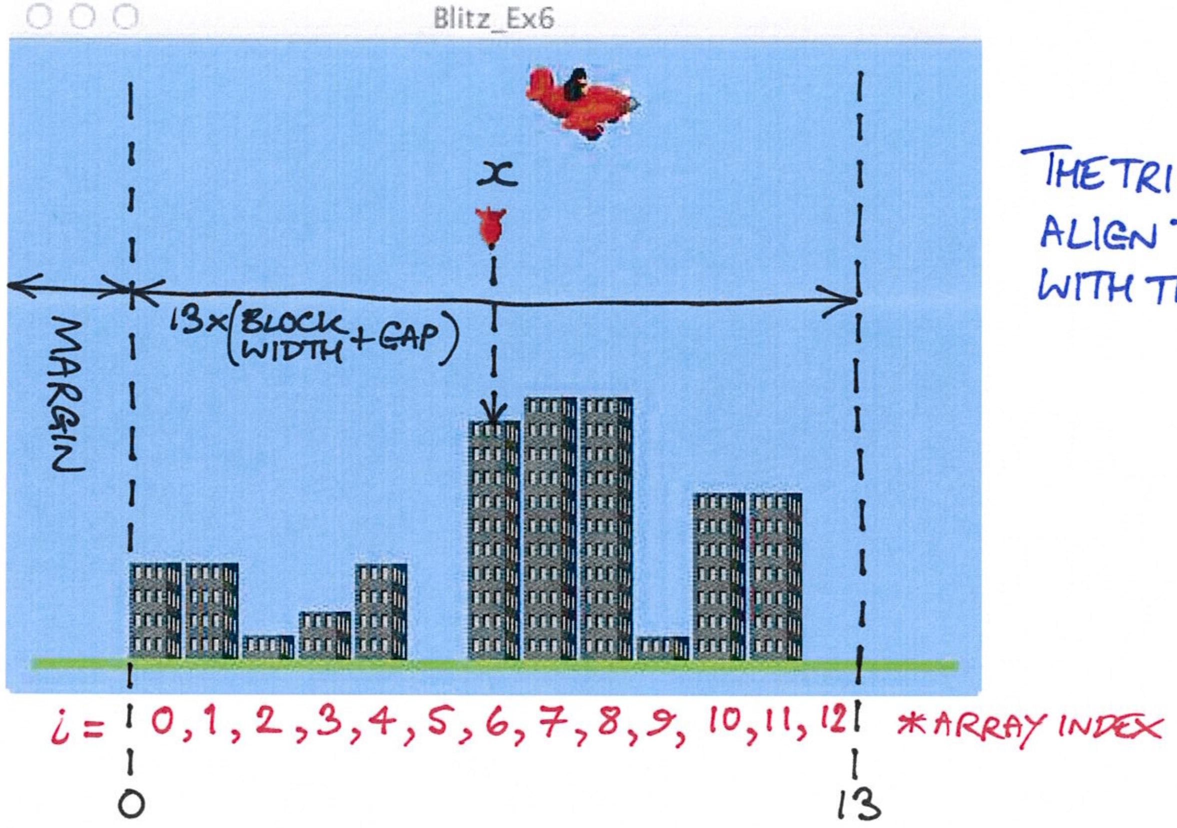
EXERCISE: ADD THESE METHODS TO CITY

WHICH BUILDING IS GOING TO GET HIT?

ALIGN THE BOMB WITH THE DOOMED BUILDING.

https://gist.github.com/stevebattle/8640176

Map x back to i



THE TRICK IS TO ALIGN THE BOMB WITH THE BUILDING.

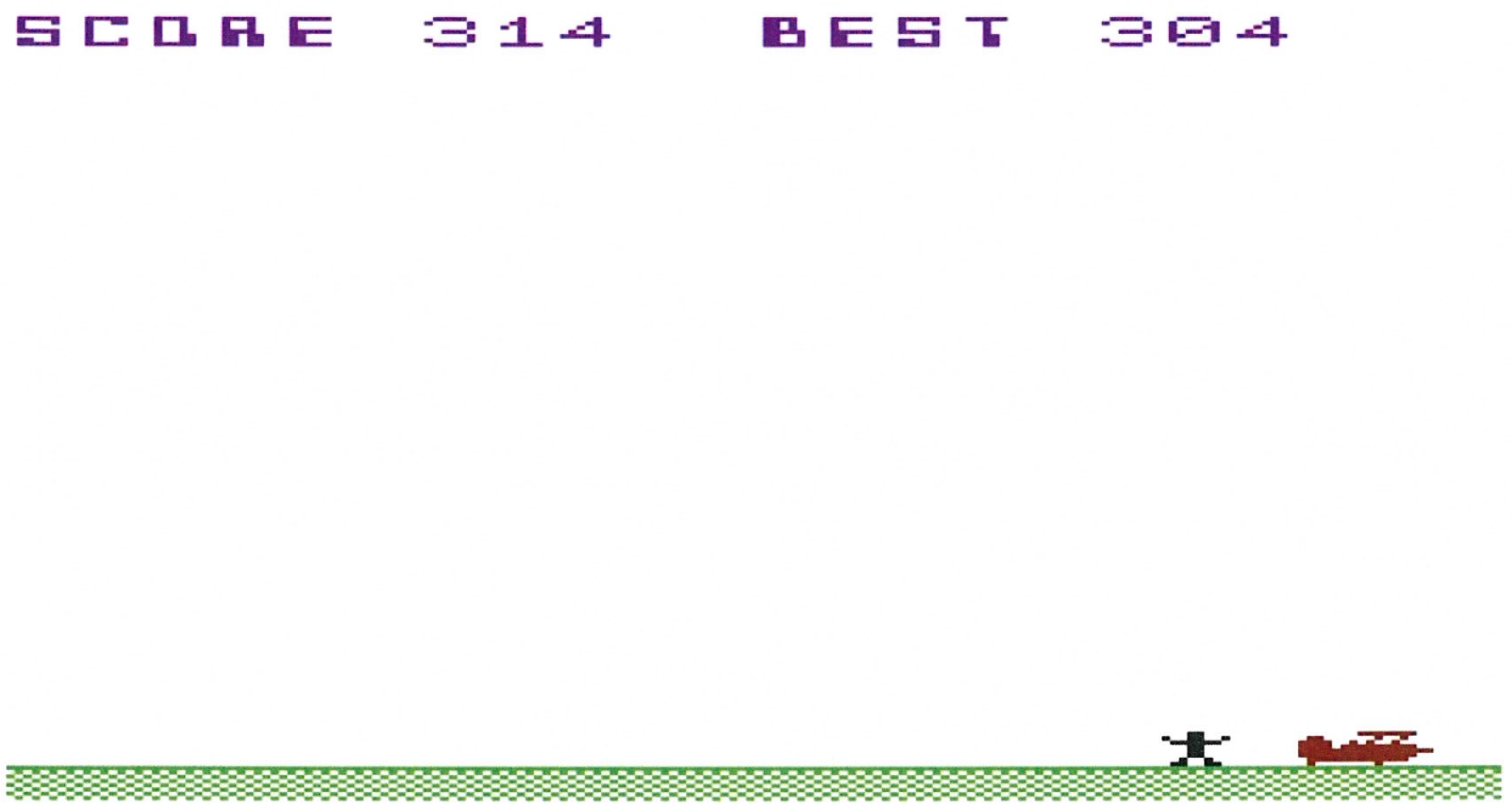
THE EXTREMES OF x THE EXTREMES OF i

13

map(x,margin,margin+~~buildings~~*(block.width+GAP),0,buildings)

Links

- Download the game:
  http://github.com/stevebattle/Blitz
- Forum:
  http://processing.freeforums.org
- Wikipedia:
  http://en.wikipedia.org/wiki/Blitz_(video_game)
- Processing 2:
  http://www.processing.org



http://bestretrogames.blogspot.co.uk/2012/01/blitz-commodore-vic-20-1981.html