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
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:

• Start Processing

• Create a new project:
  File > New

• Save the project as ‘Blitz_Ex1’:
  File > Save
Sketch out your ideas
Basics: Sky & Ground

```java
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);
}
```
Variables

Note the American spelling.

*No spaces allowed

Declaration and initialization

Capital letters typically indicate constants.

Variables have a type.

PaintImage image;

int x, y;

boolean falling = false;

*Give variables meaningful names

*Short for integer (a whole number)

*Variables are like boxes that can be empty, or contain a value.
Expressions

\[
\text{width} - 2 \times \text{BORDER}
\]

- Multiplication (and division) before subtraction (and addition).

+ add

- subtract

* times

/ 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.
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.

```java
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

```java
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;
        }
    }
}
```

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

```java
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);
}
```

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

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

if (CONDITION) ...
else ...

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

\[ y + \text{image.height} > \text{height}-\text{BORDER-GROUND} \]

\[
\begin{align*}
\text{\textgreater} & \quad \text{less than} \\
\text{\textgreater\textless} & \quad \text{greater than} \\
\text{\textless\textgreater} & \quad \text{less than or equals} \\
\text{\textgreater\textless=} & \quad \text{greater than or equals} \\
\text{\textasciitilde=} & \quad \text{not equals} \\
\text{==} & \quad \text{equals}
\end{align*}
\]

\textit{The result of a relational operator is true or false (boolean).}
Drop the bomb

```java
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);
}

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

https://gist.githubusercontent.com/stevebattle/8638453
Boolean Operators

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. |

! 'not' is a unary operator, as it only has one argument.

&& and
|| or
!

George Boole
1815-1864
Draw a building

```java
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);
        }
    }
}
```

City city;

```java
    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

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

- **Initialization** happens once at the start of the loop.
- **Test** is evaluated at the start of each iteration.
- The increment occurs at the end of each iteration.

- Declare and initialize the loop variable:
  ```java
  for (int i=1; i<=floors; i++) {
      ...
  }
  ```

- As this loop starts at 1, test for `<=` to include all floors.
- Add one. Same as `i = i + 1`.

- We **exit** the loop when the test is **false**.
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);
            }
        }
    }
}

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

```java
int[] floors;
```

*THE ARRAY INDEX.*
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.

$\text{map}(x, \text{margin}, \text{margin}+\text{buildings}*(\text{block.width}+\text{GAP}), 0, \text{buildings})$
Links

• Download the game:
  http://github.com/stevebattle/Blitz

• Forum:
  http://processing.freeforums.org

• Wikipedia:

• Processing 2:
  http://www.processing.org