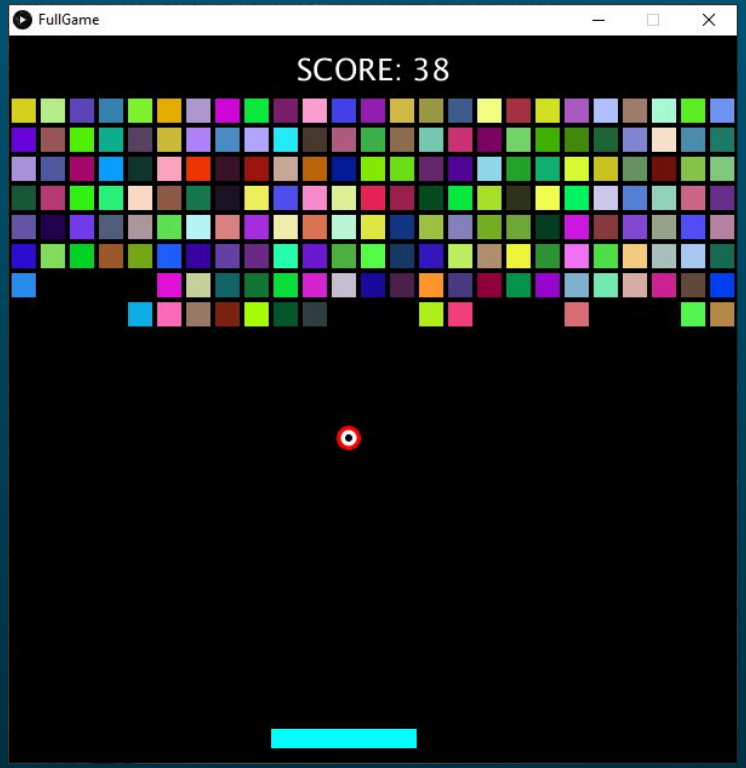
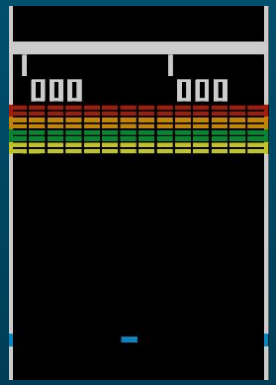
Week 11 - Interaction I
Breakout in Processing

by Mike Wong. School of Creative Media

Breakout in Processing





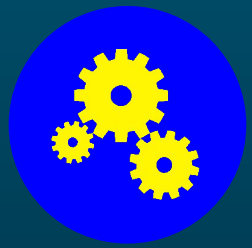
Breakout, a game released in 1976 is one of the earliest video game inspired by **PONG**.

Breakout on Wikipedia

Elements in Breakout

Ball

Brick
(paddle)



GameEngine

Objects in our Breakout

Float2

DATA
a pair of `float`
e.g. coord: (x,y)

Main Objects in our Breakout

Float2

DATA
a pair of `float`
e.g. coord: (x,y)

Ball

DATA
pos, speed and
color of a Ball.

METHODS
bounce & display

Main Objects in our Breakout

Float2

DATA
a pair of `float`
e.g. coord: (x,y)

Ball

DATA
pos, speed and color of a Ball.

METHODS
bounce & display

Brick

DATA
pos & color of a Brick.
Brick is 'active' or not

METHODS
display, check if a given point **HITs** the brick, and the direction of **HIT**.

Main Objects in our Breakout

Float2

DATA
a pair of `float`
e.g. coord: (x,y)

Ball

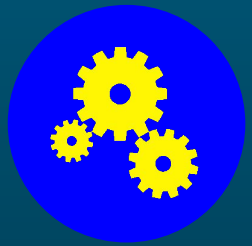
DATA
pos, speed and color of a Ball.

METHODS
bounce & display

Brick

DATA
pos & color of a Brick. Brick is 'active' or not

METHODS
display, check if a given point **HITs** the brick, and the direction of **HIT**.

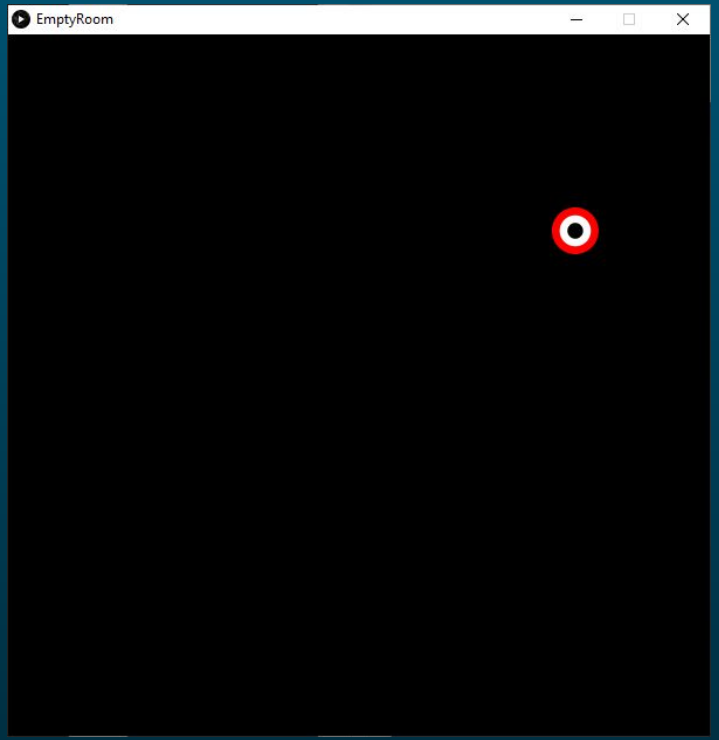


GameEngine

DATA
Ball, Paddle & Bricks, score, size of the room

METHODS
Manage the game logic, e.g. what to do with the ball, paddle & the bricks when they hit. Maintain the score etc.

Step 1 - Empty Room



GOAL

An empty room in which the Ball bounces around, and the game ends when it reaches the bottom.

CODE

the following basic objects:

- `Float2`
- `Ball`
- `GameEngine (BASIC)`

Step 1 - Empty Room (Float2)

EmptyRoom | Ball | Float2 | GameEngine

```
1 class Float2 {
2   float x, y;
3   Float2(float px, float py) {
4     x = px;
5     y = py;
6   }
7 }
```

Float2

as (x,y) coordinate will be used frequently in our sketch; this simple simple object improves the readability of our code.

Step 1 - Empty Room (Ball)

EmptyRoom | Ball | Float2 | GameEngine

```
1  class Ball {
2
3    Float2 pos;       //  Ball position
4    Float2 speed;     //  velocity in X, Y direction
5    float radius;     //  radius
6    color col;
7
8    Ball(Float2 p, Float2 s, float r, color pcol) {
9      pos    = p;
10     speed  = s;
11     radius = r;
12     col    = pcol;
13   }
14
```

Ball
Class of our Ball object.

DATA
Ball's current position, speed, radius and colors.

Step 1 - Empty Room (Ball)

```
14
15    void update() {
16      pos.x += speed.x;
17      pos.y += speed.y;
18    }
19
20    void bounceX(float bx) {
21      pos.x    = bx;
22      speed.x  = -speed.x;
23    }
24
25    void bounceY(float by) {
26      pos.y    = by;
27      speed.y  = -speed.y;
28    }
29
30    void display() {
31      ellipseMode(CENTER);
32      noStroke();
33      fill(col);
34      circle(pos.x, pos.y, radius * 2);
35    }
36
37  }
```

Ball

Class of our Ball object.

METHODS

update() refreshes the pos.

bounceX() & **bounceY()** update the Ball's position, and change its direction of movement.

display() draws the ball.

Step 1 - Empty Room (GameEngine)

EmptyRoom | Ball | Float2 | GameEngine

```
class GameEngine {

  boolean gameover;
  float rTOP, rLEFT, rRIGHT, rBOTTOM;
  Ball ball;

  GameEngine(Float2 ULC, Float2 LRC, float bRadius) {
    rTOP    = ULC.y + bRadius;
    rLEFT   = ULC.x + bRadius;
    rRIGHT  = LRC.x - bRadius;
    rBOTTOM = LRC.y - bRadius;
    gameover = false;

    //  Create the bouncing ball
    Float2 bCenter = new Float2( (rLEFT + rRIGHT)*0.5, (rTOP + rBOTTOM)*0.5 );
    Float2 bSpeed  = new Float2(random(1,4), random(-2,-4));
    ball = new Ball(bCenter, bSpeed, bRadius, color(255,0,0));
  }
```

GameEngine

Class of our GameEngine object.

DATA

gameover : the game state.

rTOP, **rLEFT**, **rRIGHT**, **rBOTTOM** : boundaries of the empty room.

ball : instance of Ball in our game.

Step 1 - Empty Room (GameEngine)

```
20  boolean insideROOM() {
21    //  Test LEFT & RIGHT
22    if (ball.pos.x <= ball.radius) ball.bounceX(rLEFT);
23    else if (ball.pos.x >= rRIGHT) ball.bounceX(rRIGHT);
24    //  Test TOP & BOTTOM
25    if (ball.pos.y <= rTOP) ball.bounceY(rTOP);
26    else if (ball.pos.y >= rBOTTOM) return false;
27    return true;
28  }
29
30  void checkHit() {
31    if (!insideROOM()) gameover = true;
32  }
33
34  //  Update the Game State
35  void update() {
36    ball.update();
37    checkHit();
38  }
39
40  void display() {
41    if (gameover) background(120,0,0);
42    else background(0);
43    ball.display();
44    if (gameover) {
45      background(120,0,0);
46      fill(255);
47      text("GAME OVER", width/2, height/2);
48      noLoop();
49    }
50  }
```

GameEngine
Class of our GameEngine object.

METHODS

`insideROOM()` : checks if the ball is still inside the room, and make it bounce if it touches the boundary.

`checkHit()` : checks if the ball hits anything.

`update()`, `display()` : update and display the game contents.

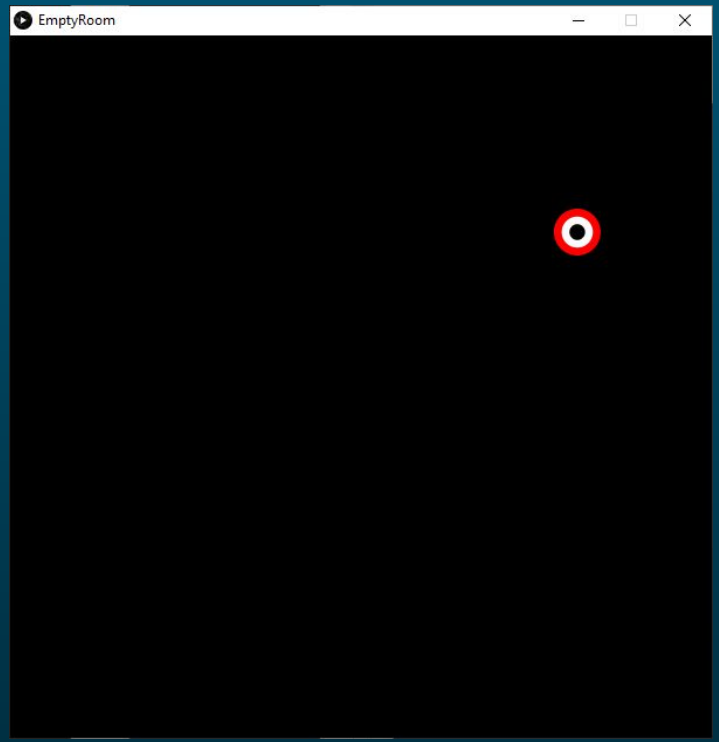
Step 1 - Empty Room (Main Sketch)

EmptyRoom | Ball | Float2 | GameEngine

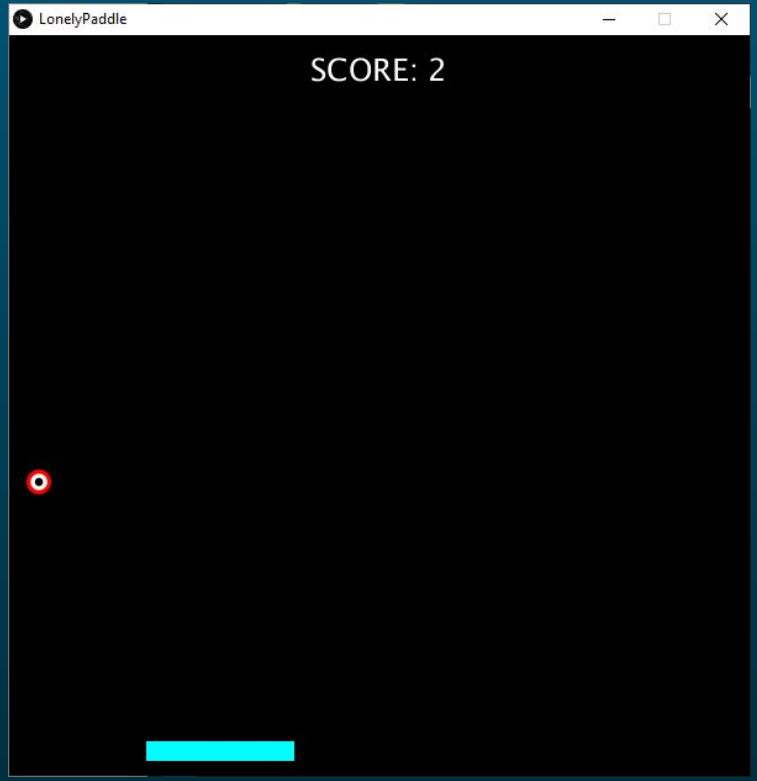
```
1  GameEngine game;
2
3  void setup() {
4    size(600, 600);
5    textAlign(CENTER, CENTER);
6    textSize(20);
7    game = new GameEngine( new Float2(0,0), new Float2(width,height), 20);
8  }
9
10 void draw() {
11   game.update();
12   game.display();
13 }
```

creates an instance of the game engine, and invokes the **`update()`** & **`display()`** methods in the main **`draw()`** call.

Empty Room Demo



Step 2 - Lonely Paddle



GOAL

Add a paddle for the player to bounce the ball, and maintain a score.

CODE

the following basic objects:

- `Brick (as our paddle)`
- `GameEngine (Update)`

Step 2 - Lonely Paddle (Brick)

LonelyPaddle | Ball | Brick | Float2 | GameEngine

```
class Brick {

  Float2 pos;
  Float2 radius;
  color col;
  boolean on;

  Brick(Float2 p, Float2 r, color pcol, boolean onOff) {
    pos    = p;
    radius = r;
    col    = pcol;
    on     = onOff;
  }

```

Brick

Class of our Brick object.

DATA

Brick's current position, radius, color and whether its active or not.

Step 2 - Lonely Paddle (Brick)

```
  void display() {
    if (on) {
      rectMode(CENTER);
      fill(col);
      rect(pos.x,pos.y, radius.x+radius.x, radius.y+radius.y);
    }
  }

  boolean hit(Ball b) {
    if (!on) return false;
    float xDist = abs(b.pos.x - pos.x);
    float yDist = abs(b.pos.y - pos.y);
    if ((xDist <= (radius.x + b.radius)) && (yDist <= (radius.y + b.radius))) {
      return true;
    }
    return false;
  }

  boolean verticalBounce(Ball b) {
    float lbx = b.pos.x - b.speed.x;
    float lby = b.pos.y - b.speed.y;
    float alpha = 0;
    float hitX;

    alpha = ((pos.y - radius.y) - lby) / b.speed.y;
    hitX = lerp(lbx, b.pos.x, alpha);

    if (abs(hitX - pos.x) >= (radius.x + b.radius)) {
      return false;
    }
    return true;
  }

}
```

Brick

Class of our Brick object.

METHODS

`hit()` checks if the input Ball hits the brick.

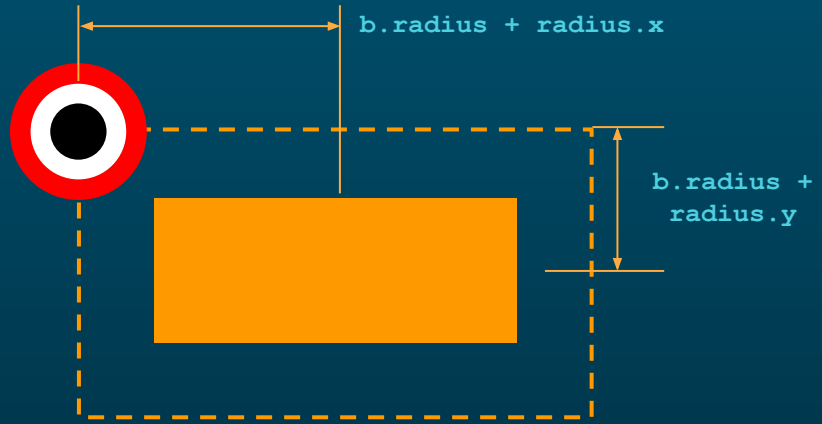
`verticalBounce()` checks if the hit is in the Y-direction.

`display()` displays the brick.

Step 2 - Lonely Paddle (Brick)

hit()

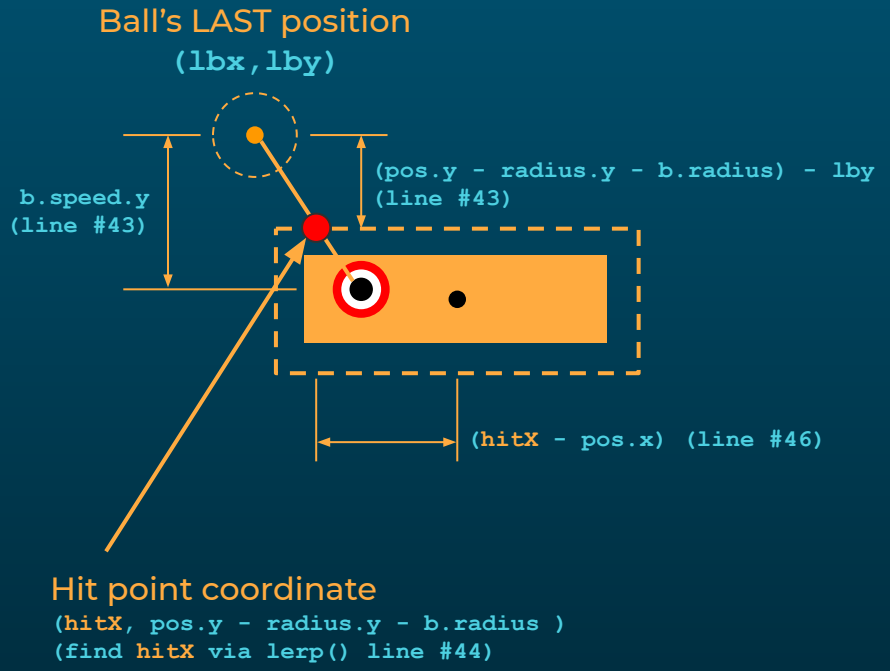
```
24  //  Checks if input Ball hits the brick or not
25  boolean hit(Ball b) {
26    if (!on) return false;
27    float xDist = abs(b.pos.x - pos.x);
28    float yDist = abs(b.pos.y - pos.y);
29    float border = sin(radians(45)) * b.radius;
30    if ((xDist <= (radius.x + border)) && (yDist <= (radius.y + border))) {
31      return true;
32    }
33    return false;
34  }
35
```



Step 2 - Lonely Paddle (Brick)

verticalBounce()

```
36  //  Checks if the input Ball hits the top/bottom edge of the brick or not
37  boolean verticalBounce(Ball b) {
38    float lbx = b.pos.x - b.speed.x;
39    float lby = b.pos.y - b.speed.y;
40    float alpha = 0;
41    float hitX;
42
43    alpha = ((pos.y - radius.y - b.radius) - lby) / b.speed.y;
44    hitX = lerp(lbx, b.pos.x, alpha);
45
46    if (abs(hitX - pos.x) >= (radius.x + b.radius)) {
47      return false;
48    }
49    return true;
50  }
```



Step 2 - Lonely Paddle (GameEngine)

LonelyPaddle | Ball | Brick | Float2 | GameEngine

```
class GameEngine {

  boolean gameover;
  float rTOP, rLEFT, rRIGHT, rBOTTOM;
  Ball ball;

  int score = 0;
  int scoreSpace = 50;
  Brick paddle;

  GameEngine(Float2 ULC, Float2 LRC, float bRadius) {
    rTOP    = ULC.y + bRadius;
    rLEFT   = ULC.x + bRadius;
    rRIGHT  = LRC.x - bRadius;
    rBOTTOM = LRC.y - bRadius;
    gameover = false;

    //  Create the bouncing ball
    Float2 bCenter = new Float2( (rLEFT + rRIGHT)*0.5, (rTOP + rBOTTOM)*0.5 + scoreSpace);
    Float2 bSpeed  = new Float2(random(1,4), random(-2,-4));
    ball = new Ball(bCenter, bSpeed, bRadius, color(255,0,0));

    //  Create the paddle
    Float2 pCenter = new Float2((rLEFT + rRIGHT)*0.5, rBOTTOM - 10);
    Float2 pRadius = new Float2(width/10, 8);
    paddle = new Brick(pCenter, pRadius, color(0,255,255), true);
  }
```

GameEngine

Class of our GameEngine object.

DATA update

score : game score.

scoreSpace : space for printing score.

paddle : our paddle

Step 2 - Lonely Paddle (GameEngine)

```
29  boolean paddleHit() {
30    if (ball.speed.y < 0) return false;
31    if (paddle.hit(ball)) {
32      if (paddle.verticalBounce(ball)) {
33        ball.bounceY(paddle.pos.y - paddle.radius.y - ball.radius);
34        return true;
35      }
36    }
37    return false;
38  }
```

GameEngine
Class of our GameEngine object.

METHODS update
`paddleHit()`: checks if the ball hits our paddle.

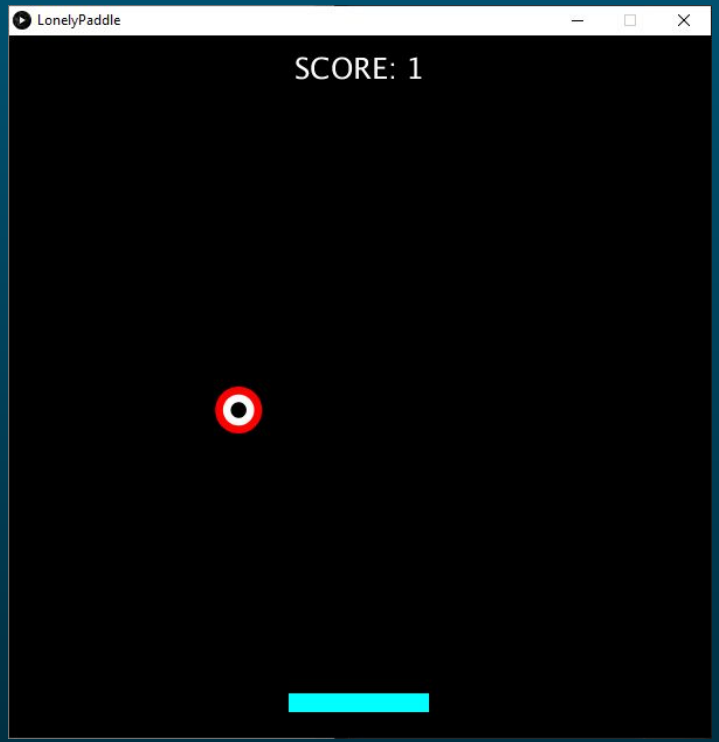
Step 2 - Lonely Paddle (GameEngine)

```
50  void checkHit() {
51    if (paddleHit()) {
52      score += 1;
53    }
54    if (!insideROOM()) gameover = true;
55  }
56
57  //  Update the Game State
58  void update() {
59    ball.update();
60    paddle.pos.x = mouseX;
61    checkHit();
62  }
63
64  void display() {
65    if (gameover) background(120,0,0);
66    else background(0);
67    fill(255);
68    text("SCORE: " + score, width/2, scoreSpace/2);
69    ball.display();
70    paddle.display();
71    if (gameover) {
72      fill(255);
73      text("GAME OVER", width/2, height/2);
74      noLoop();
75    }
76  }
```

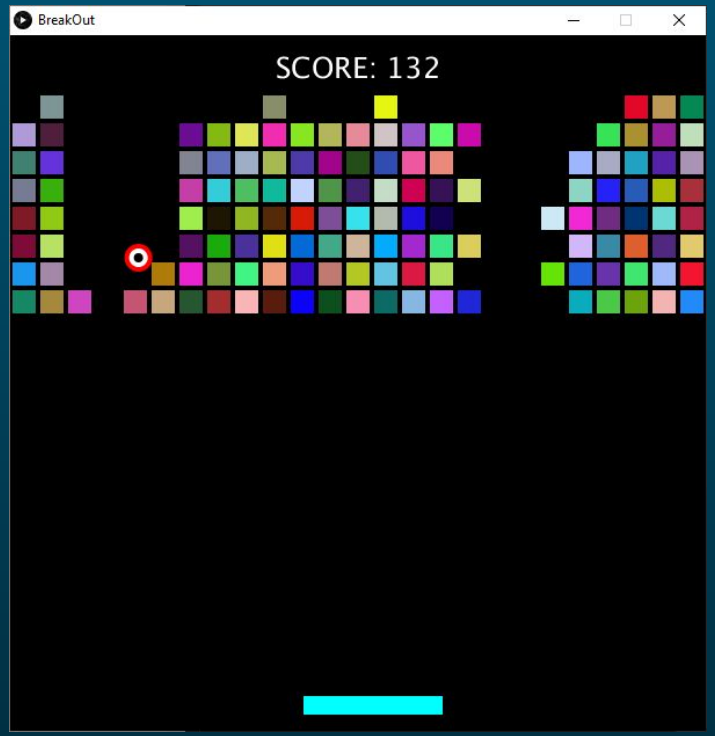
GameEngine
Class of our GameEngine object.

METHODS update
checkHit(): adds score if paddle hit.
update(): makes paddle move with mouse.
display(): adds paddle display.

Lonely Paddle Demo



Step 3 - Break Out



GOAL

Add lots of bricks to break.

CODE

Update the game engine to allow bricks creation and update the bricks state if hit.

- `GameEngine (Update)`

Step 3 - Break Out (GameEngine)

BreakOut | Ball | Brick | Float2 | GameEngine

```
1  class GameEngine {
2
3    boolean gameover;
4    float rTOP, rLEFT, rRIGHT, rBOTTOM;
5    Ball ball;
6
7    int score = 0;
8    int scoreSpace = 50;
9    Brick paddle;
10
11   int numBricks, activeBricks;
12   int bcol, brow;
13   Brick[] bricks;
```

GameEngine
Class of our GameEngine object.

DATA update
numBricks : total # of bricks.
activeBricks : # of active bricks.
bcol : # of columns of bricks
brow : # of rows of bricks
bricks[]: array of Bricks object.

Step 3 - Break Out (GameEngine)

```
32    //  Create the bricks
33    numBricks = bkc * bkr;
34    activeBricks = numBricks;
35    bricks = new Brick[bkc * bkr];
36
37    Float2 brickRadius, brickOffset;
38    brickRadius = new Float2( (width/bkc)/2 - bkGap/2, ((height/3)/bkr)/2 - bkGap/2 );
39    brickOffset = new Float2( (width/bkc)/2, ((height/3)/bkr)/2 );
40
41    for (int i = 0; i < numBricks; i++) {
42      int col = i % bkc;
43      int row = i / bkc;
44      Float2 brickCenter = new Float2( brickOffset.x + col * brickOffset.x * 2,
45        scoreSpace + brickOffset.y + row * brickOffset.y * 2);
46      color bc = color(round(random(255)),round(random(255)),round(random(255)));
47      bricks[i] = new Brick(brickCenter, brickRadius, bc, true);
48    }
```

`GameEngine`
Class of our GameEngine object.

Constructor update
Creates an array of instances of bricks as **`bricks[]`** according to **`bcol`** and **`brow`**.

Step 3 - Break Out (GameEngine)

```
53  boolean paddleHit() {
54    if (ball.speed.y < 0) return false;
55    if (paddle.hit(ball)) {
56      if (paddle.verticalBounce(ball)) {
57        ball.bounceY(paddle.pos.y - paddle.radius.y - ball.radius);
58        float speedup = (pmouseX - mouseX) * 0.2;
59        ball.speed.x -= speedup;
60        return true;
61      }
62    }
63    return false;
64  }
```

GameEngine
Class of our GameEngine object.

METHOD update
`paddleHit()` responds to the speed of the paddle and adjust the `ball.speed.x` accordingly.

Step 3 - Break Out (GameEngine)

```
67  boolean brickHit() {
68    for (int i = 0; i < numBricks; i++) {
69
70      if (bricks[i].on) {
71        if (bricks[i].hit(ball)) {
72          if (bricks[i].verticalBounce(ball)) {
73            if (ball.speed.y > 0) {    //  Bounce on TOP
74              ball.bounceY(bricks[i].pos.y - bricks[i].radius.y - ball.radius);
75            } else {
76              ball.bounceY(bricks[i].pos.y + bricks[i].radius.y + ball.radius);
77            }
78          }
79          else {
80            if (ball.speed.x > 0) {    //  Bounce on LEFT
81              ball.bounceX(bricks[i].pos.x - bricks[i].radius.x - ball.radius);
82            } else {                   //  Bounce on RIGHT
83              ball.bounceX(bricks[i].pos.x + bricks[i].radius.x + ball.radius);
84            }
85          }
86          bricks[i].on = false;  //  Hide the hit brick
87          activeBricks--;        //  Remove one brick
88          return true;           //  One brick hit, LEAVE !
89        }
90      }
91    }
92    return false;
93  }
```

GameEngine
Class of our GameEngine object.

METHOD update
`brickHit()` loops through the active `bricks[]` to check if any brick got hit using `bricks[].hit()`.
The state `.on` of the hit brick will be set to `false`.

Step 3 - Break Out (GameEngine)

```
106  void checkHit() {
107    if (paddleHit()) {
108      score += 1;
109    }
110    if (brickHit()) {
111      score += 2;
112    }
113    if (!insideROOM()) gameover = true;
114  }
115
116  //  Update the Game State
117  void update() {
118    if (activeBricks == 0) gameover = true;
119    ball.update();
120    paddle.pos.x = mouseX;
121    checkHit();
122  }
```

GameEngine
Class of our GameEngine object.

METHOD update
`checkHit()` calls `brickHit()`
`update()` checks the number of active bricks left.

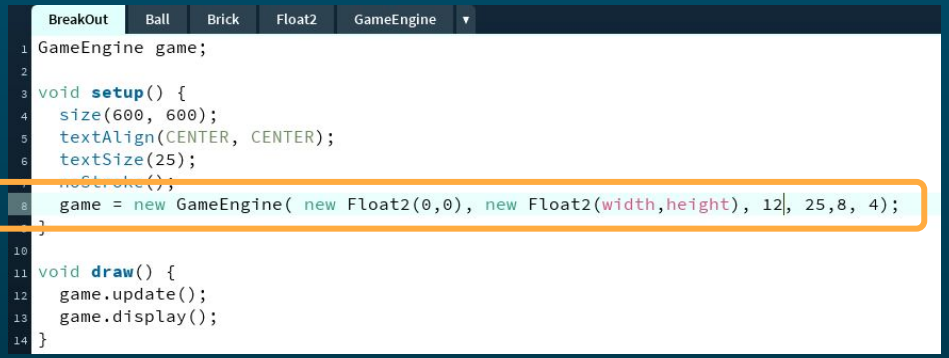
Step 3 - Break Out (GameEngine)

```
124   void display() {
125     if (gameover) background(120,0,0);
126     else background(0);
127     fill(255);
128     text("SCORE: " + score, width/2, scoreSpace/2);
129     for (int i = 0; i < numBricks; i++) {
130       if (bricks[i].on) bricks[i].display();
131     }
132     ball.display();
133     paddle.display();
134     if (gameover) {
135       fill(255);
136       text("GAME OVER", width/2, height/2);
137       noLoop();
138     }
139   }
```

GameEngine
Class of our GameEngine object.

METHOD update
`display()` calls the `bricks[].display()` method of the active bricks only.

Step 3 - Break Out (Main Sketch)



UPDATE

`GameEngine` creation accepts the bricks configuration parameters.

BreakOut Full Demo