P.PORTO

POLITÉCNICO
DO PORTO
ESMAD

COMPUTAÇÃO GRÁFICA
TSIW

Syllabus

- Animation using Objects
- Particle Systems
- Collisions

Objects and Animation

- For Canvas animations, it is usually required to define variables to control the animation flow (e.g. object color, position, size,…)
- How to set/control **multiple** properties of **multiple** objects?
- **Option A**: define all required variables for all existing objects

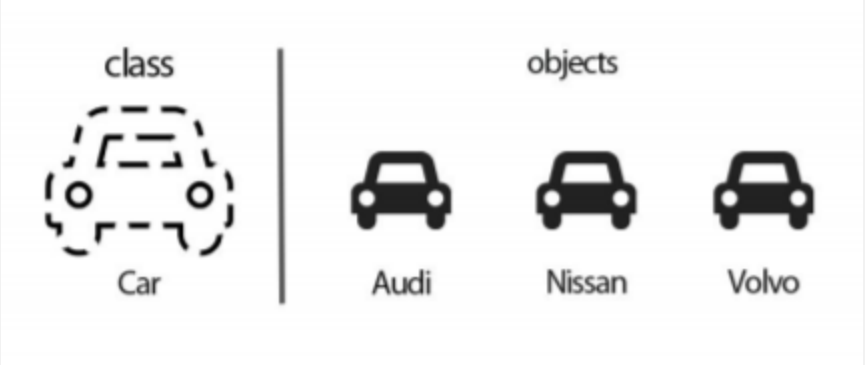
- **Option B**: define animated objects as JavaScript Classes

Objects and Animation

- JavaScript literal objects are useful when a single specific object is required

```
const car = {
    color: "red",
    changeColor: function (c) {
        this.color = c;
    }
};
```

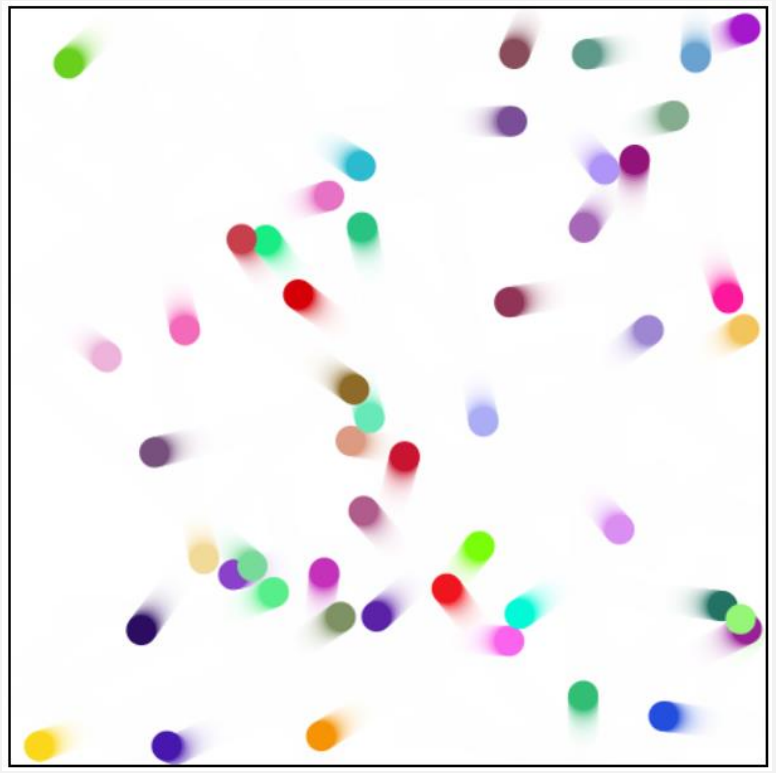
- However, sometimes it is required to have similar objects
- **Classes**: abstract description of a set of similar objects, with properties (data) and methods (procedures) that describe the content and behaviour of its instances



Objects and Animation

EXAMPLE – Bouncing Balls

- Let's start by write the classic example of multiple balls bouncing around a Canvas



1. **Class constructor**: set ball most important properties
2. **Draw method**: necessary to draw the objects
3. Instantiate multiple objects of the class and draw them in Canvas
4. **Update method**: detect collisions with Canvas and update all objects

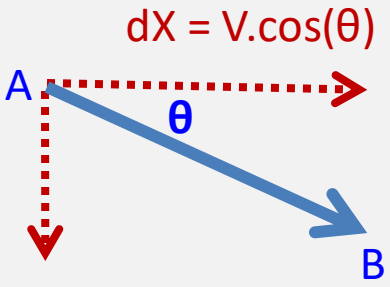
Objects and Animation

EXAMPLE – Bouncing Balls

Set the Class constructor:

- think of the most relevant object properties, necessary to draw and animate it

```
class Ball {
        constructor(x, y, r, d, c) {  // CONSTRUCTOR
            this.x = x;                          // initial X position
            this.y = y;                          // initial Y position
            // (constant) horizontal displacement (velocity): d is a direction angle
            this.dX = 2 * Math.cos(d);
            // (constant) vertical displacement (velocity): d is a direction angle
            this.dY = 2 * Math.sin(d);
            this.c = c;                          // color
            this.R = r;                          // circle radius (constant)
        }
}
```



Displacement = going from A to B

Objects and Animation

EXAMPLE – Bouncing Balls

Instantiate multiple objects (50)

```
let b = new Array(); // setup as many balls as wanted
for (let i = 0; i < 50; i++) {
        let R = Math.floor(Math.random() * 256);
        let G = Math.floor(Math.random() * 256);
        let B = Math.floor(Math.random() * 256);
        let color = `rgb(${R},${G},${B})`; // random color
        // random position (inside Canvas)
        let xInit = 20 + Math.random() * (W - 2 * 20);
        let yInit = 20 + Math.random() * (H - 2 * 20);
        // random direction
        let direction = Math.random() * 2 * Math.PI;

        b.push(new Ball(xInit, yInit, 10, direction, color))
}
```

Objects and Animation

EXAMPLE – Bouncing Balls

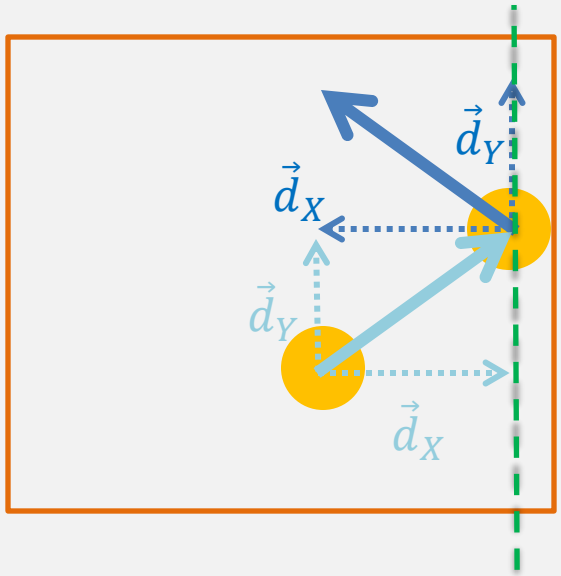
Implement a drawing method

```
class Ball {
        constructor(x, y, d, c) { … }

        draw() {
           ctx.fillStyle = this.c;
           ctx.beginPath();
           ctx.arc(this.x, this.y, this.R, 0, 2 * Math.PI);
           ctx.fill();
        }

}
```

Objects and Animation

EXAMPLE – Bouncing Balls

Implement update method(s)



maxX = Wcanvas - Rball

```
class Ball {
        constructor(x, y, d, c) { … }
        draw() { … }

        update() {
                // check Canvas vertical collisions
                if (this.x < this.R || this.x > W - this.R)
                        this.dX = -this.dX;
                // check Canvas horizontal collisions
                if (this.y < this.R || this.y > H - this.R )
                        this.dY = -this.dY;
                this.x += this.dX;    // update horizontal position
                this.y += this.dY;    // update vertical position
        }
}
```

Objects and Animation

EXAMPLE – Bouncing Balls

Draw and update all objects

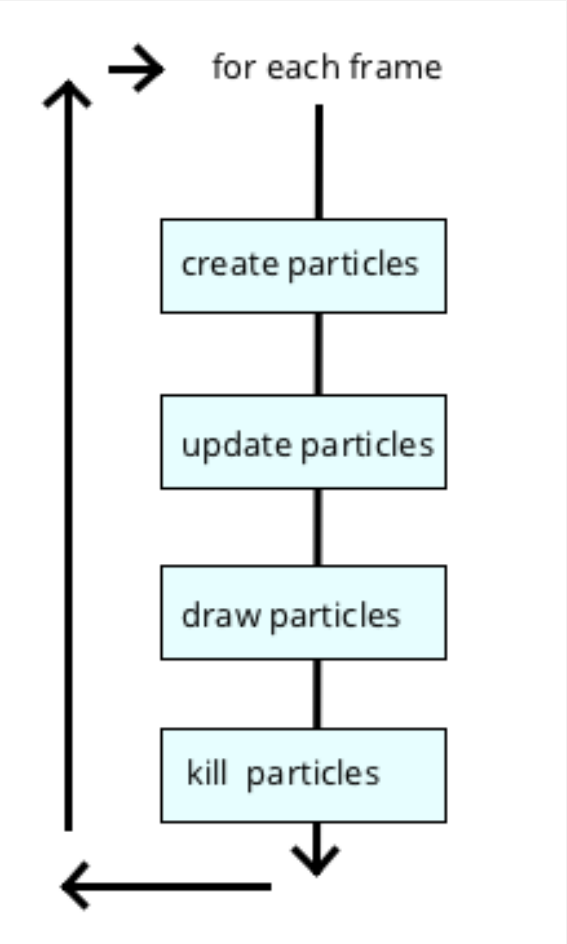
```
function render() {
        // fade Canvas
        ctx.fillStyle = "rgba(255,255,255,0.25)"
        ctx.fillRect(0, 0, W, H);

        // draw & update
        b.forEach( ball => {
          ball.draw();
          ball.update();
        });

        //new frame
        window.requestAnimationFrame(render);
    }
render(); //start the animation
```

Particle Systems

- **Particles** are very popular in animation
- They are a group of objects and set of rules that define how they move and interact with each other:
  - No interaction (http://spielzeugz.de/html5/liquid-particles/)
  - Interaction by collision (http://monochromacy.net/Resources/collision.html)
  - Short-range interaction (http://monochromacy.net/Resources/collision.html - when user clicks)
  - Long-range interaction (https://threejs.org/examples/webgl_animation_cloth.html)
- For no interaction particle systems, you do not require to know nothing more than simple programming and physics

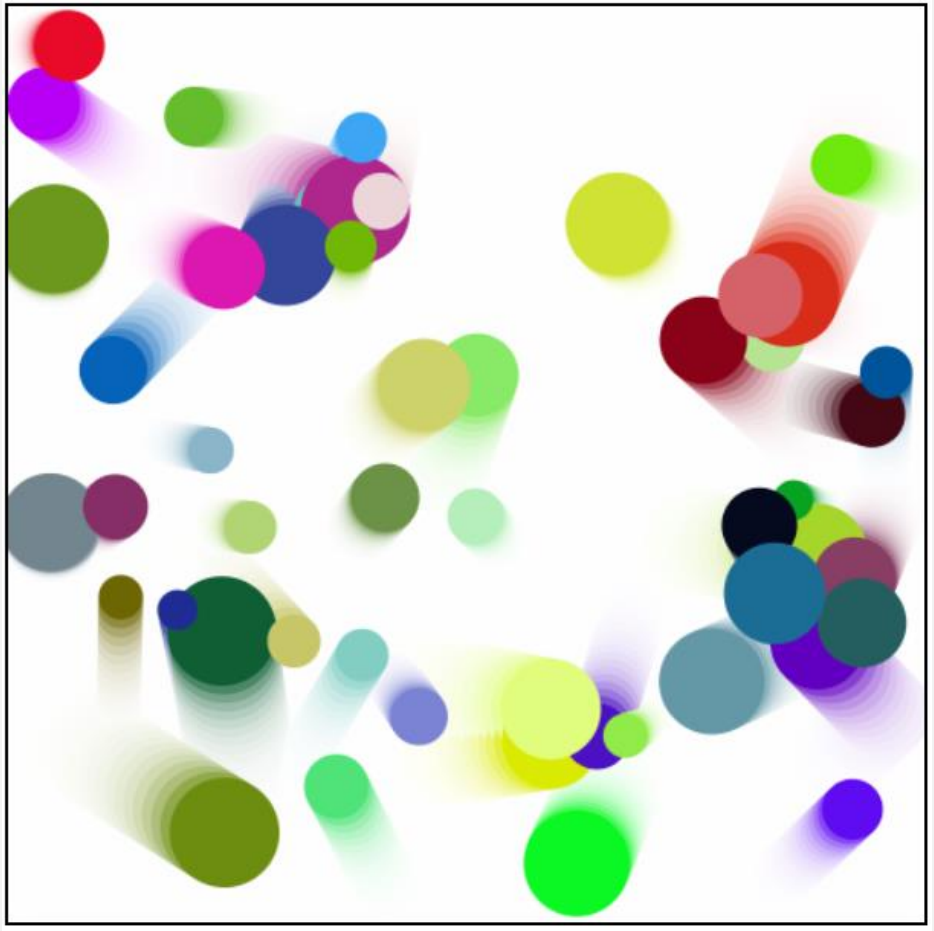
Particle Systems

- A classical particle algorithm consists on:
  - Particles are stored on an **Array** (therefore, easy to **create** and **destroy**, or **recicle**)
  - http://www.w3schools.com/js/js_array_methods.asp
  - Depending on the animation and particle type of interaction, each particle must be created as an **Object** - instance of a **Class**, where all its properties and movement characteristics are defined



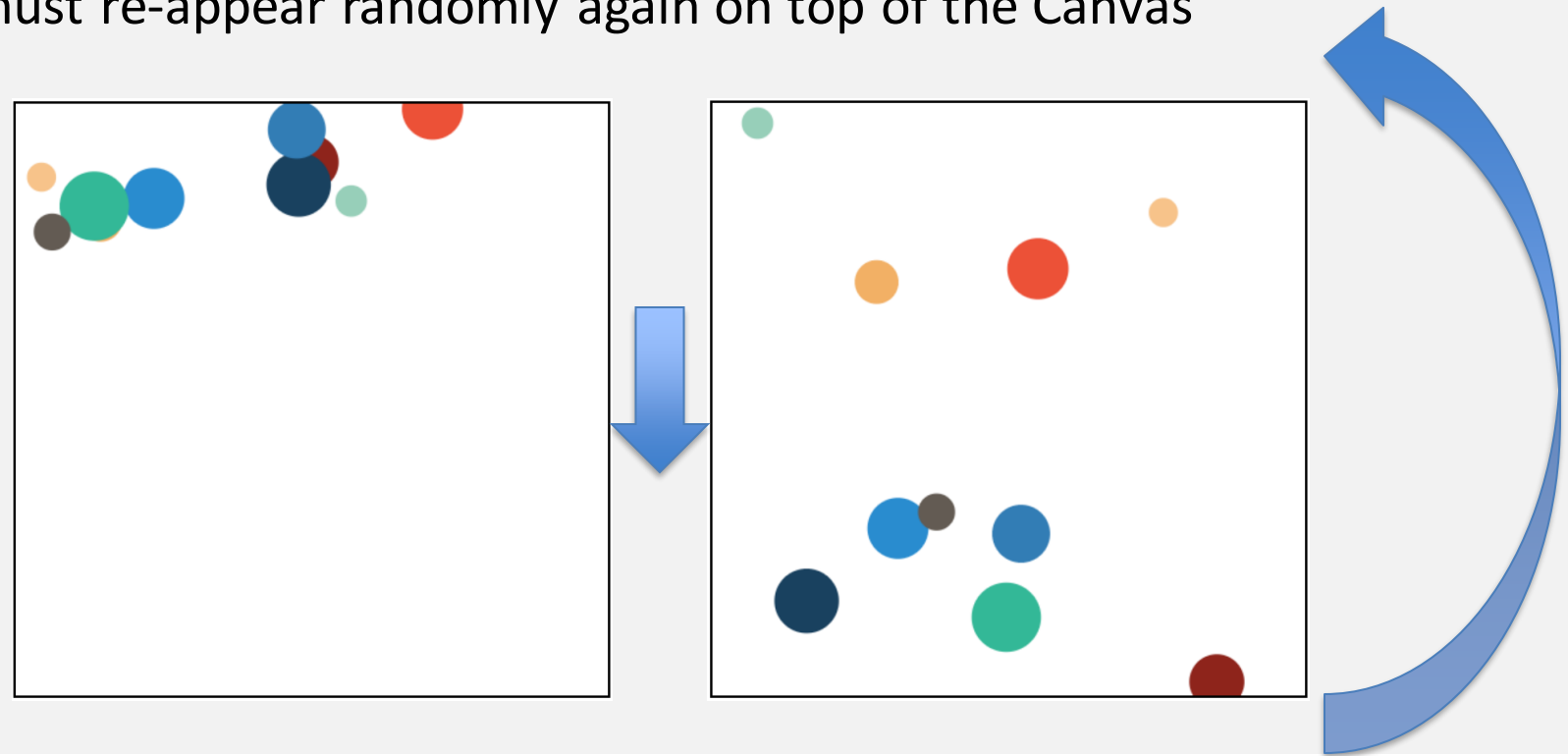
Try yourself…

1. Make the necessary alterations on the Bouncing Balls example, so that circles may have random sizes and random velocities.



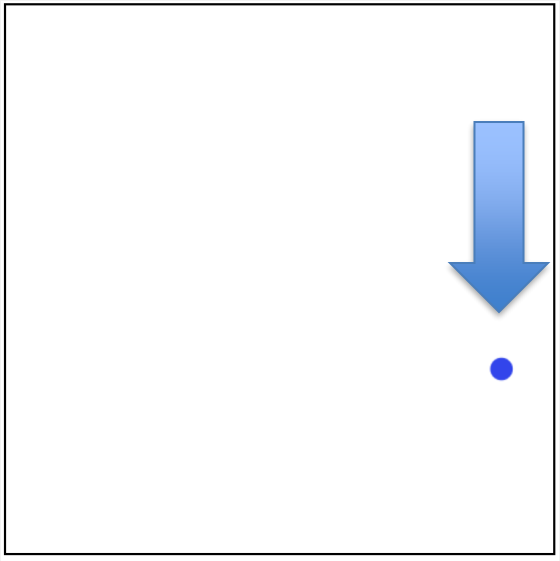
Try yourself…

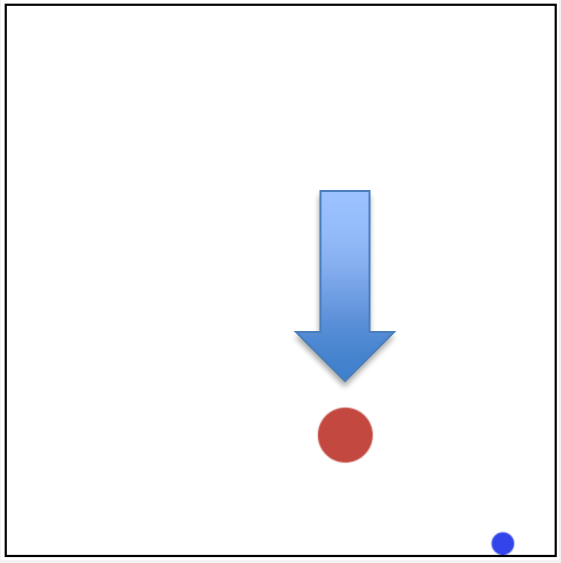
2. Start an animation with only 10 balls. All of them start above the Canvas (not visible) and fall vertically (at random velocities). When a ball disappear in the bottom of the Canvas, on the next frame it must re-appear randomly again on top of the Canvas

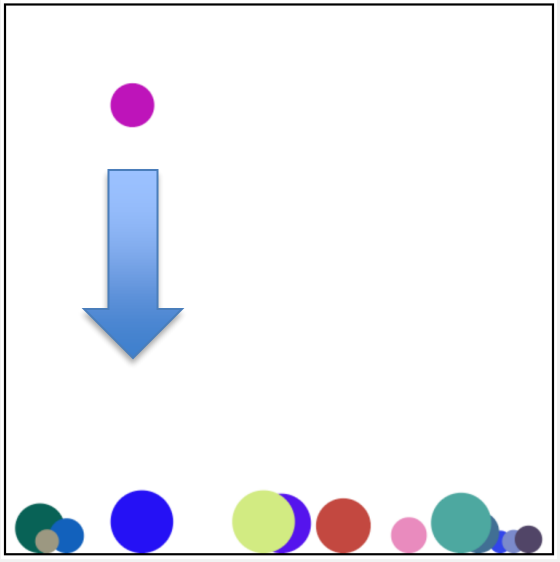


Try yourself…

3. Alter previous animate, so that it starts with only 1 ball. When the ball hits the bottom of the Canvas, it remains still in that position and a new ball is created and animated.

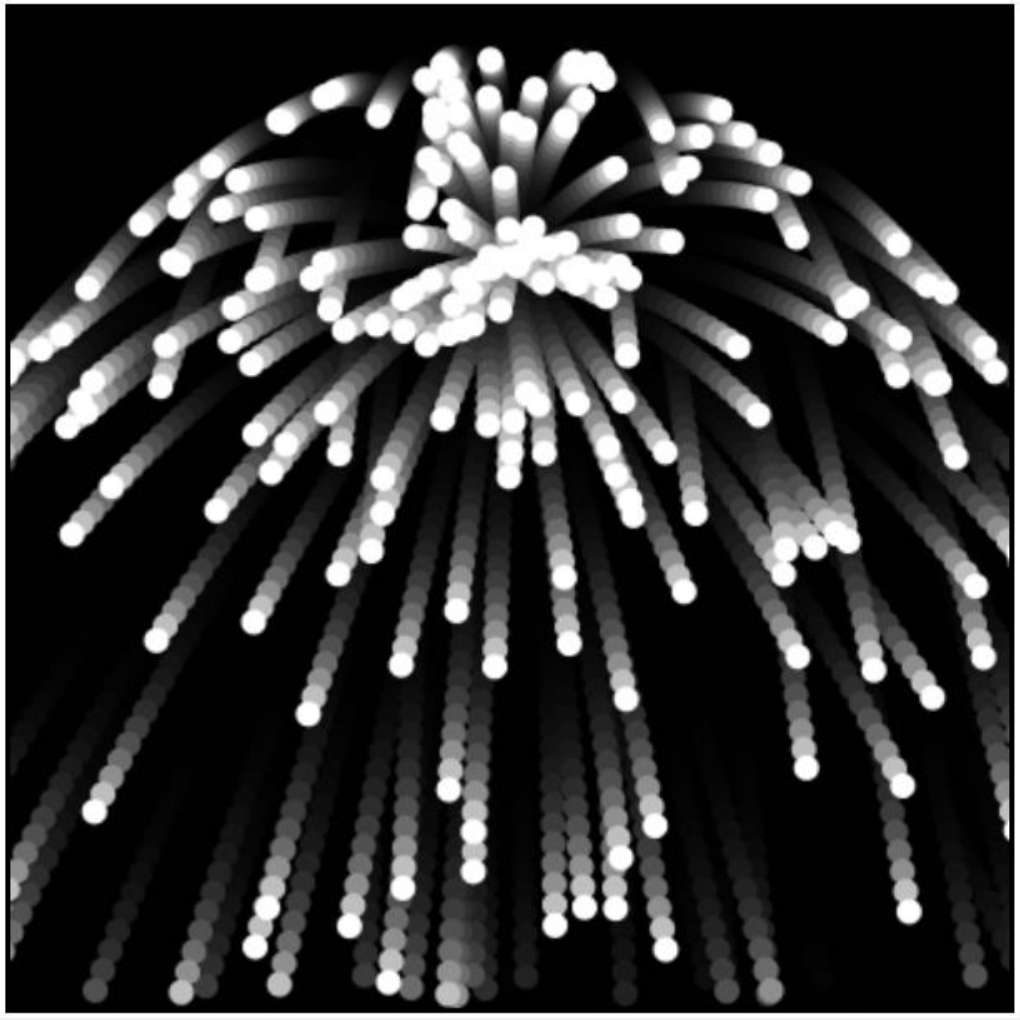






Try yourself…

4. Create a **particle emissor**. On each frame, at the source point, 3 new particles are created. All particles have a projectile motion. When a particle exits the Canvas, **it must be destroyed**.



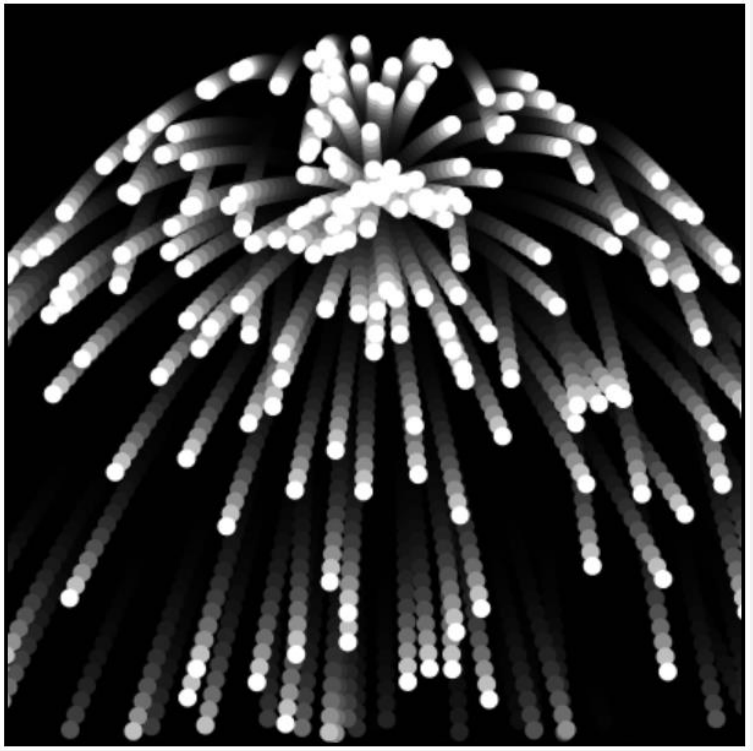
Try yourself…

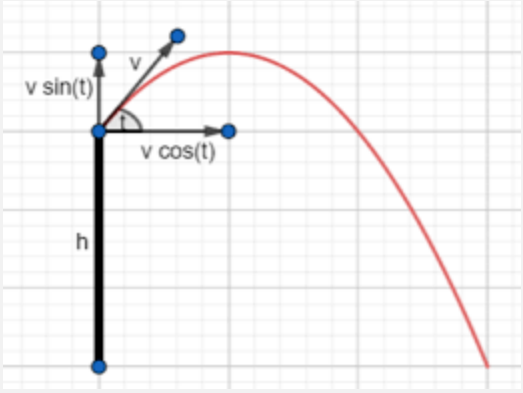
4. Create a **particle emissor**.
   Projectile motion: uniform in the X direction and accelerated by gravity in the Y direction.

   Source point: (W/2, H/4)
   Particle radius: 5 pixels

   Initial velocity:
   direction (θ) = random between [0, 2*π]
   velocity = 4 pixels per frame
   vX = v.cos(θ)
   vY = v.sin(θ)

   Gravity: 0.1 (value added to the vY velocity, on every frame)

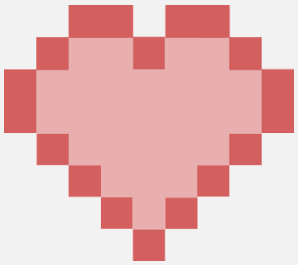


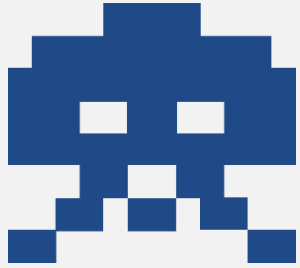


Collisions

- In simple terms, a collision occurs when two or more objects touch each other

- Since, in digital animation, since all we have is an object representation, it is required to use some techniques to detect collisions between object graphics
  - Analytical geometry
  - Temporal coherence





Collisions

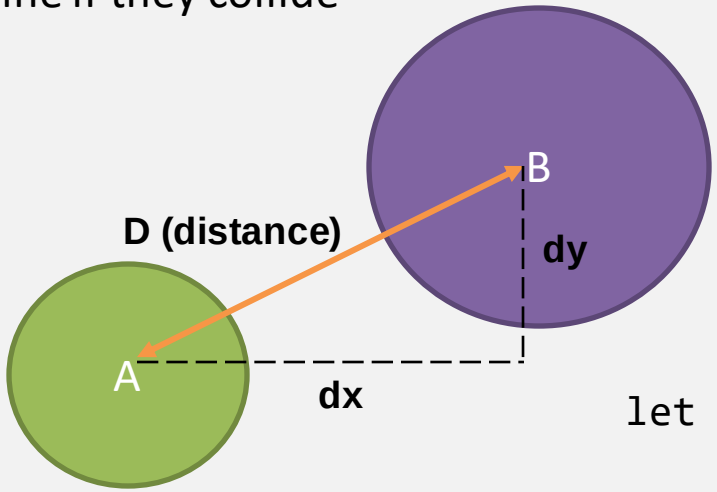
- Some of the most used methods to detect collisions in digital animation are:
  - *Bounding Polygons* (detects collision between two simple shapes)
  - *Ray casting* (detects collision using projection of rays)
  - *Separation Axis Theorem* (detects collision between two convex polygons)
- There is no better or worst method; choice must relay on simplicity versus realism

Collisions

BOUNDING POLYGONS: define **simple polygons** that bounds your objects, and check if they intersect

A. CIRCLES

- Having two objects bounded by circles, the distance to their centres define if they collide

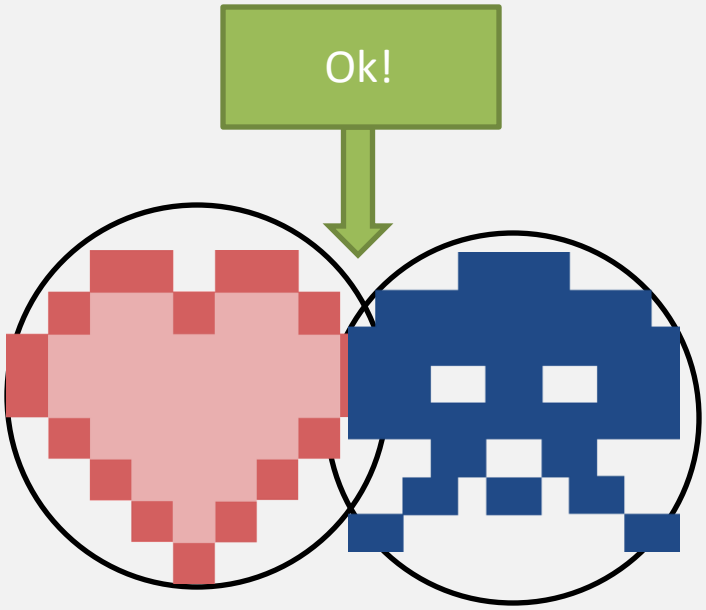



```
let dx = A.x - B.x;
let dy = A.y - B.y;
let D = Math.sqrt( dx*dx + dy*dy );
```

- Visualize it @ https://kishimotostudios.com/articles/circle_collision/

Collisions

A. CIRCLES



Collisions

2 CIRCLES or RECTANGLES

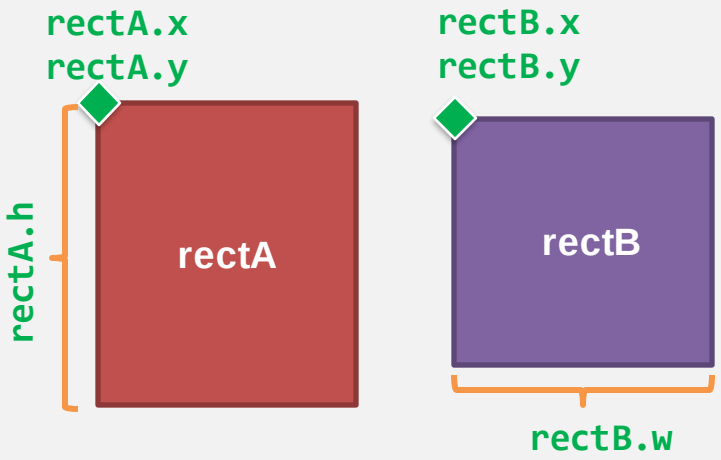
Good

Even better :)

Collisions

B. RECTANGLES

- Having two objects bounded by rectangles, you must check if one rectangle is completelly above, to the left, to the right or below, to ensure they do not collide

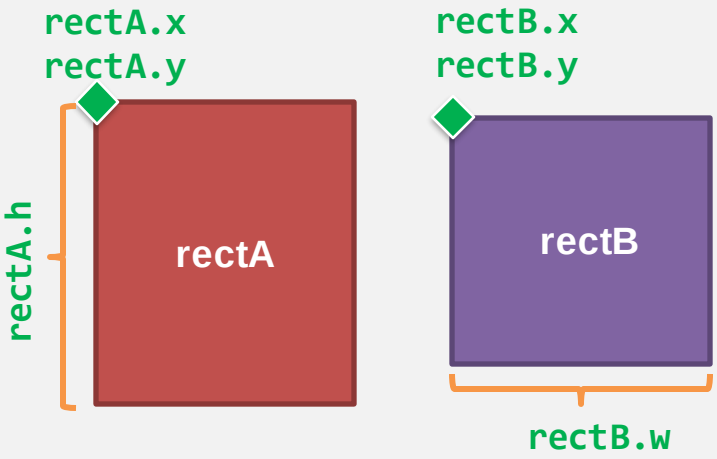



```
if (rectA.x + rectA.w < rectB.x
     //totally to the left: no collision
|| rectA.x > rectB.x + rectB.w
     //totally to the right: no collision
|| rectA.y + rectA.h < rectB.y
     //totally above: no collision
|| rectA.y > rectB.y + rectB.h) {
     //totally below: no collision
} else {
     /* they collide! */
}
```

Collisions

B. RECTANGLES

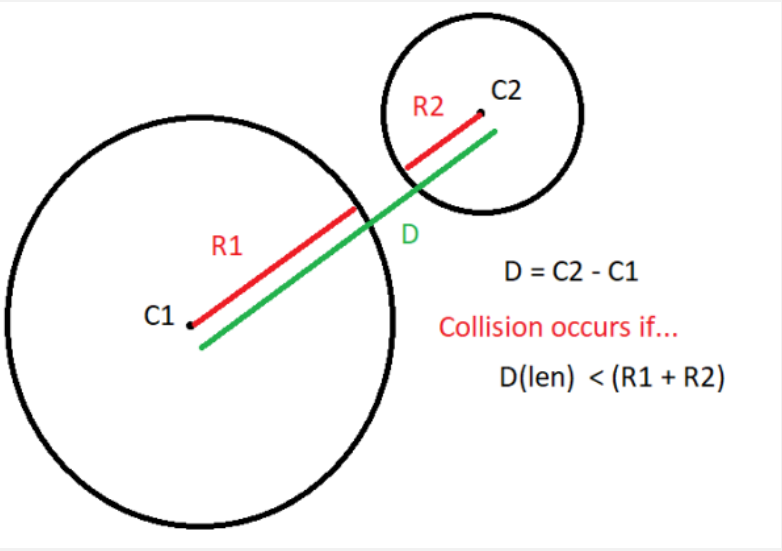
- Having two objects bounded by rectangles, you must check if one rectangle is completelly above, to the left, to the right or below, to ensure they do not collide

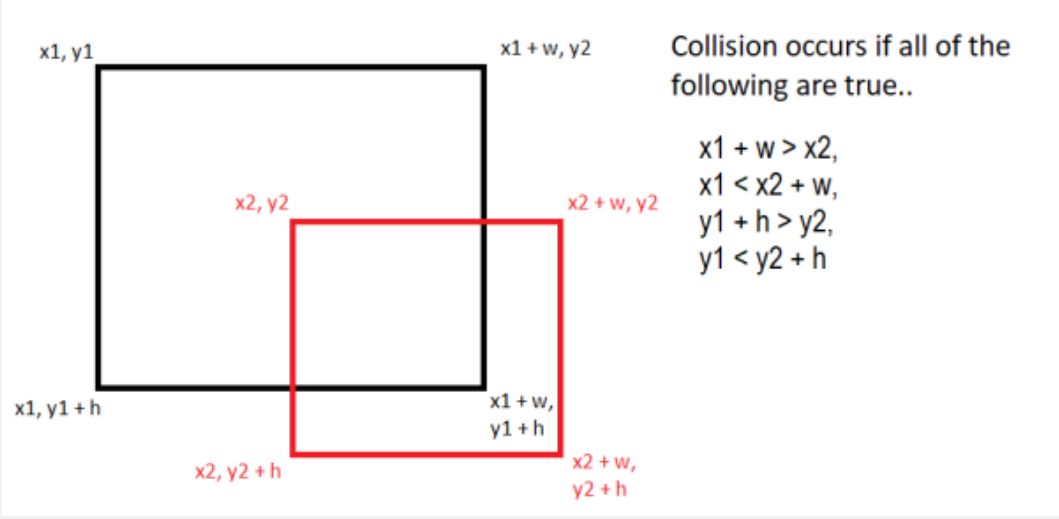



```
if (rectA.x + rectA.w >= rectB.x
     //NOT to the left
&& rectA.x <= rectB.x + rectB.w
     //NOT to the right
&& rectA.y+ rectA.h >= rectB.y
     //NOT above
&& rectA.y <= rectB.y + rectB.h) {
     //NOT below
     /* they collide! */
}
```

- Visualize it @ https://kishimotostudios.com/articles/aabb_collision/

Collisions


R2
C2
R1
D
C1
D = C2 - C1
Collision occurs if...
D(len) < (R1 + R2)


x1, y1
x1 + w, y2
x2, y2
x2 + w, y2
x1, y1 + h
x1 + w,
y1 + h
x2, y2 + h
x2 + w,
y2 + h
Collision occurs if all of the following are true..
x1 + w > x2,
x1 < x2 + w,
y1 + h > y2,
y1 < y2 + h

Collisions

MULTIPLE OBJECTS: how to handle collision detection for more than two objects?

Cenario 1: collisions between the a set of the same objects

```
//for all objects in the object array
for ( let i = 0; i < objectArray.length; i++)
{   //compare with all the REMAINING objects
    for ( let j = i + 1; j < objectArray.length; j++)
        //check if two objects collide
        checkCollision(objectArray[i], objectArray[j]);
}
```

checkCollision: function that returns a boolean if tow objects collide using one collision method of your choice (***bounding polygon***, or other…)

Collisions

MULTIPLE OBJECTS: how to handle collision detection for more than two objects?

Cenario 2: collisions between one object and a set of other objects

```
//for all objects in the object array
for ( let i = 0; i < objectArray.length; i++)
{
     //check if two objects collide
     checkCollision(objectArray[i], someOtherObject);
}
```

using a simple **for** statement

using the **forEach** array method

```
//for all objects in the object array
objectArray.forEach ( obj1 => {
    //check if it collides with other object
    checkCollision(obj1, someOtherObject);
});
```

Collisions

MULTIPLE OBJECTS: how to handle collision detection for more than two objects?

Cenario 3: collisions between two sets of objects

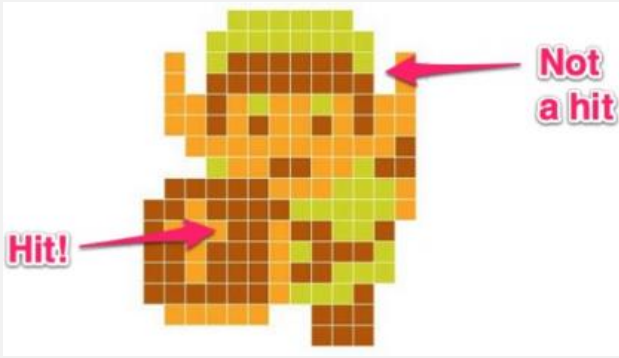
```
//for all objects in the 1st object array
for ( let i = 0; i < obj1Array.length; i++)
{   //compare with all objects of the 2nd object array
    for ( let j = 0; j < obj2Array.length; j++)
        //check if two objects collide
        checkCollision(obj1Array[i], obj2Array[j]);
}
```

using a simple **for** statement

using the **forEach** array method

```
//for all objects in the 1st object array
obj1Array.forEach ( obj1 => {
    //for all objects in the 2nd object array
    obj2Array.forEach ( obj2 => {
        //check if two objects collide
        checkCollision(obj1, obj2);
    });
});
```

Collisions



USER SELECTION: one can also call a collision to the user interaction with an object, e.g., when he clicks one with the mouse

- How to detect? Get mouse cursor coordinates `(x,y)` (or other input device) and check if:

1. COLOR
   - Get the color of the selected pixel

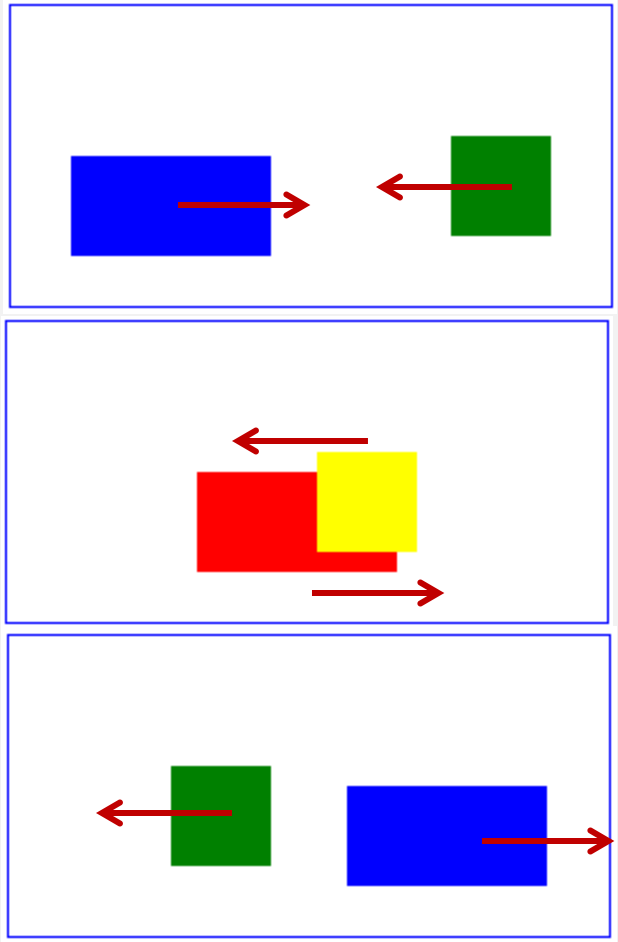
```
let pixel = ctx.getImageData(x,y,1,1)
pixel.data[0]//R  pixel.data[1]//G  pixel.data[2]//B
```

   - Compare it with the background or object colors

2. POSITION
   - Compare the pixel position relatively to the object position and dimensions

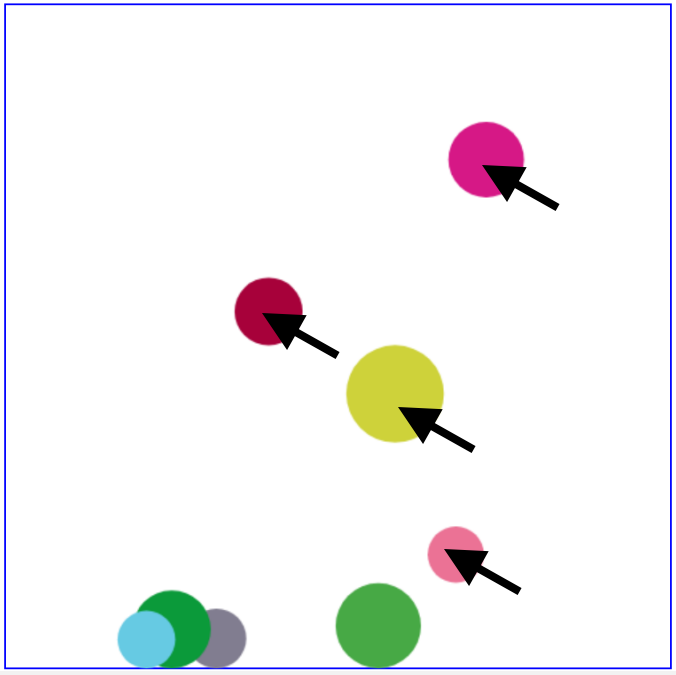
Try yourself…

1. Use bounding rectangles to alter the color of two rectangles when in collision with each other
   - Start with a simple animation, with the 2 rectangles moving horizontally, in opposite ways, 1 pixel per frame
   - Add bouncing with the Canvas boundaries
   - Finally, make the rectangles move in random directions



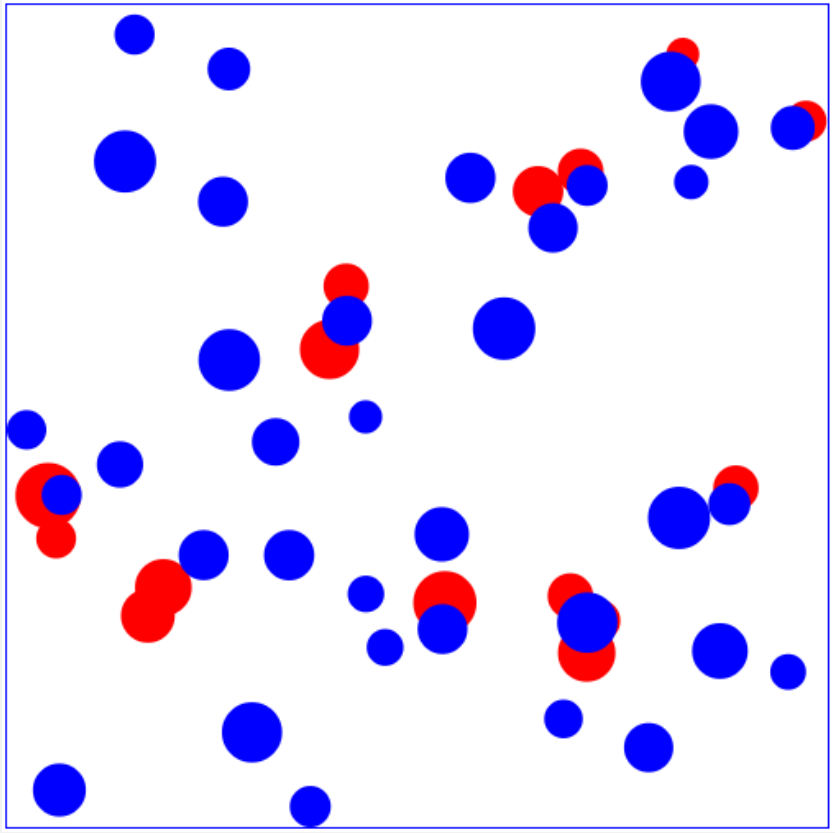
Try yourself…

2. Remember the Exercise 3 – one ball falling per frame? Alter it so that if a user clicks on a falling ball, it stops on the current position, and a new falling ball is created.



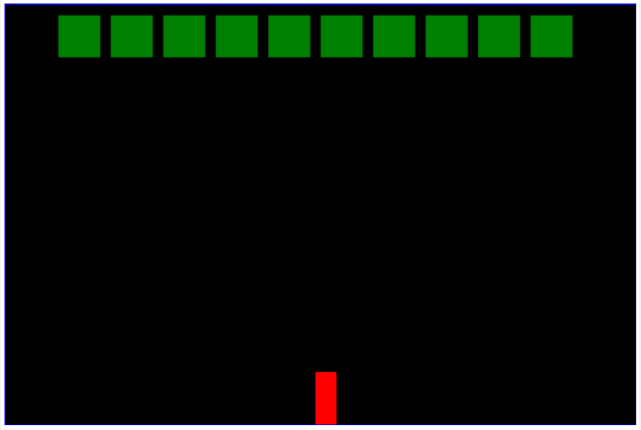
Try yourself…

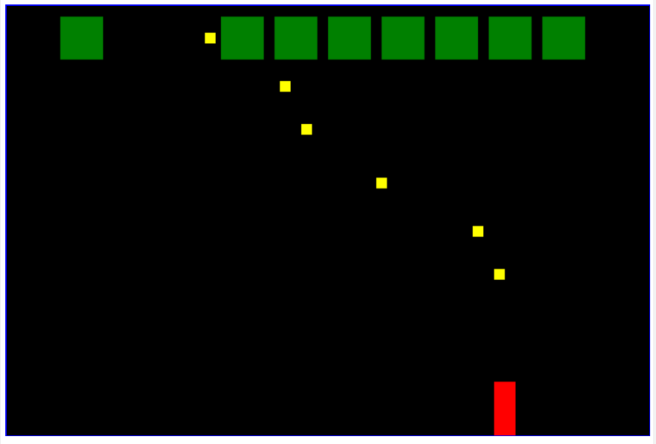
3. Remember the Bouncing Balls example? Alter it so that all balls start with the same blue color. When two balls collide, they change its color to red (both or just one of them)



Try yourself…

4. Make a simple Space Invaders game:
   - PLAYER (red): moves horizontally with the arrow keys
   - ENEMIES (green): static objects
   - BULLETS (yellow): a new bullet is created any time user presses the space bar
   - When a bullet hits an enemy, both are destroyed





Try yourself…

4. Drag & Drop:
   - Draw two shapes inside a Canvas: 1 blue circle and 1 red rectangle
   - Use the mouse events (mousedown, mousemove and mouseup) to move the shapes inside the Canvas:
     - mousedown: checks if the mouse pointer is inside a shape and signals that dragging is about to start if yes (stores the initial starting point – mouse cursor position – and the moving shape)
     - mousemove: reads the current mouse cursor position and calculates how far has the mouse moved to set the new moving shape position accordingly; stores a new starting point
     - mouseup:signals that dragging process has stopped