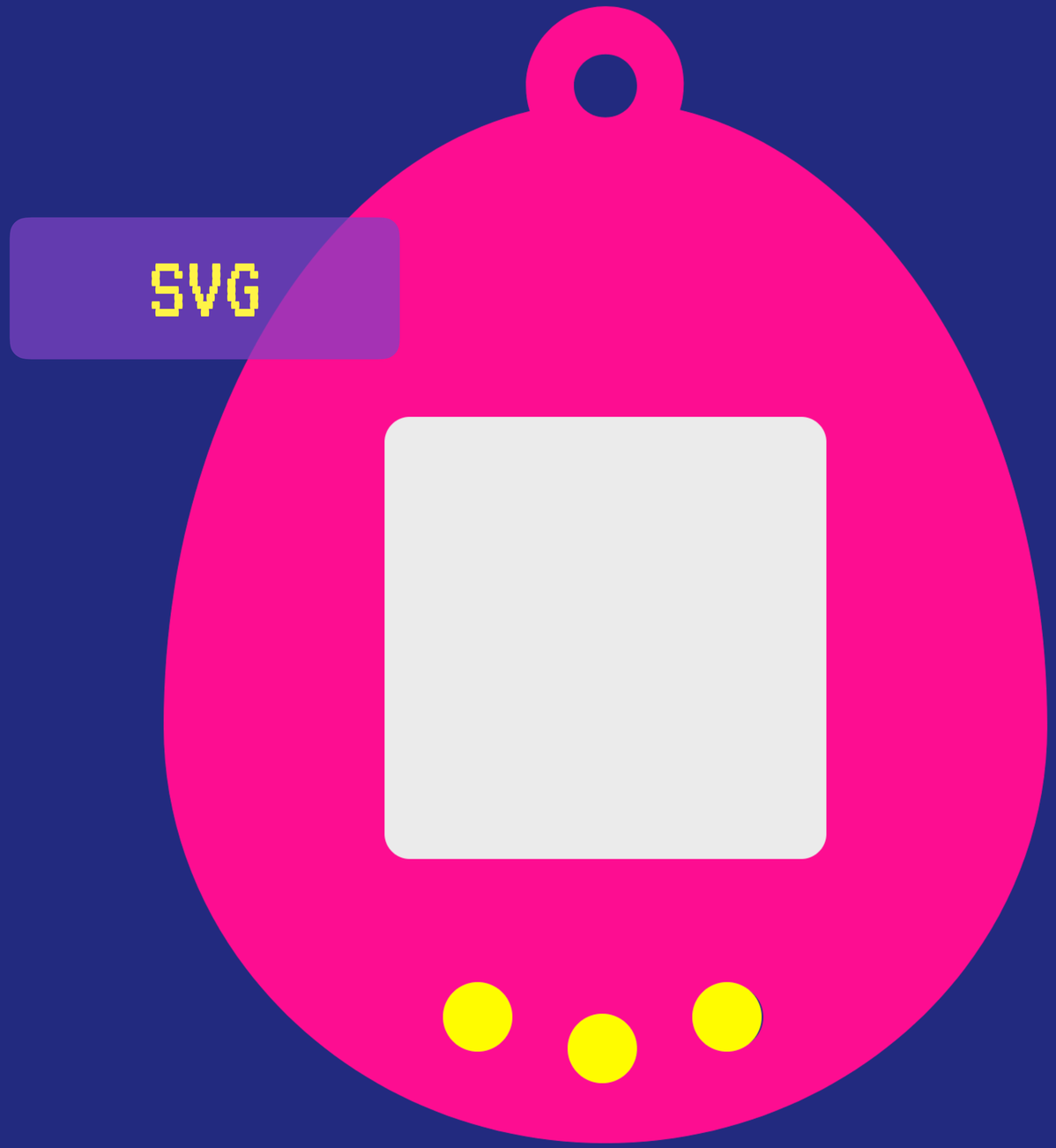
What Tamagotchis can teach you about ES6 generators

Hi, I'm Jenn.

Frontend Engineer

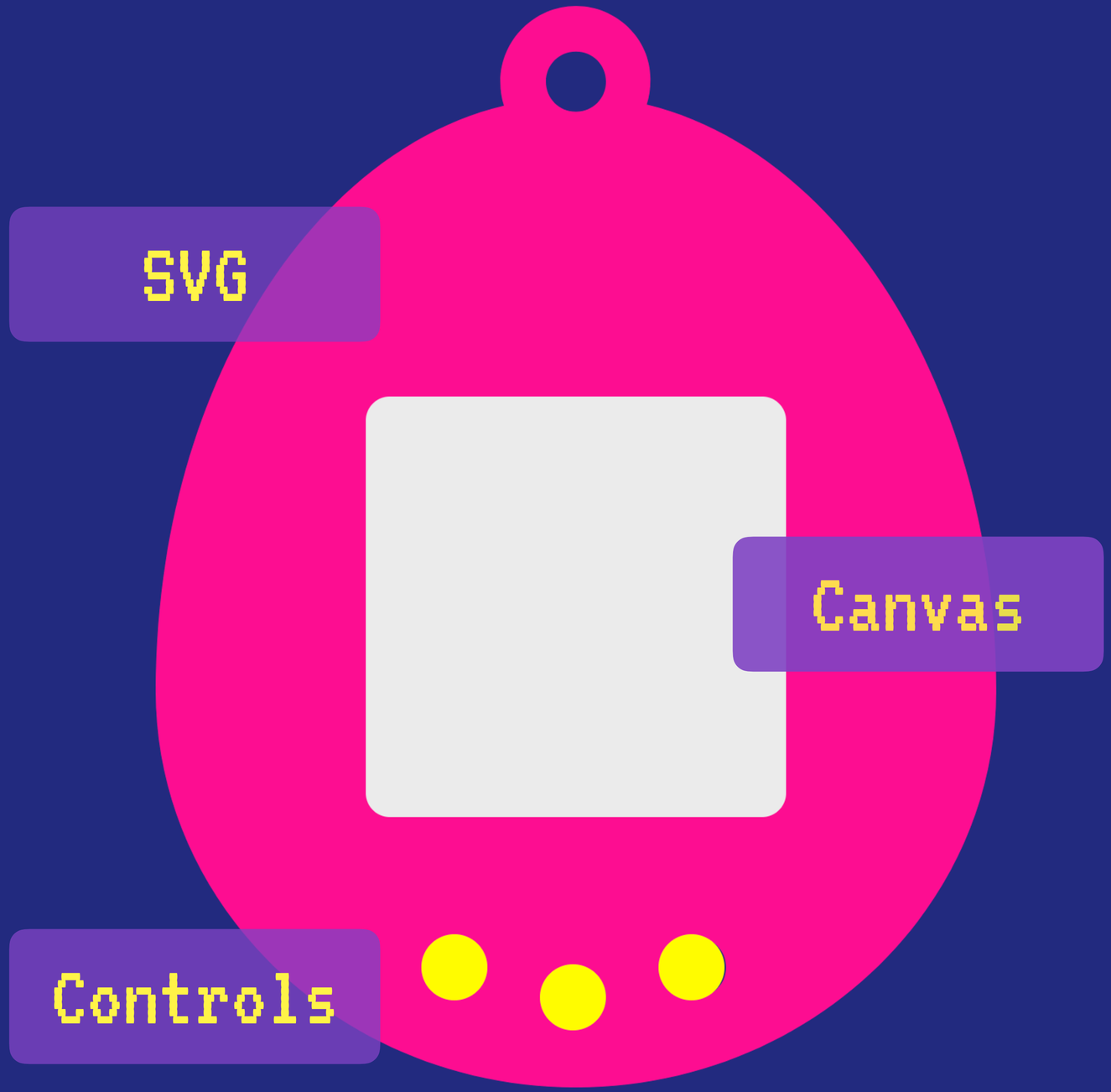
@gurlcode


@gurlcode



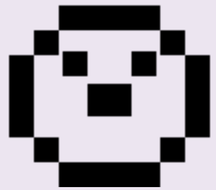
@gurlcode

@gurlcode



Canvas

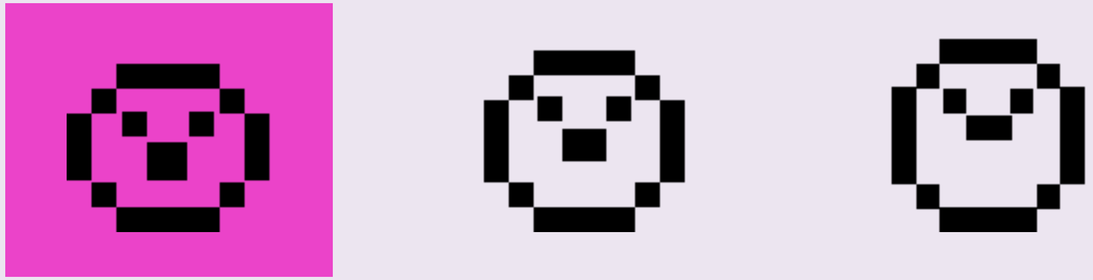
```
var canvas = document.querySelector(‘canvas’);
var context = canvas.context(‘2d’);
context.drawImage(‘tamagotchi.png’, …);
```



@gurlcode

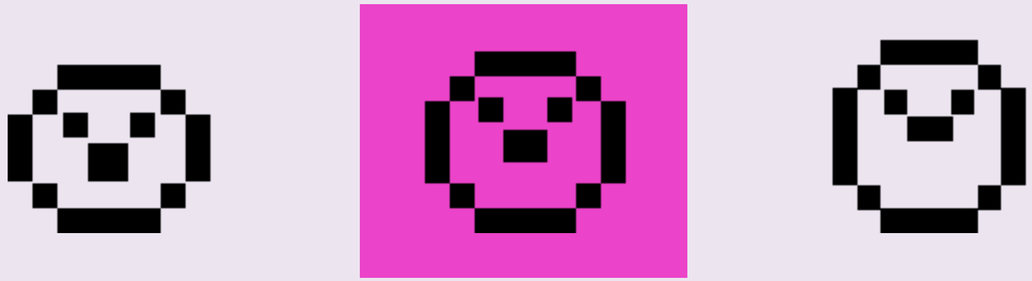
Animations

```
context.drawImage(image, 0, …coordinates);
context.clear();
context.drawImage(image, 200, …coordinates);
context.clear();
context.drawImage(image, 400, …coordinates);
```



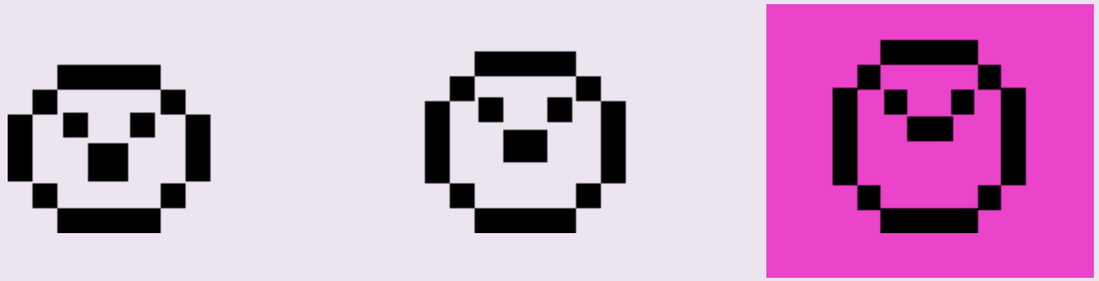
Animations

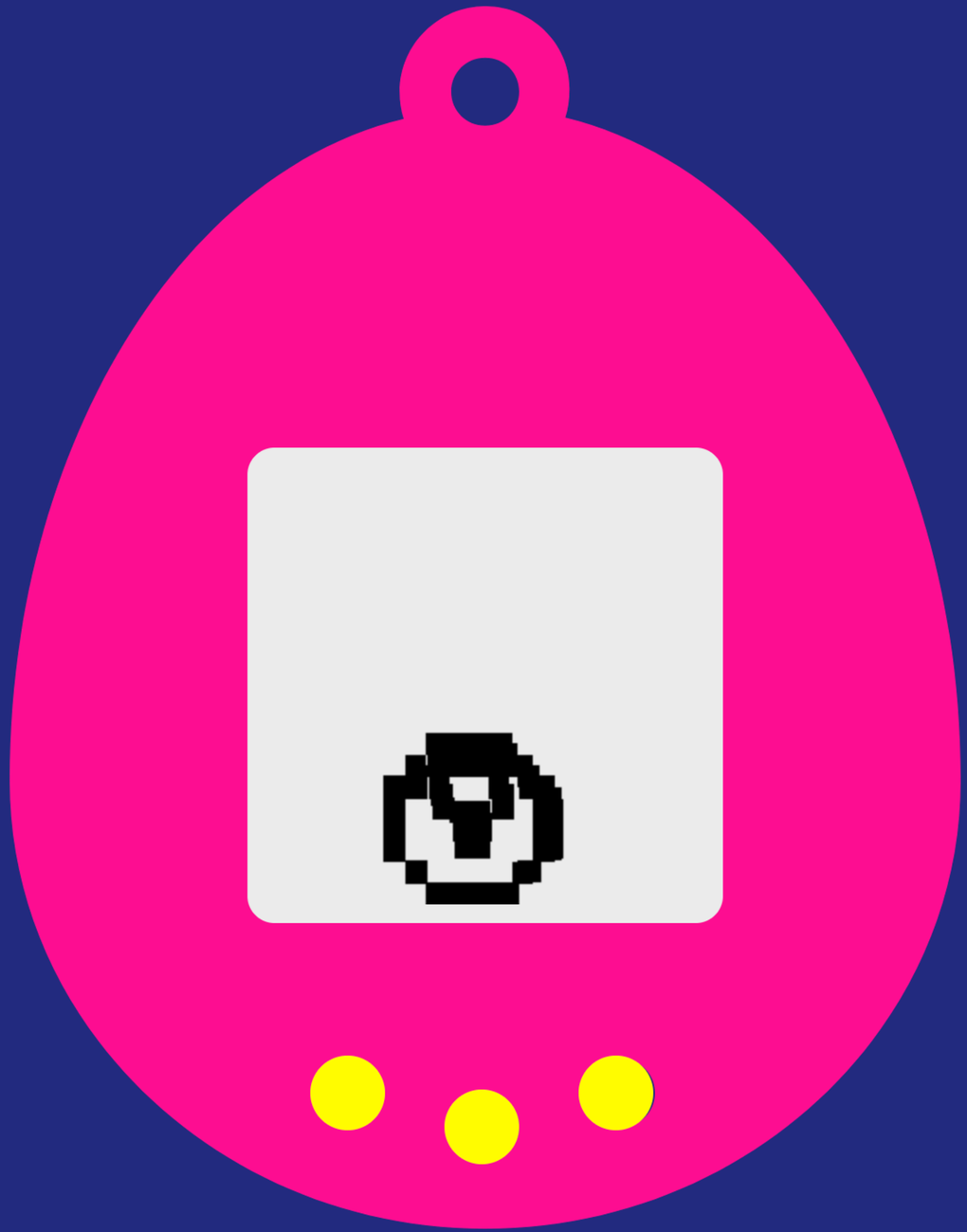
```
context.drawImage(image, 0, …coordinates);
context.clear();
context.drawImage(image, 200, …coordinates);
context.clear();
context.drawImage(image, 400, …coordinates);
```

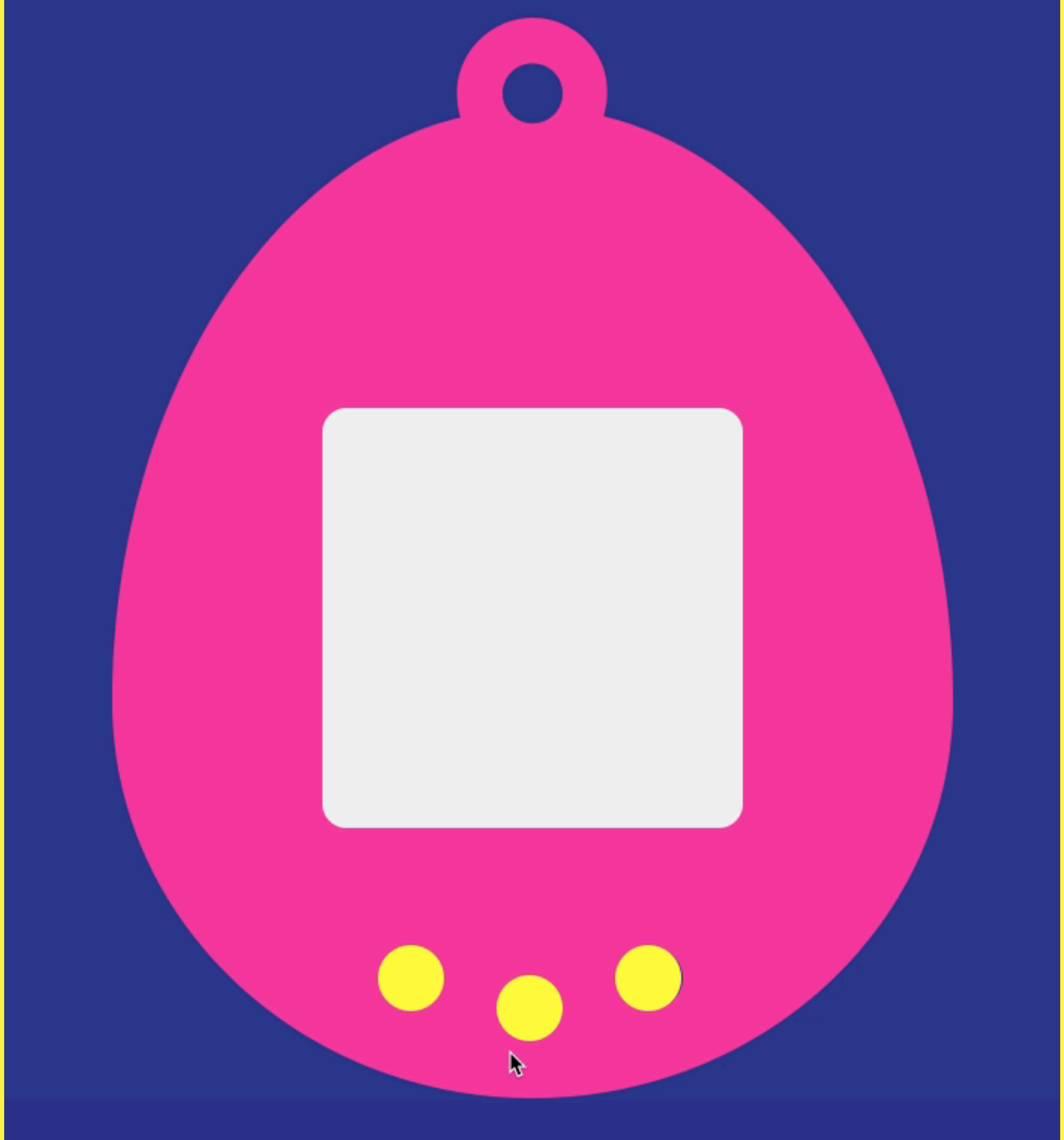


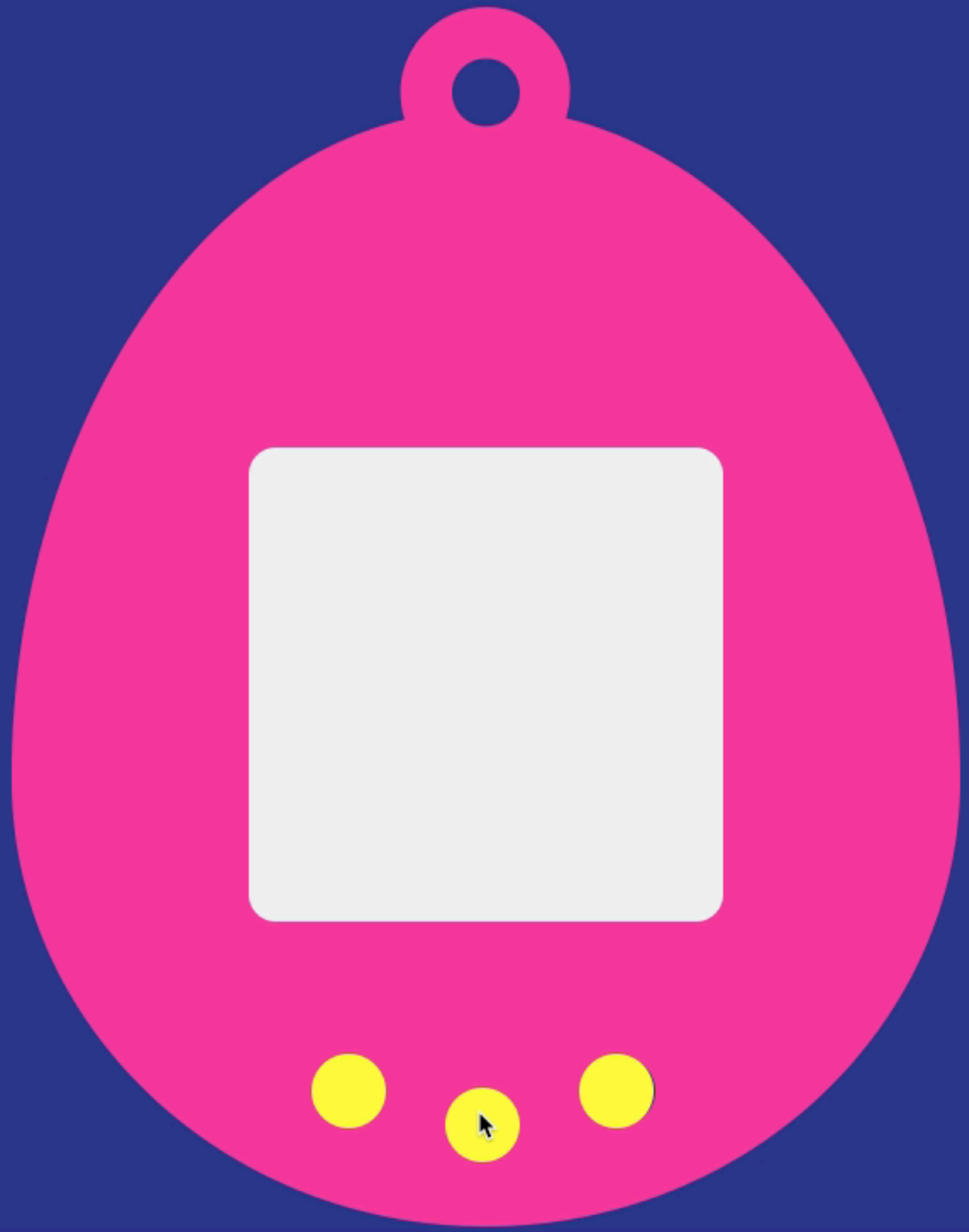
Animations

```
context.drawImage(image, 0, …coordinates);
context.clear();
context.drawImage(image, 200, …coordinates);
context.clear();
context.drawImage(image, 400, …coordinates);
```









```
function bounce(x) {
  return () => {
    context.drawImage(…, x, …);
    context.clear();
  }
}

setTimeout(bounce(0), 300);
setTimeout(bounce(200), 600);
setTimeout(bounce(400), 900);
setTimeout(bounce(200), 1200);
setTimeout(bounce(0), 1500);
```

I want to **resolve** an animation, **then** handle another animation.

```
var promise = new Promise((resolve, reject) => {
    …things
    resolve(value);
};

promise().then((value) => …more_things);
```

@gurlcode

Animation with Promises

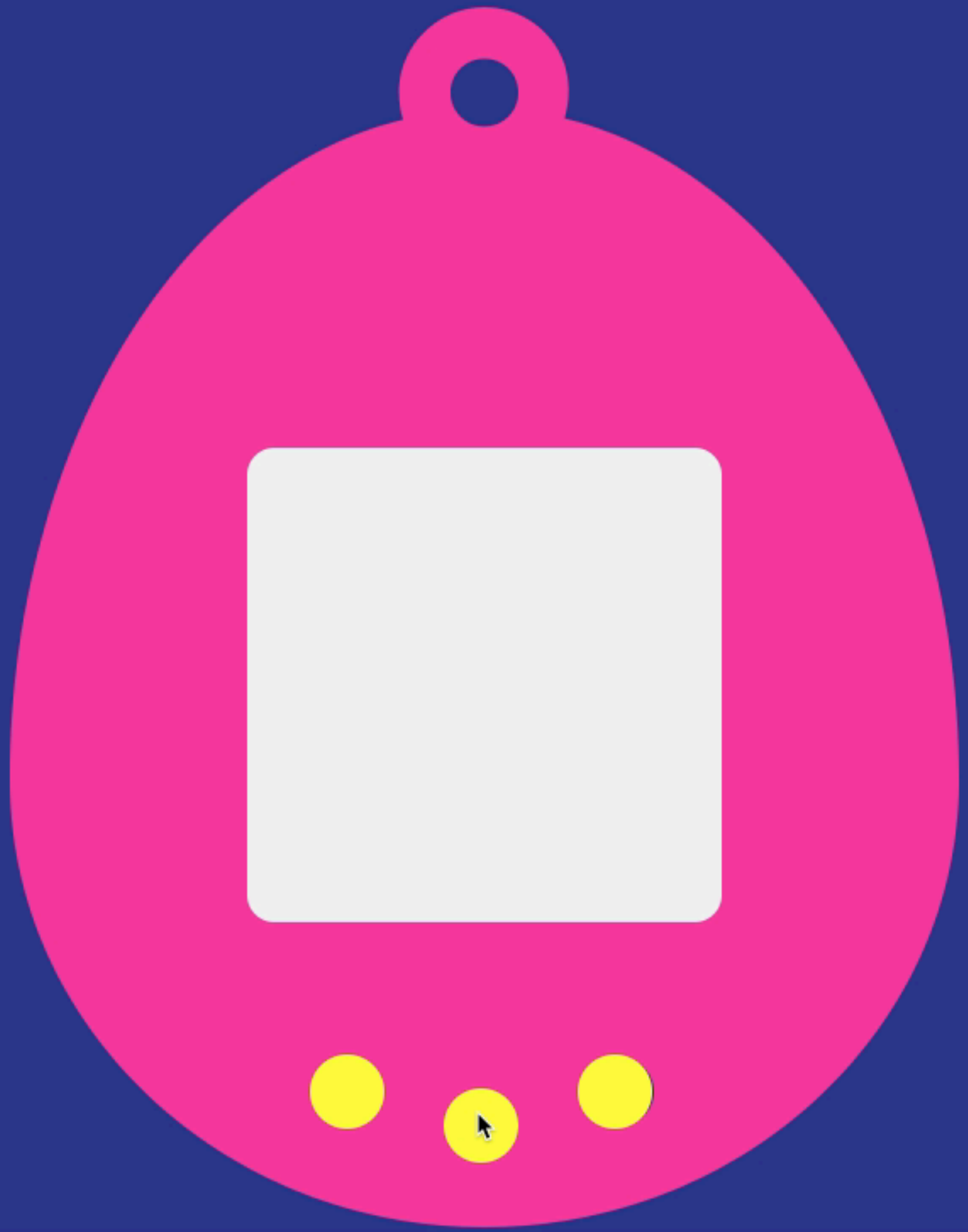
```
function animate(draw, ms) {
  return new Promise((resolve, reject) => {
    var animation = () => {
      context.clear();

      var isComplete = draw(resolve);

      if (!isComplete) {
        setTimeout(() => (animation), ms);
      }
    }

    animation();
  });
}
```

@gurlcode

Animation with Promises

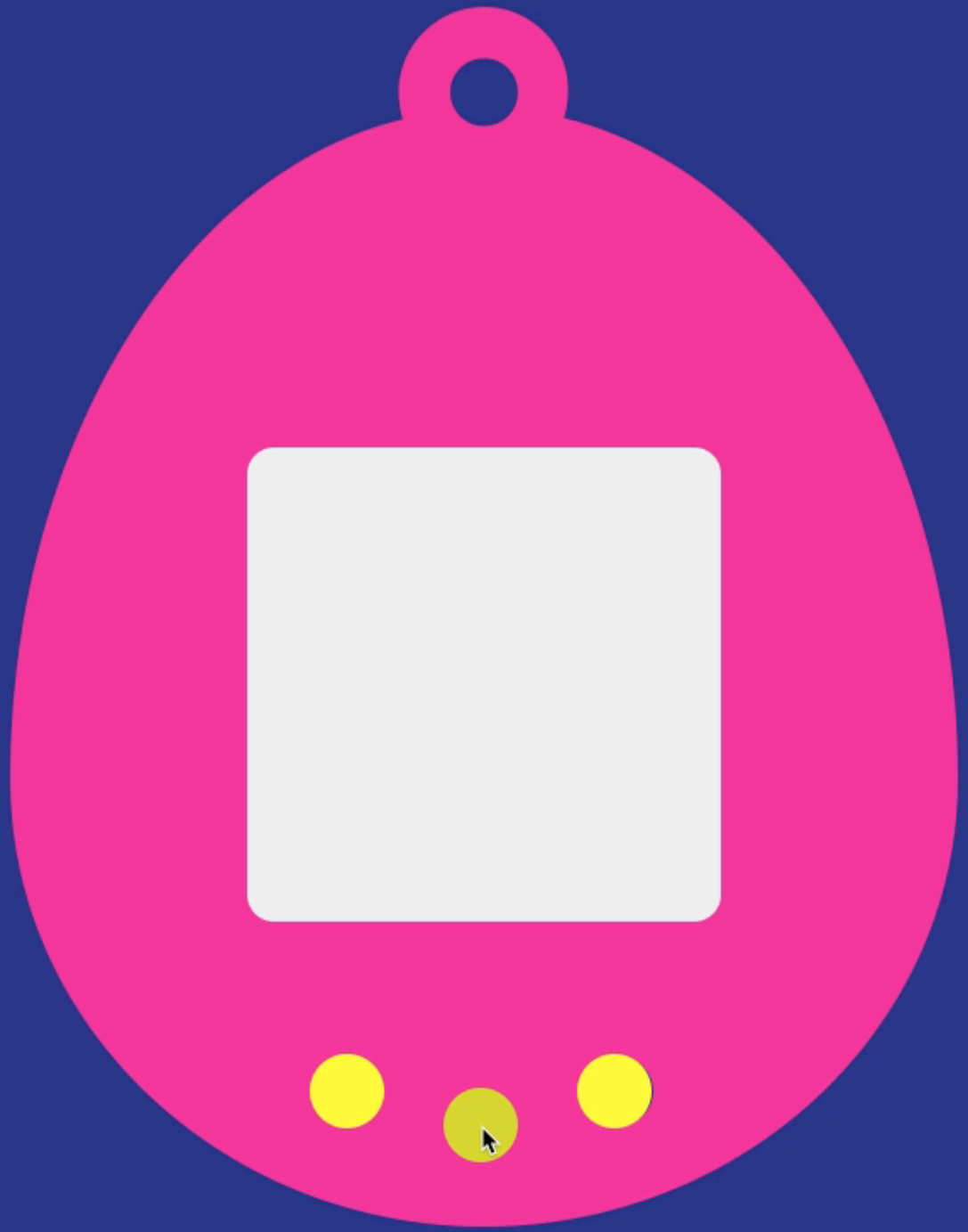
```
function bounceUp() {
  var frameWidth = 200;
  var frameCount = 2;
  var currentFrame = 0;

  return animate((resolve) => {
    context.drawImage(frameWidth * currentFrame, …);
    context.clear();
    currentFrame++;
    if (currentFrame > frameCount) {
      resolve();
      return true;
    }
  }, ms);
}
```

@gurlcode



```
function bounce() {
  return bounceUp
    .then(bounceDown);
}

bounce()
  .then(bounce);
  .then(bounce);
```

@gurlcode



```
function loop() {
  return idle()
           .then(loop);
}
```

@gurlcode

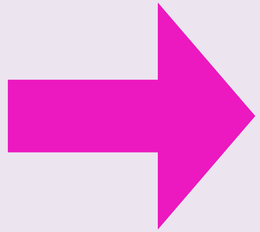
Except... we got problems

Thenable Hell

```
bounce(() => {
  moveRight(() => {
    moveLeft(() => {
      // Am I in hell?
    });
  });
})
```

Thenable Hell

```
bounce(() => {
  moveRight(() => {
    moveLeft(() => {
      // Am I in hell?
    });
  });
})
```



```
bounce()
  .then(() => moveRight(20))
  .then(() => moveLeft(40))
  .then(() => {
   var shouldFeed = …
   if () {…
   else { …
  })
  .then(bounce)
  .then( // Still in hell. )
```

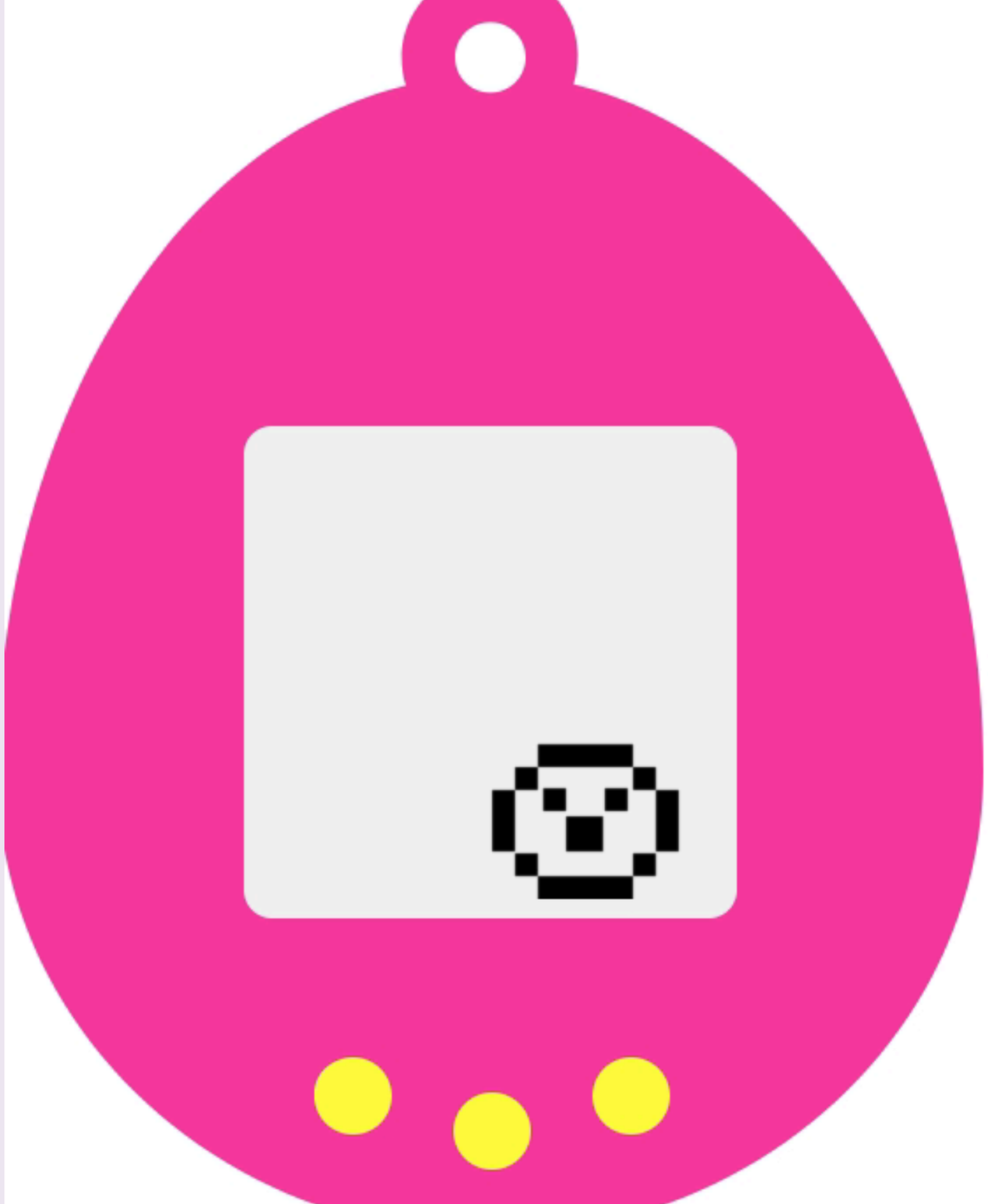
```
var pending = [];

function loop() {
  if (pending.length) {
    return handleEvent();
  }

  return tamagotchi.idle()
          .then(loop);
}
```

```
function handleEvent() {
  var event = pending.shift();
  event().then(loop);
}
```

@gurlcode

```
var pending = [];

function loop() {
  if (pending.length) {
    handleEvent();
  }

  return tamagotchi.idle()
          .then(loop);
}
```

```
function handleEvent() {
  var event = pending.shift();
  event().then(loop);
}
```



top

Filter

All levels


Promises are
Unbreakable Vows
@gurlcode

I want to pause an animation,
& yield to an event.

```
function* generator() {

    // pause

}
```

@gurlcode

Generators 101

```
function* count() {
  yield 1;
  yield;
  yield 3;
}
```

```
function* count() {
  yield 1;
  yield;
  yield 3;
}
```

```
var generator = count();
```

```
Generator {};
```

@gurlcode

```
function* count() {
  yield 1;
  yield;
  yield 3;
}

var generator = count();

generator.next();
```

```
Generator {};

{ value: 1, done: false }
```

```
function* count() {
  yield 1;
  yield;
  yield 3;
}

var generator = count();

generator.next();
generator.next();
generator.next();
generator.next();
```

```
Generator {};

{ value: 1, done: false }
{ value: undefined, done: false }
{ value: 3, done: false }
{ value: undefined, done: true }
```

@gurlcode

```
function* count() {
  yield 1;
  yield;
  yield 3;
}

var generator = count();

generator.next();
generator.next();
generator.next();
generator.next();
```

```
Generator {};

{ value: 1, done: false }
{ value: undefined, done: false }
{ value: 3, done: false }
{ value: undefined, done: true }
```

@gurlcode

```
function* count() {
  yield 1;
  yield;
  yield 3;
}

var generator = count();


generator.next();
generator.next();
generator.next();
generator.next();
```

```
Generator {};


{ value: 1, done: false }
{ value: undefined, done: false }
{ value: 3, done: false }
{ value: undefined, done: true }
```

@gurlcode

```
function* add() {
  const num = yield;
  yield 2 + num;
  yield 4 + num;
}

var generator = add();

generator.next();
generator.next(2);
generator.next();
generator.next();
```

```
Generator {};

{ value: undefined, done: false }
{ value: 4, done: false }
{ value: 6, done: false }
{ value: undefined, done: true }
```

@gurlcode

```
function* add() {
  const num = yield;
  yield 2 + num;
  yield 4 + num;
}

var generator = add();

generator.next();
generator.next(2);
generator.next();
generator.next();
```

```
Generator {};

{ value: undefined, done: false }
{ value: 4, done: false }
{ value: 6, done: false }
{ value: undefined, done: true }
```

```
function* add() {
  const num = yield;
  yield 2 + num;
  yield 4 + num;
}

var generator = add();

generator.next();
generator.next(2);
generator.next();
generator.next();
```

```
Generator {};

{ value: undefined, done: false }
{ value: 4, done: false }
{ value: 6, done: false }
{ value: undefined, done: true }
```

```
function* add() {
  const num = yield;
  yield 2 + num;
  yield 4 + num;
}

var generator = add();


generator.next();
generator.next(2);
generator.next();
generator.next();
```

```
Generator {};


{ value: undefined, done: false }
{ value: 4, done: false }
{ value: 6, done: false }
{ value: undefined, done: true }
```

Infinite Generators

```
function* forever(num) {
  while (true) {
    yield 2 + num;
  }
}

var generator = forever(2);
```

```
Generator {};
```

@gurlcode

```
function* forever(num) {
  while (true) {
    num = yield 2 + num;
  }
}

var generator = forever(2);

generator.next();
generator.next(2);
generator.next(2);
generator.next(2);
```

```
Generator {};

{ value: 4, done: false }
{ value: 6, done: false }
{ value: 8, done: false }
{ value: 10, done: false }
{ value: …, done: false }
```

yield*

```
function* outer() {
  yield 1;
  yield* inner();
  yield 2;
}

function* inner() {
  yield “a”;
  yield “b”;
}

var generator = outer();
```

```
Generator {};
```

@gurlcode

```
function* outer() {
  yield 1;
  yield* inner();
  yield 2;
}

function* inner() {
  yield “a”;
  yield “b”;
}

var generator = outer();
```

```
Generator {};
```

@gurlcode

```
function* outer() {
  yield 1;
  yield* inner();
  yield 2;
}

function* inner() {
  yield “a”;
  yield “b”;
}

var generator = outer();
```

```
Generator {};
```

@gurlcode

```
function* outer() {
  yield 1;
  yield* inner();
  yield 2;
}

function* inner() {
  yield “a”;
  yield “b”;
}
```

```
var generator = outer();
```

```
Generator {};
```

```
function* outer() {
  yield 1;
  yield* inner();
  yield 2;
}

function* inner() {
  yield “a”;
  yield “b”;
}
```

```
generator.next();
generator.next();
generator.next();
generator.next();
generator.next();
```

```
{ value: 1, done: false }
{ value: “a”, done: false }
{ value: “b”, done: false }
{ value: 2, done: false }
{ value: undefined, done: true }
```

```
function* outer() {
  yield 1;
  yield* inner();
  yield 2;
}

function* inner() {
  yield “a”;
  yield “b”;
}
```

```
generator.next();
generator.next();
generator.next();
generator.next();
generator.next();
```

```
{ value: 1, done: false }
{ value: “a”, done: false }
{ value: “b”, done: false }
{ value: 2, done: false }
{ value: undefined, done: true }
```

```
function* outer() {
  yield 1;
  yield* inner();
  yield 2;
}

function* inner() {
  yield “a”;
  yield “b”;
}
```

```
generator.next();
generator.next();
generator.next();
generator.next();
generator.next();
```

```
{ value: 1, done: false }
{ value: “a”, done: false }
{ value: “b”, done: false }
{ value: 2, done: false }
{ value: undefined, done: true }
```

```
function* outer() {
  yield 1;
  yield* inner();
  yield 2;
}

function* inner() {
  yield “a”;
  yield “b”;
}
```

```
generator.next();
generator.next();
generator.next();
generator.next();
generator.next();
```

```
{ value: 1, done: false }
{ value: “a”, done: false }
{ value: “b”, done: false }
{ value: 2, done: false }
{ value: undefined, done: true }
```

Canceling a Generator

```
function* forever(num) {
  while (true) {
    yield 2 + num;
    return;
  }
}

var generator = forever(2);

generator.next();
generator.next();
generator.next();
```

```
Generator {};

{ value: 4, done: false }
{ value: undefined, done: true }
{ value: undefined, done: true }
```

```
function* forever(num) {
  while (true) {
    yield 2 + num;
    return;
  }
}

var generator = forever(2);


generator.next();
generator.next();
generator.next();
```

```
Generator {};


{ value: 4, done: false }
{ value: undefined, done: true }
{ value: undefined, done: true }
```

```
function* forever(num) {
  while (true) {
    yield 2 + num;
    return;
  }
}

var generator = forever(2);


generator.next();

generator.next();

generator.next();
```

```
Generator {};


{ value: 4, done: false }

{ value: undefined, done: true }

{ value: undefined, done: true }
```

```
function* forever(num) {
  while (true) {
    yield 2 + num;
  }
}

var generator = forever(2);

generator.next();

generator.return();
```

```
Generator {};

{ value: 4, done: false }

{ value: undefined, done: true }
```

```
function* forever(num) {
  while (true) {
    yield 2 + num;
  }
}

var generator = forever(2);


generator.next();
generator.return();
```

```
Generator {};

 { value: 4, done: false }
{ value: undefined, done: true }
```

@gurlcode

Resuming the generator occurs outside of the generator.

```
function* generatorFunc() {
  yield 1;
  yield 2;
}

var generator = generatorFunc();

generator.next();
```

@gurlcode

WHO WATCHES THE WATCHMEN?

runs generators

@gurlcode

Coroutines!

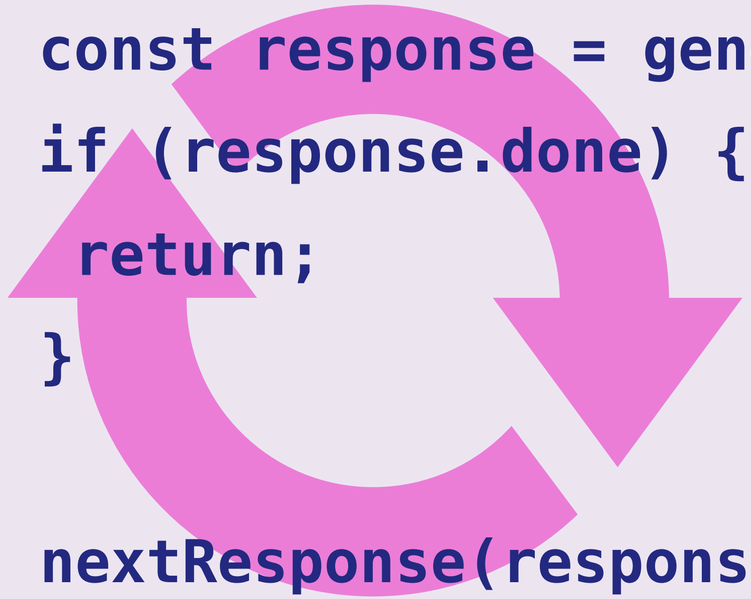
@gurlcode

Coroutines are a general control structure whereby control flow is cooperatively passed between two different routines.

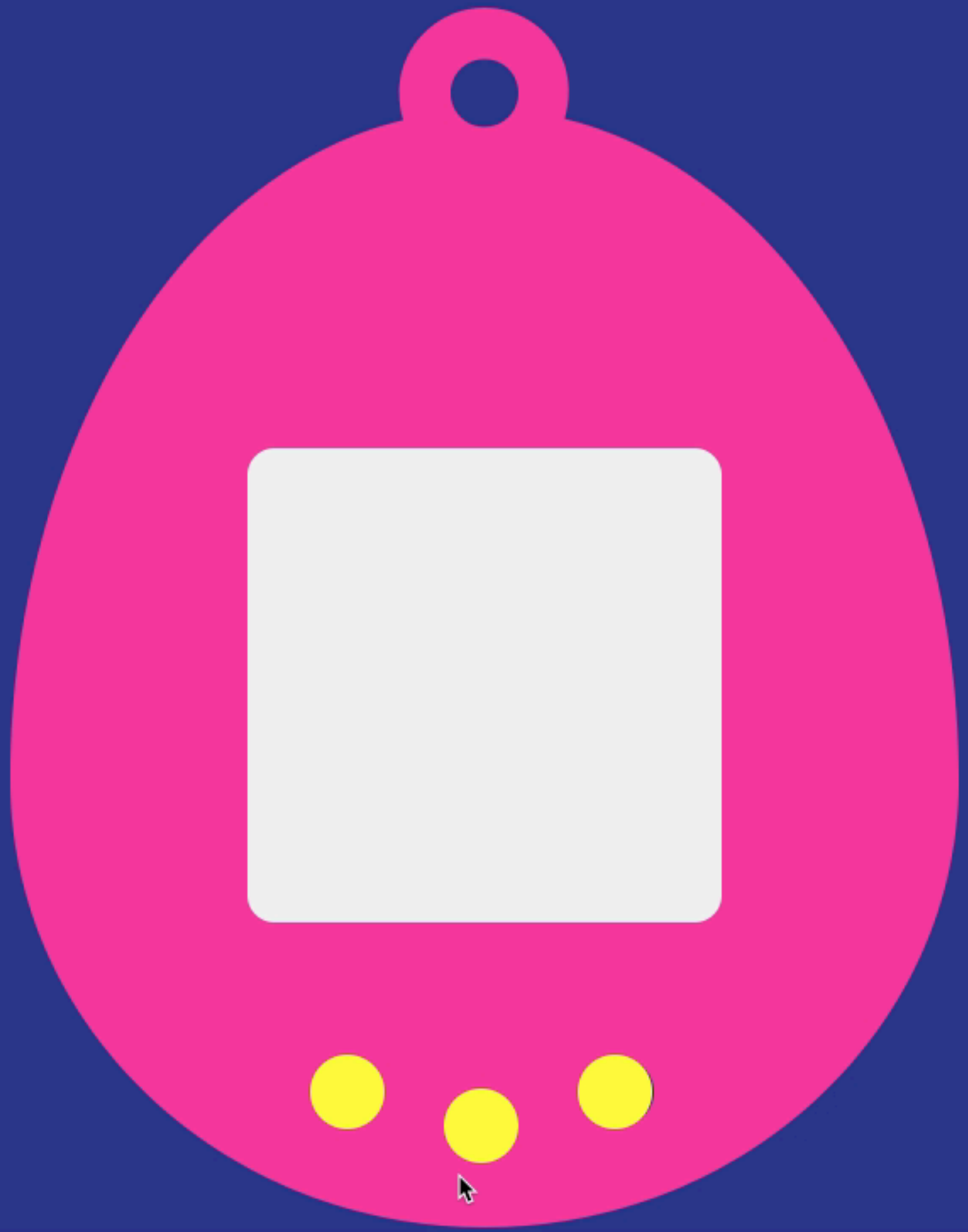
```
function coroutine(generatorFunc) {
  const generator = generatorFunc();
  nextResponse();

  function nextResponse(value) {
    const response = generator.next(value);
    if (response.done) {
     return;
    }

    nextResponse(response.value);
  }
}
```

```
function coroutine(generatorFunc) {
  const generator = generatorFunc();
  nextResponse();

  function nextResponse(value) {
    const response = generator.next(value);
    if (response.done) {
      return;
    }

    nextResponse(response.value);
  }
}
```

```
function coroutine(generatorFunc) {
  const generator = generatorFunc();
  nextResponse();

  function nextResponse(value) {
    const response = generator.next(value);
    if (response.done) {
     return;
    }

    nextResponse(response.value);
  }
}
```



@gurlcode



```
function* bounceUp() {
  yield context.drawImage(…);
  yield context.drawImage(…);
  yield context.drawImage(…);
  // …etc
}

coroutine(bounceUp);
```

@gurlcode

```
function* bounceUp() {
  yield new Promise(…);
  yield new Promise(…);
  yield new Promise(…);
  // …etc
}

coroutine(bounceUp);
```

```
function nextResponse(value) {
  const response = generator.next(value);
  if (response.done) {
   return;
  }

  handleAsync(response.value);
}
```

```
function handleAsync(value) {
  value.then(nextResponse);
}
```

```
function bounce() {
  return bounceUp
    .then(bounceDown);
}
```

==

```
coroutine(function* bounce() {
  yield bounceUp;
  yield bounceDown;
});
```

@gurlcode

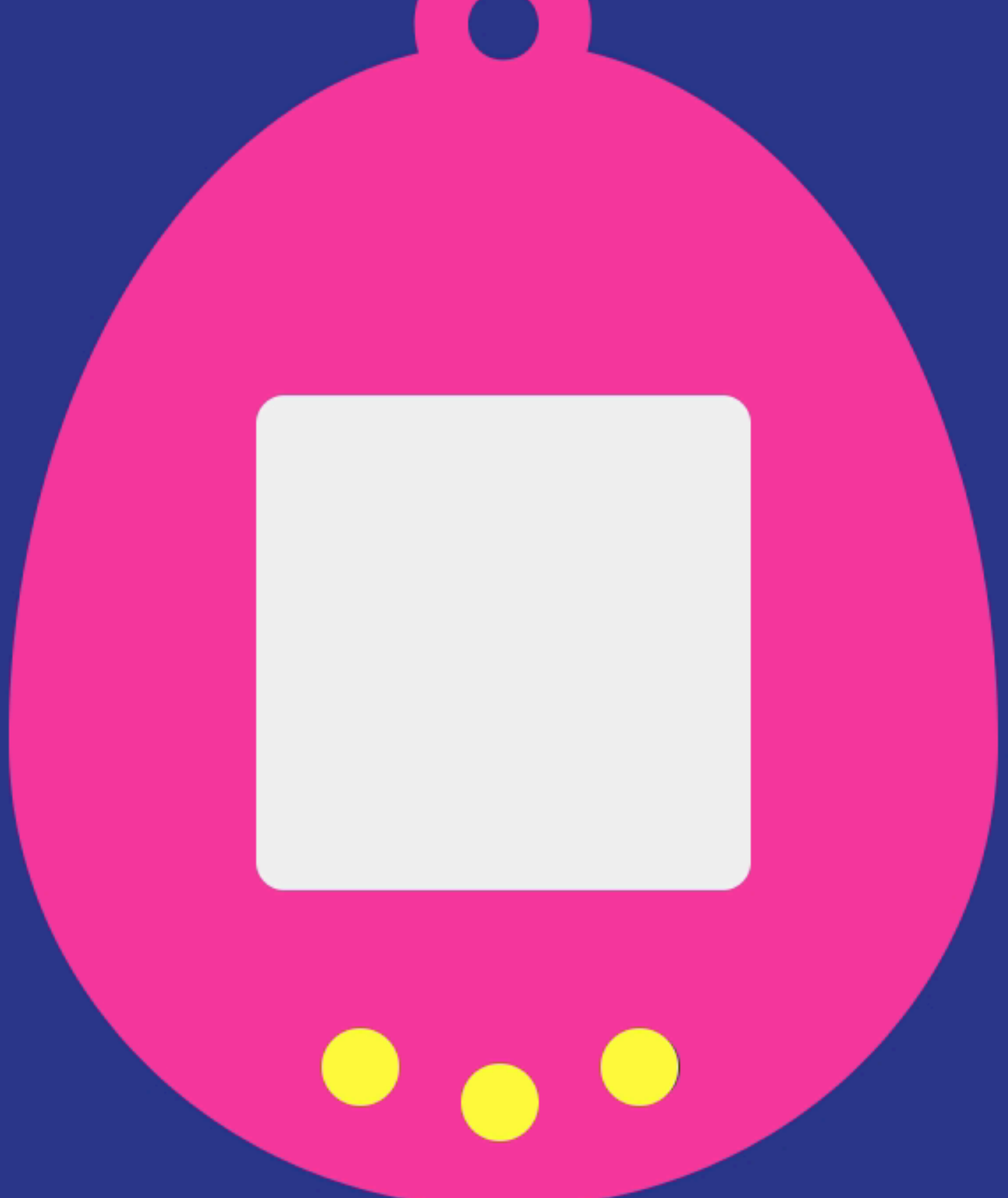
```
co(function* () {
  yield promise();
  yield (callback) => {
    // do some things
    callback()
  };
  yield generatorFunction;
});
```

https://github.com/tj/co

You can think sequentially about async code.

```
function* idle() {
  yield* bounce;
  yield* moveRight;
  yield* moveLeft;
  yield* bounce;
}
```

@gurlcode



@gurlcode

```
function delay(ms) {
  return new Promise((resolve) =>
   setTimeout(() => resolve(), ms));
}
```

```
function clear() {
  context.clear(…);
}
```

```
function draw(image, frame, …) {
  context.drawImage(image, frame, …);
}
```

```
function delay(ms) {
  return new Promise((resolve) =>
   setTimeout(() => resolve(), ms));
}
```

```
function clear() {
  context.clear(…);
}
```

```
function draw(image, frame, …) {
  context.drawImage(image, frame, …);
}
```

```
function delay(ms) {
  return new Promise((resolve) =>
   setTimeout(() => resolve(), ms));
}
```

```
function clear() {
  context.clear(…);
}
```

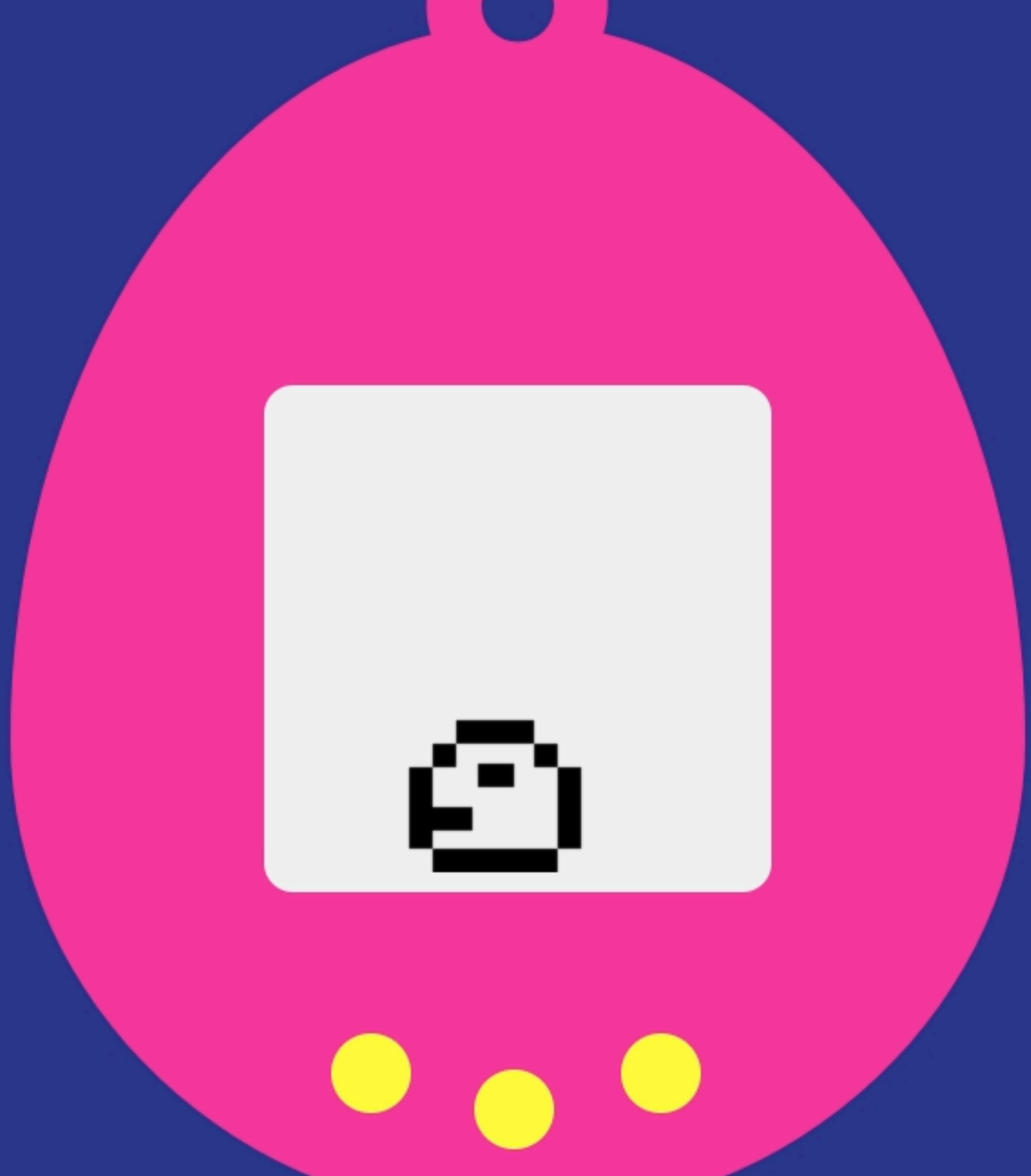
```
function draw(image, frame, …) {
  context.drawImage(image, frame, …);
}
```

```
function* drawFrame(image, frame, ms = 100) {
  clear();
  draw(image, frame);
  yield delay(ms);
}
```

```
function* dislike() {
  yield* drawFrame('dislike', 0);
  yield* drawFrame('dislike', 1);
}
```

```
function* drawFrame(image, frame, ms = 100) {
  clear();
  draw(image, frame);
  yield delay(ms);
}
```

```
function* dislike() {
  yield* drawFrame('dislike', 0);
  yield* drawFrame('dislike', 1);
}
```



```
coroutine(function* loop() {
  let done;
  let animation = idle();

  while (true) {
    if (pending) {
      animation.return();
      yield* event();
    }

    while (!done && !pending) {
      const next = animation.next();
      done = next.done;
      yield next.value;
    }

    animation = idle();
    done = false;
  }
});
```

```
coroutine(function* loop() {
  let done;
  let animation = idle();

  while (true) {
   if (pending) {
     animation.return();
     yield* event();
   }

   while (!done && !pending) {
     const next = animation.next();
     done = next.done;
     yield next.value;
   }

   animation = idle();
   done = false;
  }
});
```

```
coroutine(function* loop() {
  let done;
  let animation = idle();

  while (true) {
   if (pending) {
     animation.return();
     yield* event();
   }

   while (!done && !pending) {
     const next = animation.next();
     done = next.done;
     yield next.value;
   }

   animation = idle();
   done = false;
  }
});
```

```
coroutine(function* loop() {
  let done;
  let animation = idle();

  while (true) {
   if (pending) {
     animation.return();
     yield* event();
   }

   while (!done && !pending) {
     const next = animation.next();
     done = next.done;
     yield next.value;
   }

   animation = idle();
   done = false;
  }
});
```

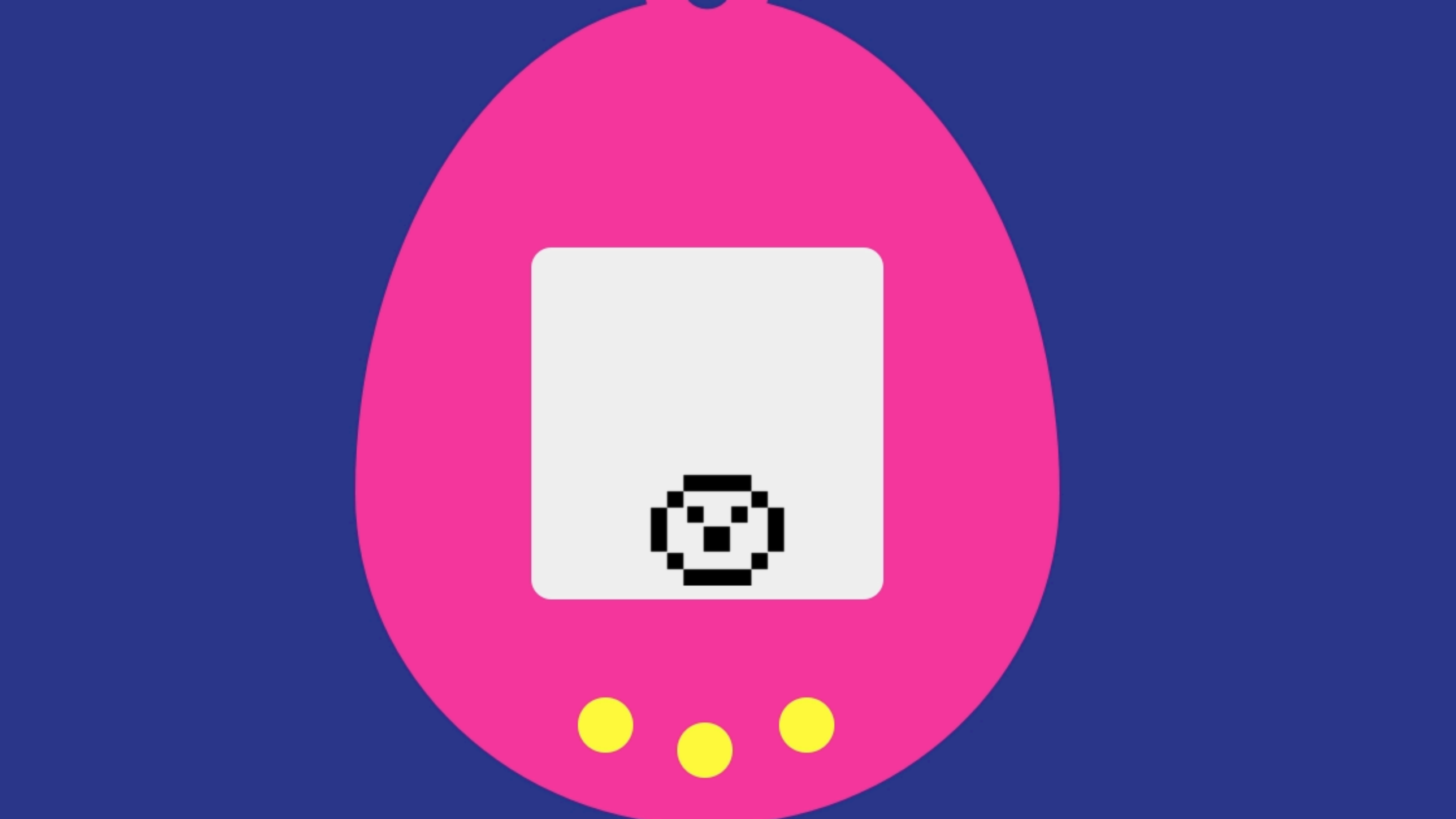
```
function* feed() {
  if (hunger === 0) {
    yield* dislike();
  }

  hunger--;
  yield* eat();
}
```

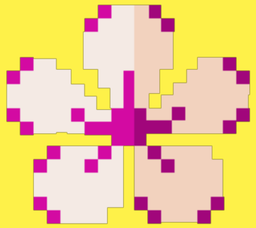
```
function* dislike() {
  yield* drawFrame('dislike', 0);
  yield* drawFrame('dislike', 1);
}
```

```
function* feed() {
  if (hunger === 0) {
    yield* dislike();
  }

  hunger--;
  yield* eat();
}
```

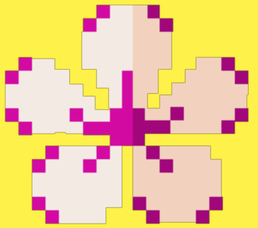
```
function* eat() {
  yield* drawFrame('eat', 0);
  yield* drawFrame('eat', 1);
}
```

@gurlcode





Thank you!



https://github.com/jcreighton/tamagotchi

@gurlcode