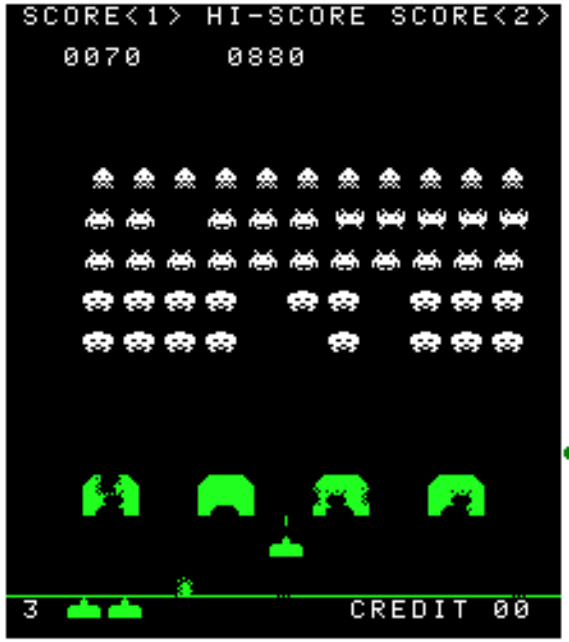
# Model, View, and Controller Classes for Flappy Bird, Event Handlers

Dr. Jason Schanker

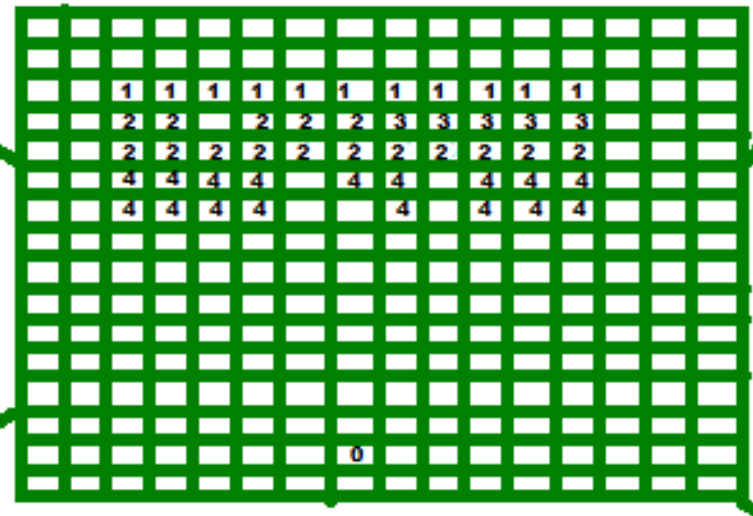
# Different Screens for Android

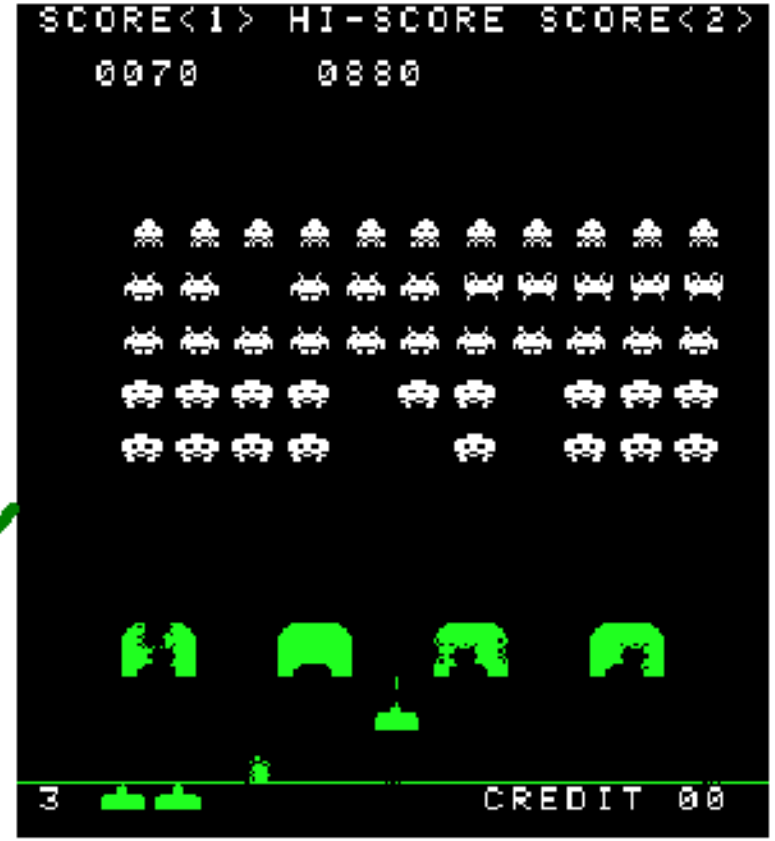
- Separating model classes from view classes allows you to support multiple view types and add new ones without touching model code
  - Less likely to introduce errors; contain errors to smaller blocks of code
  - Allows reuse of game play code in future remakes
- Mobile devices have different screen sizes, different screen resolutions, etc. making separation of the view and model classes even more important.
  - Model Grid can be used to generate View (e.g., for 16 X 16 model, each grid square can be 20 X 20 pixels on 320 X 320 pixel screen, 10 X 40 on 160 X 640 pixel screen)




VIEW 1

# SPACE INVADERS
# MODEL
# WORLD




VIEW 2

```
A A A A A A A A A A A
C C   C C C D D D D D
C C C C C C C C C C C
E E E E   E E   E E E
E E E E     E   E E E
```

**0 = SPACESHIP**

**1, 2, 3, 4 = ALIENS**
**9 = LASER**

3D VIEW

```
I
S
```

VIEW 3 (TEXT)

# MVC in a Nutshell

- Separate the Model, View, and Controller Classes
  - Model class (Brain): Stores the data, determines the logic of how it should be maintained/updated
    - Example: Bird Class keeps track of bird's position, velocity, how to update position and velocity when told to move or flap. Position is with respect to a fixed grid (e.g., 320 X 480) that does **NOT** change when the size of or dimensions of the View does. The model updates the view as necessary, but it's the view's responsibility to e.g., scale according to the screen size.
  - View class: Representation/Appearance of data
    - Example: WorldView class to update the canvas according to the bird's x and y position when directed by the Model; draws bird at scaled y position on the Canvas, sky background position at the opposite of the Bird's scaled x position.
  - Controller class: Responds to events (e.g., input, change in time, phone calls, low battery, etc.)
    - Example: Call bird's flap method on clicks; move method on redraw of screen
- Design heuristic for making applications easier to maintain, update, reuse, and extend; not a set of absolute rules
- Exact meaning can differ depending on context and who you ask

# Events and Listeners

- An event is anything that may be of interest or may need to be acted upon. Can schedule function or so called event-listener to be put on the queue to be run when a certain event occurs. Note: There can be multiple listeners for the same event.
  - Example: When the canvas element with the id game is clicked, display an alert box with the message, “You clicked me.” and log a message “Foo” to the console.

```
let greeting = () => alert("You clicked me!");
let foo = () => console.log("Foo");
document.getElementById("game").addEventListener("click", greeting);
document.getElementById("game").addEventListener("click", foo);
```

# Flappy Bird Controller Class Exercise

- Add the Bird Model Class to your code. Then create a new instance of a Bird `b`. Then define a Controller class that has a constructor taking a Bird Model `m` as its only argument. In the constructor, add an event listener that calls `m.flap.bind(m)` when the canvas element is clicked. Then create a new instance of a Controller, passing the Bird `b`.
- https://repl.it/Kwt3/80

# Flappy Bird View Class

- Create a `WorldView` class:
  - Every `WorldView` should have a `model` property that stores a reference to the Bird (model) it's going to be displaying. This bird should be passed as an argument to the constructor.
  - Every `WorldView` should also have a reference to the canvas element where it'll be displaying this bird and create a bird image and sky background image. It should store references to these newly created images.
  - It should have a `render` method that clears the canvas, draws the bird at a fixed x position (say 20), gets the bird's position from `this.model.position`, displays the bird at the y coordinate of this position and draws the sky at an x position that's the opposite of the x coordinate of the bird's position (e.g., -50 if `this.model.position.x = 50` ); draw the sky first then the bird .

# Adding to the Controller

- Create a new instance of a `WorldView` class `birdWorldView`. Then add a parameter variable `v` to the Controller class's constructor. In the constructor, define a function `runGame` that takes milliseconds as an argument, calls `m.move`, passing the number of seconds that elapsed since the last call to it and then calls `v.render`, and finally schedules runGame to be called again at the next time the screen is repainted, using `requestAnimationFrame`. Inside the constructor but outside of `runGame`, call `requestAnimationFrame(runGame)` to run it for the first time.
- **Note**: In the constructor, you will also need to define and initialize a local variable (via `let lasTime = ____`) that you update in `runGame` to determine the time that's elapsed since the last time `runGame` was called.