Ethan Yang(yy3526)

Tianshuo Jin (tj2591)

Zidong Xu(zx2507)

Sijun Li(sl5707)

Overview

1. Block Diagram
2. Game preparation
3. Register Table
4. VGA Controller
5. Background Scrolling
6. Pipe Generation
7. Collision Detection
8. Score
9. USB Keyboard
10. Demo

Overview

- Inspired by:

  Classic Flappy Bird mobile game

- Core Strategies:

  Press the space button to make the bird flap upward

  Gravity effect causes the bird to fall if no input is given

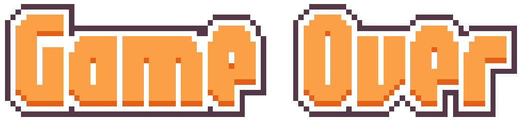
  Generate the gap of pipes randomly

  Player must control the bird through vertical gaps between pipes

  Aim to fly as far as possible to maximize score

Image processing

- Generate a memory initialization file(.mif) for each image via python
- Single-port ROM memory blocks
- 8-bit index color
- bg, base, bird0, bird1, bird2, gameover



Gameover.png

Convert.py

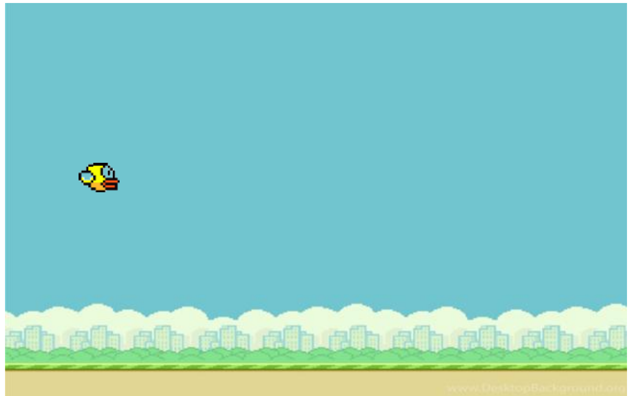
```
WIDTH=8;
DEPTH=8064;
ADDRESS_RADIX=UNS;
DATA_RADIX=HEX;
CONTENT BEGIN
0 : FF;
1 : FF;
2 : FF;
3 : FF;
4 : 00;
5 : 00;
6 : 45;
7 : 45;
8 : 45;
9 : 45;
10 : 45;
```

Gameover.mif

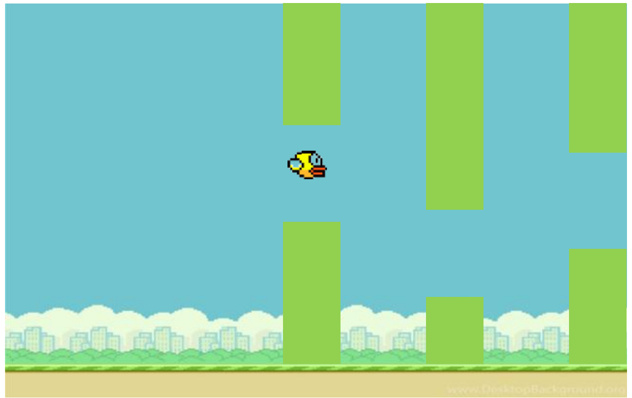
Game Logic & State

- Interaction between user and hardware
  User: Keyboard (Space)
  Hardware: VGA controller

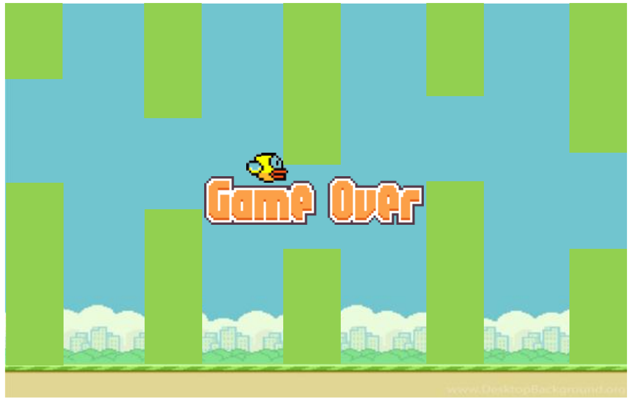
WAITING



PLAYING

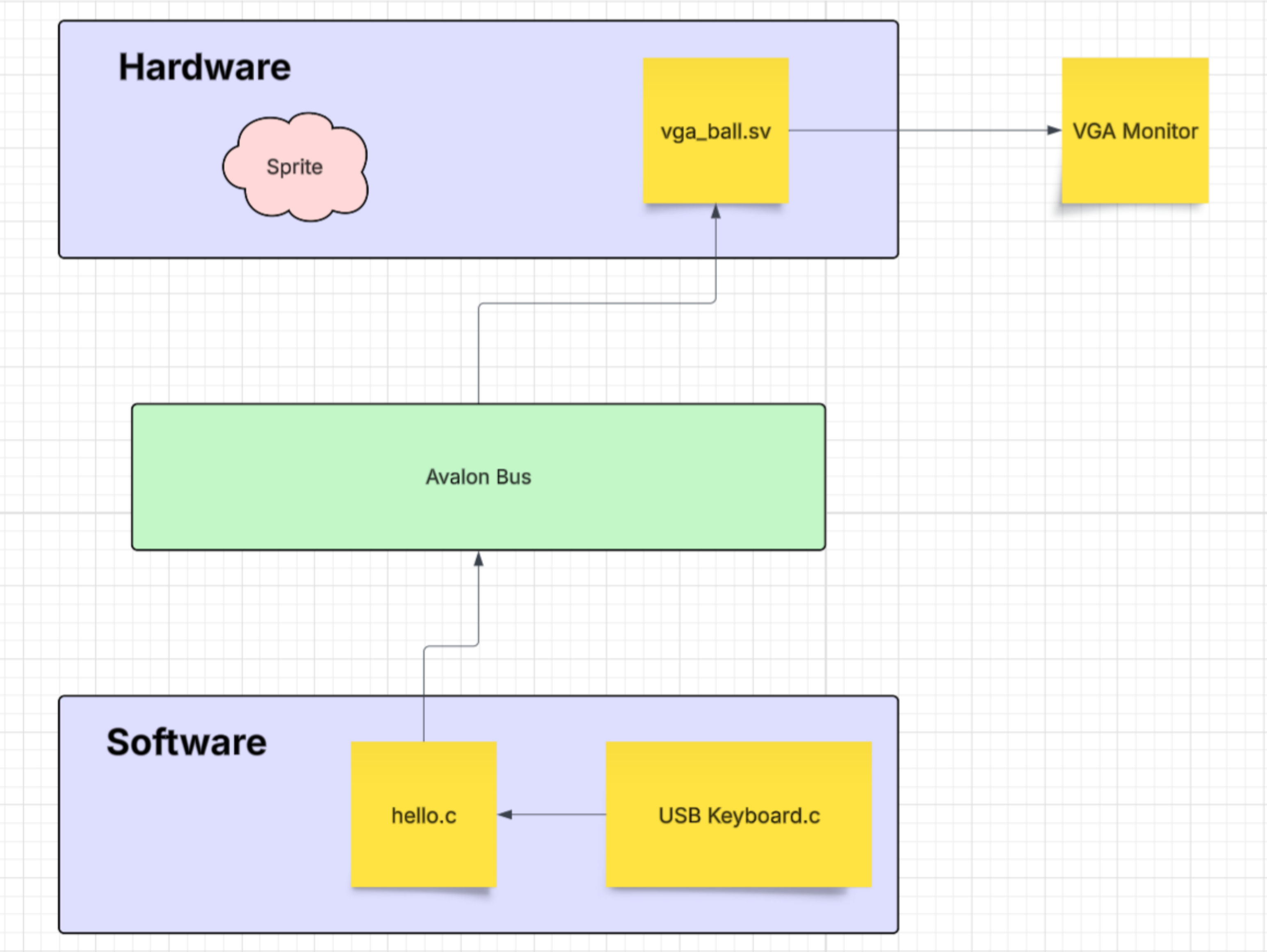


GAME_OVER



Trigger: collision == 1

Block Diagram



Register Map (Memory-Mapped & Internal)

Offset	Function	Description
0-2	Background color	R, G, B components (0-255)
3-4	X position (not in use)	Low 8 bits (3), High 2 bits (4)
5-6	Y position (not in use)	Low 8 bits (5), High 2 bits (6)
7	Flap signal (replaced radius)	Input from keyboard (0/1)

Register	Width	Purpose
bird_y	[9:0]	Vertical position of the bird
bird_velocity	signed [9:0]	Vertical speed; affected by gravity and flap
bird_frame	[1:0]	Index of bird animation frame (0–2)
animation_counter	[7:0]	Counter to cycle bird frames
vsync_reg	1	Stores VGA_VS for edge detection
game_started	1	0 = idle animation; 1 = game running
flap_latched	1	Latched flap input, reset once consumed
bird_y_idle	[9:0]	Bird idle animation Y position
idle_dir	1	Direction of idle movement (0 = up, 1 = down)
hcount, vcount	[10:0], [9:0]	VGA counters for pixel row/column
bg_addr	[18:0]	Address into background ROM
bg_color	[7:0]	Pixel color read from background ROM
bird_addr	[11:0]	Address into current bird sprite ROM
bird_color	[7:0]	Selected pixel color from current frame ROM
pipe_x[i]	[9:0] (array)	X position of each pipe pair (3 total)
pipe_gap_y[i]	[9:0] (array)	Y position of the top of the gap in each pipe pair
bird_color0/1/2	[7:0]	Outputs of 3 sprite ROMs for animation

VGA Controller

Multi-Layered Sprite System

- **ROM-Based Sprite Storage**:
  - Background, bird frames, ground, and game over sprites stored in separate ROM modules
  - Each ROM contains pre-designed pixel data with color information
  - Hardware accesses these ROMs in real-time during rendering
- **Layer Priority Hierarchy**:
  - Background (sky) - lowest priority
  - Pipes (simple colored rectangles)
  - Ground layer with texture
  - Bird sprite with animation
  - Score display and game over screen - highest priority
  - Higher priority layers override lower ones when they overlap
- **Address Calculation for Sprites**:
  - Each sprite has specific dimensions (e.g., bird is 34×24 pixels)
  - Current screen position determines which ROM address to access:
    ```
    bird_addr = (vcount - bird_y) * BIRD_WIDTH + (hcount[10:1] - BIRD_X)
    ```
  - Similar calculations for background, ground, and game over screen
  - Pipeline ensures addresses ready before pixel data needed

VGA Controller

Bird Animation System

•**Multi-Frame Bird Animation**:

- Three separate bird sprite ROMs for different wing positions
- Animation counter tracks timing between frame changes
- Current bird frame selected based on counter value:
  `animation_counter increments each cycle`
- `When counter reaches threshold, switch to next frame`
- Animation continues even during game states

Background Scrolling Technique

•**Continuous Background Scroll**:

- Background position shifts based on scroll offset counter

Transparency and Color Management

•**Sprite Transparency**:

- Special color values (0x00) indicate transparent pixels
- Rendering logic checks pixel value before display:
  `If current pixel is transparent, render lower priority layer instead`
- Allows for irregular shapes and smooth overlays

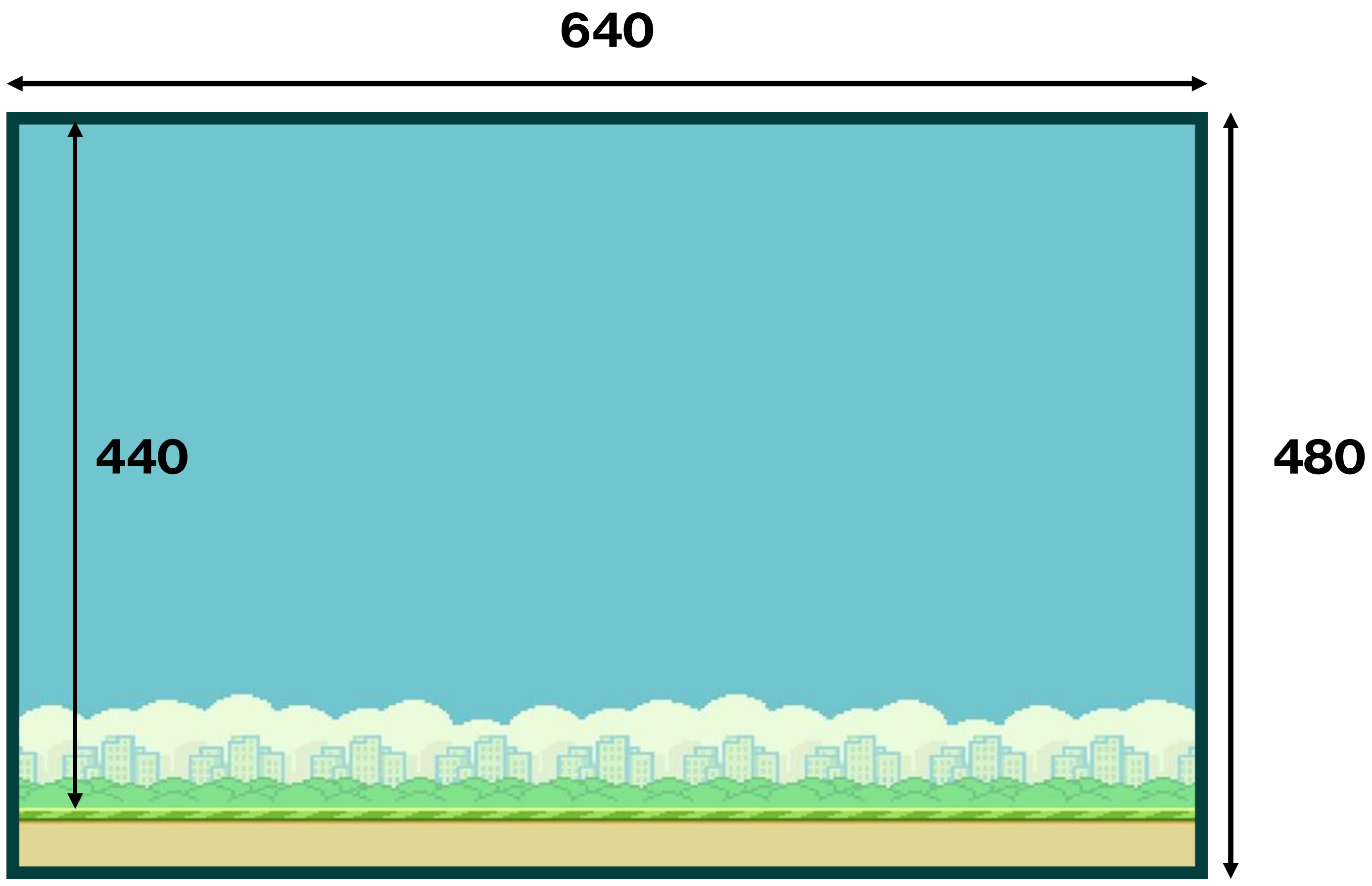
•**Color Conversion**:

- ROM stores color in 8-bit RGB332 format (3 bits R, 3 bits G, 2 bits B)
- Hardware converts to full 24-bit color (8 bits per channel):

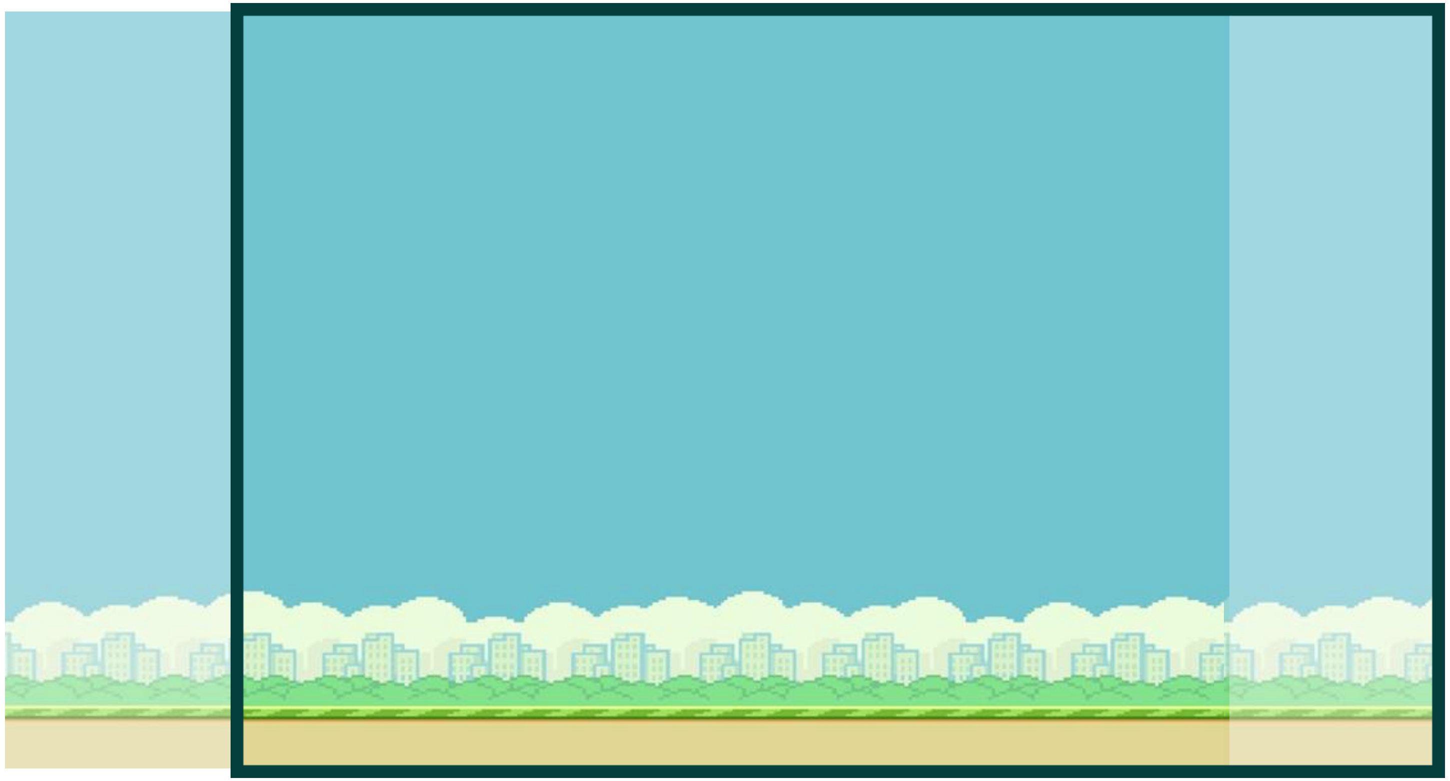
```
VGA_R = {sprite_color[7:5], 5'b00000}

VGA_G = {sprite_color[4:2], 5'b00000}

VGA_B = {sprite_color[1:0], 6'b000000}
```

•Maintains color fidelity while optimizing storage

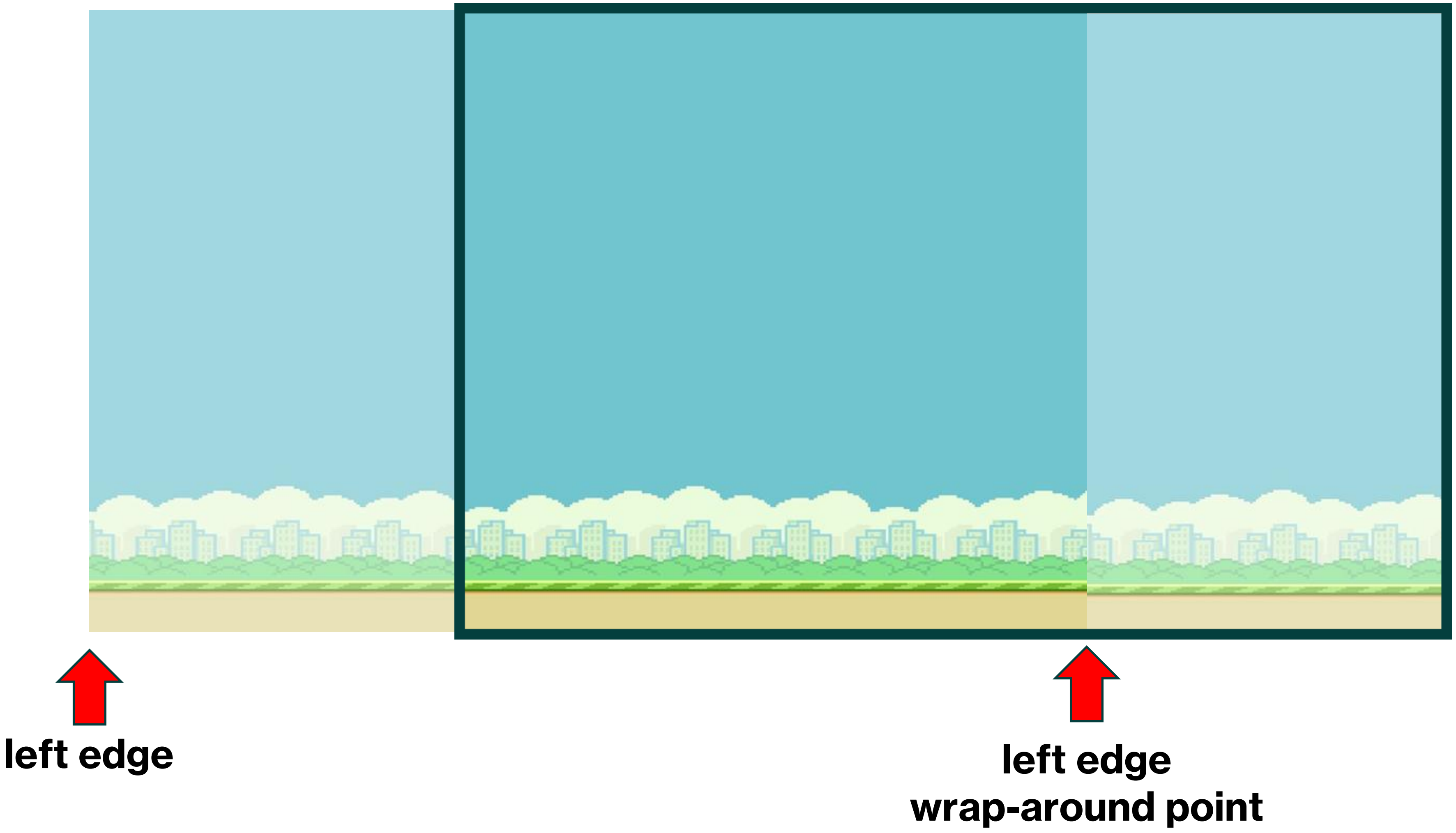
Background Scrolling



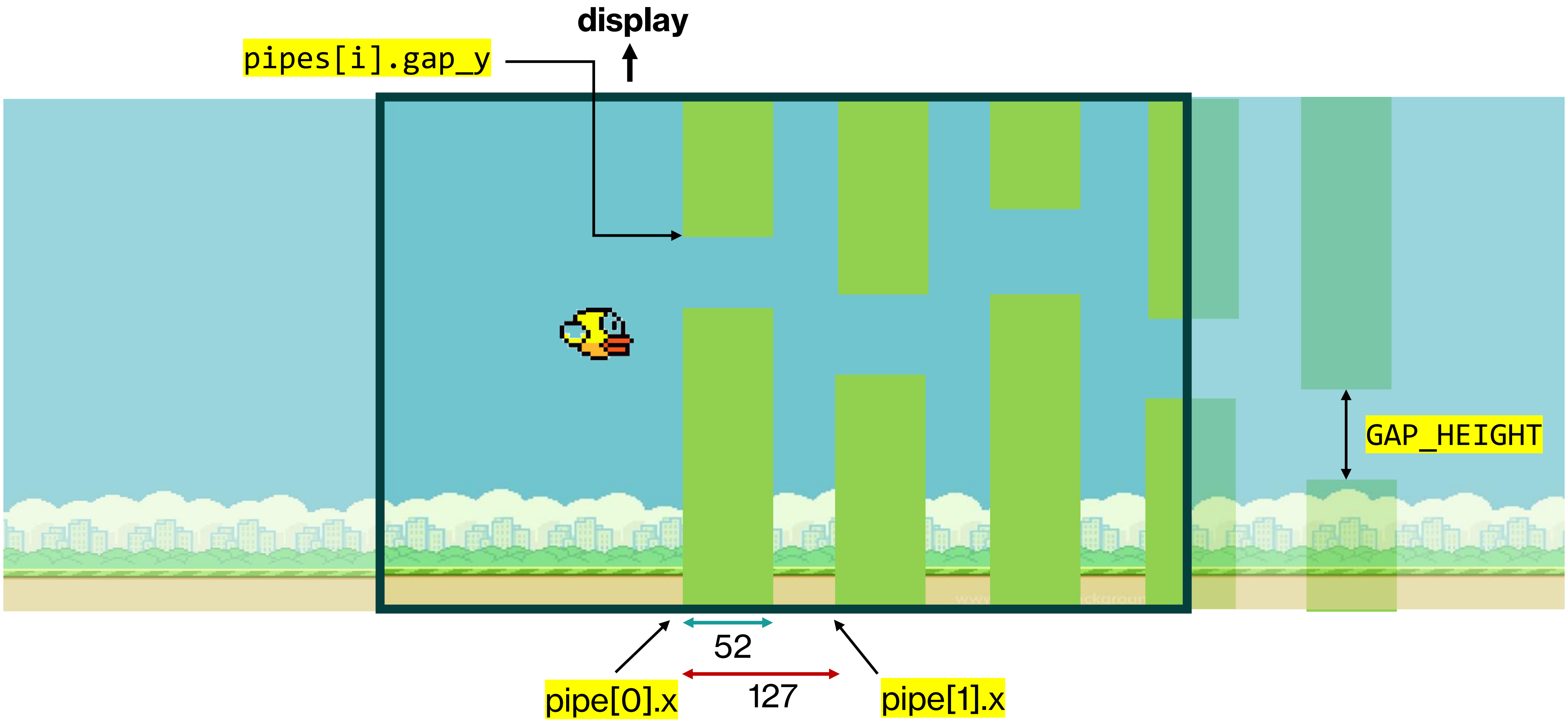
Background Scrolling



Background Scrolling

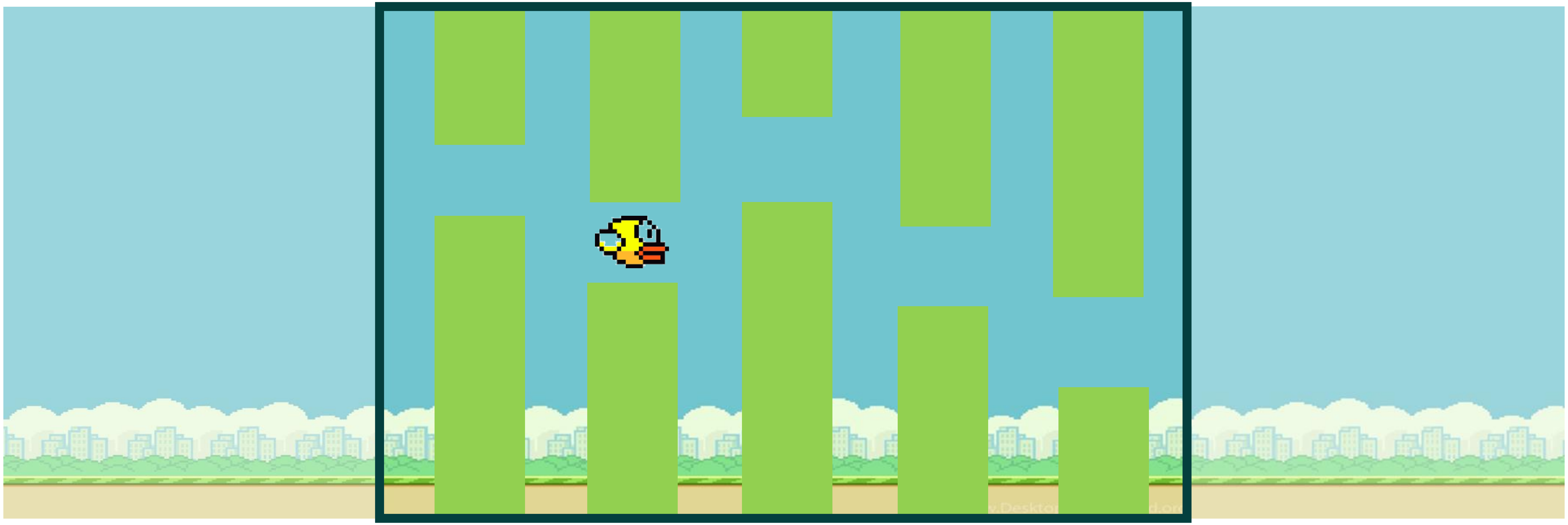


Pipe Generation

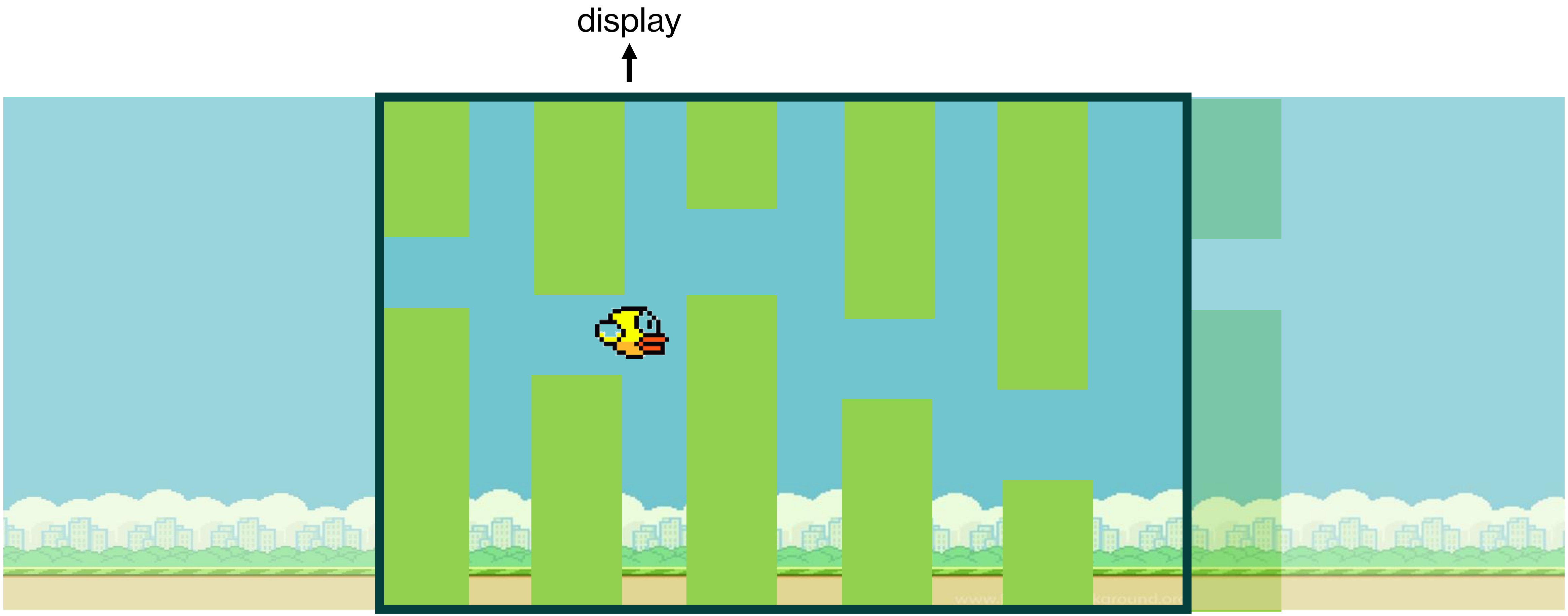


Pipe Generation

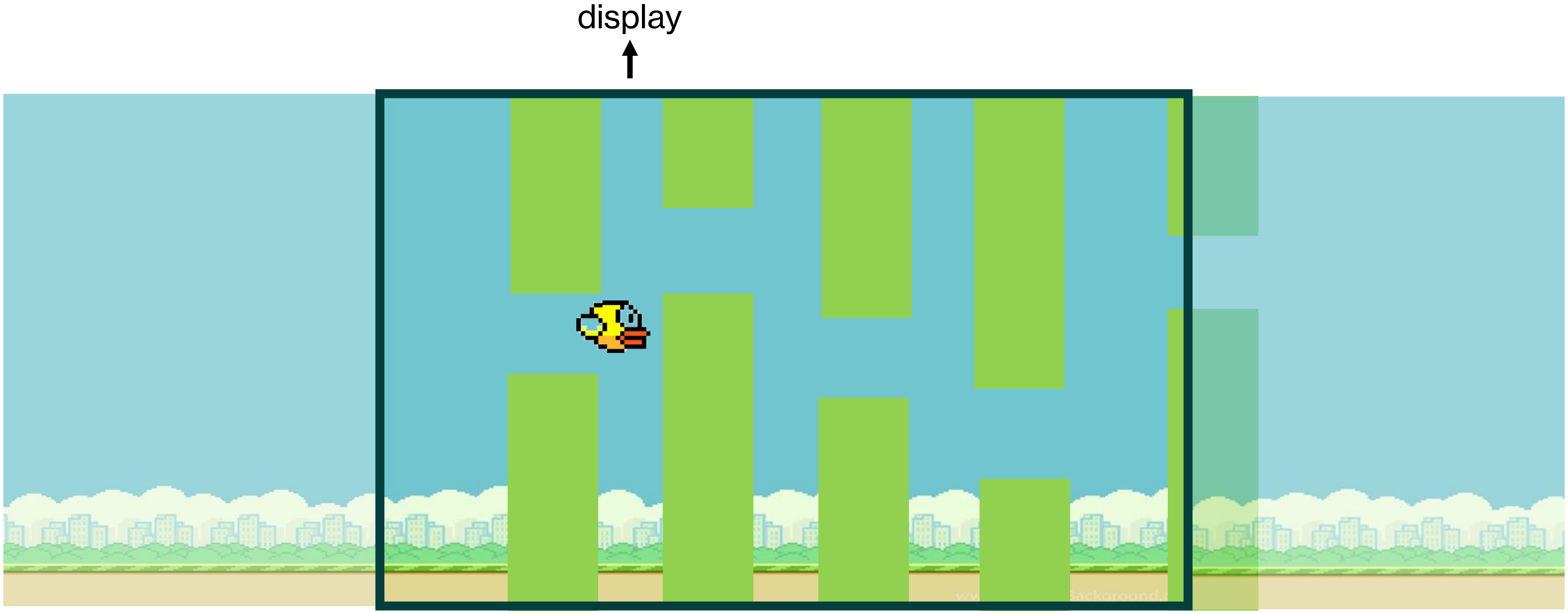
display



Pipe Generation

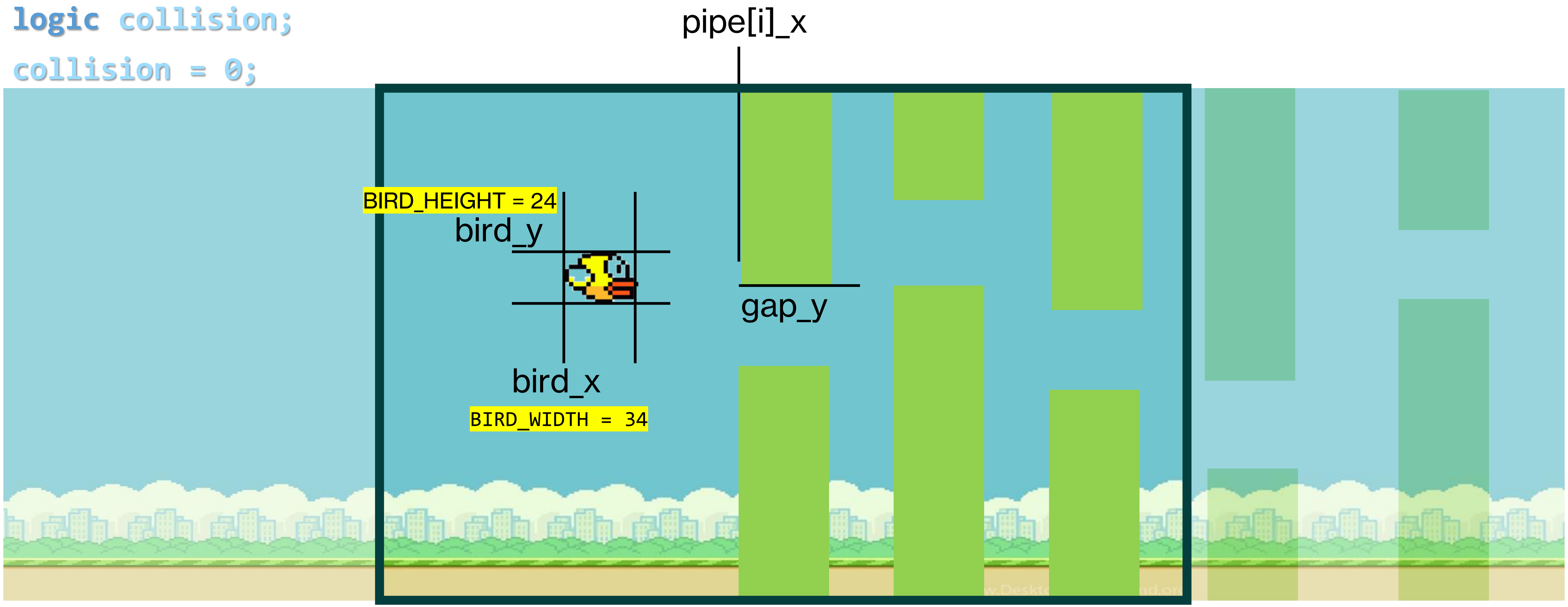


Pipe Generation



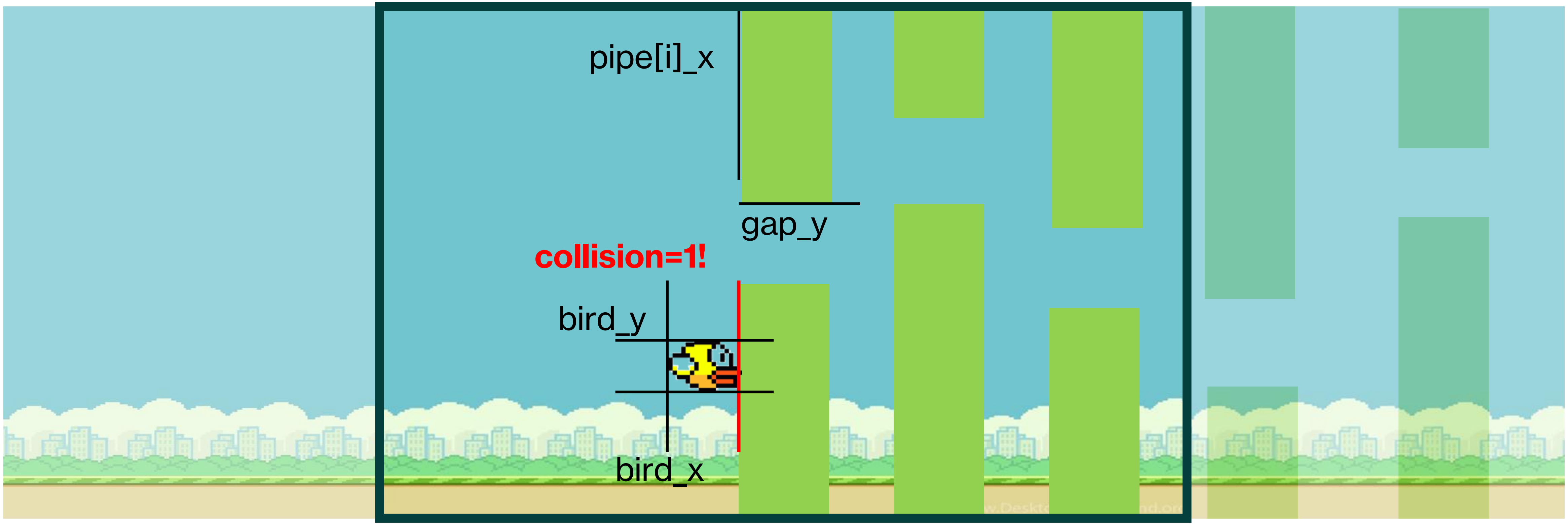
Collision Detection

```
logic collision;
collision = 0;
```



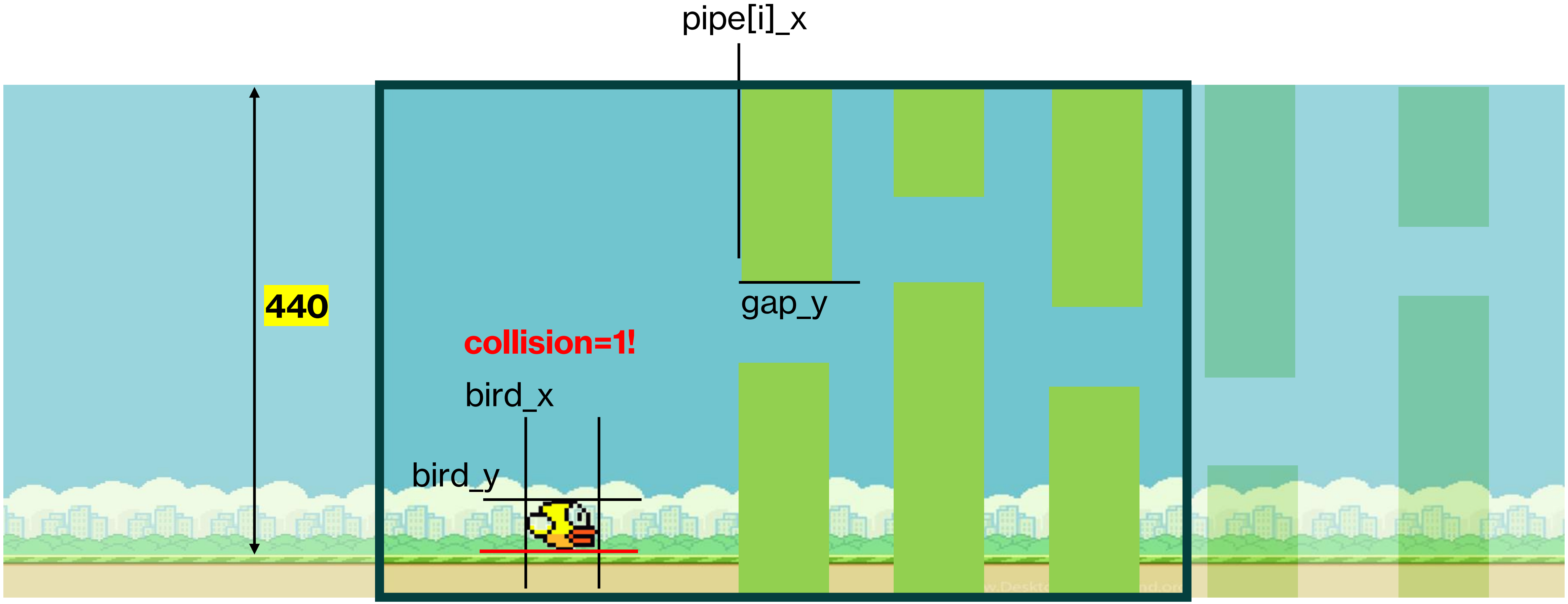
Collision Detection

```
BIRD_X + BIRD_WIDTH > pipes[i].x && BIRD_X < pipes[i].x + PIPE_WIDTH
bird_y < pipes[i].gap_y || bird_y + BIRD_HEIGHT > pipes[i].gap_y + GAP_HEIGHT
```



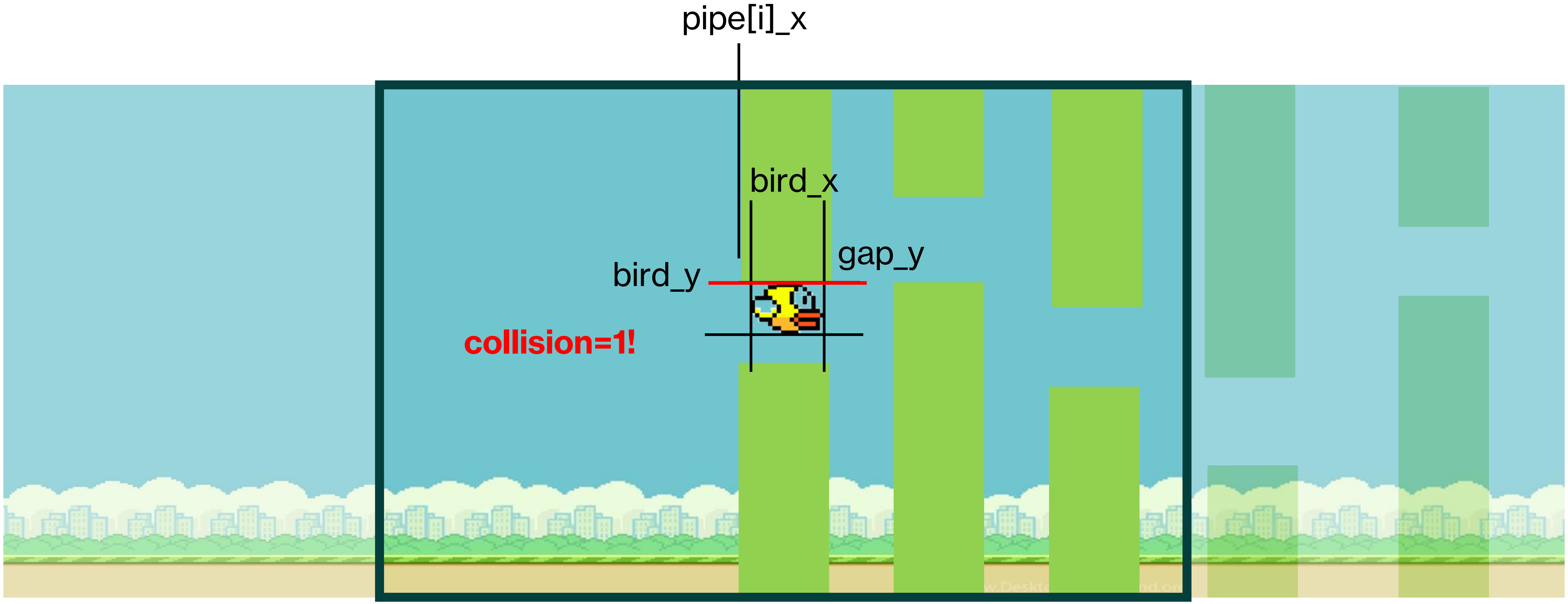
bird_x + bird_width > pipes[0].x

Collision Detection



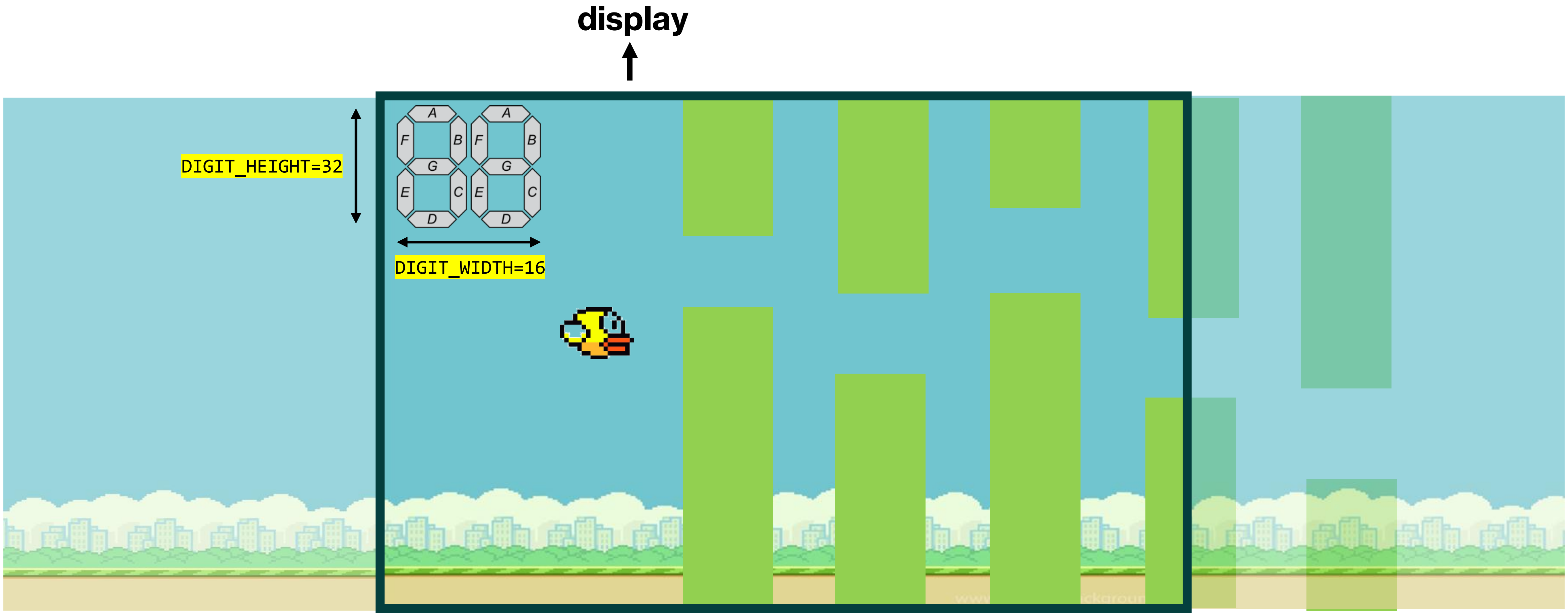
`bird_y + BIRD_HEIGHT > 440`

Collision Detection

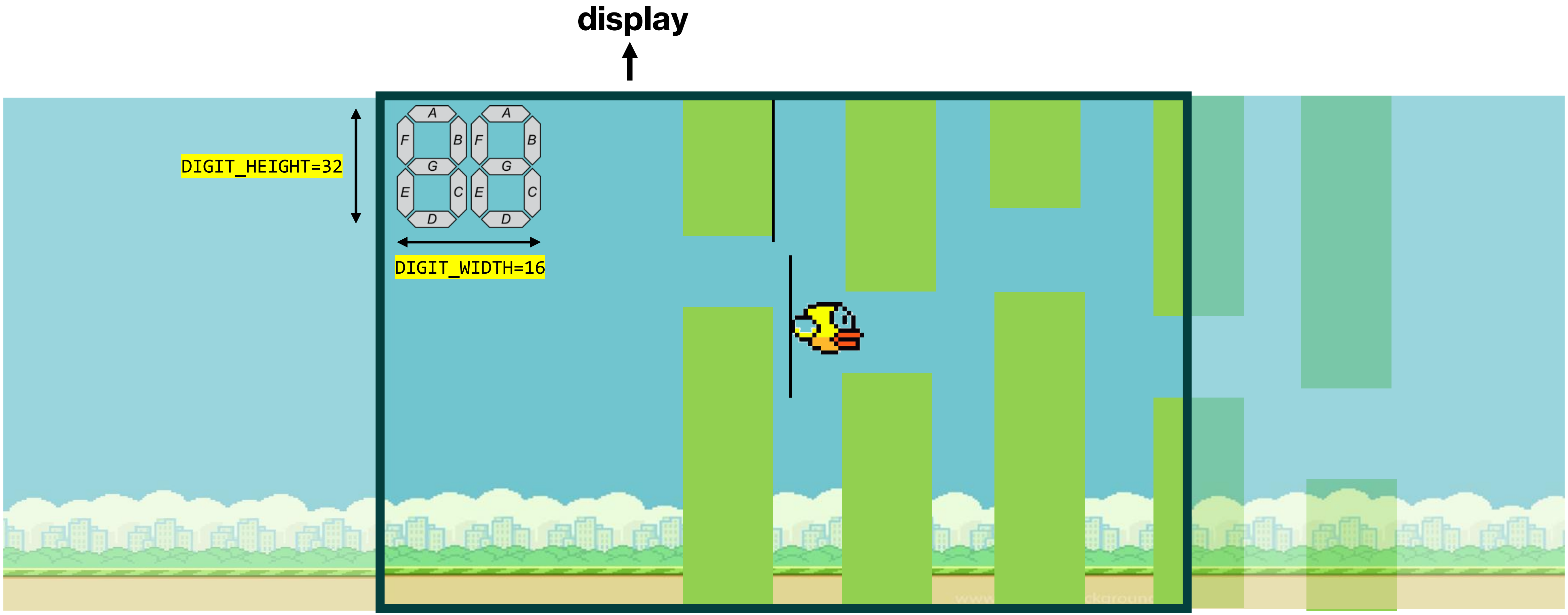


`bird_y < pipes[0].gap_y`

Score



Score



USB Keyboard

USB Keyboard Detection & Setup

- Uses libusb 1.0 to interface with USB HID devices
- `openkeyboard()` function:
  - Enumerates USB devices
  - Finds first HID device using keyboard protocol
  - Configures interface and obtains endpoint address

Application Flow

- **Key Detection**: Polls keyboard with 10ms timeout
  1. FLAP_KEY (0x2C): Spacebar triggers flap action
  2. ESC_KEY (0x29): Exits program
- **Flap Command Sequence**:
  1. Set flap bit to 1 via IOCTL (VGA_BALL_WRITE_FLAP)
  2. Wait 20ms for frame processing
  3. Reset flap signal to 0

USB Keyboard

Register Map Modification

- Register 7 repurposed from ball radius to flap signal
- Binary signal (0/1) controls flap action

Driver-Hardware Interface

- Driver function `write_flap()`:
    - Normalizes input to 0 or 1
    - Writes value to register 7 (FLAP_SIGNAL)
    - Logs action to kernel

Complete Signal Path

1. User presses spacebar
2. USB HID protocol → libusb captures keycode
3. Application detects spacebar → calls ioctl
4. Driver writes to hardware register
5. Hardware responds to flap signal
6. Bird flaps or game starts

Demo!