Criando do zero jogos para browser

Clone do Flappy Bird

Quem sou eu

- Filipe Alves
  - 4º ano graduação UNESP Rio Claro
  - Desenvolvimento de jogos
    - John Jump (Android)
  - Canal no YouTube
  - Consultoria online
  - Iniciando a pós graduação

- Mais informações, contato,
  - http://www.filipealves.com.br

Flappy Bird

- Focar em como criar um jogo do zero
  - Criar uma engine
    - inputs
    - atualizar objetos
    - desenha na tela
    - etc
  - Programar a lógica do jogo
  - Simplicidade -> tirar a complexidade da linguagem de programação
    - HTML e JS
    - Procedural
    - Sem ferramentas - Apenas um editor de texto e um navegador
      - Sublime Text e Google Chrome

- Adicionalmente (se der tempo)
  - Elementos para tornar o jogo responsivo (Rodar o jogo em um dispositivo móvel)

Interação

- Dúvidas
- Perguntas
- Sugestões no código

O projeto em si

HTML - HyperText Markup Language

- Uma forma de marcar o documento
  - Criar o HTML
  - F12 para abrir o console
  - Documentos com links e multimídia (sons e imagens)
  - Formado por tags
    - Organizado de forma hierárquica
    - Navegador cria o DOM (Document Object Model)

Javascript

- Linguagem de programação de uso geral
  - Usaremos para controlar o DOM e o jogo em si.

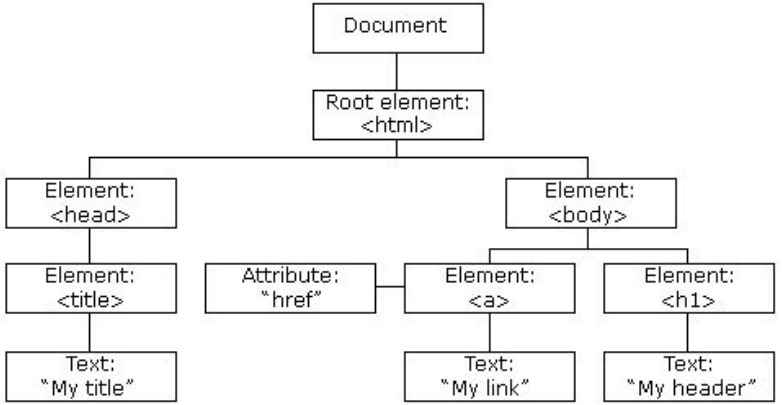
- Como criamos código JavaScript no HTML?
  - <script> Código JS </script>
  - <script link=”js/codigo.js”></script>

- HelloWorld
  - alert(“Hello World!”)

Desenhar na tela

- Elemento Canvas
  - Como se fosse um quadro
    - Desenhar formas 2D programaticamente
      - Retângulos
      - Círculos
      - Imagens
      - Textos
    - Controle dos objetos
      - Mudar posição
      - Mudar tamanho
      - Atualizar a cor
      - etc
  - Criar a canvas
    - <canvas></canvas>

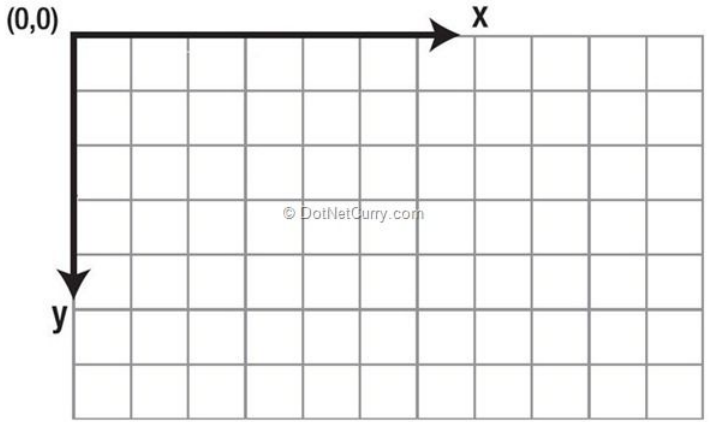
Desenhar na tela



- Desenhos programáticos (Com JS)
  - Acessar a canvas com JS
  - Variável **document**
    - Acessa o DOM
  - Função nativa
    - **document.getElementById("string")**
    - Retorna o elemento
      - Mudar suas propriedades
  - Elementos têm **atributos**
    - atributo="valor"
  - Ver as propriedades do objeto canvas utilizando o console
    - **var ctx = canvas.getContext("2d")**
    - **ctx.fillRect(0, 0, 10, 10)**
    - **ctx.clearRect(0, 0, 5, 5)**

Desenhar na Tela (Dinâmico)

- Loop infinito
  - Limpar a tela
  - Desenhar objeto em uma nova posição
  - Orientação dos eixos
  - funções
    - update
    - draw
  - Função **run**
    - **window.requestAnimationFrame(run)**



Inputs

- Teclado
  - **window.addEventListener(“keydown”, function(e) {**
    console.log(e)
    })
  - KeyCodes
  - Criar objeto literal

```
var key = {
        W: 87,
        A: 65,
        S:  83,
        D: 68
}
```

  - switch dentro da addEventListener
  - criar **var keys**
    - keydown e keyup
    - keys[e.keyCode] = true | false;

Input

- Mouse
  - window.addEventListener(“click”, function(e) {})

Ajustando a canvas

- canvas.width = 432
  canvas.height = 768
- Podemos usar CSS para alterar a posição da canvas na janela.
  - <style></style>
  - Seleciona elementos e muda suas propriedades gráficas
    - position: absolute
    - top: 0;
    - bottom: 0;
    - left: 0;
    - right: 0;
    - margin: auto;
    - border: 1px solid black;

Desenhar imagens

- Carregar o recurso da imagem
  - var spritesheet = new Image();
  - spritesheet.src = “resources/sheet.png”
- Desenhar a imagem usando
  - ctx.drawImage(image, xImg, yImg, width, height, xCanvas, yCanvas, scaleX, scaleY)
- Desenhar o fundo
  - ctx.drawImage(spritesheet, 0, 0, 432, 768, 0, 0, 432, 768)

SpriteSheet

- Todas imagens do jogo dentro de uma única imagem
  - Carregamos apenas uma imagem para a memória
  - Se tivéssemos que carregar as imagens uma a uma haveria desperdício de memória (uma imagem é mais leve que várias imagens)
  - Precisamos mapear a imagens em Sprites
    - Arquivo sprite.js
  - Função que desenha uma sprite em um dado x e y da canvas
    - **function drawSprite(sprite)**

Lógica do jogo

- Desenhar o fundo
  - drawSprite(bg_spr, 0, 0)

- Desenhar e animar o chão
  - var groundX = 0
  - var scrollSpeed = 2
  - drawSprite(ground_spr, ground_x, canvas.height - ground_spr[3])
  - groundX -= scrollSpeed
  - if (groundX <= -21)
    groundX = 0

Desenhar e animar o passarinho

- Objeto literal
  - ```
    var bird = {
        frame: 0,
        x: 80,
        y: 300
    };
    ```
  - ```
    if (bird.frame == bird_spr.length - 1)
        bird.frame = 0
    else
        bird.frame++
    ```
  - ```
    drawSprite(bird_spr[bird.frame], bird.x, bird.y)
    ```
    ```

Problema com o tempo da animação

- Variável para controlar a quantidade de frames que foram renderizados
  - Possibilidade de controlar o tempo da animação
  - Global
    - **var frames = 0**
  - Na função draw
    - **frames++**

Animação do passarinho

- Trocar o frame a cada 0.2s
  - O jogo roda a 60 FPS, então devemos trocar o frame do passarinho a cada:

    60 frames ---- 1s
    12 frames ---- 0.2s

  - **if (frames % 12 == 0) { muda o frame do passarinho... }**
    - Função mod (%) resto da divisão
    - Verifica se 12 frames foram renderizados (se passou 0.2s desde a última atualização)

- Atualizar posição vertical do passarinho se o jogo estiver no modo de espera
  - **bird.y += Math.cos(frames / 10)**

Estados do jogo

- Espera
- Jogando
- Game Over

- Criar estados
  - **var gameStates = {**
    waiting: 0,
    playing: 1,
    gameOver: 2
    }
    var currentState = gameStates.waiting;
  - Verificar se o estado atual é o de espera na função update para mudar o y do passarinho
  - Verificar se o estado atual é o de espera na função draw para desenhar o título do jogo
  - Verificar se o estado atual é o de espera na função clique para mudar para o estado jogando

Estado jogando

- Aplicar física no passarinho
- Clique faz o passarinho pular
- Inserir e atualizar canos
- Verificar colisões com os canos e chão
- Incrementar o score

Um pouco de física

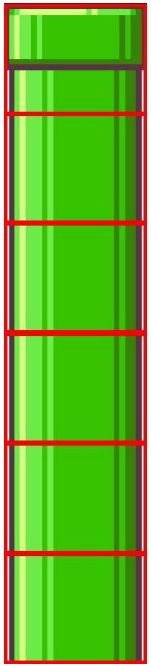
- Gravidade
  - Força constante somada a velocidade vertical de um corpo.
  - Criar velocidade vertical do passarinho
    - vy
  - Criar gravidade do jogo
    - var gravity = 1
  - Aplicar forças ao passarinho se o modo for diferente de **Waiting**
    - bird.vy += gravity
    - bird.y += bird.vy
- Pulo
  - Criar a força do pulo do passarinho
    - jump: 8
  - Se estiver no estado **Playing,** a velocidade vertical do passarinho passa a ser a -força do pulo
    - bird.vy = -bird.jump

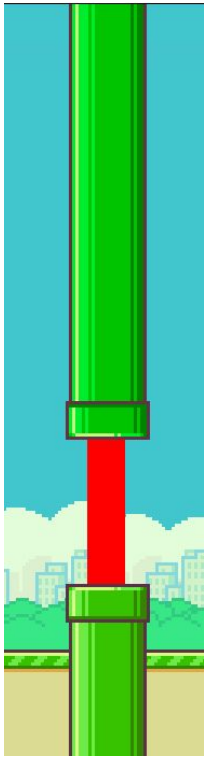
Obstáculos

- Estruturado
  - Um obstáculo pode ser um vetor
    - coordenada x
    - quantidade de “blocos” do cano inferior
    - gap de 134px entre o cano superior e inferior

  - ```
    var pipes = [];
    function createPipe() {
            var yPipe = canvas.height - ground_spr[3] -
            Math.ceil(Math.random() * 6) * pipe_down_spr[3] / 6;
            pipes.push([canvas.width, Math.ceil(Math.random() * 6)])
    }
    ```

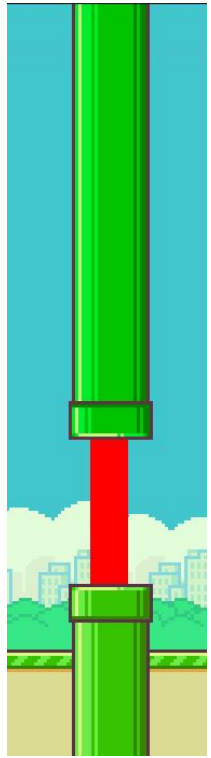
  - Criar obstáculos a cada 2 segundos
    - ```
      if (frames % 120 == 0)
              createPipe();
      ```





Obstáculos

- Desenhar pipes
  - **function drawPipes() {**
    for (var i = 0; i < pipes.length; i++) {
    var pipe = pipes[i];
    drawSprite(pipe_down_spr, pipe[0], pipe[1])
    drawSprite(pipe_up_spr, pipe[0], pipe[1] - pipe_up_spr[3] - 134)
    }
    }
  - Na função draw chamar **drawPipes()**
- Atualizar obstáculos
  - Na função update, dentro do **if** que verifica se o estado é o jogando
    - **for (var i = 0; i < pipes.length; i++) {**
      var pipe = pipe[i];
      pipe[0] -= scrollSpeed;
      }



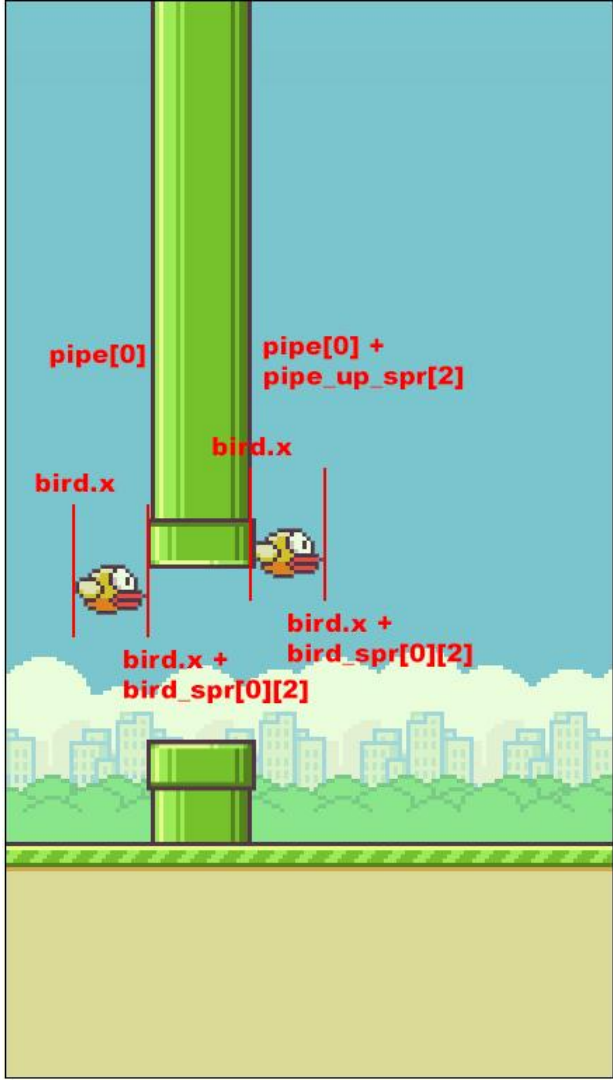
Obstáculos

- Excluir obstáculos que não são mais visíveis
  - Evitar ter que atualizar muitos obstáculos com o passar do tempo no jogo
  - **if (pipe[0] <= -pipe_up_spr[2]) {**
    pipes.splice(i, 1)
    i--
    }

Colisões

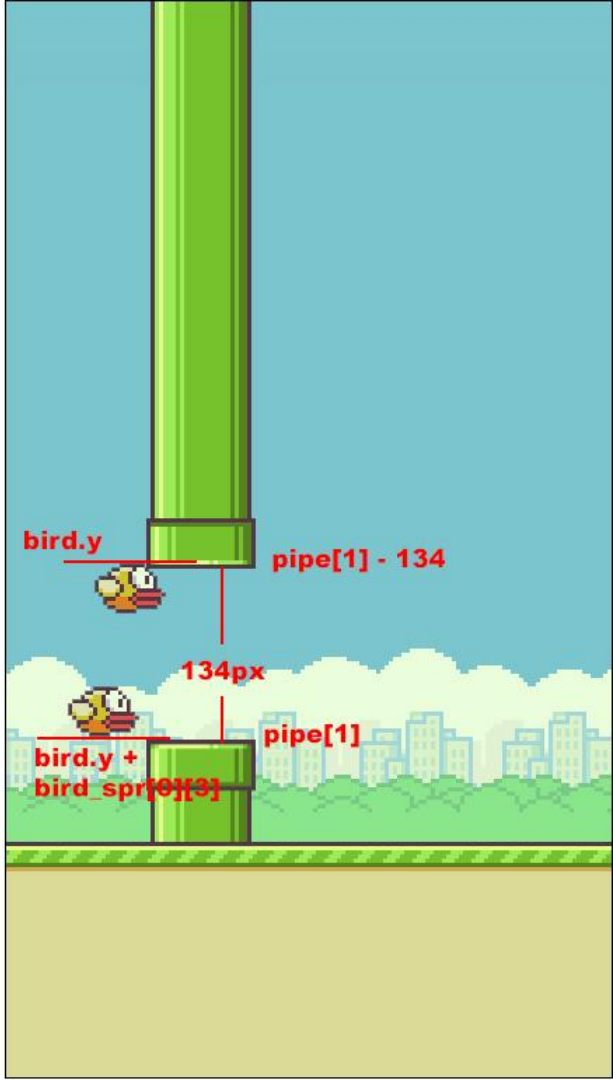
- Chão
  - `if (bird.y + bird_spr[0][2] >= canvas.height - ground_spr[3]) {`
    `currentState = gameStates.gameOver`
    `}`
- Canos
  - Colisão em x

    `if (pipe[0] <= bird.x && pipe[0] + pipe_up_spr[2] >= bird.x) {`
    `console.log("Colisão em x")`
    `}`



Colisões

- Canos
  - if (bird.y + bird_spr[0][3] >= pipe[1] || bird.y < pipe[1] - 134) {
    currentState = gameStates.gameOver
    }



Ajustes

- Atualizar o chão nos estados de espera e jogando
  - **if (currentState != gameStates.gameOver)**

Pontuação

- Criar variável **score**
  - **var score = 0**
- Adicionar novo índice no pipe, representando se a pontuação já foi contabilizada. Começa com **false** sempre
- Na condição de colisão em x
  - **else if (bird.x >= pipe[0] + pipe_up_spr[2] && !pipe[2]) {**
    score++;
    pipe[2] = true;
    console.log(score)
    }

Desenhar label do score

- Converter o score para string
  - Contar quantos caracteres nosso score possui
  - Converter cada caracter para número
    - Acessar o índice correto da sprite no **num_spr**

- Posição inicial em x do primeiro caracter do score

```
var scoreString = score.toString();
var xi = canvas.width / 2 - scoreString.length * num[0][2] / 2;
for (var i = 0; i < scoreString.length; i++) {
        var num = parseInt(scoreString[i])
        drawSprite(num_spr[num], xi, i * num_spr[i][2], 30)
}
```

- Chamar **drawScore()** na função **draw**

Estado de Game Over

- Criar função **drawGameOver()**
- Centralizar o scoreboard na janela
  - **drawSprite(scoreboard_spr, canvas.width / 2 - scoreboard_spr[2] / 2, canvas.height / 2 - scoreboard_spr[3] / 2);**
- Label de Game Over
  - **drawSprite(game_over_spr, canvas.width / 2 - game_over_spr[2] / 2, canvas.height / 2 - game_over_spr[3] - scoreboard_spr[3] / 2 - 30);**
- Desenhar moedas de acordo com a pontuação
  - **var coin = Math.min(3, Math.floor(score / 10));**
  - **drawSprite(coins_spr[coin], canvas.width / 2 - 2 * coins_spr[coin][2], canvas.height / 2 - coins_spr[coin][3] / 2 + 10)**

Estado de Game Over

- Desenhar score atual
  - ```
    var scoreString = score.toString();
    for (var i = 0; i < scoreString.length; i++) {
        var num = parseInt(scoreString[i]);
        drawSprite(small_nums_spr[num], canvas.width / 2 + scoreboard_spr[2] / 4 + small_nums_spr[i][2] * i,
    canvas.height / 2 - 35)
    }
    ```
- Desenhar a melhor pontuação
    ```

Melhor pontuação

- Score atual superar a melhor pontuação precisamos armazenar isto de alguma forma
  - **window.localStorage**
    - **getItem(“chave”)**
    - **setItem(“chave”, valor)**
- Quando iniciarmos mudarmos para **playing**:
  - ```
    bestScore = localStorage.getItem(“bestScore”)
    if (bestScore == null) {
        localStorage.setItem(“bestScore”, 0)
        bestScore = 0;
    }
    ```
- Quando ocorrer colisão:
  - ```
    if (score > bestScore) {
        localStorage.setItem(“bestScore”, score)
    }
    ```

Estado Game Over

- Desenha a melhor pontuação
  - ```
    var scoreString = bestScore.toString();
    for (var i = 0; i < scoreString.length; i++) {
            var num = parseInt(scoreString[i]);
            drawSprite(small_nums_spr[num], canvas.width / 2 + scoreboard_spr[2] / 4 + small_nums_spr[i][2] * i,
    canvas.height / 2 + 30)
    }
    ```

- Reiniciar o jogo quando clicar
  - Zerar o score
  - Posição, velocidade do passarinho
  - limpar vetor de obstáculos
  - Ir para estado **waiting**
  - setTimeout para habilitar cliques
    ```

Contato

Contatos: http://www.filipealves.com.br/contato