Animaciónes procedurales

Daniel Kauss Serna,
parte de chipburners

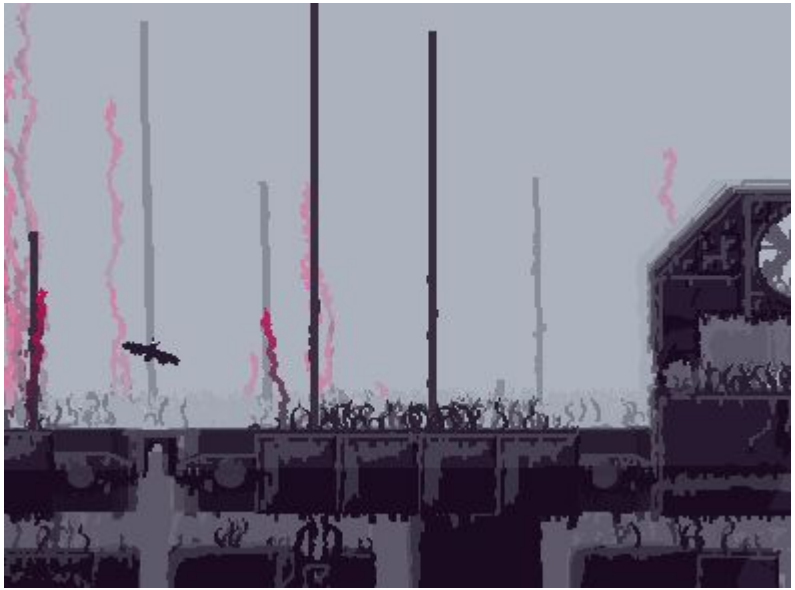
Definición

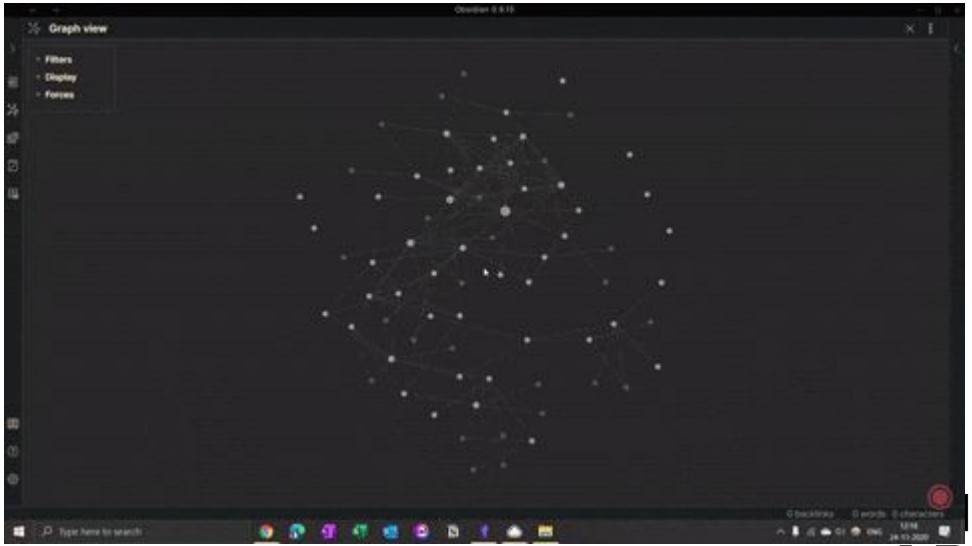
Movimientos definidos por código

Nosotros generamos también los gráficos con código

CH
IP

Ejemplos reales





Introduccón a p5.js

Dibujar mediante:

circle(x, y, r), line(x1, y1, x2, y2)

Estilo con:

noFill(), fill(color), noStroke(), stroke(color)

Transformaciones del canvas:

translate(x, y), rotate(angle)

CH
IP

Introduccón a p5.js

p5.Vector:

add, sub, mult, div

mag, heading, setMag, setHeading

dist, limit, normalize, lerp

estatico:

p5.Vector.func()

in place:

miVectror.func()

usaremos el alias

vec

para referimos a p5.Vector

CH
IP

Recursos

Todos los enlaces en: chipburners.club/e

editor online: editor.p5js.org

local: p5js.org/download/

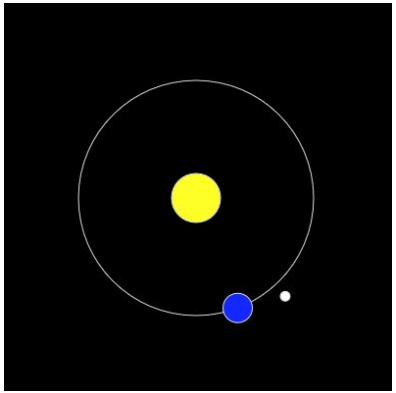
CH
IP

Animaciones controladas con el tiempo

Todos los movimientos dependen de un valor de tiempo transcurrido

CH
IP

Sistema solar



Sol central, tierra orbitando, luna orbitando tierra

Métodos de p5.js relacionados:

circle(x, y, r), fill(color), noFill(), stroke(color), noStroke(), translate(x, y), push() / pop(), millis()

tiempo usable para animaciones:

millis() / 1000

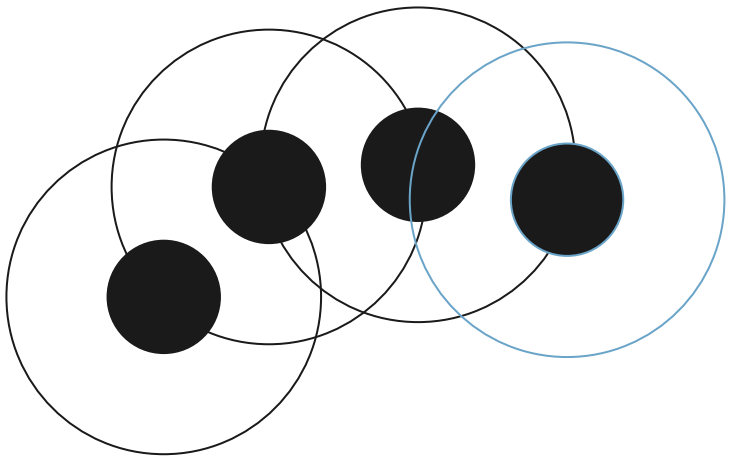
CH
IP

Cadena fijada

Cada punto de la cadena está a una distancia fija al previo

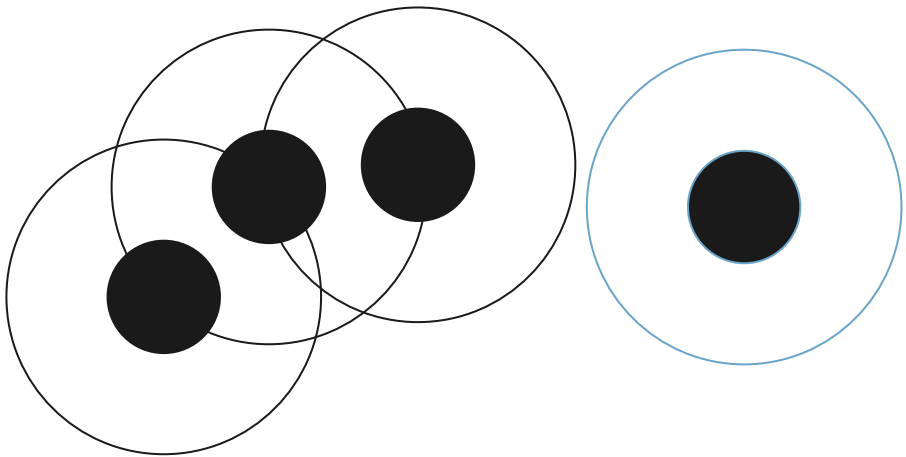
Se propaga desde el principio al final, en cada paso fijando la distancia

Cadena fijada

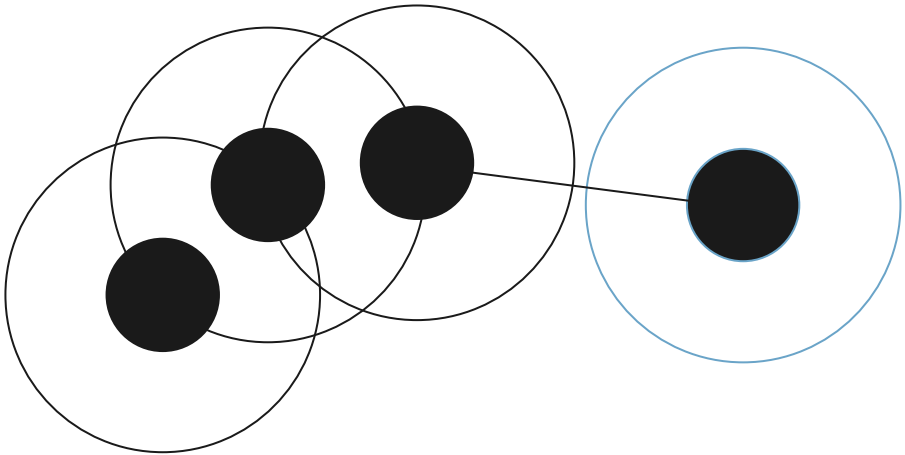


CH
IP

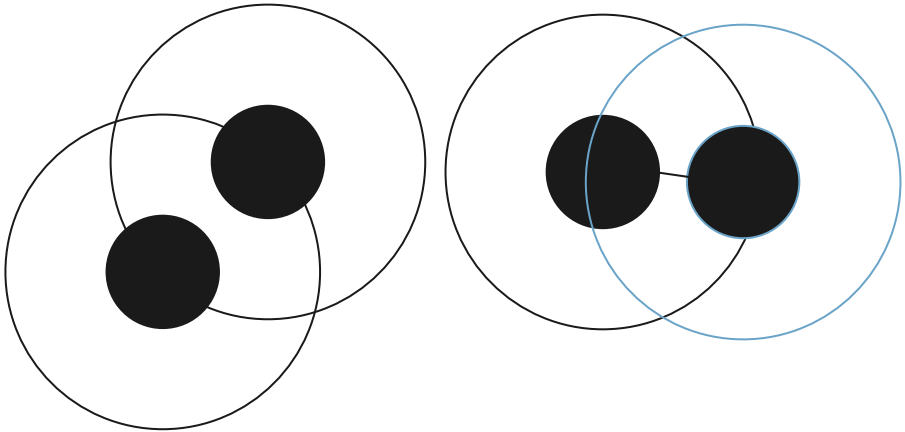
Cadena fijada



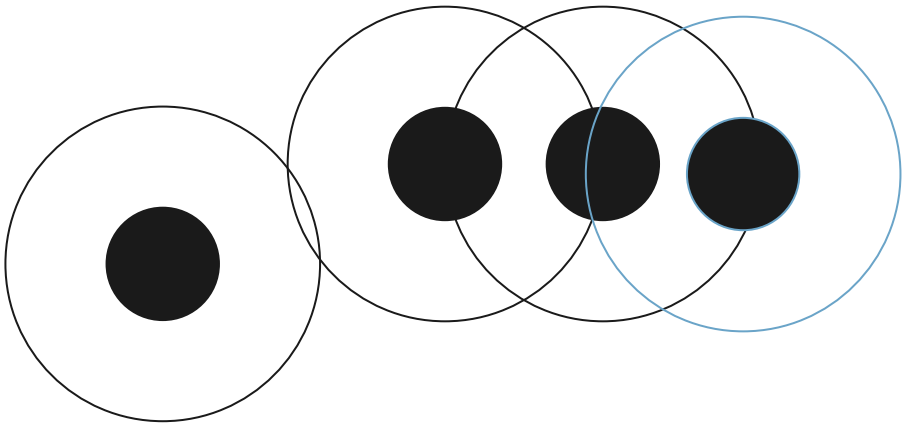
Cadena fijada



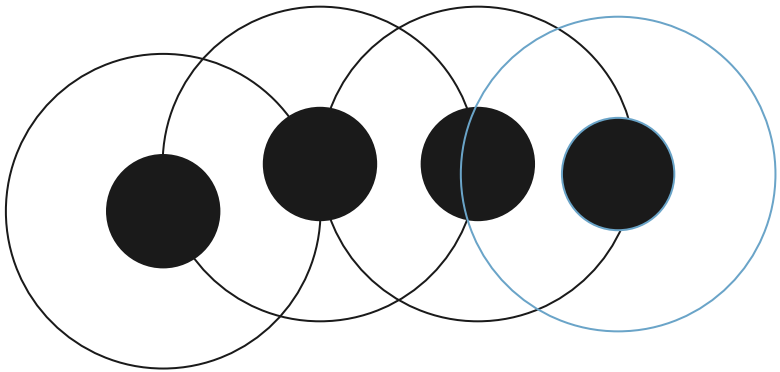
Cadena fijada



Cadena fijada



Cadena fijada



CH
IP

Implementación

Clase que contiene posición, distancia, y una función follow(targetX, targetY) que:

1. Vector que apunta de target a pos
2. Limitar a dist
3. Mover

Métodos relacionados de p5.js:

vec.add, vec.sub, vec.dist, vec.heading, vec.limit, vec.normalize, vec.mag, vec.setMag

CH
IP

Solución

```
class ConstraintPoint {
  constructor(x, y, size) {
    this.size = size;
    this.pos = new vec(x, y);
    this.forward = new vec(0, 0);
  }

  follow(targetX, targetY, parentForward = null) {
    let target = new vec(targetX, targetY);
    let targetForward = vec.sub(target, this.pos);
    let targetAngle = targetForward.heading();

    this.forward = p5.Vector.fromAngle(targetAngle);
```

Solución

```
let points = []
function setup() {
  createCanvas(400, 400);

  for (let i = 0;i < 10;i++) {
    points.push(new ConstraintPoint(0, 0, 20));
  }
}
```

```
function draw() {
  background(220);

  points[0].follow(mouseX, mouseY)
  for (let i = 1;i < 10;i++) {
    let prev = points[i - 1];
    points[i].follow(prev.pos.x, prev.pos.y);
  }
  for (let i = 1;i < 10;i++) {
    let pos = points[i].pos;
    noFill();
    circle(pos.x, pos.y, 40);
  }
}
```

CH
IP

Restricciones de ángulo

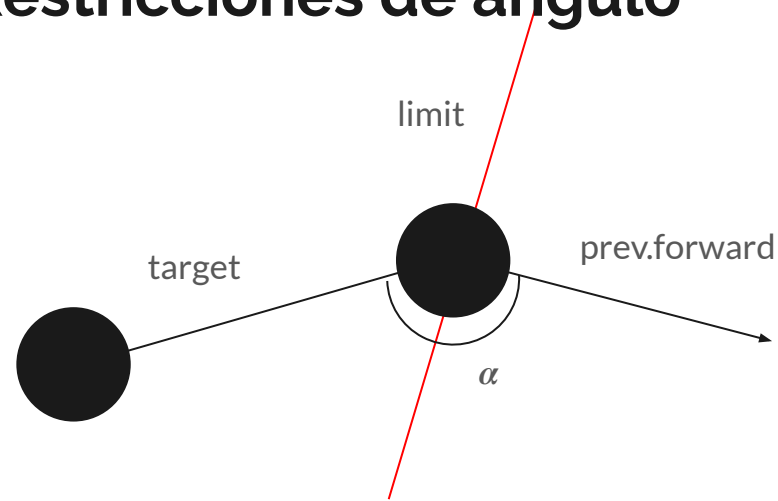
Queremos restringir el ángulo entre segmentos, para que los puntos no pueden intersectarse consigo mismo

Cada punto tendrá definido la dirección hacia delante, y con eso limitamos el ángulo

En el final de follow, definir forward del punto, y pasarlo como argumento

CH
IP

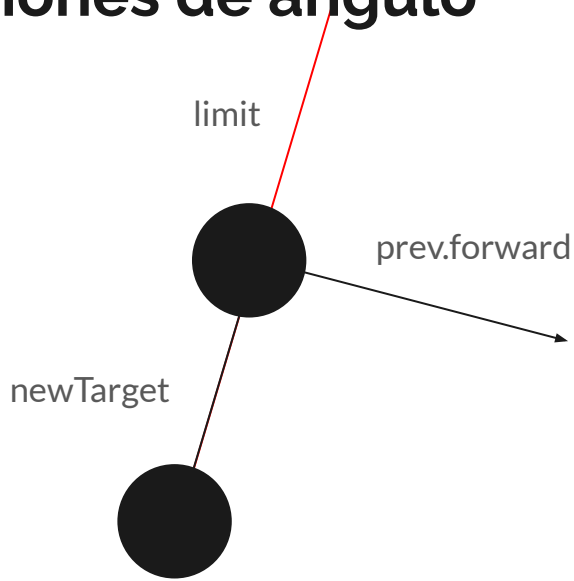
Restricciones de ángulo



α = target.angle - prev.target
limitar α a [-PI, PI]
constrain a [lowerbound, upperbound]
newTarget = fromAngle(α)

CH
IP

Restricciones de ángulo



α = target.angle - prev.target
limitar α a [-PI, PI]
constrain a [lowerbound, upperbound]
newTarget = fromAngle(prev.target + α)

CH
IP

Solución

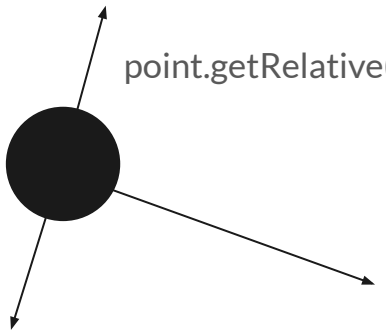
```
follow(targetX, targetY, prevForward = null) {
  let target = new vec(targetX, targetY);
  let targetForward = vec.sub(target, this.pos);
  let targetAngle = targetForward.heading();

  if (prevForward) {
    let a = targetAngle - prevForward;
    while (a > PI) { a -= TWO_PI; }
    while (a < -PI) { a += TWO_PI; }

    a = constrain(a, -0.3, 0.3);

    targetAngle = prevForward + a;
```

CH
IP

Puntos relativos

Para situar otros elementos en la base, es útil tener una función similar a getRelative(angulo, distancia)

en la clase puntos, que devuelve un punto relativo a forward con un ángulo y una distancia



CH
IP

Dibujar cuerpo

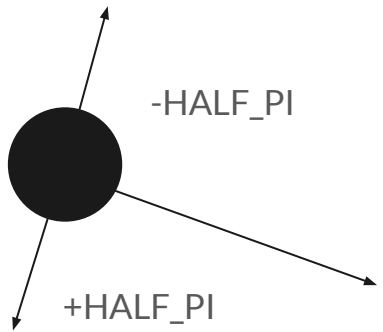
Para dibujar el cuerpo podemos crear un polígono, con los puntos laterales de cada circulo, y los semicírculos del inicio y final

En p5.js podemos crear un polígono cualquiera con:

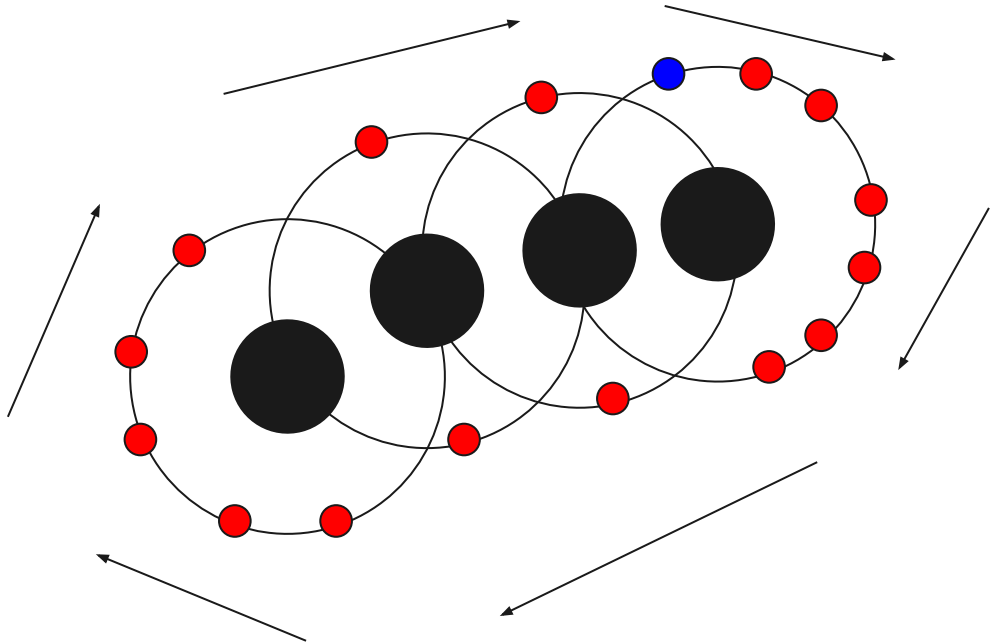
```
beginShape();

vertex(x, y);

...

vertex(x100, y100);

endShape
```

Dibujar cuerpo

Los laterales son los perpendiculares a forward, como visto antes en puntos relativos



Dibujar cuerpo



CH
IP

Dibujar ojos

Simplemente dibujamos dos circulos negros relativos a la cabeza, en angulos +-PI/4

CH
IP

Implementación

función drawPoints(points, sizes), que crea un polígono alrededor de los puntos

Le pasamos un array sizes, que determina la distancia a cada punto de los laterales

Es recomendado añadir un método a la clase de puntos getRelative(angle, dist) que devuelve otro punto relativo dado un angulo y una distancia, eg: getRelative(HALF_PI, 30) para el lado derecho

Métodos y constantes relacionadas de p5.js:

vec.add, vec.heading, beginShape, endShape, vertex, HALF_PI, PI

CH
IP

Inverse kinematics

Inverse kinematics nos deja fijar varios puntos en un principio y final

Usaremos el método de FABRIK

Forward and backward reaching inverse kinematic

CH
IP

Inverse kinematics

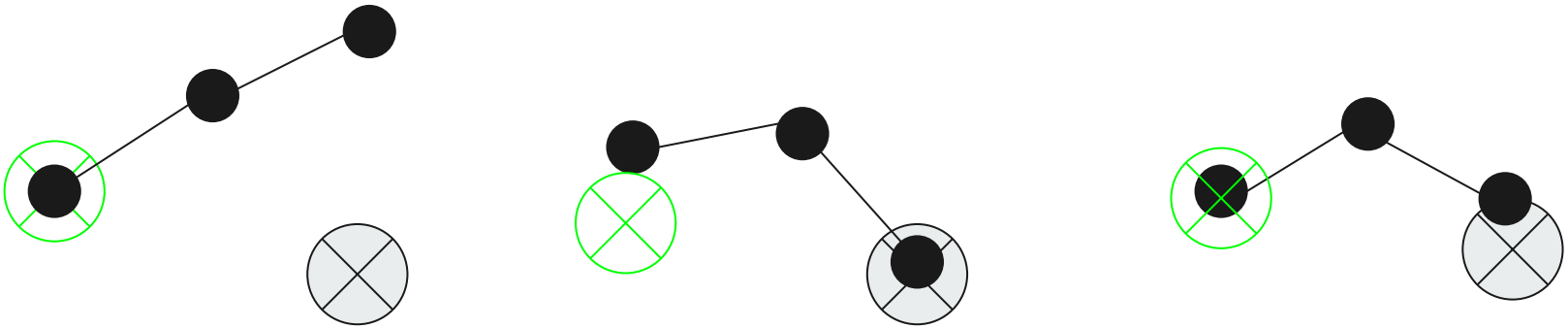
Simplemente hay que mover el brazo hacia el destino igual que antes al ratón,

y luego en orden al revez moverlo hacia el origen.

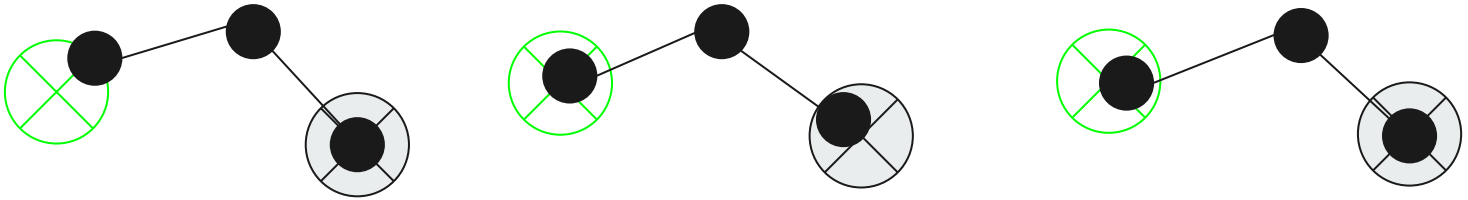
Esto se repite varias veces (~15) aproximándose cada vez más

CH
IP

Inverse kinematics



Inverse kinematics



CH
IP

Implementación

Crear una clase IKJoint, que contiene al menos una funcion reach(originX, originY, targetX, targetY)

1. Mover cadena hacia el final
2. Mover cadena hacia el principio, propagando follow en dirección opuesta
3. Repetir ~15 veces

Opcional:

La restricción de ángulos se puede usar aquí también, pero SOLO en el forward pass

CH
IP

Extremidades

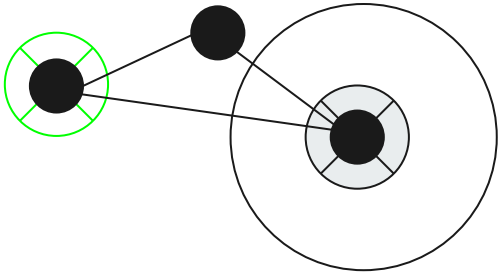
Para las piernas usamos los joints de inverse kinematics previos

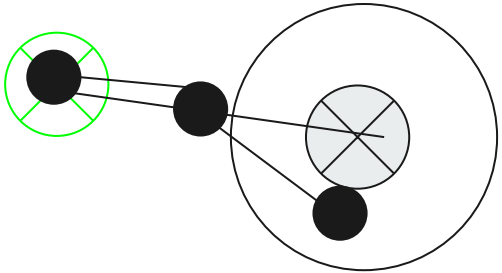
El origen será un punto relativo al cuero dentro de el, y el target será otro punto relativo fuera del cuerpo

Para que parezca que este andando, solamente actualizamos el target cuando el final del IKjoint se aleja demasiado

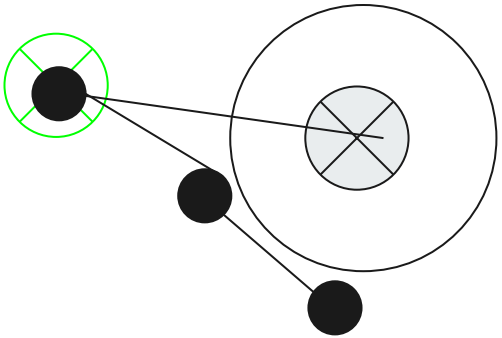
CH
IP

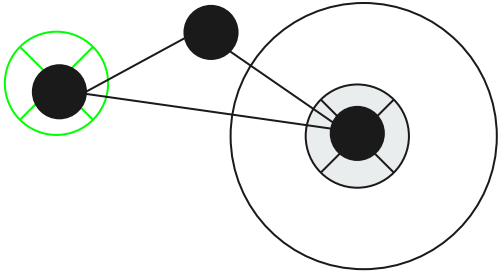
Extremidades





Extremidades





Implementación

Una clase Extremities, con una función moveTo(originX, originY, endX, endY) la cual

- Siempre mueve la base de IKjoint
- Si la distancia entre end y el final del joint es muy grande, mover IKjoint al nuevo end

También hace falta una función getPoints() que devuelve todos los puntos del joint para poder dibujarlo

CH
IP

Atar extremidades al cuerpo

Simplemente creamos 4 extremidades, y cada draw llamamos a moveTo con una base y un final, q seran relativos al cuerpo

Es recomendado que tengan unos 8 segmentos de tamaño 8

Para dibujar usaremos la misma función draw Points, con un array similar a

```
const legSizes = Array.from({ length: 20 }, () => 15);
```

CH
IP

Interpolación

Para que las extremidades no salten instantáneamente entre movimientos, podemos interpolar su posición final. Creamos un vector interpolationEnd, que se actualiza fácilmente con

interpolationEnd.lerp(endX, endY, 0.4)

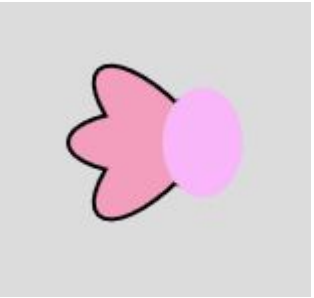
CH
IP

Dibujar branquias

Las branquias están compuestas por 3 ellipses giradas, dibujados una vez con outline y otra vez sin, y un círculo en la base para cortar el outline

CH
IP

Dibujar branquias



Posicionar branquias

Cogemos un punto relativo a un segmento cerca de la cabeza, y hacemos que las branquias giren en función del forward de ese punto.

CH
IP

Animación respirar

Copiamos el array de bodySizes, y usando una función de peso incrementamos ciclicamente los elementos centrales del cuerpo

Animación cola

Después de cada propagación de constraints del cuerpo, aplicamos un offset al punto lateralmente a su forward, ciclicamente

CH
IP

Muchas gracias!

Este taller forma parte de chipburners

Somos un hackerspace en formación, con el objetivo de juntar gente con intereses similares y compartir conocimientos.

Mas informacion en chipburners.club