UTN

Proyecto de investigación

Redes neuronales y Algoritmos Genéticos en la Era de los Videojuegos

Bartolucci Gino ginobartolucci00@gmail.com

Joaquín Bates betesjoaquin@gmail.com

Francisco Mendiburu franmendi.fm@gmail.com

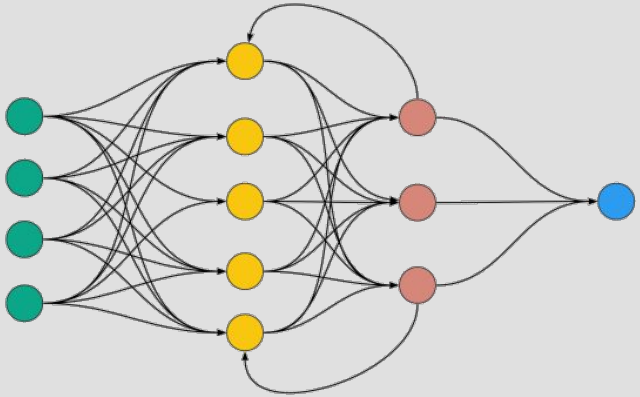
Introducción

Intersección de campos

Aprendizaje Automático
&
Algoritmos Genéticos

Francisco Mendiburu

Introducción

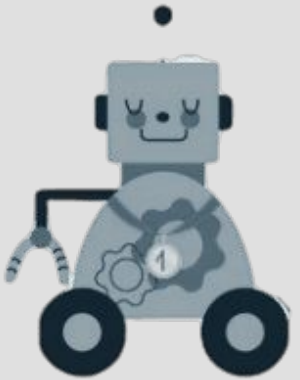


Redes neuronales

Breakout

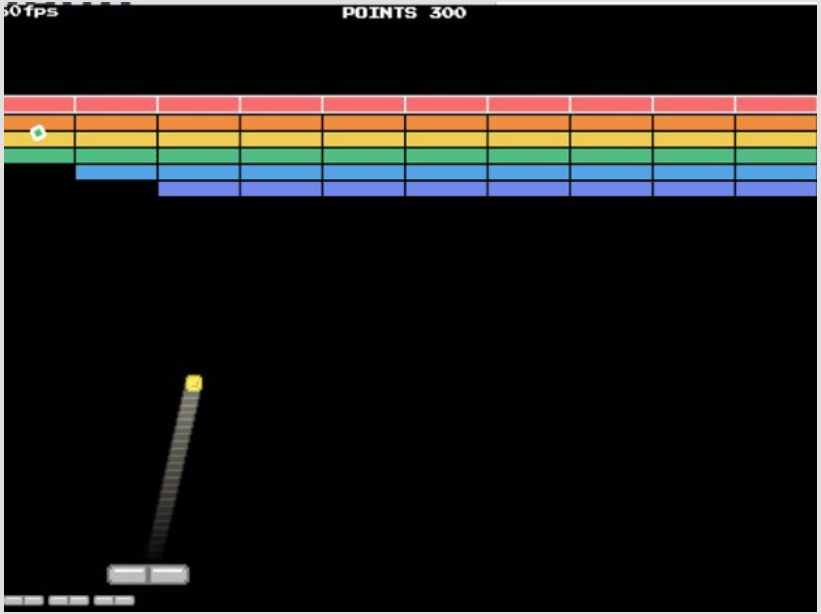
Francisco Mendiburu

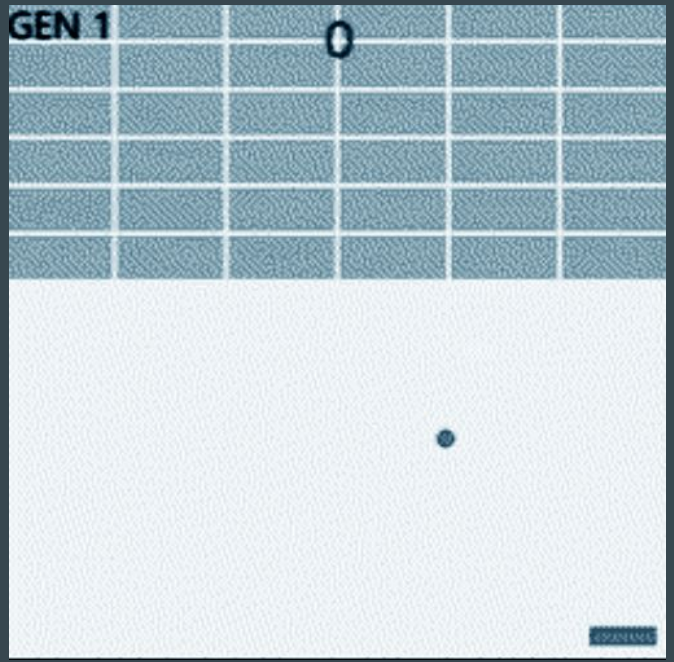
Objetivo del proyecto: creación de agentes IA's que superen el rendimiento humano en el juego.



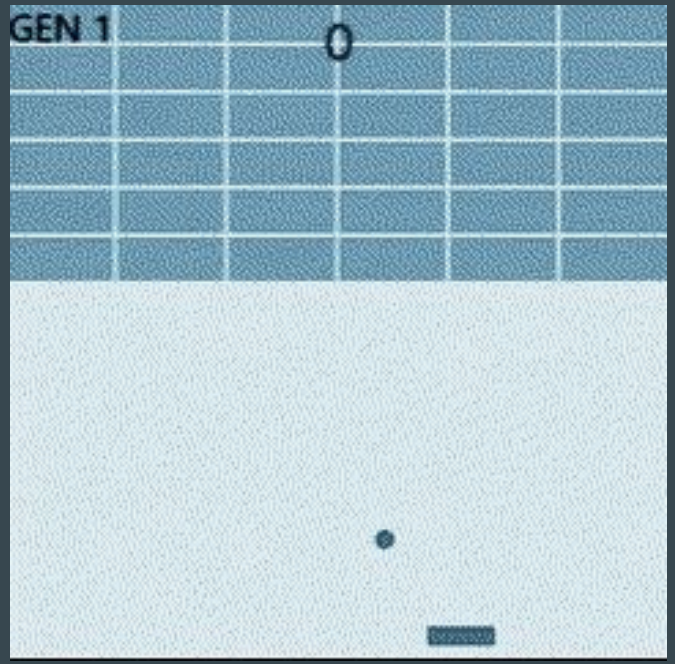
El juego

¿Crear o usar?





Francisco Mendiburu



El juego

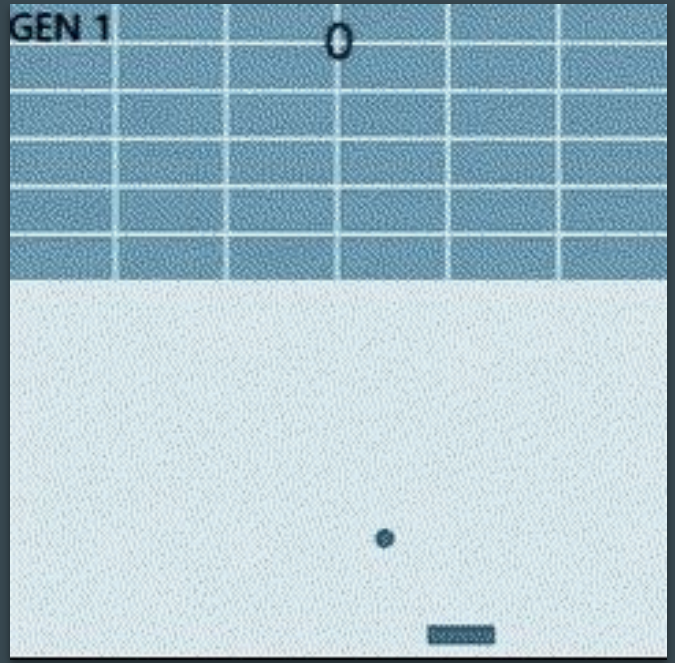
REGLAS DEL JUEGO

+ 1 punto por **bloque destruido**

Gana al destruir todos los bloques

Game over si la pelota cae

Francisco Mendiburu



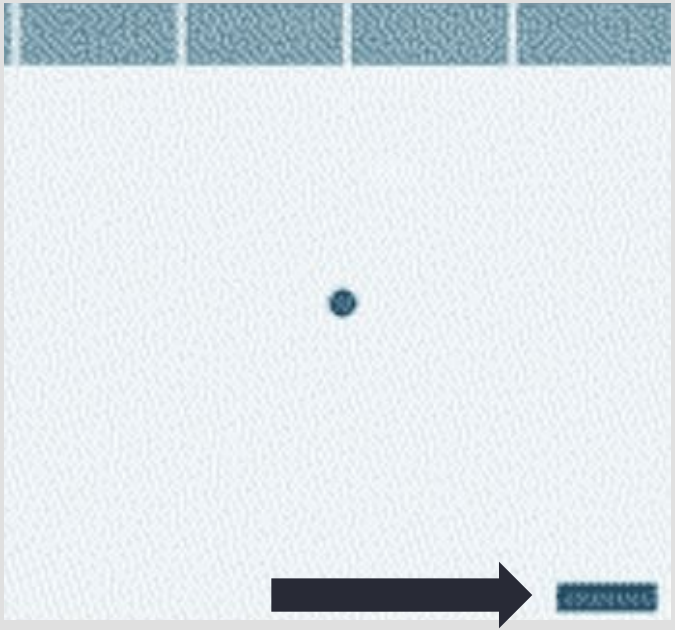
El juego

CONTROLES

- <- Moverse a la izquierda
- -> Moverse a la derecha

Francisco Mendiburu

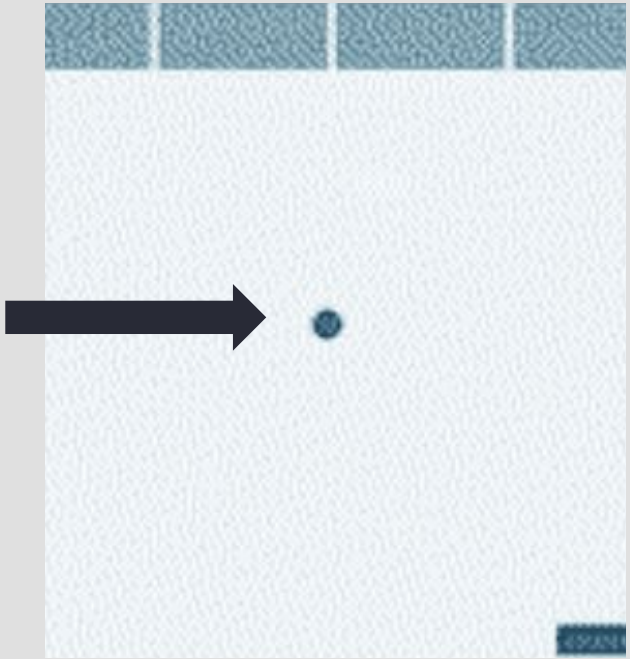
Paleta



```python
class Paddle:
    HEIGHT = 20
    WIDTH = 70
    SPEED = 8
    RED = (242, 85, 96)
    BLACK_RED = (220, 60, 70)

    def __init__(self, x, y):
        self.original_x = x
        self.original_y = y
        self.rect = Rect(x, y, self.WIDTH, self.HEIGHT)

    def move(self, left):
        if left:
            self.rect.x -= self.SPEED
        elif not left:
            self.rect.x += self.SPEED

    def draw(self, window):
        pygame.draw.rect(window, self.RED, self.rect)
        pygame.draw.rect(window, self.BLACK_RED, self.rect, 3)

    def reset(self):
        self.rect.x = self.original_x
        self.rect.y = self.original_y
```

Francisco Mendiburu

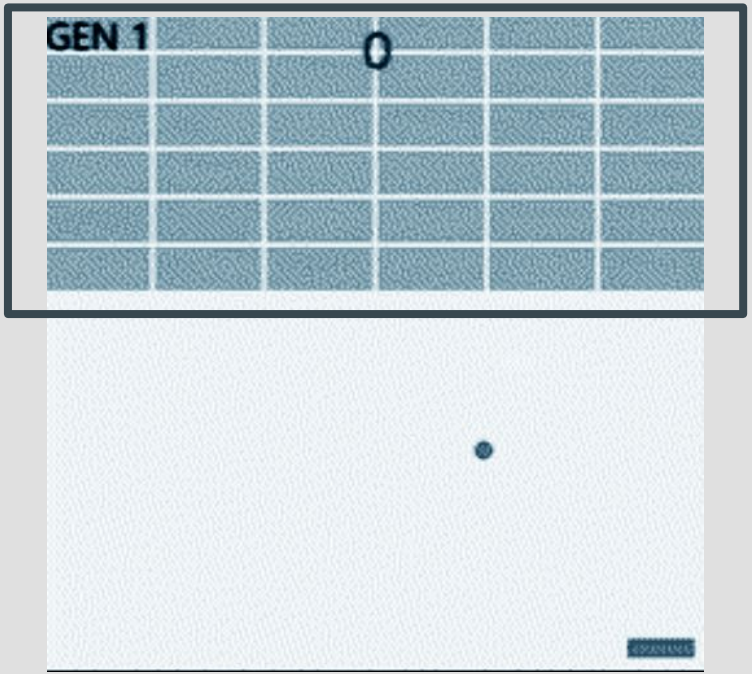
Pelota



```python
class Ball:
    INICIAL_VEL = 6
    MAX_VEL = 11
    RADIUS = 10
    # VARIABLES sobre estilos

    def __init__(self, x, y):
        self.original_x, self.original_y = (x - self.RADIUS,y)
        self.rect = Rect(self.original_x, self.original_y,
                         self.RADIUS * 2, self.RADIUS * 2)
        self._inicial_vel()

    def _get_random_angle(self, min_angle, max_angle, excluded):
        angle = 0
        while angle in excluded:
            angle = math.radians(random.randrange(min_angle, max_angle, 1))
        return angle

    def _inicial_vel(self):
        angle = self._get_random_angle(30, 75, [0, 44, 45, 46])
        pos = 1 if random.random() < 0.5 else -1
        self.vel_y = - abs(math.sin(angle) * self.INICIAL_VEL)
        self.vel_x = pos * math.cos(angle) * self.INICIAL_VEL

    def increase_vel(self, increment):
        if self.MAX_VEL >= math.sqrt(self.vel_x**2 + self.vel_y**2):
            self.vel_x *= increment
            self.vel_y *= increment

    def reset(self):
        self.rect.x,self.rect.y  = (self.original_x, self.original_y)
        self._inicial_vel()

    def move(self):
        self.rect.x += self.vel_x
        self.rect.y += self.vel_y

    def draw(self, window):
      # Para dibujar la pelota en la pantalla
```

Francisco Mendiburu

Pared




```python
class Wall():
    BLUE = (69, 160, 215)

    def __init__(self, window_width, rows, cols, bg_color):
        self.width = window_width // cols
        self.height = 50
        self.rows = rows
        self.cols = cols
        self.bg_color = bg_color
        self.reset()

    def create_individual_block(self,col,row, width,height):
        block_x, block_y =( col * width, row * height) // posiciones
        return pygame.Rect( block_x, block_y, width, height)

    def create_wall(self):
        self.blocks = []
        for row in range(self.rows):
            # reset the block row list
            block_row = []
            # creamos las filas
            for col in range(self.cols):
                new_block = self.create_individual_block(col,row,self.width,self.height)
                block_row.append(new_block)
            self.blocks.append(block_row)

    def draw_wall(self, window):
        for row in self.blocks:
            for block in row:
                pygame.draw.rect(window, self.BLUE, block)
                pygame.draw.rect(window, self.bg_color, block, 2)

    def reset(self):
        self.create_wall()
```

Francisco Mendiburu

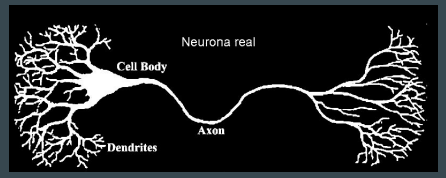
Info del juego

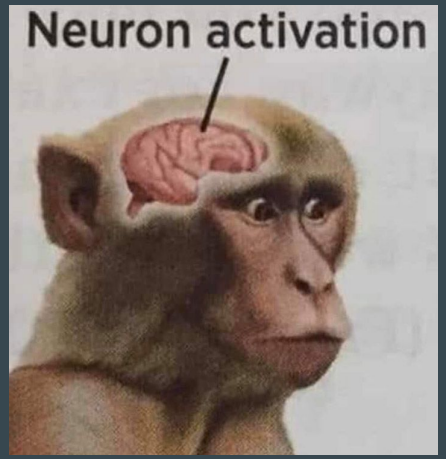
- Puntos
- Posiciones:
  - Pelota
  - Paleta
- Golpes de la pelota a la paleta
- Fin del juego

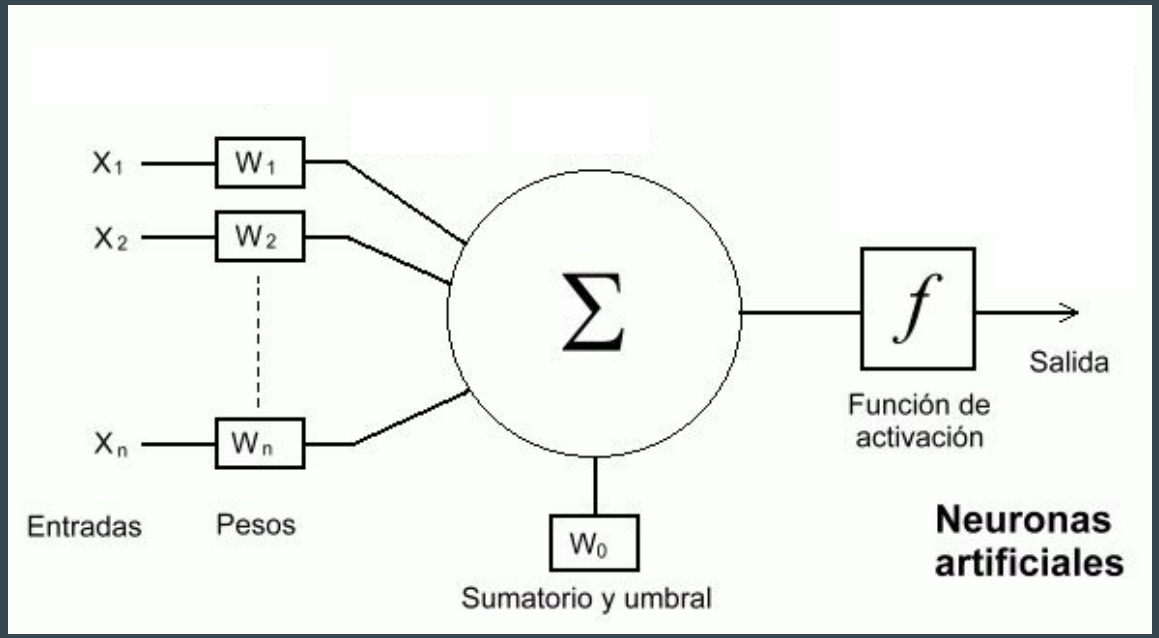
```
class GameInfo:
    def __init__(self, game_over, points,
                 paddle_hits, ball_pos_x,
                 ball_pos_y, paddle_pos_x,
                 paddle_pos_y):
        self.points = points
        self.ball_pos_x = ball_pos_x
        self.ball_pos_y = ball_pos_y
        self.paddle_pos_x = paddle_pos_x
        self.paddle_pos_y = paddle_pos_y
        self.paddle_hits = paddle_hits
        self.game_over = game_over
class Game:
    def __init__(self, window, window_width,
                 window_height, cols, rows):...
    def _draw_hits(self):...
    def _blocks_collision(self):...
    def _window_collision(self):...
    def _paddle_collision(self):..
    def draw(self):...
    def move_paddle(self, left):...
    def reset(self):...
    def loop(self):...
```

Francisco Mendiburu

Redes neuronales



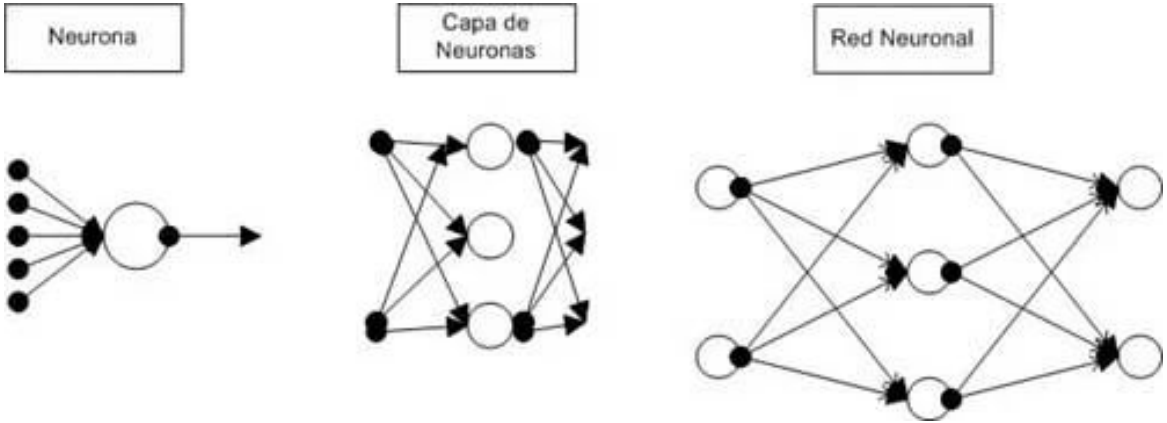




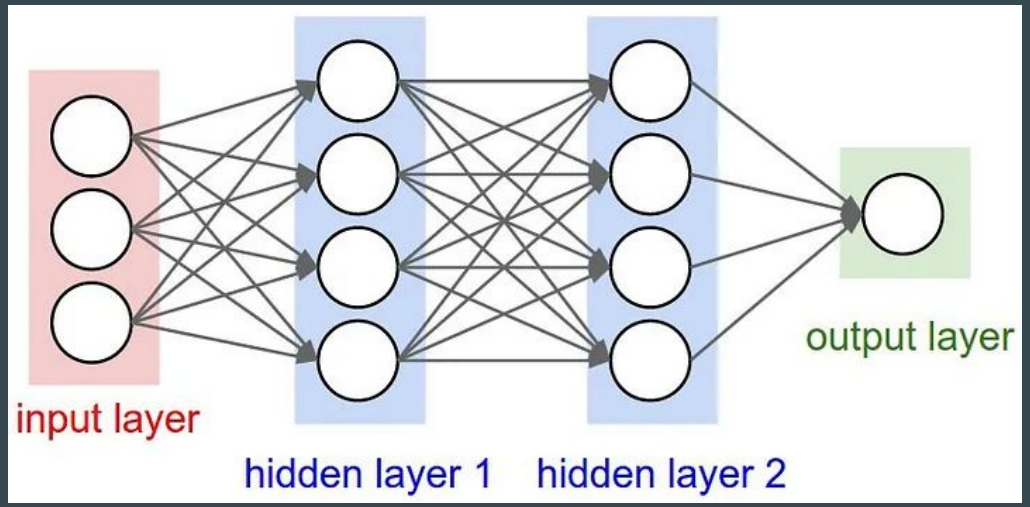
Francisco Mendiburu

Topología de una red neuronal

Organización y **disposición** de las neuronas en la red



Nuestro modelo



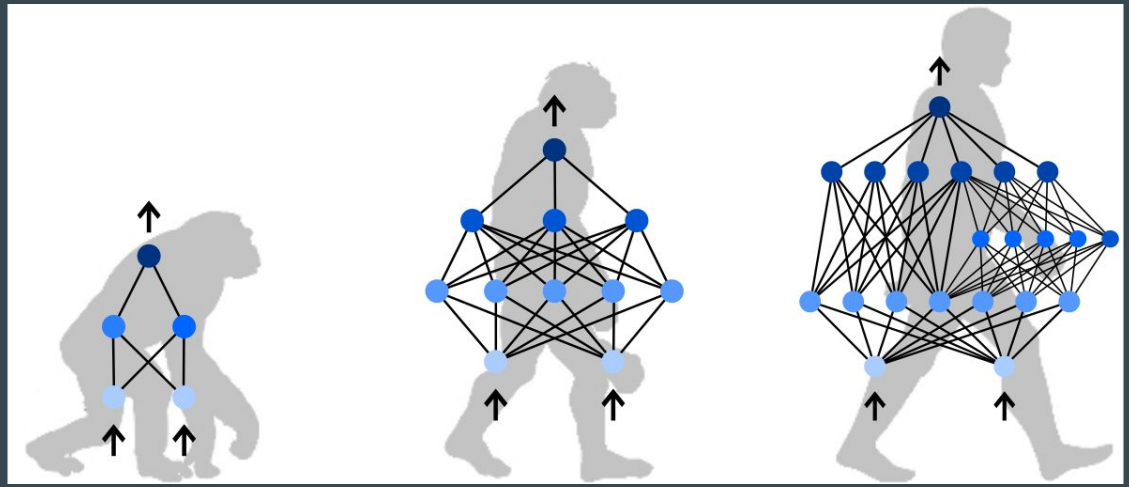
Red neuronal feedfowards

Función de activación:
RELU f(x)= max(0,x)

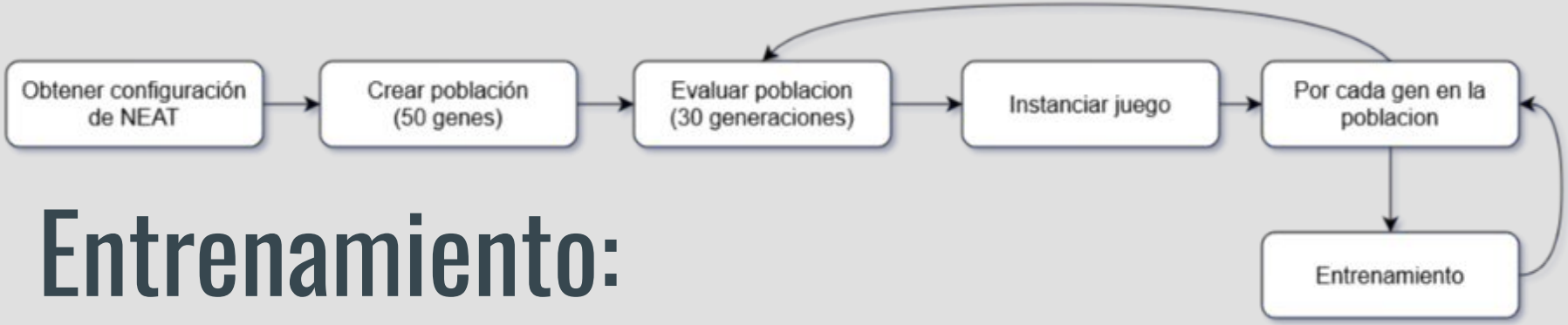
NEAT: Combinar redes neuronales con algoritmos genéticos

NEAT significa "NeuroEvolution of Augmenting Topologies" en inglés, que traducido sería "Neuroevolución de Topologías Adicionales".

NEAT tiene la capacidad para **evolucionar** tanto la **estructura** como los **pesos** de las **redes neuronales**. A diferencia de otros métodos de optimización que solo ajustan los pesos de una red neuronal predefinida.



Joaquin Betes



Entrenamiento:

El funcionamiento de NEAT consta de, mediante algoritmos genéticos, elegir la mejor estructura para una red neuronal, esta configuración es un gen que se va mejorando con cada entrenamiento.

- Elección de parámetros de la población: Mutación, crossover, elitismo.
- Cantidad de generaciones, veces que se evalúan los genes.
- Definir entradas, salidas y acciones de la red neuronal.
- Calcular correctamente el fitness de cada gen.

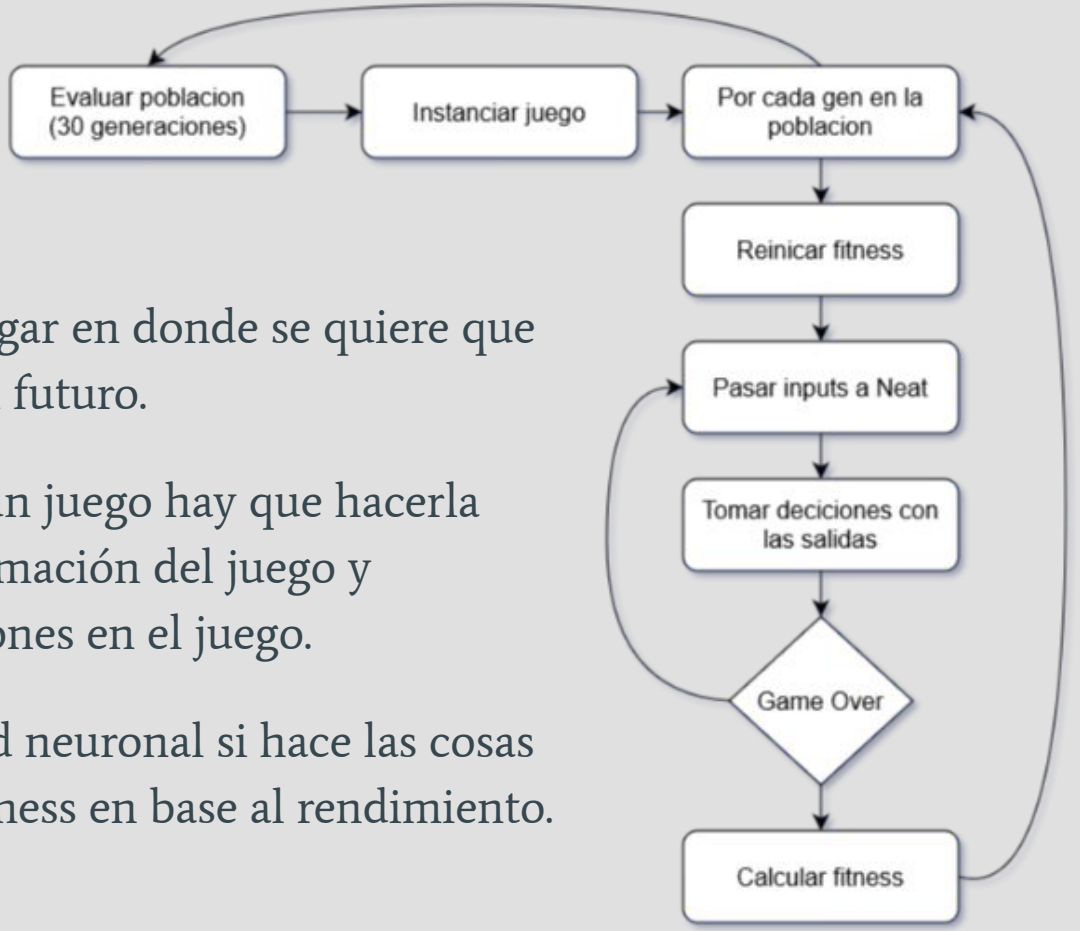
Gino Bartolucci

Entrenamiento:

El ámbito de entrenamiento es el lugar en donde se quiere que la red neuronal se desenvuelva en el futuro.

Para entrenar una red neuronal en un juego hay que hacerla jugar. Dandole como entradas información del juego y interpretando las salidas como acciones en el juego.

Es muy importante indicarle a la red neuronal si hace las cosas bien o mal, para eso se define un fitness en base al rendimiento.

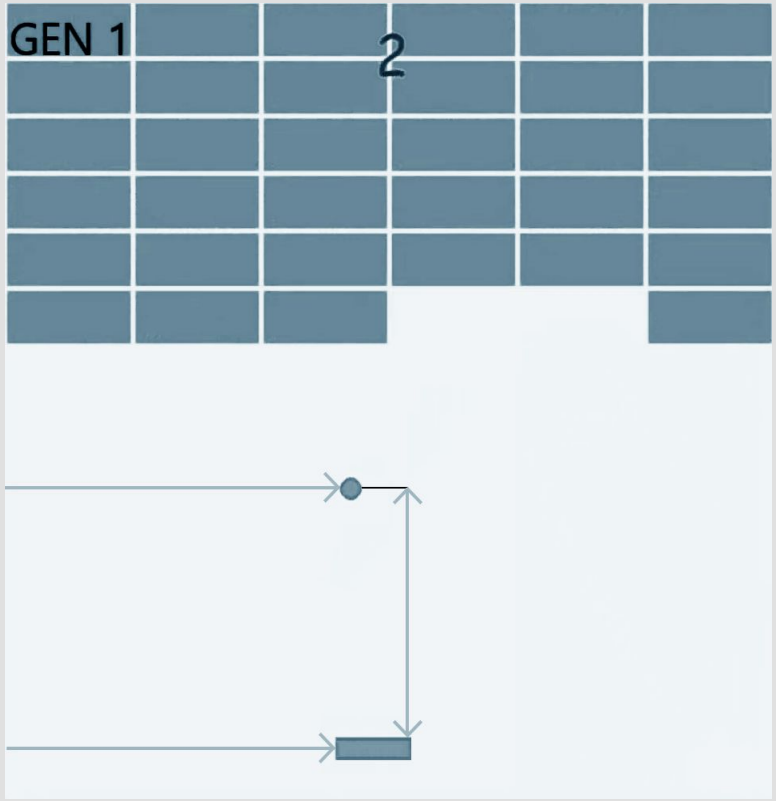


Gino Bartolucci

Entradas:

La IA debe saber la posición de la pelota respecto a la paleta en todo momento.
A su vez se deben usar la menor cantidad de entradas posibles.

```
output = net.activate((
  self.game.game_info.paddle_pos_x,
  abs(self.game.game_info.paddle_pos_y -
  self.game.game_info.ball_pos_y),
  self.game.game_info.ball_pos_x))
decision = output.index(max(output))
```

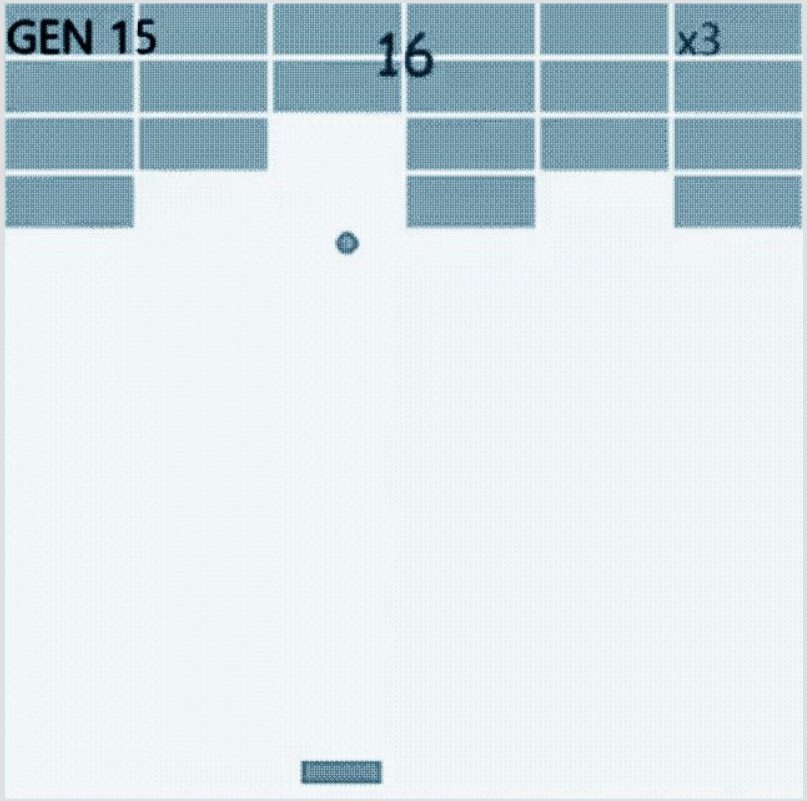


Gino Bartolucci

El fitness

Para elegir un fitness óptimo primero debemos determinar cuándo queremos que el algoritmos sea recompensado y cuando castigado.

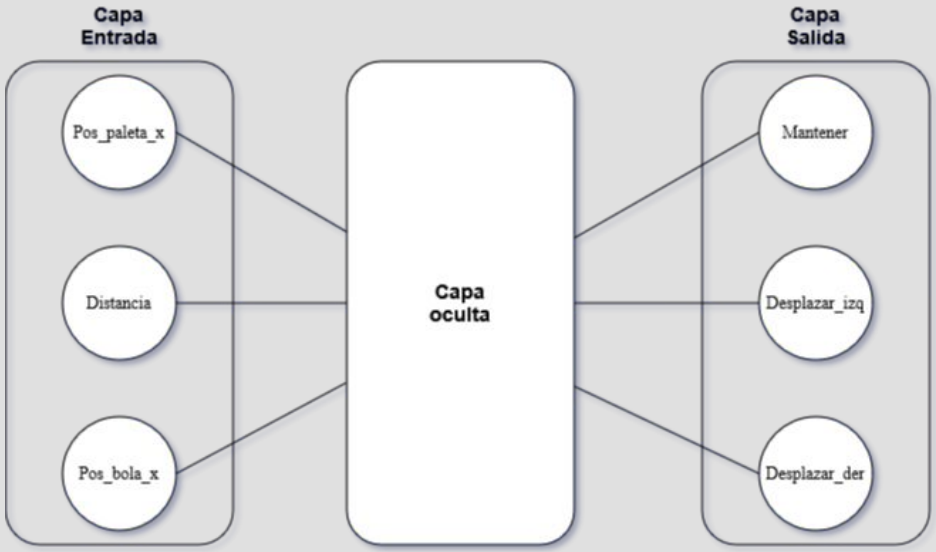
En nuestro caso queríamos que gane el juego pero a su vez, para alcanzar ese objetivo, debe saber jugar.



Gino Bartolucci

Salidas:

Solo existen 2 movimientos pero además esta la opcion de no moverse.



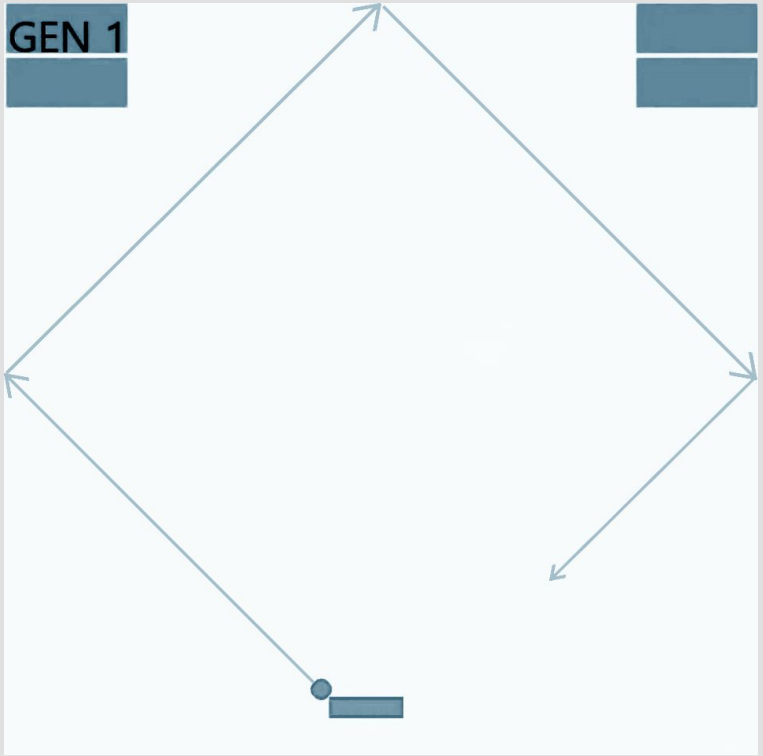

```
output = net.activate((
self.game.game_info.paddle_pos_x,
abs(self.game.game_info.paddle_pos_y -
self.game.game_info.ball_pos_y),
self.game.game_info.ball_pos_x))

decision = output.index(max(output))

if decision == 0:
   pass
elif decision == 1:
   self.game.move_paddle(True)
else:
   elf.game.move_paddle(False)
```

Gino Bartolucci

Inconveniente:

Si otorgamos puntos por pegarle a la pelota, lo que creímos era enseñarle a jugar, el algoritmo tiende a entrar en un loop infinito para sumar puntos, consecuentemente no ganar el juego.

Lo lograba pegandole a 45° con la paleta.



Gino Bartolucci

Solución:

Solo sumar fitness de los bloques rotos cuando se pierde, que son las primeras veces que la IA juega y así evitar el loop.
Cuando **gane dar una cantidad muy alta de puntos** mayor a la máxima que puede obtener si pierde, a esto restarle un equivalente a los golpes de la pelota, de esta forma no entra en loop y mientras más bloques rompe con **menos golpes mejor**.

```
def calculate_fitness(self, genome, game_info):
  if game_info.points == self.game.rows*self.game.cols:
    genome.fitness += 50 - game_info.paddle_hits/10
  else: genome.fitness += game_info.points
```

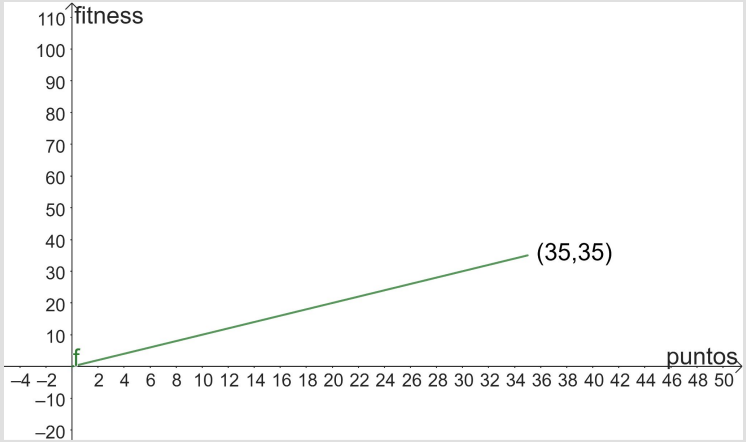
El fitness matemáticamente:

Representación como función matemática.

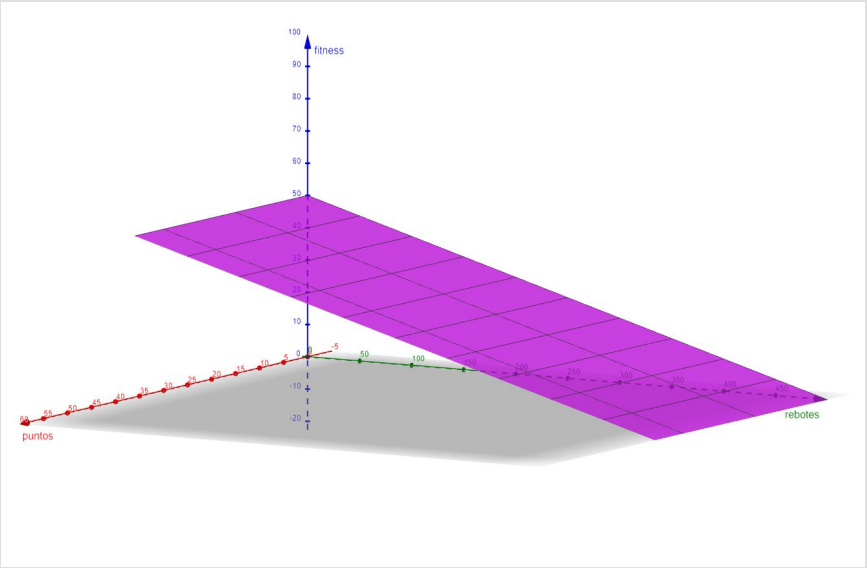
x: Cantidad de bloques rotos.

y: Golpes de la pelota en la paleta.

C: Total de bloques en el juego.



Si pierde, la función se puede ver como una función de una variable lineal.

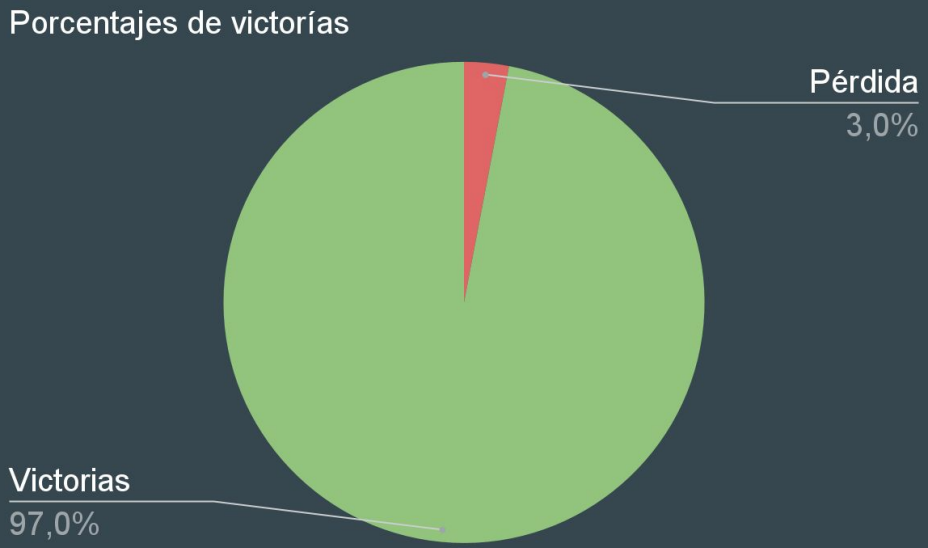


$$z(x,y) = \begin{cases} 50 - \frac{y}{10} & \text{si } x = C \\ x & \text{en otro caso} \end{cases}$$

Gino Bartolucci

Resultados

Fitness última generación = **47.1** o **0.942**



Vídeo demostración:

Joaquin Bettes

Problemas resueltos

OverFitting

Superar el desempeño humano
vs
ganar siempre el juego

Fitness vs aleatoriedad del entorno.

Generación	Porcentaje de victorias	Mejor fitness
5	54%	22
10	62%	46,9
25	77%	48
30	97%	47,1

Joaquin Betes

Conclusiones

Hemos investigado con éxito la **combinación** de **algoritmos genéticos** y **Redes neuronales** en entornos de juegos dinámicos y complejos.

NEAT nos permitió evolucionar las topologías neuronales de forma que las redes neuronales puedan adaptarse y cambiar continuamente para abordar el juego Breakout de una forma muy **efectiva** sin perder la **flexibilidad** necesaria para abordar un **entorno dinámico** con altos niveles de aleatoriedad.

Este enfoque ha demostrado ser prometedor para mejorar la capacidad de los agentes IA en *juegos o entornos* **dinámicos**, **aleatorios** e **interactivos.**

Preguntas