

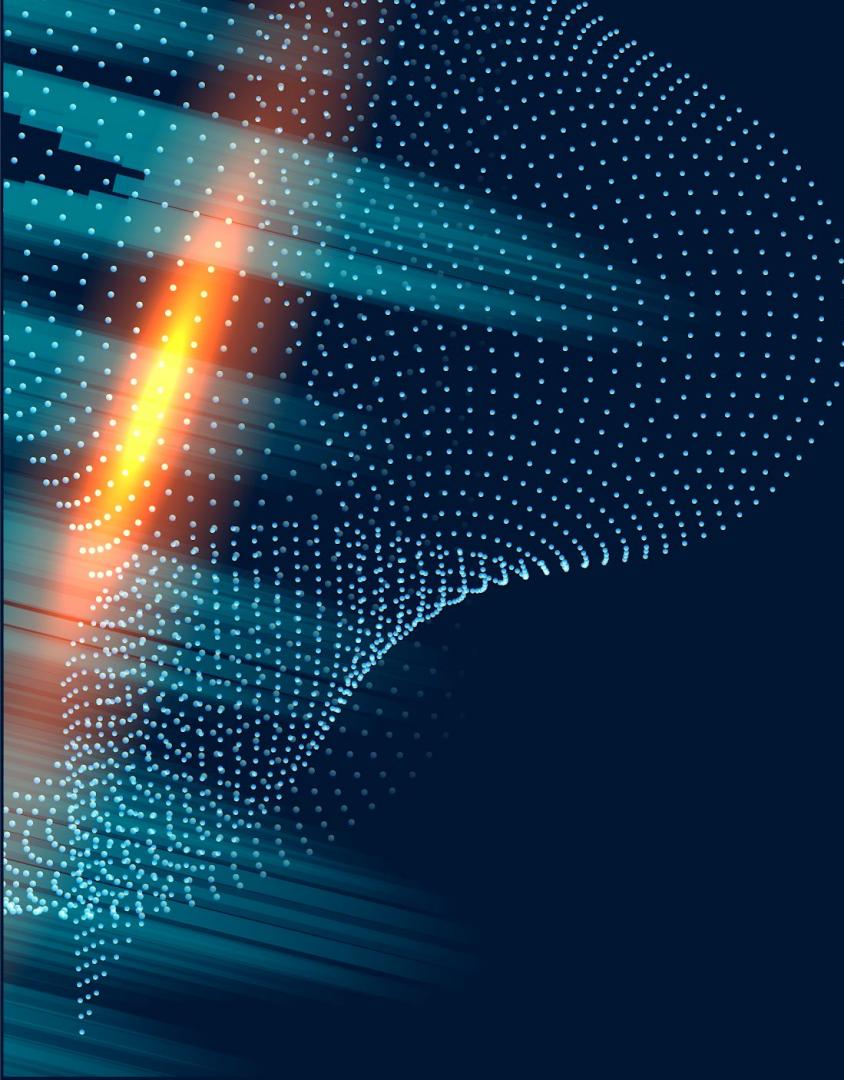
# Algoritmos Genéticos

---

Sistemas de Inteligencia Artificial

Banfi, Malena  
Fleischer, Lucas  
Perez Rivera, Mateo  
Szejer, Ian

Grupo 3



# Ejercicio 1

---

## Consigna

Cómo implementarán, mediante Algoritmos Genéticos, un programa que tome una imagen cuadrada y trate de representar de la mejor manera posible dicha imagen en un mapa de NxN caracteres ASCII.

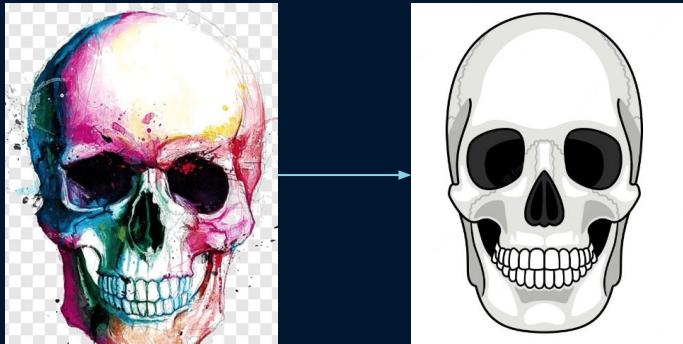
## Posible implementación



# Resolución

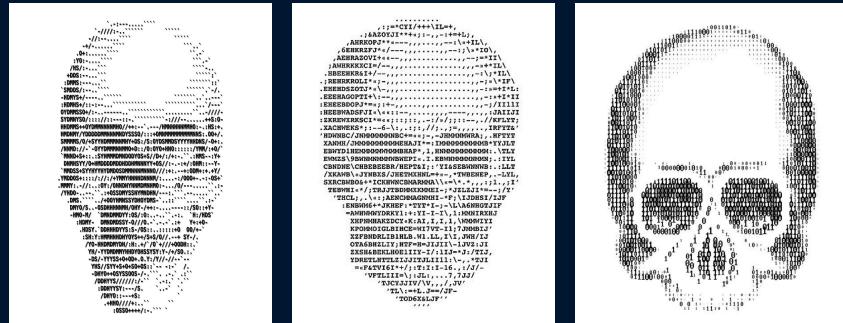
## 1. Procesamiento de la imagen

La imagen debe ser cargada y redimensionada a una resolución adecuada (nos permitirá establecer el tamaño del mapa de caracteres que se generará a partir de la imagen). Además, se debe convertir la imagen en escala de grises para facilitar el proceso de conversión a ASCII.



## 2. Generación de la Población inicial

Se generan varias matrices de caracteres ASCII al azar de tamaño NxN. Cada matriz representa una posible solución al problema de convertir la imagen en ASCII.



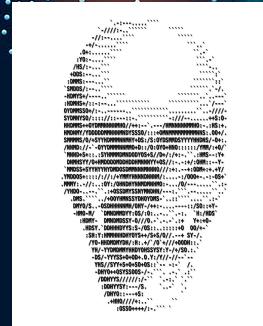
# Resolución

## 3. Fitness

Cada matriz de caracteres ASCII se evalúa viendo cuán bien representa la imagen original. Esto se puede evaluar utilizando una función de evaluación que mide la similitud entre la imagen original y la imagen representada en caracteres ASCII.

Nuestra función de evaluación es la suma de las diferencias absolutas entre los valores de los píxeles en las dos imágenes.

$$f(A,B) = \sum |A(i) - B(i)|$$



## 4. Selección de padres

Se seleccionan dos matrices de caracteres ASCII de la población actual para ser los padres de la siguiente generación.

La probabilidad de que una matriz sea seleccionada como parente depende de su aptitud relativa (fitness) en comparación con otras matrices.

Nuestra técnica de selección será la selección por ruleta. Este enfoque puede ser útil si se desea una selección más diversa en la población.

# Resolución

## 5. Crossover

Se combinan los caracteres de los padres seleccionados para generar un nuevo hijo.

Nuestra estrategia es una crusa de dos puntos, que consiste en seleccionar dos puntos aleatorios en la matriz y cambiar los bloques de caracteres entre ambos puntos entre dos padres para generar dos hijos. De esta manera, se conservan las estructuras de los individuos originales y se pueden generar nuevos patrones a partir de las combinaciones de bloques.

## 6. Mutacion

Se introduce aleatoriedad en la población al cambiar aleatoriamente un pequeño número de caracteres ASCII en cada hijo generado.

Nuestra visión sobre esto es que se podrían cambiar algunos caracteres en la cadena ASCII (de forma totalmente aleatoria) del hijo, utilizando mutación multigen limitado

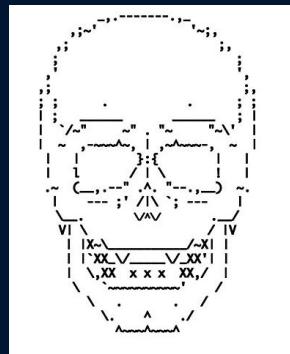
# Resolución

## 7. Nueva generación

Los hijos generados se agregan a la población actual. La población se vuelve a evaluar en términos de aptitud (fitness) y se seleccionan nuevos padres para la siguiente generación.

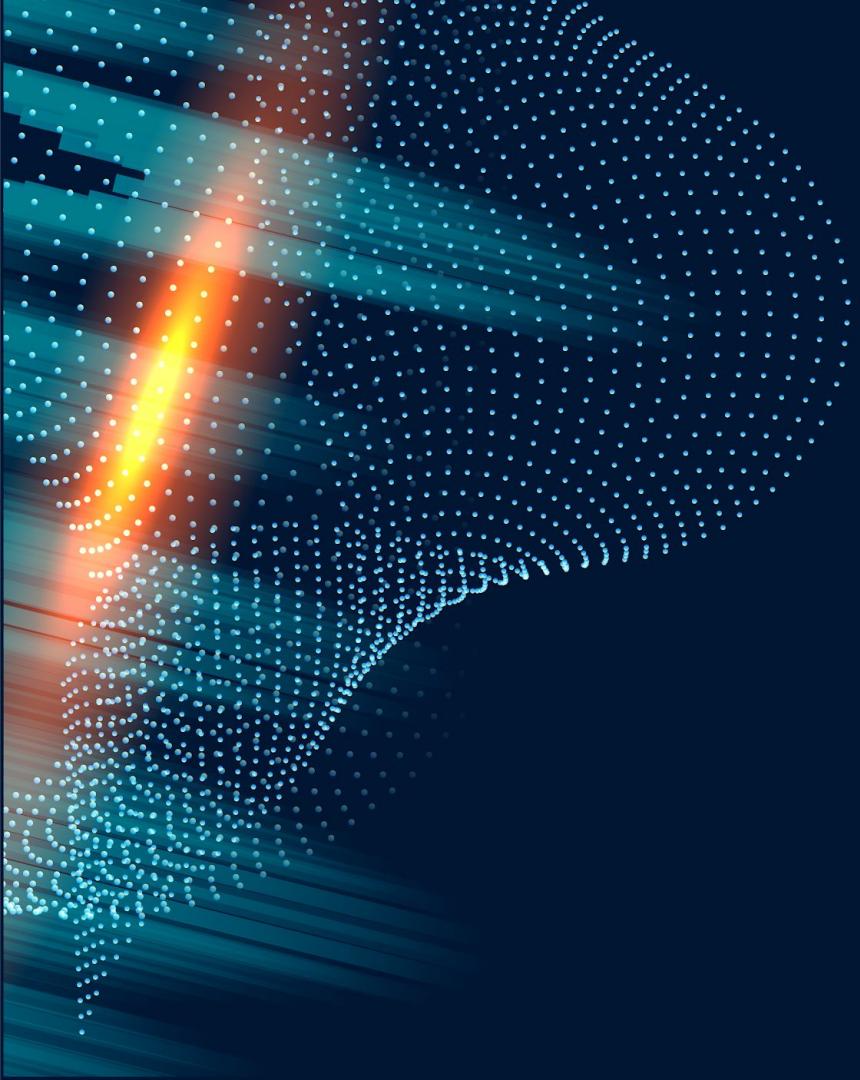
## 8. Convergencia

El proceso se repite durante un número de generaciones predefinido o hasta que se alcanza cierto criterio de convergencia.



## 9. Resultados

La matriz de caracteres ASCII con la mayor aptitud (fitness) al final del proceso se utiliza para representar la imagen original en ASCII.



# Ejercicio 2

# Consigna - Introducción

Mediante Algoritmos Genéticos, encontrar la forma de mezclar proporciones de los diferentes colores de su paleta para lograr el color que más se le acerque al color deseado y poder ayudar al artista.

- Input:
  - Paleta de colores
  - Color objetivo
  - Hiperparametros
- Output:
  - Proporción de cada color de la paleta a utilizar
  - Similitud con el color objetivo

# Estrategia

## Población Inicial

Paleta de Colores

## Fitness

Distancia Euclidea

## Cruza

Cruza Uniforme

## Mutacion

Mutacion Multigen  
Uniforme

## Seleccion

- Elite
- Rank
- Roulette
- Probabilistic Tournament

## Condicion de corte

- Limite generacional
- Condicion optima de fitness

# Nuestra configuración

```
{ config.json > ...  
{  
  "colors_palette" : "colors_palette.txt",  
  "selection_algorithm": "roulette",  
  "mutation_rate": 0.2,  
  "max_generations": 90,  
  "expected_fitness": 0.95,  
  "population_number": 60,  
  "fitness_cut": true  
}
```

```
colors_palette.txt  
56 200 131  
255 0 0  
0 255 0  
0 0 255  
0 255 255  
255 255 0  
255 0 255  
0 0 0  
255 255 255
```

Nos manejamos con un archivo .json para poder cambiar los siguientes hiperparametros:

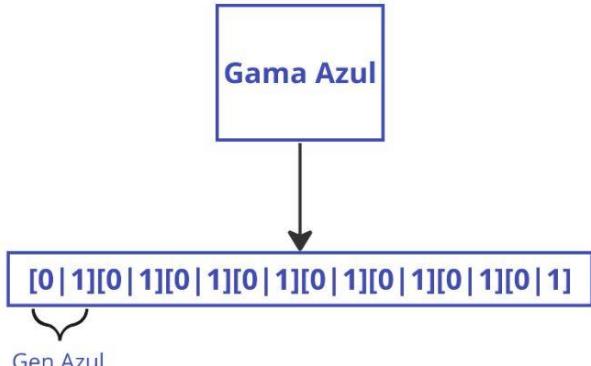
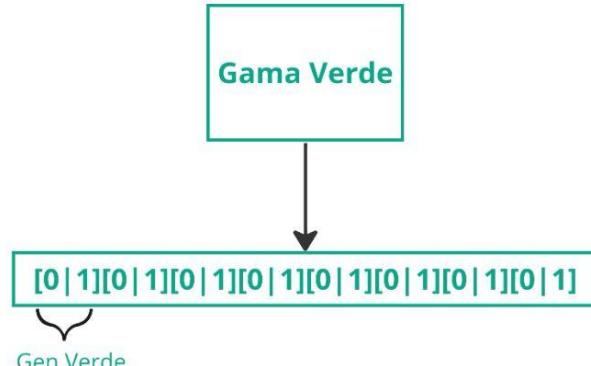
- Metodo de seleccion
- Mutation rate
- Cantidad máxima de generaciones
- Fitness esperado
- Numero de poblacion inicial
- Condicion de corte

Luego, en el colors\_palette.txt, pasamos el color objetivo en la primera linea y lo que sigue abajo es la paleta que se utiliza.

# Estructura del individuo

**Genotipo:** Se compone de 24 bits, los cuales se dividen en 3 grupos de 8 bits y así lograr formar las 3 gamas de RGB (Rojo, Verde y Azul).

**Gama:** Representa la proporción de cada color RGB respectivamente. Compuesta por 8 genes.



## **Fitness - Distancia Euclidea**

La distancia euclídea se utiliza para medir la distancia entre dos puntos en un espacio n-dimensional. En este caso, estamos trabajando en un espacio tridimensional (rojo, verde y azul), por lo que la distancia euclídea se calcula como:

$$Distancia\ euclídea = \sqrt{(R_{eval} - R_{obj})^2 + (G_{eval} - G_{obj})^2 + (B_{eval} - B_{obj})^2}$$

## Fitness - Distancia Euclidea

En nuestra implementación, luego de calcular la distancia euclíadiana, a la distancia máxima posible se resta la distancia euclíadiana obtenida, para así obtener una puntuación de aptitud.

$$Fitness = self.MAXDISTANCE - Distancia\ euclíadiana$$

Luego, se normaliza la puntuación de aptitud dividiéndola por la distancia máxima posible, lo que da como resultado un valor de 0 a 1. Un valor de 1 significa que el color eval es idéntico al color objetivo, mientras que un valor de 0 significa que es totalmente diferente.

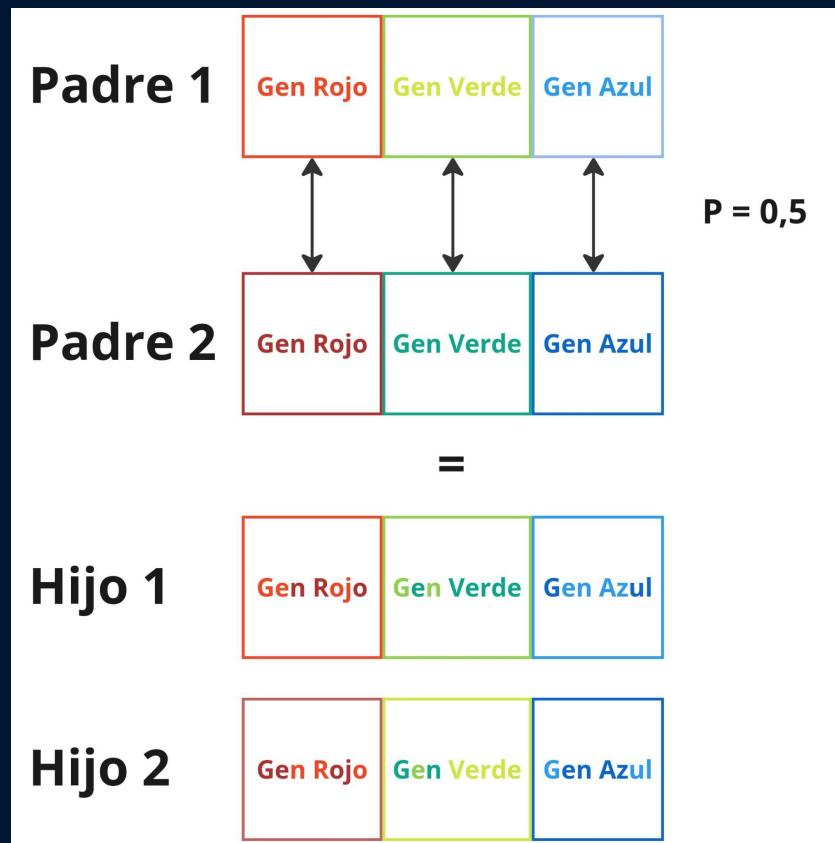
$$Normalización = \frac{Fitness}{self.MAXDISTANCE}$$

# Cruza - Cruza uniforme

Se produce un intercambio de alelos en cada gen con probabilidad  $P$  entre  $[0,1]$ . Por lo general  $P=0,5$ .

Es el único tipo de cruce visto que no mantiene relación posicional entre alelos.

La idea sería que los hijos resultantes de esta cruce tengan una combinación de los valores RGB de ambos padres, permitiendo explorar nuevas combinaciones y aumentar la diversidad genética de la población.



# Mutación - Mutación Multigen Uniforme

Es una variación en la información genética que se almacena en el cromosoma. Para que la mutación sea más eficiente, es necesaria una correcta arquitectura y separación de genes.

**Una vez que los individuos se crean, esa generación se considera inmutables**

En nuestra implementación de Mutación Multigen Uniforme, se define una probabilidad de mutación ( $P_m$ ). Teniendo en cuenta la  $P_m$ , se seleccionan aleatoriamente los bits de cada color y se cambia el valor entre 0 y 1. De esta manera, se estarían explorando nuevas combinaciones de colores en la población, lo que podría llevar a una mejor solución, enriqueciendo la diversidad genética.



Cada gen con probabilidad  $P_m$  (pasada por parámetro) de mutar

# Condición de corte

En nuestra implementación, la condición de corte es un parámetro de entrada, el cual el usuario puede elegir entre las siguientes opciones:

## Limite generacional

## Condición óptima de fitness

```
 } config.json > ...
{
  "colors_palette" : "colors_palette.txt",
  "selection_algorithm": "roulette",
  "mutation_rate": 0.2,
  "max_generations": 90,
  "expected_fitness": 0.95,
  "population_number": 60,
  "fitness_cut": true
}
```

## Métodos de Selección

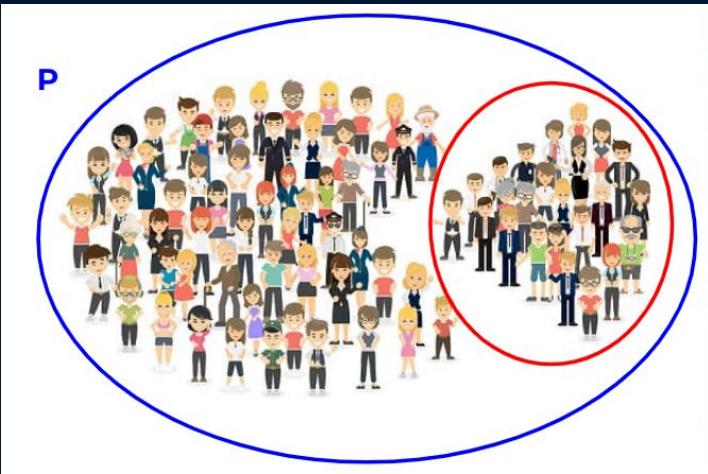


# Elite

Sea  $P$  la población total.

Se seleccionan los individuos con mejor fitness.

El término "selección élite" en genética se refiere al proceso de selección de los individuos con los mejores rasgos genéticos para ser utilizados como progenitores en un programa de mejoramiento genético de una especie vegetal o animal.



# Convergencia prematura

Se produce cuando el algoritmo se detiene prematuramente, antes de haber encontrado la solución óptima, porque se ha estancado en un mínimo local en lugar de continuar explorando el espacio de soluciones para encontrar el mínimo global.

Esto puede suceder cuando el algoritmo se enfoca demasiado en un área específica del espacio de soluciones y no explora otras regiones que pueden contener soluciones aún mejores.

La convergencia prematura puede ser especialmente problemática en algoritmos de aprendizaje automático, donde una solución subóptima puede tener un impacto significativo en el rendimiento del modelo final.

# ¿Cómo evitar la convergencia prematura?

Para evitar la convergencia prematura, es importante ajustar los parámetros del algoritmo de manera adecuada y utilizar técnicas de exploración del espacio de soluciones, como la mutaciones y el uso de diferentes estrategias de búsqueda.

- Mutaciones incluidas
- Agregar padres de la generación actual

# Resultados obtenidos



# **Consideraciones generales para los resultados**

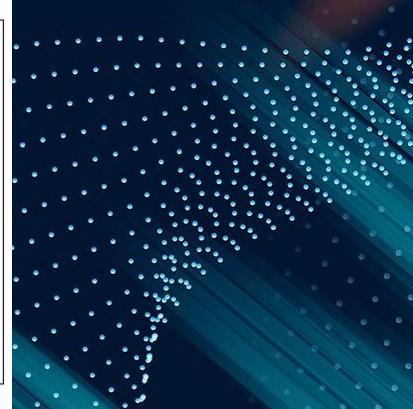
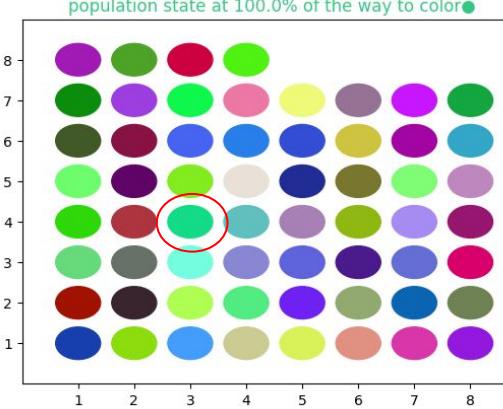
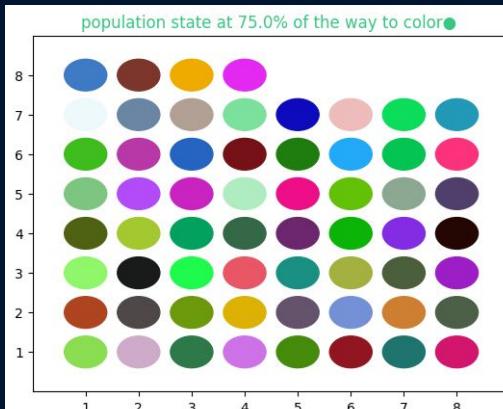
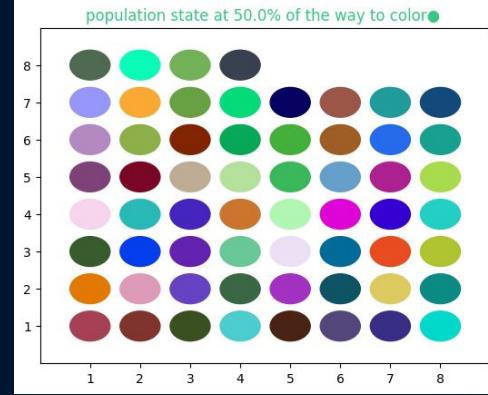
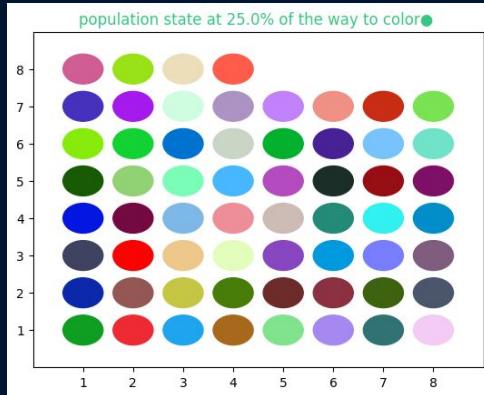
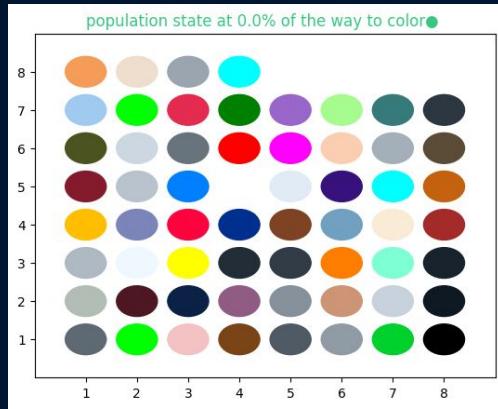
- Los gráficos de cada método de selección fueron realizados tomando solo un valor variable y dejando todos los demás constantes, para una buena comparación.
- Se realizan los distintos gráficos para cada método de selección para una mejor comprensión

# Configuración del archivo de entrada

```
{  
  "colors_palette" : "colors_palette.txt",  
  "selection_algorithm": "elite",  
  "mutation_rate": 0.2,  
  "max_generations": 90,  
  "expected_fitness": 0.95,  
  "population_number": 60,  
  "fitness_cut": true  
}
```

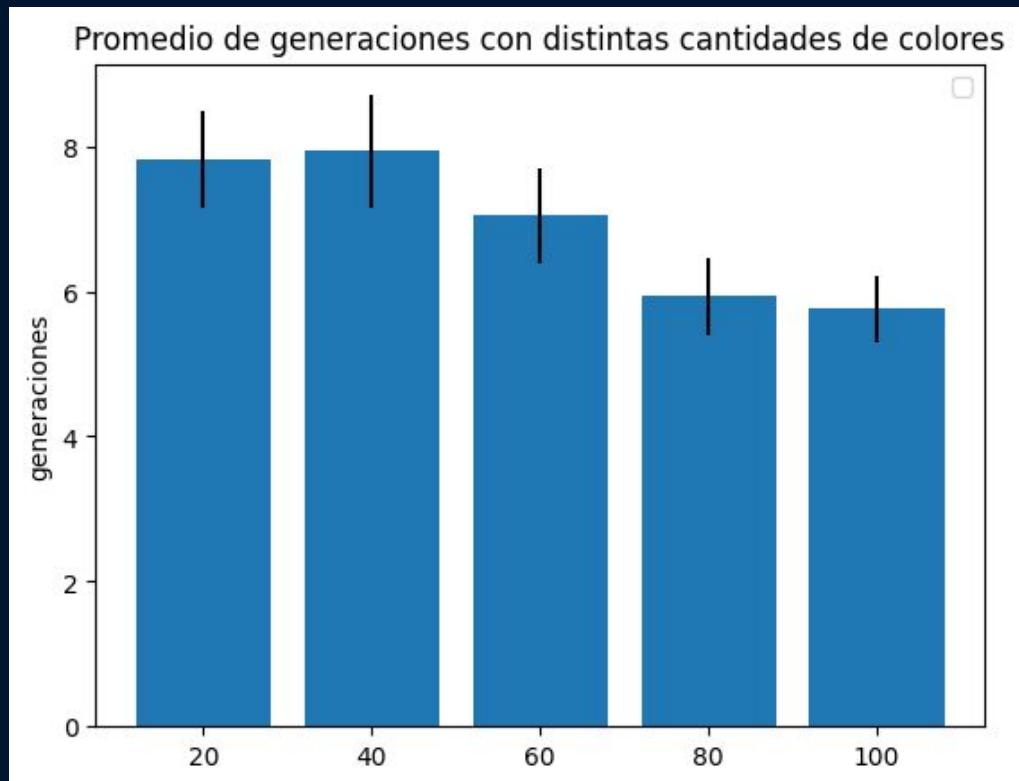
# Evolución de la paleta de colores hasta el color obtenido

Color objetivo: 56 200 131

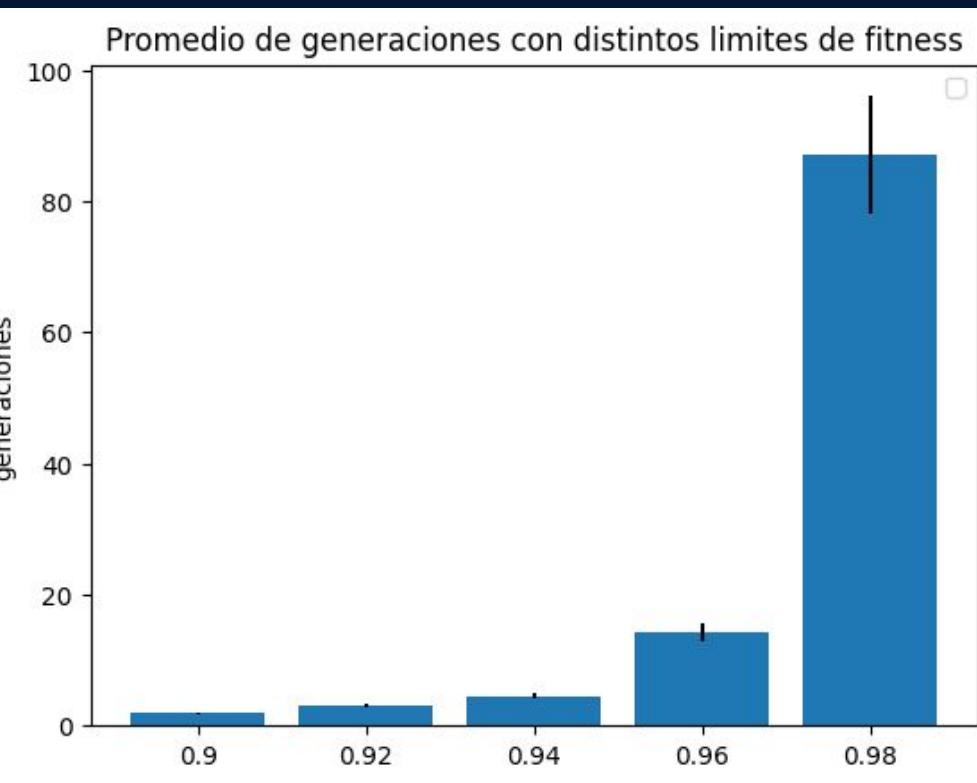


# Promedio de generaciones con distintas cantidades de población

Gracias al siguiente gráfico podemos observar que: a mayor población, mayor variedad de cromosomas, mayor beneficio para la evolución de las generaciones. Por lo que con una población mayor, menos cantidad de generaciones para llegar al objetivo.



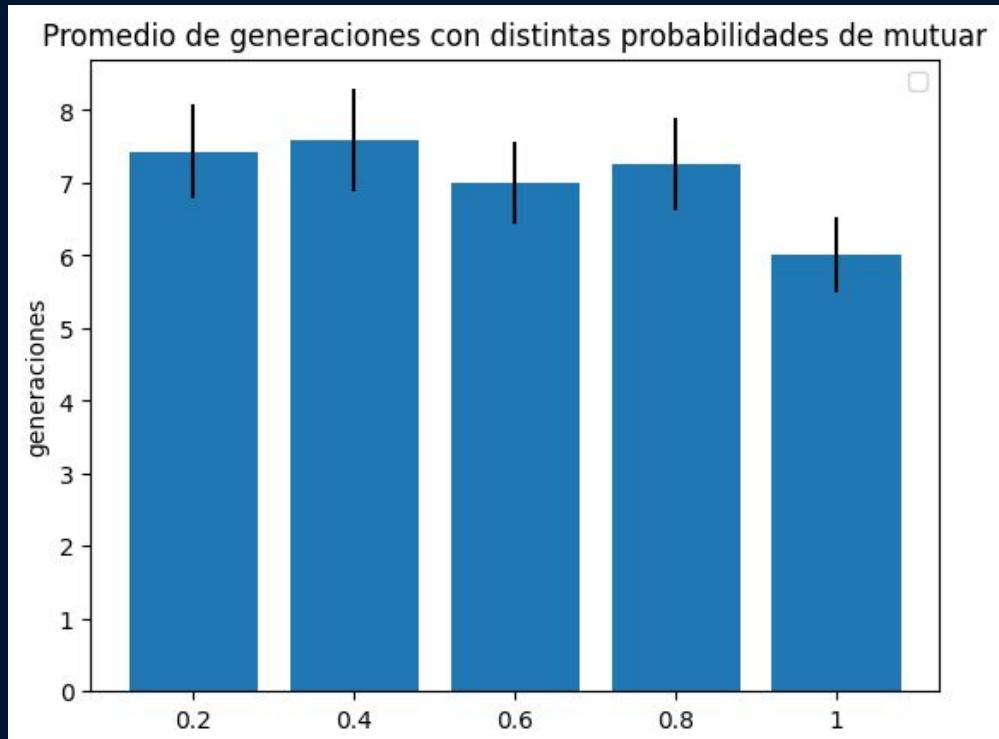
# Promedio de generaciones con distinto límite fitness



Está claro por el gráfico, que la cantidad de generaciones que deben pasar para llegar a mayor fitness aumenta exponencialmente al acercarse al 1.

# Promedio de generaciones con distintas probabilidades de mutación

Con este gráfico podemos concluir, que el aumento de  $P_m$  a la hora de analizar la mutación de un gen, presenta una mejora al sistema, pero muy menor.

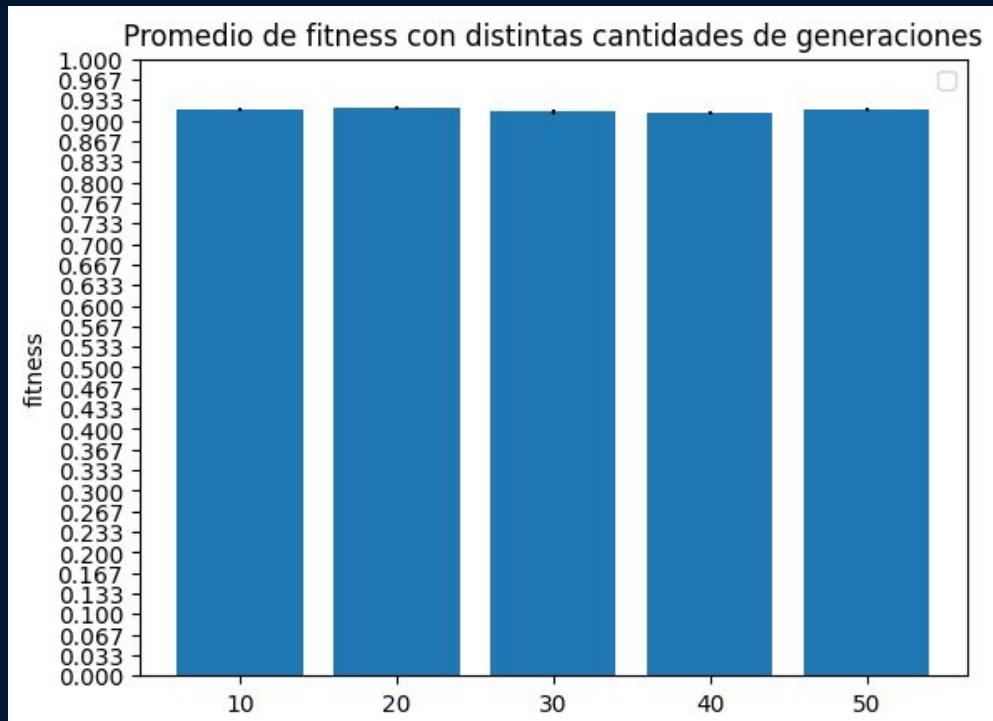


# Promedio de fitness con distintas cantidades generacionales

Como se puede observar, nuestro algoritmo nunca llega al ideal 100%, ya que en un punto aunque sigas iterando no se aproxima más.

Los valores exactos son:

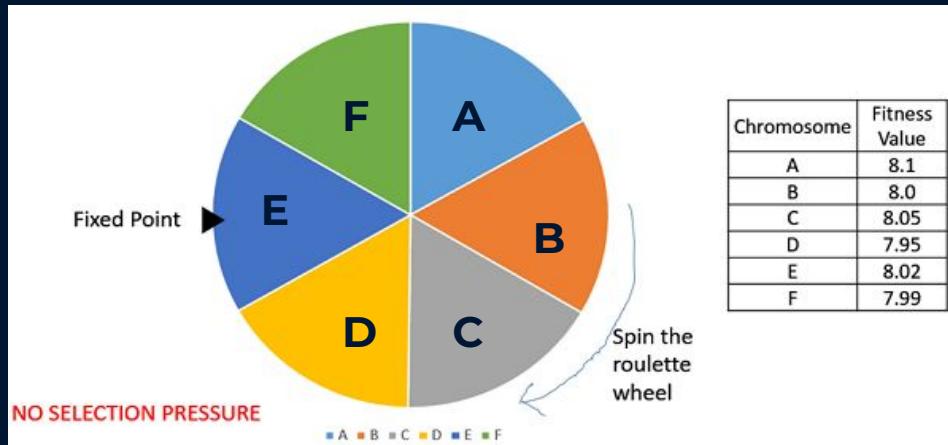
```
[0.9180924526060111,  
0.9200520671474273,  
0.9140033748867457,  
0.9134438375452271,  
0.9184085117482099]
```



# Rank

Se remueve el concepto de fitness value mientras se selecciona un parent. Cada individuo de la población es rankeada de acuerdo a su fitness.

La selección del parent depende del ranking individual y no del fitness en sí. Los mejores rankeados son preferidos ante los peores rankeados.



Chromosome	Fitness Value
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99



Chromosome	Fitness Value	Rank
A	8.1	1
B	8.0	4
C	8.05	2
D	7.95	6
E	8.02	3
F	7.99	5

# Resultados obtenidos

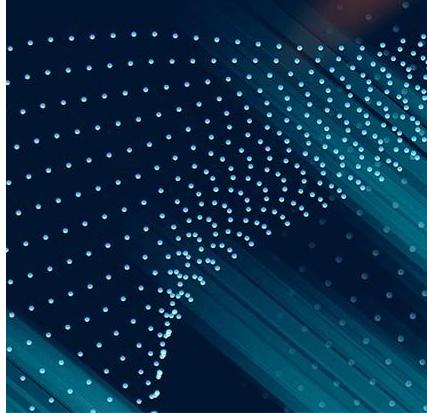
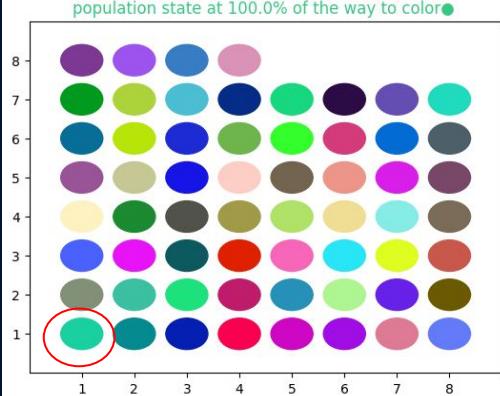
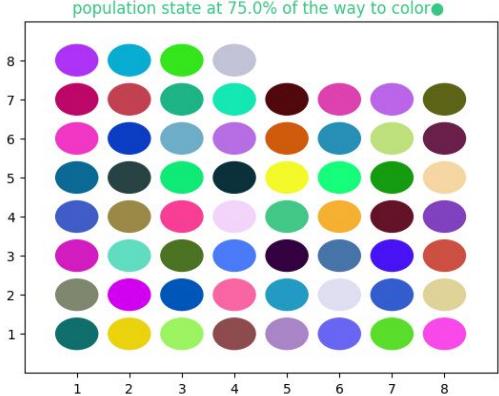
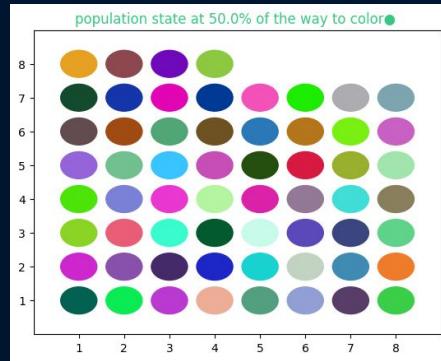
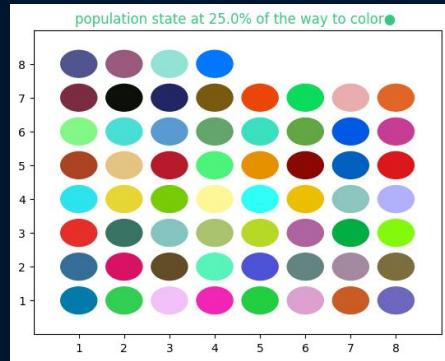
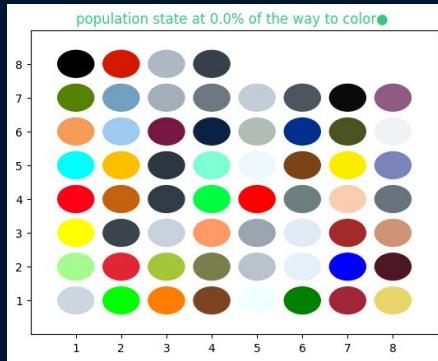


# Configuración del archivo de entrada

```
{  
  "colors_palette" : "colors_palette.txt",  
  "selection_algorithm": "rank",  
  "mutation_rate": 0.2,  
  "max_generations": 90,  
  "expected_fitness": 0.95,  
  "population_number": 60,  
  "fitness_cut": true  
}
```

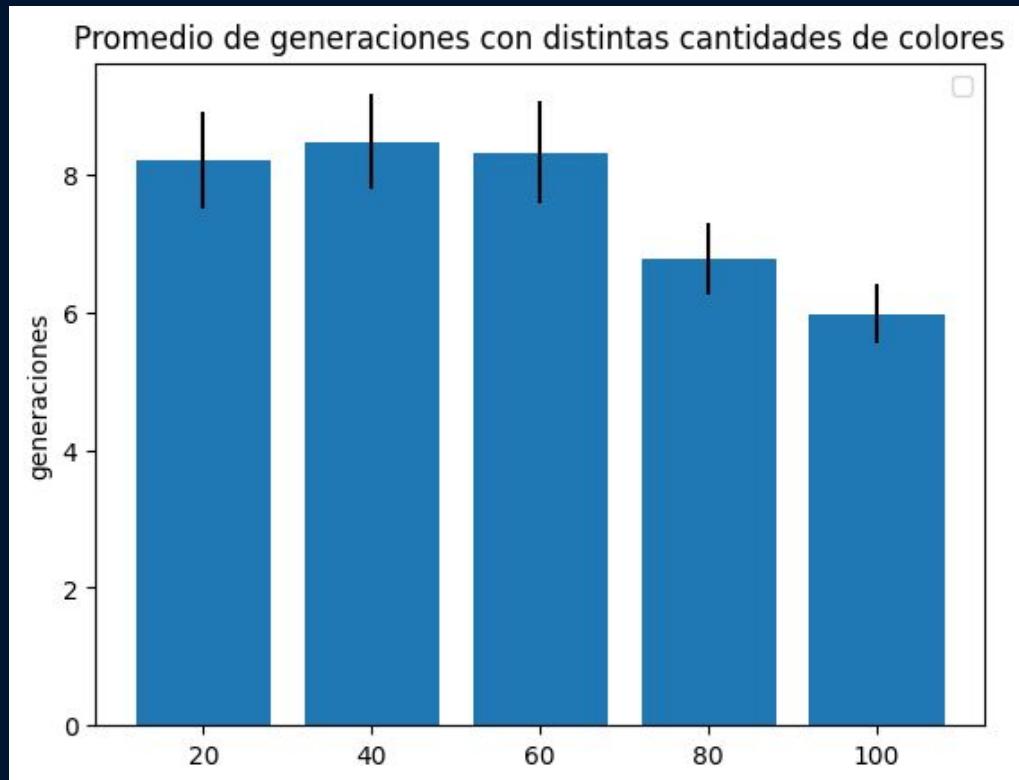
# Evolución de la paleta de colores hasta el color obtenido

Color objetivo: 56 200 131

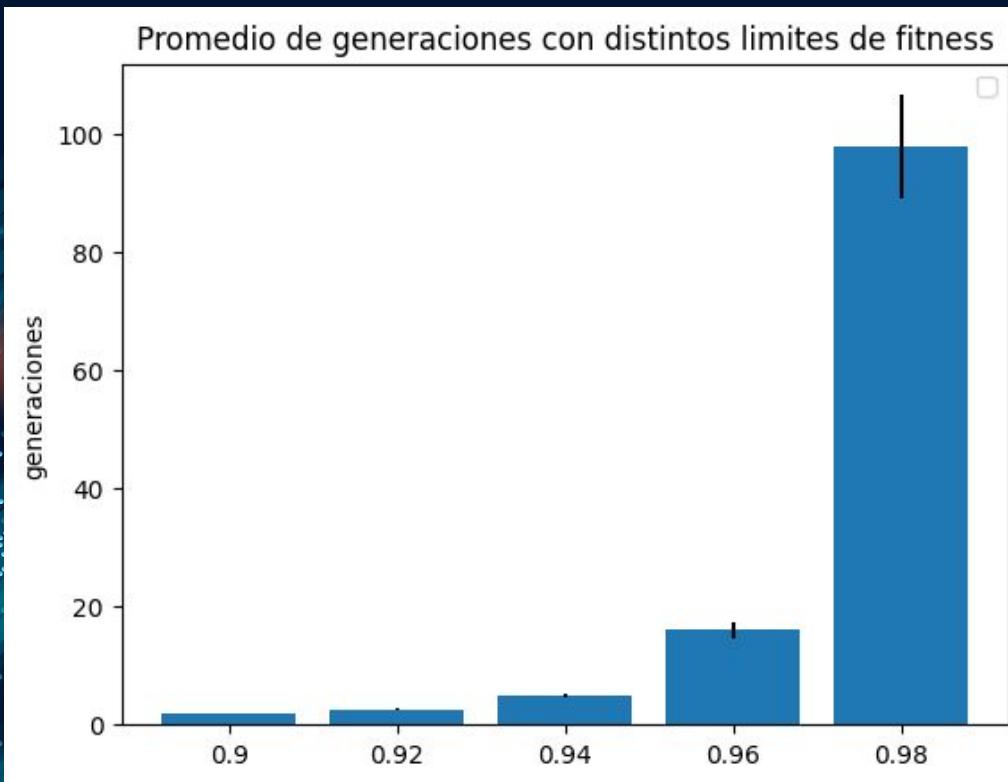


# Promedio de generaciones con distintas cantidades de población

Gracias al siguiente gráfico podemos observar que: a mayor población, mayor variedad de cromosomas, mayor beneficio para la evolución de las generaciones. Por lo que con una población mayor, menos cantidad de generaciones para llegar al objetivo.



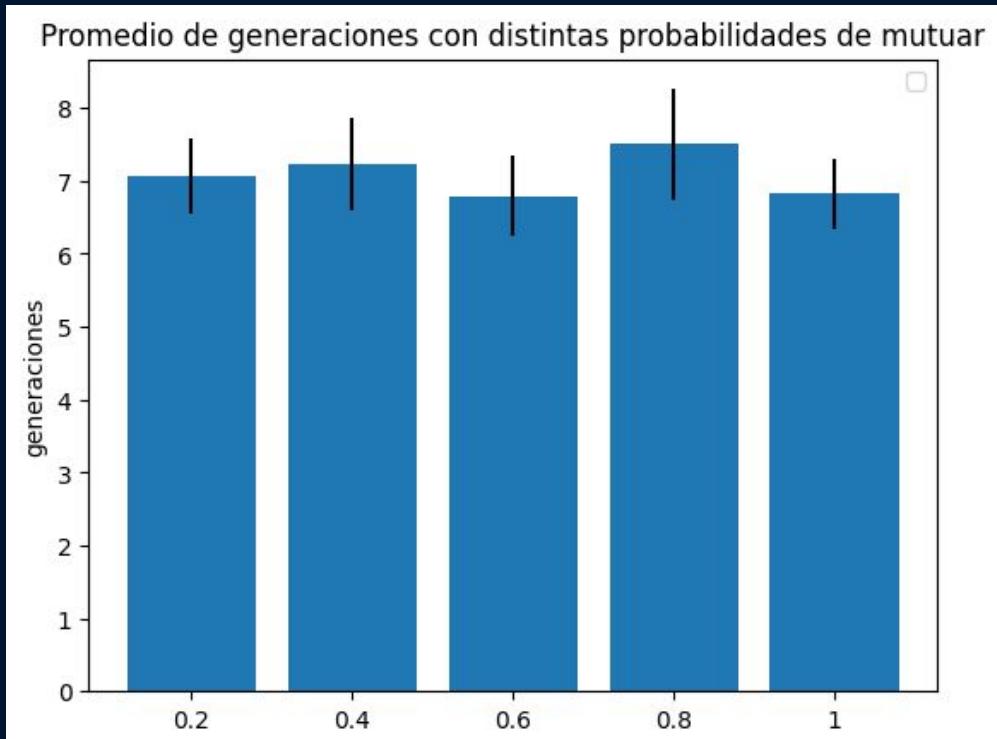
# Promedio de generaciones con distinto límite fitness



Está claro por el gráfico, que la cantidad de generaciones que deben pasar para llegar a mayor fitness aumenta exponencialmente al acercarse al 1.

# Promedio de generaciones con distintas probabilidades de mutación

Con este gráfico podemos concluir, que el aumento de  $P_m$  a la hora de analizar la mutación de un gen, presenta una mejora al sistema, pero muy menor.

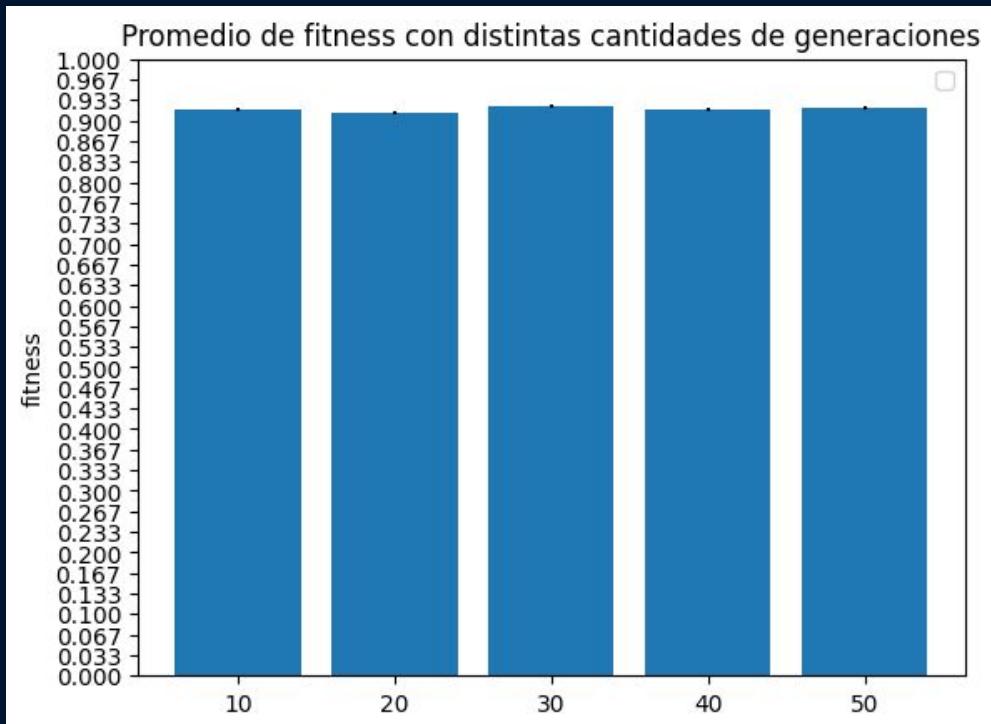


# Promedio de fitness con distintas cantidades generacionales

Como se puede observar, nuestro algoritmo nunca llega al ideal 100%, ya que en un punto aunque sigas iterando no se aproxima más.

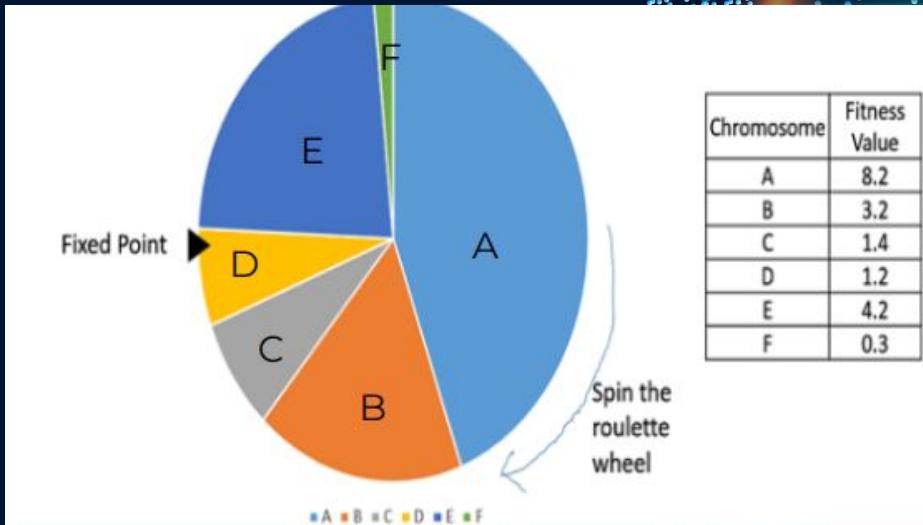
Los valores exactos son:

```
[0.9171687724822357,  
0.913326554986464,  
0.9235799169592264,  
0.9183596434579573,  
0.9214061773033366]
```



# Roulette

En la selección por ruleta, cada individuo en la población tiene una probabilidad proporcional a su aptitud para ser seleccionado como padre o madre en la reproducción. Es decir, los individuos más aptos tienen una mayor probabilidad de ser seleccionados que los menos aptos, pero incluso los menos aptos todavía tienen una pequeña probabilidad de ser seleccionados.



A E B C

# Resultados obtenidos

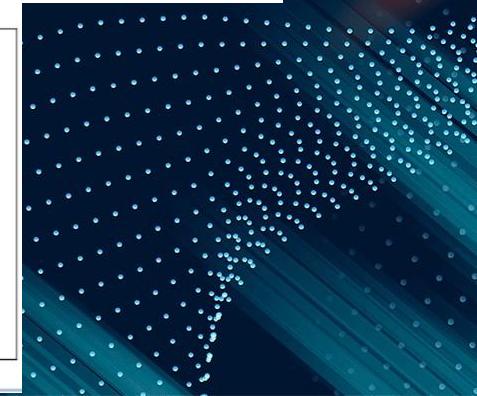
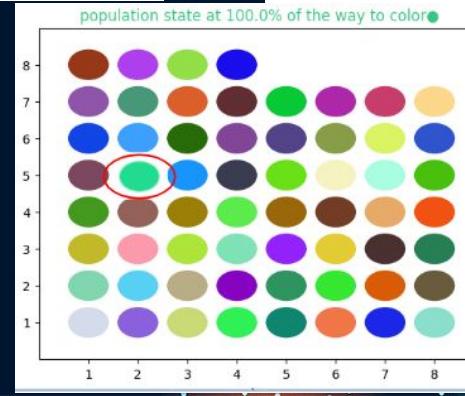
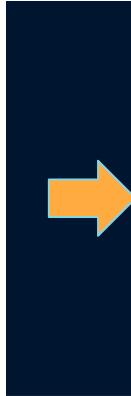
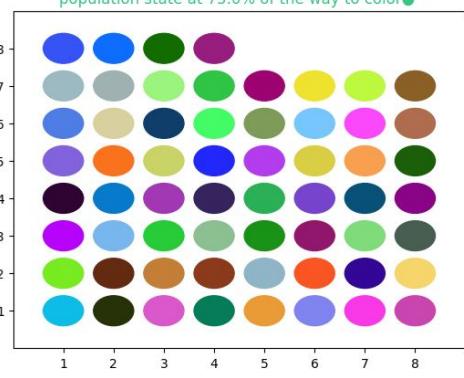
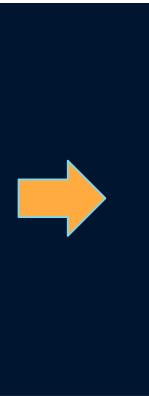
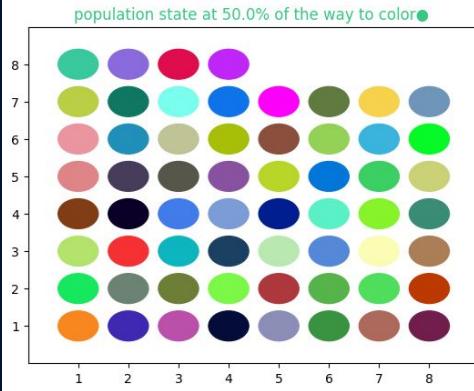
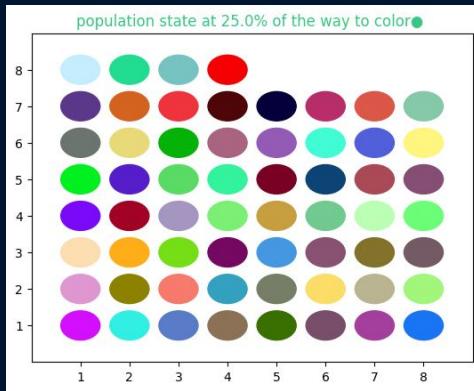
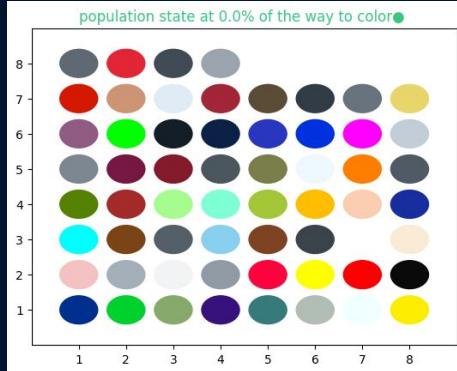


# Configuración del archivo de entrada

```
{  
  "colors_palette" : "colors_palette.txt",  
  "selection_algorithm": "roulette",  
  "mutation_rate": 0.2,  
  "max_generations": 90,  
  "expected_fitness": 0.95,  
  "population_number": 60,  
  "fitness_cut": true  
}
```

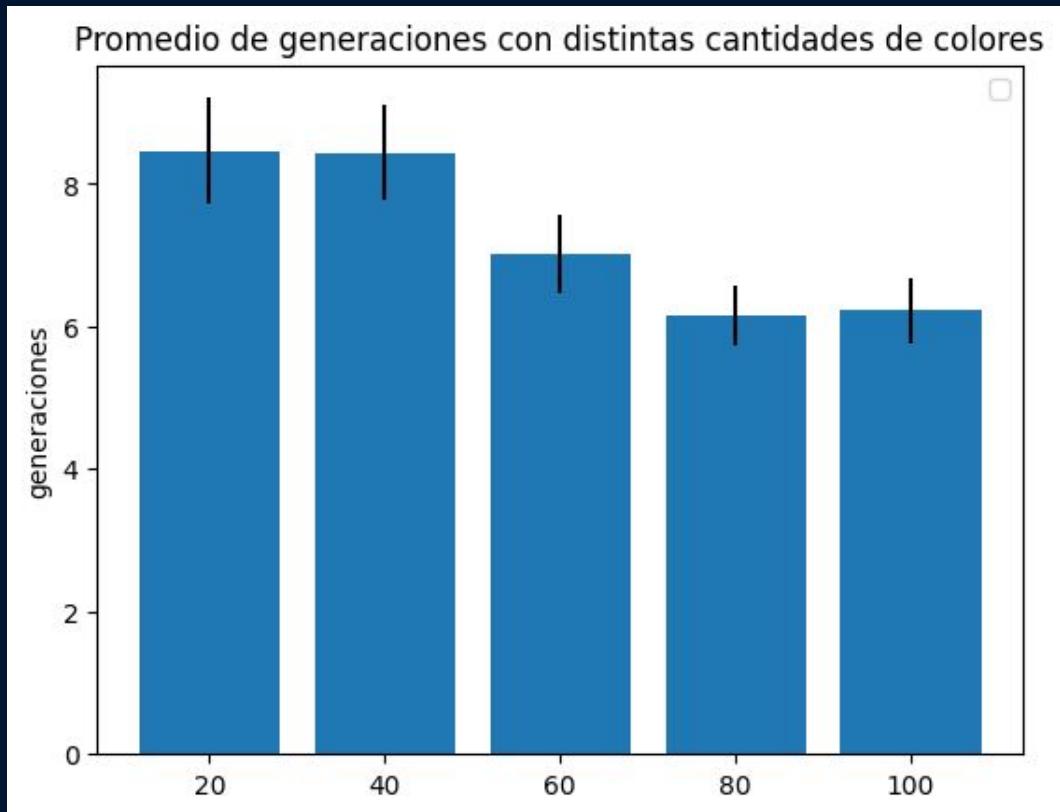
# Evolución de la paleta de colores hasta el color obtenido

Color objetivo: 56 200 131

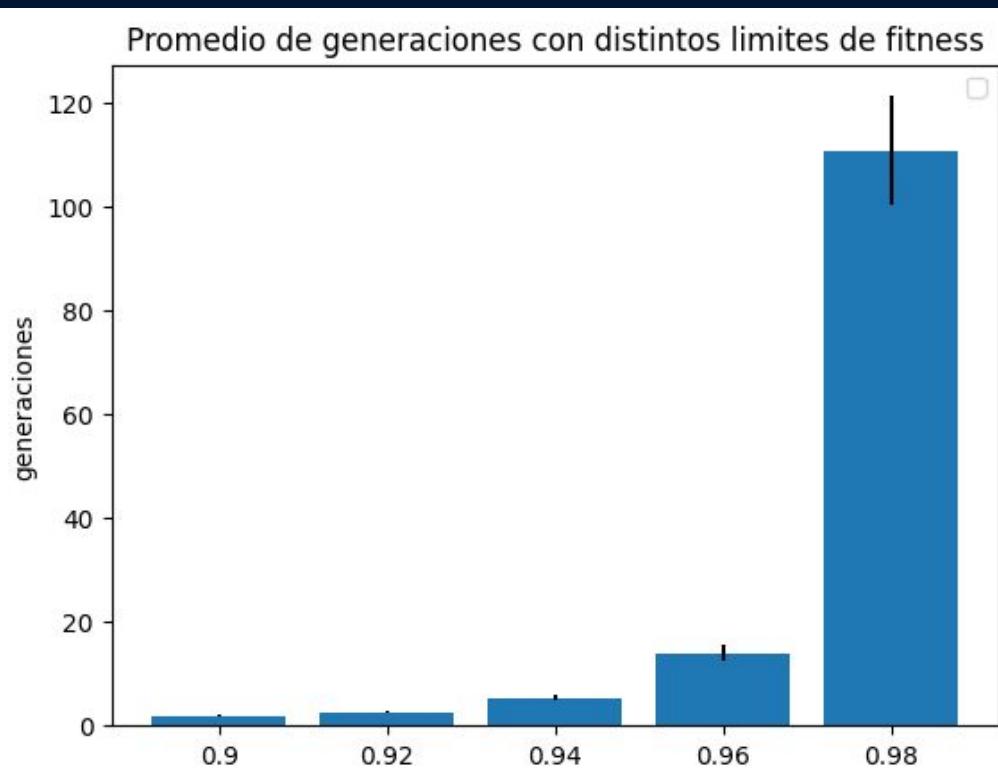


# Promedio de generaciones con distintas cantidades de población

Gracias al siguiente gráfico podemos observar que: a mayor población, mayor variedad de cromosomas, mayor beneficio para la evolución de las generaciones. Por lo que con una población mayor, menos cantidad de generaciones para llegar al objetivo.



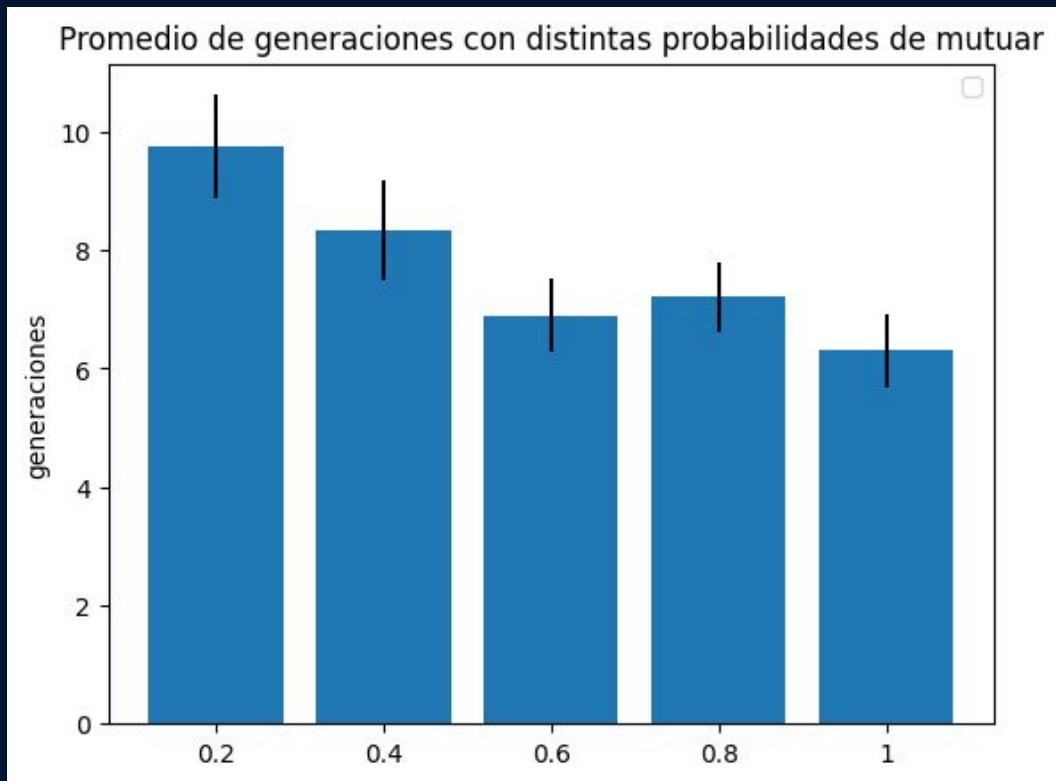
# Promedio de generaciones con distinto límite fitness



Está claro por el gráfico, que la cantidad de generaciones que deben pasar para llegar a mayor fitness aumenta exponencialmente al acercarse al 1.

# Promedio de generaciones con distintas probabilidades de mutación

Con este gráfico podemos concluir, que el aumento de  $P_m$  a la hora de analizar la mutación de un gen, presenta una mejora al sistema, pero muy menor.

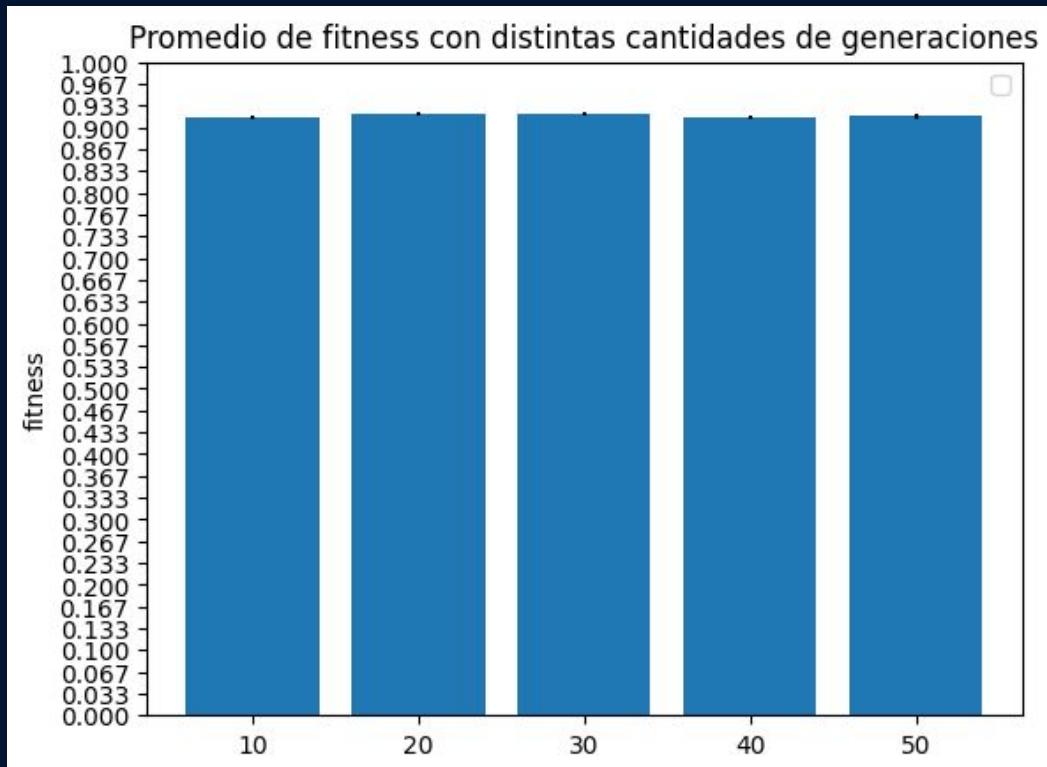


# Promedio de fitness con distintas cantidades generacionales

Como se puede observar, nuestro algoritmo nunca llega al ideal 100%, ya que en un punto aunque sigas iterando no se aproxima más.

Los valores exactos son:

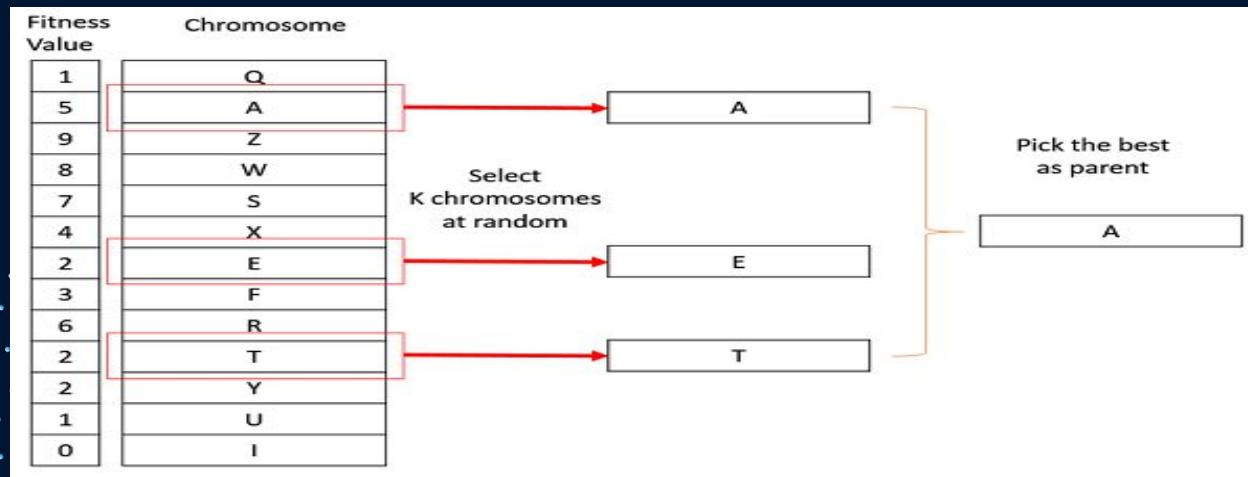
```
[0.9157729542759049,  
 0.9214908272879195,  
 0.9211090349942783,  
 0.9140557139984046,  
 0.917050526568967]
```



# Probabilistic Tournament

Se elige un valor de **Threshold** en  $[0.5, 1]$  aleatorio. De la población de tamaño **N**, se eligen 2 individuos al azar. Se toma un valor **r** al azar uniformemente en  $[0, 1]$ .

Si  $r < \text{Threshold}$  se selecciona el más apto. Caso contrario, se selecciona el menos apto. Se repite el proceso hasta conseguir los **K** individuos que se precisan.



# Resultados obtenidos

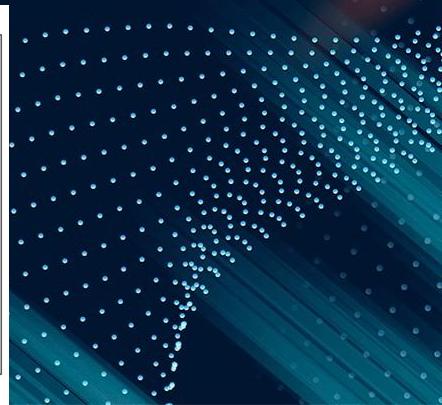
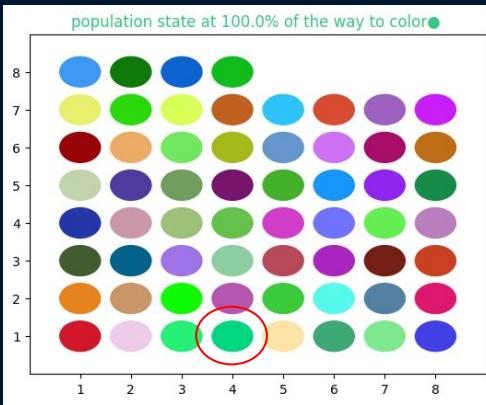
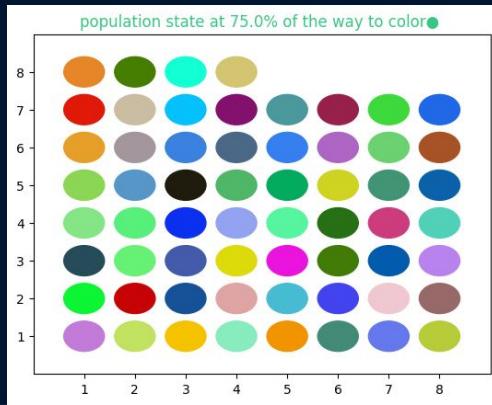
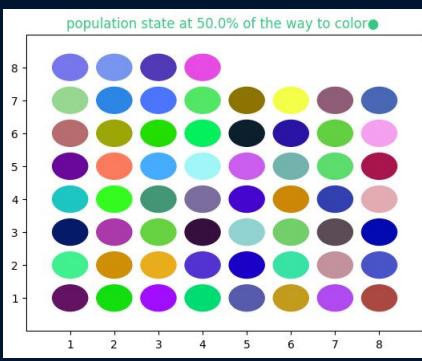
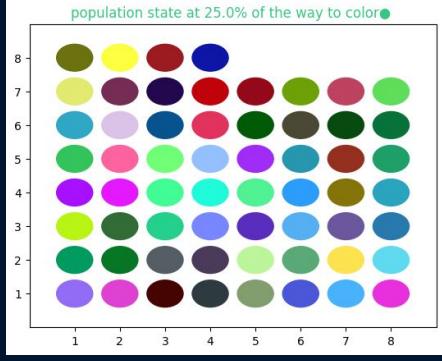
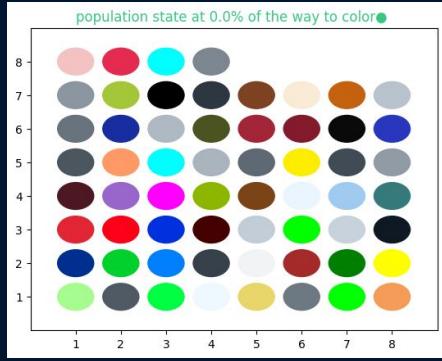


# Configuración del archivo de entrada

```
{  
  "colors_palette" : "colors_palette.txt",  
  "selection_algorithm": "probabilistic_tournament",  
  "mutation_rate": 0.2,  
  "max_generations": 90,  
  "expected_fitness": 0.95,  
  "population_number": 60,  
  "fitness_cut": true  
}
```

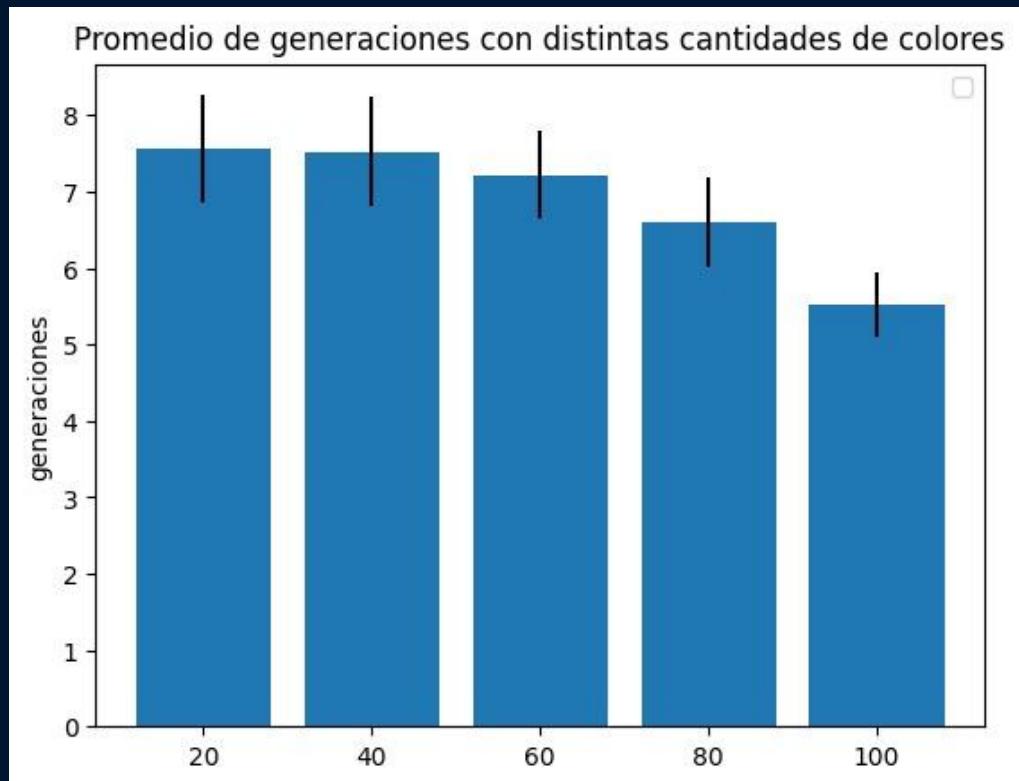
# Evolución de la paleta de colores hasta el color obtenido

Color objetivo: 56 200 131

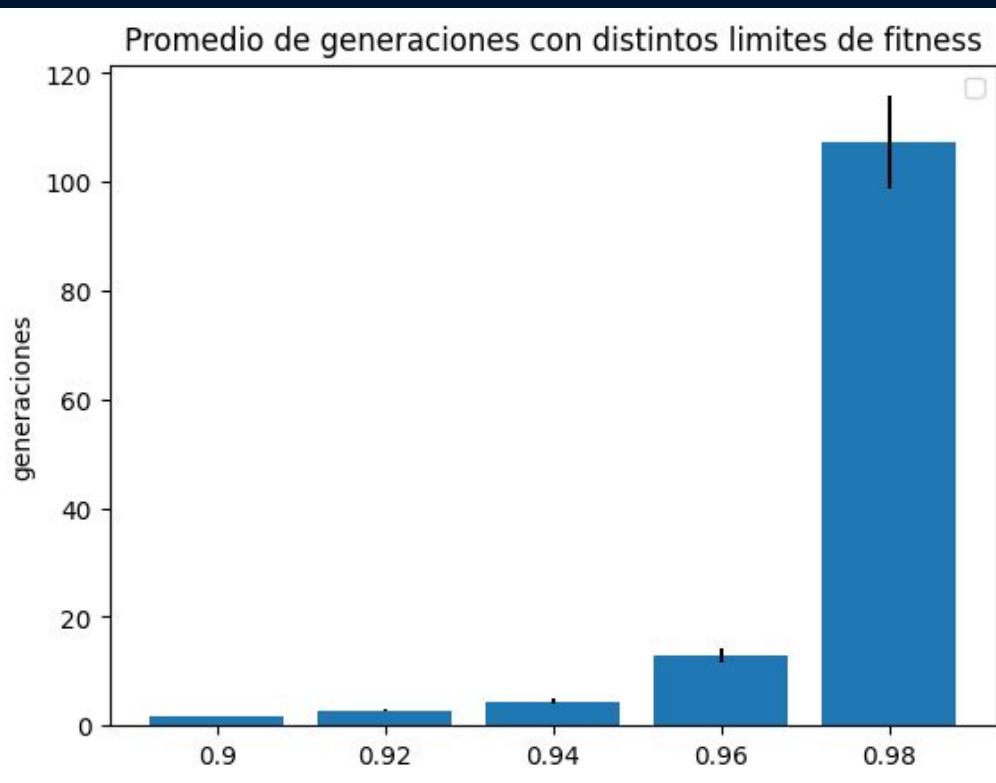


# Promedio de generaciones con distintas cantidades de población

Gracias al siguiente gráfico podemos observar que: a mayor población, mayor variedad de cromosomas, mayor beneficio para la evolución de las generaciones. Por lo que con una población mayor, menos cantidad de generaciones para llegar al objetivo.



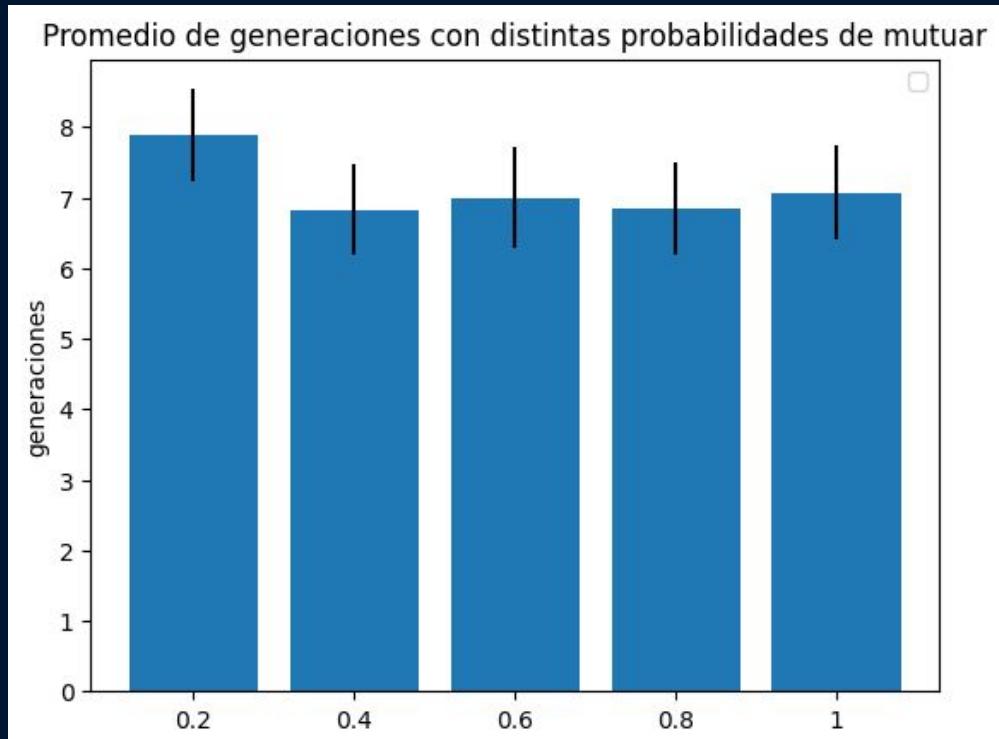
# Promedio de generaciones con distinto límite fitness



Está claro por el gráfico, que la cantidad de generaciones que deben pasar para llegar a mayor fitness aumenta exponencialmente al acercarse al 1.

# Promedio de generaciones con distintas probabilidades de mutación

Con este gráfico podemos concluir, que el aumento de  $P_m$  a la hora de analizar la mutación de un gen, presenta una mejora al sistema, pero muy menor.

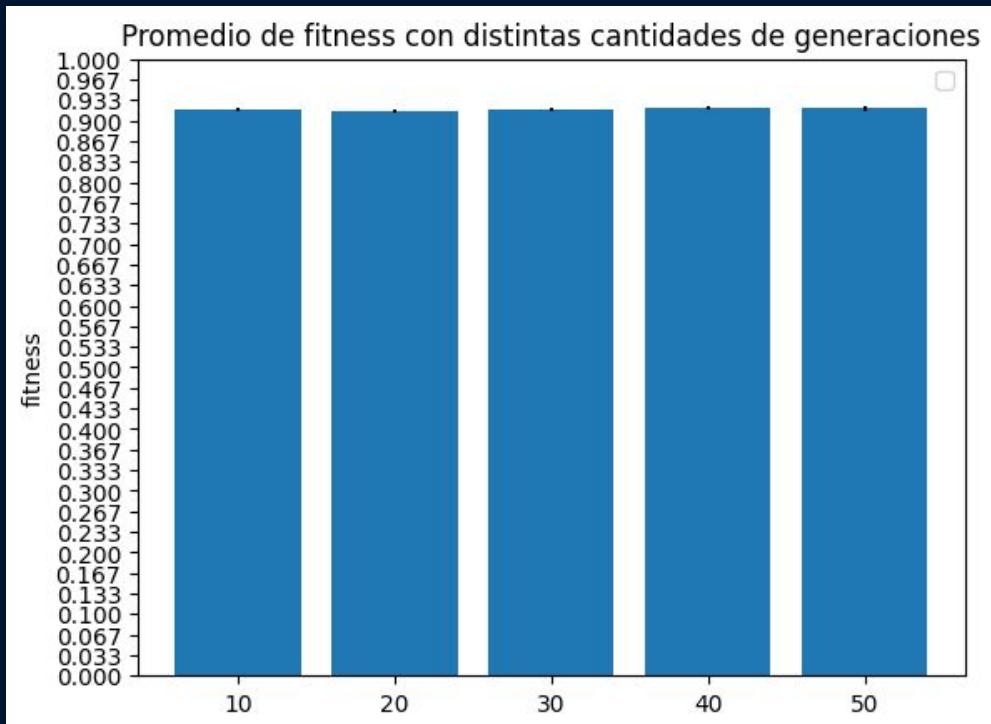


# Promedio de fitness con distintas cantidades generacionales

Como se puede observar, nuestro algoritmo nunca llega al ideal 100%, ya que en un punto aunque sigas iterando no se aproxima más.

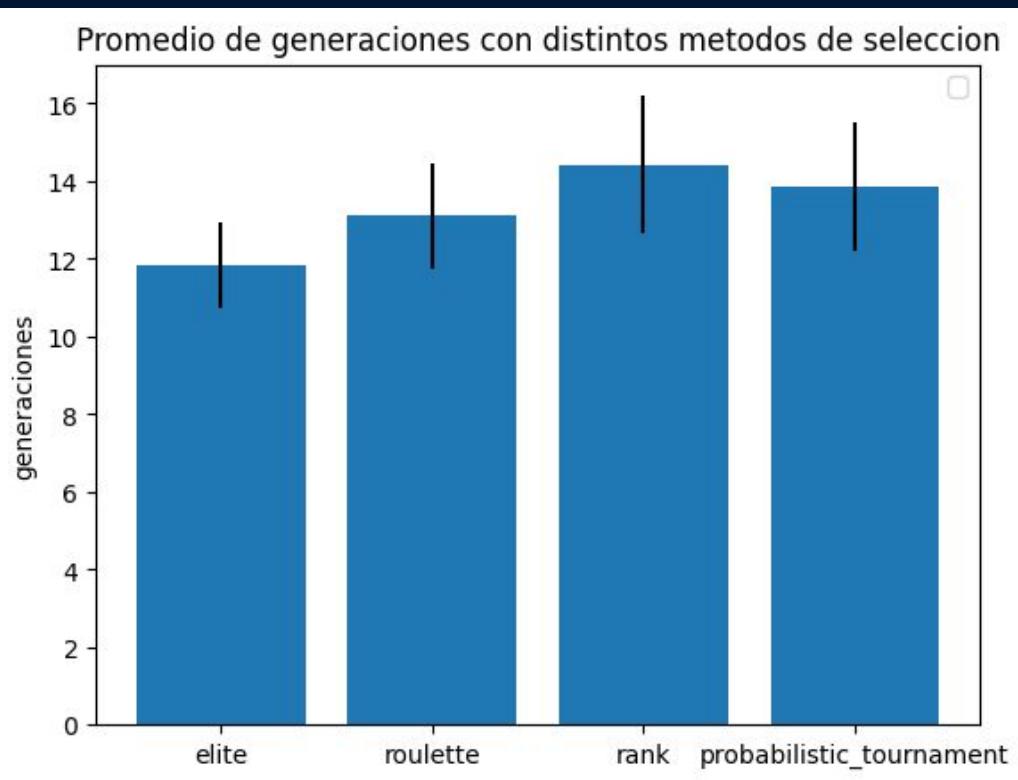
Los valores exactos son:

```
[0.9182801156197234,  
0.9155425910803308,  
0.9175608372202094,  
0.9200429212656889,  
0.9194510393876851]
```



# Conclusión

# Conclusión



En este caso las selecciones de élite y de roulette, demostraron llegar a el resultado más rápidamente. Pero no hay que dejar de tener en cuenta el riesgo de convergencia prematura que estos métodos presentan.

# Conclusión

Con este trabajo práctico, pudimos ver de forma experimental cómo afectan a un algoritmo genético la variación de los hiperparametros, como la cantidad de la población, los distintos cortes, la probabilidad de mutación, además de las diferencias entre los diferentes métodos de selección.



Fin.