

2023-24

# Random Peace - Documentación



Guillermo Dylan Carvajal Aza

Santiago Fernández Carballal

Universidad de Oviedo

Informática Audiovisual

## Contenidos

Introducción .....	2
1. Resumen del proyecto .....	2
1.1. Modo Usuario – IA .....	2
1.2. Modo Multiusuario.....	3
1.3. Modo Autómata.....	4
1.3.1. Reglas del Autómata Celular .....	4
1.4 La pantalla de inicio.....	5
2. Arquitectura .....	6
2.1. Infraestructura: White Box .....	6
2.2. Diagrama de Clases .....	7
2.2.1. BaseMode.....	7
2.2.2. AutomataMode.....	7
2.2.3. MultiUserMode.....	8
2.2.4. Paquete Util .....	8
2.3. La REST API .....	8
3. Recursos multimedia .....	8
4. Requisitos del sistema .....	9
5. Manual de instalación.....	9
5.1. Primer paso: alojar la web localmente.....	9
5.2. Segundo Paso : Alojar la REST API localmente .....	9
5.3 Conexión con la REST API local .....	10
6. Manual del usuario .....	11
7. Desafíos y soluciones .....	12
8. Mejoras futuras.....	13
9. Conclusión.....	14
10. Anexo .....	14
11. Referencias y Bibliografía.....	16

## Introducción

Random Peace es un proyecto que busca hacer un homenaje directo a la obra Random War de Charles Csuri. La idea principal es mandar un mensaje pacifista, contrario al que se propone en la obra del autor original.

### 1. Resumen del proyecto

La idea de la obra es reivindicar un mensaje de paz frente al actual panorama global, dando un giro a esta obra del año 1968, pero manteniendo sus conceptos básicos.

Pese a que no era la idea original del autor, la obra era vista como una manera de mejorar el avance militar mediante ordenador.

*“One could introduce military intelligence reports into the programme with an estimate of the enemies' capabilities and the tactics they may use.[...] Once the real battle starts, the computer can predict the outcome and its consequences many hours before the battle ends. [...] The military computer could process one per cent of each of the variables and predict the outcome. [...]” [1]*

Este enfoque, de índole militarista, es algo que a día de hoy es una realidad, los ordenadores son capaces de procesar cantidades masivas de datos, y muchos de los cuerpos de inteligencia de diferentes países (por no decir todos), emplean técnicas predictivas mediante ordenadores para apoyar sus tácticas militares.

*“These kinds of AI decision support systems (AI-DSS) are computerised tools that use AI software to display, synthesise and/or analyse data and in some cases make recommendations – even predictions – in order to aid human decision-making in war.” [2]*

Frente a esta idea del uso de la tecnología nosotros proponemos una idea tal vez un tanto optimista. De la misma manera en la que se utilizan estos ordenadores para predecir y elaborar estrategias, se puede “calcular” la paz mediante ordenador, o dicho de otra manera, es posible emplear estas técnicas de predicción por ordenador no solo para engendrar la guerra, sino para acabar con ella.

*“It is not enough to win a war; it is more important to organize the peace.”  
— Aristotle*

El funcionamiento del proyecto es bastante simple, al usuario se le presenta una pantalla en blanco en la que debe colocar sus figuras, dependiendo del modo que decida utilizar, estas interactuarán de forma diferente.

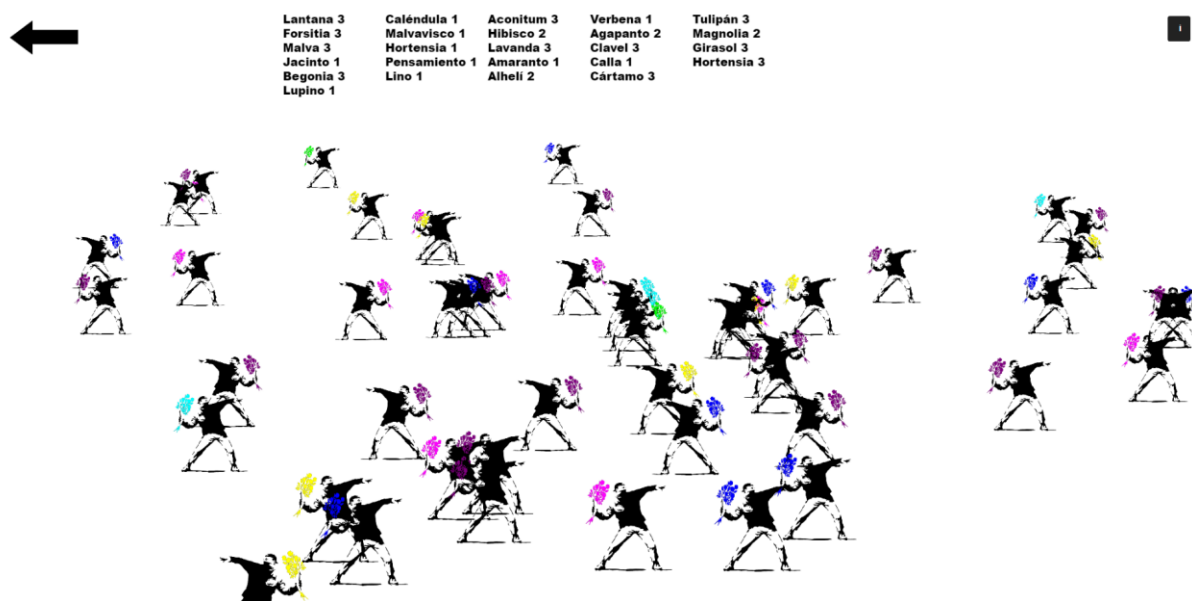
#### 1.1. Modo Usuario – IA

En este modo, una vez el usuario coloca todas sus figuras, el ordenador coloca las suyas de manera aleatoria, una vez ambos “bandos” tienen sus figuras colocadas se presenta en la parte superior de la pantalla un conjunto de flores seguidas del número de veces que se “han lanzado” estas. Este modo de aplicación es el que más se aproxima a la obra original de Charles Csuri, en la que los soldados se

colocaban de manera aleatoria por el ordenador y luego este mismo calculaba sus nombres, rangos y estado tras la "guerra".

Cuando el usuario toca la pantalla, sus figuras se colocan de manera aleatoria en un rango determinado alrededor del punto en el que haya hecho clic. Esta decisión fue tomada para darle menos control al usuario sobre lo que está haciendo, pretendiendo que no sea capaz de entender como está interactuando con la aplicación, aportando así aún más aleatoriedad a la obra. Además de esto, el color de cada ramo de cada figura es aleatorio con respecto al resto.

Las figuras de la "Inteligencia Artificial", son colocadas en la pantalla con la peculiaridad de que miran en dirección opuesta a las del usuario, esta decisión se tomó por la simple razón de que era más fácil para el usuario de entender que si todas las figuras mirasen en al misma dirección.



## 1.2. Modo Multiusuario

El modo multiusuario pretende conectar a diferentes usuarios sobre el mismo lienzo, de manera que estos puedan ver las figuras que el resto a colocado. Las figuras se colocan del mismo modo que el modo Usuario-IA, con la peculiaridad de que en este modo no hay texto con cada flor "lanzada" (así los usuarios tienen más espacio en la pantalla). Cada versión final con todas las figuras de cada usuario es diferente para cada jugador, ya que los colores de cada figura se recalculan aleatoriamente cuando un usuario recibe las posiciones de los demás, manteniendo así el toque de aleatoriedad.

Algo a destacar sobre este modo es la pantalla de carga, que utiliza otras obras de Banksy para representar a los cuatro jugadores que componen esta actividad online. La decisión de utilizar estas figuras para representar una pantalla de carga fue tomada por el hecho de haber logrado encontrar más de una obra de Banksy en formato SVG, se podría decir que estas figuras fueron "reutilizadas" desde su propósito original para poder formar una representación de unidad entre todos los usuarios.



4/4



### 1.3. Modo Autómata

El modo autómata busca replicar el comportamiento de un autómata celular, parecido a el Juego de la Vida de Conway <sup>[3]</sup>. El funcionamiento es bastante similar al modo "base", el usuario debe colocar sus figuras y, una vez haya colocado todas, estas empezarán a interactuar entre si siguiendo las reglas descritas.

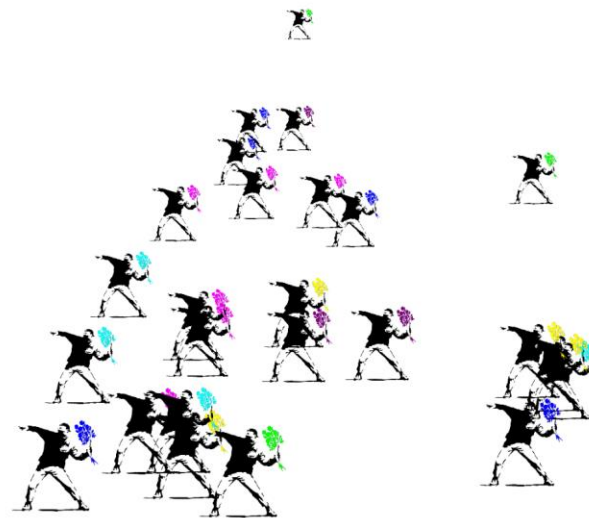
A parte de esto, cuenta también con la funcionalidad de poder parar el proceso de evolución del autómata, mostrando a su vez el número de la iteración actual en la que se encuentra.

Las reglas que sigue este no se corresponden con ningún algoritmo existente para autómatas celulares (a nuestro conocimiento).

#### 1.3.1. Reglas del Autómata Celular

El autómata toma como "vecinos" aquellas figuras que se encuentren a una distancia euclídea dentro del plano de menos de 300 (píxeles).

- Si la figura tiene 1 o 2 vecinos se mantiene "viva", sin alterar el plano.
- Si hay 3 vecinos, entonces la figura se cambia por su vecino más cercano.
- En cualquier otro caso, la figura se desplaza por la pantalla de manera aleatoria, dentro de un rango de entre 100 y 50 píxeles (hacia todas las direcciones).



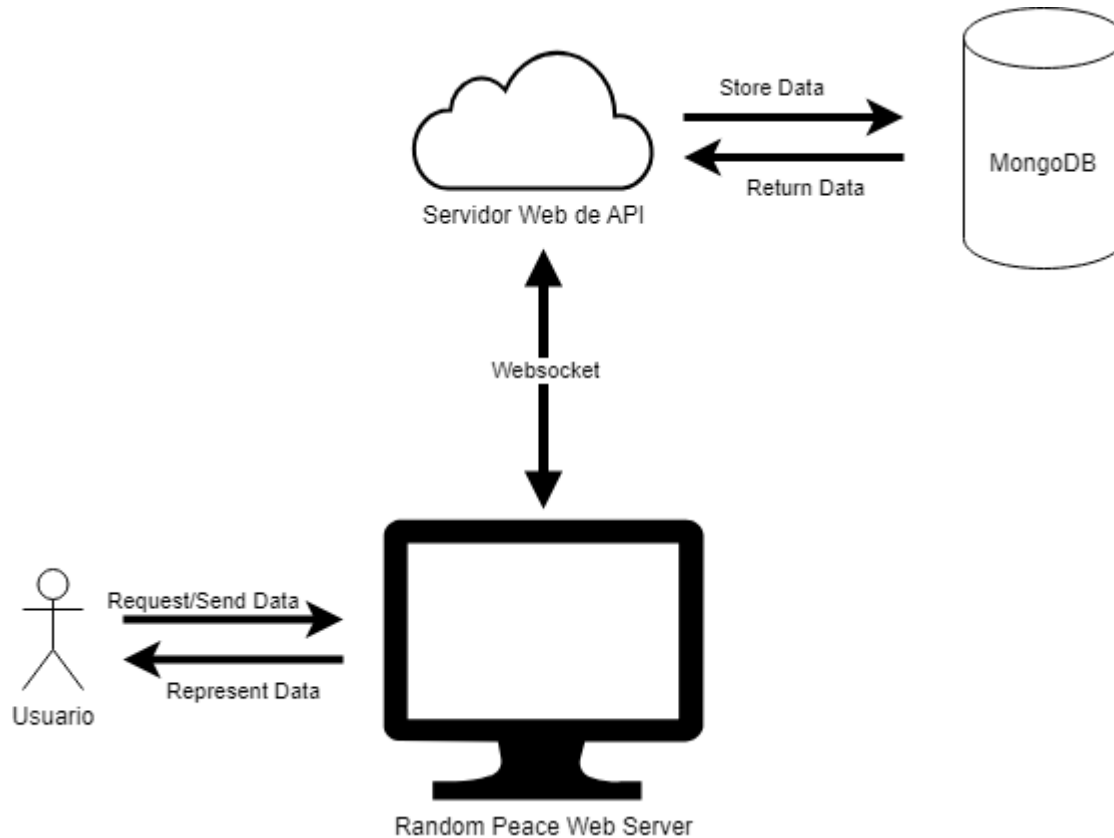
#### 1.4 La pantalla de inicio

Pese a no ser un modo de uso, y solo ser la portada de la aplicación, cabe mencionar que los colores de la pantalla de inicio están seleccionados de forma que resalten el color del ramo de la figura, el color predominante es el blanco (color asociado con la paz), y resalta sobre ella el título y botones negros para contrastar con el fondo y cuando se pasa por encima de una de las opciones, remarcándola en azul (otro color asociado con sentimientos como la relajación y la sinceridad). Un pequeño detalle es que la posición y color de la figura varía cada vez que se visita la pantalla (sin salir de la misma sesión).

## 2. Arquitectura

El proyecto se basa en una arquitectura web, con un acceso a una base de datos a través de una API y un Websocket.

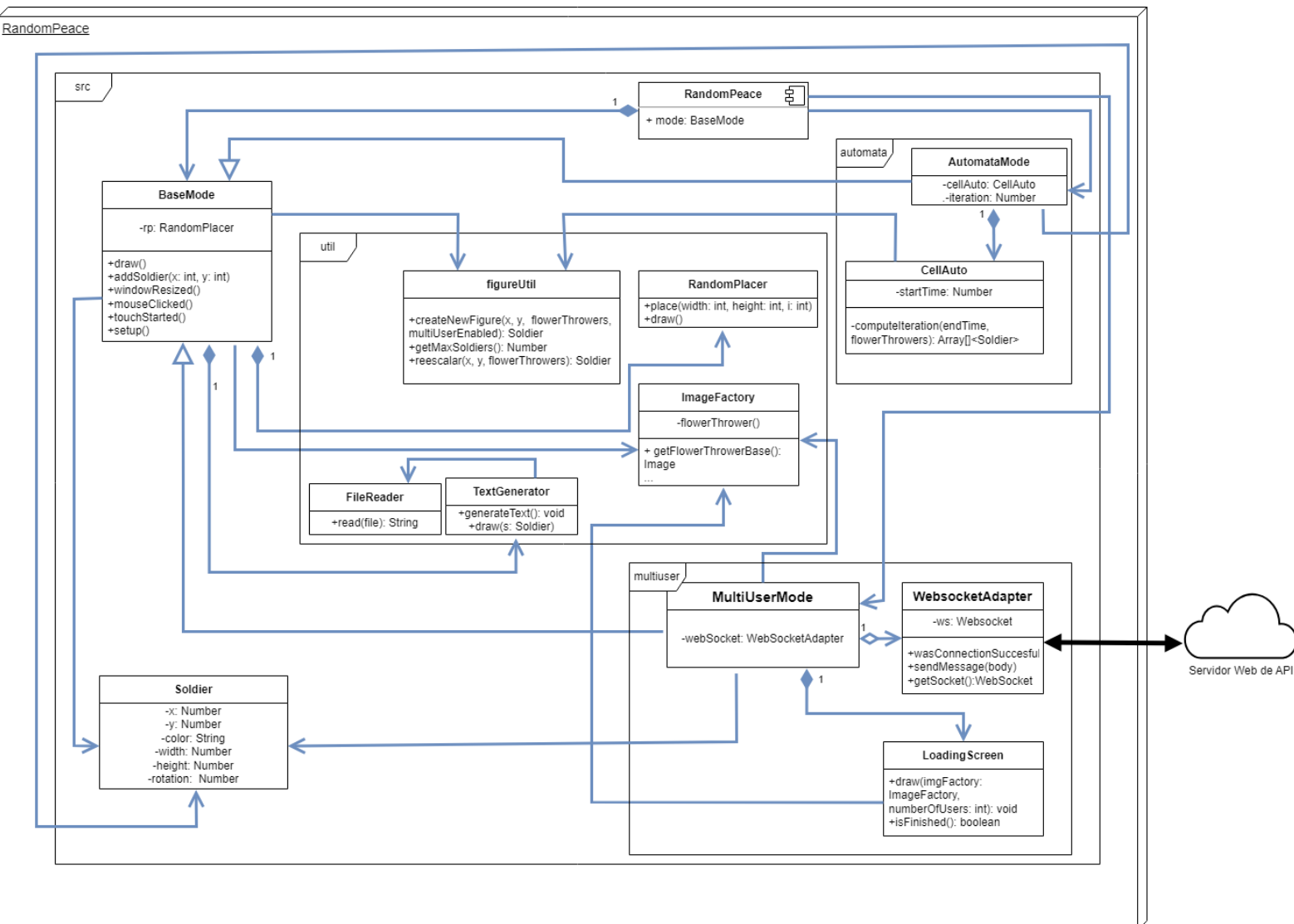
### 2.1. Infraestructura: White Box



Los usuarios se conectan a la aplicación a través de su navegador web, todo el código de los modos en los que solo hay un usuario es manejado en el propio cliente del usuario, pero para el modo multiusuario se hace uso de un servidor web de api que proporciona un enlace con una base de datos (en MongoDB) a través de un Websocket.

## 2.2. Diagrama de Clases

El sistema cuenta con un módulo base (llamado Random Peace), que maneja el comportamiento global de la aplicación. El resto de clases se organizan en diferentes carpetas, excepto por el modo Usuario-IA, llamado *BaseMode*, y la clase que maneja los atributos de las figuras en la pantalla (denominada *Soldier*, en “homenaje” a la obra original). Las tres clases principales son las que definen los modos de la aplicación: *BaseMode*, *MultiUserMode* y *AutomataMode*.



### 2.2.1. BaseMode

La clase del modo Usuario-IA, es la más básica de todas y actúa como “superclase” para el resto de modos, que heredan métodos de ella. Todo el código correspondiente a esta clase ha sido creado por el equipo.

### 2.2.2. AutomataMode

Hace uso de la clase *CellAuto* para poder generar las iteraciones del Autómata Celular. El código de este autómata ha sido desarrollado por el equipo, con ayuda de diferentes proyectos basados en Javascript como cellauto <sup>[3]</sup> o este código de Higland Solutions <sup>[4]</sup>.



### 2.2.3. MultiUserMode

Esta clase es la encargada del modo multiusuario, para ello, hace uso de un WebSocketAdapter, que encapsula el comportamiento de la clase WebSocket <sup>[5]</sup> propio de Javascript. La clase intenta conectar con el servidor, si no es capaz, muestra un mensaje, y de otra manera permite al usuario acceder a la sesión. El código para el multi usuario ha sido desarrollado en su totalidad por el equipo, exceptuando ciertos aspectos como las peticiones a la REST API, que se implementaron basándose en varios ejemplos de internet.

### 2.2.4. Paquete Util

Todas las clases incluidas en este paquete han sido desarrolladas por el equipo, sirven como clases de apoyo encapsulando funcionalidad que es reutilizada o que era demasiado pesada para ser añadida en las clases principales.

Como clase destacable está el FigureUtil, clase que tuvo que crearse para manejar la carga de las imágenes para que el resto de clases no consumiesen recursos cargándolas constantemente, esta clase es utilizada como un Singleton <sup>[6]</sup> de manera global, cargando las imágenes cuando se inicia la ejecución del programa. Las únicas imágenes que se cargan a petición del resto de clases son las del Flower Thrower con diferentes colores, esto se debe a que, pese a que se podrían tener cargadas desde un inicio, no consumen demasiados recursos como para tenerlas siempre cargadas desde un inicio, además de que fue un factor que facilitó el desarrollo del resto del código.

## 2.3. La REST API

Este servicio de conexión a la base de datos está basado en Python, utilizando el framework de desarrollo de APIs FastAPI <sup>[7]</sup>, que agiliza el proceso de creación de este tipo de contenidos web. La REST API interactúa con la base de datos, desplegada en MongoDB, en la que se guardan los usuarios que se encuentran en la sesión actual con las posiciones de sus figuras.

## 3. Recursos multimedia

Los principales recursos multimedia de este proyecto provienen de las imágenes empleadas para “reemplazar” a los soldados de la obra original, estas son una versión modificada por nosotros de la obra Flower Thrower <sup>[8]</sup> de Banksy, cambiando el color del ramo de flores, este cambio pretende cambiar el significado de la obra original, en ella todos los soldados de un bando son de color negro, mientras que los del otro son de color rojo, nosotros decidimos que, en cada bando, haya un número aleatorio de flores de cada color, así, una vez completa la obra, el usuario no es capaz de diferenciar entre sus piezas y las del resto.

Además de esta imagen, también hemos utilizado otras dos obras del mismo autor: “Butterfly Suicide Girl” y “CND Soldiers” <sup>[9]</sup>.

Somos conscientes del uso de estas imágenes y nos comprometemos a cumplir con la política de copyright del autor respecto a sus imágenes explicada en su página web <sup>[10]</sup> <sup>[11]</sup>.

## 4. Requisitos del sistema

Para

Versiones de bibliotecas utilizadas:

- **annotated-types** 0.6.0
- **anyio** 3.7.1
- **click** 8.1.7
- **dnspython** 2.4.2
- **fastapi** 0.104.1
- **h11** 0.14.0
- **idna** 3.4
- **motor** 3.3.1
- **pydantic** 2.4.2
- **pydantic-core** 2.10.1
- **pymongo** 4.5.0
- **sniffio** 1.3.0
- **starlette** 0.27.0
- **typing-extensions** 4.8.0
- **uvicorn** 0.23.2
- **websockets** 12.0

Versiones de software utilizados:

- **Python:** 3.10.1

## 5. Manual de instalación

Para poder instalar el proyecto en un ordenador local, deberemos seguir los siguientes pasos:

### 5.1. Primer paso: alojar la web localmente

Primero debemos bajarnos el proyecto desde la url de Github o clonando el repositorio localmente.

Una vez lo tengamos en nuestro ordenador, desde la carpeta base o desde **src** ejecutamos el siguiente comando:

```
python -m http.server
```

O en caso de que utilizamos python3:

```
python3 -m http.server
```

Con esto ya tendremos la aplicación ejecutándose localmente en la dirección: *localhost:8000*, a la que podremos acceder desde nuestro navegador.

### 5.2. Segundo Paso : Alojamiento de la REST API localmente

Este paso es opcional, y solo se debe llevar a cabo si no se desea hacer uso de la REST API desarrollada por el equipo, y alojada en un servidor web, a la que el proyecto accede por defecto.

Accedemos a la carpeta `/server` mediante el comando:

```
cd server
```

Y a continuación ejecutamos los siguientes comandos:

```
python -m venv serverEnv
.\serverEnv\Scripts\activate
pip install -r .\serverEnv\requirements.txt
```

Opcionalmente y si el instalador lo sugiere ejecutamos también:

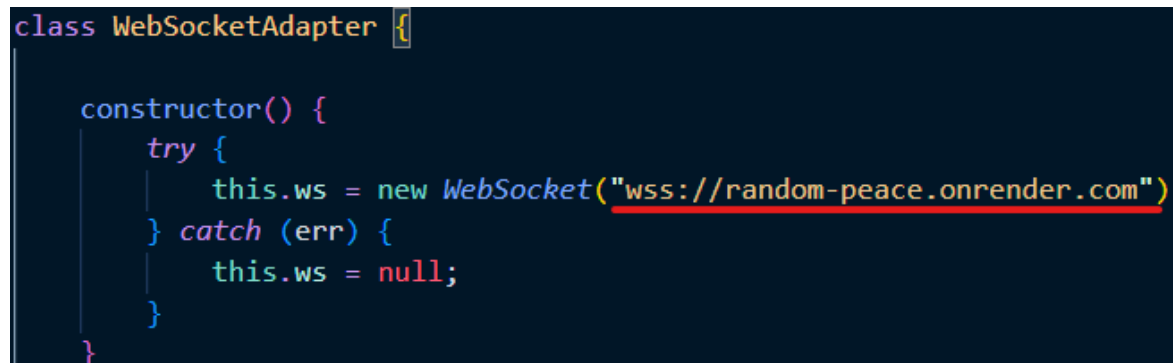
```
python.exe -m pip install --upgrade pip
```

Finalmente, para poner en marcha la REST API ejecutamos el siguiente comando sin salir de la carpeta:

```
uvicorn main:app --reload
```

### 5.3 Conexión con la REST API local

Para poder conectarnos con la API que acabamos de lanzar, es necesario cambiar la dirección IP de la clase `WebSocketAdapter` (que se encuentra en `src/multiuser/websocket.js`). Aquí debemos escribir dentro de los `""` la dirección en la que alojamos la REST API (por lo general, esta se aloja en `ws://127.0.0.1:8000/`, en caso de duda siempre podemos comprobarlo a través de la terminal)



```
class WebSocketAdapter {
  constructor() {
    try {
      this.ws = new WebSocket("wss://random-peace.onrender.com")
    } catch (err) {
      this.ws = null;
    }
  }
}
```

También deberemos especificar, dentro de las variables locales, una conexión a nuestra base de datos en MongoDB. Para ello necesitaremos cambiar la siguiente línea especificada por nuestra URL de acceso (entre `""`). También se podría establecer la URL como una variable de entorno en Windows con el nombre `MONGODB_URL`, (se recomienda utilizar la primera opción siempre que se vaya a utilizar el local por simplicidad).

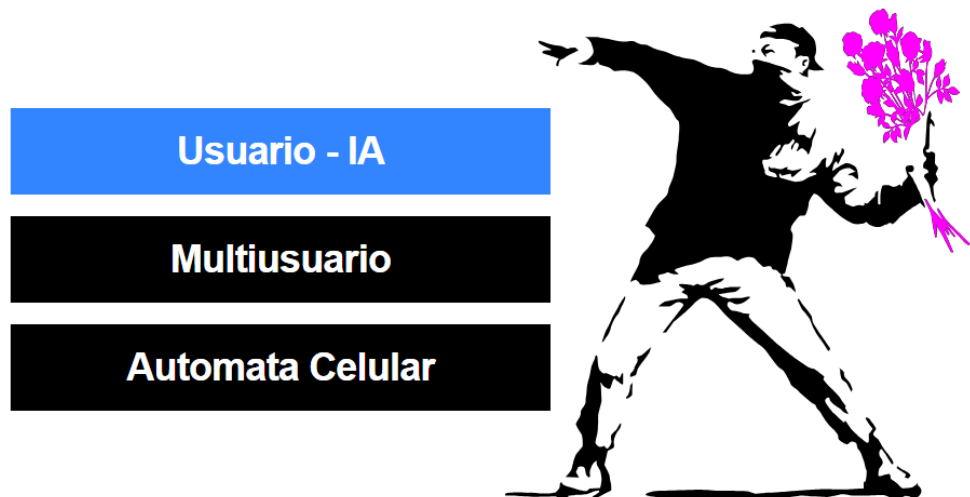
```
# Creación de la aplicación
app = FastAPI(title="RandomPeace API",summary="A simple API for the RandomPeace application")

client = motor.motor_asyncio.AsyncIOMotorClient(os.environ["MONGODB_URL"])
db = client.get_database("randompeace")
users_collection = db.get_collection("users")
```

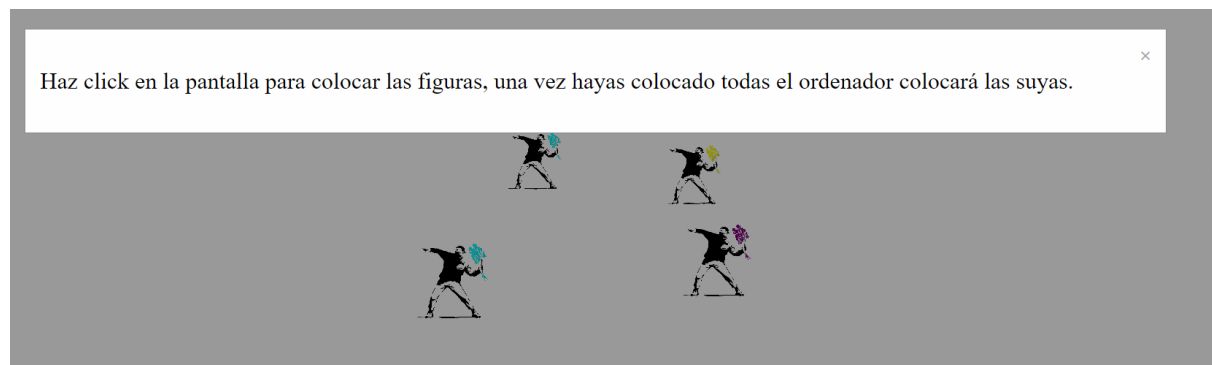
## 6. Manual del usuario

El primer paso para comenzar con Random War es seleccionar el modo que se desea probar de entre los tres: Usuario-IA, Multiusuario, Automata. Solo con hacer clic en uno de los tres botones comenzará a ejecutarse el elegido.

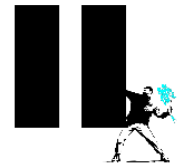
# Random Peace



La aplicación proporciona instrucciones en todo momento, ya sea mediante la pantalla o haciendo clic en el botón de información que se encuentra arriba a la derecha.



En el caso de seleccionar el autómata celular, se podrá utilizar cualquier tecla del teclado para pausar la ejecución del autómata, permitiendo ver la iteración en la que se encuentra este.



### 7. Desafíos y soluciones

El problema principal para este proyecto fue, inicialmente, el enfoque que se le dio para poder transmitir un mensaje que difiriese de aquel de la obra original, ya que en un principio se pretendía simplemente realizar una versión “actual” de la obra de Charles Csurí. Esta idea original se replanteó y derivó en una versión pacifista, más adaptada a la sociedad actual, teniendo en cuenta los recientes eventos trágicos, como las guerras, que han ocurrido en todo el mundo.

Uno de los principales problemas cuando se comenzó el proyecto fue el tratado de imágenes, para empezar, solo contábamos con una imagen en SVG, en color negro, del Flower Thrower de Banksy. Esto supuso un problema a la hora de cambiar el ramo de color, ya que la idea original era utilizar la función *tint()* de P5JS, pero al estar completamente en negro no cambiaba el color de la imagen. Por lo tanto, se acabó decidiendo cambiar el color de las imágenes utilizando un programa externo, en este caso *InkScape*.

Otro de los problemas del proyecto fue el desarrollo de la REST API, ya que al principio se planteó desplegarla en un servidor como Azure o AWS, pero para agilizar el proyecto, se acabó optando por utilizar la plataforma Render, especializada en este tipo de servicios webs que utilizan FastAPI.

Aunque no fue un desafío demasiado obstruyente, el algoritmo que determina las reglas del autómata celular, así como el resto del código necesario para su funcionamiento, supuso una serie de intentos con diferentes reglas hasta poder dar con una que funcionase. Esto se debe a la diferencia principal entre los autómatas celulares clásicos y nuestra aplicación, estos autómatas utilizan rejillas con cuadrículas marcadas, teniendo así cada una un estado (vivo/muerto), pero nosotros no podíamos hacer esto. Una pantalla de alta definición de un ordenador tiene 1.280 x 720 píxeles, habría sido demasiado separar todas y cada una de ellas en una matriz y asignarle a cada pixel un estado, así que, en vez de esto, lo que se hizo fue utilizar las propias figuras como el estado y colocarlas en la pantalla en función de su posición. Así es más simple controlar cuantas figuras hay en pantalla, dónde se encuentran y cuáles son sus vecinos.

## 8. Mejoras futuras

Como posibles expansiones para el proyecto, se podría considerar realizar más de un algoritmo para el autómata celular, ofreciendo al usuario la capacidad de cambiar entre ellos durante la evolución del autómata o antes de su ejecución. Se podrían utilizar algoritmos clásicos como el Juego de la Vida de Conway <sup>[12]</sup>, Day and Night <sup>[13]</sup> o el autómata celular de Von Neumann <sup>[14]</sup>.

Otras de las posibles mejoras sería aumentar el número de usuarios del Multiusuario, pudiendo hacer que más usuarios participen en una sesión o que se llevasen a cabo varias sesiones con varios usuarios, ya que actualmente solo se permite una sesión activa a la vez. Sobre el multiusuario también sería muy posible y bastante productivo, como mejora, extraer la configuración de direcciones IP/URLs a un fichero, en vez de introducirlas mediante código.

También se podría mejorar el comportamiento de las figuras cuando se posicionan en el plano, tanto por parte del usuario como por parte del ordenador. Actualmente las figuras colocadas en el plano solo constan de una posición (aleatorizada) y un color, pero no tienen una rotación aleatoria (como en la obra original), para darle más aleatoriedad a la interacción con el usuario, (esto en realidad se consideró para el proyecto, se puede ver reflejado en uno de los parámetros de la clase *Soldier*, pero no se logró llegar a implementar).

Otra de las muchas mejoras que se podrían llevar a cabo a futuro sobre el proyecto sería la generación de texto de la versión "base" (Usuario-IA). Estaría bien replantear la idea de otra manera, ya que las flores, pese a tener sentido gracias a la figura del Flower Thrower, no aportan demasiado contexto a la obra, como si hacía en la obra original, en la que se mostraban el nombre, rango y estado de cada soldado que se encontraba en el lienzo. Nosotros en cambio, mostramos un número aleatorio de flores por cada soldado que se encuentra en la pantalla, como si perteneciesen al ramo, pero no estaría mal plantear esta cuestión de alguna otra manera.

Otro aspecto que se podría plantear sería la adaptabilidad de la aplicación a distintos tipos de dispositivos, ya que pese a funcionar parcialmente en dispositivos móviles y tablets, la mayoría de las funcionalidades no se pueden ejecutar como en los ordenadores, además de no tener acceso a el "lienzo" completo por ser dispositivos con una resolución más pequeña, lo cual no se tuvo en cuenta cuando se comenzó a desarrollar el proyecto. Por tanto, al no estar inicialmente pensado para poder usarse en dispositivos de este tipo, sería bastante interesante poder llevar la aplicación a ellos.

Por último, como expansión, se podría pensar en incluir otro modo, tal vez uno en el que el usuario no tenga interacción y las posiciones de todas las figuras sean colocadas por ordenador aleatoriamente (como en la obra original). Se podría basar este concepto en la idea de los Juegos de Cero Jugadores <sup>[15]</sup>, que son aquellos juegos en los que no hay jugadores humanos.

## 9. Conclusión

Este proyecto nos ha ayudado a desarrollar nuestras competencias con el contenido multimedia, en este caso las imágenes, aprendiendo a cómo tratar con ellas en un entorno web mediante lenguajes como P5JS. Además de esto, nos ha ayudado a expandir nuestro conocimiento sobre diversos temas informáticos como el funcionamiento de los autómatas celulares, habiendo programado uno para este trabajo hemos visto como desarrollar funciones matemáticas de una manera eficiente.

Además de esto, nos ha ayudado a aprender a buscar una manera de ver una idea desde “fuera de la caja”, y a darle un enfoque distinto para hacer que sea más respetuosa y solidaria con el mundo actual.

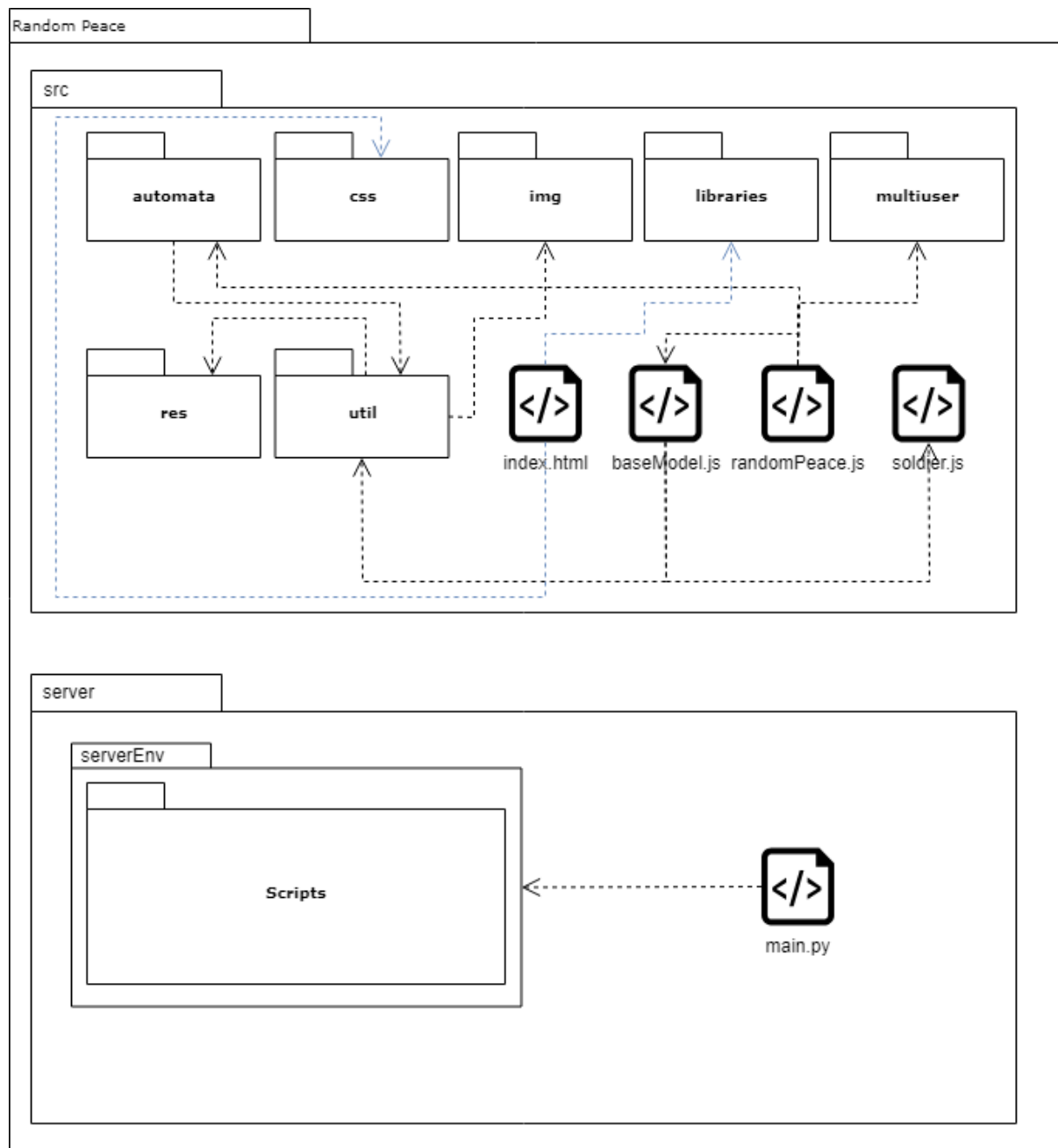
Como mayor logro cabe destacar haber logrado tener un producto elaborado y eficiente, no solo cumpliendo nuestras expectativas, sino superándolas.

## 10. Anexo

El proyecto se organiza en varias subcarpetas, siendo la principal *src*, donde se encuentra todo el código del cliente que se ejecuta para desplegar la página web.

Dentro de la carpeta *server*, se encuentra el código necesario para poner en marcha la REST API, el archivo *main.py* es el encargado de ejecutar este servicio web. El resto de las carpetas contenidas en esta son librerías y scripts de ejecución del archivo de Python.

En el siguiente diagrama de paquetes se pueden ver las dependencias entre las diferentes carpetas y scripts. No se han incluido el resto de las relaciones del archivo *index.html*, ya que este es el que contiene y sirve en la página web el resto de los archivos, estando así conectado con todos.





## 11. Referencias y Bibliografía

- [1] J. Reichard, «An interview with Charles Csuri,» *Cybernetic Serendipity: Comput. Arts*, p. pp. 81–84, July 1968 .
- [2] H. L. & P. B. «Algorithms of war: The use of artificial intelligence in decision making in armed conflict,» [En línea]. Available: <https://blogs.icrc.org/law-and-policy/2023/10/24/algorithms-of-war-use-of-artificial-intelligence-decision-making-a>.
- [3] «Cellauto.js,» [En línea]. Available: <https://sanojian.github.io/cellauto/>.
- [4] H. «Cellular automata in javascript - highland,» [En línea]. Available: <https://www.highlandsolutions.com/insights/cellular-automata-in-javascript>.
- [5] M. W. Docs, «WebSocket - Web APIs | MDN,» [En línea]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/WebSocket>.
- [6] «Singleton Pattern,» [En línea]. Available: [https://en.wikipedia.org/wiki/Singleton\\_pattern](https://en.wikipedia.org/wiki/Singleton_pattern).
- [7] S. Ramírez, «FastAPI,» [En línea]. Available: <https://fastapi.tiangolo.com/>.
- [8] W. t. f. e. Contributors to Wikimedia projects, «Flower thrower - wikipedia,» [En línea]. Available: [https://en.wikipedia.org/wiki/Flower\\_Thrower](https://en.wikipedia.org/wiki/Flower_Thrower).
- [9] B. E. «CND soldiers, 2005 - banksy explained,» [En línea]. Available: <https://banksyexplained.com/cnd-soldiers-2005/>.
- [10] Banksy, «Banksy,» [En línea]. Available: <https://www.banksy.co.uk/licensing.html>.
- [11] P. C. O. «Pest Control Office.,» [En línea]. Available: <https://pestcontroloffice.com/use.asp>.
- [12] t. f. e. Wikipedia, «Conway's game of life - wikipedia,» [En línea]. Available: [https://en.wikipedia.org/wiki/Conway's\\_Game\\_of\\_Life](https://en.wikipedia.org/wiki/Conway's_Game_of_Life).
- [13] t. f. e. Wikipedia, «Day and Night (cellular automaton),» [En línea]. Available: [https://en.wikipedia.org/wiki/Day\\_and\\_Night\\_\(cellular\\_automaton\)](https://en.wikipedia.org/wiki/Day_and_Night_(cellular_automaton)).

- [14] t. f. e. Wikipedia, «Von Neumann cellular automaton,» [En línea]. Available:  
[https://en.wikipedia.org/wiki/Von\\_Neumann\\_cellular\\_automaton](https://en.wikipedia.org/wiki/Von_Neumann_cellular_automaton).
- [15] l. e. l. Wikipedia, «Juego de cero jugadores,» [En línea]. Available:  
[https://es.wikipedia.org/wiki/Juego\\_de\\_cero\\_jugadores](https://es.wikipedia.org/wiki/Juego_de_cero_jugadores).