ALGORITMOS GENÉTICOS Y EVOLUTIVOS

DESAFÍO FINAL

Grado en Ingeniería Informática

Campus de Colmenarejo

Curso 2020/2021

Autores

Eduardo Ureña Toledano - 100329937@alumnos.uc3m.es

Andrés Sánchez Tamargo - 100346113@alumnos.uc3m.es

Francisco José Montero Galloso - 100346228@alumnos.uc3m.es

Martín Casamayor Morales - 100346181@alumnos.uc3m.es

Santiago Vidal Carretero - 100346107@alumnos.uc3m.es

Alberto Viejo López - 100346079@alumnos.uc3m.es

Índice

1. Planificación inicial y final

La planificación inicial consistía en:

1. Entender cómo funciona el programa y con qué parámetros podemos jugar.
2. Programar un par de coches por integrante del grupo mediante reglas.
3. Jugar 100 partidos contra “AgentAGE2” con cada par de coches.
4. Comparar el porcentaje de partidos ganados por cada agente (par de coches) con respecto al individuo proporcionado por el profesorado.
5. Elegir el agente que mejores resultados ha dado, e intentar mejorarlo haciendo uso de alguna de las técnicas de Computación Evolutiva vistas en clase.

De este modo y una vez comprendido el funcionamiento del programa y los atributos más importantes a la hora de modificarlo para generar los nuevos competidores no pasó mucho tiempo hasta que logramos obtener algunos agentes capaces de derrotar al coche proporcionado (“AgentAGE2”) por el profesorado las suficientes veces como para considerarlo mejor, pero con muy poco margen de mejora debido a unas reglas relativamente pobres. El índice de victoria lo obtuvimos tras realizar 100 intentos (partidos) contra dicho agente con cada par de coches creados por el equipo, obteniendo con los dos mejores un 71% y un 67% de victorias. El número de partidos seleccionado fue asignado tras valorar que resultaba ser una cifra significativa y que además permitía calcular el porcentaje de victoria de forma sencilla obteniendo así un grado de eficiencia de los vehículos.

Derrotado al agente proporcionado, lo siguiente a realizar fue poner a nuestros agentes a competir entre sí para elegir un ganador, sin embargo, debido a los ajustes realizados no había una gran diferencia entre los mismos lo cual nos llevó a la conclusión de que debíamos programar un mejor individuo que pudiera superar también a los que habíamos creado antes de mejorarlo mediante las técnicas de Computación Evolutiva.

Es por esto que cambiamos la estrategia pasando así a realizar una sesión de trabajo conjunta con el objetivo de generar un agente más sofisticado haciendo uso de la herramienta Python. Para crear una inteligencia artificial lo más apta posible de cara a obtener un mayor número de victorias, una de las estrategias que seguimos fue estudiar los vehículos con mayor ratio de partidos ganados en la página Bit Runner 2048 implementando las mejores estrategias en nuestro agente de cara a obtener mejores resultados. Finalmente, una vez desarrollado el “agente definitivo” que hacía uso de las normas establecidas a mano, lo empezamos a probar en los torneos que proporciona dicha web, viendo así la mejora del individuo y el límite que podía alcanzar.

2. Roles designados a los componentes del equipo

Roles	Nombre y apellidos
Jefe del equipo de pruebas	Francisco José Montero Galloso
Jefe del equipo técnico	Andrés Sánchez Tamargo
Programador	Eduardo Ureña Toledano
Tester	Martín Casamayor Morales
Programador	Santiago Vidal Carretero
Tester	Alberto Viejo López

3. Relación de tareas realizadas por cada miembro del equipo

Tareas realizadas	Nombre y apellidos
Programación del primer agente. Dirección del equipo de pruebas. Realización de las pruebas sobre el agente. Programación de la estrategia Programación del agente definitivo Asistencia en la redacción de los puntos 4, 5, 6 y 7 de la memoria.	Francisco José Montero Galloso
Programación del primer agente. Dirección del equipo de programación. Programación de la estrategia. Programación del agente definitivo. Asistencia en las pruebas sobre el agente. Asistencia en la redacción de los puntos 4, 5, 6 y 7 de la memoria.	Andrés Sánchez Tamargo

Programación del primer agente. Asistencia en las pruebas sobre el agente. Establecimiento del formato de la memoria. Programación del main. Asistencia en la redacción de los puntos 1, 4, 5, 6 y 7 de la memoria.	Eduardo Ureña Toledano
Programación del primer agente. Realización de las pruebas sobre el agente. Redacción de la memoria.	Martín Casamayor Morales
Programación del primer agente. Asistencia de pruebas sobre el agente. Programación del main. Redacción de la memoria.	Santiago Vidal Carretero
Programación del primer agente. Realización de las pruebas sobre el agente. Asistencia en la redacción de los puntos 4, 5, 6 y 7 de la memoria.	Alberto Viejo López

4. Descripción de la técnica implementada

4.1 Parámetros iniciales 1 + 1

Los valores iniciales de los parámetros, tanto de la parte funcional como de las varianzas, se inicializan aleatoriamente, pero siempre asegurándose de que no se obtienen valores que hagan que el individuo sea no-funcional, reparándolos en ese caso.

4.2 Funciones del 1 + 1

Individual(min,max)

Esta función tiene como objetivo el crear un individuo con valores aleatorios para sus genes definidos entre “min” y “max”, y con una longitud de “largo”.

En este caso, los valores “min” y “max” corresponden a valores aleatorios entre 0 y 10000 respectivamente en el caso de la población y un número aleatorio entre dos valores propuestos arbitrariamente para las varianzas.

Crearpoblacion()

Esta función dará lugar a una función del tamaño otorgado por la variable “num”, que en nuestro caso es de 23, siendo 19 velocidad y 4 distancias.

CrearVarianzas()

Esta función se encarga de crear un vector de varianzas para cada individuo de la población, cogiendo como valores, números aleatorios entre dos valores arbitrarios, en este caso entre 1 y 200.

CalcularFitness(individual)

Esta función se encarga de calcular el fitness del individuo y de devolverlo como parámetro. También guardará los valores reparados del individuo en un fichero txt.

Comenzamos reparando los posibles errores del individuo. Debido a las varianzas, podemos encontrarnos con decimales, tanto en las velocidades como en las distancias. Si esto sucede, redondeamos el valor. Tampoco debería haber valores negativos ni en las distancias ni en las velocidades, por lo que si encontramos uno lo pasaremos a valor absoluto.

En caso de encontrar una velocidad o una distancia de 0 la repararemos pasando su valor a 1, ya que ninguna de las dos puede tomar un valor de 0.

No podemos exceder la velocidad máxima de 200 o la distancia máxima de 10.000, si se pasan estos valores se realizará un random para volver a calcular ambas. Para la velocidad un randint(1,200) y para la distancia un randint(1,10000).

Una vez hecho esto se guardan los datos del individuo en un fichero txt que se genera automáticamente y se procede a calcular el fitness.

Para calcular el fitness recuperamos los resultados de los partidos simulados. Estos resultados están almacenados en 2 ficheros txt, cada fichero contiene fila a fila los goles marcados en cada partido. Al recuperar los resultados los almacenamos en dos listas con las cuales calcularemos la diferencia de goles, los goles que hemos marcado menos los goles que nos han marcado. Cabe destacar que nuestro equipo siempre será el equipo 2 para obtener el fitness correctamente. Para calcular el fitness, calculamos la diferencia de goles entre los dos equipos, si la diferencia es negativa significa que hemos perdido, si es positiva, hemos ganado. En caso de haber ganado sumamos 10 puntos al resultado. Pasaremos los resultados negativos a positivos, los resultados de menos de 11 puntos serán resultados en contra y con más o igual de 11 serán partidos a favor y ganados. Una vez tenemos esos dos datos, dividimos los puntos en contra de los puntos a favor y obtendremos el fitness. Buscamos obtener un fitness de 0.

Mutation(population,varianzas)

Esta función se encarga tanto de mutar al individuo como al vector de varianzas.

Para mutar a la población, se obtiene un hijo (individuo mutado) sumando al valor del padre, un valor aleatorio de la normal entre 0 y su varianza correspondiente. Este nuevo individuo, es guardado en el vector poblacionmutacion[].

Una vez creado el nuevo vector, se compara el fitness del hijo creado con el de su padre, si el nuevo individuo presenta un mejor fitness que el anterior existente, ese nuevo individuo reemplazará a su padre y, a partir de él se creará el siguiente. En caso contrario, el nuevo individuo creado, es desechado.

Para mutar las varianzas, se sigue la regla del 1/5.

$$\sigma^{t+n} = \begin{cases} c \cdot \sigma^t & si\ \psi_s^t < \frac{1}{5} \\ \frac{\sigma^t}{c} & si\ \psi_s^t > \frac{1}{5} \\ \sigma^t & si\ \psi_s^t = \frac{1}{5} \end{cases}$$

Para mutar las varianzas, se deben de seguir unos pasos:

En primer lugar, se tiene un vector s, en el que se irá registrando el número de veces que nuestro individuo mejora entre las últimas “s.length” generaciones.

Dependiendo del valor obtenido, se aplicará la regla del 1/5.

Un ejemplo sería, si nuestro vector s tiene un tamaño de 10 y ha mejorado en las últimas 4 generaciones, el resultado obtenido será de 0.4.

Una vez obtenido este valor, se pasará a aplicar la regla del 1/5, en la que, dependiendo del valor obtenido, se multiplicarán las varianzas o se dividirán por un número "c" de manera que se penalice o bonifique la cercanía a la solución óptima. Cuanto menores son las varianzas, más cerca se está de la solución óptima.

Calcularnuevoreseteo(mejorfitness)

Esta función se encarga de resetear las varianzas, en el caso de que las varianzas se vuelvan extremadamente pequeñas, será necesario resetearlas ya que, en caso contrario, el individuo dejará de cambiar, por lo tanto, dejará de mejorar y, por lo tanto, se quedará estancado.

Actualizarvaloracion(valor)

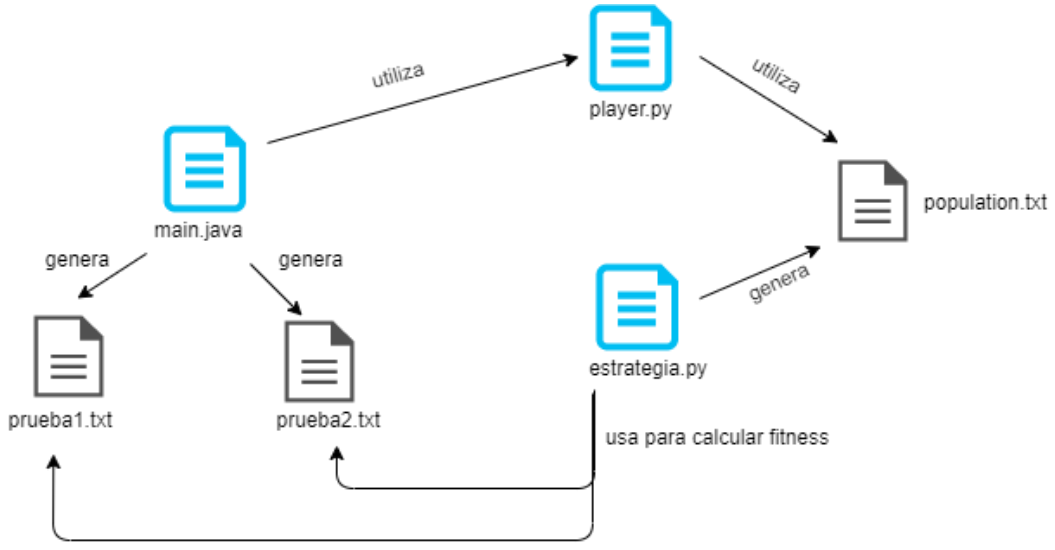
Esta función se encarga de ir actualizando el vector s, añadiendo un 0 si el nuevo individuo no ha mejorado con respecto al anterior o un 1 en caso de haber mejorado al anterior. Este vector *s* se irá actualizando cíclicamente de tal forma que solo se tengan en cuenta las últimas s generaciones.

Sumalista(listaNumeros)

Esta es una función simple que se encarga de dar como resultado la suma de los valores de una lista dada por parámetro.

4.3 Ejecución de la estrategia + Netbeans

Para entender mejor la ejecución del programa, hemos creado el siguiente diagrama, el cual explicaremos a continuación.



Lo primero que se debe hacer antes de ejecutar el programa será comprobar las rutas en las que tenemos guardado los archivos. Guardaremos la carpeta Desafio_final_g1 en C: para evitar posibles errores por las rutas. El main creará dos ficheros .txt llamados prueba1 y prueba2 de los cuales habrá que introducir la ruta de creación en el propio main. En cuanto al fichero estrategia.py se deberán incluir las rutas de prueba1 y prueba2 y añadir una nueva ruta donde se creará el fichero population.txt. Por último el fichero player.py deberá tener la ruta donde se haya creado el fichero population.txt.

La primera vez que queramos ejecutar el programa, deberemos empezar por estrategia.py, ejecutaremos una vez y una vez termine podremos ir al main en NetBeans para ejecutarlo. Una vez termine de ejecutar NetBeans podremos volver a dar a la estrategia. Este proceso se repetirá hasta que terminen todas las ejecuciones.

No hemos conseguido desarrollar un script para realizar este trabajo automáticamente, pero si hemos hecho uso de la aplicación PlayStatic 2.0 para crear una macro que simule el movimiento del ratón y del teclado. Hemos optado por esto ya que era la solución más rápida si buscábamos ponernos a hacer pruebas lo antes posible y podíamos dejarlo toda la noche. El único inconveniente de esto es que inhabilita el uso del ordenador, ya que el programa necesita que el ratón se mantenga quieto.

4.4 Funcionamiento del agente

Tal y como se había mencionado previamente en el primer punto de esta memoria, con el sistema a desarrollar elegido, pasamos a la recopilación de la información más importante para nuestro agente: las estrategias que debía emplear y las situaciones en las que debía hacerlo. De este modo y tras realizar varias pruebas desarrollando agentes competentes, llegamos a la conclusión de que lo mejor sería ver en acción al mejor agente que pudiéramos encontrar y practicar algo parecido a la ingeniería inversa con el mismo, es decir, gracias a la web anteriormente mencionada (Bit Runner 2048), realizamos un análisis de las prácticas llevadas a cabo por los jugadores con mayor índice de victoria y el cómo cada una de ellas influía sobre el resultado. Así, con la información recopilada, nuestro agente pasó a incorporar las siguientes estrategias clave para su funcionamiento (cada una comprendida por un comportamiento definido para cada coche):

- **Acechar** (modo acechador):
    - En el caso de que el enemigo de Mario tome la pelota, se dirigirá al centro y se posicionará a la espera del mismo.
    - En caso de que sea el enemigo de Luigi quien tome la pelota, intentará alcanzar al mismo.

- **Robar** (modo ladrón) se llevará a cabo una vez finalizado el modo acechador:
    - Si el enemigo de Mario se acerca a la portería o se "miran" de frente ambos vehículos (Mario y su enemigo), intentará golpearlo para robarle el balón.
    - Al igual que Mario, Luigi se lanzará a golpear a su rival para robarle la bola en caso de que éste se acerque a la portería, se miren de frente, o si Luigi pasa por el centro.
- **Robar con aceleración** (modo ladrón+):
    - No aplica para Mario.
    - En caso de que el enemigo tenga la bola y se cumpla una condición de distancia entre Luigi y el rival, se cambiará la velocidad de Luigi para que intente robar el balón al contrario de un modo mucho más efectivo.
- **Respaldo** (modo ladrón 2):
    - En caso de que uno de los hermanos tenga una de las bolas y la otra se encuentre en posesión de su adversario asignado, como nuestro compañero no puede defenderlo, se le hace la ayuda. A efectos prácticos es como si temporalmente se intercambiaran los rivales de ambos hermanos.
- **Campear** (modo campero):
    - Cuando nadie tiene el balón, Mario se mantendrá en el centro del tablero esperando a que se cumpla alguna de las condiciones que disparan los otros modos.
    - No aplica para Luigi.
- **Buscar bola** (modo pelota):
    - No aplica para Mario.
    - Cuando nadie tiene el balón, Luigi intentará atrapar la última bola que haya sido creada.
- **Recoger bola** (modo pelota+):
    - No aplica para Mario.
    - En el momento en el que Luigi, estando en modo pelota, se encuentra a una determinada distancia de la bola, cambia su valor de aceleración para cogerla con un índice de precisión mayor.
- **Apuntar** (modo enfilar):
    - Cuando uno de los hermanos atrapa la bola, gira hasta mirar directamente a portería.
- **Disparar** (modo gol):
    - Cuando uno de los hermanos tiene la bola y está mirando directamente a portería, cambia su aceleración y se lanza a marcar gol.

- **Robo con ayuda** (modo ayuda):
    - En caso de que el enemigo de su hermano tenga la bola y el suyo no, se ayudará a robarle el balón al enemigo generando una posición de 2v1 más ventajosa.
- **Despejar el camino** (modo guardaespaldas):
    - En caso de que su hermano tenga la bola, se lanza hacia el centro para quitar a los coches enemigos camperos o con intención de robar la bola que se puedan encontrar en el camino, ayudando así a asegurar el gol.

5. Justificación, ventajas e inconvenientes

Para mejorar al individuo generado, hemos decidido hacer uso de una estrategia 1+1. La razón de escoger dicho método es la obtención de mejores resultados con respecto al resto de algoritmos. Además, en vista de los resultados obtenidos en la segunda práctica con respecto a la obtención de una solución óptima de cara a optimizar el tiempo, y aún por encima eso, con respecto a la facilidad de su implementación con respecto a los demás algoritmos. Es por esto que, si bien somos conscientes de que un sistema evolutivo $\mu+\lambda$ podría proporcionar mejores resultados en menos tiempo, la dificultad de implementar dicho sistema de manera funcional con respecto al 1+1, que además había supuesto la mayoría de implementaciones en la práctica anterior por parte de los miembros del equipo, ha supuesto una diferencia remarcable en favor del 1+1.

6. Pruebas (gráficos, tablas, etc..)

COCHE POR DEFECTO

Pruebas realizadas entrenando contra el coche por defecto:

Nº GENERACIONES	%VICTORIA	FITNESS
63	72	0.051
71	78	0.039
57	82	0.023
84	75	0.054
MEDIA GENERACIONES: 69	**% MEDIO DE VICTORIAS:** 77	**MEDIA FITNESS:** 0.041

```
Una vez ejecutado el main pulse enter para continuar.
Victorias: 82; Derrotas: 18; porcentaje partidos ganados: 82.0%
Fitness obtenido: 0.023936170212765957
```

COCHE MEJOR INICIAL

Pruebas realizadas entrenando contra el mejor coche inicial de entre los que se hicieron por los integrantes del equipo:

Nº GENERACIONES	%VICTORIA	FITNESS
53	76	0.042
26	91	0.015
43	81	0.026
23	78	0.039
MEDIA GENERACIONES: 36	**% MEDIO DE VICTORIAS:** 82	**MEDIA FITNESS:** 0.03

```
Estoy en la generacion: 26
Esta es mi población antes:[[128, 115, 51, 162, 50, 169, 130
, 128, 194, 194, 39, 169, 84, 60, 189, 31, 95, 139, 55, 6330
, 8391, 3698, 7091]]
El mejor fitness anterior era:0.014827018121911038
```

COCHE DEFINITIVO

Pruebas realizadas entrenando contra el coche definitivo con los valores puestos a mano:

Nº GENERACIONES	%VICTORIA	FITNESS
100	52	0.175
100	51	0.187
100	59	0.137
100	57	0.127
MEDIA GENERACIONES: 100	**% MEDIO DE VICTORIAS:** 55	**MEDIA FITNESS:** 0.157

```
Una vez ejecutado el main pulse enter para continuar.
Victorias: 57; Derrotas: 43; porcentaje partidos ganados: 57.0%
Fitness obtenido: 0.1273972602739726
```

Una vez realizadas las pruebas con cada uno de los coches adversarios, se evaluó el mejor coche resultante de cada una de ellas con el fin de quedarnos con el mejor de los tres.

Para ello, se probó como se comportaba cada uno de esos coches ganadores contra el resto de adversarios.

Una vez realizadas las pruebas, el coche vencedor resultó ser el siguiente:

```
Estoy en la generacion: 26
Esta es mi población antes:[[128, 115, 51, 162, 50, 169, 130
, 128, 194, 194, 39, 169, 84, 60, 189, 31, 95, 139, 55, 6330
, 8391, 3698, 7091]]
El mejor fitness anterior era:0.014827018121911038
```

7. Conclusiones

Debido a las circunstancias, no se ha podido realizar el trabajo como nos gustaría. Esto se debe a que, acostumbrados a lo largo de todos estos años al trabajo presencial en equipo y las reuniones en grupos de trabajo que permitían realizar una gran interacción interpersonal facilitando la captación de ideas y generación de nuevas alternativas conjuntas, el cambio a los sistemas de comunicación electrónica que anula dicha posibilidad nos ha supuesto una gran pérdida de información en las reuniones realizadas al perderse algunas herramientas de trabajo como pizarras, toda la información proporcionada por la comunicación no verbal e incluso tener así una mayor dificultad a la hora de concentrarse en la tarea establecida para cada reunión.

Sin embargo, no todo esto es malo, ya que el trabajo a distancia, por otra parte, ha facilitado la realización de dichas reuniones ya que la disponibilidad de los miembros del grupo se limitaba a la disponibilidad general de los mismos, es decir, no se requería de invertir tiempo o gastos en viajes que permitieran realizar una reunión presencial. Del mismo modo, las herramientas empleadas para desarrollar este tipo de práctica, al tratarse de recursos informáticos, se podían emplear y compartir mediante los sistemas electrónicos disponibles. De esta forma las reuniones presenciales se han limitado a una cantidad reducida de encuentros en el recinto universitario aprovechando la necesidad de asistir a clases o tutorías y se ha realizado la mayor parte del trabajo de forma remota.

De cara a realizar la práctica de la forma más óptima posible y no “atropellar” el trabajo de los compañeros, se ha practicado la división del equipo en dos partes diferenciadas resultando así en una mayor facilidad a la hora de realizar las comunicaciones y distribuir

las tareas y horas de trabajo de los integrantes. Así, teniendo un equipo principal de programación y otro de pruebas asistido por el primero, las reuniones que permitían compartir ideas entre los grupos podía reducirse a tan solo los líderes de cada equipo, los cuales informaban del progreso realizado y necesidades requeridas del otro grupo.

Resulta de interés que, aunque no se pudiera realizar la misma carga de trabajo en cada apartado por los dos equipos (ahí radica la fuerza de esta división en grupos), sí que se ha conformado la asistencia de todos los miembros del equipo en las partes críticas de trabajo, es decir, si bien un miembro del equipo de pruebas no tenía el requerimiento de programar el algoritmo, sí que asistía a la programación del mismo y aprendía su funcionamiento de modo que las futuras pruebas pudieran resultar más llevaderas y la comprensión del código fuera lo mayor posible. De la misma manera todas las partes han contribuido a la realización de la memoria aunque en los apartados relativos al trabajo realizado por cada uno.

En general el trabajo en equipo nos ha resultado sencillo, esto se debe a que, si bien no resulta fácil coordinar un grupo de tantas personas, cada una con sus respectivas responsabilidades y tareas ajenas al proyecto, la experiencia de trabajo de años anteriores con los otros miembros del equipo, la disponibilidad de las herramientas informáticas mencionadas anteriormente y la buena división del trabajo por equipos nos ha permitido realizar, con el menor número de problemas interpersonales la práctica requerida.

En general podemos concluir que la práctica nos ha dejado un buen sabor de boca, esto de debe a que, si bien ha habido dificultades a lo largo de su realización como los problemas causados por el largo tiempo de ejecución del código (cuatro minutos y medio por cada cien partidos suponiendo una inmensa cantidad de tiempo por generación y mejora) y la simultaneidad de la realización de esta práctica en conjunto con otras, la temática elegida ha sido, en nuestra opinión, muy acertada, resulta un enfoque realmente estimulante de cara a un alumnado que ha crecido con juegos similares o las respectivas evoluciones de los mismos (como por ejemplo el actual Rocket League). Del mismo modo, dado que ya habíamos realizado previamente el estudio de las alternativas para hacer evolucionar nuestro agente, esta práctica ha resultado más, como su nombre bien indica, un desafío divertido en el que podíamos poner en práctica lo aprendido y además disfrutar haciéndolo y a la vez, desarrollando una posible solución a un problema real que es la generación de una buena IA para un juego.

Cabe destacar sin lugar a dudas que el torneo final con la tensión añadida de la forma de visualizar los resultados ha sido, con diferencia, más divertido que los anteriores (hemos podido disfrutar con gritos de emoción por el partido en una llamada aparte), estimulando más la competitividad entre los alumnos pero disfrutando igualmente de una justa derrota (en nuestro caso concreto al quedar segundos).

Nos gustaría mencionar también que, tal y como se ha podido comprobar en prácticas anteriores de los miembros del equipo, aunque hay personas que no comparten esta opinión y que podrían considerar incluso como una ofensa o falta de respeto este tipo de prácticas, que todas las referencias tanto en nombres, como imágenes de la presentación e impresiones realizadas se han pensado con el único propósito de aumentar el interés y realizar un código mucho más ameno y amigable tanto para los que lo programan (en este caso nosotros) como para los que lo analizan (equipo de pruebas, profesorado o gente interesada), haciendo así del desarrollo de la práctica también algo un poco más divertido.

Para finalizar, diremos que la asignatura ha resultado, por momentos, curiosa, interesante, divertida, estresante (cuando no teníamos tiempo para realizar alguna práctica o nos estancábamos), necesaria y, de forma global, estimulante. Es por esto que agradecemos tanto al profesorado como a los excelentes compañeros con los que hemos podido realizar esta asignatura que nuestra experiencia haya podido resultar algo tan positivo y agradable.