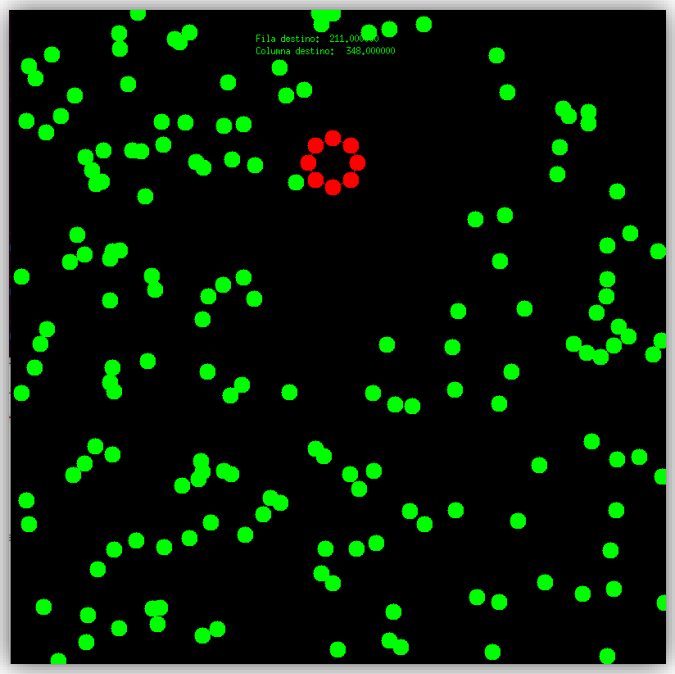
Metodología de la programación

PROYECTO FINAL PRÁCTICAS (SEGUNDA ENTREGA)



Ejemplo de prueba Wally.cpp (200 puntos)

Iván Cortón da Silva 15426623V - 1ºD

Índice:

Descripción de la práctica

Durante el desarrollo de la segunda entrega de prácticas se nos encomendó, en primer lugar, la comprobación de la correcta implementación de las clases **Punto2D** y **PoliReg** mediante el uso de un fichero aportado en PRADO de nombre *pruebas.cpp.* Realizar dicha comprobación no me resultó una tarea muy engorrosa, ya que la sobrecarga de operadores me resultó relativamente sencilla.

Por otro lado, tras pasar los test correspondientes del fichero *pruebas.cpp*, se nos pidió desarrollar una función que ordenara 10 polígonos generados aleatoriamente mediante la sobrecarga del operador **mayor que**, que, a mi parecer, tampoco fue una tarea demasiado complicada.

En cuanto a la segunda parte de esta segunda entrega de la práctica, se nos pidió el diseño de un "robot Wally" que simulara el comportamiento de un robot recogedor de basura que se encontrara en la pantalla generada por el módulo *miniwin* ya visto en la primera parte de esta entrega. Por lo general, la implementación del robot no fue demasiado complicada, ya que el código aportado en PRADO fue lo suficientemente adecuado como para no encontrar dificultades, al menos, en el ámbito del cálculo y las matemáticas necesarias para programar al robot correctamente, algo que me pareció idóneo.

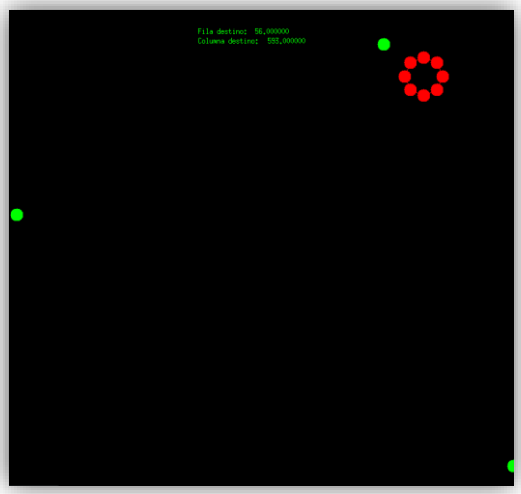
En cuanto a las funciones auxiliares utilizadas para el diseño de Wally, podría destacar; la función *enPantalla*, encargada de comprobar cuándo un punto se encuentra dentro del límite hasta el que puede llegar a recoger basura nuestro robot sin salir del mapa, la función *estaDentro*, necesaria en mi caso, para que una vez lleguemos al último punto de basura que se puede recoger según el método anterior, finalice el algoritmo. Además de estas dos funciones necesité de una tercera función que se la asigné a la clase **Punto2D**; la función *eliminarPunto*, necesaria para poder eliminar un punto del espacio cuando hayamos llegado hasta él. La función *enRango* solicitada por el profesor también fue implementada para que cada vez que pasáramos por un punto de basura sin que fuera el destino hacia el que nos estábamos moviendo, nuestro robot fuera recogiendo dicha basura.

Por último, en lo relativo a la implementación del robot, cabe destacar la implementación de la búsqueda del próximo destino al que viajar, mediante la **búsqueda del primer mejor vecino** encontrado en el entorno del robot, lo que claramente provoca que no podamos comprobar del todo el correcto funcionamiento del método *enRango*, sin embargo, modificando el valor de la variable ***min*** por 0 (líneas 50 y 58) y el intercambio de símbolos (línea 63) del fichero ***Wally.cpp***, podemos cambiar el tipo de búsqueda de Wally haciendo que éste viaje hacia el vecino más lejano, momento en el que podríamos comprobar el correcto funcionamiento de *enRango.*

En cuanto al grado de dificultad de ambas partes de la entrega final, me ha parecido bastante adecuado para poder comprender la implementación de una correcta metodología de programación, ya que en especial, la segunda parte de la entrega de la práctica final, me ha parecido bastante interesante y divertida, tanto que pensé en incluir algoritmo capaces de generar planes en un inicio, como el algoritmo A*, simulando el comportamiento de un agente reactivo con conocimiento completo del problema, aunque en vista de la complejidad del algoritmo y de todos los criterios a tener en cuenta, descarté esta opción, tratando de simular un algoritmo más próximo a la Búsqueda Local (*Local Search*).

En condiciones normales, hubiera tratado de profundizar más, aunque con lo ajustada que está la planificación de las entregas de otras asignaturas, que actualmente estoy cursando de tercer curso, no pude mejorar mucho más el robot.

Para finalizar, creo que la labor del profesorado tanto en la parte de teoría como en la parte de prácticas ha sido inverosímil, en cuanto a flexibilidad, ajustes de planificación de clases y horarios, atención al alumnado… Y sobre todo, me gustaría destacar la gran implicación del profesor de prácticas por haber podido dedicarnos tiempo a todos los alumnos que solicitaban ayuda o necesitaban de ella a través de Skype a modo de tutoría. En un escenario tan malo como en el que nos encontramos, creo que hemos recibido un muy buen trato por parte de los profesores de la asignatura, lo cual agradezco encarecidamente.



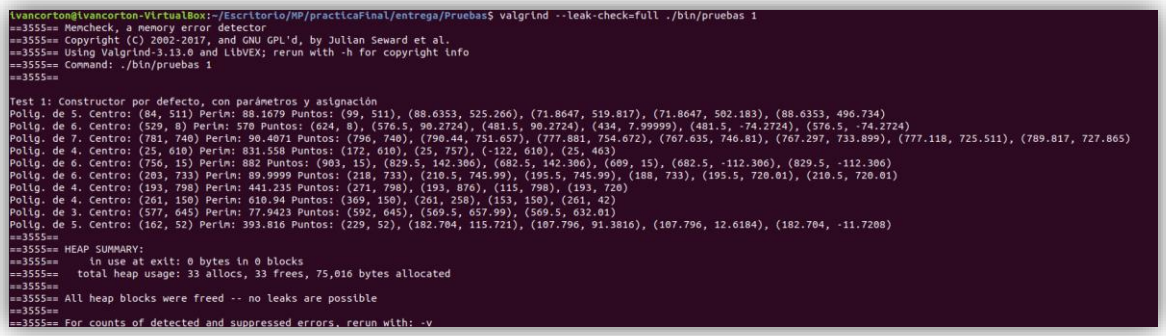
Demo de la función *enRango*

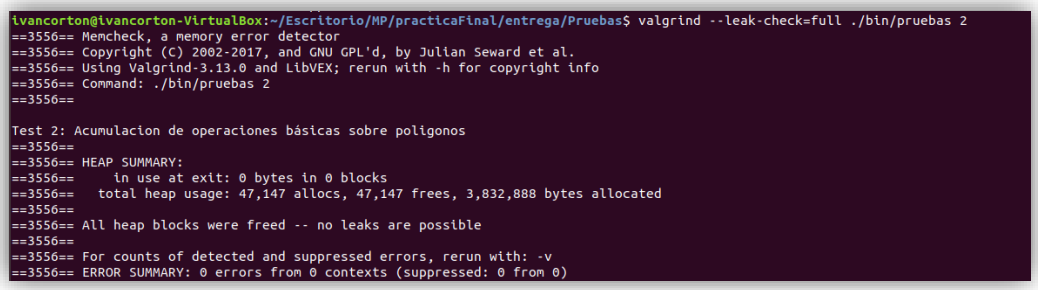
Autoevaluación: En mi opinión, creo que merezco un notable

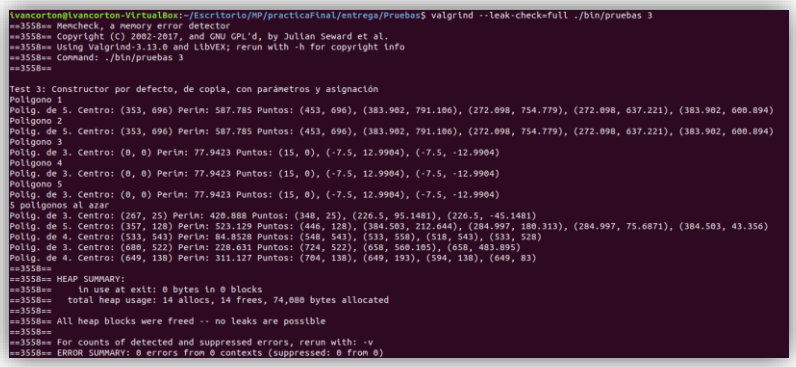
Análisis de Valgrind

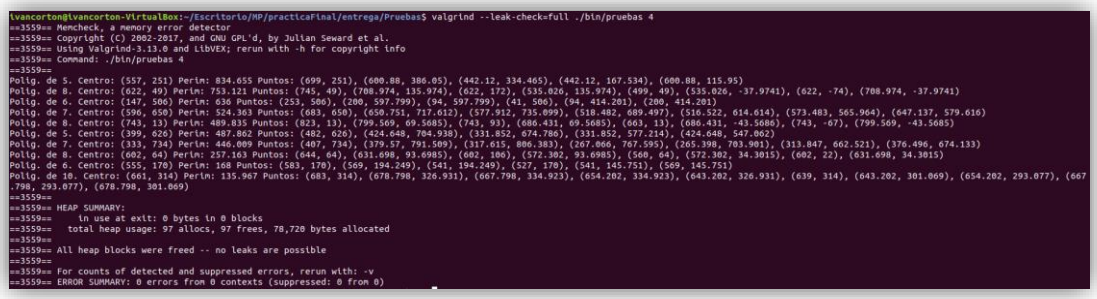
- Órdenes utilizadas para obtener las salidas:

  - $valgrind --leak-check=full ./bin/pruebas < 1 | 2 | 3 | 4 >$









Valgrind sobre pruebas.cpp

- $valgrind --leak-check=full ./bin/Wally$

```
ivancorton@ivancorton-VirtualBox:~/Escritorio/MP/practicaFinal/entrega/Wally$ valgrind --leak-check=full ./bin/Wally
==3652== Memcheck, a memory error detector
==3652== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3652== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==3652== Command: ./bin/Wally
==3652==
==3652==
==3652== HEAP SUMMARY:
==3652==     in use at exit: 448 bytes in 2 blocks
==3652==   total heap usage: 1,991 allocs, 1,989 frees, 202,932 bytes allocated
==3652==
==3652== 288 bytes in 1 blocks are possibly lost in loss record 2 of 2
==3652==    at 0x4C31B25: calloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3652==    by 0x40134A6: allocate_dtv (dl-tls.c:286)
==3652==    by 0x40134A6: _dl_allocate_tls (dl-tls.c:530)
==3652==    by 0x5ABB227: allocate_stack (allocatestack.c:627)
==3652==    by 0x5ABB227: pthread_create@@GLIBC_2.2.5 (pthread_create.c:644)
==3652==    by 0x10BE11: _maybe_call_main() (miniwin.cpp:673)
==3652==    by 0x10BE83: _process_event() (miniwin.cpp:689)
==3652==    by 0x10BF75: main (miniwin.cpp:733)
==3652==
==3652== LEAK SUMMARY:
==3652==    definitely lost: 0 bytes in 0 blocks
==3652==    indirectly lost: 0 bytes in 0 blocks
==3652==      possibly lost: 288 bytes in 1 blocks
==3652==    still reachable: 160 bytes in 1 blocks
==3652==         suppressed: 0 bytes in 0 blocks
==3652== Reachable blocks (those to which a pointer was found) are not shown.
==3652== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==3652==
==3652== For counts of detected and suppressed errors, rerun with: -v
==3652== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 273 from 1)
```

Valgrind sobre Wally.cpp