Algoritmos y Programación II 75.04

Trabajo práctico 1: war!

Facultad de Ingeniería - UBA
Segundo cuatrimestre de 2013

1. Objetivos

Ejercitar conceptos relacionados con estructuras de datos y programación C++ orientada a objetos de mayor complejidad. Escribir un programa en este lenguaje (y su correspondiente documentación) que resuelva el problema que presentaremos más abajo.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá será entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe impreso de acuerdo con lo que mencionaremos en la sección 6, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Introducción

En esta segunda instancia, el objetivo esencial es extender los programas del trabajo práctico anterior de manera tal de implementar la lógica del juego ya presentado. Para simplificar la tarea, cada jugador (i.e., cada programa) resolverá su turno de forma *offline* y comunicará las decisiones tomadas a un juez centralizado a través de archivos de texto. El juez contabilizará los puntajes parciales de cada jugador y a su vez informará a éstos las sucesivas disposiciones del territorio conforme progresen los turnos.

A los efectos de poner en práctica nociones básicas de programación orientada a objetos, implementaremos también un patrón de diseño que nos facilite el desarrollo de múltiples algoritmos de ataque y defensa.

4.1. El juego: revisión

Como hemos dicho en el enunciado del primer trabajo práctico, este juego consta de una disputa entre dos ejércitos en un territorio. Entendemos por *ejército* un conjunto de soldados, mientras que el *territorio* no es más que una grilla de $n \times m$ casilleros en donde cada uno de éstos, en cualquier instante de la batalla, puede estar libre o bien ocupado por uno o más soldados de cualquiera de los ejércitos.

Cada soldado tiene un *rango de influencia* en el que podrá moverse o atacar. El rango de influencia (x, y) de un soldado es un rectángulo cuyo vértice superior izquierdo está x casilleros a izquierda de la posición del soldado e y casilleros arriba, y cuyo vértice inferior derecho está, de manera análoga, x casilleros a derecha de su posición e y casilleros por debajo de ella. Naturalmente, los límites del territorio imponen una restricción al rango de influencia.

Al comienzo, todos los soldados inician con un nivel de energía e máximo. Con el transcurso de la batalla, este nivel de energía puede disminuir pero nunca aumentar. Un valor de cero indicará que el soldado está muerto y por ende no podrá ser utilizado posteriormente.

En cada turno, los jugadores deben decidir qué acción aplicar a cada uno de sus soldados disponibles. Hay sólo dos acciones posibles, y a lo sumo una de ellas puede aplicarse a cada soldado: *ataque* y *desplazamiento*. El ataque consiste en arrojar una granada a un casillero a elección que esté por supuesto dentro del rango de influencia del soldado. Por otro lado, el desplazamiento permite que un soldado pueda reubicarse en una nueva posición del territorio, también dentro de su rango de influencia.

Las granadas tienen también su rango de influencia, y serán más agresivas cuanto más próximas estén al objetivo. Una misma granada puede herir a más de un soldado –incluso a soldados del propio ejército.

Dada una granada g arrojada a la posición p y un soldado s con energía ϵ ubicado en la posición q, la siguiente función δ describe el nivel de energía final de s al ser alcanzado por g. En otras palabras, δ mide el daño que g causa a s, y es el valor que computará el juez por cada granada y soldado cada vez que finalice un turno:

$$\delta(g, s) = \begin{cases} \epsilon & \text{si } q \text{ no está en el rango de influencia de } g \\ \left\lfloor \epsilon - \text{mín}(\epsilon, \frac{e}{2^{d(p,q)}}) \right\rfloor & \text{en otro caso} \end{cases}$$

siendo $d(p, q)$ la distancia euclideana entre los puntos p y q:

$$d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Una vez determinadas las acciones de ambos bandos, éstas serán ejecutadas en simultáneo por el juez. Los jugadores proseguirán a definir nuevas acciones a partir de la nueva disposición del territorio resultante de la ejecución de las acciones del turno anterior.

El juego dispone de dos mecanismos de finalización: o bien cuando un ejército exterminó a cada miembro de su enemigo o bien al cabo de un número máximo t de turnos (indicado por el juez en los archivos de entrada, tal como mostraremos en las secciones posteriores). Al finalizar, se contabilizarán los puntos obtenidos por cada jugador de acuerdo a cuánto daño se inflingió a los soldados contrarios. Por cada soldado rival r que participó en la contienda, el juez sumará $e - \epsilon_r$ puntos, siendo ϵ_r el nivel de energía final de r y e el nivel máximo de energía determinado al inicio del juego. El jugador con mayor puntaje se considerará ganador de la batalla.

5. El programa

Siguiendo la línea del primer trabajo, la interacción con el programa se dará a través de la línea de comando. Dependiendo de cómo esté estructurado el contenido del stream de entrada, el programa deberá tomar distintos cursos de acción.

Además, queremos experimentar la efectividad de distintas tácticas. Es por ello que el usuario del programa podrá indicar qué tipo de estrategia prefiere utilizar: agresiva, evasiva o default. En el primer caso, todos los soldados deben realizar acciones de ataque, arrojando granadas a una posición válida cualquiera. Cuando se trate de evasión, todos los soldados deberán desplazarse. Por último, la estrategia por defecto permite que los programas tomen sus propias decisiones según lo que consideren más apropiado. Este concepto será materializado a través del patrón de diseño presentado en la sección 5.2.

Con el objeto de poder identificar unívocamente cada ejército durante la competencia, se requiere también que cada grupo elija un nombre arbitrario que pueda ser informado al juez cuando éste lo solicite a través de una nueva opción de línea de comando (ver sección 5.4).

5.1. Sintaxis de entrada y salida

5.1.1. Entrada

El stream de entrada seguirá el siguiente formato, similar pero ligeramente distinto al ya descripto:

- En la primera línea aparecerá el número del turno actual, siendo `0` el primer valor válido. En tal caso, el programa debe seguir la lógica de lo hecho en el trabajo anterior y colocar a los soldados en las posiciones elegidas.

- La segunda línea sólo contendrá el número máximo de turnos de la partida. Junto con el valor del turno actual, esto permitirá a cada programa tener más control sobre las acciones a tomar según el momento del juego.
- La línea siguiente describirá la dimensión del territorio: un número natural indicando la cantidad de filas de la grilla seguido por otro que represente la cantidad de columnas. Pueden estar separados por una cantidad arbitraria de espacios.
- A continuación seguirá el rango de influencia de todas las grandas, ocupando una línea. Esto se representa mediante dos números naturales, de la manera usual.
- Luego de esto vendrá una línea en blanco.

Dependiendo del número de turno, el programa deberá comportarse de distintas maneras. En el turno inicial (i.e., `0`), cada línea subsiguiente mostrará los datos de cada soldado del ejército propio: una cadena de caracteres unívoca y arbitraria, sin espacios, que indique su nombre, seguida de un número natural que indique el nivel actual de energía del soldado, seguida a su vez de otros dos números naturales que representen su rango de influencia. Todos estos valores pueden estar separados, nuevamente, por una cantidad de espacios arbitraria.

Por otro lado, para cualquier otro turno, se indicarán primero los datos actuales de los soldados propios seguidos por los datos de los soldados del ejército rival. Estas listas de soldados se separarán por una línea en blanco, y siguen el mismo formato del párrafo anterior, pero agregando al final la posición actual del soldado.

5.1.2. Salida

Para el primer turno la salida debe mostrar el estado inicial de los soldados, ocupando exactamente una línea cada uno. Así, cada línea detallará las características del soldado representado: su nombre seguido en primer lugar por su energía, luego su rango de influencia y en última instancia su posición.

En cualquier otro turno, cada línea comenzará de igual manera que lo mencionado en el párrafo anterior. Inmediatamente después debe seguir un caracter que describa la acción tomada por este soldado (`A` para atacar; `D` para desplazarse), seguido a su vez de dos números naturales que indiquen la posición de ataque o de desplazamiento respectivamente. Como siempre, la cantidad de espacios de separación entre todos estos valores puede ser arbitraria.

5.2. El patrón de diseño *Strategy*

Un *patrón de diseño* es una solución reutilizable para un problema que ocurre comúnmente en el proceso de diseño de software. Para el caso de *Strategy*, éste se trata de un patrón que define una familia de algoritmos, cada uno representado con un objeto, y los hace intercambiables en tiempo de ejecución. *Strategy* permite que el algoritmo varíe en forma independiente del cliente que requiera su utilización [1].

Esto se logra definiendo una interfaz básica y homogénea para los algoritmos involucrados. Siguiendo esta interfaz, los clientes podrán interactuar de igual forma con cada uno de los algoritmos, sin tener que hacer distinciones innecesarias.

En C++, definiremos esta interfaz como una clase *abstracta* (esto es, una clase que no poseerá instancias). Allí se deberá definir el prototipo de cada mensaje que nuestros algoritmos deben saber responder. La implementación concreta de cada uno de ellos se hará en la definición de la clase de cada algoritmo. Por supuesto, cada una de ellas será subclase de la interfaz mencionada.

Los detalles restantes para la correcta implementación de este patrón quedará a criterio de cada grupo.

5.3. Competencia

Como hemos prometido en el trabajo anterior, los programas realizados por cada grupo tendrán la posibilidad de medirse en el campo de batalla. Si bien la implementación debe ser lo suficientemente genérica como para satisfacer la especificación presentada en este enunciado, para la competencia intergrupal fijaremos estos valores:

- Dimensión del territorio: 25×25
- Cantidad inicial de soldados: 15
- Rangos de influencia de los soldados:
  $\langle (5,5), (4,4), (4,4), (3,3), (3,3), (3,3), (3,3), (3,2), (3,2), (2,3), (2,3), (1,6), (6,1), (1,4), (4,1) \rangle$
- Cantidad máxima de turnos: 20
- Nivel inicial de energía: 100
- Rango de influencia de las granadas: $(2,2)$

La modalidad será eliminatoria directa, y el fixture será determinado en forma aleatoria al momento de hacer la simulación. Los tres primeros puestos gozarán de los siguientes premios:

- **Primer puesto**: un alfajor Cachafaz por cada integrante del grupo y un paquete grande de galletitas Toddy.
- **Segundo puesto**: un alfajor Cachafaz por cada integrante del grupo.
- **Tercer puesto**: un alfajor Guaymallén por cada integrante del grupo.

5.4. Línea de comando

Las opciones `-i` y `-o` permitirán seleccionar los streams de entrada y salida de datos respectivamente. Por defecto, éstos serán `cin` y `cout`. Lo mismo ocurrirá al recibir "`-`" como argumento de cada una.

Además, la opción `-m` la utilizaremos para describir el modo de juego. Ésta puede tomar **únicamente** los valores `agresivo`, `evasivo` o `default`, siendo este último el valor por defecto cuando la opción no esté presente.

Por otra parte, a través de la opción `-n` podremos conocer el nombre escogido por el grupo para representar al ejército. Dada esta opción, las opciones `-i` y `-m` mencionadas líneas arriba carecerán de efecto (i.e., su presencia o ausencia no debe alterar el comportamiento deseado de escribir el nombre a un archivo).

Al finalizar, todos nuestros programas retornarán un valor nulo en caso de no detectar ningún problema; y, en caso contrario, devolveremos un valor no nulo (por ejemplo `1`).

Como comentario adicional, el orden de las opciones es irrelevante. Por este motivo, no debe asumirse un orden particular de las mismas a la hora de desarrollar la toma de argumentos.

5.5. Ejemplos

Primero, usamos un archivo vacío como entrada del programa:

```
$ ./tp1 -i /dev/null
```

Observar que la salida es también vacía.

A continuación, lanzamos el programa con una entrada que sólo contiene espacios: por ende, en este caso, la salida deberá nuevamente ser vacía:

```
$ (echo; echo; echo) | ./tp1 -i - 2>&1 | wc -c
0
```

En lo que sigue veremos un caso de ejecución para el turno `0`:

```
$ cat entrada0.txt
0
20
10 5
2 2
```

```
zangief 10 2 4
goro 100 2 2
yoshimitsu 40 1 1
$ ./tp1 -i entrada0.txt -o salida.txt -m agresivo
$ cat salida.txt
zangief 10 2 4 4 2
goro 100 2 2 9 4
yoshimitsu 40 1 1 0 4
```

Notar que la opción `-m` no surte efecto sobre este escenario.
El ejemplo de abajo, por su parte, muestra una posible ejecución para un turno arbitrario no inicial:

```
$ cat entrada1.txt
4
20
10 5
2 2

zangief 1 2 4 5 3
goro 73 2 2 7 1

dhalsim 58 3 3 0 4
kazuya 35 1 4 9 2
reptile 21 4 2 0 0
$ ./tp1 -i entrada1.txt -o salida.txt -m default
$ cat salida.txt
zangief 1 2 4 5 3 D 6 4
goro 73 2 2 7 1 A 9 2
```

En otras palabras, el programa decidió que `goro` tire una granada a la posición $(9, 2)$ y que `zangief` se mueva al casillero $(6, 4)$.

Por último, a continuación podemos observar cómo reacciona el programa al solicitarle el nombre del ejército:

```
$ ./tp1 -i sin_uso.txt -o nombre.txt -m default -n
$ cat nombre.txt
expendables
```

De esta manera, el programa en cuestión comanda el ejército denominado `expendables`.

5.6. Portabilidad

Es deseable que la implementación desarrollada provea un grado mínimo de portabilidad. Sugerimos verificar nuestros programas en Windows/MS-DOS, usando el compilador DJGPP [2]; y/o alguna versión reciente de UNIX: BSD, o Linux[1].

6. Informe

El contenido mínimo del informe deberá incluir:

- Una carátula que incluya los nombres de los integrantes y el listado de todas las entregas realizadas hasta ese momento, con sus respectivas fechas.
- Documentación relevante al diseño e implementación del programa.
- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente.

[1] RedHat, SuSe, Debian, Ubuntu.

- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, en lenguaje C++ (en dos formatos, digital e impreso).
- Este enunciado.

7. Fechas

La última fecha de entrega y presentación será el jueves 21/11. Por otra parte, la competencia se realizará durante la clase del jueves 5/12.

Referencias

[1] E. Gamma, R. Helm, R. Johnson J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st Ed., Addison-Wesley, 1994. Págs. 315-325.

[2] DJGPP. `http://www.delorie.com/djgpp/`.