Programació 1

Enunciat de la pràctica

(Curs 2024/25 - Q1)

11-11-2024

NORMATIVA DE LA PRÀCTICA

Aquesta és la normativa de la pràctica. És important que la llegiu i la seguiu. Incomplir la normativa pot suposar el suspens de l'assignatura.

1. La qualificació obtinguda a la pràctica fa mitjana ponderada amb les notes dels controls i activitats amb un pes del 25% sobre la nota total de l'assignatura.
2. La pràctica és obligatòria. La no presentació de la pràctica resultarà en un NO PRESENTAT com a nota final de l'assignatura.
3. La nota de la pràctica està formada per dues parts: programa (NotaPrograma) i prova de validació individual (PVI). La nota de la PVI serà un número real entre 0 i 1. La fórmula per calcular la nota final de la pràctica és la següent: NotaPrograma * NotaPVI.
4. La pràctica es realitzarà en equips de màxim dues persones. Serà motiu de no acceptació de la pràctica l'haver estat realitzada per més de dues persones o per una sola persona.
5. La pràctica ha de compilar amb la comanda `pro1++` que heu estat fent servir durant el curs. Aquesta comanda té una sèrie de paràmetres que fan que la compilació sigui molt sensible a errors. **Procureu no tenir-ne cap. Si la pràctica no compila amb aquesta comanda, la nota final serà 0**.
6. Si dos o més treballs presentats són iguals, no serà acceptat **cap** dels treballs. El criteri del professor decidirà si dues pràctiques es poden considerar iguals o no. No s'admetran reclamacions sobre aquesta consideració.
7. Les pràctiques acceptades s'avaluaran mitjançant:

   - La correctesa i claredat del programa.
   - La correctesa de l'especificació i/o comentaris del programa.
   - L'execució del programa.

   Una pràctica que no funcioni pot ser igualment acceptada i avaluada pel professor, encara que, si no passa els jocs de prova, la nota de la pràctica baixarà un 50%.
8. El lliurament de la pràctica s'efectuarà a través del campus digital, en les condicions i format especificats a l'enunciat. L'incompliment de qualsevol d'aquestes condicions pot suposar la nota de NO PRESENTAT a la pràctica.
9. Un cop presentada la pràctica, tindrà lloc la Prova de Validació Individual que serà una defensa presencial. El professor farà públic un mecanisme (una llista per apuntar-se o similar) per tal de reservar data i hora per a cada membre de l'equip.
10. Els estudiants del mateix equip poden obtenir notes diferents a la PVI, resultant en notes de pràctica diferents per a cada un.

11. La no compareixença a la PVI d'un estudiant farà que la seva nota de pràctica sigui NO PRESENTAT, encara que hagi fet el lliurament de la pràctica.

1 Introducció

Volem construir un programa que sigui capaç de trobar la sortida d'un laberint rectangular. El programa controlarà un robot que haurà de buscar un camí que el porti des de l'entrada del laberint fins a la sortida.

El robot comença amb una quantitat d'energia determinada. A cada pas, el robot perd una unitat de la seva energia i, si la sortida és molt lluny, pot quedar completament descarregat abans d'arribar-hi. Afortunadament, al laberint de vegades es troben bateries de recanvi, que li permeten augmentar la seva reserva d'energia i allargar la distància que pot recórrer.

Degut a això, si el robot no té prou energia per anar a la sortida pel camí més curt, cal trobar un camí alternatiu més llarg, però on s'hi trobin bateries de recanvi suficients per recorre'l.

Aquest és un exemple d'un laberint de 8 files i 38 columnes, on l'entrada s'indica amb la lletra E, la sortida amb la lletra S, els # són parets que no es poden travessar, els . són caselles lliures on el robot pot desplaçar-se i els dígits són bateries de recanvi d'energia entre 1 i 9:

```
#.E.#######...###.....2.##########..#.
#.#.#.#.#..##.######.##.............#
#.#.#.#.#.##..#.##....##.##.#.######..
........#.##.##.#..##.##.########.1..#
####.##...........###....#..S.......##
....####.#.######..##.#.######.#######
###.####.#########.##.#.##............
.....5......##.......6.....###########
```

El programa ha de trobar un camí que permeti anar de l'entrada fins a la sortida amb l'energia inicial més la que es reculli pel camí. Se suposa que **el robot no es mou fins que ha decidit quin és el millor camí**, per tant, el programa explora tots els camins possibles (mostrant per la pantalla les exploracions que va fent) descartant aquells que no arriben a la sortida per quedar estancats, o per necessitar més energia de la disponible.

Per fer el programa, usarem una estructura de dades `laberint` predefinida, amb les següents propietats:

- El laberint està representat per una taula de dues dimensions, on cada casella és una posició possible del robot.
- Cada casella té unes coordenades, amb origen en (0,0) situat a la cantonada superior esquerra.
- Cada casella pot estar lliure o ocupada per un obstacle.
- Una casella lliure pot contenir bateries energètiques.

- El robot sols pot passar per les caselles lliures.
- El robot pot moure's verticalment o horitzontalment, però no en diagonal.
- El robot parteix de l'entrada del laberint i ha d'arribar, si hi ha un camí que hi porti, a la sortida.
- Cada casella visitada consumeix una unitat d'energia, incloent les caselles d'entrada i de sortida i també les que tenen bateries energètiques. En cas de visitar una casella amb bateria energètica, es consumeix una unitat d'energia i es recuperen les unitats de la bateria.

L'objectiu general de la pràctica és escriure un programa que permeti al robot trobar la sortida o sortides del laberint.

Per aconseguir-ho, cal seguir els passos que es donen en la resta del document.

2 Classes bàsiques

Abans de començar a construir el laberint i el programa que el recorre, definirem algunes classes bàsiques que ens resultaran útils més tard.

Les classes bàsiques a definir són:

- Classe `coord`: Ens permetrà tenir variables de tipus `coord` que guardin un parell de coordenades (x,y). Tindrem operacions per sumar coordenades i saber aixi les caselles veïnes en certa direcció (p.e. `(3,5)+(0,-1)=(3,4)`).
- Classe `direccio`: Contindrà la llista de direccions possibles a seguir des d'una coordenada (Nord/Sud/Est/Oest). Per tant, tindrem variables de tipus `direccio` amb operacions per saber la següent direcció de la llista, o per obtenir les components del desplaçament en aquella direcció (p.e. `Est -> (+1,0)`).
- Classe `casella`: Contindrà la informació d'una casella del laberint. Aquesta informació inclou les característiques de la casella (lliure, obstacle, entrada, sortida), la quantitat de recàrrega de bateria disponible (que pot ser zero), quines direccions hem explorat des de la casella, etc.
- Classe `laberint`: Guarda la taula de caselles del laberint. També ofereix mètodes per escriure el laberint a la pantalla, o per carregar un laberint des d'un fitxer.

El primer que haureu de fer és programar aquestes classes i uns programes que en provin el funcionament.

Per fer-ho, cal partir de l'esquelet dels fitxers .hpp que es troben a Atenea i que us facilitaran la feina de programar-les. Un cop completats (substituint els ???? per les instruccions adequades) es podran crear i implementar els fitxers .cpp

`coord.hpp`	Definició dels atributs i mètodes de la classe `coord`.
`direccio.hpp`	Definició dels atributs i mètodes de la classe `direccio`.
`casella.hpp`	Definició dels atributs i mètodes de la classe `casella`.
`laberint.hpp`	Definició dels atributs i mètodes de la classe `laberint`.

També trobareu els següents programes de prova, que cal completar i usar per verificar que heu programat correctament les classes anteriors.

`proves_coord.cpp`	Programa de prova de la classe `coord`.
`proves_direccio.cpp`	Programa de prova de la classe `direccio`.
`proves_casella.cpp`	Programa de prova de la classe `casella` .
`proves_laberint.cpp`	Programa de prova de la classe `laberint`.

- Tots els fitxers tenen comentaris amb pistes sobre què cal fer.

- A les sessions de laboratori es donaran més detalls sobre l'estructura del programa a construir.
- Per fer la constructora de la classe `laberint` (que carrega el laberint d'un fitxer) i el mètode `mostrar` (que dibuixa el laberint per la pantalla), us seran útils els suggeriments donats als apartats 2.1 i 2.2.

2.1 Càrrega del laberint

El programa haurà d'obtenir les dades inicials d'un fitxer de text que representa el laberint. Podeu obrir el fitxer com a `stream` de C++ utilitzant la llibreria `fstream`. Un cop obert, es llegeix amb l'operador `>>` o amb la funció `getline`, igual que ho feu amb qualsevol altre canal (p.e. `cin`).

El fitxer comença amb dos enters, que poden estar indistintament a la mateixa línia o en línies diferents, i que indiquen les dimensions (files × columnes).

Després de les dimensions, venen tantes línies com files té el laberint, i cada línia té tants caràcters com columnes hi ha al laberint. Per tant, cada caràcter correspon a una posició en el laberint.

Cal omplir cada posició del laberint amb els valors adequats segons el caràcter trobat, usant el mètode `omplir` de la classe `casella`.

Suposarem que el fitxer no conté errors, és a dir, el número de files i columnes coincideix amb les mides donades, el laberint conté exactament una entrada i una sortida i no hi ha cap caràcter apart dels autoritzats.

Per exemple, el següent és un laberint correcte (té les mides indicades, una entrada i una sortida, i tots els caràcters són vàlids):

```
8 38
#.E.#######...###.....2.##########..#.
#.#.#.#.#..##.######.##.............#
#.#.#.#.#.##..#.##....##.####.######..
........#.##.##.#..##.##.########.1..#
####.##...........###....#..S.......##
....####..#######..##.#.######.#######
###.####.#########.##.#.##...........
.....5......##.......6.....###########
```

2.2 Visualització de l'estat del laberint

A part de llegir i processar el laberint, el vostre programa haurà de visualitzar-lo, per mostrar l'evolució dels moviments del robot.

Per fer-ho heu de programar el mètode `mostrar()` de la classe `laberint` que l'imprimeix per pantalla amb un aspecte similar al que tenia el fitxer d'entrada, però mostrant, a més,

les caselles visitades pel robot fins al moment. Per fer-ho, cal cridar el mètode `mostrar()` de la classe `casella` que mostra les caselles visitades amb el caràcter 'o'.

Per exemple, quan el robot hagi visitat unes quantes caselles, obtindrem:

```
#.Eo#######...###.....2.##########..#.
#.#o#.#.#..##.######.##.............#
#.#o#.#.#o##..#.##....##.####.######..
...ooooo#o##.##.#..##.##.########.1..#
####.##ooo........###....#..S.......##
....####..#######..##.#.######.#######
###.####.#########.##.#.##...........
.....5......##.......6.....##########
```

3 Classe `laberint` i buscar la sortida

A continuació haureu de programar la cerca de la sortida (si n'hi ha), tant en versió iterativa com recursiva. El programa rebrà els següents paràmetres:

1. Nom del fitxer que conté el laberint.
2. Energia inicial.
3. Tipus de solució a aplicar ('i'-iterativa, 'r'-recursiva).
4. (opcional) Test pas a pas: Si hi ha un quart paràmetre, es mostrarà l'evolució del robot pas a pas. Si no hi és, només es mostrarà el resultat final.

Al principi, el programa mostrarà el laberint i l'energia inicial, després cercarà si hi ha un camí cap a la sortida. Si no n'hi ha, caldrà indicar-ho. En cas de que sí n'hi hagi caldrà també indicar-ho, es mostrarà el laberint amb el camí trobat i l'energia final que ens ha quedat.

Per fer-ho, cal partir d'un esquelet de programa principal:

`sortir.cpp` Esquelet programa que busca la sortida del laberint.

Us poden ser útils les següents funcions, definides en el fitxer `util.hpp`, que permeten dibuixar repetidament el laberint en el mateix lloc de la pantalla amb un temps determinat d'espera entre cada redibuix per simular una animació. Així podem veure el comportament del robot amb més facilitat:

```
//---- Neteja la pantalla
void neteja();
```

```
//---- Espera el numero de segons indicat abans de continuar.
//---- És útil per evitar que el programa s'executi tan depressa
//---- que no doni temps de veure com es va dibuixant el camí.
void espera(float);
```

- Tots els fitxers tenen comentaris amb pistes sobre què cal fer.
- A les sessions de laboratori es donaran més detalls sobre l'estructura del programa a construir.
- Per fer la funció iterativa que cerca un camí fins a la sortida, podeu seguir les pistes que trobareu a l'apartat 3.1.
- Per fer la funció recursiva que cerca un camí fins a la sortida, podeu seguir les pistes que trobareu a l'apartat 3.2.

3.1 Trobar la sortida amb una funció iterativa

Haureu de fer una versió iterativa del procediment per trobar la sortida del laberint. Per fer-ho, mantindrem una `stack<coord>` on guardarem el camí recorregut fins al moment. Al principi, la pila només contindrà la posició de l'entrada. Cada cop que canviem de casella, apilarem la posició de la nova. Així, el `cim` de la pila contindrà sempre la posició actual i la resta de la pila contindrà el camí que hem seguit per arribar-hi. Per tornar enrere desfent el camí sols caldrà desapilar.

La funció iterativa consistirà en un bucle que no pararà fins que trobem la sortida, o (si el laberint no té solució) fins que haguem explorat tots els camins possibles sense trobar-la. A cada iteració, caldrà mirar la posició actual (`cim` de la pila) i decidir si se segueix una direcció nova a partir d'allà, o bé si es torna enrere. Us pot ser molt útil definir un booleà per saber si en la iteració actual estem anant endavant o tornant enrera, per actuar en conseqüència.

Per portar el compte de quines direccions s'han seguit des de cada casella, cal usar els mètodes `direccio_actual()`, `queden_direccions()`, `avancar_direccions()` i `iniciar_direccions()` de la classe `casella`, que porten el compte de quines direccions s'han explorat des de cada casella.

També caldrà usar els mètodes `marcar()` i `desmarcar()` de la classe `casella` per tal de marcar la casella com a visitada quan anem a explorar una nova direcció, o desmarcar-la quan tornem enrera descartant aquell camí. Així evitem caure en un bucle infinit a l'evitar provar camins que passin dos cops pel mateix lloc.

Per tal de veure quins camins explora el robot, podem fer que la funció iterativa dibuixi l'estat del laberint a cada pas (només si `test pas a pas` és cert). Així es podran veure com evolucionen les caselles visitades.

3.2 Trobar la sortida amb una funció recursiva

Observeu que trobar un camí que porti des d'una posició qualsevol fins a la sortida es pot descriure recursivament com fer un pas cap a una casella veïna i seguidament trobar un camí que ens porti des d'aquesta fins a la sortida. Clarament es tracta d'un problema de recursivitat múltiple, ja que cada posició té quatre caselles veïnes.

També cal tenir en compte que, si al provar en una direcció ja hem trobat un camí fins a la sortida, no cal provar altres direccions.

Per tal de no caure en un bucle infinit i evitar provar camins que passin dos cops pel mateix lloc, cal marcar cada casella on hem estat com a *visitada*, usant el mètode `marcar()` de la classe `casella`. Igualment, si cap direcció troba el camí buscat, cal tornar enrere, deixant el laberint com estava (és a dir, desmarcant la casella).

Així doncs, la funció recursiva que busca el camí tornarà enrere (i, per tant, ja no explorarà les caselles veïnes doncs són casos directes) quan:

- La casella actual és la sortida (camí trobat, problema resolt).
- La casella actual és un obstacle (no es pot seguir per aquí).
- La casella actual és visitada (no cal anar-hi, ja hi hem passat).
- No queda energia (no tenim forces per seguir per aquest camí).

Observeu que no cal emmagatzemar les coordenades per les que va passant el robot per poder tornar enrere (el que abans teníem a la pila), ja que això s'emmagatzema en la seqüència de crides recursives, és a dir, la recursió tornarà enrere quan el robot hagi de recular perquè ha arribat a un cul de sac.

Per tal de veure quins camins explora el robot, podem fer que la funció recursiva dibuixi l'estat del laberint a cada pas (només si `test pas a pas` és cert). Així es podran veure com evolucionen les caselles visitades.

4 Lliurament

- *Data límit:* **Dilluns, 16 de desembre a les 23:59.**
- *Codi a presentar:*

Definicions classes	*Implementacions classes*	*programes principals*
`coord.hpp`	`coord.cpp`	
`direccio.hpp`	`direccio.cpp`	
`casella.hpp`	`casella.cpp`	
`laberint.hpp`	`laberint.cpp`	
`util.hpp`		
		`sortir.cpp`

- El lliurament s'ha de fer pel campus digital.
- El lliurament consistirà en **un únic** fitxer comprimit, amb nom `Cognom1.Nom1-Cognom2.Nom2.zip` format pel primer cognom i el nom dels membres del grup, per exemple: `Rios.Alex-Casas.Bernardino.zip`. Si el nom del fitxer no compleix aquest format, pot significar la qualificació de NO PRESENTAT per al lliurament.
- Aquest fitxer contindrà tots els fitxers del lliurament, sense cap altre subdirectori addicional.
- Només cal que un membre de l'equip faci el lliurament.
- El codi ha d'estar comentat descrivint quin és el resultat de cada funció (postcondició), i en quines condicions és aplicable (precondicio).
- Cal incloure els programes de prova, fitxers de dades, i qualsevol altre material necessari per a provar la pràctica (opcional).
- Un .zip incomplert pot significar la qualificació de NO PRESENTAT per al lliurament.