Exercici L2

El Joc dels daus

Fonaments de Programació II – Curs 2024-25

Objectiu

L'objectiu d'aquesta pràctica és que practiqueu amb el disseny d'un problema de manera modular, de manera que el programa estigui dividit en funcions que tinguin una responsabilitat específica. Per practicar-ho, implementarem un senzill joc de tirades de daus, on l'usuari competirà contra l'ordinador en llançament de diversos daus, i es comptabilitzarà qui guanya.

Aquest exercici busca que practiqueu la vostra capacitat per saber dividir el problema en subproblemes i implementar una solució modular.

Descripció del joc

- L'usuari i l'ordinador juguen un joc en el qual cadascun tira un dau de 6 cares.
- Quan comença el joc, es pregunta a l'usuari quantes tirades (`n`) es jugaran.
- Per a cada tirada:
    - Es genera un número aleatori entre 1 i 6 per a l'usuari.
    - Es genera un número aleatori entre 1 i 6 per a l'ordinador.
    - Es determina el guanyador de la tirada:
        - Si el número de l'usuari és més gran, l'usuari guanya la tirada.
        - Si el número de l'ordinador és més gran, l'ordinador guanya la tirada.
        - Si els números són iguals, la tirada no es comptabilitza.
- Al final de les `n` tirades, es determina qui ha guanyat més tirades:
    - Si l'usuari guanya més tirades, l'usuari és el guanyador del joc.
    - Si l'ordinador guanya més tirades, l'ordinador és el guanyador del joc.
    - Si hi ha el mateix nombre de tirades guanyades per cada jugador, el joc acaba en empat.
    - En qualsevol d'aquests casos, s'imprimeix un missatge per informar del resultat.

Requisits del disseny modular

El programa ha d'estar estructurat en funcions que compleixin els requisits que hem vist a la classe de teoria. Recordeu que per dissenyar l'estructura del programa heu de pensar:

1. Identificar l'objectiu global
2. Dividir l'objectiu global en objectius més petits (que seran les nostres funcions)
3. Dissenyar aquestes funcions (i testejar-les per separat!)
4. Unir totes les peces

Cada funció ha de tenir, a ser possible, una única finalitat. Reflexioneu també en les precondicions i postcondicions que cada funció ha de complir, fent que sigui la funció la responsable de comprovar la precondició.

Recordeu que també ha de tenir un nom explicatiu, han de ser reutilitzables, i no han de tenir efectes col·laterals inesperats.

Tasques a fer

- **Disseny Modular**: Desenvolupeu i implementeu les funcions anteriors i el programa principal. Abans de posar-vos a implementar-ho, feu un petit esbós de quines funcions dissenyareu i , comenteu amb el professor la vostra proposta.
- **Documentació**: Documenteu cadascuna de les funcions fent servir el format Doxygen. A l'Annex hi trobareu una breu explicació sobre aquest format.

Per reflexionar

Un cop implementat aquest exercici, com ho faries si canviéssim les regles del joc i, en comptes que cada jugador llencés un sol dau, en llencéssim tres, avaluant la suma dels tres daus? Reflexiona sobre com podries incloure aquest canvi en el teu programa i com això afectaria el disseny modular.

Annex: Documentació amb Doxygen

Doxygen és una eina per generar documentació a partir dels comentaris del codi. La documentació de cada funció es col·loca just abans de la seva definició[1].

Els comentaris s'escriuen utilitzant blocs de comentaris especials que comencen amb `/**` i es tanquen amb `*/`. L'estructura general de la sintaxi de Doxygen és la següent:

```
/**
 * @brief Breu descripció de la funció.
 * @param nom_parametre1 Descripció del primer paràmetre.
 * @param nom_parametre2 Descripció del segon paràmetre (si n'hi ha).
 * @return Descripció del valor que retorna la funció.
 */
```

Cada bloc de documentació hauria d'incloure les etiquetes següents, segons el cas:

- **@brief**: Descripció breu de la funció. Indica de manera concisa què fa la funció.
- **@details**: Descripció més detallada del comportament de la funció. És opcional però recomanable per a funcions complexes. (Si no s'especifica, Doxygen assumeix el contingut de **@brief** com a **@details** si aquest ocupa més d'una línia)
- **@param**: Per a cada paràmetre que rep la funció. Descriu el seu nom i el seu propòsit. El format és: `@param nom_del_parametre Descripció del paràmetre`.
- **@return**: Descripció del valor retornat per la funció. S'usa només si la funció retorna un valor.
- **@pre**[2]: Precondicions que han de ser satisfetes abans de cridar la funció.
- **@post**: Postcondicions que seran certes després d'executar la funció.
- **@note**: De manera opcional, aquí hi van les notes addicionals sobre l'ús de la funció, com per exemple comportaments especials.
- **@warning**: De manera opcional, aquí hi van les advertències sobre l'ús incorrecte de la funció o riscos potencials.

A la pàgina següent tens un exemple sobre la documentació d'una funció específica.

[1] Realment, aquesta documentació hauria d'anar damunt de la capçalera (declaració, no definició) de la funció, al corresponent fitxer header *.h. Com que encara no fem servir aquest tipus d'estructura, de moment ho posarem damunt de la definició. Però tingueu en compte que la manera correcta de fer-ho és al fitxer *.h

[2] Aquestes etiquetes no són part de l'estàndard bàsic de Doxygen, però s'utilitzen sovint per especificar precondicions i postcondicions.

```
/**
 * @brief Calcula el factorial de n de manera iterativa.
 *
 * @param n Enter no negatiu del qual es vol calcular el factorial.
 * @pre n >= 0.
 * @post Retorna n! com a enter positiu.
 * @return El factorial de n. Pot desbordar si n és massa gran.
 *
 * @note La funció assumeix que n és no-negatiu.
 * @warning Si n és negatiu, el resultat és indefinit i podria produir
errors.
*/

int factorial(int n) {
    int resultat = 1;
    for (int i = 1; i <= n; ++i) {
        resultat *= i;
    }
    return resultat;
}
```