**ReAkThor Fejlesztői Dokumentáció**

**Fejlesztők:**

* ***Kulcsár Dénes***
* ***Glonczi Tamás***

**Áttekintés**

Ez a C# program egy egyszerű reaktor szimulációt valósít meg, amely lehetővé teszi a felhasználó számára a reaktor beindítását, leállítását, hűtővíz beengedését és a programból való kilépést. A program folyamatosan figyeli a reaktor állapotát és hőmérsékletét, és megfelelően reagál a felhasználói beavatkozásokra. Ha a reaktor hőmérséklete meghalad egy kritikus értéket, a reaktor felrobban.

**Részletes Működés**

**Beindítás**

A reaktor beindítása során a program véletlenszerű késleltetést alkalmaz (1-5 másodperc), mielőtt a reaktort működő állapotba helyezi. A hőmérséklet és az energia értékek is véletlenszerűen növekednek a beindítás során. Ez szimulálja a valóságos reaktorok indítási folyamatát, ahol az idő és a feltételek változhatnak.

**Leállítás**

A reaktor leállítása csak akkor lehetséges, ha a hőmérséklet 70 fok alatt van. Ez a biztonsági mechanizmus biztosítja, hogy a reaktort nem lehet leállítani, ha az túl forró, mivel ez potenciálisan veszélyes lehet. Ha a hőmérséklet magasabb, a program figyelmezteti a felhasználót a magas hőmérsékletről, és kéri, hogy előbb hűtse le a reaktort.

**Hűtővíz Beengedése**

A hűtővíz beengedése során a program fokozatosan csökkenti a reaktor hőmérsékletét, amíg az el nem éri a 40 fokos biztonságos szintet. A hűtési folyamat során a program véletlenszerű késleltetést alkalmaz, hogy szimulálja a hűtővíz beengedésének időigényes folyamatát.

**Hőmérséklet Növekedés**

A program folyamatosan növeli a reaktor hőmérsékletét, amíg az működik. Ez a növekedés véletlenszerű késleltetésekkel történik, hogy szimulálja a reaktor normál működése során bekövetkező hőmérséklet-ingadozásokat. Ha a hőmérséklet meghaladja a 100 fokot, a reaktor felrobban, és a program véget ér.

**Osztályok és Metódusok**

**Reaktor Osztály**

**Mezők**

* `**private bool mukodik**`: A reaktor működési állapotát jelöli. Ha `true`, a reaktor működik; ha `false`, a reaktor leállt.
* `**private int homerseklet**`: A reaktor hőmérsékletét tárolja Celsius fokban.
* `**private int energia**`: A reaktor által termelt energia mennyiségét tartalmazza gigawattban (GW).

**Tulajdonságok**

* `**public bool Mukodik { get => mukodik; }**`: Csak olvasható tulajdonság a reaktor működési állapotának lekérdezésére.
* `**public int Homerseklet { get => homerseklet; }**`: Csak olvasható tulajdonság a reaktor hőmérsékletének lekérdezésére.
* `**public int Energia { get => energia; }**`: Csak olvasható tulajdonság a reaktor által termelt energia mennyiségének lekérdezésére.

**Konstruktor**

* `**public Reaktor()**`: Inicializálja a `mukodik` mezőt `false` értékre, a `homerseklet` mezőt 40 fokra, és az `energia` mezőt 0-ra. Ez biztosítja, hogy a reaktor kezdetben leállt állapotban és biztonságos hőmérsékleten legyen.

**Metódusok**

* `**public void Beinditas()**`:

1. Beindítja a reaktort egy véletlenszerű késleltetéssel (1-5 másodperc).
2. Beállítja a `mukodik` mezőt `true` értékre.
3. Véletlenszerűen növeli a hőmérsékletet 40 és 100 fok között.
4. Véletlenszerűen növeli az energia mennyiségét az előző értéknél nagyobb, de legfeljebb 10 GW-val, legfeljebb 100 GW-ig.

* `**public void Leallitas()**`:

1. Leállítja a reaktort, ha a hőmérséklet 70 fok alatt van.
2. Ha a hőmérséklet magasabb, figyelmezteti a felhasználót a magas hőmérsékletről.

* `**public void HutovizBeengedese()**`:

1. Fokozatosan csökkenti a hőmérsékletet 40 fokig, ha a reaktor működik.
2. Véletlenszerű késleltetésekkel szimulálja a hűtési folyamatot.

* `**public void Melegites()**`:

1. Növeli a hőmérsékletet egy fokkal véletlenszerű késleltetéssel, ha a reaktor működik.

* `**public void Menu()**`:

1. Megjeleníti a reaktor aktuális állapotát (hőmérséklet és energia), valamint a menüpontokat a konzolon.

**Program Osztály**

* `**static void Main(string[] args)**`: A program belépési pontja.Létrehozza a `**Reaktor**` példányát, és folyamatosan figyeli a felhasználói bemeneteket egy külön szálon. Az alábbi fő funkciókat látja el:

1. Létrehoz egy blokkoló gyűjteményt (`**BlockingCollection<ConsoleKeyInfo>**`) a billentyűlenyomások tárolására.
2. Egy új feladatot (`**Task**`) indít, amely folyamatosan olvassa a felhasználói bemeneteket és hozzáadja azokat a blokkoló gyűjteményhez.
3. Folyamatosan frissíti a konzolon megjelenített információkat a reaktor állapotáról.
4. Kezeli a felhasználói bemeneteket a menü opciók alapján (1-4 billentyűk).
5. Folyamatosan növeli a reaktor hőmérsékletét és ellenőrzi, hogy a hőmérséklet nem haladja-e meg a kritikus értéket (100 fok), ebben az esetben a program jelzi, hogy a reaktor felrobbant, és leállítja a programot.

**Fő Funkciók**

**Menü Kijelzése**

A program folyamatosan frissíti a konzolon megjelenő menüt és a reaktor aktuális állapotát. A menü tartalmazza a következő opciókat:

`**1. Beindítás**`: A reaktor beindítása.

`**2. Leállítás**`: A reaktor leállítása.

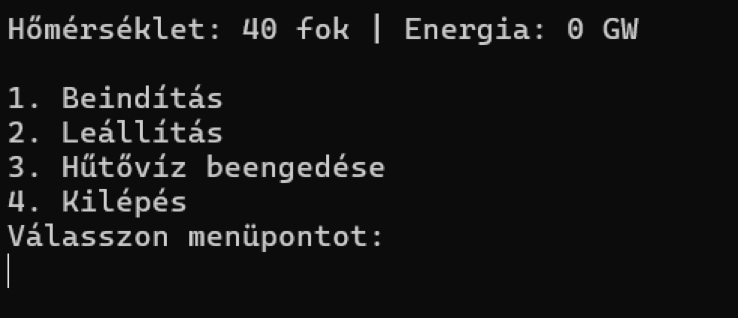
`**3. Hűtővíz beengedése**`: Hűtővíz beengedése a reaktor hőmérsékletének csökkentésére.

`**4. Kilépés**`: Kilépés a programból.

**Felhasználói Bemenetek Kezelése**

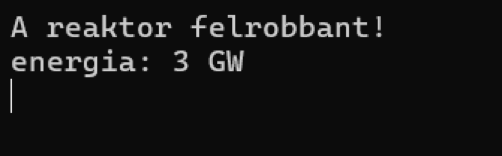
A program egy külön szálon olvassa a felhasználói bemeneteket és hozzáadja azokat egy blokkoló gyűjteményhez. Ez lehetővé teszi a program számára, hogy folyamatosan figyelje a felhasználói bemeneteket anélkül, hogy blokkolná a fő szálat. A bemenetek alapján a program a következő műveleteket hajtja végre:

1. `**ConsoleKey.D1**`: Beindítja a reaktort.
2. `**ConsoleKey.D2**`: Leállítja a reaktort.
3. `**ConsoleKey.D3**`: Hűtővizet enged be a reaktor hőmérsékletének csökkentésére.
4. `**ConsoleKey.D4**`: Kilép a programból.



**Hőmérséklet Növekedés és Robbanás**

A program folyamatosan növeli a reaktor hőmérsékletét véletlenszerű késleltetésekkel. Ha a hőmérséklet meghaladja a 100 fokot, a reaktor felrobban, és a program véget ér. A robbanás esetén a program kiírja a konzolra a "**A reaktor felrobbant!**" üzenetet és a reaktor által termelt energia mennyiségét.



**További Fejlesztési Lehetőségek**

1. **További Kontroll Mechanizmusok**:

* Automatikus hűtőrendszer bevezetése, amely automatikusan hűti a reaktort, ha a hőmérséklet egy bizonyos szint fölé emelkedik.
* Vészleállító mechanizmus, amely automatikusan leállítja a reaktort kritikus helyzetekben.

2. **Reaktor Naplózás**:

* Minden esemény és állapot változás naplózása egy fájlba a későbbi elemzés érdekében.
* A naplózás kiterjesztése, hogy részletes adatokat tartalmazzon a reaktor működéséről és a felhasználói beavatkozásokról.

3. **Grafikus Felület**:

* Egy grafikus felhasználói felület (GUI) bevezetése a konzol alapú megoldás helyett.
* A GUI lehetővé tenné a felhasználó számára, hogy könnyebben interakcióba lépjen a reaktorral és jobban megértse a reaktor állapotát.

4. **További Paraméterek**:

* Több reaktor paraméter hozzáadása, például nyomás, hűtőfolyadék szint, üzemanyag szint, stb.
* Ezek a paraméterek még realisztikusabbá tennék a szimulációt és bonyolultabb kontroll mechanizmusokat tennének lehetővé.

5. **Szimuláció Pontosítása**:

* Bonyolultabb szimulációk implementálása a valósághűbb működés érdekében.
* A reaktor fizikai és kémiai folyamatait pontosabban szimuláló algoritmusok bevezetése.

6. **Események és Riasztások**:

* Riasztási rendszer bevezetése, amely figyelmezteti a felhasználót kritikus eseményekre, például túlhevülésre vagy alacsony hűtőfolyadék szintre.
* Az események és riasztások logikájának finomítása, hogy a program még biztonságosabb és használhatóbb legyen.

7. **Több Reaktor Kezelése**:

* A program kiterjesztése több reaktor kezelésére, amelyek egymással is interakcióba léphetnek.
* Ez lehetővé tenné a felhasználó számára, hogy egy összetettebb reaktor hálózatot kezeljen és optimalizáljon.

**Kódkarbantartás és Bővítés**

A kódkarbantartás és bővítés során fontos figyelembe venni a következő szempontokat:

1. **Dokumentáció**:

* A kód jól dokumentált legyen, hogy más fejlesztők könnyen megérthessék és továbbfejleszthessék a programot.
* A dokumentáció frissítése minden kódváltoztatás után.

2. **Kód Struktúra**:

* A kód legyen jól szervezett és modularizált, hogy könnyen lehessen hozzáadni új funkciókat vagy módosítani a meglévőket.
* Az osztályok és metódusok logikus elrendezése és elnevezése.

3. **Tesztelés**:

* A kód rendszeres tesztelése, hogy biztosítsuk a megfelelő működést és a hibák korai felismerését.
* Automatizált tesztek írása a fő funkciókhoz.

4. **Biztonság**:

* A program biztonsági mechanizmusainak felülvizsgálata és frissítése a potenciális biztonsági rések kiküszöbölése érdekében.
* Az adatvédelem és a biztonságos kódolási gyakorlatok betartása.

5. **Teljesítmény Optimalizálás**:

* A kód teljesítményének folyamatos monitorozása és optimalizálása.
* Az erőforrás-használat minimalizálása és a program futási idejének csökkentése.

Ezek a szempontok biztosítják, hogy a program hosszú távon is fenntartható és könnyen továbbfejleszthető legyen.

**Verzió: 1.0**