Közös követelmények:

- A megvalósításnak felhasználóbarátnak, és könnyen kezelhetőnek kell lennie. A megjelenítéshez lehet vezérlőket használni, vagy elemi grafikát. Egyes feladatoknál különböző méretű játéktábla létrehozását kell megvalósítani, ekkor ügyelni kell arra, hogy az ablakméret mindig alkalmazkodjon a játéktábla méretéhez.
- A program szerkezetében törekedni kell az objektumorientált szemlélet megtartására (a főprogram kivételével a szerkezetnek teljesen objektumorientáltnak kell lennie, de nem kötelező a nézet és a modell szétválasztása).
- A dokumentációnak tartalmaznia kell a feladat elemzését, felhasználói eseteinek (WHEN-GIVEN-THEN szerkezetű) leírását (UML felhasználói esetek diagrammal kiegészítve), a program statikus szerkezetének leírását (UML osztálydiagrammal), valamint az esemény-eseménykezelő párosításokat és az eseménykezelő tevékenységek rövid leírását.

Feladatok:

1. **Misszionárius-kannibál**

   Készítsünk programot, amely bemutatja a misszionárius-kannibál problémát.

   Adott egy folyó, amelynek az egyik partján n darab kannibál és n darab misszionárius várakozik, hogy átkeljenek. Átkelésükhöz adott továbbá egy csónak, amely maximum k személyt tud egyszerre szállítani. Az átkelések alkalmával nem szabad, hogy egy időben akár a csónakban, akár valamelyik parton több kannibál legyen, mint misszionárius, mivel akkor megeszik a misszionáriusokat (kivéve, ha ott egyetlen misszionárius sem tartózkodik).

   Jelenítsük meg a két parton és a csónakban lévő misszionáriusokat és kannibálokat. Lehessen a két partról a csónakba, valamint visszahelyezni bárkit, illetve kezdeményezni átkelést, amely csak akkor történik meg, ha a játékállás a feltételeket az átkelés után is kielégíti.

   A program biztosítson lehetőséget új játék kezdésére n és k értékének megadásával (2-től 5-ig), és ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, hány lépéssel (átkeléssel) győzött a játékos, majd kezdjen automatikusan új játékot.

2. **Farkas-kecske-káposzta**

   Készítsünk programot, amely bemutatja a farkas-kecske-káposzta problémát.

   Adott egy folyó, amelynek az egyik partján egy farkas, egy kecske, egy káposzta található. Átkelésükhöz adott továbbá egy csónak, amely a csónakoson kívül legfeljebb egy dolgot (farkast, kecskét, vagy káposztát) tud egyszerre szállítani. Az átkelések alkalmával nem szabad, hogy valamelyik parton ott maradjon a farkas a kecskével, vagy a kecske a káposztával.

Jelenítsük meg a játék egy-egy állapotát: farkast, kecskét, vagy káposztát valamelyik parton vagy a csónakban, illetve a csónak helyét: egyik vagy másik part, illetve a folyó közepe. Lehessen a két partról a csónakba helyezni, valamint visszahelyezni farkast, kecskét, vagy káposztát, illetve kezdeményezni átkelést, amely csak akkor történik meg, ha a játékállás a feltételeket az átkelés során és után is kielégíti.
A program ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, hány lépéssel (átkeléssel) oldották meg a problémát, majd kezdjen automatikusan új játékot.

3. Átkelés a hídon

Készítsünk programot, amellyel az utazó kereskedők problémáját mutatjuk be.
Ebben néhány kereskedő éjszaka ér egy mély szakadék felett átívelő rozoga hídhoz. Ezen a hídon a sötétben csak lámpával lehet biztonságosan átkelni, de az utazóknak sajnos mindössze egyetlen lámpájuk van, továbbá egyszerre legfeljebb hárman juthatnak át, és valakinek vissza kell menni a lámpával a többiekért. A kereskedők különböző korúak (fiatal, középkorú, idős), ezért eltérő idő kell nekik a hídon való átkeléshez. Természetesen, amikor többen mennek át egyszerre, akkor a lassúbbhoz kell igazítani a lépést. Jelenítsük meg a szakadék két partján lévő kereskedőket, biztosítsuk azt, hogy mindig felváltva tudjunk egyik vagy másik partról 1-3 személyt kiválasztani, amely átkerül majd a másik oldalra. A program folyamatosan számolja az átkelés idejét (természetesen nem valós időben, hanem az előre megadott átkelési idők segítségével).
A program biztosítson lehetőséget új játék kezdésére a fiatal, középkorú és idős kereskedők számának megadásával (0-tól 5-ig, minimum 3 különböző összeállításból választva), és ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, milyen idővel győzött a játékos, majd kezdjen automatikusan új játékot.

4. Készítsen egy alakzat-rajzoló alkalmazást!

A program segítségével különféle alakzatokat (itt elég téglalapot és ellipszist) lehessen rajzolni egy adott téglalap alakú felületre.
A rajzolás az egér jobb fülének lenyomva tartásával történjen: amikor lenyomjuk (kezdő pozíció), elkezdődik a rajzolás, amely addig tart, amíg az egérfület felengedjük (végpozíció). Az alakzat a kezdő és a végpozíció által meghatározott téglalapban jelenjen meg. A két pozíció egy vízszintes-függőleges oldalú téglalap átlóját jelöli ki.
Azt, hogy éppen milyen alakzatot rajzolunk, egy rádiógomb segítségével (az ábrán ezeket alul találja, egy-egy bitmap képpel ellátva) lehessen kiválasztani.
A már korábban kirajzolt alakzatokat kék színnel, az éppen rajzolás alatt álló (kezdő és aktuális pozíció közötti) alakzatot piros színnel ábrázoljuk.
Tegyük lehetővé az eddig rajzolt alakzatok törlését például egy nyomógomb megnyomásának hatására.

5. Tili-toli

Készítsünk programot, amellyel a következő játékot lehet játszani.

Adott egy 4×4 mezőből tábla, amelyen véletlenszerűen elhelyezünk 15 számozott bábut (1, 2, ..., 15), egy mezőt pedig üresen hagyunk. A játékos feladata az, hogy a bábuk tologatásával kirakjuk a "sorfolytonos" sorrendet, vagyis a számok sorban következzenek az első sorban balról jobbra, majd a második sorban 5-től indulva balról jobbra, és így tovább. A tologatások során egy bábút áthelyezhetünk az egyetlen üres mezőre, ha azzal szomszédos mezőn áll (csak vízszintesen és függőlegesen lehet mozogni, átlósan nem).

A program biztosítson lehetőséget új játék kezdésére, és ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, hány lépéssel győzött a játékos, majd kezdjen automatikusan új játékot.

6. Rubik óra

Készítsünk programot, amellyel egy Rubik órát lehet kirakni.

A játéktáblán helyezzünk el 9 darab órát 3 sorba és 3 oszlopba rendezve, mindegyik 1-12 közötti értéket tud mutatni (kezdetben véletlenszerű beállítással). Azokon a helyeken, ahol négy óra sarka is összeér (összesen 4 ilyen pozíció van) legyen egy-egy nyomógomb is a táblán abból a célból, hogy egy lenyomással a kapcsolódó négy óra állását tudják eggyel meg lehessen növelni. Ennél fogva négy olyan óra lesz (a négy sarkokban), amit csak egy gombbal növelhetünk, négy (a négy oldalon), amit kettővel, és lesz egy óra (középen), amelyet mind a négy gomb növelhet. A játék célja az, hogy a gombok segítségével minden órát 12 órára állítsunk be.

A program biztosítson lehetőséget új játék kezdésére, és ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, hány lépéssel (állítással) győzött a játékos, majd kezdjen automatikusan új játékot.

7. Rubik tábla

Készítsünk programot, amellyel egy Rubik táblát lehet kirakni.

A Rubik tábla lényegében a Rubik-kocka két dimenziós változata. A játékban egy 4×4 mezőből álló táblán 4 különböző színű mező lehet, mindegyik színből pontosan n darab, kezdetben véletlenszerűen elhelyezve. A játék célja az egyes sorok, illetve oszlopok mozgatásával (ciklikus tologatásával, azaz ami a tábla egyik végén lecsúszik, az ellentétes végén megjelenik) egyszínűvé alakítani vagy a sorokat, vagy az oszlopokat (azaz vízszintesen, vagy függőlegesen csíkokat kialakítani).

A program ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, hány lépéssel győzött a játékos, majd automatikusan kezdjen új játékot.

8. Királynők

Készítsünk programot, amellyel a következő játékot lehet játszani.
Adott egy 8×8-es tábla, melyen királynőket helyezhetünk el sorban egymás után. A tábla kezdetben üres, és a játék célja, hogy elhelyezzünk 8 királynőt úgy, hogy azok közül semelyik kettő ne üsse egymást (vízszintesen, függőlegesen, vagy átlósan). Minden elhelyezés után jelöljük meg a táblán azokat a mezőket, ahova már nem rakhatunk újabb királynőt (amelyeket az eddig elhelyezett bábuk ütnek), és természetesen ne is engedjük ezeket a mezőket használni. A lehelyezett királynőt lehessen visszavenni, ekkor a program szabadítsa fel a megfelelő mezőket.
A program ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, hány lépéssel győzött a játékos, majd automatikusan kezdjen új játékot.

9. Lovagi torna

Készítsünk programot, amellyel a következő két személyes játékot lehet játszani. Adott egy 8×8 mezőből álló tábla, amelynek a négy sarkába 2 fehér, illetve2 fekete ló figurát helyezünk el (az azonos színűek ellentétes sarokban kezdenek).
A játékosok felváltva lépnek, a figurák L alakban tudnak mozogni a játéktáblán. Kezdetben a teljes játéktábla szürke színű, de minden egyes lépés után az adott mező felveszi a rá lépő figura színét (bármilyen színű volt előtte). A játék célja, hogy valamely játékosnak függőlegesen, vízszintesen, vagy átlósan egymás mellett 4 ugyanolyan színű mezője legyen. Ha ezt valamelyik játékos elérte, vége a játéknak.
A program ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, melyik játékos győzött, majd automatikusan kezdjen új játékot.

10. Áttörés

Készítsünk programot, amellyel a következő kétszemélyes játékot lehet játszani. Adott egy 8×8 mezőből álló tábla, ahol a két játékos bábui egymással szemben helyezkednek el, két sorban (pont, mint egy sakktáblán, így mindkét játékos 16 bábuval rendelkezik, ám mindegyik bábu ugyanolyan típusú). A játékos bábúival csak előre léphet egyenesen, vagy átlósan egy mezőt (azaz oldalra, és hátra felé nem léphet), és hasonlóan ütheti a másik játékos bábúját előre átlósan (egyenesen nem támadhat). Az a játékos győz, aki először átér a játéktábla másik végére egy bábuval.
A program ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, melyik játékos győzött, majd automatikusan kezdjen új játékot.

11. 4-es játék

Készítsünk programot, amellyel a következő két személyes játékot lehet játszani.
Adott egy 5×5 mezőből álló tábla, amelynek mezői 0 és 4 közötti értékeket tartalmaznak. Kezdetben minden mezőn a 0 érték van. Ha a soron következő játékos a tábla egy tetszőleges mezőjét kiválasztja, akkor az adott mezőn és a szomszédos

négy mezőn az aktuális érték eggyel nő felfelé, ha az még kisebb, mint 4. Aki a lépésével egy, vagy több mező értékét 4-re állítja, annyi pontot kap, ahány mezővel ezt megtette. A játékosok pontjait folyamatosan számoljuk, és a játékmezőn eltérő színnel jelezzük, hogy azt melyik játékos billentette 4-esre. A játék akkor ér véget, amikor minden mező értéke 4-et mutat. Az győz, akinek ekkor több pontja van.
A program ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, melyik játékos győzött, majd automatikusan kezdjen új játékot.

12. Reflex teszt

Készítsünk Qt alkalmazást az alábbi játékhoz. A játékfelületen $4 \times 4 + 1$ mező foglal helyet 4 különböző színben (piros, sárga, zöld, kék), ebből 4×4 alkot egy táblát, 1 pedig egy kitüntetett ún. kijelölőmező. A táblán a színek meghatározott időközönként (mondjuk 2 másodpercenként) váltakozzanak. A kijelölő mezőn megjelenik a színek egyike, és a játékos feladata, hogy olyan színű mezőt találjon a játéktáblán, amit a kijelölő mező is mutat, és kattintson rá. A kattintást követően a kijelölő mező színe változzon meg. A program jelenítse meg a játékos eredményét (jó/rossz találatok aránya), kérésre lehessen bármikor a játékot szüneteltetni (ekkor nem telik az idő, de nem is lehet lépni), illetve új játékot kezdeni.