2. beadandó feladat: kétrétegű grafikus felületű alkalmazás

Közös követelmények:

kommunikál a nézettel.

- A programot kétrétegű (modell/nézet) architektúrában kell felépíteni, amelyben a megjelenítés rétege elkülönül a játéklogikától. A modell nem tartalmazhat semmilyen grafikus felületbeli osztályra történő hivatkozást, csak eseményeket küldhet a grafikus felületnek (pl.: játék vége / frissíteni kell a játéktáblát / érvénytelen lépés / idő túllépés vagy idő lejárta). Ilyen esemény küldés mindenképpen legyen az alkalmazásban. A nézet nem tartalmazhat semmilyen játékbeli adatot, és nem végezheti a betöltést és mentést.
- A program játékfelületét dinamikusan kell létrehozni futási időben. A megjelenítéshez lehet vezérlőket használni, elemi grafikát, vagy grafikus képernyőt. Egyes feladatoknál különböző méretű játéktábla létrehozását kell megvalósítani, ekkor ügyelni kell arra, hogy az ablakméret mindig alkalmazkodjon a játéktábla méretéhez.
- Azon feladatoknál, ahol szüneteltetni is lehet, szünet alatt ne menjen a játék, és a játékos se tudjon tevékenységet végezni.
- 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 tevékenység rövid leírását.
- A program modelljéhez automatikusan futtatható egység-teszteket kell készíteni.

Feladatok:

1. **Maci Laci**

   Készítsünk programot, amellyel a következő játékot játszhatjuk.
   Adott egy $n \times n$ mezőből álló erdő, amelyben Maci Lacival kell piknikkosarakra vadásznunk, amelyek a játékpályán helyezkednek el. A játék célja, hogy Maci Laci irányításával a piknikkosarakat minél gyorsabban begyűjtsük.
   A játékpályán a piknikkosarak mellett akadályok (pl. fa) is elhelyezkedhetnek, amelyekre nem léphetünk. A pályán emellett vadőrök is járőröznek, akik adott időközönként lépnek egy mezőt (vízszintesen, vagy függőlegesen). A járőrözés során egy megadott irányba haladnak egészen addig, amíg akadályba (vagy az pálya szélébe) nem ütköznek, ekkor megfordulnak, és visszafelé haladnak (tehát

folyamatosan egy vonalban járőröznek). Egy vadőr a járőrözés közben a vele szomszédos mezőket látja (átlósan is, azaz egy 3×3-as négyzetet).
A játékos kezdetben a bal felső sarokban helyezkedik el, és vízszintesen, illetve függőlegesen mozoghat (egyesével) a pályán, a piknikkosárra való rálépéssel pedig felveheti azt. Ha Maci Lacit meglátja valamelyik vadőr, akkor a játékos veszít.
A pályák méretét, illetve felépítését (piknikkosarak, akadályok, vadőrök kezdőpozíciója) tárolhatjuk fájlban, vagy létrehozhatjuk véletlenszerűen (előre rögzített paraméterek mellett). A programot legalább 3 különböző méretű pályával lehessen használni.
A program biztosítson lehetőséget új játék kezdésére a pálya kiválasztásával, valamint játék szüneteltetésére (ekkor nem telik az idő, és nem léphet a játékos). Ismerje fel, ha vége a játéknak, és jelezze, győzött, vagy veszített a játékos. A program játék közben folyamatosan jelezze ki a játékidőt, valamint a megszerzett piknikkosarak számát.

2. Bombázó

Készítsünk programot, amellyel a következő játékot játszhatjuk.
Adott egy $n \times n$ mezőből álló játékpálya, amelyen az ellenfelek járőröznek. A játék célja, hogy a játékosunkat úgy irányítsuk, hogy az ellenfeleket bombák segítségével minél gyorsabban legyőzze.
A játékpályán a bejárható mezők mellett falak is elhelyezkednek, amelyekre nem léphetünk. Az ellenfelek adott időközönként lépnek egy mezőt (vízszintesen, vagy függőlegesen) úgy, hogy folyamatosan előre haladnak egészen addig, amíg falba nem ütköznek. Ekkor véletlenszerűen választanak egy új irányt, és arra haladnak tovább.
A játékos figurája kezdetben a bal felső sarokban helyezkedik el, és vízszintesen, illetve függőlegesen mozoghat (egyesével) a pályán, de ha találkozik valamely ellenféllel (ugyanarra a mezőre lép), akkor meghal.
A játékos bombát rakhat le az aktuális pozíciójára, amely rövid időn belül robban megsemmisítve a 3 sugáron belül (azaz egy 7×7-es négyzetben) található ellenfeleket (falon át is), illetve magát a játékost is, ha nem menekül onnan.
A pályák méretét, illetve felépítését (falak, ellenfelek kezdőpozíciója) tároljuk fájlban, vagy hozzuk létre véletlenszerűen (előre rögzített paraméterek mellett). A programot legalább 3 különböző méretű pályával lehessen használni.
A program biztosítson lehetőséget új játék kezdésére a pálya kiválasztásával, valamint játék szüneteltetésére (ekkor nem telik az idő, és nem léphet a játékos). Ismerje fel, ha vége a játéknak, és jelezze, győzött, vagy veszített a játékos. A program játék közben folyamatosan jelezze ki a játékidőt, valamint a felrobbantott ellenfelek számát.

3. Elszabadult robot

Készítsünk programot, amellyel a következő játékot játszhatjuk.
Adott egy $n \times n$ mezőből álló játékpálya, amelyben egy elszabadult robot bolyong, és a feladatunk az, hogy betereljük a pálya közepén található mágnes alá, és így elkapjuk.
A robot véletlenszerű pozícióban kezd, és adott időközönként lép egy mezőt (vízszintesen, vagy függőlegesen) folyamatosan előre haladva egészen addig, amíg falba nem ütközik. Ekkor véletlenszerűen választ egy új irányt, és arra halad tovább.
A játékos a robot terelését úgy hajthatja végre, hogy egy mezőt kiválasztva falat emelhet rá. A felhúzott falak sajnos nem túl strapabíróak. Ha a robot ütközik a fallal, akkor az utána eldől. A ledőlt falakat már nem lehet újra felhúzni, ott a robot később akadály nélkül áthaladhat.
A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (7×7, 11×11, 15×15), valamint játék szüneteltetésére (ekkor nem telik az idő, nem lép a robot, és nem lehet mezőt se kiválasztani). Ismerje fel, ha vége a játéknak, és jelenítse meg, hogy milyen idővel győzött a játékos. A program játék közben folyamatosan jelezze ki a játékidőt.

4. Labirintus

Készítsünk programot, amellyel a következő játékot játszhatjuk.
Adott egy $n \times n$ elemből álló játékpálya, amely labirintusként épül fel, azaz fal, illetve padló mezők találhatóak benne, illetve egy kijárat a jobb felső sarokban. A játék célja az, hogy egy játékost minél előbb kivezessünk a labirintusból a bal alsó sarokból indulva.
A labirintusban nincs világítás, csak egy fáklyát visz a játékos, amely a 2 szomszédos mezőt világítja meg (azaz egy 5×5-ös négyzetet), de a falakon nem tud átvilágítani.
A játékos figurája kezdetben a bal alsó sarokban helyezkedik el, és vízszintesen, illetve függőlegesen mozoghat (egyesével) a pályán.
A pályák méretét, illetve felépítését (falak, padlók) tároljuk fájlban. A programot legalább 3 különböző méretű pályával lehessen használni.
A program biztosítson lehetőséget új játék kezdésére a pálya kiválasztásával, valamint játék szüneteltetésére (ekkor nem telik az idő, és nem léphet a játékos), továbbá ismerje fel, ha vége a játéknak. A program játék közben folyamatosan jelezze ki a játékidőt.

5. Menekülj

Készítsünk programot, amellyel a következő játékot játszhatjuk.
Adott egy $n \times n$ mezőből álló játékpálya, ahol egy játékost kell úgy irányítanunk, hogy elmeneküljön két üldöző elől, illetve próbálja őket aknára csalni.

Kezdetben a játékos játékpálya felső sorának közepén helyezkedik el, a két üldöző pedig az alsó két sarokban. Az ellenfelek adott időközönként lépnek egy mezőt a játékos felé haladva úgy, hogy ha a függőleges távolság a nagyobb, akkor függőlegesen, ellenkező esetben vízszintesen mozognak a játékos felé.
A pályán véletlenszerű pozíciókban aknák is elhelyezkednek, amelyekbe az ellenfelek könnyen beleléphetnek, ekkor eltűnnek (az akna megmarad).
A játékos vízszintesen, illetve függőlegesen mozoghat (egyesével) a pályán, és célja, hogy az ellenfeleket aknára csalja, miközben ő nem lép aknára. Ha sikerül minden üldözőt aknára csalnia, akkor győzött, ha valamely ellenfél elkapja (egy pozíciót foglal el vele), vagy aknára lép, akkor veszített.
A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (11×11, 15×15, 21×21), valamint játék szüneteltetésére (ekkor nem telik az idő, és nem léphet senki). Ismerje fel, ha vége a játéknak, és jelenítse meg, hogy győzött, vagy veszített-e a játékos. Ezen felül szüneteltetés alatt legyen lehetőség a játék elmentésére, valamint betöltésére. A program játék közben folyamatosan jelezze ki a játékidőt.

6. Lopakodó

Készítsünk programot, amellyel a következő játékot játszhatjuk.
Adott egy $n \times n$ mezőből álló játékpálya, amely falakból és padlóból áll, valamint őrök járőröznek rajta. A játék célja, hogy egy játékost a kiindulási pontból eljuttassuk a kijáratig úgy, hogy közben az őrök nem látják meg. Természetesen a játékos, illetve az őrök csak padlón tudnak járni.
Az őrök adott időközönként lépnek egy mezőt (vízszintesen, vagy függőlegesen) úgy, hogy folyamatosan előre haladnak egészen addig, amíg falba nem ütköznek. Ekkor véletlenszerűen választanak egy új irányt, és arra haladnak tovább. Az őr járőrözés közben egy 2 sugarú körben lát (azaz egy 5×5-ös négyzetet), ám a falon nem képes átlátni.
A játékos a pálya előre megadott pontján kezd, és vízszintesen, illetve függőlegesen mozoghat (egyesével) a pályán.
A pályák méretét, illetve felépítését (falak és kijárat helyzete, játékos és őrök kezdőpozíciója) tárolhatjuk fájlban, vagy létrehozhatjuk véletlenszerűen (előre rögzített paraméterek mellett). A programot legalább 3 különböző méretű pályával lehessen használni.
A program biztosítson lehetőséget új játék kezdésére a pálya kiválasztásával, valamint játék szüneteltetésére (ekkor nem telik az idő, és nem léphet a játékos). Továbbá ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, hogy győzött, vagy veszített-e a játékos.

7. Snake

Készítsük programot, amellyel a klasszikus kígyó játékot játszhatjuk.

Adott egy $n \times n$ mezőből álló játékpálya, amelyben akadályok (falak) találhatóak. A játékos egy kezdetben 5 hosszú kígyóval indul a képernyő közepén, amely vízszintesen, illetve függőlegesen halad rögzített időközönként a legutoljára beállított irányba. A kígyóval elfordulhatunk balra, illetve jobbra. A pályán véletlenszerű pozícióban mindig megjelenik egy tojás, amelyet a kígyóval meg kell etetni. Minden etetéssel eggyel nagyobb lesz a kígyó. A játék célja, hogy a kígyó minél tovább elkerülje az ütközést az akadályokkal, a pálya szélével, illetve saját magával.
A pályák méretét, illetve felépítését (falak helyzete) tárolhatjuk fájlban, vagy létrehozhatjuk véletlenszerűen (előre rögzített paraméterek mellett). A programot legalább 3 különböző méretű pályával lehessen használni.
A program biztosítson lehetőséget új játék kezdésére a pálya kiválasztásával, valamint játék szüneteltetésére (ekkor nem telik az idő, és nem mozog a kígyó). Továbbá ismerje fel, ha vége a játéknak. Ekkor jelenítse meg, hány tojást sikerült elfogyasztania a játékosnak.

8. Fénymotor párbaj

Készítsük programot, amellyel a Tronból ismert fénymotor párbajt játszhatjuk.
Adott egy $n \times n$ mezőből álló játékpálya. A két játékos a bal, illetve jobb oldal közepén indul egy-egy fénymotorral, amely egyenesen halad (rögzített időközönként) a legutoljára beállított irányba (függőlegesen, vagy vízszintesen). A motorokkal lehetőség van balra, illetve jobbra fordulni. A fénymotor mozgás közben fénycsíkot húz, ami a játék végéig ott marad. Az a játékos veszít, aki előbb nekiütközik a másik játékos motorjának, bármelyikük fénycsíkjának vagy a pálya szélének.
A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (12×12, 24×24, 36×36), valamint játék szüneteltetésére (ekkor nem telik az idő, és nem mozognak a motorok). Ismerje fel, ha vége a játéknak, és jelenítse meg, melyik játékos győzött.

9. Sudoku

Készítsünk programot a közismert Sudoku játékra.
Adott egy $\mathbf{9} \times \mathbf{9}$ mezőből álló tábla, amelyet úgy kell a 1-9 számjegyekkel kitölteni, hogy minden sorában, minden oszlopában és minden házában egy számjegy pontosan egyszer szerepeljen. (Háznak a $\mathbf{9} \times \mathbf{9}$-es táblát lefedő, de egymásba át nem érő kilenc darab $\mathbf{3} \times \mathbf{3}$ résztáblát nevezzük.)
A 81 darab kis négyzet bármelyikét kiválasztva a felirata változzon meg. Az üres felirat helyett 1-esre, az 1-es helyett 2-esre, és így tovább, végül a 9-es helyett üresre, azonban csak szabályos számokat engedélyezzen a program (az ütköző számokat egyszerűen ugorja át). Ennek megfelelően bármelyik négyzeten néhány (legfeljebb kilenc) kiválasztással egy tetszőleges érték állítható be. A program

ellenőrizze, hogy az adott négyzetbe írt érték nem szerepel-e a négyzetet tartalmazó sorban, oszlopban illetve házban.
A program biztosítson lehetőséget új játék kezdésére a kezdőtábla betöltésével, játék szüneteltetésére (ekkor nem telik az idő, de nem léphet a játékos), illetve mentésére. A betöltött mezők értékeit utólag ne lehessen módosítani (ezeket jelölje külön a program). A program folyamatosan jelezze a játékos gondolkodási idejét. A program „sürgetős" üzemmódban is tudjon működni. Ilyenkor a felhasználónak egy adott időn (ez legyen beállítható) belül kell a soron következő lépését megtenni, különben veszít.

10. Aknakereső

Készítsünk programot, amellyel az aknakereső játék két személyes változatát játszhatjuk.
Adott egy $n \times n$ mezőből álló tábla, amelyen rejtett aknákat helyezünk el. Egy mezőre lépve – feltéve, hogy nem robbantunk fel – megtudhatjuk, hogy a vele szomszédos 8 mezőn hány akna helyezkedik el.
A játékosok felváltva léphetnek. Minden játékosnak adott idő alatt kell lépnie, különben felrobban. Egy mezőre lépve felfedjük annak tartalmát. Ha az akna, a játékos veszített. Amennyiben a mező nullát rejt, akkor a vele szomszédos mezők is automatikusan felfedésre kerülnek (és ha a szomszédos is nulla, akkor annak szomszédjait is, és így tovább). A játék addig tart, amíg valamelyik játékos aknára nem lép, vagy fel nem fedték az összes nem aknamezőt (ekkor döntetlen lesz a játék).
A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (6×6, 10×10, 16×16), valamint játék mentésére és betöltésére. Ismerje fel, ha vége a játéknak, és jelenítse meg, melyik játékos győzött, illetve azt is, ha döntetlen lett a vége.