Description:

Életjáték

Nagybeadandó

A feladatok egymásra épülnek, ezért érdemes a leírás sorrendjében megoldani őket! A függvények definíciójában lehet, sőt javasolt is alkalmazni a korábban definiált függvényeket.

Tekintve, hogy a tesztesetek, bár odafigyelés mellett íródnak, nem fedik le minden esetben a függvény teljes működését, határozottan javasolt még külön próbálgatni a megoldásokat beadás előtt!

A feladat összefoglaló leírása

Az életjáték („the game of life") egy speciális sejtautomata, amit John Conway angol matematikus fejlesztett ki 1970-ben. Rengeteg különböző változata létezik, a következőkben leírt variáns a klasszikusnak számító változat.

Az életjáték végtelen kétdimenziós térben, egy négyzetrácson játszódik. A négyzetrácsot *cellák* alkotják. Minden cellában maximum egy élő *sejt* lehet. Gyakori ábrázolási mód, hogy az üres cellákat fehér, míg az élő sejtet tartalmazó cellákat fekete négyzetek jelölik, így egy kép rajzolható ki az aktuális állapotról.

A játék nem igényel aktív beavatkozást, elegendő csupán a sejtek kezdeti állapotát megadni. Ez után minden körben kiszámítunk egy új generációt, tehát azoknak a celláknak a helyét, ahol a lépés után élő sejtek lesznek. Három eset lehetséges: egy sejt életben maradhat a következő generációra, meghalhat, vagy új sejt születhet egy üres cellában. Ezt a következő szabályok határozzák meg.

1. A sejt *túléli* a kört, ha két vagy három szomszédja van.
2. A sejt elpusztul, ha kettőnél kevesebb, vagy háromnál több szomszédja van. Az előbbit *elszigetelődés*nek, az utóbbit pedig *túlnépesedés*nek nevezzük.
3. Új sejt *születik* minden olyan cellában, melynek környezetében pontosan három sejt található.

Egy cella környezete a hozzá legközelebb eső 8 mező (tehát a cellához képest átlósan elhelyezkedő cellákat is figyelembe vesszük, és feltesszük, hogy a négyzetrácsnak nincs széle).

Minden cellát a koordinátái írnak le. A koordináta rendszer *x* tengelye jobbra növekszik, *az y tengelye pedig lefelé*. Mivel a tér végtelen, negatív koordináták is megengedettek, de csak egész értékekkel dolgozhatunk.

Ügyeljünk arra, hogy a koordináták előbb az y, majd az x tengelyen vett pozíciót tartalmazzák, a számítógépes grafikában megszokott módon!

```
type Coordinate = (Integer, Integer)
```

A játék egy állapota az aktuális generáció (az élő sejtek) celláinak megadásából áll:

```
type Generation = [Coordinate]
```

Egy lépés során tehát az a dolgunk, hogy az aktuális generációból kiszámítsuk az új generációt a fenti szabályok alapján. *Ez minden cellára szimultán történik, tehát az új generáció kiszámítása során csak az előző generációt vesszük figyelembe, a már éppen megszülető vagy elpusztuló sejteket nem!*

Példák

Az alábbi példákat vagy másoljuk be a megoldásunkba, vagy töltsük le a feladathoz csatolt base filet!

Elszigetelődött sejt

Egyetlen sejt, amely a következő generációra elszigetelődés miatt kihal:

```
single :: Generation
single = [ (42, 42) ]
```

Ne feledjük, mivel egy minden irányba végtelen sejtautomatát képzelünk el, ahol csak ez az egyetlen élő sejt, így pozíciók bárhol lehetnek ugyanazt az állapotot írják le.

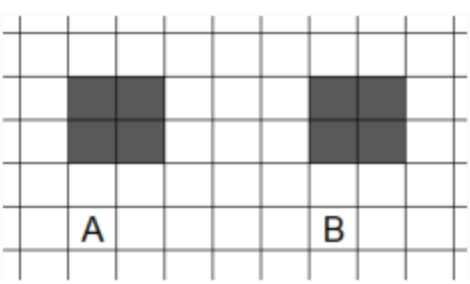
Pl. a következő `Generation` megegyezik az előzővel.

```
single2 :: Generation
single2 = [ (93, 16) ]
```

Csendélet

Csendélet vagy blokk, ahol mindig ugyanezek a sejtek maradnak életben, és nem születnek újak sem:

```
block :: Generation
block = [ (5, 6), (5, 7)
        , (6, 6), (6, 7) ]
```

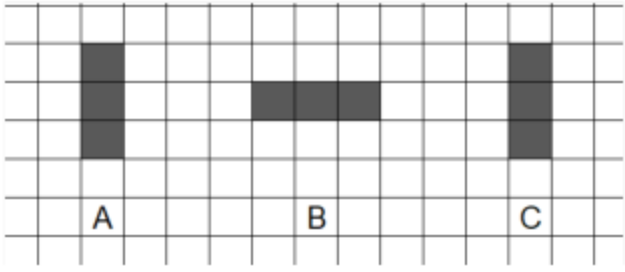


Oszcillátor

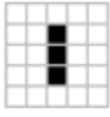
Kétlépéses oszcillátor, ahol az egyik folyton a másikba alakul egy lépés után. Valójában így egy alakzatnak számítanak, ennek a neve `blinker`.

```
row :: Generation
row = [ (10, 1), (10, 2), (10, 3) ]
```

```
column :: Generation
column = [ (9,  2)
         , (10, 2)
         , (11, 2) ]
```

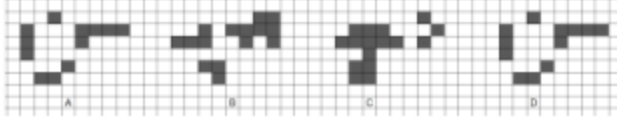


Blinker animálva:

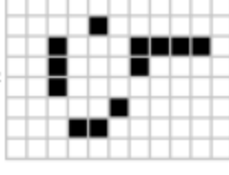


Háromlépéses oszcillátor, amely önmagába tér vissza három lépés után:

```
caterer :: Generation
caterer = [ (2, 4), (3, 2), (3, 6), (3, 7), (3, 8), (3, 9)
          , (4, 2), (4, 6), (5, 2), (6, 5), (7, 3), (7, 4) ]
```



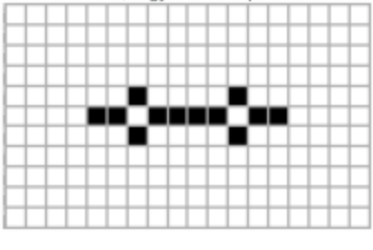
Caterer animálva:



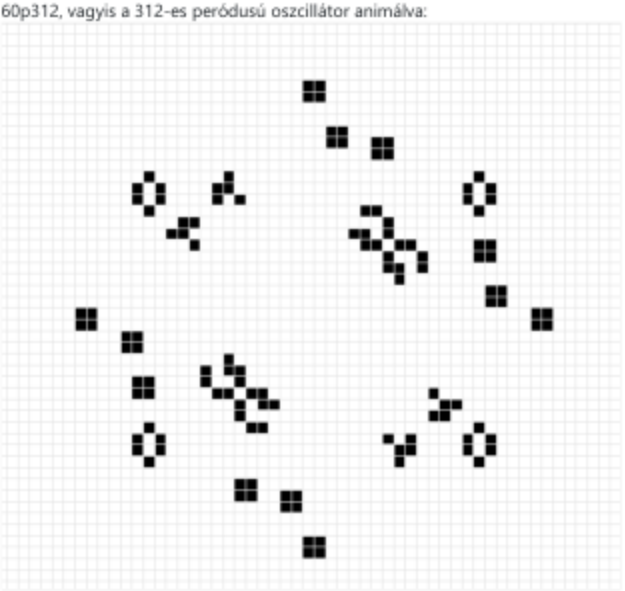
15-ös és 312-es periódusú oszcillátorok:

```
o15 :: Generation
o15 = [(0,1),(1,1),(2,0),(2,2),(3,1),(4,1),(5,1),(6,1),(7,0),(7,2),(8,1)

o312 :: Generation
o312 = [(1,21),(1,22),(2,21),(2,22),(6,32),(6,33),(7,23),(7,24),(7,31),(
```

Pentadecathlon, vagyis a 15-ös peródusú oszcillátor animálva:



60p312, vagyis a 312-es peródusú oszcillátor animálva:



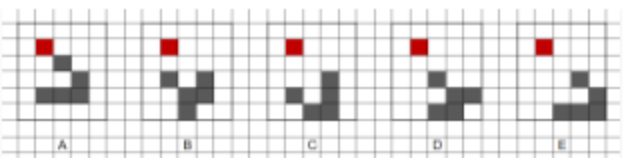
Űrhajó

Az űrhajó olyan alakzat, amely önmagába tér vissza valahány lépés után, de eközben "utazik", vagyis bármely irányba elmozdul.

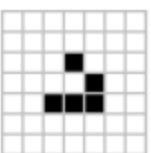
A legkisebb űrhajó a `glider`, vagyis sikló.

```
glider :: Generation
glider = [(6,5), (7,3), (7,5), (8,4), (8,5)]
```

Magyarázat: A könnyebb olvashatóság miatt egy piros négyzetet és fekete keretet helyeztem a sikló köré. A piros négyzet csak egy tetszőlegesen kiválasztott fehér négyzet.



Glider animálva:

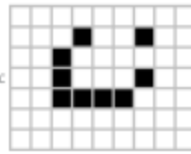


Még néhány hajó a a tesztekhez:

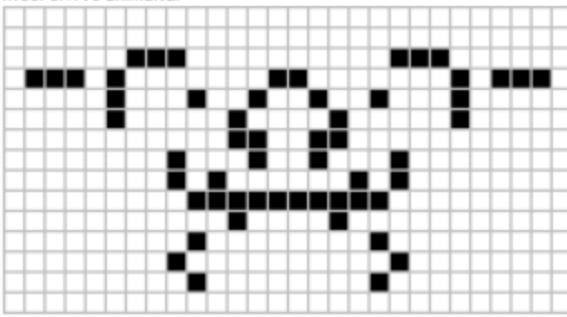
```
lwss :: Generation
lwss = [(8,1),(8,2),(8,3),(8,7),(8,8),(8,9),(1,0),(1,3),(1,7),(1,10),(2,

five6P6H1V0 :: Generation
five6P6H1V0 = [(1,6),(1,7),(1,8),(1,19),(1,20),(1,21),(2,1),(2,2),(2,3),

one19P4H1V0 :: Generation
one19P4H1V0 = [(1,34),(2,17),(2,33),(2,35),(3,7),(3,9),(3,16),(3,22),(3,

corderShip :: Generation
corderShip = [(1,33),(1,34),(1,36),(2,32),(2,33),(2,34),(2,36),(2,43),(2
```

Megjegyzés, a teszteseteken szereplő generációk meglehet, hogy eltérő irányba haladnak az alábbi animációkhoz képest!

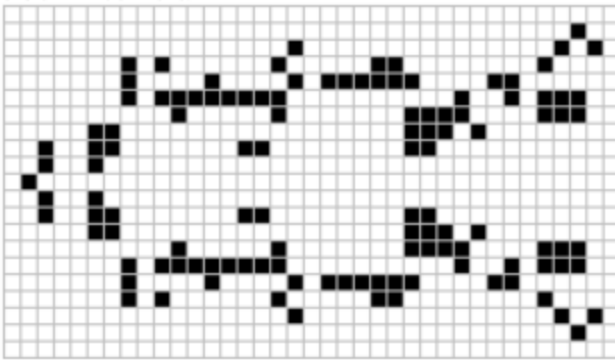
lwss, vagyis a Lightweight spaceship animálva:



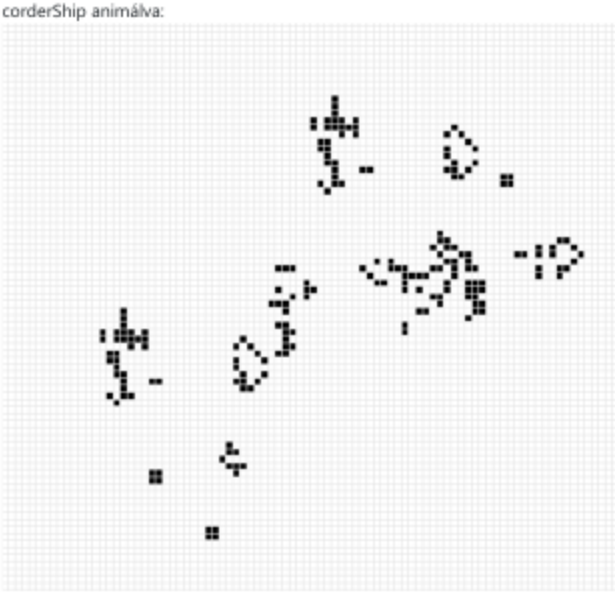
five6P6H1V0 animálva:



one19P4H1V0 animálva:



corderShip animálva:



Alapfeladat

Szomszédos cellák

Készítsünk egy függvényt, amely egy cella koordinátái alapján megadja a vele szomszédos cellák koordinátáit! Mivel a tér végtelen, minden cellának pontosan 8 szomszédja van. A szomszédokat olyan sorrendben adjuk meg, hogy sorokban fentről lefelé, ezen belül pedig balról jobbra legyenek felsorolva.

```
neighbors :: Coordinate -> [Coordinate]
```

Az alábbi tesztesetek közül mindegyiknek `True` -t kell adnia:

```
neighbors (1, 1) == [(0,0),(0,1),(0,2),(1,0),(1,2),(2,0),(2,1),(2,2)]
neighbors (0, 0) == [(-1,-1),(-1,0),(-1,1),(0,-1),(0,1),(1,-1),(1,0),(1,
neighbors (100, -200) == [(99,-201),(99,-200),(99,-199),(100,-201),(100,
```

Szomszédos élő sejtek

Határozzuk meg egy generáció és egy cella alapján, hogy egy adott cellának hány élő sejtet tartalmazó szomszédja van!

```
livingNeighbors :: Generation -> Coordinate -> Int
```

Az alábbi tesztesetek közül mindegyiknek `True` -t kell adnia:

```
livingNeighbors single (0, 0) == 0
livingNeighbors single (1, 1) == 0
livingNeighbors row (10, 0) == 1
livingNeighbors row (10, 1) == 1
livingNeighbors row (10, 2) == 2
livingNeighbors row (9, 2) == 3
```

Szabályok kiszámítása

A fent megadott szabályok alapján, az aktuális generációból kiindulva határozzuk meg, hogy egy cella élő sejtet fog-e tartalmazni a következő generációban/iterációban! Természetesen ehhez azt is figyelembe kell venni, hogy az adott generációban van-e élő sejt!

```
staysAlive :: Generation -> Coordinate -> Bool
```

Az alábbi tesztesetek közül mindegyiknek `True` -t kell adnia:

```
not $ staysAlive single (1, 1)
not $ staysAlive row (10, 1)
staysAlive row (10, 2)
staysAlive row (9, 2)
not $ staysAlive column (9, 2)
staysAlive column (10, 2)
staysAlive column (10, 1)
```

Szabályok alkalmazása élő sejtekre

Alkalmazzuk a fenti szabályokat az aktuális generációra! Mivel egy generáció csak az élő sejtek celláinak koordinátáit tárolja, ezért két dolog történhet minden cellával: a benne lévő sejt vagy életben marad, vagy meghal.

```
stepLivingCells :: Generation -> Generation
```

Az alábbi tesztesetek közül mindegyiknek `True` -t kell adnia:

```
sort (stepLivingCells row) == [(10,2)]
sort (stepLivingCells column) == [(10,2)]
sort (stepLivingCells glider) == [(7,5),(8,4),(8,5)]
sort (stepLivingCells caterer) == sort [(3,6),(3,7),(3,8),(4,2),(4,6),(7
```

Élő sejtekkel szomszédos üres cellák

Adjuk meg azon üres cellák listáját, amelynek az adott generációban van élő sejtes szomszédja! Ügyeljünk rá, hogy minden cella csak egyszer szerepeljen!

```
deadNeighbors :: Generation -> [Coordinate]
```

Az alábbi tesztesetek közül mindegyiknek `True`-t kell adnia:

```
sort (deadNeighbors single) == sort [(41,41),(41,42),(41,43),(42,41),(42
sort (deadNeighbors row) == sort [(9,0),(9,1),(9,2),(10,0),(11,0),(11,1)
sort (deadNeighbors column) == sort [(8,1),(8,2),(8,3),(9,1),(9,3),(10,1
sort (deadNeighbors glider) == sort [(5,4),(5,5),(5,6),(6,4),(6,6),(7,4)
sort (deadNeighbors caterer) == sort [(1,3),(1,4),(1,5),(2,3),(2,5),(3,3
```

Szabály alkalmazása üres cellákra

Alkalmazzuk a fenti szabályokat az aktuális generáció élő sejtjei körüli üres cellákra! Természetesen ezeken a helyeken csak újabb sejtek születhetnek.

```
stepDeadCells :: Generation -> Generation
```

Az alábbi tesztesetek közül mindegyiknek `True`-t kell adnia:

```
stepDeadCells single == []
sort (stepDeadCells row) == [(9,2),(11,2)]
sort (stepDeadCells column) == [(10,1),(10,3)]
sort (stepDeadCells glider) == [(6,4),(7,6)]
sort (stepDeadCells caterer) ==  sort [(3,3),(3,5),(4,1),(4,3),(2,7),(2,
```

Új generáció kiszámítása

Számítsuk ki a következő generációt az aktuális generációból! Ehhez a következő lépéseket kell elvégezni:

- számítsuk ki, hogy az élő sejtek közül melyek maradnak életben,
- határozzuk meg, hogy az élő sejtek körül mely cellákban születik új sejt,
- vegyük az előző két pont eredményének unióját, majd rendezzük.

```
stepCells :: Generation -> Generation
```

Az alábbi tesztesetek közül mindegyiknek `True`-t kell adnia:

```
sort (stepCells single) == []
sort (stepCells row) == sort [(9,2),(10,2),(11,2)]
sort (stepCells column) == sort [(10,1),(10,2),(10,3)]
sort (stepCells glider) == sort [(6,4),(7,5),(7,6),(8,4),(8,5)]
sort (stepCells caterer) == sort [(2,7),(2,8),(3,3),(3,5),(3,6),(3,7),(3
sort (stepCells row) == column
stepCells column == row
sort ((stepCells . stepCells . stepCells) caterer) == caterer
sort (stepCells o15) == [(1,0),(1,1),(1,2),(2,0),(2,2),(3,0),(3,1),(3,2)
sort (stepCells o312) == [(1,21),(1,22),(2,21),(2,22),(6,32),(6,33),(7,2
sort (stepCells lwss) == [(-1,2),(-1,8),(0,1),(0,2),(0,3),(0,7),(0,8),(0
sort (stepCells Five6P6H1V0) == [(0,7),(0,20),(1,2),(1,6),(1,7),(1,20),(1
sort (stepCells one10P4H1V0) == [(1,34),(2,33),(2,34),(3,8),(3,16),(3,17
sort (stepCells corderShip) == [(1,32),(1,34),(2,32),(2,34),(2,36),(2,44
```

Játék léptetése a megadott számú generációba

Készítsük el azt a függvényt, amely egy kezdeti generációra elvégzi az adott számú lépést! A kezdeti generációt vegyük a nulladiknak. Ha a megadott generáció száma negatív, akkor térjünk vissza `Nothing`-al, különben `Just`-ba csomagolva adjuk meg a megfelelő generációt!

A tesztesetek futtatásához szükséges a `Data.Maybe` module importálása!

```
play :: Generation -> Int -> Maybe Generation
```

Az alábbi tesztesetek közül mindegyiknek `True`-t kell adnia:

```
fmap sort (play single 0) == Just [(42,42)]
fmap sort (play single 1) == Just []
fmap sort (play single 2) == Just []
fmap sort (play single (-10)) == Nothing
fmap sort (play glider 4) == Just [(7,6),(8,4),(8,6),(9,5),(9,6)]
fmap sort (play glider 55) == Just [(20,17),(20,19),(21,18),(21,19),(22,
column == (fromJust $ play row 1)
row == sort (fromJust $ play row 2)
row == sort (fromJust $ play column 1)
column == sort (fromJust $ play column 2048)
caterer == sort (fromJust $ play caterer 9)
```

Stabil generációk detektálása

Állapítsuk meg egy generációról, hogy stabil-e. Egy generáció akkor stabil, ha a léptetése után változatlan marad!

```
isStill :: Generation -> Bool
```

Az alábbi tesztesetek közül mindegyiknek `True` -t kell adnia:

```
not $ isStill single
isStill block
not $ isStill row
not $ isStill glider
not $ isStill caterer
```

Extra feladatok

A következő feladatok megoldása nem kötelező, viszont az itt szerzett pontok hozzáadódnak a vizsga pontszámához (az átmenő pontszám megszerzését követően). Ebben a részben összesen (1+2) azaz 3 pont szerezhető.

Oszcillátorok detektálása (1 pont)

Állapítsuk meg egy generációról, hogy oszcillátor-e, azaz visszatér-e önmagába egy megadott lépésnyi távolságon belül. Ha igen, adjuk meg azt a legkisebb pozitív számot, ahány generáció múlva ez megtörténik `Just` -ba csomagolva. Ha nem, adjunk vissza `Nothing` -ot.

```
isOscillator :: Generation -> Int -> Maybe Int
```

Az alábbi tesztesetek közül mindegyiknek `True` -t kell adnia:

```
isOscillator row 2 == Just 2
isOscillator column 10 == Just 2
isOscillator caterer 3 == Just 3
isOscillator caterer 100 == Just 3
isOscillator [] 1 == Just 1
isOscillator single 10 == Nothing
isOscillator row 1 == Nothing
isOscillator caterer 2 == Nothing
isOscillator glider 2 == Nothing
isOscillator o15 2 == Nothing
isOscillator o15 8 == Nothing
isOscillator o15 14 == Nothing
isOscillator o15 15 == Just 15
isOscillator o15 66 == Just 15
isOscillator o15 5000 == Just 15
isOscillator o312 311 == Nothing
isOscillator o312 312 == Just 312
```

Űrhajók detektálása (2 pont)

Állapítsuk meg egy generációról, hogy űrhajó-e. Egy generáció akkor űrhajó, ha felveszi újra a kiindulási formáját egy megadott lépésen belül és nem önmagába tér vissza. Ha űrhajó, adjuk meg irányvektorok formájában azt a

legkisebb távolságot, amely az azonos formák eltolódásából keletkezett. Amennyiben nem veszi fel a kiindulási formát egyszer sem vagy önmagába tér vissza, adjunk vissza `Nothing` -ot.

```
isSpaceShip :: Generation -> Int -> Maybe (Integer, Integer)
```

Az alábbi tesztesetek közül mindegyiknek `True` -t kell adnia:

```
isSpaceShip glider 0 == Just (1,1)
isSpaceShip row 2 == Nothing
isSpaceShip column 10 == Nothing
isSpaceShip caterer 3 == Nothing
isSpaceShip caterer 100 == Nothing
isSpaceShip [] 1 == Nothing
isSpaceShip single 10 == Nothing
isSpaceShip row 1 == Nothing
isSpaceShip caterer 2 == Nothing
isSpaceShip glider 2 == Nothing
isSpaceShip oil5 2 == Nothing
isSpaceShip oil5 8 == Nothing
isSpaceShip oil5 14 == Nothing
isSpaceShip oil5 15 == Nothing
isSpaceShip oil5 66 == Nothing
isSpaceShip oil5 5000 == Nothing
isSpaceShip lwss 4 == Just (-2,0)
isSpaceShip lwss 6 == Just (-2,0)
isSpaceShip lwss 500 == Just (-2,0)
isSpaceShip lwss 0 == Nothing
isSpaceShip lwss 1 == Nothing
isSpaceShip lwss 2 == Nothing
isSpaceShip lwss 3 == Nothing
isSpaceShip fiveEP6HDV0  7 == Just (-1,0)
isSpaceShip oneI9P4HDV0 4 == Just (0,-1)
isSpaceShip oneI9P4HDV0 3 == Nothing
isSpaceShip corderShip 96 == Just (-8,-8)
```