

## Plants vs. Zombies

Kategória:	Beadandók
Elérhető:	2022. 11. 27. 23:59
Pótolható határidő:	
Végső határidő:	2022. 12. 16. 23:59
Kiírta:	Erdei Zsófia

Leírás:

### Plants vs. Zombies

#### 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

A feladat során egy leegyszerűsített verzióját fogjuk a *Plants vs Zombies* játéknak implementálni. A játékban egy 5 soros pályán zombik masíroznak jobbról balra, míg a játékos növények lerakásával próbálja megvédeni magát. A zombik nyerne, ha egy zombi elér a pálya bal oldalára. A játékos nyer, ha az összes zombi meghal. A játék során Napokat kell gyűjteni, amellyel új növényeket lehet venni. Az eredeti videójátékkal kapcsolatban egyéb információ a [Wikipédián](#) olvasható.

Mivel az implementációban nem lehet grafikai felületünk, ezért csak egy szimulációt fogunk elvégezni, vagyis önmagától fognak zajlani a körök. A játék menetét diszkrét időintervallumokra bontjuk (úgy nevezett körökre), amelyek eltelése után minden játékban szereplő zombi és növény elvégezhet egy automatikus műveletet. *Például:* A zombi megy előre, míg a növény lő.

#### Típusok definíciója

A játékmmodell fogja tartalmazni a növényeket, zombikat és a játékos Napjainak számát. Először definiáljuk a részadatszerkezeteket, majd egy nagy közös tároló modellt. A koordináták és a Napok reprezentáláshoz az alábbi típuszinonímákat fogjuk használni:

```
type Coordinate = (Int, Int)
type Sun = Int
```

#### Növények

Növényekből szükségünk lesz az alap növényekre: **Peashooter** , **Sunflower** , **Walnut** és **CherryBomb** . Definiáljunk egy **Plant** adattípust az előbb említett konstruktorokkal. Minden konstruktornak legyen egy **Int** típusú paramétere, amely a maradék életpontjukat reprezentálja a példányoknak. A növények funkcionalitását később implementáljuk.

#### Zombik

Definiáljuk a **Zombie** adattípust az alábbi konstruktorokkal: **Basic** , **Conehead** , **Buckethead** és **Vaulting** . Minden konstruktornak legyen két **Int** típusú paramétere, amelyből az első a maradék életpontját, míg a második a mozgási sebességét reprezentálja.

#### Modell

Definiáljuk a **GameModel** adattípust, amelynek egy **GameModel** nevű konstruktora van. A konstruktor tárolja, hogy mennyi Napja van a játékosnak egy **Sun** típusú

paraméterben, illetve a növények és a zombik helyét és pozícióit egy `[(Coordinate, Plant)]` és `[(Coordinate, Zombie)]` típusú paraméterekben.

A feladatban lévő típusokra kérjük meg a fordítót, hogy implementálja `Eq` és `Show` típusosztályokat.

### Példák

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

```
defaultPeashooter :: Plant
defaultPeashooter = Peashooter 3

defaultSunflower :: Plant
defaultSunflower = Sunflower 2

defaultWalnut :: Plant
defaultWalnut = Walnut 15

defaultCherryBomb :: Plant
defaultCherryBomb = CherryBomb 2

basic :: Zombie
basic = Basic 5 1

coneHead :: Zombie
coneHead = Conehead 10 1

bucketHead :: Zombie
bucketHead = Buckethead 20 1

vaulting :: Zombie
vaulting = Vaulting 7 2
```

## Alapfeladat

### Növények vásárlása

A játékban minden növénynek van egy előre megadott ára:

- `Peashooter` -nek `100 Nap` ,
- `Sunflower` -nek és `Walnut` -nak `50 Nap`
- `CherryBomb` -nak `150 Nap` .

Definiáljuk a `tryPurchase` függvényt, amely azt szimulálja, hogy a játékos egy adott koordinátára egy növényt próbál megvenni. Ha azon a helyen van már növény, vagy az `5 * 12` -es játéktéren kívülre akarna vásárolni, esetleg nincs elég Napja a játékosnak, akkor adjunk vissza `Nothing` -ot!

**Segítség:** A `Data.List.lookup` függvény hasznos lehet a feladat során.

```
tryPurchase :: GameModel -> Coordinate -> Plant -> Maybe GameModel
```

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

```
tryPurchase (GameModel 50 [] []) (0,0) defaultPeashooter == Nothing
tryPurchase (GameModel 50 [] []) (0,0) defaultWalnut == Just (GameModel 0 [(0,0), defaultWalnut])
tryPurchase (GameModel 150 [((0,0), defaultWalnut)] []) (0,0) defaultCherryBomb == Just (GameModel 0 [(0,0), defaultCherryBomb])
tryPurchase (GameModel 50 [] []) (5,0) defaultWalnut == Nothing
```

### Zombik lerakása

A zombik az eredeti játékban mindig valamelyik sor végén jelennek meg - néhány irreleváns kivétellel. Ezt a sémát próbáljuk meg a szimulációban is követni. Definiáljuk a `placeZombieInLane` nevű függvényt, amely egy zombit lehelyez valamelyik sáv végére. Ha az adott sáv végén már van zombi vagy a sáv száma nem megfelelő, akkor adjunk vissza `Nothing` ot. A játéktérben 5 sáv van és azok 12 hosszúak. Az indexelést 0-tól kezdjük.

```
placeZombieInLane :: GameModel -> Zombie -> Int -> Maybe GameModel
```

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

```
placeZombieInLane (GameModel 0 [] []) coneHead 0 == Just (GameModel 0 [] [(0,1)])
placeZombieInLane (GameModel 0 [] []) vaulting 5 == Nothing
placeZombieInLane (GameModel 0 [] [(0,11), coneHead]) bucketHead 0 == Nothing
placeZombieInLane (GameModel 0 [] [(0,3), coneHead]) basic 0 == Just (GameModel 0 [] [(0,3)])
```

## Zombik mozgása és támadása

A zombik minden kör alatt a sebességüknek megfelelő mezőt mennek előre, amennyiben tudnak. Ha egy zombi nem tud előre menni, mert a mezőn, amin áll, van egy növény, akkor a zombi beleharap a növénybe és csökkenti az életpontját *1* -gyel és továbbra is azon a mezőn marad. Ez alól csak a *Vaulting* zombi a kivétel: ha még a sebessége *2* , akkor az első növényt átugorja és halad tovább, viszont a sebessége *1* -re csökken.

Definiáljuk a *performZombieActions* függvényt amely a modell összes zombijára elvégzi a fent említett megfelelő műveletet. Ha egy zombi eljutna a játéktér végére, adjunk vissza *Nothing* -ot! A függvénynek nem kell kitörölnie a halott növényeket!

**Segtiség:** Segíthet egy olyan segédfüggvény implementálása, amely egy adott koordinátájú növény életpontját csökkenti 1-gyel.

```
performZombieActions :: GameModel -> Maybe GameModel
```

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

```
performZombieActions (GameModel 0 [] [(0,0), coneHead]) == Nothing
performZombieActions (GameModel 0 [] [(0,1), coneHead]) == Just (GameModel 0 [] [(0,1)])
performZombieActions (GameModel 0 [(0,1), defaultWalnut]) [(0,1), coneHead] == Just (GameModel 0 [(0,1), defaultWalnut])
performZombieActions (GameModel 0 [(0,1), defaultWalnut]) [(0,1), vaulting] == Just (GameModel 0 [(0,1), defaultWalnut])
```

## Pályatisztítás

Amikor egy növény lelő egy zombit vagy egy zombi megeszik egy növényt, és az életpontja *0* -ra vagy az alá esik, akkor ezeket a halott lényeket el kell tüntetni a pályáról. Definiáljuk a *cleanBoard* függvényt, amely letöröl mindent a pályáról, aminek legfeljebb *0* életpontja van.

```
cleanBoard :: GameModel -> GameModel
```

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

```
cleanBoard (GameModel 0 [(0,0), Peashooter 0]) [] == GameModel 0 [] []
cleanBoard (GameModel 0 [(0,0), defaultPeashooter]) [(0,0), Basic 0 1]) == Just (GameModel 0 [(0,0), defaultPeashooter])
```

## Extra feladatok

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

## Növények műveletei (1 pont)

A növények minden körben valami műveletet végeznek el:

- a *Peashooter* meglövi a sorban lévő első előtte lévő zombit,
- a *Sunflower* produkál 25 Napot,
- a *CherryBomb* felrobban, megölve saját magát és az összes zombit egy *3 \* 3* -as területen, ahol a növény a középpontja (itt a megölés az életpontok 0-ra való állítását jelenti).

Definiáljuk a `performPlantActions` függvényt, amely a növények fent említett műveleteit elvégzi minden növényre. Mivel a sebességük nem egyezik a zombiknak, ezért lehet, hogy ugyanazon a mezőn fognak állni. Ebben az esetben az összes releváns zombit támadjuk meg! A függvény ne törölje ki a halott zombikat!

**Segítség:** Segíthet egy olyan segédfüggvény implementálása, amely egy adott koordinátájú zombi életponját megváltoztatja.

```
performPlantActions :: GameModel -> GameModel
```

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

```
performPlantActions (GameModel 0 (replicate 5 ((0,0), defaultSunflower))) [] ==> True
performPlantActions (GameModel 0 [((0,0), defaultPeashooter)] [((0,3), conehead)] ==> True
performPlantActions (GameModel 0 [((3,3), defaultCherryBomb )] [((2,2), buchu)]) ==> True
```

## Szimuláció (2 pont)

### Játék Menete

Egy játék menete az alábbi módon néz ki:

1. **Csak a következő feladatban!** Felhasználói beavatkozás
2. Növények támadnak
3. Pálya tisztítás
4. Zombik támadnak, ha bejutottak, nyertek
5. Új zombik lerakása
6. Pálya tisztítás
7. +25 Nap

Ezt a 7 műveletet ismétli a játék, amíg valaki nem nyer. A zombik az eredeti játékban hullámonként jönnek, tehát a játék valamennyi zombit lerak a fenti kör után. Definiáljuk a `defendsAgainst` függvényt amely egy statikus növényvédelem ellen leszimulál egy zombi hordát! A második paraméter a körönként beszűrendő zombikat tartalmazza! Ha a növények nyernek, akkor `True`-t adjunk vissza egyébként `False`-ot! Ha egy zombit nem tudunk beszűrni, a program dobjon kivételt!

**Segtíség:** Az előző feladatokban implementált `performZombieAction` , `performPlantAction` és `cleanBoard` függvények használata jóval leegyszerűsítheti a programot!

```
defendsAgainst :: GameModel -> [(Int, Zombie)] -> Bool
```

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

```
defendsAgainst (GameModel 0 (map (\x -> ((x,0), Peashooter 2)) [0..4])) [] ==> True
not $ defendsAgainst (GameModel 0 (map (\x -> ((x,0), Peashooter 2)) [0..4])) [(0,0)] ==> True
defendsAgainst (GameModel 0 (map (\x -> ((x,0), Peashooter 2)) [0..4])) [] ==> True
```

### Interaktív szimuláció

A játéknak nagyon fontos része az emberi interakció. A játékciklus fentebb specifikálja, hogy mikor és hogyan szólhat bele a játékos (ebből csak a mikort fogjuk implementálni). Definiáljuk a `defendsAgainstI` függvényt, amely engedi a játékosnak a modell megváltoztatását a körök elején! Ez annyit jelent, hogy az interakciót a játékostól, mint függvényparaméter vesszük át és ezt a függvényt alkalmazzuk a "Felhasználói beavatkozás" fázisban. (Ez nem biztosítja, hogy a játékos mondjuk nem törli ki simán az összes zombit, de reméljük senki sem csinál.)

```
defendsAgainstI :: (GameModel -> GameModel) -> GameModel -> [(Int, Zombie)] -> Bool
```

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

```
defendsAgainstI id (GameModel 0 (map (\x -> ((x,0), Peashooter 2)) [0..4]) |
not $ defendsAgainstI id (GameModel 0 (map (\x -> ((x,0), Peashooter 2)) [0
defendsAgainstI id (GameModel 0 (map (\x -> ((x,0), Peashooter 2)) [0..4]) |
defendsAgainstI (\(GameModel s p _) -> GameModel s p []) (GameModel 0 (map
```

## Megoldás

Letöltés

**Név:** Nagybeadandó.zip  
**Feltöltés ideje:** 2022. 12. 16. 18:47  
**Értékelés:**  
**Státusz:** Sikertelen tesztelés  
**Feltöltések száma:** 3  
**Értékelte:**  
**Megjegyzések:**

### Automatikus tesztelés eredményei

Elért pontszám: 3/3 pont.

Teszteken sikeresen átmenő definíciók: defaultPeashooter, defaultSunflower, defaultWalnut, defaultCherryBomb, basic, coneHead, bucketHead, vaulting, tryPurchase, placeZombieInLane, performZombieActions, cleanBoard, performPlantActions, defendsAgainst, defendsAgainstI.

Minden teszt sikeresen lefutott.

## Mellékelt fájlok

tms\_ready\_tests.txt

Letöltés