1. feladat

Az első feladat egy egyszerű absztrakt adatstruktúra létrehozása és ennek a műveleteinek bemutatása egy tesztprogramon keresztül. Hozzon létre egy `Bag_Package` sabloncsomagot, két **diszkrét** sablonparaméterrel (`Index` és `Counter` néven). A csomag tartalmazzon egy átlátszatlan típust, ez legyen a `Bag`. A belső reprezentáció egy `Index` indexelésű, `Counter` elemeket tartalmazó tömb. Valósítsa meg a csomag az alábbi műveleteket:

- `Empty(Bag)`, mely egy paraméterül kapott `Bag`-et inicializál, az `Counter` legelső értékével.
- `Add(Bag, Index)`, egy `Bag` és egy `Index` típusú paraméterrel, mely növeli az `Index` pozíción található `Counter` értékét.
- `Remove(Bag, Index)`, mely csökkenti az adott `Index` helyén a `Counter` értékét.
- `Has_Any(Bag, Index) -> Boolean`, mely megadja, hogy az adott `Index`-en a `Counter` legelső értékénél nagyobb elem van-e.

A fentit tesztelendő, adjunk meg egy egyszerű példaprogramot, mely segítségével kockadobást szimulálunk. Példányosítsunk a fenti sablonból egy `Dice_Bag` csomagot, mely `Counter`-ként természetes számot, `Index`-ként egy 1-től 6-ig terjedő típust adjunk meg. Ciklus és egy `Dice_Bag.Bag` használatával adjuk meg 20 random generált dobás eloszlását (a dobás értékével indexeljünk). Ezután ciklusok segítségével addig vegyük ki az elemeket a `Bag`-ből, amíg találunk egy olyan `Index`-et, ahol már nem tudunk elemet eltávolítani (`Has_Any` hamisat jelez). Ha jól oldottuk meg a feladatot, megkapjuk a legkevesebbszer dobott számo(ka)t.

2. feladat

A feladat egy játékszimuláció elkészítése, mely három fő részből fog állni. Ezek a pálya, a harcos, illetve maga a szimuláció. Harcosunk királyi feladata, hogy megtisztítsa a birodalmat minden akadálytól, illetve betakarítsa a terményt. Ha túléli a küldetést, elnyeri a királylány kezét, és boldogan élnek, amíg meg nem halnak.

Pálya

A birodalmat jelképezze egy `NxN`-es mező, melyet valóstíson meg sabloncsomag segítségével, `Map_Package` néven. Három sablonparamétert vár, ezek

- `Field`: diszkrét típus, a mező fajtáját fogja megadni;
- `Warrior`: tetszőleges definit típus, a harcosunk;
- `Field_Action(Field, Warrior)`: eljárás, mely megadja, hogy az adott mezőn mit kell tennie a harcosnak. Az akció után a mező legyen üres;
- `Empty`: bemeneti `Field` típusú változó.

A csomag tartalmazzon egy átlátszatlan, diszkriminánsos `Map` típust (a diszkrimináns lesz a `Map` mérete), mely belső reprezentációja legyen rekord.

A rekord attribútumai

- `Fields` mátrix, pozitív indexen értelmezve, tartalmazza a `Field` mezőket;
- `Position` a harcos aktuális koordinátája.

A megvalósítandó műveletek:

- `Init(Map)`: inicializálja a pályát, random `Field` értékekkel;
- `Put(Map)`: kirajzolja a pályát;
- `Move(Map, Warrior)`: random irányba mozgatja a harcost (túlfutás esetén a pálya túlsó oldalán terem), szimulálja az adott mezőn történő akciót a `Field_Action` meghívásával;
- Is_Empty(Map): megadja, ha üres-e a pálya, tehát a harcos minden mezőt bejárt.

Harcos

A harcost valósítsa meg a `Warrior_Package` sabloncsomag segítségével. Sablonparaméterek a következők:

- `Resource`: diszkrét típus, a betakarítandó termék;
- `Obstacle`: diszkrét típus, ellenséges mező;
- `Tool(Obstacle) -> Resource`: függvény, mely megadja, hogy egy ellenséges mezőn milyen eszközt használhatunk, hogy túljussunk rajta;
- `Bag`: az 1. feladatban definiált `Bag` típust fogjuk itt használni, egyelőre tetszőleges paraméter;
- továbbá a `Bag_Package` csomag műveletei.

A csomag tartalmaz egy átlátszatlan `Warrior` típust, mely belső reprezentációja egy rekord. A rekord attribútumai a következők:

- `Status`: jelzi, hogy él-e a harcos, alapértelmezetten él;
- `Inventory`: egy `Bag` típus, mely a harcos eszközeit, valamint ezek mennyiségét fogja tartalmazni.

A megvalósítandó műveletek:

- `Init(Warrior)`: inicializálja a harcos táskáját, minden `Resource`-ból kapjon egyet.
- `Fight(Warrior, Obstacle)`: ha a harcosnak van elég `Resource` a táskájában, mely segítségével le tudja győzni az adott `Obstacle`-t, akkor legyőzi azt és elveszít egyet a használt eszközből. Ha nincs nála ilyen eszköz, meghalt és státusza eszerint emgváltozik;
- `Collect(Warrior, Resource)`: a talált `Resource` hozzáadódik az `Inventory`-hoz;
- `Is_Alive(Warrior) -> Boolean`: függvény, mely megadja, hogy él-e a harcos.

Szimuláció

A szimuláció során a fenti sabloncsomagok példányosításán túl az alábbiakra lesz szükség: * `Field` felsorolásos típus, mely megadja a mező lehetséges értékeit. Ez most legyen `Empty`,

`Stone`, `Wood`, `Sword`, `Water`, `Trap`, és `Enemy`. Ezek közül betakarítandó termék (`Resource`): `Stone`, `Wood`, `Sword`; ellenséges mező (`Obstacle`): `Water`, `Trap`, és `Enemy`.

- Ha a harcos `Water`-re lép, akkor `Stone` segítségével gázlót építve juthat át.
- Ha a harcos `Trap`-re téved, csak `Wood`-ból készült deszka mentheti meg.
- Ha a harcos `Enemy`-vel találkozik, harcra kerül a sor és kardot ránt. A harc után a `Sword` kicsorbul és a harcos eldobja.
- Itt kell megadni a `Field_Action(Field, Warrior)` eljárást, melyen `Resource` esetén a harcos betakarítja a terményt (`Collect`), `Obstacle` esetén harcot kezdeményez (`Fight`), `Empty` esetén nem csinál semmit. Ha a harcos túléli a mezőt, legyen annak az értéke `Empty`.
- A program törzsében inicializáljuk a pályát és a harcost, majd addig lépegessen a harcos (`Move`), amíg él, vagy nem végezte el küldetését.
- Az eredményt, valamint a főbb lépéseket (lépés, harc, gyűjtögetés) mindig írja ki a képernyőre.