Alkalmazói Rendszerek 4 Unity beadandó

Cheese Hunt – Sajtvadászat

Az Alkalmazói Rendszerek 4 tárgy Unity beadandó feladatában egy felülnézetes játékot kell megvalósítani, melyben a játékos egy egeret irányít, aki sajtokat gyűjt össze a labirintus szerű pályákon. Az egér bármerre mehet (felfelé-lefelé is), nincs ugrás, nincs gravitáció. A projekt elkészítéséhez szükséges segédfájlok, képek megtalálhatók Canvas-en.

A játékban a következő elemekre lehet szükség: Pálya elemek (tilemap-ek), játékos karakter, felszedhető jutalom, csapdák, két típusú ellenfél, ajtó és ajtót nyitó kulcs, csapda lövedéke.
A leírásban és képeken ezek az elemek egeres játék témájában kerülnek bemutatásra (tehát a játékos az egér, felszedhető jutalom a sajt stb.), de ezek szabadon módosíthatók bármilyen más design elemekre. Tetszőlegesen használhatók a megadott képek helyett sajátok vagy egyéb interneten fellelhető képek és egeres játék helyett szabadon lehet űrhajós, katonás, kiskutyás vagy akármilyen más köntösben tálalni a játékot. Ameddig, a funkcionalitás megegyezik és kielégíti a leírást, addig a design szabadon választható.

Egy egyszerű ingyenes sprite lelőhely: Kenney
A Unity-nek van saját áruháza, ahol lehet ingyenes 2D sprite-okat vadászni: Unity Asset Store
Továbbá saját készítésű képek is felhasználhatók.

A beadandó játékprojekt részfeladatokat tartalmaz, minden részfeladatra pontok járnak. Az elért pontokból az alábbi táblázatban leírt módon határozható meg az elért Unity játékfejlesztés jegy. Vannak feladatok, ahol szükséges programozni, vannak, ahol nem (mert csak Unity konfiguráció vagy a kód adott hozzá vagy a gyakorlatokon látott kód felhasználható). Egyes későbbi, nehezebb feladatok a gyakorlatokon nem látott funkciók használatát kérik. Ezek használata és működése a feladat szövegében le van írva és a Unity hivatalos dokumentációjában is bővebben megtekinthető a működésük. A feladatokat nem sorrendben kell elkészíteni (pl. helyesen elkészített és felhasznált prefabre/sablonra jár a pont, ha az objektum működéséhez szükséges script nem készült el).

Pontszám	Érdemjegy
25	2 (Minimum)
31	3
38	4
45	5
60	Maximum

Projekt létrehozása

Készítsünk egy új Unity 2D projektet (lehet sima 2D vagy 2D URP is).

Tilemap

Tilemap importálása (2 pont)

Importáljuk az egyiket a tilemap-ek közül. (elég csak egyet használni, de nincs tiltva több használata sem)
Importálásnál adjuk meg, hogy egy tilemap-en belül több tile/csempe is van (Sprite Mode Multiple), majd szeleteljük fel (Slice) a képet.
Állítsuk be, hogy egy tile/csempe a Unity játékterében egy egység területet fedjen le (Pixels Per Unit).

A különböző dinamikus objektumokhoz, mint ellenfelek, csapdák, lövedékek az objects_tilemap képet fogom használni, ebből fogok 1-1 sprite-ot felhasználni. Ehhez szintén Sprite Mode Multiple beállítás, majd szeletelés és Pixels Per Unit állítás szükséges. Tetszőlegesen használható ez a tilemap vagy saját/egyéb sprite-ok is.

Tilemap hozzáadása a játéktérhez (1 pont)

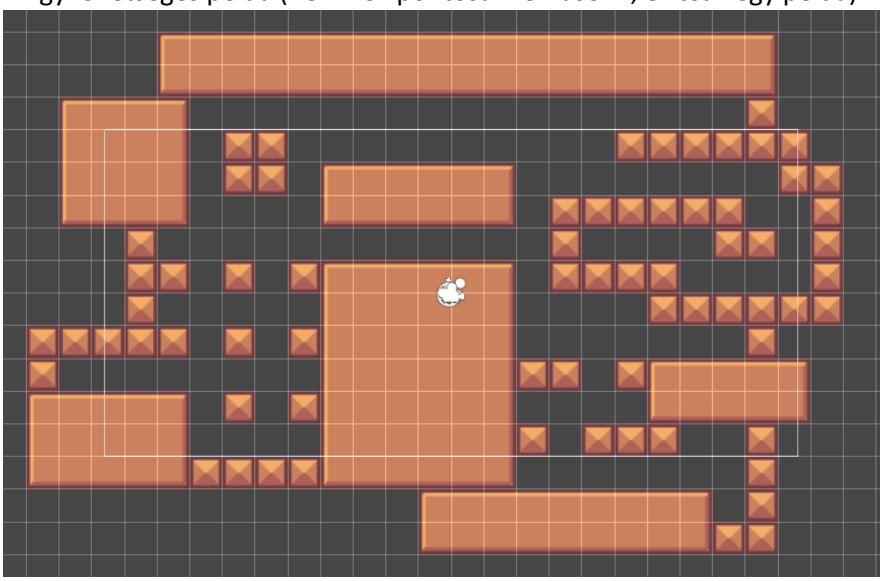
Hozzunk létre egy új Tilemap játékelemet (Új objektum, 2D, Tilemap, Rectangular).
Hozzunk létre egy új Tile Palettát (Window, 2D, Tile Palette) és rendeljük hozzá az importált Tilemapet.

Pálya elkészítése (3 pont)

Az importált Tilemappel rajzoljunk egy tetszőleges zárt pályát. Használjunk legalább 4 féle tilet/csempét és rajzoljunk legalább 60 elemet a játéktérre. A pálya alakja és formája tetszőleges, lehetnek nagyobb terek, lehetnek szűkebb folyosók stb.

(A csapdákat, ellenfeleket, felszedhető jutalmakat és egyéb dinamikus objektumokat ne rajzoljuk a Tilemap-re, mert ezeket külön-külön objektumonként kell elhelyezni, hogy megkülönböztethessük őket a falaktól. Tilemap-re csak a szilárd falakat rajzoljuk, melyek a játék során soha nem változnak/mozdulnak és szerepük mindössze annyi, hogy megállítsanak a mozgásban.)

Egy lehetséges példa (nem kell pontosan lemásolni, ez csak egy példa)



Játékos karakter (pl. egér)

Játékos karakter (pl. egér) importálása (1 pont)

Importáljuk az játékos karakter képet/sprite-ot a segédfájlok közül (mouse.png az egér). Állítsuk be, hogy a kép/sprite a Unity játékterében egy egység területet fedjen le (Pixels Per Unit).

Játékos karakter (pl. egér) objektum (2 pont)

Helyezzük el az játékos karaktert a játéktér egy tetszőleges pontjára.
Importáljuk a segédfájlok közül a PlayerMovementScript kódot és adjuk hozzá az objektumhoz. Így az játékos mozgatható a WASD vagy kurzor billentyűkkel, de ki tud menni a képernyőről és a falakon átmegy.
A kamerát tegyük a játékos objektum egy alárendelt objektumává / gyerek objektumává (a hierarchiában a kamera az játékos karakter része legyen). Ezután a kamera relatív pozícióját állítsuk 0,0,-10-re. Így a kamera követi a játékost.

Falakkal ütközés (3 pont)

A játékos ütközzön a falakkal, ne tudjon átmenni a falakon.
Ehhez a Tilemap objektumon ütköző komponensre van szükség.
A játékos karakter objektumon ütköző komponensre és Rigidbody komponensre van szükség. Az ütköző mérete nagyjából passzoljon az karakter képéhez. A Rigidbody komponensen a Gravity Scale legyen 0 (nincs szükség gravitációra, ebben a játékban a játékos nem tud ugrani, bármerre mehet) és fagyasszuk be az elforgatást, ne tudjon elfordulni a karakter.

Újjáéledés (3 pont)

(ahhoz, hogy ez a feladat teljes legyen, előbb szükséges valamelyik későbbi csapda vagy ellenfél elkészítése)

Ha a játékos karakter ellenféllel vagy csapdával ütközik, akkor a karakter odaveszik és azonnal a kezdőhelyre kerül. A játékos objektum nem semmisül meg, csupán a pozícióját a kezdeti pozícióra állítjuk. Ehhez szükséges egy script a játékos karakteren, mely detektálja, ha a játékos ellenféllel ütközik (kiegészíthető az a script is, mely a jutalmak felvételét kezeli).

Vegyünk fel a script-ben egy változót, melyben a Start függvény eltárolja az objektum kezdeti pozícióját (transform.position), ahová vissza kell teleportálni, ha odaveszett a karakter.

Az OnTriggerEnter2D függvényben detektálhatja a script, ha összeütközik valamivel. Ellenőrizzük, hogy ellenféllel ütköztünk-e (.CompareTag) és ekkor teleportáljuk a játékost a kezdő pozícióra (transform.position-nek értékül adjuk a Start-ban eltárolt pozíciót).
Ezt követően, ha a játékos ellenféllel vagy csapdával ütközik azonnal visszakerül a kezdő helyre (ha a kezdő helyen is van egy ellenfél, akkor a játék „kiakad” és folyamatosan teleportál a játékos, ezzel nem foglalkozunk, ügyes pálya tervezés kell hozzá).

Felszedhető jutalom (pl. sajt)

Felszedhető jutalom (pl. sajt) importálása (1 pont)

Importáljuk a felszedhető jutalom képet/sprite-ot a segédfájlok közül (cheese.png a sajt). Állítsuk be, hogy a kép/sprite a Unity játékterében egy egység területet fedjen le (Pixels Per Unit).

Felszedhető jutalom (pl. sajt) objektum (4 pont)

Helyezzük el egy jutalmat a játéktéren.
Implementáljuk, hogy a játékos karaktere fel tudja venni a jutalmat. Ha a játékos összeütközik a jutalommal, a jutalom eltűnik, az objektum megsemmisül.
Ehhez a jutalom objektumon egy trigger ütközőre van szükség, mérete nagyjából passzoljon a jutalom képéhez.
Adjunk hozzá egy új tetszőleges Tag-et a jutalom objektumhoz, így a játékos karakter ütközésnél meg tudja különböztetni a jutalmat a többi objektumtól.
Készítsünk egy új script-et, melyet a játékos karakterre helyezünk ütközik (kiegészíthető az a script is, mely az ellenfelekkel ütközést kezeli). Ez a script detektálja az ütközést (OnTriggerEnter2D). Ha jutalommal ütköztünk (.CompareTag), akkor semmisítsük meg a jutalom objektumot amivel ütköztünk.

Felszedhető jutalom (pl. sajt) prefab/sablon készítése (2 pont)

Készítsünk a jutalom objektumból egy prefabet/sablont. Helyezzünk el legalább 8 jutalmat a játéktér tetszőleges pontjain. Az elhelyezéshez a prefab/sablont használjuk.

Folyamatosan aktív csapda (pl. tüskék)

Folyamatosan aktív csapda (pl. tüskék) importálása (1 pont)

Importáljuk a csapda képet/sprite-ot a segédfájlok közül (objects_tilemap.png-ben találhatók a tüskék). Állítsuk be, hogy a kép/sprite a Unity játékterében egy egység területet fedjen le (Pixels Per Unit).

Folyamatosan aktív csapda (pl. tüskék) objektum (3 pont)

Helyezzük el egy csapdát a játéktéren.
Implementáljuk, hogy a játékos karaktere ütközéskor odavesszen és a kezdőhelyre teleportáljon.
Ehhez a csapda objektumon egy trigger ütközőre van szükség, mérete nagyjából passzoljon a csapda képéhez.
Adjunk hozzá egy új tetszőleges Tag-et a csapda objektumhoz, így a játékos karakter ütközésnél meg tudja különböztetni a csapdát a többi objektumtól.
Ezután a játékos karakternek detektálnia kell az ütközést, ez a rész az Újjáéledés részfeladatban található.

Folyamatosan aktív csapda (pl. tüskék) prefab/sablon készítése (2 pont)

Készítsünk a csapda objektumból egy prefabet/sablont. Helyezzünk el legalább 8 csapdát a játéktér tetszőleges pontjain. Az elhelyezéshez a prefab/sablont használjuk.

Periodikusan aktív csapda (pl. fáklya)

Periodikusan aktív csapda (pl. fáklya) importálása (1 pont)

Importáljuk a csapda képet/sprite-ot a segédfájlok közül (objects_tilemap.png-ben találhatók a fáklya. Én kettőt használok: egyet aminek nincs lángja és egyet aminek van). Állítsuk be, hogy a kép/sprite a Unity játékterében egy egység területet fedjen le (Pixels Per Unit).

Periodikusan aktív csapda (pl. fáklya) objektum (4 pont)

Helyezzük el egy csapdát a játéktéren.
A csapda periodikusan ki-és be kapcsol, kikapcsolt állapotban a játékos áthaladhat rajta, bekapcsolt állapotban nem, ekkor a játékos a kezdőhelyre kerül. Ehhez egy script lesz szükséges a csapdán, mely periodikusan ki-és be kapcsolja az ütközőjét és megjelenítését a csapdának. (kikapcsolható a teljes sprite megjelenítése, ekkor teljesen eltűnik a csapda. Én két sprite-ot használok hierachiába téve: egy tűz nélküli fáklya, mely mindig látszódik és egy tüzes fáklya, mely ki-be kapcsolgat).
Ehhez a csapda objektumon egy trigger ütközőre van szükség, mérete nagyjából passzoljon a csapda képéhez.
Adjunk hozzá egy tetszőleges Tag-et a csapda objektumhoz, így a játékos karakter ütközésnél meg tudja különböztetni a csapdát a többi objektumtól. Készítsünk egy script-et, melynek változóiban megadható a periódus (hány másodperc alatt kapcsol ki-be), az ütköző (Collider2D) és a megjelenítő (SpriteRenderer) komponens. A script egy float változóban tárolja el az időt, mikor váltania kell. Az Update függvényben ellenőrizzünk, hogy a játékidő (Time.time) már nagyobb-e, mint a váltás ideje. Ha nagyobb, akkor állítsuk be a következő váltás időt (Time.time + periódus) és az ütköző és megjelenítő komponenseket váltsuk ki-és bekapcsolt állapot között (mindkét komponensnek van .enabled paramétere, mely egy bool és ezzel állítható, hogy ki-vagy bekapcsoltak éppen).
Ezután a játékos karakternek detektálnia kell az ütközést, ez a rész az Újjáéledés részfeladatban található.

Periodikusan aktív csapda (pl. fáklya) prefab/sablon készítése (2 pont)

Készítsünk a csapda objektumból egy prefabet/sablont. Helyezzünk el legalább 4 csapdát a játéktér tetszőleges pontjain. Az elhelyezéshez a prefab/sablont használjuk.

Oda-vissza mozgó ellenfél (pl. légy)

Oda-vissza mozgó ellenfél (pl. légy) importálása (1 pont)

Importáljuk az ellenfél képet/sprite-ot a segédfájlok közül (objects_tilemap.png-ben található a légy). Állítsuk be, hogy a kép/sprite a Unity játékterében egy egység területet fedjen le (Pixels Per Unit).

Oda-vissza mozgó ellenfél (pl. légy) objektum (4 pont)

Helyezzük el egy ellenfelet a játéktéren.
Implementáljuk, hogy az ellenfél oda-vissza mozogjon a falak között. Ehhez az ellenfél objektumon egy trigger ütközőre van szükség, mérete nagyjából passzoljon az ellenfél képéhez. Mivel az objektum script-ből fog mozogni, ezért egy kinematikus Rigidbody2D komponensre is szükség lesz. Adjunk hozzá egy tetszőleges Tag-et az ellenfél objektumhoz, így a játékos karakter ütközésnél meg tudja különböztetni az ellenfelet a többi objektumtól.
Ezután a játékos karakternek detektálnia kell az ütközést, ez a rész az Újjáéledés részfeladatban található.
A script tárolja el egy változóban mozgásnak az irányát (pl. Vector2 típus). Az Update függvényben ezt a mozgást adjuk hozzá az objektum pozíciójához (transform.position, ügyeljünk a valós idő használatára is Time.deltaTime). Ha az objektum összeütközik valamivel (OnTriggerEnter2D), akkor az eltárolt mozgási irányt fordítsuk meg / invertáljuk.

Oda-vissza mozgó ellenfél (pl. légy) prefab/sablon készítése (2 pont)

Készítsünk az ellenfél objektumból egy prefabet/sablont. Helyezzünk el legalább 4 ellenfelet a játéktér tetszőleges pontjain. Az elhelyezéshez a prefab/sablont használjuk.

Véletlenszerűen mozgó ellenfél (pl. méhecske)

Véletlenszerűen mozgó ellenfél (pl. méhecske) importálása (1 pont)

Importáljuk az ellenfél képet/sprite-ot a segédfájlok közül (objects_tilemap.png-ben található a méhecske). Állítsuk be, hogy a kép/sprite a Unity játékterében egy egység területet fedjen le (Pixels Per Unit).

Véletlenszerűen mozgó ellenfél (pl. méhecske) objektum (3 pont)

Helyezzük el egy ellenfelet a játéktéren.
Implementáljuk, hogy az ellenfél folyamatosan mozogjon, fallal ütközés során és véletlenszerűen irányt váltson. A mozgás, fallal ütközés és játékossal ütközés megegyezik az oda-vissza mozgó ellenfél feladatban leírtakkal, ezen az ellenfelen szükséges egy script, mely az oda-vissza mozgó ellenfél funkcióit tudja. Ezt a script-et kell kiegészíteni a véletlenszerű irányváltoztatással:
A script egy float változójában tároljuk el a játékidőt, mikor következőleg irányt kell változtatnia az ellenfélnek. Az Update függvényben ellenőrizzük, hogy a Time.time nagyobb-e, mint az eltárolt idő, ha igen, akkor irányt kell változtatni. Ekkor tároljuk el az új időt (valahány másodperc a jövőben, a periódust akár Unity Editor-ból állítható változóként is tárolhatjuk) majd a mozgás irányának generáljunk egy új Vector2-t, melynek x és y komponensei [-1;1] közötti véletlenszerű számok.

Véletlenszerűen mozgó ellenfél (pl. méhecske) prefab/sablon készítése (2 pont)

Készítsünk az ellenfél objektumból egy prefabet/sablont. Helyezzünk el legalább 2 ellenfelet a játéktér tetszőleges pontjain. Az elhelyezéshez a prefab/sablont használjuk.

Nyitható terület (pl. kulcs és eltűnő faldarab)

Nyitó objektum (pl. kulcs) importálása (1 pont)

Importáljuk az nyitó objektum képet/sprite-ot a segédfájlok közül (objects_tilemap.png-ben található a kulcs). Állítsuk be, hogy a kép/sprite a Unity játékterében egy egység területet fedjen le (Pixels Per Unit).

Nyitható terület (pl. kulcs és faldarab) objektumok (3 pont)

Helyezzünk egy nyitó objektumot a játéktéren és egy faldarabot, melyet nyitni fog (a nyitható faldarabot ne a tilemap-re rajzoljuk, hanem egy külön önálló objektumként helyezzük el).
Implementáljuk, hogy a nyitó objektum (pl. kulcs) felvételekor a faldarab eltűnik és ezáltal egy elzárt terület járhatóvá válik. Ehhez a nyitó objektumon egy trigger ütközőre és script-re van szükség. A script-ben detektáljuk az ütközést (OnTriggerEnter2D) és ellenőrizzük, hogy az objektum, amivel ütköztünk az a játékos-e (van előre definiált Player Tag, ezt állítsuk be a játékos objektumán). A script-ben vegyünk fel egy GameObject típusú változót, mely az eltűntetendő objektumot tárolja, ide Unity Editorban tudjuk megadni a faldarabot. Játékossal ütközéskor semmisítsük meg ezt az objektumot.

Lövedéket lövő ellenfél/csapda (pl. kardot lövő csiga)

Csapda (pl. csiga) és lövedék (pl. kard) objektum importálása (1 pont)

Importáljuk a lövedéket lövő csapda és a lövedék objektum képet/sprite-ot a segédfájlok közül (objects_tilemap.png-ben található a csiga és a kard). Állítsuk be, hogy a kép/sprite a Unity játékterében egy egység területet fedjen le (Pixels Per Unit).

Lövedék (pl. kard) prefab/sablon készítése (3 pont)

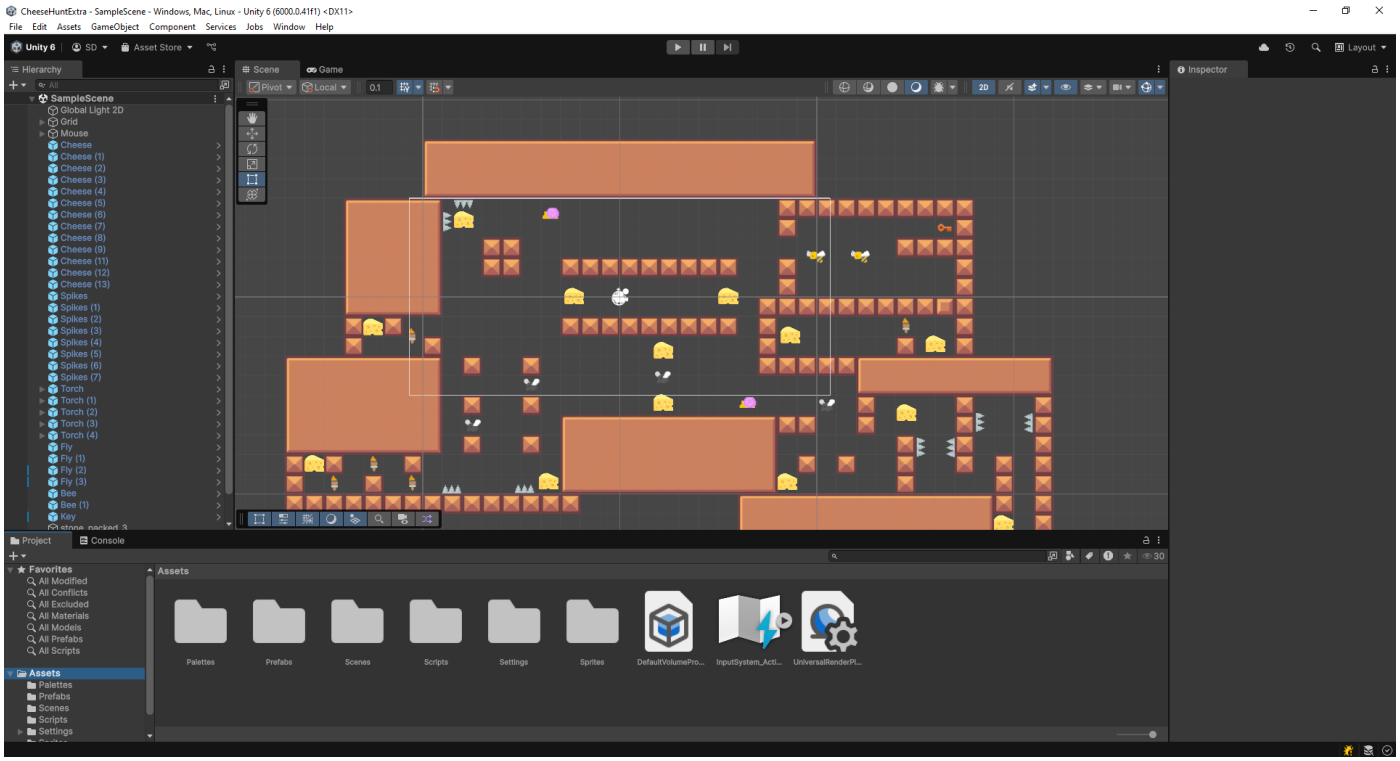
Helyezzük el egy lövedéket a játéktéren.
Implementáljuk, hogy az ellenfél mozogjon egy kezdő irányba és semmisüljön meg, ha összeütközik valamivel. A mozgás, fallal ütközés és játékossal ütközés megegyezik az oda-vissza mozgó ellenfél feladatban leírtakkal, ezen az ellenfelen szükséges egy script, mely az oda-vissza mozgó ellenfél funkcióit tudja. Ezt a script-et kell módisítani: ha összeütközik valamivel, akkor nem irányt változtat, hanem megsemmisül az objektum. Továbbá a mozgás iránya kívülről módosítható kell legyen (a csapda mondja meg milyen irányba megy a lövedék), így publikus legyen a változó vagy vezessünk be publikus függvényt, mellyel megadható értéke.
Ezután a játékos karakternek detektálnia kell az ütközést, ez a rész az Újjáéledés részfeladatban található.
Készítsünk a lövedék objektumból egy prefabet/sablont. Ezután a játéktérről törölhetjük a lövedék objektumot, a prefab/sablon a lövedéket lövő csapda által lesz példányosítva script-ből.

Csapda (pl. csiga) objektum (4 pont)

Helyezzünk egy csapda objektumot a játéktéren.
Implementáljuk, hogy a csapda objektum (pl. csiga) periodikusan lő egy lövedéket. Ehhez a csapda objektumon, mindössze egy script-re van szükség (ütközőre és Tag-re nem, nem a csapdától hal meg a játékos, hanem a lövedékétől). A script-en vegyünk fel változókat: egy prefab/sablon változó, melybe Unity Editorban megadhatjuk a lövedéket (célszerű olyan típusúnak lennie, mint a lövedékre helyezett saját script-ünk), egy irányvektor, amerre a kilőtt lövedékek mozognak majd. A periodikus lövedék ellövése nagyon hasonló a periodikus csapda és véletlenszerűen mozgó ellenfél működéséhez (idő eltárolása, Update-ben Time.time ellenőrzés, majd következő idő beállítása), abból a leírásból/kódból kell kiindulni. Ha az Update függvényben detektáltuk, hogy eltelt az idő és lőni kell, akkor az Instantiate függvénnyel példányosítsunk egy lövedék objektumot a csapda pozíciójában. Az Instantiate függvény visszaadja az új lövedék objektum példányt: be kell állítani rajta a mozgás irányát a csapda script-en megadott mozgási irányra.
Ezt követően Unity Editor-ban megadva a lövedék prefabet/sablont a csapdán, periodikusan lőnie kell egy lövedéket.

Kész!

A projekt elkészült! Töröljük a felesleges mappákat (Library), csomagoljuk be a projektet és töltsük fel Canvas-en!



Tippek:

- Elég összesen 2 új Tag felvétele: egy az ellenfelekhez, csapdákhoz és lövedékekhez és egy a jutalmakhoz (Player alapból van, az a harmadik amire szükség lehet)
- A játékos objektumon elég összesen 2 script: Egyik a megadott PlayerMovementScript, másik pedig a részfeladatokat együttesen megvalósító script (Pl. PlayerScript). Ez a script képes kezelni a Start-ban a kezdőhely elmentését, az OnTriggerEnter2D-ben az ellenfelekkel, csapdákkal és jutalmakkal ütközést, az Update-ben pedig a lövést. Haladók akár a PlayerMovementScript-et is kiegészíthetik mindezzel, úgy az az egyetlen script is elég.
- Több ellenfél és csapda is van, sok munkának tűnik megoldani, hogy a játékos mindegyikkel ütközzön, de ez nincs így. Ha a játékos objektumon van egy script, ami az OnTriggerEnter2D-ben ellenőrzi, hogy egy ellenfél Tag-el ütközött-e (.CompareTag), akkor, ha mindegyik csapdára és ellenfélre ugyanezt a Tag-et állítjuk be, akkor ugyanúgy fog reagálni a játékos karakter mindegyik objektumra, mely meg akarja ölni őt. Ez nekünk most teljesen jó megközelítés, ennél több rugalmasság nem kell.
- Script-ben a saját objektumunk pozíciója: transform.position
- Script-ekben a pozíciók Vector3 típusok, előállításuk: new Vector3(x, y, 0); //x és y tetszőleges változók, értékek, kifejezések, Z koordinátát 2D-ben nem használjuk ezért 0
- Script-ben, ha detektálni akarjuk, milyen Tag-el rendelkezőobjektummal ütköztünk, akkor OnTriggerEnter2D(Colldier2D collision) függvényre van szükség. A collision paraméter az objektum amivel ütköztünk. Egy if feltételébe helyezve a collision.CompareTag(„”) feltételt, a CompareTag paraméterében string-ként megadható a Tag, amit ellenőrizni akarunk az objektumon amivel ütköztünk.
- Script-ben, ha meg akarunk semmisíteni egy objektumot amivel összeütköztünk, akkor OnTriggerEnter2D(Colldier2D collision) függvényre van szükség. A collision paraméter az objektum amivel ütköztünk. Destroy(collision.gameObject); megsemmisíti az objektumot amivel ütköztünk.
- Script-ben felvehetünk változókat, melyeket a Unity Editor-ban elérünk. Ha valamilyen Unity komponens típusát használjuk a változó típusának, akkor Unity-ben behúzhatunk egy objektumot a játéktérről vagy prefab-et/sablont a projektből, melyet el fog érni az a script.
    - Másik objektum pozíciójának elérése
        - public Transform valtozonev; //változó deklarálása az osztályban

        - valtozonev.position //pozíció elérése valamelyik függvényben
    - Másik objektum elérése
        - public GameObject valtozonev; //változó deklarálása az osztályban
    - Prefab/sablon elérése
        - public GameObject valtozonev; //változó deklarálása az osztályban (GameObject helyett lehet egy a sablonon/prefabon lévő komponens is
        - Instantiate(valtozonev); //új objektum létrehozása a sablontól valamelyik függvényben
        - Instantiate(valtozonev, new Vector3(x, y, 0), Quaternion.Identity); //új objektum létrehozása a sablontól adott x, y pozícióban valamelyik függvényben
        - GameObject newObject = Instantiate(…) //Az új objektumot megkapjuk, módosíthatjuk, állíthatjuk paramétereit
        - MyScript newObject = Instantiate(…) //GameObject helyett használhatjuk a saját script osztályunkat (ilyenkor a prefab/sablon változó típusa is ugyanez kell legyen)
- Véletlenszerű számgeneráláshoz használható a hagyományos C# megoldás is, de Unity-ben van saját beépített generáló:
    - UnityEngine.Random.Range(-1f, 1f) //-1 és 1 között ad vissza float számot
    - 1f = float típus, azért, hogy törtszámokat is generálhasson. Ha csak 1-et írunk f nélkül, akkor int típust adunk meg, tehát csak -1 és 0 lehet a két egész szám amit generálhat.
- A Vector2 és Vector3 típusokon használhatjuk a .normalized tulajdonságot. Ekkor egység hosszúra normalizálja a vektort, melyet visszatérési értékként ad vissza (pl. véletlenszerű x, y komponenseknél hasznos lehet)