Thief Ekedition rendszerterv

* A rendszer célja: Egy FPS lopakodós-tolvajos játék készítése.
* Projekt terv: Három egyenrangú munkásunk van a feladatra, és fejenként a többi elfoglaltságunk mellett nagyjából heti X órát tudunk rászánni a feladatra. Ilyen körülmények között a cél, hogy készítsünk egy TestLevel-t, melyben a félév során leprogramozott objektumok, scriptek szerepelnek, összeállva egy játszható pályává, ez lesz a 0.1-es verzió.

Ebben csupán a Játékos Karakter mozgásának, a fények és hangok számításának működésének, és néhány használható felszerelési tárgy bemutatásának kell szerepelnie. Mindez természetesen a ThiefObject osztályhierarchia alapján.

* Üzleti folyamatok modellje:
* Követelmények: Hasonló irányítás az előd-játékhoz, grafikusan tovább lépje azt.
* Funkcionális terv: Minden objektumnak, amely rendelkezik Interact eljárással, használatkor történnie kell valaminek. A szereplőnek reagálnia kell a fényekre (fénykő), a zuhanásra, sebződésre (sérülés), illetve a lopakodás funkciók működjenek (sétálás, vagy guggolás, kisebb zaj csapás, ha más felületen mozgunk.) Törvényi előírás tudtunkkal nincs a „tolvaj szimulátor játékok készítésére.
* Fizikai környezet: Unity-ben fejlesztünk elsődlegesen a Windows 10 operációs rendszert célozva.
* Absztrakt domain modell: A felhasznló egy karaktert irányithat, a tárgyakat lophat el, feladatokat old meg a játék közben (pl. bejutás a pályára).
* Architekturális terv: Mivel a játék egyjátékos módban játszható, ezért az egyetlen követelmény, hogy a felhasználó számítógépe rendelkezzen olyan erőforrással, amely képes futtatni a programot.
* Adatbázis terv: Idővel a 0.1-es verzió után szerepelni fog mentés/betöltés a játékban, melyet adatbázis segítségével fogunk megoldani.
* Implementációs terv:
  + Csinálunk egy ThiefObject absztrakt osztályt. Ebben objektum összetétellel megvalósítunk egy Material típusú mezőt.
  + Ebből több absztrakt osztályt is létrehozunk, elkülönítve a tereptárgyakat, a gyűjthető tárgyakat, valamint a mesterséges intelligenciával rendelkező objektumokat (lényeket).
  + Ezeknek szintén lesznek még gyermekosztályaik, s végül nem csak absztrakt, de konkrét osztályok is.
  + A 3D térben elhelyezett alakzatok mind kapnak egy-egy ilyen ThiefObject scriptet, amennyiben a játék szabályai által kezelt objektummá akarjuk tenni őket.
  + Az AbstractMaterial script alapján minden gyerekosztálynak rendelkeznie kell az alábbi 3 metódust:
    - float NoiseGeneration(float noise); ami bemenő paraméterben megkapja, hogy milyen intenzitással léptek (vagy ütötték meg) az adott objektumot, és visszaad egy a felület alapján módosított értéket, valamint a játékos által is érzékelhető, a felületnek megfelelő hangot ad. (egy szőnyeg hangosabb, mint egy fémlépcső)
    - bool Soft(); igaz, ha a felület puha, és a belelőtt nyílvesszők beleragadnak. Hamis, ha rálőtt vesszők lepattannak. (fa vs kő)
    - bool SeeThrough(); Igaz, ha a felület átlátszó, és átereszti a fényt. Hamis, ha akadályozza a fény terjedését. (ablak vs téglafal)
  + Lesz egy script, ami a Játékos Karakter mozgásáért felel. Ebben az összetett Player objektum minden tagja kezelve van. A Player objektum áll egy kamerából, 3 láthatósági pontból és egy a fizikai ütközéseket érzékelő collider-ből.
  + Az aktuális mozgási állapot bool változók beállításával történjen.
  + Legyen egy script, ami objektumokból kapaszkodópontot csinál. A mozgásért felelős scriptben a megfelelő gomb lenyomásakor ha x tavolságra raycastolva egy ilyen kapaszkodható objektum van, akkor másszon fel a tetejére.
  + Az előző módszert követve, ha a játékos a kamera pontjából indított, legfeljebb x távolságra mutató raycast olyan Thiefobjecet talál el, ami gyűjthető(Collectible), akkor hívja meg a destruktorát, és adja hozzá a scriptben a megfelelő int értékhez (ha Valuable) vagy adja hozzá az Equipment listához (ha Equipment).
  + Az előző módszer alapján ha az eltalált objektum DinamicFieldObject, akkor hívja meg annak az Interact(bool withRightClick) függvényét TRUE értékkel.
  + A játékos mozgását irányító scriptbe kell egy lépéskezelő ciklus. Minden esetben, amikor ez végigér (leér a lábunk) vagy amikor ugrás után földet érünk, hívja meg annak a ThiefObjectnek a material.NoiseGeneration(float) függvényét egy értékkel, amit a lefelé irányzott raycast eltalál.
  + Csinálunk egy scriptet IsInShadow néven. Ennek átadunk egy fényekből álló listát. Minden fény esetén a range mező figyelembe vételével megnézi, hogy elég közel vagyunk-e a fényhez. Ha igen, akkor raycastAll segítségével megnézi, hogy a Player láthatósági pontjai és a fény középpontja közt van-e olyan ThiefObject akadály, aminek a material.SeeThrough(bool) metódusa FALSE-t ad vissza. Ha igen, akkor a pont nincs megvilágítva a fény által. Ha nem, akkor egy metódusba kiszedett függvény alapján kiszámol egy értéket hozzá. Miután minden fényre kiszámolta, a három láthatósági ponthoz rendelt értékek alapján megadja az össz (overall) megvilágítottságát a játékosnak. Ezt átadja a Fénykő objektumnak, ami a felhasználó számára is megjeleníti ezt az értéket.
  + Egy újabb script érzékeli a NoiseGeberation által generált zaj értékeket. 0.1 megoldás, később ez a script be lesz építve a Creature osztályba.
* Tesztterv: Egy-egy issue megoldása után kipróbáljuk az adott beállitásokat, hogy működnek-e, majd ezt követően Commitoljuk.
* Telepítési terv: Egyelőre nincs ötlet a telepítésre.
* Karbantartási terv: Ha kiadnánk a játékot, a visszajelzésekből tudnánk következtetni a játék hibás és nem megfelelően működő részeire, melyeket javitanánk egy újabb verzióban.