MŰEGYETEM 1782

A PROGRAMOZÁS ALAPJAI 2.

HÁZI FELADAT DOKUMENTÁCIÓ

THE WAREHOUSE EMERGENCY

KÉSZÍTETTE: MATUS ZOLTÁN, R9FJIV
zoltan.matus@edu.bme.hu

KÉSZÍTÉS FÉLÉVE: 2024/21/5

TARTALOMJEGYZÉK

Felhasználói dokumentáció

A program használata egyszerű, nem igényel különösebb informatikai tudást. A programot elindítva egy térképet látunk, ahol különböző színekkel vannak jelölve felvehető tárgyak és a játék célja, a megjavítandó generátor. A színek a következő tárgyakat jelentik:
- Kék: Vezeték
- Piros: Üzemanyag
- Sárga: Akkumulátor
- Fehér: Biztosíték
- Zöld: Generátor

Ezen a képernyőn továbbá látható a továbbjutáshoz szükséges gomb megnyomására buzdító szöveg is, amit, ha követünk, akkor a Bevezető képernyőre jutunk, ahol a játék tájékoztatás ad arról, hogy mi a feladatod és, hogy milyen gombokkal tudod irányítani a karakteredet. Ezeket a rend kedvéért ide is leírom:
- W – Előre mozgás
- A – Balra mozgás
- S – Hátra mozgás
- D – Jobbra mozgás
- Szóköz – Ugrás
- F – Interakció
- SHIFT + P – Kilépés a játékból (program bezárása)
- Egér X/Y – Körbenézés

A feladatot is ezen a képernyőn láthatjuk, mint már említettem, ami nem lenne más, mint összegyűjteni:
- 1 legalább 70%-os töltöttségű akkumulátort,
- 25 liter üzemanyagot,
- 20 méter vezetéket,
- 1 darab használható biztosítékot

Ezek összegyűjtésére van 1.5 percünk (5 perccel túl könnyűnek bizonyult a játék…).

Ezek után, ha megnyomjuk a továbblépéshez szükséges gombot, akkor a játék elindul és ezzel együtt nekünk is el kell indulni megkeresni a tárgyakat, amik a pálya különböző részein vannak elszórva.
- Ami nehezíti a játékot, hogy a tárgyak véletlenszerű tulajdonságokkal „születnek”, így nem olyan egyszerű pont mindig a pontos méretű / egységű / állapotú tárgyat kivinni.
- Ezen felül még egy olyan nehezítés is implementálva lett a játékban, hogy a leltárunk súlyától függ a sebességünk, ezért nem érdemes mindent felvenni.

Ha már viszont mindent megtaláltunk, amit a játék a zöld színre váltó szövegekkel a bal felső sarokban jelez, akkor a generátor felé vehetjük az irányt és úgymond beindíthatjuk. Ha ezt megtettük időn belül, akkor megnyertük a játékot.

Persze nem csak megnyerni kell, hanem rekordot is elérni, mivel egy rekordot számító és állító funkció is implementálásra került, ami az akkumulátorunk töltöttségéből és a fennmaradó időből számolja a rekordot, ezért a játék lehetséges elvesztésén kívüli még a rekordunk miatt sem érdemes sok időt tölteni a tárgyak keresgélésével.

A végén lehetőségünk van újra játszani a szóköz billentyű lenyomásával és kilépni is a már említett Shift + P-vel (ezt akármikor meg lehet tenni, nem csak a játék végén).

Osztályok statikus leírása

Player_Controller

Felelőssége

Ez az osztály felelős elsősorban a játékos mozgatásáért, és nem utolsó sorban a teljes játékmenet vezérléséért. Különböző bemenetekhez csatol függvényeket az adott jelenethez megfelelően.

Ősosztályok

- APlayerController
  Azért öröklődik ebből az osztályból, mert a játék motor saját osztályait is igyekeztem a lehető legtöbb helyen alkalmazni, nem véletlenül, hiszen ezek segítségével tudom használni magát a motort. Bővebben erről az osztályról: APlayerController - Unreal Engine 5.3

Attribútumok

Privát

- PlayerCharacter[Player_Character*]
  A játékos által irányított karakter osztálya.
- PalyerCharacterHUD[HUD*]
  A játékos képernyőjére rögzített elemek osztálya.
- GameTimer[Timer]
  A játék időzítő osztálya.
- TimerHandle
  Az időzítőt segítő osztály.

Metódusok

Privát

- BeginPlay()
  A játék kezdetén meghívódó függvény.
- ExitAction()
  A játék bezárását elintéző függvény.
- RestartAction()
  A játékmenet újrakezdését elintéző függvény.
- RunTimerAction()
  Az időzítő futtatását végző függvény.
- SwitchToIntroductionAction()
  A bevezető képernyőre váltó függvény.
- SwitchToGameAction()
  A játék képernyőre váltó függvény.
- CallMoveX()
  Az előre vagy hátra mozgás függvényt meghívó függvény.
- CallMoveY()
  Az oldal irányba mozgás függvényt meghívó függvény.
- CallTilt()
  A kamera bólintás függvényt meghívó függvény.
- CallTurn()
  A kamera forgatás függvényt meghívó függvény.
- CallJump()
  Az ugrás függvényt meghívó függvény.

- PickEventFlagOn()
  A tárgy felvételének „bekapcsolása” függvényt meghívó függvény.
- PickEventFlagOff()
  A tárgy felvételének „kikapcsolása” függvényt meghívó függvény.
- ProcessTheHighest()
  Ez a függvény dolgozza fel a rekordot és állítja be az újakat.
- BindStartState()
  Kezdő képernyőn használatos függvények vezérlőkhöz rendelése.
- BindIntroductionState()
  Bevezető képernyőn használatos függvények vezérlőkhöz rendelése.
- BindGameState()
  Játék képernyőn használatos függvények vezérlőkhöz rendelése.
- BindResultState()
  Játék kiértékelése képernyőn használatos függvények vezérlőkhöz rendelése.
- ClearAllBindings()
  Minden függvény leszedése a vezérlőkről.

Publikus

- Player_Controller()
  Az osztály konstruktora.
- Tick()
  Minden képkockánál lefutó függvény.
- SetupInputComponent()
  A kezdő(játék kezdeténél) függvény hozzárendelések vezérlőkhöz.
- SetGameSceneState()
  Ezzel a függvénnyel lehet beállítani, hogy milyen fázisban legyen a játék.
- SwitchToResultAction()
  Az összegző képernyőre váltó függvény.

Player_Character

Felelőssége

Ez az osztály képezi a karakterünk fizikai tulajdonságait, mint a test vagy a kamera (látásért felelős).
Ezen felül a test és a kamera mozgatását végző függvények is itt találhatóak.
Nem utolsó sorban pedig a játékos leltára is itt kap helyet.

Ősosztályok

- ACharacter
  Azért öröklődik ebből az osztályból, mert ez is a motorhoz kapcsolódó objektum és így lehetséges a kommunikáció a motorral.
  Bővebben erről az osztályról: ACharacter - Unreal Engine 5.3

Attribútumok

…

Metódusok

…

UML osztálydiagramm

…

Összegzés

Mit sikerült és mit nem sikerült megvalósítani a specifikációból?

Úgy vélem, hogy a specifikációból mindent sikeresen teljesítettem, bár néhány akkori elképzelés kicsit másképp lett elkészítve. Erre jó példa a HUD, mivel jóval több dolog kijelzésére képes, mint ami az eredeti elképzelés volt, továbbá az időzítő, ezesetében nem egészen a logika változott, hanem az idő, mivel az 5 perc nem jelentett túl nagy kihívást, ezért 1.5 percre állítottam át, viszont ez is állítható még egy külső fájlból, így talán ez nem olyan nagy eltérés.

Ami viszont sajnos nem egészen a szokásos módon tetten érhető, az a hibakezelés, mivel az Unreal Engine nem támogatja a hagyományos C++-os exception kezelést, hanem saját verziója van rá. Így én is kénytelen voltam az Unreal saját hibakezelési rendszerét használni, ami főként makrókból áll és kifejezetten sok funkcióval rendelkezik.

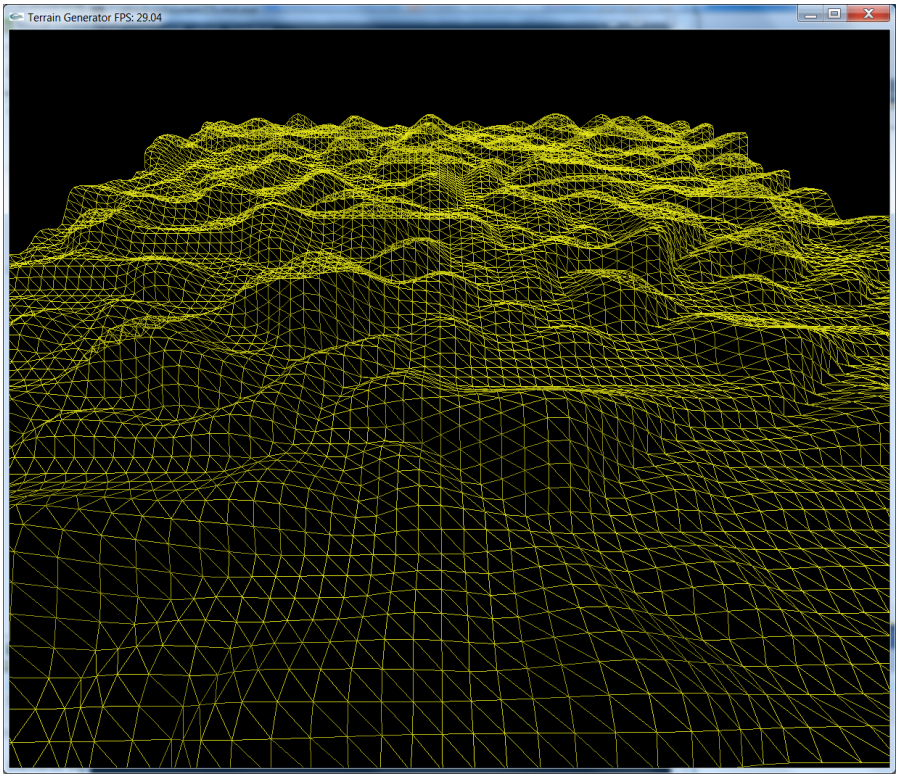
Mit tanultál a megvalósítás során?

Sokkal könnyebb volt, mint amire számítottam még úgy is, hogy ezelőtt nem készítette soha teljes játékot Unreal Engine-ben. Nehézségnek tudnám be azt, hogy maga a környezet azért viszonylag sok beletanulást igényelt, mivel nem csak egy sima C++, hanem a megszokottól eltérő, más logikákat is használó, játékkészítésre tervezett és optimalizált C++ variáns.

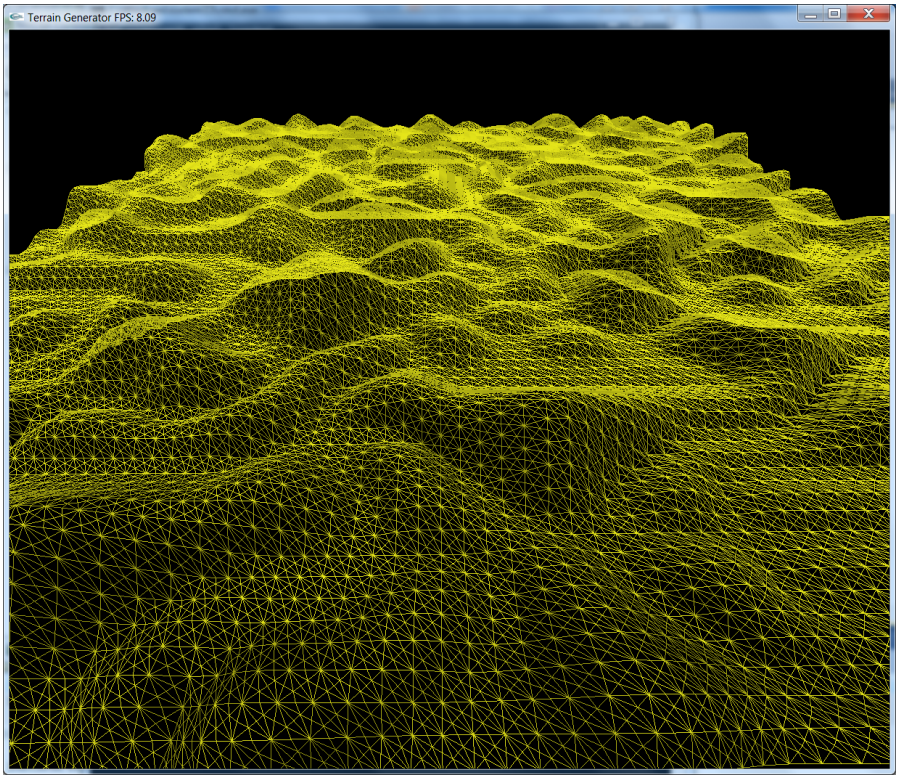
Továbbfejlesztési lehetőségek

Bármi szóba tud jönni egy videojáték esetén továbbfejlesztési ötletként, ezért reális keretek között azt tudnám mondani, hogy több pályával ki lehetne egészíteni, több gyűjthető tárggyal, esetlegesen egy többjátékos móddal is, és persze amire talán a legjobban ráfér a továbbfejlesztés az a felhasználói felület, mivel az sajnos elég egyszerű lett.

Képernyőképek a futó alkalmazásról



2. ábra A Triangle osztály felhasználása dinamikus terepgenerálásra



1. ábra Tesszaláció nagyobb felbontással