Dokumentáció

Guti Olivér-HSGASE

Panyik Péter Csaba-J83KNA

Szemesi Mónika-D0YVVS

Neumann János Egyetem Gamf Műszaki és Informatikai Kar mérnökinformatikus szak

Vizuális Programozás beadandó 2021.05.

Beadandó neve: PewPew Paradise

Release letöltése: <http://home.pewpewparadise.nhely.hu/>

Teljes source elérés: <https://github.com/Peter226/PewPew-Paradise>

Fejlesztői dokumentáció:

Program célja:

A program célja, hogy egy gyerekek számára élvezhető játékot nyújtson és elért eredményeiket egy adatbázisban tárolja. Az előzetes eredmények mutatását lekérdezéseken keresztül egy gombnyomással érhetik el a felhasználók.

Előzetes rendszerterv:

A program beolvassa az elkészített adatbázist és a hozzá készített design elemeket. A program betölt egy WPF ablakot, ami egy fő menüt tartalmaz. Ott gombok segítségével kiválaszthatjuk, hogy mit szeretnék elérni (2 játékmód, beállítások, segítség, ranglista, kilépés). Egy játékmód kiválasztásával betölti a játszható karaktereket, ahol ki tudjuk választani, amivel játszani szeretnénk, illetve nevet tudunk adni magunknak, ami később a ranglistán jelenik meg. Ha elkezdjük a játékot egy pályát töltünk be megfelelő mennyiségű ellenséggel, illetve betöltjük a lövedékek és a gyümölcsök képeit. A játékos tud mozogni és le tudja lőni az ellenséget, aki utána egy véletlenszerű gyümölcsöt tölt be a helyén, amit a játékos pontokért tud gyűjteni. A pálya az ellenség halála után automatikusan változik és a teljesített pályák számával nehezedik a játék. Ha a játékos hozzáér egy ellenséghez 3-szor, akkor meghal. Ha minden játékos halott vége a játéknak, akkor a játéknak vége, az adatok betöltésre kerülnek az adatbázisba. További lehetőségek: a játék bármikor megállítható anélkül, hogy a játékos meghaljon, zene és effektek hangerejének állítása, adatbázis adatainak teljes törlése, ranglista megtekintése.

Vizuális tervek: PewPew Paradise.bmpr

Megvalósítási tervek: PewPew Paradise RoadMap.docx

Részletes rendszerterv:



AnimatorComponent, SpriteAnimation és AnimationCollection:

A Sprite-ok animációjához szükséges objektumok. A SpriteAnimation egy Vector2 szekvenciát tárol, melynek segítségével az AnimatorComponent a Sprite textúra atlaszából kiválasztja a megfelelő képkockát. Az AnimationCollection osztály példányaiban tároljuk a Sprite típusoknak szánt animációt. Egy AnimationCollection osztályt létrehozva az automatikusan eltárolja magát a „SpriteManager” egy statikus szótárában, a könnyebb elérés érdekében. Így az AnimatorComponent-nek csak a „SetAnimation” függvényét kell meghívnunk, melynek átadjuk a létrehozott animáció gyűjtemény nevét.

Tartalmaz még az AnimationCollection egy statikus „LoadAll” függvényt is, melyben betöltjük az összes animációt, amit a játékban használunk, a rendezettség érdekében.

Az animációknak beállíthatjuk a képkockasebességüket, és a prioritásukat is. A prioritás felel az animációk megfelelő lejátszásáról, egy már játszódó animációt csak egy magasabb prioritású írhat felül az AnimatorComponent „PlayAnimation” metódusát használva. Ezt a kikötést kikerülhetjük a „ForcePlayAnimation” metódussal, mely prioritástól függetlenül írja felül az animációt.

PhysicsComponent és CollideComponent:

A Sprite-ok „fizikájához” szükséges objektumok. A PhysicsComponent-ben a Sprite-ok „gravitációját” határozzuk meg. Az Update függvényben ha éppen az adott Sprite leugrik/leesik az eltelt idő függvényében növeljük az esési sebességét. Ezt az esési sebességet a Disabled függvény segítségével ki is tudjuk kapcsolni.

A CollideComponent-ben folyamatosan vizsgáljuk hogy az adott Sprite-unk pozíciója éppen hol van. Ennek segítségével tudjuk vizsgálni hogy a Sprite érintkezik-e platformmal, és ha igen, mit csináljon. Minden oldalról vizsgáljuk hogy melyik az a távolság a platform és a Sprite között ami a legkisebb, és ennek függvényében áll meg a platformon, áll meg a fal mellett vagy ugrás közben nem engedi átugrani a felette lévő platformon a Sprite-ot.

A Physics és CollideComponentet a játékos, ellenség és gyűjthető tárgyak Sprite-jaira hívjuk meg.

CollisonEditor és DebugSprite:

A CollisionEditor egy csak pályaszerkesztésre használt osztály, nincs játékbeli szerepe. Segítségével lehet a pályákon a hitboxokat (szilárd téglalap alakú objektumokat, amiken nem esnek át a tárgyak) megrajzolni.

Használata: Az első dolog CollisionEditor.Init meghívása, mely után használható a szerkesztő. Ezután bal egérgombot lenyomva és az egeret elhúzva rajzolhatunk. Ctrl+Z-vel vissza lehet vonni, Delete billentyű törli a kijelölt elemet. Ctrl-t lenyomva tartva rajzolva 8x8 pixeles egységekre ugrik a doboz mérete. Shift-et lenyomva tartva egy nagyítót kapunk. Ctrl+S elmenti a változtatásokat.

A DebugSprite segít a hitboxok vizualizálásában, megjeleníti azokat. SpriteManager.DebugRect metódussal lehet őket létrehozni. Használatuk után a DebugSprite-ok törlődnek a megadott időn belül.

Enemy és EnemySprite:

Az EnemySprite a Sprite osztály gyerekosztálya, aminek nincs új paramétere a Sprite-hoz képest. EnemySprite létrehozásakor azonnal hozzáadjuk a komponenseket inaktívan, az animációját Enemyre állítjuk, illetve az Enemy osztály enemyList nevű listájához adja magát. Ha az EnemyDeath függvényt meghívjuk, akkor lejátssza az Enemy 3. animációját, és a FinishDeath függvényt feliratkoztatjuk az OnAnimationEnded eseményre. A FinishDeath függvény az animáció végén létrehoz egy random FruitSpriteot a saját helyén. Az Update függvény overrideolása alatt készítjük el az enemy AI-t. Létrehozunk egy timert, ami növekedése szerint jobbra vagy balra mozog az EnemySprite és minden második másodpercben ugrik. Ugyanitt lejátsszuk az animációit a mozgásnak. Ha egymáshoz érnek az EnemySprite-ok, akkor az egyik begyorsul. Az Enemy osztályból tehetünk aktívvá egy EnemySprite-ot az EnemyLoad függvénnyel, illetve az AddEnemy függvénnyel hozhatunk létre új EnemySpritot.

FruitSprite és FruitType:

A FruitSprite és FruitType objektumainkban a gyűjthető gyümölcseink adatait tároljuk és határozzuk meg. A FruitSprite objektumban két listát hozunk létre. Egy fruitTypes listát melybe a gyümölcseink nevét és értük járó pontszámot tároljuk egy FruitType struktúra segítségével, és egy fruitList listát melyben tároljuk a pályán lévő gyümölcsöket.

A konstruktorban meghivjuk rájuk a Collide,Physics és PortalComponent-et, majd hozzáadjuk a fruitList listához.

LoadImages függvényben az összes gyümölcsöt és hozzájuk tartozó pontokat hozzáadjuk a fruitTypes listához, és tároljuk az adatait a FruitType struktúrában. A LoadImage paranncsal hozzárendeljük a fentebb létrehozott gyümölcshoz a hozzá tartozó képet.

A FruitCollect függvényben ha egy gyümölcsöt felvesz a játékos akkor ezt a gyümölcsöt kitörli a fruitList listából és a Destroy függvény eltünteti az adott gyümölcsöt a pályáról.

GameManager és GameOptions:

A GameManager osztály tartalmazza a játék futásához szükséges OnUpdate eseményt, melyet képkockánként meghívunk. Erre feliratkozva bármely más objektumból, kapunk egy hasonló időközönként lefutó metódust. Ez használható például animációra, mozgatásra, időzítésre. Ha a GameManager Init metódusa 60-as számot kapott, akkor az Update másodpercenként körülbelül 60-szor fut le (16-17ms-enként). A Begin metódus elindítja az Update-et, a Stop leállítja. A GameManager DeltaTime értéke mondja meg hány ms telt el az előző és a mostani Update hívás között. Emellett van még OnPreUpdate és OnPostUpdate, melyek az OnUpdate előtt, és után futnak le. Az Update metódust egy DispatcherTimer hívja meg, melynek működését a „Dummy” UIElement ösztönzi.

A GameOptions struktúra a játék beállításait (pl. karakterek, hangerő) tárolja, melyet elmentünk egy json file-ba.

MapSprite és MapLoad:

A MapSprite a Sprite objektum gyerekosztály, aminek két új paramétere van, az enemy, ami a maphoz tartozó enemy neve, illetve map\_background, ami a pálya háttérszíne. MapSprite létrehozásakor meghívja a DeserializeMap függvényt, ami betölti a maphoz tartozó hitboxokat. A SerializeMap elmenti a megrajzolt hitboxokat. A MapLoaded és a MapUnloaded invoke-olja a hozzá tartozó eseményt. Az Update override-ján megnézi, hogy mikor kell betöltenie a mapnak, aktiválja az animációkat és változtatja a háttérszínt. Ha nincs aktív EnemySprite, akkor egy timer segítségével 4 másodperc elteltével a következő pályát tölti be, Inaktiválja a maradék FruitSpriteot. A MapLoad létrehozásakor betölti a pályákhoz és ellenségekhez tartozó képeket, a pályáknak MapSprite-ot hoz létre és egy listába menti őket. A LoadMap függvény betölti a pályát a megfelelő PlayerSprite(ok)kal. Meghívja a Dangerzone függvényt kétszer és a PortalDanger függvényt, Rect-be menti őket. Random pozícióba EnemySpriteot hoz létre úgy, hogy a Rect-be mentett adatokon kívül essen a pozíció, a mennyiségük a teljesített pályák mennyiségétől függ. A Dangerzone és PortalDanger függvények miatt nem hozhatunk létre EnemySprite-ot a PlayerSprite-ok közelében, illetve a „portál” felé. A ClearAll függvény törli a listák tartalmát, alaphelyzetbe állítja a játékot.

PortalComponent:

A PortalComponent A SpriteComponent gyerekosztálya. A PreUpdate override-ján, ha a Sprite pozíciója nagyobb, mint a pálya vége, akkor a „portált” használva felteleportál a map tetején lévő lyukba.

PlayerSprite és ProjectileSprite:

A PlayerSprite és a ProjectileSprite a Sprite objektum gyerekosztálya. A ProjectileSprite nem tartalmaz új paramétert Sprite-hoz képest. A PlayerSprite-nak 2 új paramétere van: player\_id, azaz melyik játékosé, illetve projectile, ami annak a képnek a neve, ami a hozzá tartozó lövedéké. A PlayerSprite létrehozásakor hozzáadjuk a komponenseket inaktívan, illetve feltöltjük a \_keys nevű listát a hozzá tartozó irányítással és a Life nevű integert a maxLife értékére állítja. A Life frissíti magát amikor a PlayerSprite életereje változik, és ahhoz megfelelő életerő Sprite-okat tölt be. A MoveLeft, MoveRight és Jump mozgatja a PlayerSprite-okat és lejátssza az animációt. A Shoot készít egy ProjectileSprite-ot. Az Update override-ja közben megnézzük, hogy az adott billentyűk le vannak lenyomva, ha igen meghívjuk a billentyűhöz tartozó irányítást. Ugyanitt megnézzük, hogy a PlayerSprite érintkezett-e FruitSprite-tal, ha igen, akkor a megfelelő pontokat hozzáadjuk egy számlálóhoz, illetve érintkezett-e EnemySprite-tal, ha igen életerőt veszít. Néhány funkció csak időnként történhet meg (pl: lövés, sebződés), erre egy timer-t használunk. A FinishDeath meghívásakor inaktiválja azt a PlayerSprite-ot, amelyik életereje 0, illetve eltárolja az adatait (elért pont, elért pálya), ha minden aktív játékos életereje 0, akkor megjeleníti az eredményeket egy másik rácsban, visszaállítja a PlayerSprite-ok adatait (életerő, nagyság stb.) alaphelyzetbe. A ProjectileSprite létrehozásakor komponenseket adunk hozzá. Az Update függvény override-ján mozog a ProjectileSprite egyenesen, abba az irányba amerre a PlayerSprite nézett (PlayerSprite size vektora alapján), és ha érintkezik egy EnemySprite-tal, meghívja az adott EnemySprite-on az EnemyDeath-et.

Sprite, SpriteManager és SpriteComponent:

A SpriteManager felel a Sprite-ok megfelelő létrehozásáért, képük betöltésért, animációk eltárolásáért.

A Sprite osztály az alapja az összes játékbeli objektumnak. Lényegében példányai képet jelenítenek meg magukról a képernyőn egy Rectangle UIElement segítségével. Két fő vektor tulajdonsága a pozíciója és a mérete. Ezek megváltoztatása a képernyőn való frissítést eredményezi. Játéktér szerinti egységeket használ, melyeket megjelenítéskor képernyő szerinti koordinátákká konvertál a SpriteManager segítségével. Virtuális metódusai miatt kifejezetten hasznos gyerekosztályok hozhatók létre. Az IsActive tulajdonságával ki lehet kapcsolni, így a képernyőn is eltűnik, és működése leáll a következő bekapcsolásáig. A ki- és bekapcsolás az OnEnable és OnDisabled metódusok meghívását vonja maga után, melyek felülírható metódusok. A Start metódus a Sprite létrehozásakor fut le, az Update pedig képkockánként, ezek szintén felülírható metódusok. A Destroy metódussal lehet végleg kitörölni az objektumot.

Az AddComponent felel a komponensek hozzáadásáért, A RemoveComponent az eltávolításáért, a GetComponent-el pedig le lehet kérni egy komponenst a Sprite-ról (ha nincs hozzáadva akkor null). A komponensek extra funkciók hozzáadását teszik lehetővé a Sprite-oknak, melyeket gyakran használnánk teljesen különböző Sprite-okon. A komponenseknek szintén van IsActive tulajdonságuk, Enabled, Disabled, Destroy, Start, Update metódusuk, valamint PreUpdate, PostUpdate és egy OnParentDestroyed metódus, mely akkor fut le, ha a Sprite amire helyeztük a komponenst kitörlődik.

AccessData, CharacterTable, Connection és Highscore:

Ezek az objektumok az adatbázis kezeléséhez szükségesek. A Highscore és a CharacterTable ad tulajdonságot az adatbázis tábláinak. A Connection egy statikus függvényt tartalmaz, a Connect-et, amely visszaadja a kapcsolatot a létrehozott kapcsolat nevéből. Az AccessData függvényei adják vissza az általunk lekért adatokat, vagy hoznak létre/törölnek. A GetScore függvény kilistáz minden adatot, a ScoreAmount pedig megmondja mennyi adat van a Highscore táblában. A GetOrderedScore és GetOrderedFloor rendezi az adatokat. A GetMostplayedCharacter visszaadja a legtöbbször szereplő characterid-t. AddScore, AddChar és InitDB adatot tölt fel, ClearDB töröl minden adatot. A GetCharname visszaad egy character nevet id alapján.

Vector2:

Egy kétdimenziós vektor struktúra, melyet főleg a Sprite-ok használnak. Konvertálható a következő struktúrákká: Point, Size, Vector. Operátor overload-jai miatt a legtöbb alapművelet végrehajtható rajta. Ezek mellett konvertálható szöveggé, kerekíthető, normalizálható, ki lehet számítani a hosszát, abszolút értékét, két vektor távolságát, és egy Lerp (linear interpolation) függvénye is elérhető.

CharacterSelect:

A Characterselect objektum létrehozásakor betöltjük a lövedékek, a karakterek és az életerő képeit, illetve új PlayerSprite-okat hoz létre és ezeket 2 listába menti (chars1, chars2). A függvények meghívásakor meg kell adni a player\_numbert, ami arra utal, hogy 1 vagy 2 játékosra akarjuk. A LoadChar betölti a PlayerSprite-okat a karakterválasztáshoz kellő adatokkal. A NextChar és PreChar az a PlayerSprite-ok váltásához jó karakterválastásnál. Az UnloadChar inaktívvá teszi a PlayerSprite-okat. A SelectedChar visszaadja az éppen aktív PlayerSprite-ot. A CharacterLoad a játékhoz teszi aktívvá a kiválasztott PlayerSprite(oka)t, megfelelő helyen méretben aktív komponensekkel. Az UnLoadCharacter inaktívvá tesz egy PlayerSprite-ot a komponenseikkel. A PlayerCurrentPosition visszaadja az aktív PlayerSprite jelenlegi pozícióját.

Confirm.xaml és Confirm:

A Confirm ablak akkor nyitódik meg, ha Clear gombra kattintunk, az ablak 2 gombot tartalmaz és egy labelt, amin felirat van. Ha a yes gombra kattintunk, akkor az adatbázis Highscore táblájának tartalma törlődik. Ha a no gombra kattintunk vagy az ablak kikerül a fókuszból, akkor bezáródik (IsForeground függvény).

Mainwindow.xaml és Sliders.xaml:

Az ablak elementjeit egy GameWindow nevű rács tartalmazza. Az ablakok tartalmának a változtatása a rácsok láthatóságának változtatásával történik. A legtöbbet használt elementek a rácsok, címkék, a gombok és a csúszkák. Van 1 ButtonControlTemplat1 kulcsú minta, ami egyedivé teszi a gombok kinézetét. A Sliders.xaml nevű alapanyagtárban tároljuk a csúszkák mintáját.

MainWindow:

A főablak osztálya. Itt inicializáljuk az ablakot és a főbb játékrendszereket. Tartalmazza még a menü gomb eseményeit, billentyűgomb eseményeket (játékos vezérlés), program kilépésekor lefutó eseményt. Mellékes funkciója még az animálás, beállítások mentése és betöltése, és a fő játék ablak méretének, valamint minden gyerek elemének méretezése a saját maga átméreteződése szerint.

SoundManager:

A hangok és zenék lejátszásáért felel NAudio használatával. Segítségével akár több zenét és hangeffektet lehet egyszerre lejátszani, eltérő hangerővel. A zenék között sima váltást hoz létre. PlaySong metódussal lehet zenét, PlaySoundEffect metódussal hangeffektet lejátszani. Elég az mp3 típusú hangfájl nevét megadni, ha a megfelelő mappába helyeztük el őket (Sounds/Music vagy Sounds/Effects). A SoundManager használata előtt fontos meghívni az Init függvényt, amely előkészíti a hangkeverőt és memóriába tárazza a hangeffekteket. Működése a PewPewSoundMixeren alapul, melynek külön lehet állítani két fő hangerejét.

Tesztelés:

A tesztelés nagyrésze a játékkal való játszással történt. E mellett használtunk objektumokat pl: DebugSprite, elementek megjelenítését és Console-ra kiírt adatokat.

Egyed-kapcsolat diagram:

Highscore tábla:

id (INT): tárolja a teljesített eredmény azonosítóját, elsődleges kulcs

characterid (INT): tárolja annak a karakternek az azonosítóját, amivel az eredményt érték el, idegen kulcs

floorcount (INT): tárolja azt a számot, ami megadja melyik pályán ért véget az adott játékos játéka

uname (STRING): tárolja az eredményhez tartozó megadott játékosnevet (default: player1/player2)

score (INT): tárolja az elért eredmény értékét

Character tábla:

characterid: tárolja a karakterek azonosítóját, elsődleges kulcs

charactername: tárolja a karakterek nevét



Kapcsolat: characterid-n keresztül

Továbbfejlesztési lehetőségek:

* Nem lokális többjátékos mód megvalósítása a játékhoz
* Karakterek, pályák, ellenségek, zenék hozzáadása
* Adatok felvitele egy online adatbázisba, ami elérhető mindenki számára, így láthatják az eredményeiket minden játékoshoz viszonyítva
* Effektek lejátszásának fejlesztése
* Ellenség AI fejlesztése

Felhasználói dokumentáció:

Futtatáshoz szükséges környezet:

~Szoftver:

A futtatáshoz egy működő Windows operációs rendszerre és a fájlokat tartalmazó mappára van szükség, illetve DirectX minimum 9-es verziójú szoftverjére. Fontos, hogy a mappákban található fájlok ne legyenek eltávolítva/ elkülönítve eredeti helyükről, mert az a program helytelen működését eredményezheti

~Hardver:

* >800MHz processzor
* 512MBytes RAM
* DirectX 9 futtatására képes videókártya

A program Visual Studio 2019-ben lett fejlesztve, ezért továbbfejlesztéséhez ajánlott ennek megléte, illetve a Microsoft oldalán megadott hardverkövetelmények.

Program kezelése:

A program indítása az PewPew Paradise.exe fájllal történik. Ekkor a program betölti a játék ablakját. Az ablak méretezése kedv szerint történhet, de ha kikattintunk az ablakból akkor a karakterek irányítása megszűnhet. A fő menüben gombokat látunk, amin keresztül érhetjük el a játék egyes funkcióit. A „Singleplayer” gombra kattintva indíthatunk játékot, ha egyedül szeretnénk játszani. A gombra kattintás után megadhatja a nevét és kiválaszthatja karakterét, majd a „Play” gomb megnyomásával megkezdheti a játékot. A pálya felett láthatja a nevét a pontjait és hogy mennyi életerője van. A játékot az escape gombbal a billentyűzetén megállíthatja és folytathatja. A játéknak 2 célja van, a pontok gyűjtése és a túlélés. A pontokat akkor szerezhet, ha lelő egy ellenséget és összegyűjti az általuk dobott gyümölcsöt. Ha egy pályán minden ellenséget megölt, akkor egy új pályára fog érkezni, attól függetlenül, hogy összegyűjtötte-e a gyümölcsöket. Életerőt akkor veszít, ha egy ellenség hozzáér a karakteréhez. Ha minden életerője elveszik a játéknak vége és kiléphet a játékból („Exit”) vagy visszatérhet a fő menübe („Main Menu”). A „Multiplayer” gombra kattintva minden ugyan úgy működik, ahogy a „Singleplayer” alatt, azzal a kivétellel, hogy két karakter irányítása lehetséges. A játéknak itt akkor van vége, ha minden játékos életereje eléri a 0-t. Az „Options” gombra kattintva állíthatja a zene és az effektek hangerejét, illetve törölheti a ranglista tartalmát. A „Help” gombra kattintva találja a ponttáblát, illetve a karakterek irányítását. A kis kupa ikonra kattintva megtalálja a ranglistát. A feliratokra kattintva rendezheti az adatokat legtöbb pálya vagy legmagasabb pont szerint. Az „Exit” gombra kattintva léphet ki a programból.

Az összes funkció bemutatását videón keresztül egyszerűbbnek gondoltuk.

A bemutató link: <https://youtu.be/jdD2su64T7Y>

Külső források:

NuGet csomagok:

Dapper

Entity Framework

NAudio

Newtonsoft.Json

System.Data.SQLite

System.Security.AccessControl

System.Security.Principal.Windows

Microsoft.Win32.Registry