Témalabor Dokumentáció

Györgydeák Levente, NC1O2T

Architektúra

A szoftver a Model View Controller architektúrát követi. A megjelenítésre, és input érzékelésre a Win2D könyvtárat használtam. A vektorokra, mátrixokra, és néhány lineáris algebrai problémára a MathNet könyvtárat használtam.

Controller:

Itt történik az input feldolgozása, és ennek hatására a model változtatása.

Model:

Ez a réteg további 2 rétegre van bontva:

- **Logikai Model:** A játék mindentől független logikája itt található. Ez azt jelenti, hogy ha A 2D-re specifikus logikát és kinézetet le akarnánk cserélni 3D-sre, vagy esetleg egy táblás, körönkénti játékra, akkor ehhez nem kellene hozzányúlnunk, hiszen ez nem tartalmaz semmilyen referenciát bármely más rétegre
- **Model Fizikai Része:** Ez a réteg alapvetően a modell része, azonban olyan szempontból köze van a megjelenítéshez, hogy 2D-re specifikus megvalósítások találhatók itt. Az alapvető fizikával kapcsolatos interfészek, amik a játékhoz kellenek itt vannak megvalósítva pl.: ütközés-ellenőrzés és alapvető mozgatás. Mivel a játékhoz csak minimális fizikai logika kell a jelen állapotában, ezért ez a legkisebb réteg. Ez a réteg csak a modell-beli osztályokra tartalmaz referenciát, tehát ehhez sem szükséges hozzányúlni a View és/vagy Controller réteg lecserélése esetén.

View:

Itt található minden megjelenítésre specifikus információ. Ez a réteg az Observer mintához hasonlóan általában mindig a játék jelenlegi állapotát rajzolja ki, ezért ez a réteg a modell-től függ.

Eventek

A rétegek között a függetlenség megtartása érdekében sokszor eventekkel történik a kommunikáció. A két fő példa erre:

- A játékos lenyomja a támadásra szolgáló gombot. Ekkor a játék egyből nem hozhatja létre a támadás objektumot, mert előbb egy animáció játszódik le. Az animáció lejátszása végeztével egy event-en keresztül meghívódik a függvény, ami a modellben létrehozza a támadás objektumot.
- Amikor egy olyan objektum létrejön a modellben, amihez egy nézetnek is kell tartoznia (pl.: egy új tárgyat kap a játékos egy ellenség megöléséért), akkor ezt a nézetet létre kell hozni, anélkül, hogy a modell ismerné a nézetet.

Osztálystruktúra, Legfontosabb osztályok

Controller réteg

Controller.Controller osztály

Ez az osztály felelős a játék alapvető vezérléséért. Bonyolultabb logikát nem használ, inkább továbbdelegál más osztályoknak. A játék léptetéséhez szükséges függvények meghívása itt történik. Az osztály `Tick(double delta)` metódusa adja a játék folyamatos működését. Ez a függvény folyamatosan megívódik. A játék menete folyamán keletkező nézeteket ezen az osztályon keresztül kell hozzáadni a játéktérhez.

A controller rétegben történik a játék inicializálása. Ennek egy részére szolgál a `Controller.Initialization.DefaultGameConfigurer` nevű oszály, aminek a controller osztály delegál tovább.

Input feldolgozás

Az Input 2 (vagy 3) mapping-en megy keresztül, mire elér odáig, hogy végrehajtódjon az ahhoz tartozó kód.

- A 0. lépés az az input srtinggé alakítása. Erre nincs külön osztály, ez a `Controller.Input.InputHandler` osztályban megtörténik, amikor az input hatására meghívódik a szükséges függvény. Példa: A játékos megnyomja a “W” gombot. Ennek hatására meghívódik a `void InputHandler.KeyDown(object sender,KeyRoutedEventArgs e)` metódus, ami paraméterben tartalmazza ezt az információt, de nem stringként. A függvényen belül ezt az inputot a “W” stringgé alakítjuk, majd tovább adjuk.
- Ez után a `RawInputProcessedInputMapper` egy általánosabb stringek gyűjteményévé alakítja az `IEnumerable<string> toProcessedInput(string input)` metódus segítségével. Ezt nevezem processed input-nak. Több inputhoz tartozhat ugyanaz a string, és egy inputhoz tartozhat több string is. Példa: A “W” stringből, és az “Up” stringből is “move-forward” string lesz, a “MouseLeft” stringből pedig lesz “initiate-interaction” és “light-attack” string.
- A végső lépés pedig ezeket az általánosabb stringeket egy végrehajtandó kódhoz rendelni. Ezt a `ProcessedInputActionMapper` teszi meg az

`IEnumerable<IInputAction> ToAction(IEnumerable<string> keys)` metódusával. A paraméterbe kapott összes stringet egy előre definiált szabály alapján egy `IInputAction` objektumhoz rendeli.

Controller.Interfaces.IInputAction interfész

Az interfésznek mindössze 3 alapértelmezetten üresre definiált metódusa van:

```
void OnPressed() { } //Az input lenyomásakor
void OnReleased() { } //Folyamatosan, amíg az input le van nyomva
void OnHold() { } //Az input felengedésekor
```

Ezek a függvények hívódnak meg a hozzá rendelt input gomb megfelelő állapotaiban. Új inputra történő viselkedés felvételekor nincs más dolgunk, mint egy új megvalósítást létrheozni, majd ezt a `ProcessedInputActionMapper.Builder` osztály `Builder AddMapping(string name, IInputAction action)` metódusával a létrehozás előtt hozzárendelni egy string-hez. Lehet builder nélkül is lérehozni ilyen mappelést, ekkor csak a `ProcessedInputActionMapper(Dictionary<string, IInputAction> inputActionMap)` konstruktort kell használni.

Controller.UnitControl.PlayerControl osztály

Ez az osztály azért felelős, hogy a játékos olyan inputjait, amikkel az avatárt akarja irányítani (és nem pl. az egyik ablakban csinálni valamit) megfelelően delegálja a modellnek, illetve a hozzá tartozó nézeteket létrehozza.

Példa: A játékos lenyomja a bal egérgombot. Az input átmegy a korábban leírt feldolgozáson, majd a hozzárendelt IInputAction egy megvalósítása meghívja a `PlayerControl.LightAttack()` metódust. Ennek hatására Elindul A támadás animációja, és az animációhoz egy event segítségével hozzákapcsoljuk a megfelelő támadás objektum létrehozását a modellben.

Controller.Window.WindowControl osztály

Ez az osztály a CustomWindow ősosztály leszármazottait kezeli. A CustomWindow osztálynak a leírása a View rész osztályainak leírásai között szerepel. Minden ablak objektumhoz csatol egy stringet, amin keresztül el tudjuk érni, és ezáltal meg tudjuk nyitni, vagy be tudjuk zárni.

Model réteg

Először a fizikai logika fontosabb osztályait, majd a többit mutatom be

Physics.TwoDimensional.Collision.IShape2D interfész

Egy 2D-s alakzattal kapcsolatos viselkedést modellez. Leginkább ütközésellenőrzéshez használt, de akár (főleg debuggoláskor) megjelenítéshez is használható (ShapeView osztály). Tartalmaz egy GameObject típusú referenciát a birtokos objektumra, ugyanis amikor valamivel ütközik, akkor ennek az objektumnak szól erről.

Physics.TwoDimensional.Collision.CollisionNotifier osztály

Sok IShape2D objektumot vezérel. A `void NotifyCollisions(double delta)` metódusa minden játékciklusban meghívódik. Ez a metódus felel azért, hogy ha 2 IShape2D ütközik, akkor értesítse őket, és a birtokos objektumukat.

Physics.TwoDimensional.Movement.MovementManager2D osztály

A játékobjektumok 2D mozgásának a megvalósítása ebben az osztályban történik. A pozícióra, és a mozgásra az `IPositionUnit` és `IMovementUnit` interfészeket megvalósító `PositionUnit2D` és `MovementUnit2D` használatos.

Model.Tickable.GameObject osztály

Majdnem minden játékban szereplő objektum ebből az absztrakt osztályból származik le. Nagyon magas absztrakciós szinten ír le egy játékbeli objektumot. A megvalósított interfészek: `ITickable, ISeparable, IExisting, ICanQueueAction`. Tehát minden játékbeli objektum

- tud reagálni a `Tick(double)` eseményre,
- szétválasztható a `void Separate(Dictionary<string, List<ISeparable>> dict)` metódussal úgy, hogy olyan kulcsú listába teszi magát, amilyet a leszármazott definiál,
- létezése valamilyen értelemben definiált a `bool Exists { get; }` absztrakt propertyvel
- képes elvégzendő műveleteket végrehajtási sorba tenni a `void QueueAction(Action action)` metódussal, azaz amikor az ő köre jön (`OnTick`, de akár máskor is), akkor végrehajtja ezeket.
- képes interakcióba lépni egy `Hero` objektummal a `void InteractWith(Hero interactionStarter)` metódussal

Model.Tickable.FightingEntity.Unit osztály

Absztrakt osztály, egy egységet ír le. Egy egység tud mozogni, támadni, támadást elszenvedni, képességet használni, és effekteket lehet rájuk tenni. Jelenleg 2 leszármazottja van: `Hero` és `Enemy`.

Model.Tickable.FightingEntity.Hero osztály

A játékos által vezérelt hős. A `Unit` osztályból származik. Az ősosztálya funkcionalitását azzal bővíti ki, hogy képes tapasztalati pontot és aranyat gyűjteni, illetve ezeket elkölteni képességekre, illetve tárgyakra. Ezen túl az `ICollector` interfészt is implementálja, tehát képes `ICollectible`-öket gyűjteni a `void Collect(ICollectible collectible)` metódusával.

Model.Tickable.FightingEntity.Enemy osztály

A gép által vezérelt játékos számára ellenséges. A `Unit` osztályból származik. A `Target` mező az éppen általa célpontba vett `Unit` típusú egységet jelenti. Az ellenség a célpontját követi, de ez a viselkedés felüldefiniálható a `IMovementStrategy movementStrategy` tagváltozó átállításával. A támadás módja is felüldefiniálható a `Dictionary<string, AttackBuilder> attacks` megfelelő kulcsú elemének az átkonfigurálásával. Ezek miatt ennek az osztálynak nincsenek leszármazottai, mert a viselkedés különböző elemei kicserélehtőek dinamikusan, így tetszőleges ellenséget létre lehet hozni. Persze, létre lehet honi leszármazott osztályt, ha olyat akarunk módosítani, amit egyébként nem lehetne.

Model.Attribute namespace

Ebben a namespace-ben szerepelnek a(z) (általában egy `Unit`-hoz tartozó) különböző attribútumokat kezelő osztályok. Ilyen például a `Health` osztály, ami egy életerő vezérléséért felelős. Ezek az osztályok általában primitív vagy beépített típusokkal dolgoznak, és nincsenek függőségeik. Kivétel az Inventory, ami kifejezetten `Item` tipusú objektumokkal dolgozik.

Model.Tickable.Item.Item osztály

Absztrakt osztály, egy tárgyat definiál. A tárgy egy olyan `GameObject`, ami bónuszokat ad viselőjének.

Model.Tickable.Item.Weapon.Weapon osztály

Egy fegyvert ír le. Az `Item` osztályból származik. A működését azzal bővíti ki, hogy képes támadást idézni. Mivel különböző fegyverek csak abban térnek el egymástól modell szinten, hogy más támadást hoznak létre, ezért ennek az osztálynak nincsenek leszármazottai.

Model.Tickable.Attacks.Attack osztály

Egy támadást reprezentáló osztály. Ilyen például a kard által létrehozott suhintás, ami megsebzi az ellenségeket, de ilyen egy nyílvessző is, amit az íj lő ki. Egy támadás viselkedése több szempontból is konfigurálható, például létrehozáskor az `IMovementStrategy movementStrategy` paraméter megadásával meg lehet határozni, hogy mi szerint mozogjon, vagy az `IAttackStrategy attackStrategy` paraméterrel, hogy hogyan támadjon. Mivel egy támadás ennyire konfigurálható, ezért nincsenek leszármazottai az `Attack` osztálynak

View réteg

View.Interfaces.IAnimation interfész

Metódus nincs bene, egy eventje van: `event Action<T> OnAnimationCompleted`. Ez az event akkor hívódik meg, amikor egy ezt megvalósító animáció a végéhez ér.

View.Interfaces.TransformationAnimation2D osztály

Absztrakt osztály, implementálja az `IAnimation<TransformationAnimation2D>` interfészt. Ez az osztály felelős az idő (és egyebek) függvényében 2 dimenziós 3x2-es mátrixokkal

leírható transzformációkért. Az egyetlen absztrakt metódusa a `Matrix3x2 OnGetImage(DrawingArgs animationArgs)`. A leszármazottnak meg kell mondania, hogy milyen mátrix-szal legyen transzformálva egy adott időpillanatban az őt használó kép. Ez nem csak az időpillanattól függhet, hanem az egér pozíciójától, vagy hogy jelenleg a képernyőn hol helyezkedik el az animált objektum. Ezeket mind tartalmazza a `DrawingArgs animationArgs` paraméter

View.Animation.ImageSequence.ImageSequenceAnimation osztály

Implementálja az `IAnimation<TransformationAnimation2D>` interfészt. Ennek az osztálynak a felelőssége megmondani az idő függvényében, hogy éppen melyik a kirajzolandó kép. Ezt a `ICanvasImage CalculateImage(double delta)` metódusával teszi.

View.Animation.Animators.Animator oszály

A fentiek alapján nagyon sok animáció leírható a 2 kombinációja által. Ezt foglalja egybe az Animator absztrakt osztály. El lehet indítani, illetve be lehet fejezni külön-külön akár transzformáció, akár képsorozat animációkat. A fő funkcionalitás a `void Animate(DrawingArgs animationArgs)` metódus, ugyanis itt történik a valódi kirajzolás, miután az animáció objektumok segítségével összerakott egy képet, amit ki kell rajzolni. Ez a függvény a lehető leggyorsabban folyamatosan meghívódik.

View.EntityView.IDrawable

Interfész, ami egy kirajzolható objektum közös metódusait tartalmazza. Egy ezt megvalósító osztály a `void OnRender(DrawingArgs drawingArgs)` metódusban rajzolja ki magát.

View.EntityView.UnitView oszály

Megvalósítja az IDrawable interfészt. Egy `Unit` objektum kirajzolásáért felelős. Az observer mintát használva mindig a jelenlegi állapotát rajzolja ki az objektumnak.

View.EntityView.AttackView oszály

Megvalósítja az IDrawable interfészt. Egy `Attack` objektum kirajzolásáért felelős. Az observer mintát használva mindig a jelenlegi állapotát rajzolja ki az objektumnak.

View.EntityView.ShapeView oszály

Megvalósítja az IDrawable interfészt. Egy `IShape2D` objektum kirajzolásáért felelős. Az observer mintát használva mindig a jelenlegi állapotát rajzolja ki az objektumnak.

View.EntityView.BaseItemView oszály

Megvalósítja az IDrawable interfészt. Absztrakt osztály. Leszármazottai `Item` objektum kirajzolásáért felelősek. Az observer mintát használva mindig a jelenlegi állapotát rajzolja ki az objektumnak.

View.EntityView.EquippedWeaponView oszály

A `BaseItemView` osztályból származik, és egy `Weapon` objektum pályára való kirajzolásáért felelős. Ez azt jelenti, hogy ha a játékos felszerel az avatárjának egy kardot, akkor az a kard a pályán megjelenik. Ennek a kardnak a kirajzolását végzi ez az osztály

View.EntityView.InventoryItemView oszály

A `BaseItemView` osztályból származik, és egy `Item` objektum ikonjának kirajzolásááért felelős. Amikor a játékos leltárában van egy tárgy, akkor annak az ikonjának a kirajzolását ez az osztály végzi.

View.Image.DrawingImage oszály

Olyan megvalósítása az IDrawable interfésznek, ami csak egy képet rajzol ki, és nem használ `Animator`-t. Ez lehetővé teszi, hogy egyszerű képeket lehessen kirajzolni önmagukban, és nem kell, hogy ezek egy animáció részei legyenek.

View.UIElements namespace

Ebben a namespaceben szepelnek a felhasználói felület megjelenítéséért felelős osztályok. Ezek WPF-et használnak, tehát minden UI leírása XAML formátumban szerepel.

View.UIElements.CustomWindow osztály

Ez az osztály az összes játékbeli ablak ősosztálya. Az ablakok egérgomb lenyomva tartásával való elhúzása itt van megvalósítva, ezt minden leszármazott megörökli.

View.UIElements.Hud osztály

A képernyő alján látható HUD megjelenítéséért felelős.