EVA beadandó – 3.

Ezen dokumentum a 2024/25/1 félév Esemény-vezérelt Alkalmazások EA+GY tárgy 3. beadandójának dokumentációja, melyet készített Kmeczó András (NEPTUN kód: ROXFTL). A feladat - melynek leírása lejjebb található – a .Net, azon belül a Avalonia UI keretrendszerben készítendő el.

**Feladatkiírás: (másolva)**

1. Aknamező

Készítsünk programot a következő játékra.

A játékban egy tengeralattjárót kell irányítanunk a képernyőn (balra, jobbra, fel, illetve le), amely felett ellenséges hajók köröznek, és folyamatosan aknákat dobnak a tengerbe. Az aknáknak három típusa van (könnyű, közepes, nehéz), amely meghatározza, hogy milyen gyorsan süllyednek a vízben (minél nehezebb, annál gyorsabban).

Az aknákat véletlenszerűen dobják a tengerbe, ám mivel a hajóskapitányok egyre türelmetlenebbek, egyre gyorsabban kerül egyre több akna a vízbe. A játékos célja az, hogy minél tovább elkerülje az aknákat. A játék addig tart, ameddig a tengeralattjárót el nem találta egy akna.

A program biztosítson lehetőséget új játék kezdésére, valamint játék szüneteltetésére (ekkor nem telik az idő, és nem mozog semmi a játékban). Ismerje fel, ha vége a játéknak, és jelenítse meg, mennyi volt a játékidő. Ezen felül szüneteltetés alatt legyen lehetőség a játék elmentésére, valamint betöltésére.

**Feladat elemzése:**

Az elkészítendő feladat a következőképpen kerüljön megvalósításra: a felhasználót egy főmenü fogadja, ahol új játékot lehet indítani, mentett játékot betölteni, illetve kilépni az alkalmazásból. Játék indításakor egy tengeralattjáró kerül a képernyőre, melyet a játékos a WASD billentyűkkel irányíthat. A játékban bizonyos időközönként (5 másodperc) egy véletlenszerűen generált típusú és pozíciójú akna kerül lehelyezésre, mely a képernyő tetejéről adott sebességgel a képernyő alja felé esik. A játék oldalnézetes 2 dimenziós grafikával van megvalósítva, valós idejű frissítéssel. A játék akkor ér véget, ha kilépünk belőle, vagy a tengeralattjáró ütközik egy aknával, elveszítve az aktuális játékot.

**Felhasználói esetdiagram (csatolva megtalálható JPG-ként):**

A képen diagram, szöveg, vázlat, rajz látható

Automatikusan generált leírás

**UML-diagram: csatolva megtalálható SVG formátumban**

**UML-diagramok magyarázata:**

Perzisztencia:

* A perzisztencia szint az operációs rendszertől függő dolgokat valósítja meg (I/O, fájlból olvasás és fájlba írás)
* Ezen szint jelen modellünkben két interfészt definiál, melyen keresztül a model szint használni tudja a perzisztencia szint funkcionalitását
  + ISerializer: ez egy olyan objektum, mely a generikus paraméter (T) szöveggé konvertálását és szövegból beolvasását teszi lehetővé két metóduson keresztül
  + ISerializable: egy metódusa van, mely visszaadja az adott típushoz tartozó Serializer objektumot
* Ezen kívül definiálunk egy IFileManager interfészt, mely a mentés és betöltés függvényeket definiálja, illetve egy tényleges implementáció GameFileManager néven, mely GameState objektumokat tud kezelni

Model:

* A model szint a perzisztencára épül, és a játékban található objektumokat definiálja, valamint minden olyat, ami nem közvetlenül a megjelenítéshez kapcsolódik
* Tartalamaz egy interfészt az idő kezelésére (ITimer)
* A fő adatstrukrúra a GameState, mely tartalmazza a játék teljes állapotát. Implementálja az ISerializable interfészt, és implementációjában visszaad egy GameStateSerializert, mely JSON formátumúra konvertálja a GameState aktuális állapotát.
* A Submarine a játékos által irányított tegneralattjáró modellje. Mérete és poziciója van. A Move(dir, float) művelet a mozgatást implementálja, a MoveDirection enum segítségével adjuk meg a mozgatás irányát, a delta paraméter pedig (minden más helyen is) a legutóbbi mozgatás óta eltelt időt jelzi, így a mozgás sebessége nem fog függeni az aktuális képfrissítési rátától.
* A Mine egy absztrakt osztály, mely a különböző aknák funkcionalitását írja le. Leszármazott osztályainak implementálniuk kell a MovementSpeed, illetve Size változókat, melyek nehézségtől függnek, illetve a be- és kiolvasás megkönnyítésére található még benne egy MineType nevezetű enum, ám ezt a játékhoz nem használjuk fel, így nem szerepel a diagramon, implementációs részlet csupán.
* Az akna mozgatásának függvénye mindössze a mozgási sebességnek megfelelően növeli az akna Y koordinátáját
* Mindezt összefogja a GameManager osztály, melynek feladata a játék elemeit létrehozni, menedzselni, és frissíteni, valamint a megfelelő eseményeket a nézet szint számára elérhetővé tenni.
* Az Update függvény menete:
  + növeljük az elapsedSeconds változót
  + mozgatjuk a tengeralattjárót, ha kell
  + ha elérte az elapsedSeconds az 5 másodpercet, akkor lenullázzuk, és egy véletlenszerű aknát lerakunk
  + ezután minden aknát mozgatunk, és megnézzük, ütköztek-e a tengeralattjáróval
  + ha igen, akkor kiküldjük az OnGameEnded eseményt, egyébként pedig az OnUpdate esemény kerül meghívásra

ViewModel:

* A ViewModel feladata a Modelben található objektumok és a View közötti kapcsolat megteremtése
* Található benne:
  + ViewModelBase: a ViewModel elemei számára biztosít egy újrafelhasználható alapot, mely a különböző megfigyelt Property változásairól tájékoztatja a feliratkozott osztályokat
  + IDisplayable: az App layer fogja implementálni, ennek a segítségével tudjuk a Model által használt elemek View-beli megjelenítését frissíteni
  + ViewState: a gombok állapotát befolyásolja
  + DelegateCommand: egy lefuttatható parancsot foglal magában, feltétellel együtt
  + MineViewModel: a Mine model és view réteg közötti kapcsolatát valósítja meg
  + SubmarineViewModel: ugyanez a Submarine osztály számára
  + MinefieldGameViewModel: a játék elemeit foglalj magában, és a view által meghívható parancsokkal interakcióba lép a Model-el, majd eventek formájában adja vissza a View számára a változásokat

App - View:

* A View szint fő eleme a MainWindow, mely egy Canvas elemet tartalmaz, aminek segítségével megjeleníthetőek a játék elemei tetszőleges helyen és méretben
* Az App osztály tartalmazza a játék magját, és előkészíti a játék elemeinek megjelenítését, valamint a ViewModel parancsainak hívását, és az eseményeinek lekezelését tartalmazza
* Tartalmaz továbbá egy ITimer implementációt (GameTimer)

**Tesztelés:**

A játék funkcionalitását a model szintjén egységtesztekkel biztosítjuk. Az egységtesztek a következőképpen vannak felosztva:

* Point2DTests: A Modelben bevezetésre került egy Point2D osztály, mely konvertálható Point és Size típusokra, ennek a működését teszteljük
* MineTests: Teszteljük a három különböző nehézségi szint létrehozását, és mozgását
* SubmarineTests: Teszteljük a tengeralattjáró létrehozását és mozgását
* GameStateSerializerTests: Teszteljük a játék ki és beolvasásához szükséges JSON átalakítás működését
* GameManagerTests: Teszteljük új játék létrehozását, illetve akna és tengeralattjáró ütközését