PIRATE FRENZY

Fejlesztői Dokumentáció

1. Megjelenítési réteg

Az a réteg két részre bontható. Az első a menüelemek megjelenítéséért, míg a második a játéktér kirajzolásáért felelős.

A menüben használt osztályok majdnem egésze saját implementáció. Minden menüelem őse a „UIElement” absztrakt osztály, „IDrawable”. Egy ilyen osztályban gyerek UIElement osztályok is helyezhetőek. Ez a megadott adatokból (pozíció, méretek, ecset) egy Drawing-ot hoz létre, amit a „Draw(DrawingGroup)” metódusában hozzáad a kapott DrawingGroup-hoz, majd meghívja a gyerek elemek Draw metódusait is, a kapott DrawingGrouppal.

A „UIPage”-ek inicializálják a rajtuk megjelenő menüelemeket.

Ezekhez a UI elemekhez tartoznak interakciós interfészek: IPressable, IClickable, IDraggable.

Az ezen interfészek által definiált metódusokat a „MouseEventTrigger” hívja meg.

Ez egérgomb lenyomás hatására végigfut az aktuális UIPage összes menüelemén. Ha talál olyat, ami IPressable és az egér pont fölötte van, meghívja annak az OnPressed() metódusát. Amennyiben ez igaz értékkel tér vissza, az egéreseményt lekezeltnek veszi, a menüelemet, mint aktív elem elmenti, és befejezi a keresgélést.

A többi egéreseménynél (elengedés, mozgatás) már csak az aktív elemet vizsgálja. Ha le tudja kezelni az adott interakciót, akkor arra lehetőséget kap. (Az OnClick() csak akkor hívódik meg, ha az aktív elem felett lett elengedve az egérgomb.)

Magát a játékteret a „GameRenderer” rajzolja ki. Ebben minden játékobjektumra található kirajzoló metódus. A vásznat (egy DrawingGroup) úgy állítja be, hogy a játékos a képernyő közepén legyen, és felfelé nézzen. Így a render metódusok minden játékobjektumot a bennük tárolt pozícióra rajzoljnak ki.

Logikailag minden osztály felett áll a „GameScreen” avagy game controller. Ez ütemezi a GameLogic frissítéseit, és ez hívja meg a render osztályok Render() metódusait.

A hanghatások lejátszását a Mark Helath által készített „NAudio” nuget csomag segíti.

2. A logika

Ugyan a logika központi eleme a „GameLogic” osztály, a logika legnagyobb része mégsem itt található.

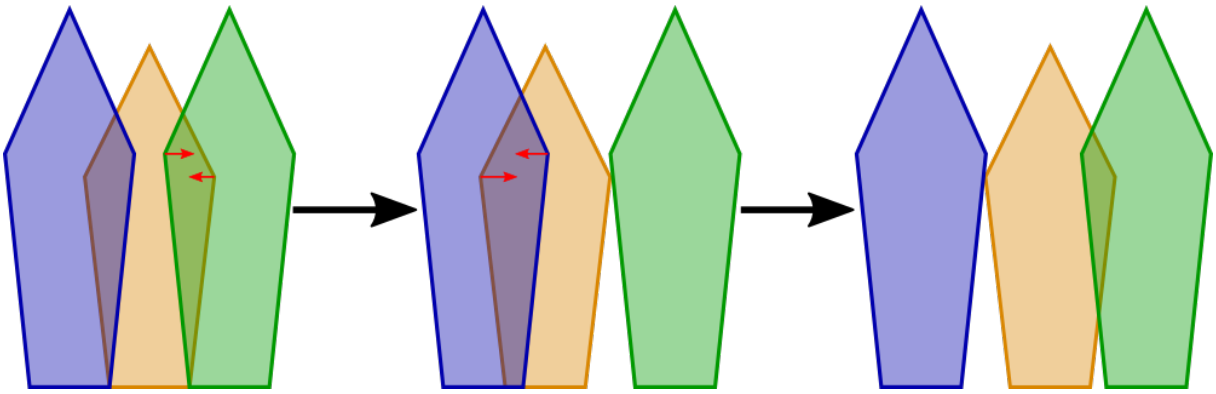
A logika Unity-s mintára tovább van delegálva a játékobjektumok felé.

A GameLogic fő metódusa a „void Update(delta)”. Ez frissíti a játékteret. Ehhez azonban meghívja a játékobjektumok „void Update(delta)” metódusait, hogy frissítsék magukat (lásd szekvencia diagram). Ezen játékobjektumok közül is sokan átadják a feladatot a rájuk csatolt IAI osztálynak. Ilyenek például a hadihajók (meg az összes többi). Őket „EscortGuardian” AI osztályok irányítják. Ennek a megoldásnak előnye, hogy több féle játék objektum is tud azonosan viselkedni, kód duplikálás nélkül.

A játékobjektumok őse az absztrakt „GameObject” osztály. Ebben vannak implementálva a transzformációk (TransformVector/Direction/Point, InverseTransformVector/Direction/Point).

Ennek leszármazottja az ugyan csak absztrakt „CollidingGameObject”. Ennek ICollider adattagjai vannak (egy komplett lista). Ez definiálja az „InBounds”, „HitByRay” és a „virtual OnCollision” metódusokat.

A logikának azon része, melyet tényleg a GameLogic végez, az az ütközés kezelés. Minden CollidingGameObject párt megvizsgál. Ha ütköznek, elmozdítja őket, hogy ne fedjék egymást. Ha egy objektumot két másik objektum szorít össze, az átfedés nem szűnik meg teljesen (minden párt csak egyszer vizsgál meg).



Ezért a logika többször is (2x) végigfut az összes objektumon.

3. Multiplayer

A játék a többjátékos módot is támogatja. Az adatátvitel egy adatbázissal valósul meg. Minden 0,4 másodpercentként feltöltődnek a lokális játékos adatai, és letöltődnek a többi játékos adatai. Az adatbázist nem közvetlenül a C# kód, hanem php szkriptek kezelik, melyek a webszerveren találhatóak (www.danielpolish.hu) (lásd komponens diagram). Ennek előnye, hogy az adatbázis felhasználó belépési adatai nem kerülnek bele a publikált termékbe.

A többjátékos mód menedzselését a „MultiplayerManager” végzi. Ez ütemezi be a fel-, illetve letöltéseket, ez hozza létre, és ez frissíti a távoli játékosokat reprezentáló hajókat.

Ezekre a hajókra ún. „RemotePlayer” AI-k vannak csatolva. Ezek az adatbázisból származó adatok alapján mozgatják a hajókat (pozíció, sebesség, irány, forgási sebesség). Mivel a késleltetés ~0,2s, ezért nem abba a pontba mozgatják a hajókat, amit kaptak, hanem egy kicsit előrébb. A RemotePlayer feladata továbbá a parancsok végrehajtása. Egy ilyen parancs van: *A* hajó lőtt, és eltalálta *B*-t (ha eltalálta).

A MultiplayerManager Update() metódsát is a logika hívja meg, még mielőtt végigmenne a játékobjektumokon (lásd szekvencia diagram).

4. Ütközés kezelés

A játék teljesen egyedi ütközés kezelést használ. A matek a „CollisionLib” nevű F# projektben van implementálva. E köré készült a „CollisionLibWrapper” projekt, hogy kényelmesebb legyen a használata.

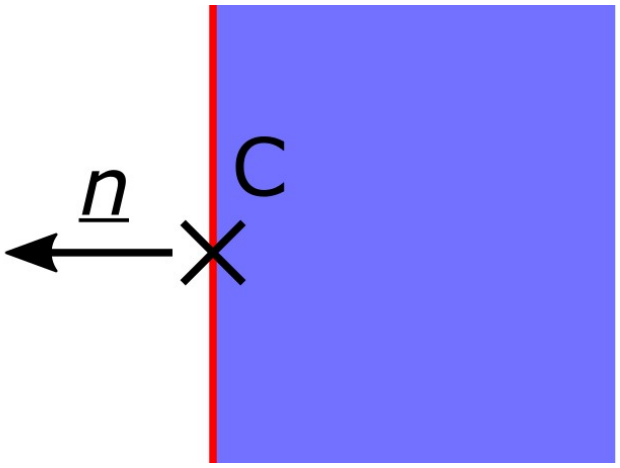
4.1. Ütközés detektálás

• A határütköző

Mindennek az alapja az úgynevezett „határütköző” (boundary collider).

Az F# kódban struktúraként jelenik meg. Mindössze egy középpontból és egy normálvektorból áll.

Ezt úgy kell elképzelni, mint egy egyenest, ami a síkot két részre osztja. Az a síkrész van „előtte”, amely a középponttól a normálvektor irányában van. Egy pont akkor ütközik, ha az az egyenes előtti síkrészen vagy pont az egyenesen található.



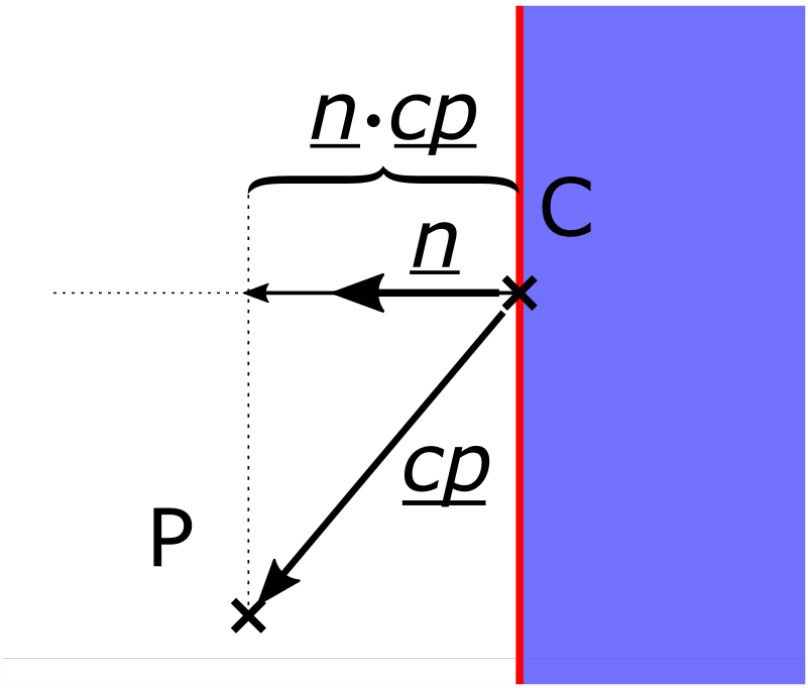
Ahhoz, hogy ennek a működését megértsük, összvissz egy képletet kell ismernünk. Ez a vektor levetítése egy másik vektorra.

$$\boldsymbol{p} = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{|\boldsymbol{b}|^2} * \boldsymbol{b}$$

Ahol ***p*** az ***a*** vektor levetítve ***b***-re. Ha ***b*** egységnyi hosszúságú, a képlet leegyszerűsíthető:

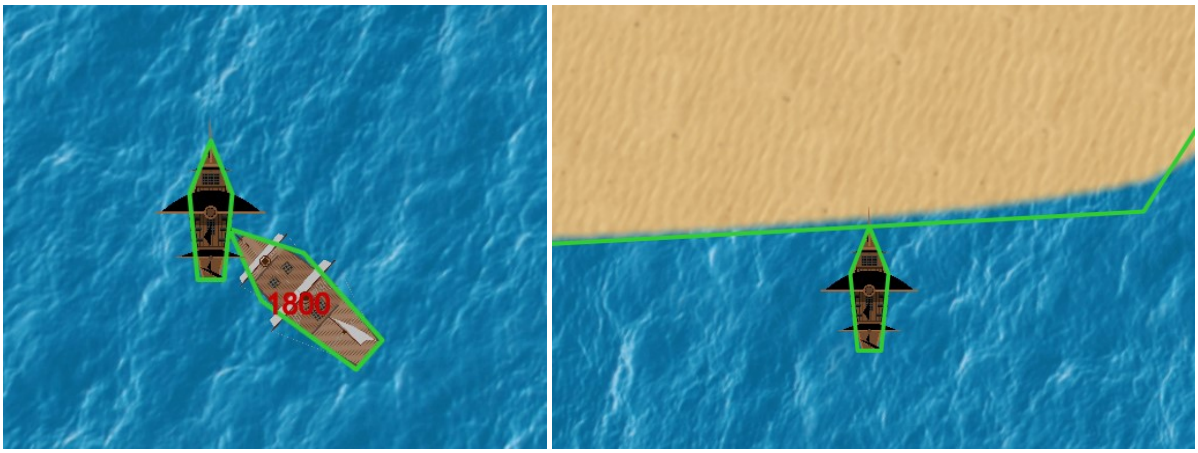
$$\boldsymbol{p} = (\boldsymbol{a} \cdot \boldsymbol{b}) * \boldsymbol{b}$$

Nekünk igazából nem egy vektor vetülete kell, hanem csak a szorzó ($\boldsymbol{a} \cdot \boldsymbol{b}$), sőt annak is csak az előjele. Ugyanis ha ezen szorzó nem negatív, akkor a pont vagy az egyenesen, vagy az egyenes előtti síkrészen van. Tehát egy P pontosan pont akkor „ütközik” a határütközővel, ha $(P - C) \cdot \boldsymbol{n}$ nem negatív.



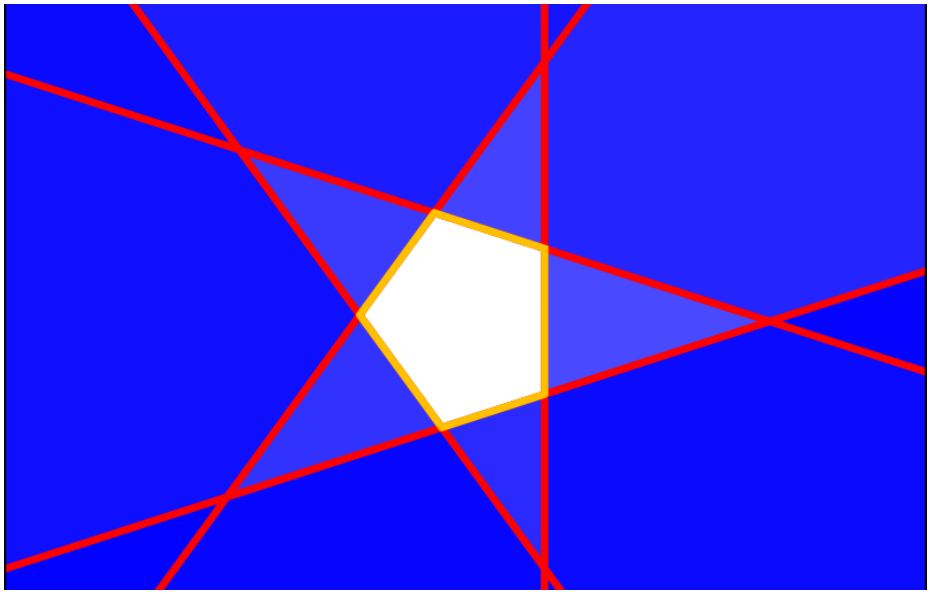
- A sokszög-ütköző

Az ütközők királya a sokszög-ütköző (polygon colllider).

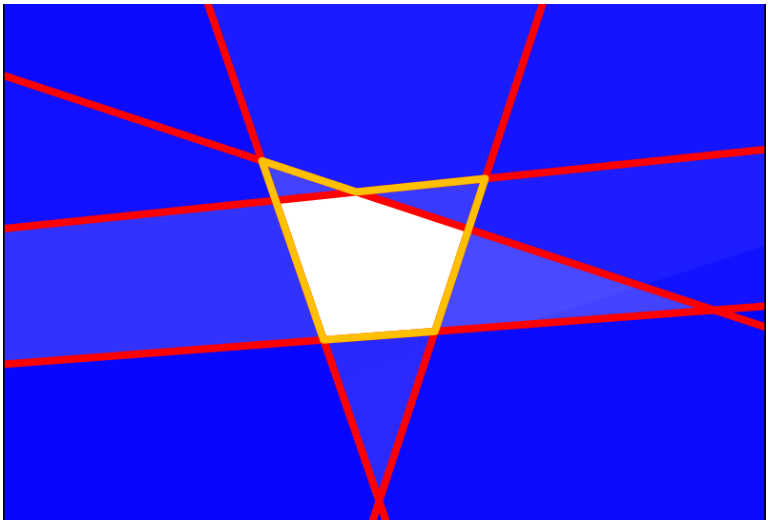


Egy sokszög-ütköző nem más, mint több határütköző összeéselve.

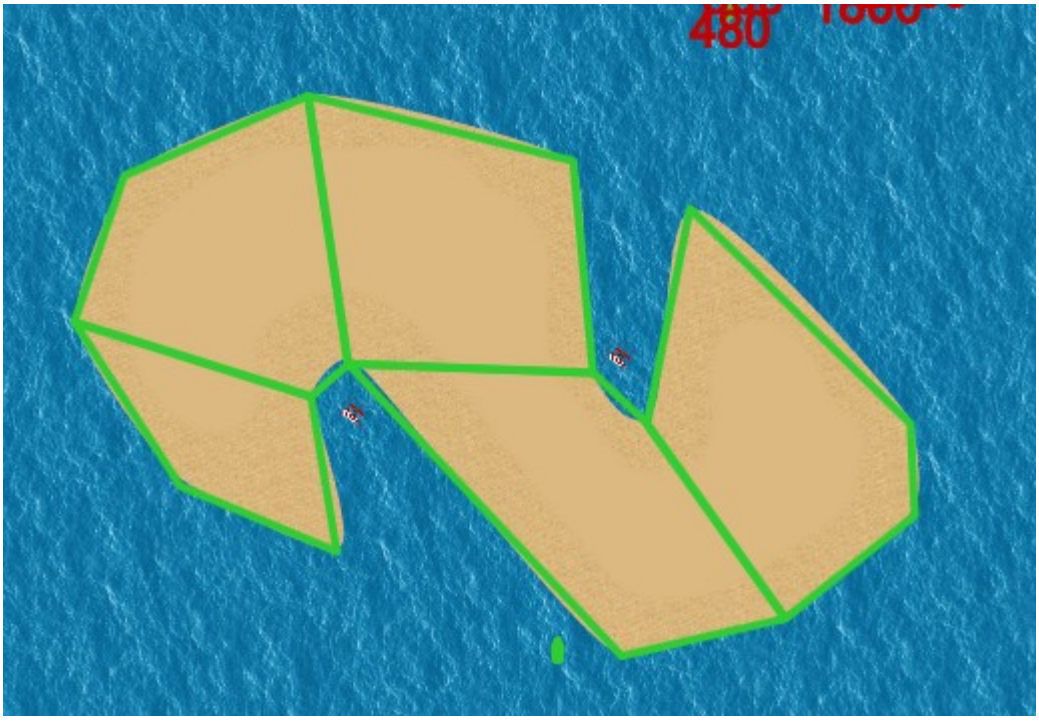
A háttérben annyi történik, hogy oldalanként létrejön egy-egy határütköző. Egy pont akkor ütközik a sokszög-ütközővel, ha az összes határütközőjével ütközik. A szemléltető ábrán látható, hogy pontosan a sokszög határán és belsejében lévő pontok ilyenek (ami fehéren maradt).



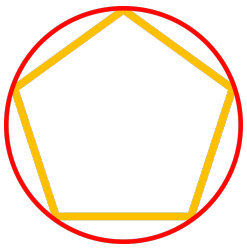
Ebből a képletből azonban az is adódik, hogy konkáv sokszögeken nem működik. A végeredmény mindig konvex lesz. Az ábrán egy konkáv ötszöget alakítottam ki (sárga), de az ütközés itt is csak a fehér, konvex ötszögben lesz detektálva.



Emiatt a játékban található szigetek több konvex ütközőt tartalmaznak.



Ahhoz, hogy az ütközésdetektálás még gyorsabban történjen, a sokszög-ütköző egy körütközőt is létrehoz, amely teljesen lefedi a sokszöget. Ha a pont nem ütközik a körütközővel, a sokszöggel sem ütközik.



- A körütköző

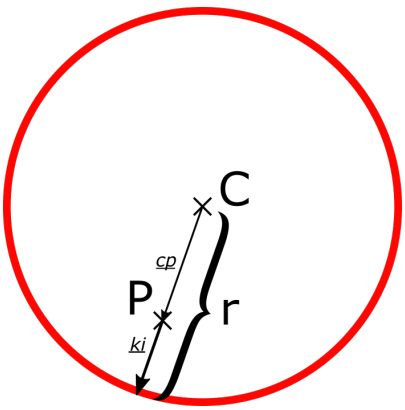
A körütköző az ütközők legegyszerűbbike. Egy pont akkor ütközik vele, ha az közelebb van középpontjához, mint a sugarának a hossza.

4.2. A kiút.

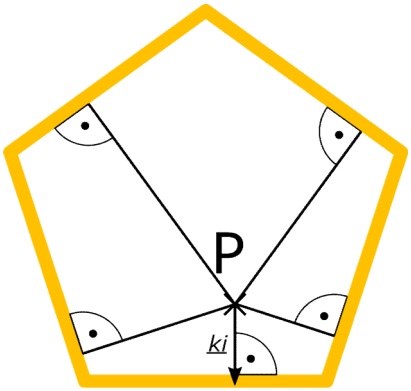
Most már minden pontról pontosan meg lehet mondani, hogy ütközik-e az adott ütközővel, vagy nem. Az ütközés kezelésnek ez azonban még csak a fele. Egy pontról nem csak azt kell tudnunk, hogy ütközik-e, hanem azt is, hogy ha ütközik, akkor melyik az ütközőből kivezető legrövidebb út (eltolási vektor).

Körütköző esetén könnyű dolgunk van. A ***ki*** vektor sugárirányú, hossza a sugár hosszának és a ***cp*** vektor hosszának különbsége.

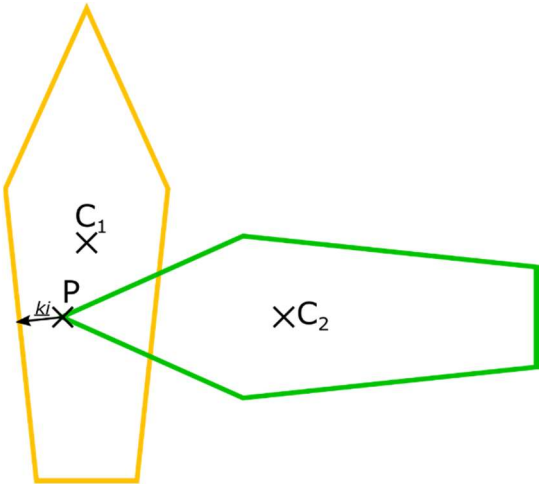
$$\boldsymbol{ki} = \frac{\boldsymbol{cp}}{|\boldsymbol{cp}|} * (r - |\boldsymbol{cp}|)$$



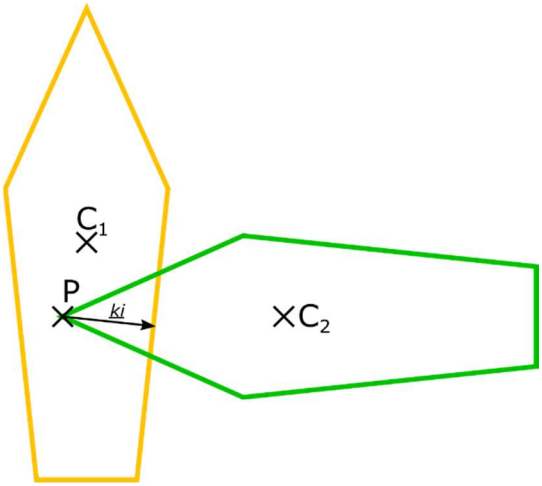
Sokszög-ütköző esetén a ki kell számolni a P pont minden oldaltól mért távolságát. A ***ki*** vektor merőleges a P-hez legközelebbi oldalra. Hossza a P távolsága a legközelebbi oldaltól.



Ez azonban csak akkor igaz, ha egy egyedülálló pontot akarunk kimozgatni. Amennyiben az a pont egy másik sokszög-ütköző egy csúcsa, akkor a következő szituációba futhatunk bele:



A ***ki*** vektor a P pontot tényleg kimozgatja az ütközőből, de ezzel a két ütköző csak jobban fedésbe kerül. A fenti módszert úgy kell módosítani, hogy csak az lehessen a ***ki*** vektor, amely a $\boldsymbol{pc_2}$ vektorral legfeljebb 90°-ot zár be ($\boldsymbol{ki} \cdot \boldsymbol{pc_2} \geq 0$).

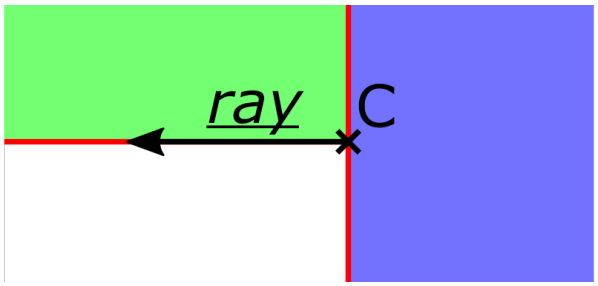


4.3. Raycast

Az ütközés detektálásnak egy másik aspektusa is van, a raycast. Ebben a játékban ezen van a legnagyobb hangsúly, hiszen ez a lövöldözés alapja. Itt nem pontokkal, hanem egy sugárral kell ütközni. A sugárnak három összetevője van: kiinduló pont, irány és maximális távolság.

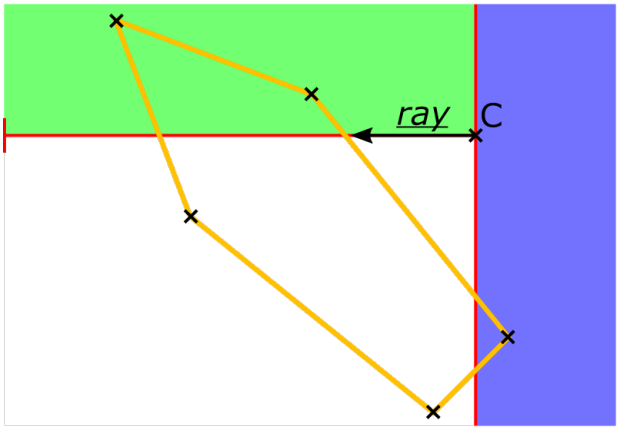
Biztos vagyok benne, hogy az általam alkotott módszer az egyik leghatékonyabb.

A sugár két határütközőre vezethető vissza. Az egyik határütköző a sugár terjedési irányába „néz”, a másik arra merőlegesen.

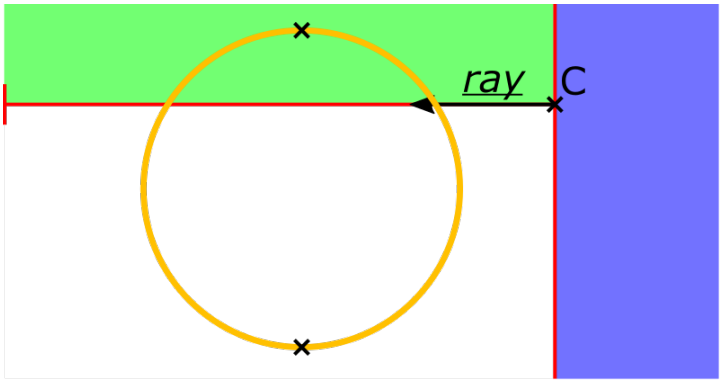


Mind a kör-, mind a sokszög-ütközőre igaz, hogy nem ütköznek a sugárral, ha az belőlük ered (a játékos nem lövi le saját magát).

Ha egy sokszögről kell eldönteni, hogy ütközik-e a sugárral, venni kell azon pontjait, melyek ütköznek az első határütközővel (előtte vannak), de nincsenek tőle távolabb, mint a maximális távolság. Ha ezen pontok között van olyan, ami a másik határütközővel is ütközik, meg olyan is, ami nem, akkor sugár eltalálja a sokszöget.

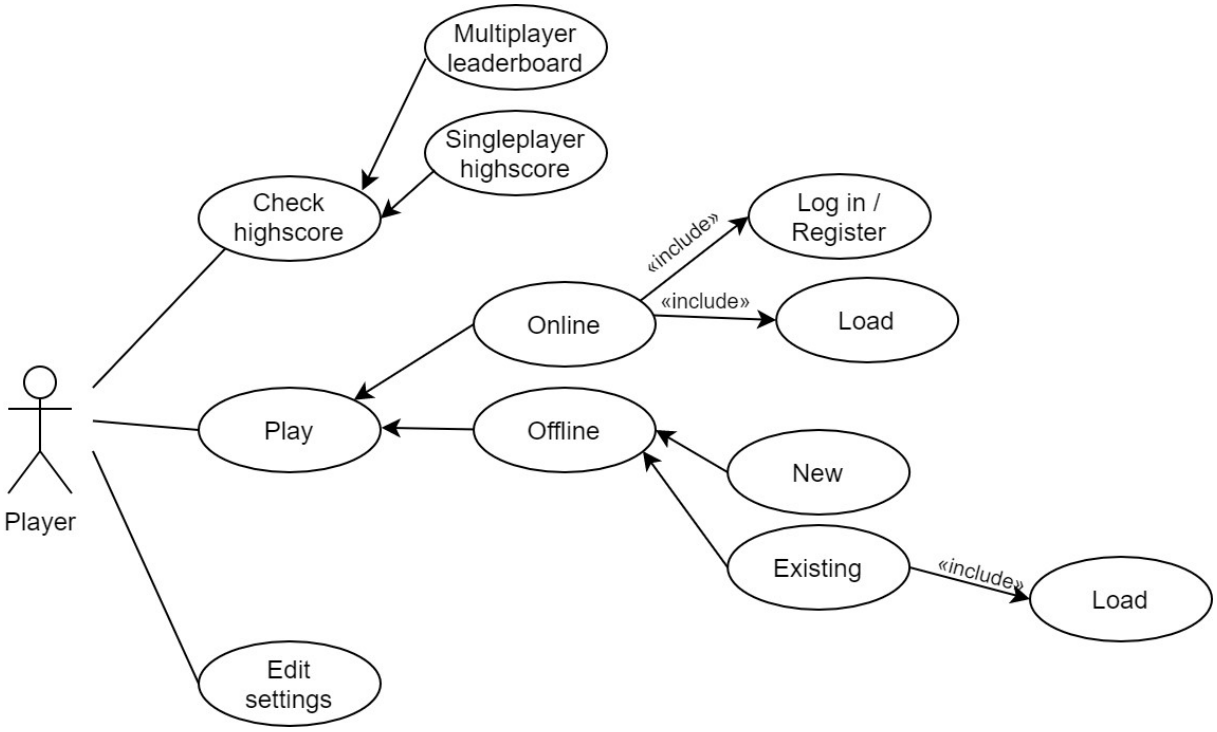


Körütközőnél is hasonló a helyzet, ott azonban nem a csúcsokat, hanem annak legszélsőbb pontjait kell vizsgálni.

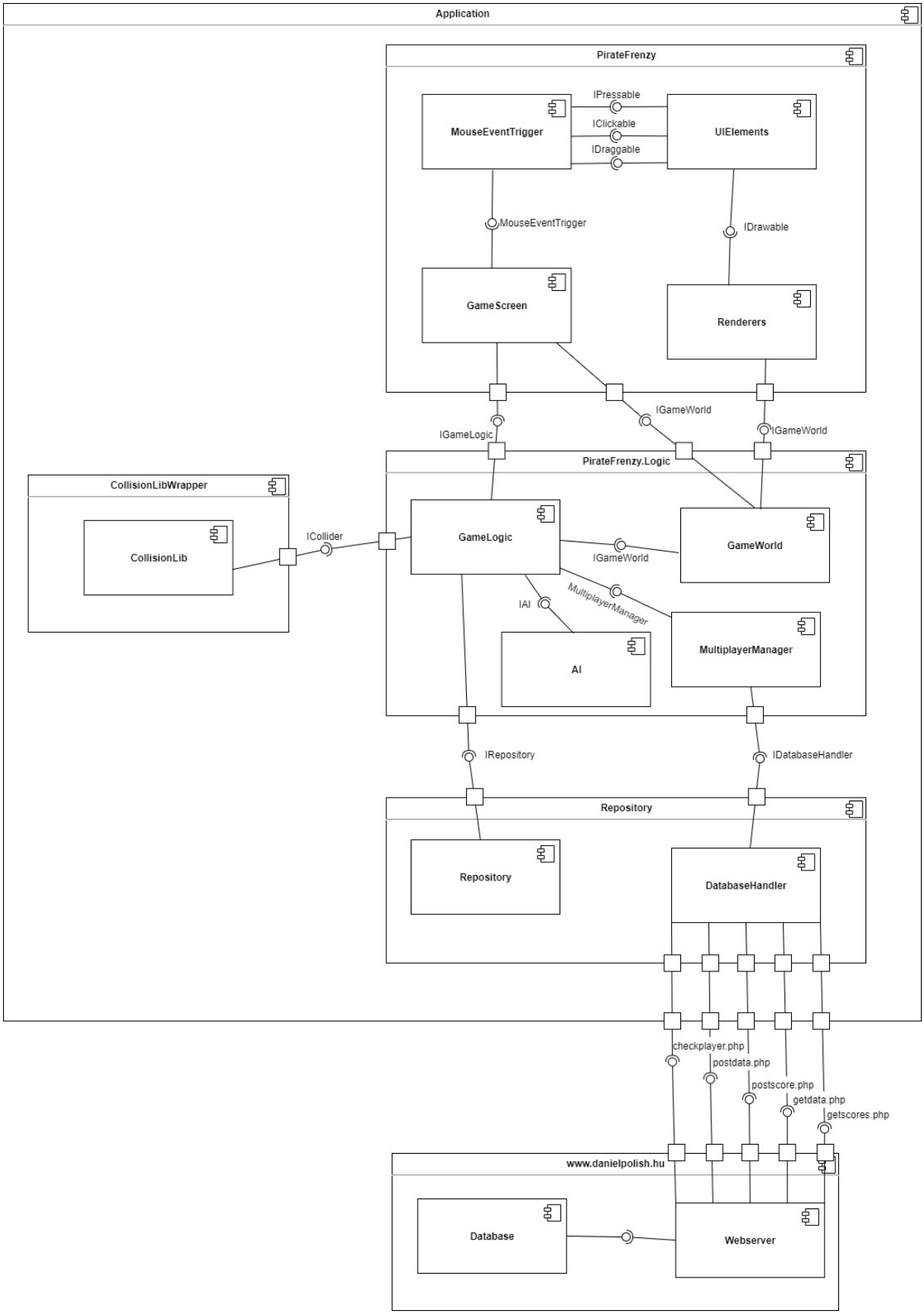


5. Diagramok

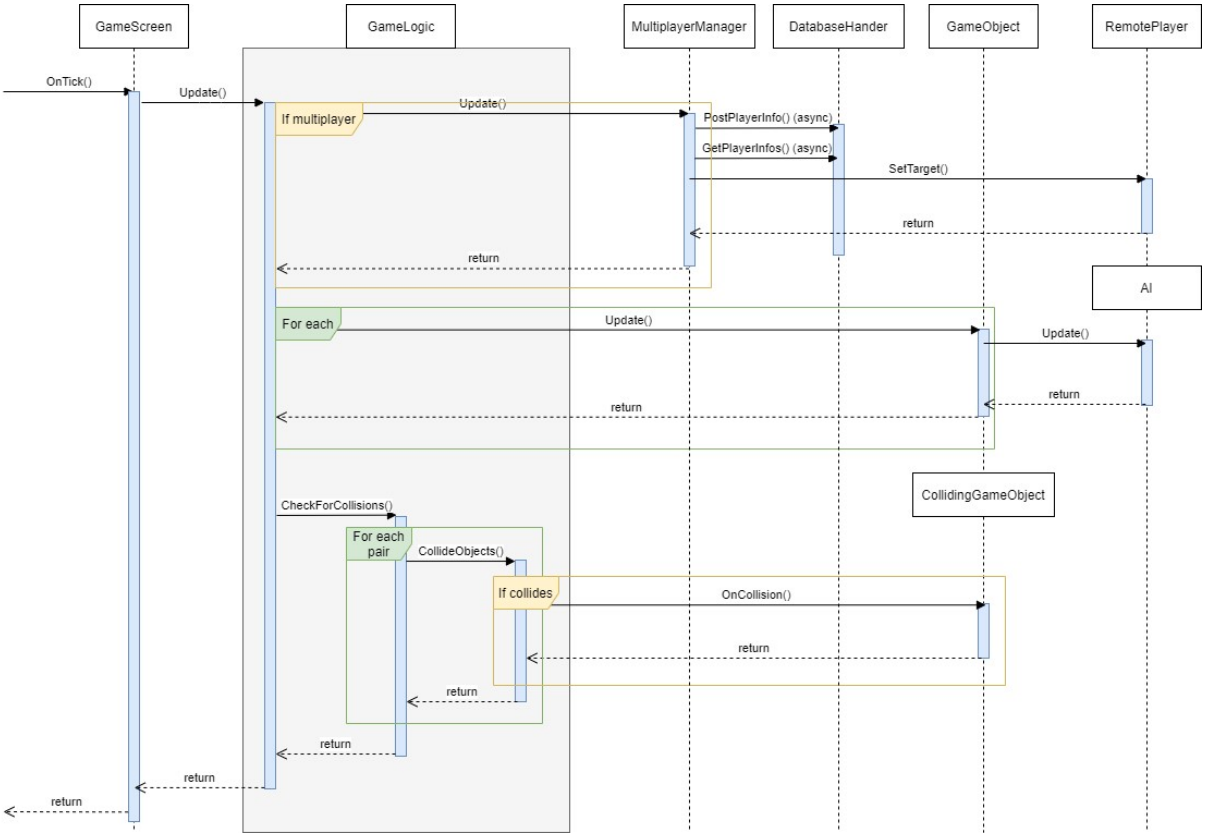
- Használati eset diagram



- Komponens diagram



- Szekvencia diagram



Osztály diagram

