Czimbal Balázs Beadandó Projekt

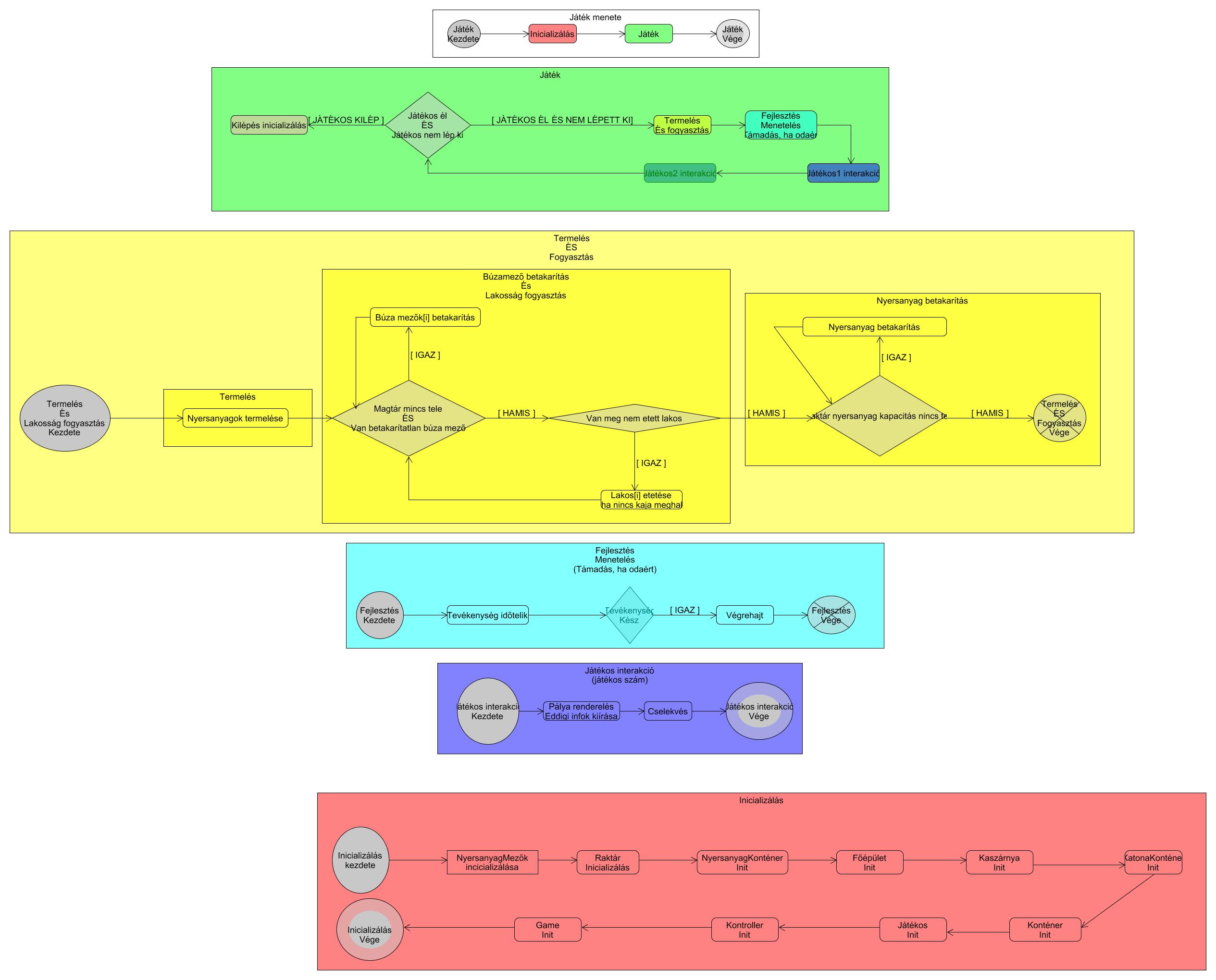
1. Program feladata, célja, fejlesztésének története

A program egy stratégiai játék alap. A programot két játékos játssza (multiplayer). A játékosok interakciója a fejlesztés jelenlegi szakaszában igencsak korlátozott (támadásra képesek). A kifelé való kommunikáció korlátozott, mert nem is tartottam fontosnak a program fejlesztése során, hogy az hogyan kommunikál a felhasználóval. Sokkal inkább tartottam fontosnak a játékmenet alapfeltételeinek megteremtését.

Amikor nekiálltam gondolkodni, hogy akkor most mit csináljak, először körbe néztem, hogy egy átlagos stratégiaijáték miből áll. Ebből elkezdtem megalkotni, hogy kb. hogy néz ki egy stratégiaijáték játékmenete. Darabokra szedtem a játékmenet működését, majd elkészítettem az Activity Diagramot, rajta a játékmenettel. Ezek után a játékmenet különböző részproblémáinak modelljeit készítettem el szépen sorban.

1. Program belső működése

Játék Menete (UML Activity Diagram):



A diagram alapján látható, hogy a játékmenetnek 2 főbb része van:

Inicializálás (Játék, játékosok, mezők, stb. inicializálása)

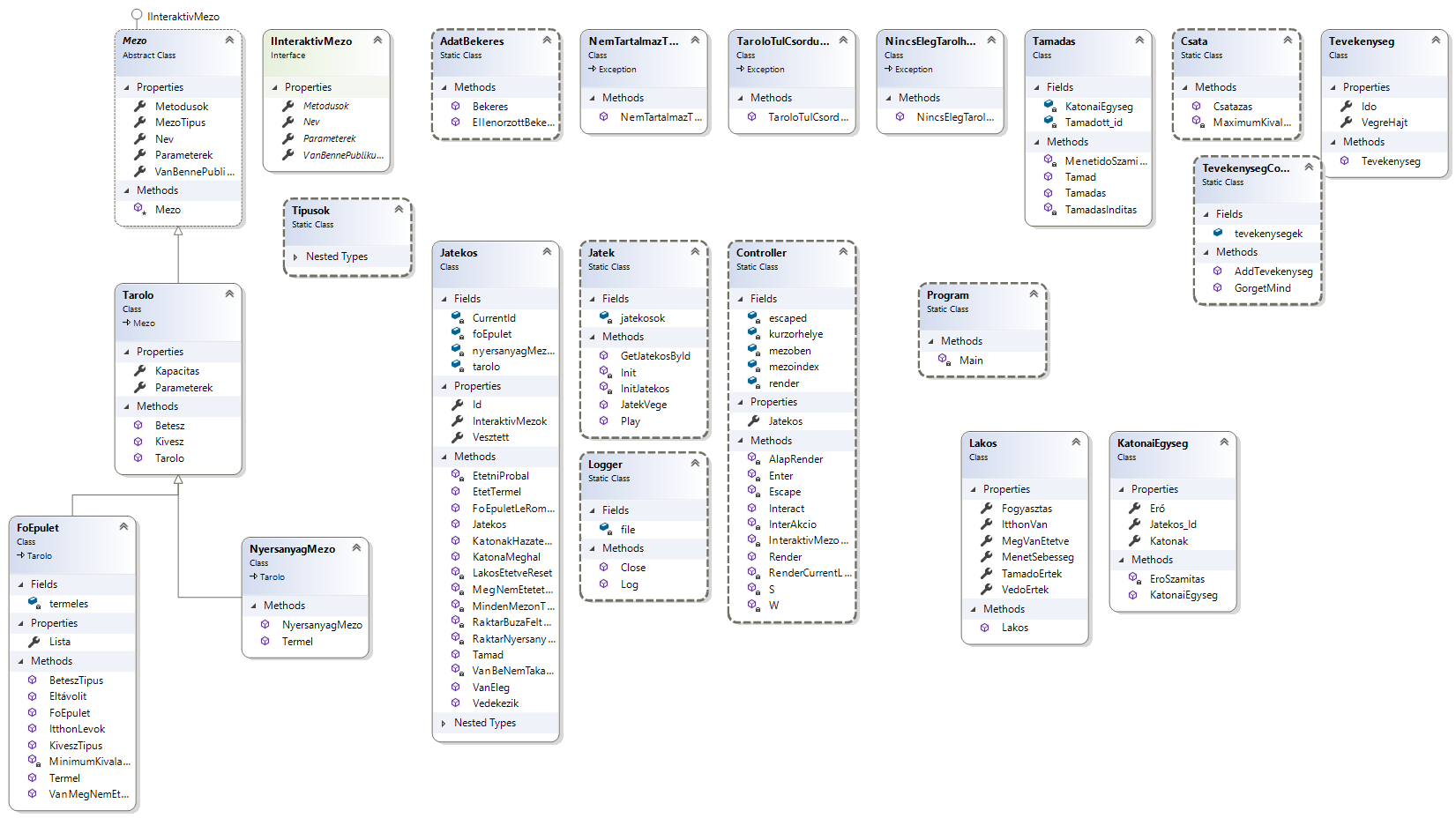
Játékmenet maga, ami további részekre bontható:

Termelés, fogyasztás, éhhalál

Idő teléssel kapcsolatos (fejlesztés, tevékenység, menetelés, stb.)

Játékos interakció

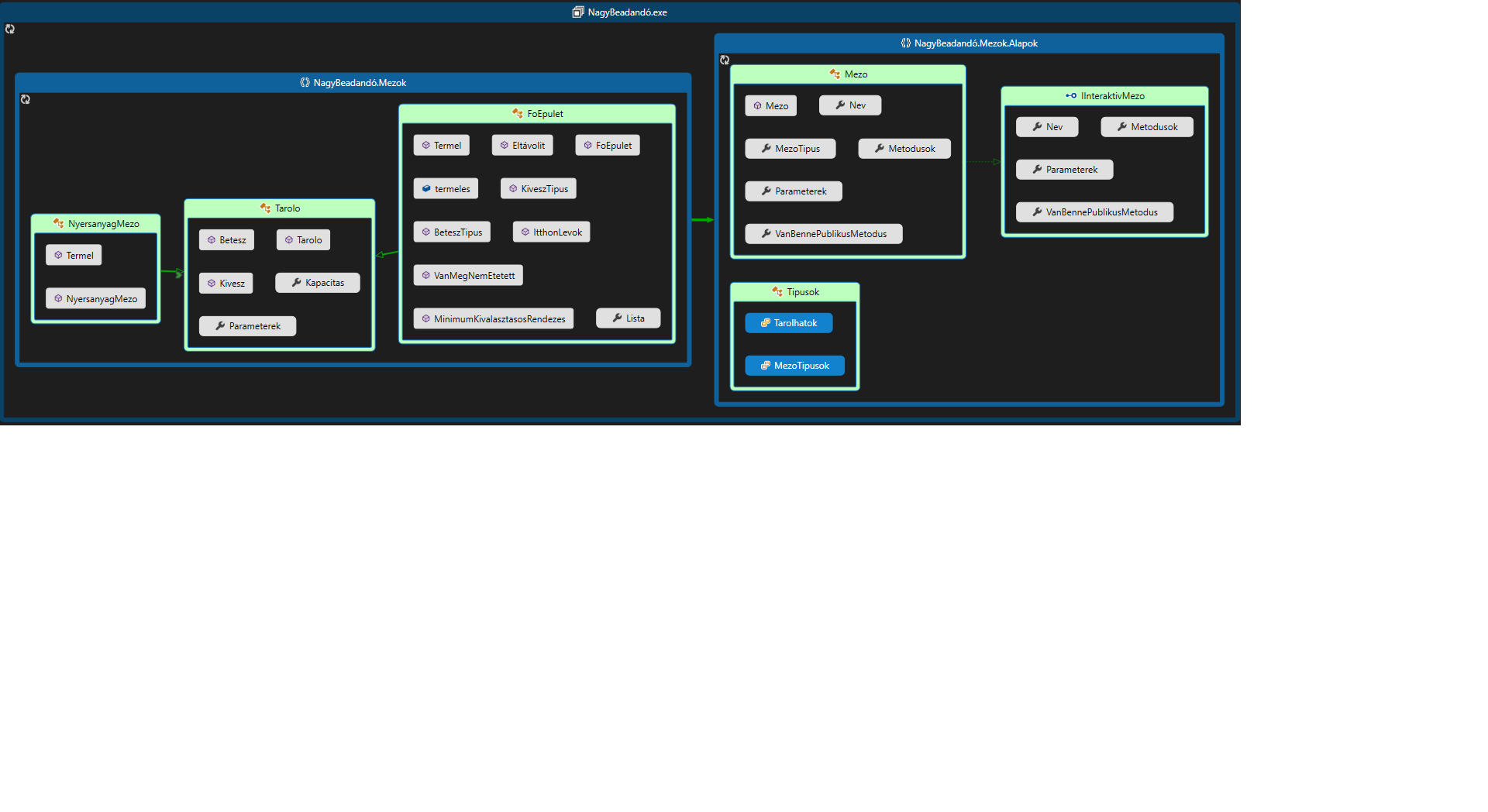
Játék felépítése (UML Class Diagram)



A Diagramon látszik, hogy sok osztály van.

Hogy ezt megfelelően tudjuk strukturálni, használunk package-t, vagy névteret, vagy dll-t programozási nyelvtől függően. Menjünk sorban a játékmenet és logikai sorrend - mi használ mit – szerint (rétegek).

1. Mezők:



Ő az alap csomagunk, itt vannak a játékban használható mezők, a hozzátartozó interface-ek és ősosztályaik.

Feladatok:

1. IInteraktivMezo (interface):

* Console-ra kiíráshoz nyújt segítséget (a mezők adattai (név, paraméterek, stb.))
* A mezőn játékos által végrehajtható metódusok meghívásához fontos (VanBennePublikusmetódus, Metódusok (dictionary tárol string-et és delegáltat, amit meghív, stringet kiírja, delegáltat meghívja))

1. Mező (abstract):

* Minden mező alapja
* Konstruktor beállítja nevet, paraméterek egy részét

1. Típusok (static):

* Tárolja a mezőtípusokat > ebből lesz a név, mezőben kell konstruktornak
* Tárolja, hogy miket lehet tárolni > tárolónak kell, hogy mit tárolhat

1. Tároló:

* Mindennek, aminek tároló szerepe van, alapot nyújt
* Tárolja, hogy mit tárol, abból mennyit, és mennyi a maximum tárolható mennyiség

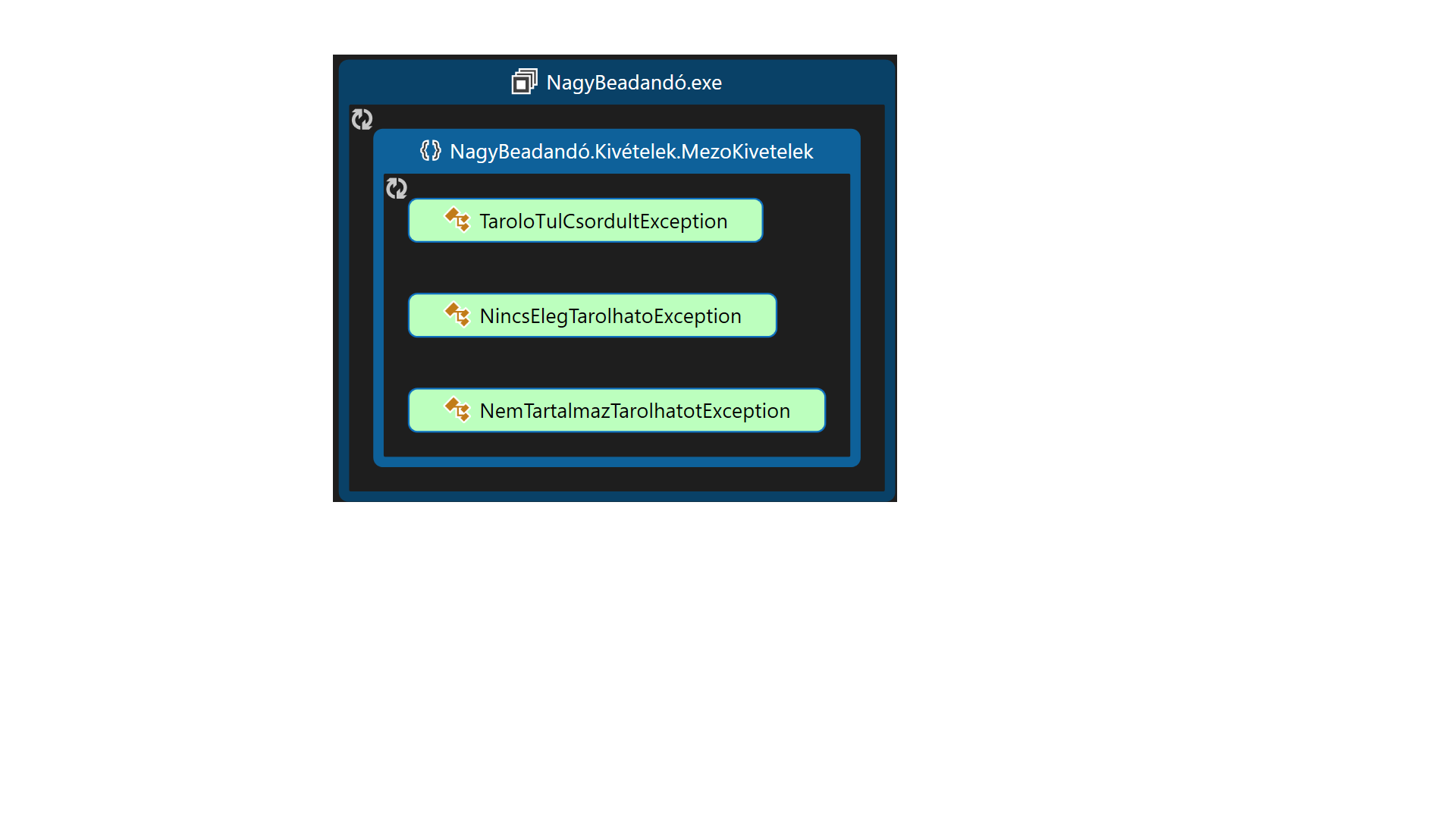
1. NyersanyagMező : Tarolo

* Termeléssel egészíti ki a tárolót, ami iterációs egységenként lefut

1. Főépület : Tarolo

* Játékos Lakosainak termeléséért, tárolásáért felel

1. Kivetelek:



Itt vannak a kivételek, amik a program során dobódhatnak. Jelen pillanatban, csak a mezőknek van szüksége kivételt dobni.

Feladatok:

1. TaroloTulCsordultException : Exception

* Akkor dobódik, amikor egy tárolóba szeretnék betenni olyan mennyiségű tárolhatót, amennyihez a jelenlegi tároltmennyiséget hozzáadva, az összeg nagyobb lesz, mint a tároló kapacitása.

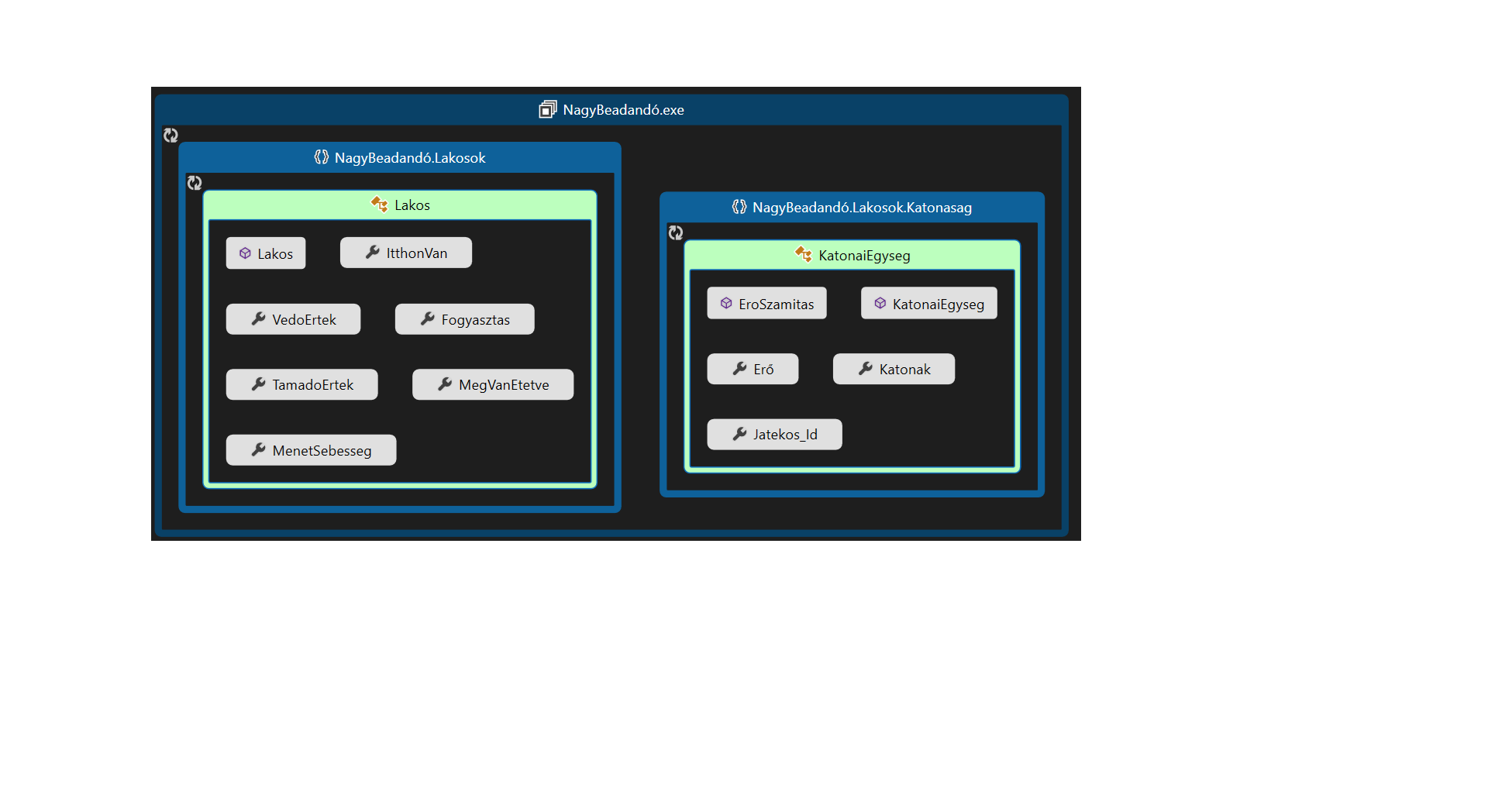
1. NincsElegTarolhatoException : Exception

* Akkor dobódik, amikor szeretnénk kivenni olyan mennyiségű tárolhatót a tárolóból, amennyi a tároló aktuális tárolt mennyiségénél kisebb.

1. NemTartalmazTarolhatotException : Exception

* Akkor dobódik, amikor egy tárolóból szeretnénk olyan tárolhatót kivenni, amilyent a tároló nem tartalmaz.

1. Lakosok



Itt találhatók meg a tárolható lakosok, illetve a lakosokat tartalmazó egységek, amik a lakosokat használják (itt lesznek később a különböző katonák is, amik majd a Lakos leszármazottai).

Feladatok:

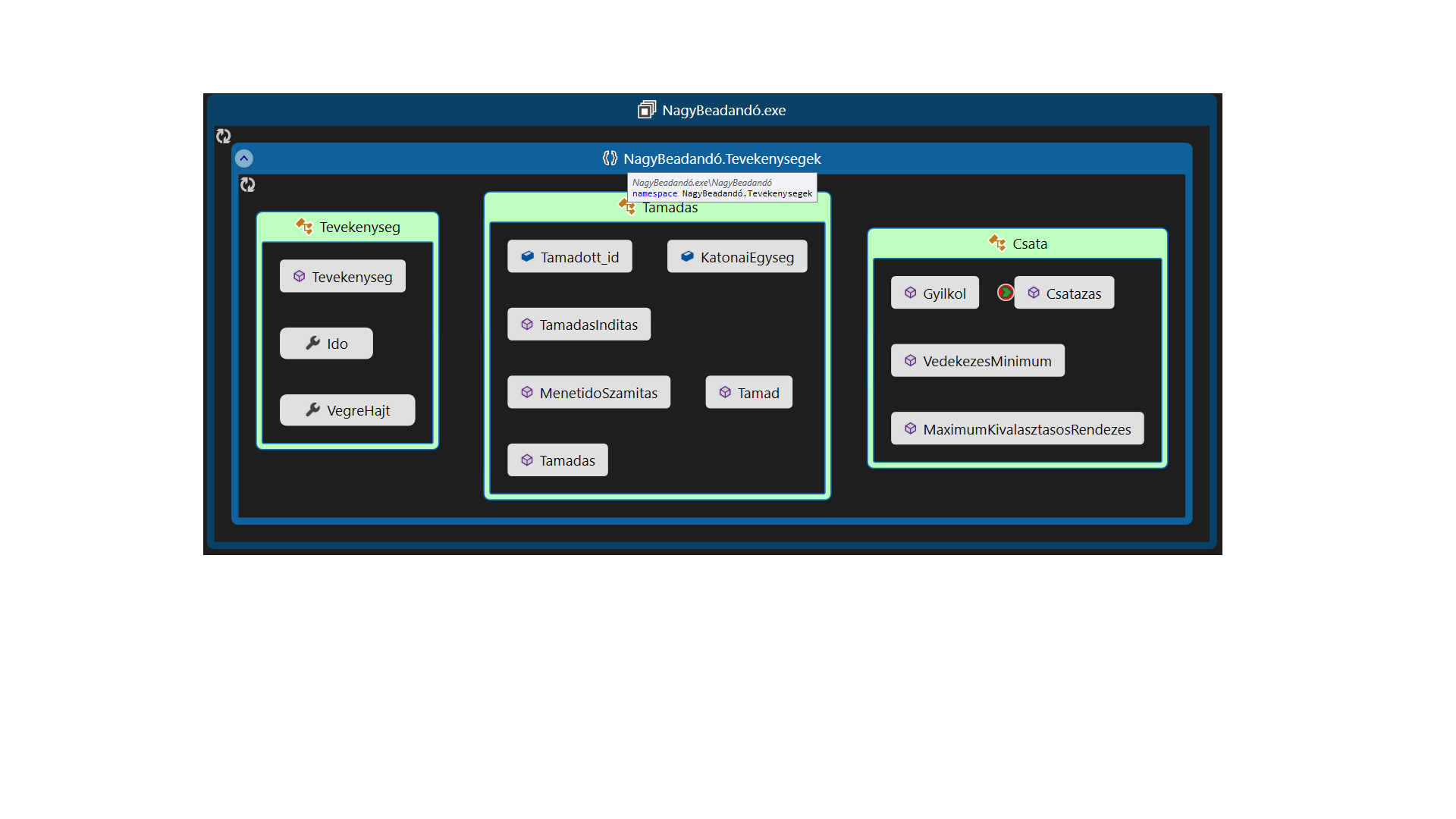
1. Lakos

* Tárolja a lakos adatait:
* Itthon van-e
* Mennyi a támadó, védő ereje (harc-nál használjuk fel)
* Menetsebessége (támadásnál, menetidő számításra használjuk fel)
* Fogyasztása (fogyasztási-termelési ciklusnál használjuk fel)

1. KatonaiEgység

* Lakosok listáját használja, tárolja
* Erőt tárol (támadás, védekezés függvényében), számol
* Játékos ID-jét tárolja, aki használja

1. Tevekenysegek



Itt tároljuk azokat az eseményeket, amik időtől függnek, vagy azokat használják. Tehát bármilyen eseményt, amelynek bizonyos iterációs egységnyi időt (termelés-fogyasztási ciklus, tevékenységek görgetése, két játékos interakciója) követően végre kell hajtódjon.

Piros pötty > break point

1. Tevekenyseg

* Tárolja, hogy a tevékenység mennyi idő után fusson le, és hogy mi az ami lefusson (Vegrehajt > delegált > Action > visszatérés, paraméter nélküli)

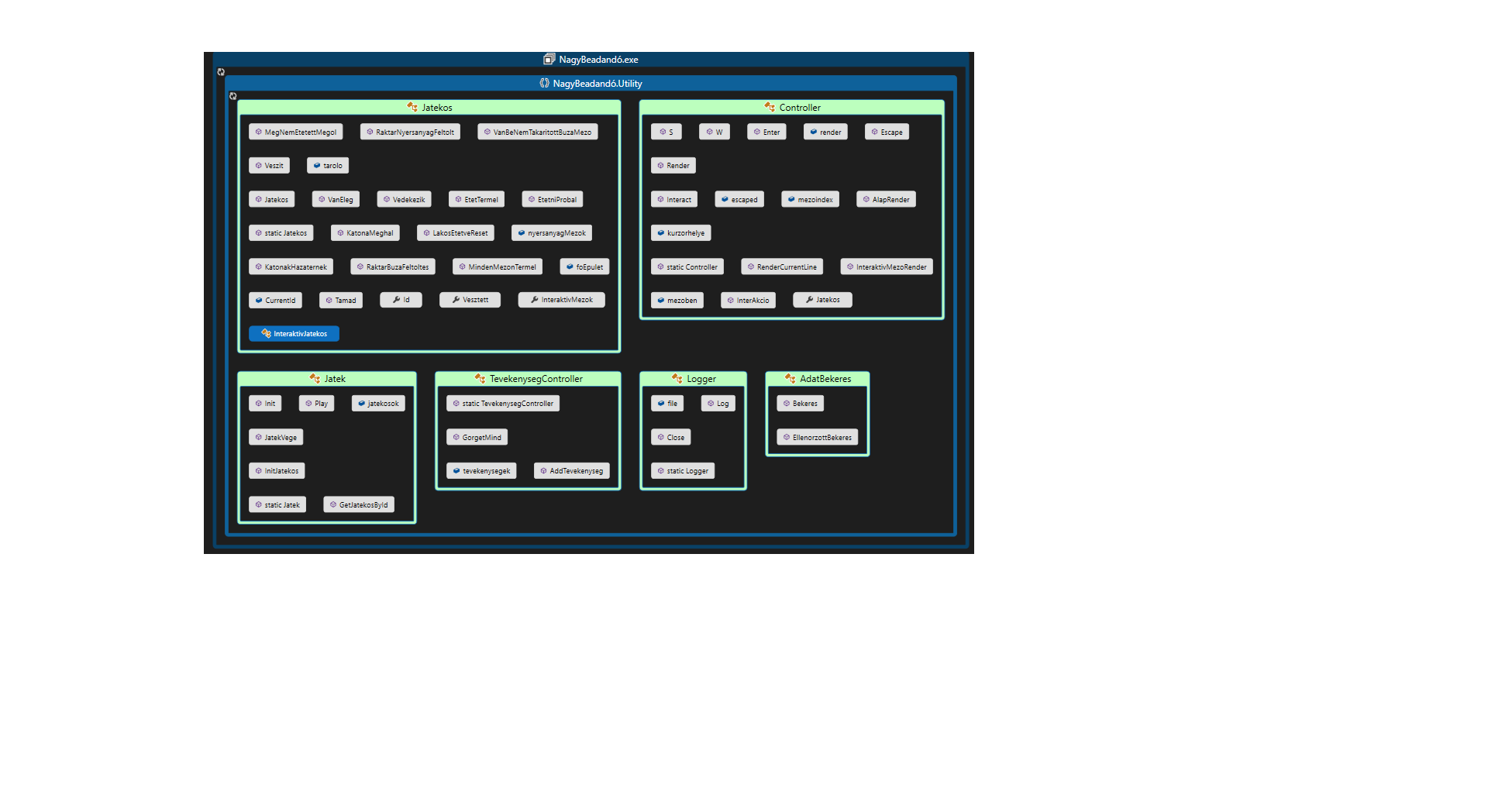
1. Tamadas

* Tárolja a támadott játékos ID-jét (nyilván valóan két játékosnál nem az a támadott aki a támadást indítja)
* Tárolja a katonaiegységet, ami tartalmazza a lakosokat, hogy ebből tudja levonni a csata közben elhalálozó katonákat
* Elindítja a támadást > generál egy védekezést > katonaiegységet a védekező játékostól
* Kiszámítja a menetidőt a katonák menetsebességétől és a távolságtól függően (mindkettő random szám)

1. Csata (static)

* Csatázás lefolyásáért felel
* Ha támadó erő nagyobb mint a védőé, támadó nyert
* Ellenkező esetben a védő erőből közel támadó erőnyit von le (Azért csak közel, mert a katonaiegységeket alkotó lakosok ereje random szám, így csak addig lehet kivonogatni a támadó erőt a védőerőből, amíg a maradék erő >= védőerő lakosainak legkisebb védőértéke)

1. Utility



Itt taroljuk a legfelsőbb rétegben található osztályokat, amik kezelik a program futását: Kezelik a játékot, játékosokat, mezőket, „renderelés-t” (kiíratást), log-olást, stb.

Feladatok:

1. Jatek (static)

* Ő kezeli a játék futását
* Inicializálja a játékot, játékosokat
* Ő adja át a console-t az éppen aktuális játékosnak interakcióra
* Ő kezeli a tevékenységek-et
* Ő kezeli a játékosok termelési-fogyasztási ciklusát

1. TevekenysegController (static)

* Ő kezeli a tevékenységeket (körönként csökkenti minden tevékenység idejét 1-el és ha így annak ideje nulla, akkor meghívja a végrehajt delegáltat bezáró metódust, majd törli a tevékenységek közül)
* Ő tárolja a tevékenységeket (hozzáad, töröl)

1. Logger (static)

* Ő kezeli az aktuális üzenetek file-ba, console-ra írását (menüt nem, interakciót sem kezel, csak kiír)

1. AdatBekérés (static)

* Ő kezeli az adatbekéréseket
* Jelenleg még nincs használva

1. Controller (static)

* Ő kezeli az aktuális menü console-ra rajzolását (játéks innteraktív mezőinek segítségével)
* Ő kezeli az aktuális játékos interakcióját (WASD, Enter, Escape) a menüben

1. Jatekos

* Ő tárolja és kezeli a játékos összes mezőjét
* Ő definiálja, hogy a játékos vesztett-e a játékban
* Ő kezeli a játékos termelési-fogyasztási ciklusát (minden nem búza mezőn termel, majd betakarít, majd amíg van be nem takarított búza mező és meg nem etet lakos addig eteti a lakost és búza mezőn betakarít, majd a meg nem etetett lakosokat elveszi a foepulet lakosok listajabol, és betakaritja a maradek buzamezoket)
* Kezeli a játékos védekezését (az összes hazai lakost csatába küldi a támadóval szemben)

1. Fejlesztési lehetőségek

* Játék gép ellen > játékos stílus definiálása > Mindig támad, sose támad, …
* Tároló Kapacitás újra gondolása (máshogy tárol a nyersanyagmező, és máshogy a Főépület > nyersanyagmező csak azt tárolja, mennyi van, főépület pedig konktér listát tárol a lakosokból) > megoldható tuple-el
* Mezők fejleszthetősége > többet tárol, termel, stb. > kell nézni mennyi a fejlesztés költsége (nyersanyag, stb.), van-e annyi nyersanyag, majd végrehajtani x idő után a fejlesztést> megváltoztatni az értékeket
* Új Katona típusok > fix támadó-, védő értékkel, menetsebességel, stb.
* Speciális katona típusok pl kém, lovas (kell lovarda > tároló > lovakat tárol)
* Speciális képzések > lakosból lesz katona, kovács, lovász, stb.
* Építmények : lovarda, kovácsműhely, stb.
* Vizuális megjelenítés > pályán jelölni a mezőket és kiírni a paramétereiket
* Játékos kód darabokra bontása > nem szép, ha mindent a játékos kezel (termelés, tárolás, védekezés, támadás, termelési ciklus, katonák kezelése, stb.)
* Nyersanyagmezők termelésének jogát elvenni a játékostól > nem az ő feladata, hanem a játék mechanika része
* Játékosnak lehetnének message-ei string lista, amit kiír minden renderelésnél az utolsó x-et
* Message > keletkezés időpontja, szöveg
* Adatbázis : tárolások, termelések, nyersanyag szükségletek > ne programban tároljuk mert sok memóriát foglal feleslegesen mert csak akkor van szükség rájuk amikor éppen használjuk, nem a program egész élet ciklusában