

Házi feladat

Programozás alapjai 2.

Funk Gábor

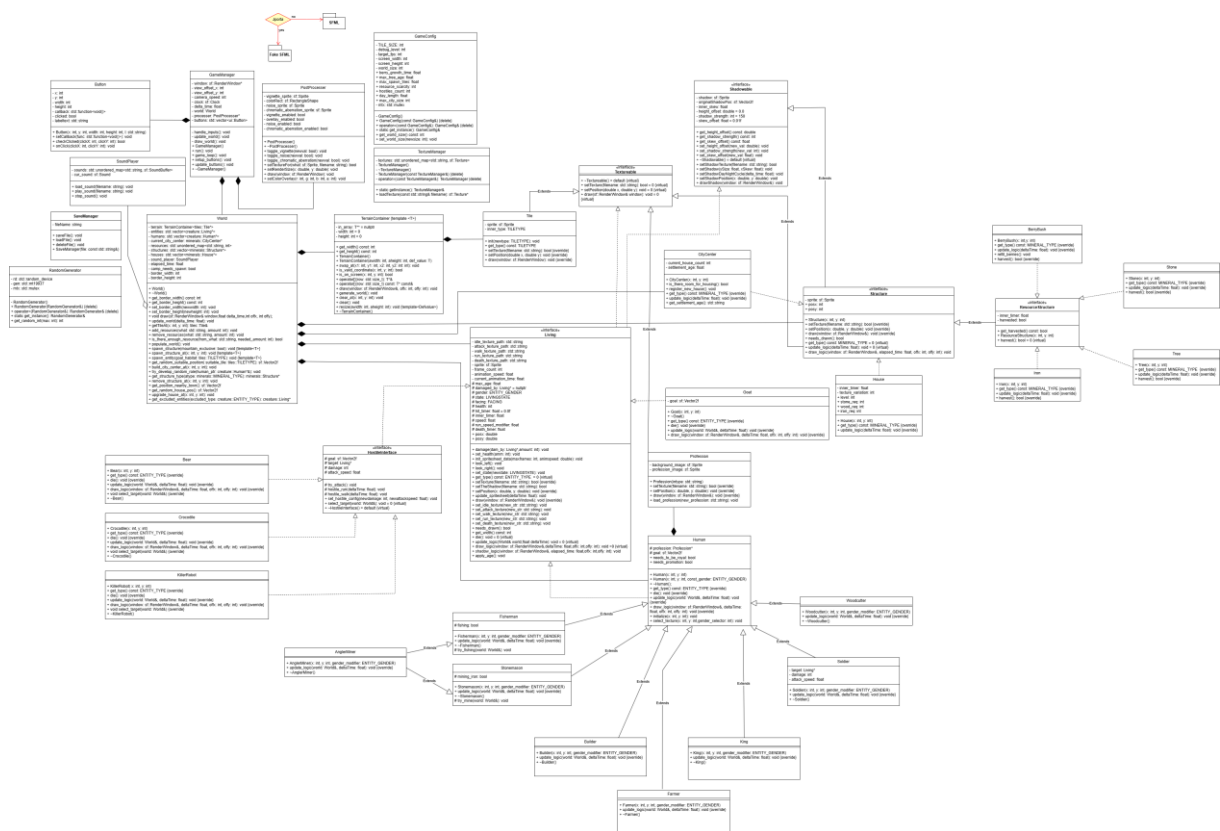
YSDDH7

NHF Terv (2. rész)

Valós idejű ember csoport szimulátor terve

Terv és pontosított specifikáció

Az osztálydiagram (UML):



A diagram tartalmazza az osztályokat, annak tagfüggvényeit és adattagjait. Sajnos a PDF formátumba exportálás elrontja a minőségét a nagy képeknek.

Az osztálydiagram elérhető nagy felbontásban ezen a linken: <https://bugfr.ee>

A Ctrl és “+” lenyomásával nagyítani lehet a képet, a Ctrl és “-” lenyomásával pedig kicsinyíteni. A görgő egyszeres lenyomásával és az egér mozgatásával fel,le,jobbra,balra lehet navigálni.

Külső források:

A grafikus megjelenítéshez és hangokhoz a program az SFML könyvtárat fogja használni.

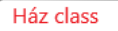
Annak érdekében, hogy a Jportán tesztelhető legyen a program, készíteni fogok egy Kamu_SFML könyvtárat, amibe hasonló osztályok és metódusok lesznek, mint az igazi SFML grafikus könyvtárban, csak nem rajzolnak ki semmit. Ezek a saját osztályok csak kiabáló osztályok lesznek, például, ha betöltök egy textúrát egy fájlból egy változóba, akkor kiíródik, hogy **“Textúra betöltve innen: ./kepek/kep.png”**. Hibakezelés is lesz, például, ha nem találja a képet a program, akkor a **“Textúra betöltése sikertelen innen: ./kepek/hianyzo.png”**.

Játékmenedzser:

Ez lesz a fő osztály, ez fog felelni a világ szimulációjának elindításához, a grafikus ablak létrehozásához és majd ha a program befejeződik, felszabadítja a világot. Ez az osztály fogja végezni a gombok tárolását és frissítését is, valamint az irányításkezelést is.

A világban a felülnézetes kamerát a nyilakkal (jobb, bal, fel, le-vel) lehet mozgatni. A menü 4 gombból fog állni: mentés, betöltés, új szimuláció. A gombokat az egér bal gombjával lehet aktiválni.

A következő oldalon található a játékmenedzserhez (Game Manager) tartozó classok UML osztálydiagramja. Ezen csak a az egyszerűbb és átláthatóbb képért a szomszédos osztályok láthatóak.



- Run(): Elindítja a világ szimulálását.
- Game_loop(): Ez a függvény felelős a képek kirajzolása közötti idő determinálásáért (delta time kiszámítása), az input kezeléséért és a világ szimulálásáért.
- Handle_inputs(): Ebben a függvényben történik a gombok kattintásának érzékelése és a jobb-bal-fel-le nyilakkal a kamera mozgatása. A kamera nem tud kimozogni a világ határain kívül, (például nem tud x:-1 y:-1 helyen lenni, mert a

világ x:0 y:0-tól kezdődik) csak akkor frissíti a kamera helyét, ha az nem megy ki a világból.

Világ:

A világ egy előre meghatározott méretű grid alapú szimuláció less, aminek a terepe véletlen generálást használ. Az élőlényeknek van egy maximum élettartamuk. Ha ezt eléri akkor meghalnak. Az élőlényeknek rengeteg célja lehet, amit a belső működésük határoz meg, de ahhoz, hogy ezt végre tudják hajtani, oda kell menniük ahol végre akarják a célt hajtani. Példa: A favágó ahhoz, hogy fát tudjon vágni, oda kell menni a fához.

A világ fontosabb adattagjai:

- Terrain: Ez tárolja a terep kockákat. Ez egy speciális dinamikus 2 dimenziós tömb, amely képes föld, víz, hegy kockákat tárolni
- Entities: Ez a heterogén kollekció tárolja az összes entitást az emberek kivételével
- Humans: Ez a heterogén kollekció tárolja az összes embert.
- Jelenlegi városközpont: Egy pointer a jelenlegi város központra.
- Structures: Eltárolja a fákat, bokrokat, követ és minden erőforrás struktúrát egy heterogén kollekcióba.
- Houses: A házak tárolására alkalmas dinamikus tömb.
- Resources: Az emberek által bányászott erőforrások tárolása.

A világ fontosabb metódusai:

- Draw(): Kirajzol mindent, ami a világba van, a terepkockákat, entitásokat és az erőforrás lelőhelyeket.
- Update(): Frissít minden entitást és struktúrát az előző frissítés óta eltelt idő függvényébe.
- Populate_world(): Új állatokat és erőforrásokat idéz a világba. Amikor a világ létrejön (A konstruktor meghívja) akkor is meghívódik ez. Ezen kívül időnként meghívódik, hogy ez emberek sose fussanak ki a fából, kőből, vasból vagy ételből.

Textureable és Shadowable interface:

2 Fontos interface-t tervezek, melyek egyszerűsítik a kirajzolást:

«interface» Textureable
<pre>+ ~Textureable() = default {virtual} + setTexture(filename: std::string): bool = 0 {virtual} + setPosition(double x, double y): void = 0 {virtual} + draw(sf::RenderWindow& window): void = 0 {virtual}</pre>

«interface» Shadowable
<pre>- shadow: sf::Sprite - originalShadowPos: sf::Vector2f - inner_skew: float - height_offset: double = 0.0 - shadow_strength: int = 150 - skew_offset: float = 0.01f</pre>
<pre>+ get_height_offset() const: double + get_shadow_strength() const: int + get_skew_offset() const: float + set_height_offset(new_val: double): void + set_shadow_strength(new_val: int): void + set_skew_offset(new_val: float): void + ~Shadowable() = default {virtual} + setShadowTexture(filename: std::string): bool + setShadow(ySize: float, xSkew: float): void + setShadowDayNightCycle(delta_time: float): void + setShadowPosition(x: double, y: double): void + drawShadow(window: sf::RenderWindow&): void</pre>

Textureable:

- setTexture(): Egy textúra beállítása. A textureManager nevű class loadTexture() nevű metódusát használva megnézi, hogy be van-e töltve már a keresett textúra. Ha van akkor azt beállítja magának, ha nincs akkor betölti a fájlból és azután állítja be magának. Fontos, hogy az, hogy hova állítja be a textúrát majd a megvalósítástól függ.

- setPosition(x,y): megvalósítástól függően beállítja a pozíciót, hogy hova kell kirajzolódnia.

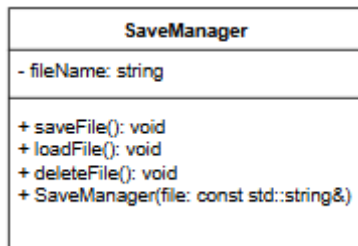
-draw(): Kirajzolja a sprite-okat vagy sprite-ot megvalósítástól függően.

Shadowable:

Ez az interface azért fontos, mert a világba lesz napszak és ez fogja megvalósítani, hogy például az emberek árnyéka időtől függően merre álljon. Ha dél van akkor az árnyék nem látszik, ha este van, akkor se.

Egy kirajzolható dolognak lehet a kinézetétől függetlenül másik árnyéka, és lehet egy olyan entitás is, ami láthatatlan, ezért fontos, hogy ez a 2 interface külön legyen.

Mentés:



A mentés

A mentés gomb menti az állapotot, a betöltés pedig betölti, az új szimuláció pedig új állapotot hoz létre ami nem írja át automatikusan a mentett állapotot.

Az állapotot a program a `save_data.dat` nevű fájlból olvassa és menti. A mentés tartalmazni fogja:

- Élőlényeket
- Erőforrás lelő helyeket
- A világ terepéhez szükséges seedet
- A világban eltelt időt
- Az emberek által termelt erőforrásokat
- A napszakot
- Az emberek által épített házakat

A mentés fájl formátuma:

1. Sor: **<a világ nagysága szám>|<az eltelt idő>|<a világ generálásához szükséges seed>**
2. Sor: **<emberek által gyűjtött vas száma>|<kő száma>|<étel száma>|<farönk száma>**
3. Sor: entitások és attribútumaik ;-vel elválsztva. Példa (*Név;X pozíció;Y pozíció; kor évben*): **<"Kecske";4;6;7>|<"Ember";20;41;5>...**
4. Sor: az emberek által épített házakat és erőforrás lelő helyeket tárolja ugyanabban a formátumban, mint a 3. Sor.

A világ terepe:

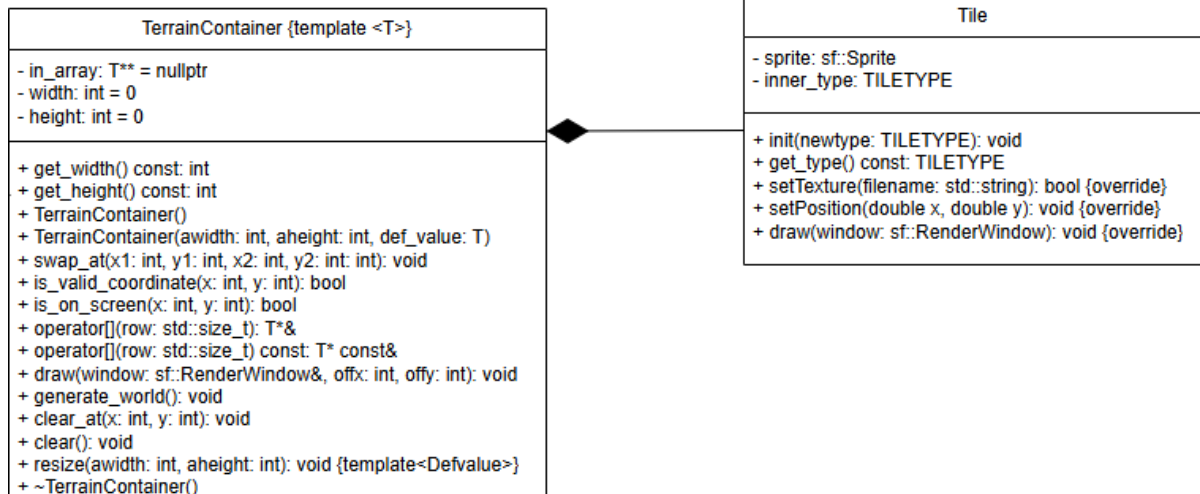
A világ egy 2 dimenziós, speciális dinamikus tömb lesz, ami képes tárolni terepkockákat.

Inicializálásnál meg kell adni egy N számot, amely a világ nagysága lesz. N -től függően a világ $N*N$ terepkockából fog állni.

A terep tároló felelős a saját terepkockáiért, ha megszűnik ez a tároló, akkor megszünteti a tárolt terepkockákat is.

A terep tároló képes lesz egy véletlen világot generálni, ahol a tengerek, hegyek és tavak véletlenül fognak elhelyezkedni.

Az alábbi UML diagram részlet bemutatja, körülbelül hogy fog kinézni a terep tároló a végleges fázisban.



A grid 3 féle tile-ból fog állni:

-Víz: Itt lelhetőek halak, a halászok mindíg ezeket a tile-okat fogják keresni.

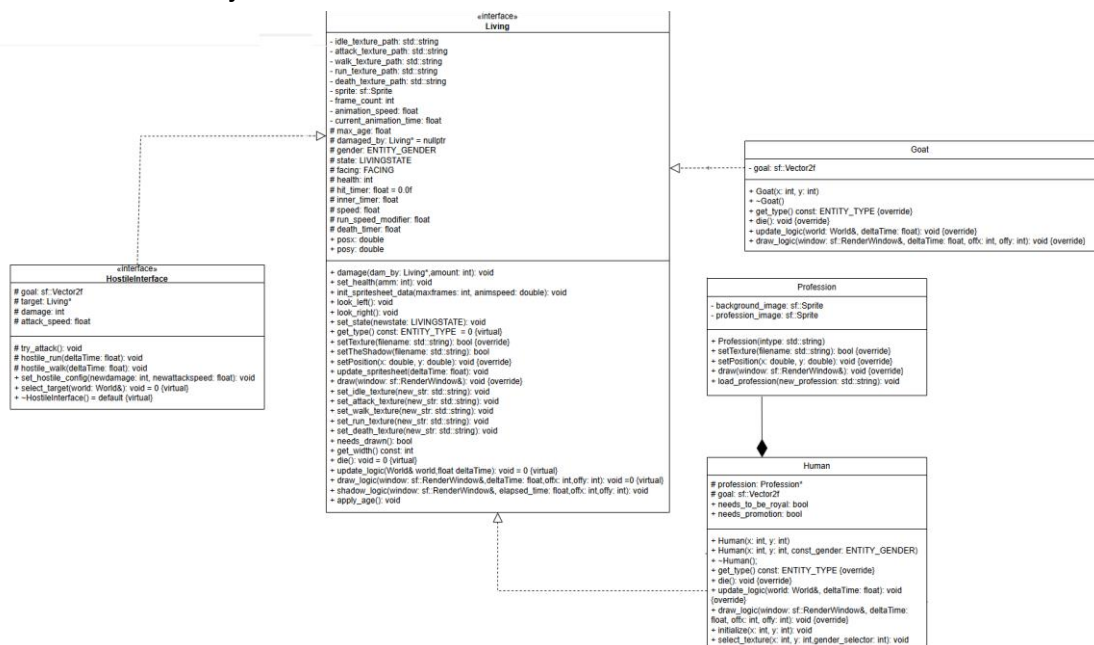
-Mező: Itt lelhető fa és étel, ezen kívül itt medvék fognak idéződni.

-Hegy: Itt lelhető vas és kő.

A tile-ok típusa nem befolyásolja, hogy az élőlények milyen gyorsak azokra lépve.

Entitások:

Az entitások generalizálására készíték egy Living interface-t, amely megköti, hogy mi kötelező egy entitásnak. Ez a diagram tartalmazza a fontosabb interfaceket és osztályokat, amelyeknek még lesz sok leszármozottja:



Az ember osztályból fognak a különböző emberek szakmától függően öröklődni. A szakma (Profession) osztály az csak egy jelző lesz, ami kirajzolja annak a szakmának az ikonját, amit az ember művel. Az ellenséges interface-ből (HostileInterface) fognak azok az állatok / entitások öröklődni, amelyek más állatokat vagy embereket képesek megtámadni. A kecske egy semleges entitás, ezért ő csak a Living interfaceből öröklődik. A

living interface megkívánja, hogy a belőle öröklődő osztályok megvalósítsák ezeket a tagfüggvényeket:

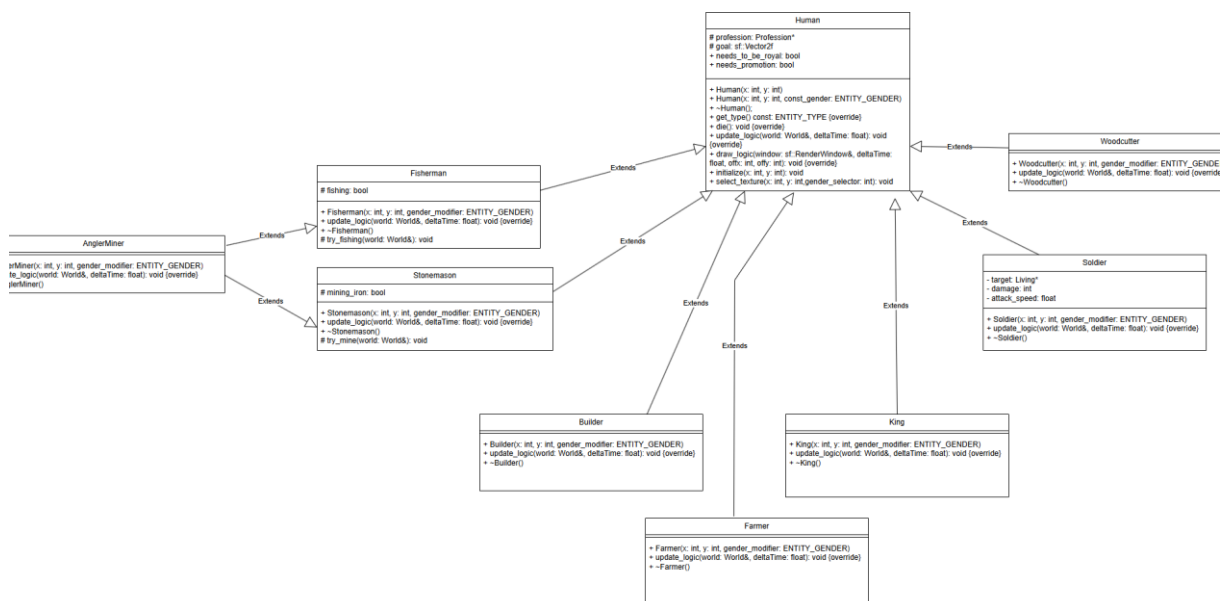
- Die(): Mi történjen, ha meghal az entitás?
- Get_type(): Ez egy enumerációt ad vissza. "HUMAN" (Ember)-t, ha ember az élőlény, vadállatot.
- Update_logic(): Itt kell leírni egy élőlény viselkedését. Például a krokodil odamegy más állatokhoz és megeszi őket. A halász pedig tavat keres, hogy tudjon halászni.
- Draw_logic(): Hogyan rajzolódjon ki az állat? Például az embernél a szakma ikon-t is ki kell rajzolni.

Emberek:

Az emberek speciális élőlények, amik képesek város létrehozására és építésére is.

Amennyiben már van létező városeelem akkor megpróbálnak egy szakmát felvenni. Egy szakma felvétele a városnak erőforrásba fog kerülni. Egy ember csak 1 szakmát vehet fel.

Szakmák:



Az emberek szakmától függően fognak viselkedni. Minden szakma az ember osztályból öröklődik. Az "Angler Miner" kivétel, mert ő a bányászból és a horgászból fog öröklődni. Az ő esetében fennáll a gyémánt öröklődés.

Minden ember feje felett ott fog lebegni egy ikon, ami mutatja, hogy milyen szakmája van. Ha nincs szakmája akkor egy "Zzz" alvó ikon lesz a feje fölött. Ezek a választható szakmák:

Király: Ezt a kasztot akkor kapja meg egy ember, ha ő hozta létre a várost. Létrehozás után csak sétál össze-vissza, így "felügyeli" a királyságát. Szakma ikon: arany korona.

Harcos: Ha egy ember ezt a kasztot viseli, akkor az a feladata, hogy vadásszon állatokat és megvédje a többi embert az ellenséges lényektől. Szakma ikon: kard

Építész: Ha van építésre elég erőforrás akkor a város közepe köré megpróbál új házakat építeni. Ha vannak régi házak, azokat is megpróbálja korszerűsíteni. Szakma ikon: téglá.

Farmer: Bogyóbokrokat keres és leszedi őket. Szakma ikon: kasza.

Halász: Vízet keres és a víz tile-okon halászik időnként. Szakma ikon: horgászbót.

Bányász: Vasércet és követ termel. Szakma ikon: csákány.

Favágó: Fákat keres és kivágja őket. Szakma ikon: fejsze.

Angler-Miner: Ez egy speciális kaszt, mivel akinek ez a szakmája az halászni is tud és bányászni is. Szakma ikon: csákány amin halak lógnak.

Más előlények: (kecske,krokodil,medve,gyilkos robot)

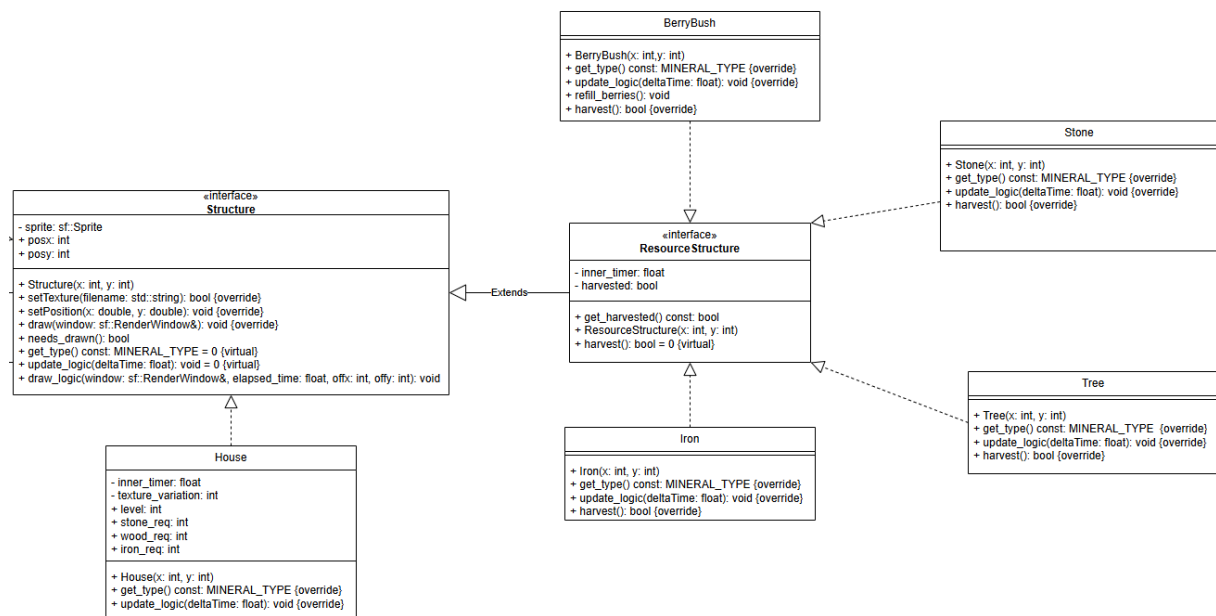
Kecske: Ártalmatlan állat.

Krokodil: Lassú de erős vadállat.

Medve: Gyors és nagyon ellenséges vadállat.

Gyilkos robot: Nagyon ritkán idéződik, az a célja, hogy mindenkit elpusztítson. 999 évig él, így szinte csak akkor tűnik el, ha az emberek elpusztítják.

Erőforrások:



Az erőforrás struktúra interface-ből öröklődik 4 darab “épület”, ami a világba megjelenik. Ha ezeket az emberek lebontják, akkor különféle erőforrásokat kapnak. Létezik a sima struktúra is, amelyet nem akarnak az emberek lebontani. Ilyen a városközpont és a ház.

4 féle erőforrás van:

stone (kő): Házak és bizonyos szakmákhoz kell. Kőhegyből lehet szerezni.

wood (fa): Fa vágásával szerezhető. Szinte mindenhez kell.

food (étel): Bogyóbokorból, halászatból és vadászatból szerezhető. Szükséges, hogy az emberek életben maradjanak.

iron (vas): Fejlettebb szakmákhoz és modern házakhoz kell. Vasércből lehet kinyerni.

Erőforrás menedzsment:

A világ nyilvántart egy globális erőforrástárolót, amibe minden ember bele tud nézni, bele tud rakni és ki is tud venni akármennyi erőforrást.

A világ bizonyos mennyiségű időnként "megpróbálja megetetni" az embereket. Ha egy ember nem kap ételt akkor meghal.

Épületek:

Az épületek ahhoz, kell, hogy időnként új emberek idéződjének. Épületet az építész tud készíteni, kivéve a városközép (egy kút) épületet. Azt egy ember akkor épít, ha nem talál létező várost. Az építész fejleszteni fogja a lakóházakat, ha van erőforrás maximum 2-szer. A fejlesztett házakból gyorsabban idéződnek új emberek.

TextureManager és PostProcessor:

A gyorsabb fejlesztés érdekében létrehozok segédosztályokat, melyek arra szolgálnak, hogy gyorsítsák a programot:

TextureManager:

Eltárolja a már betöltött textúrákat, hogy később, ha szükség van még egyszer ugyanarra a textúrára, ne kelljen kétszer betölteni.

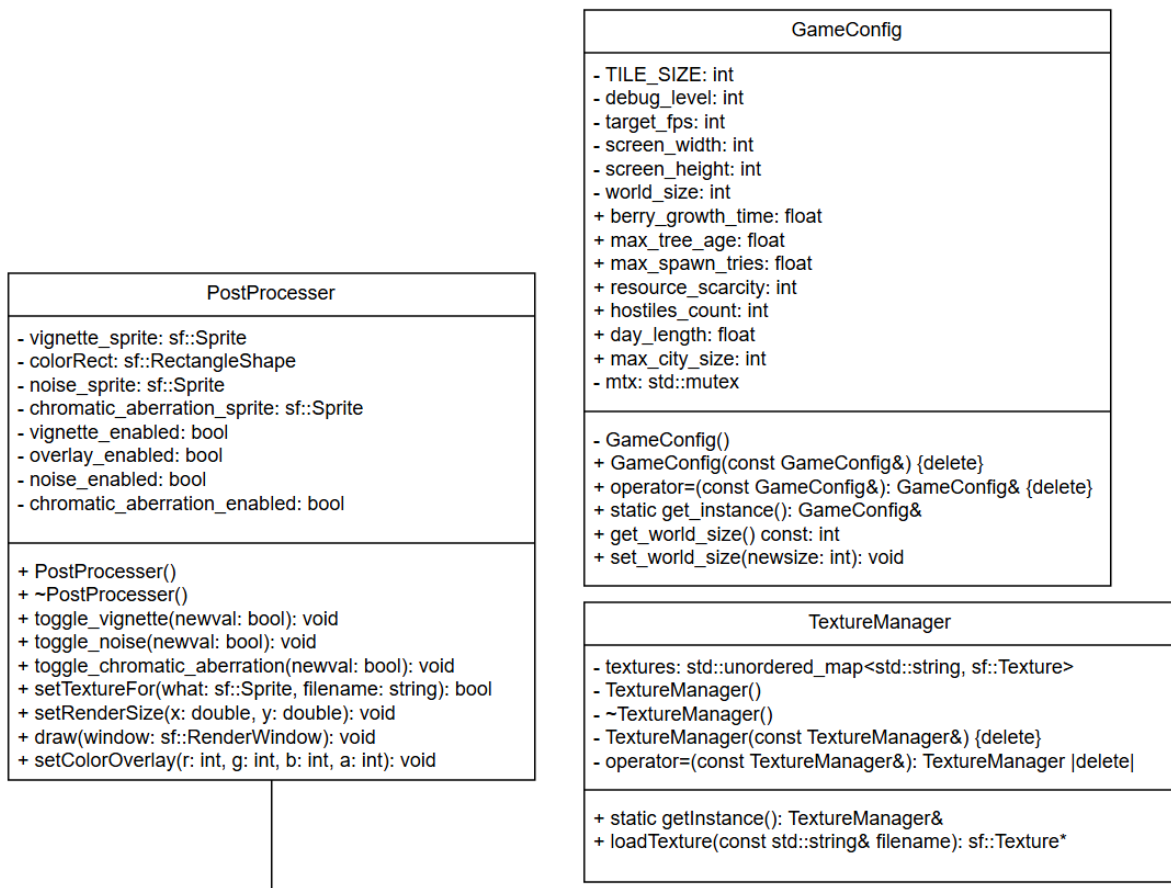
Postprocessor:

Vizuális effektek beállítására való osztály. Lehet zajt, elmosódást és színhatás effekteket beállítani vele. Ezen felül még beállítható az, hogy a képernyő széle sötétebb legyen, ezzel az éjszaka napszakot szimulálva.

GameConfig:

Ez egy Config osztály, ami azt segíti, hogy ne rendszertelenül legyenek szétszórva azok a változók, amelyek fontosak a szimulációhoz. Így könnyebb kísérletezni, hogy milyen beállításokkal kapok élethűbb szimulációt.

A kezdetleges UML diagramja ezeknek az osztályoknak:



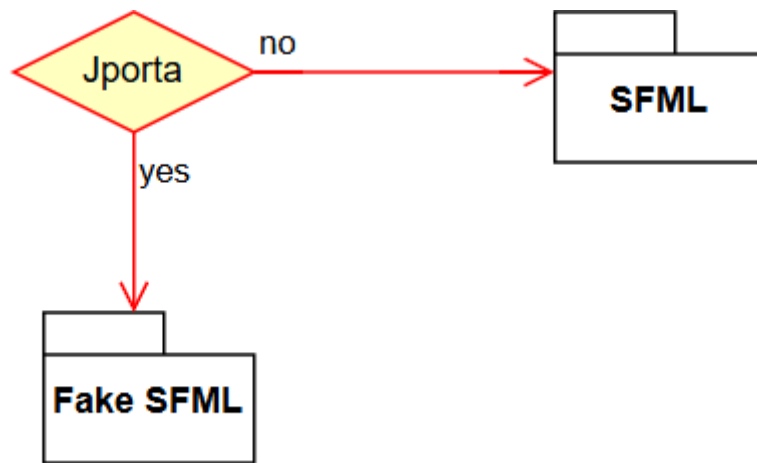
Kikötések:

- A városközpont lerakásakor a globális erőforrástárolóba 10-10 erőforrás bekerül és 5 ember leidéződik, hogy életképes legyen a szimuláció hosszú távon is. (Ez azért kell, hogy a király ne haljon meg egyből, amikor megcsinálja a várost).
- Időnként a világ idéz erőforrásokat, hogy ne foggyon ki belőlük.
- Csak 1 város lesz és minden ember eléri a többi ember által szerzett erőforrásokat.

Tesztek:

A tesztelés az SFML könyvtár nélkül fog történni így:

Az SFML könyvtár helyett csinállok egy billboard SFML könyvtárat, ami megvalósítja az SFML-ből használt metódusokat, classokat, viszont ezeknek a funkciójit megváltoztatja. A rajzolások és egyéb SFML működések helyett ezek a billboard megvalósítások csak kiírják, hogy milyen művelet történt. Példa a setPosition-ra: "setPosition meghívva az x és y pozícióra" fog kiíródni meghíváskor.



A jportára beadott változatba az én nem igazi SFML könyvtáram lesz beadva, így tesztelhetővé válik ott is.