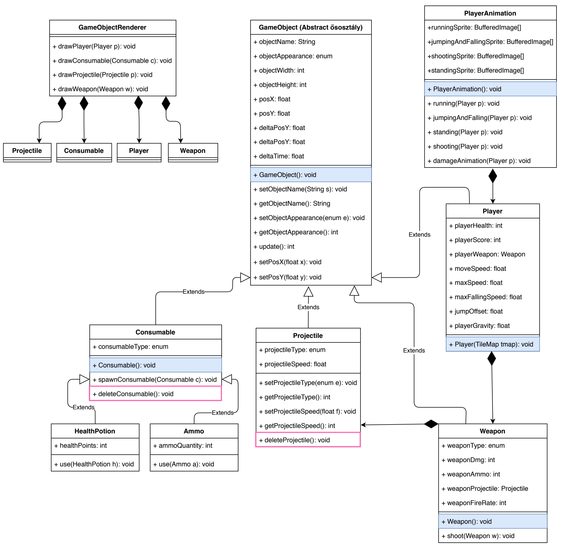
CLASS DIAGRAMS

I. A pályán elhelyezett objektumok

BM ZB BD



Konstruktorok

GameObject: A játékmenet során dinamikusan létrejövő objektumok ebből az abstract ősosztályból származnak. Az objektumok mérete (objectWidth, objectHeight), pozíciója (posX, posY), pozícióváltozása adott idő alatt (deltaPosY, deltaPosX, deltaTime) és az adott objektum sprite-ja (objectAppearance) a leszármazott osztályokban mind megtalálható. Az egyedi attribútumok és a logikát vezérlő metódusok a leszármazottakban kerülnek definiálásra.

GameObjectRenderer: A GameObject leszármazottjainak megjelenítésének megvalósításáért és kirajzolásáért felelős osztály.

Player: A játékosoknak van életerejük, pontszámuk, fegyverük, mozgási sebességük, esési sebességük, ugrási offszetük és hat rájuk a gravitáció. A Player egy TileMap-on létezik (azért szerepel ez a kostruktorban is), ahol a CollisionDetection-t meg kell majd valósítani.

PlayerAnimation: A játékosok sprite kezelése itt történik. Futó, eső / ugró, álló és lövő állapotokat különböztetünk meg. Az adott állapotban egy képsorozatot játszunk le (lehet 1 kép is). Sebzéskor a karakter színe vöröses árnyalatot kap.

Weapon: A weaponType az egyes fegyvertípusokat különbözteti meg (*kezdetben csak default kerül implementálásra*). A Weapon-nak van sebzése, tüzelési sebessége, aktuális töltényszáma.
A shoot() metódus indítja el a golyókat, amik ha eltalálnak valamit akkor eltűnnek a képernyőről, különben a mapon megtett távolság függvényében eltűnnek (*vagy idő alapján*).

Projectile: A lövedékeknek van típusa (fegyvertől függ, de jelenleg csak default típus) és sebessége.

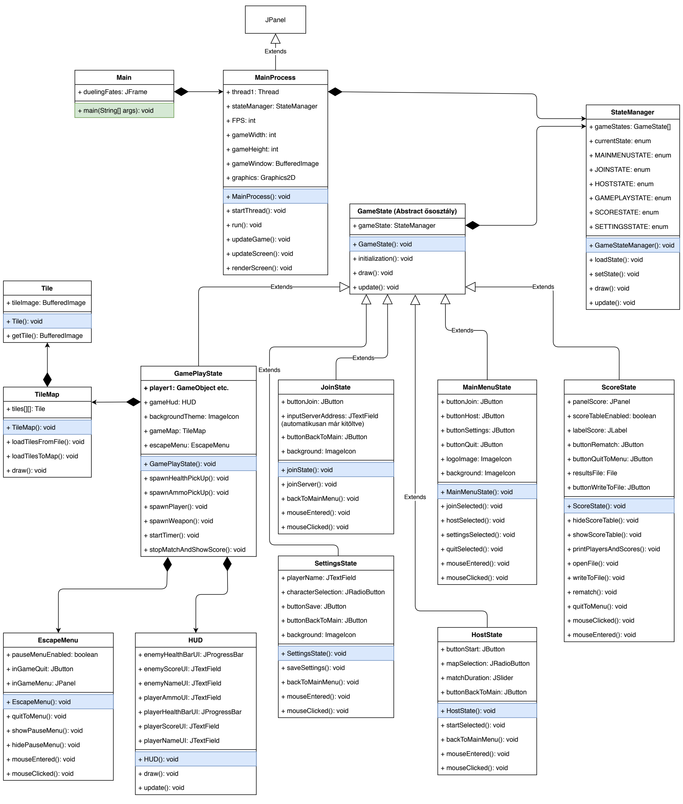
Consumables: A ConsumablesType változóban különböztetjük meg a HealthPickUp-ot és az AmmoPickUp-ot, melyek leszármaznak ebből az ősosztályból. Előbbi a használatakor (felvételekor) healthPoints értékkel növeli a Player aktuális életerejét, utóbbi pedig az ammoQuantity értékkel növeli a Player lőszerének mennyiségét. A "use()" metódusok a Playerrel való érintkezéskor hívódnak meg. (Ezután eltűnnek és a hatásuk végrehajtása megtörténik).

Megjegyzés, kérdések:

– Gombok lenymását hol kezeljük? A GamePlayState-ben? (a többi állapotban egérrel navigálunk)
– Kell foglalkoznunk az eltűntetett lövedékekkel, vagy a felvett itemek törlésével (amik már nincsenek a képernyőn)?

Audio implementálásával a fejlesztés végén foglalkozunk ha jut időnk rá, így nem szerepel a diagrammokban.

II. A rendszer felépítése



Konstruktorok
Start pont

Main: A játék futtatása innen indul (belépési pont), létrehozzuk az ablakot amiben futni fog a programunk és a MainProcess egy példányát.

MainProcess: A szálak létrehozása és elindítása, egy StateManager állapotgép példányosítása. A játék egy képként értelmezhető (BufferedImage), amire ki tudunk rajzolni (Graphics2D) elemeket. updateGame() a logika frissítése, updateScreen() a képen lévő elemek frissítése, majd renderScreen() ezek kirajzolása megadott képfrissítés (FPS szám) szerint.

StateManager: Állapotgép melyben definiáljuk az elérhető állapotokat tartalmazó GameState típusú elemekből álló tömböt, és az aktuális állapotot külön nyílvántartjuk. Az aktuális állapot frissítése – update(), kirajzolás – draw(), állapotváltás – setState() és betöltés – loadState() metódusokat használjuk.

GameState (Abstract): Az osztály konstruktora egy állapotgépet GameStateManagert hoz létre, ezáltal tudja használni azt. Az absztract metódusokat az egyes State-ekben külön-külön írjuk meg.

MainMenuState: A default főmenü, mely az indítás után megjelenik. Háttérkép, logo és gombok elhelyezése melyekkel a többi állapot elérhető. Navigálás és kattintás egérrel lehetséges.

JoinState: Ha a helyi hálózaton fut egy szerver, akkor tudunk csatlakozni az éppen futó Sessionhöz és a GameplayState-be lépünk. A "Back" gombbal visszatérünk a MainMenuState állapotba.

HostState: A "Start" gomb megnyomására az adott gép lesz a Server Host amihez a hálózaton lévő gépek tudnak csatlakozni, GamePlayState-be lépünk. Pályát lehet választani a felsorolt képek kiválasztásával. A játék időtartamát pedig egy slideren lehet beállítani (0-5 perc). A "Back" gombbal visszatérünk a MainMenuState állapotba.

SettingsState: Az Input mezőben a player nevét adhatjuk meg; a karakterválasztás a maphoz hasonlóan kiválaszthatló. A karakter a designon (Sprite kinézetén) kívül nem mutatkozik meg másban. A "Save" gomb megnyomásával a változtatások mentésre kerülnek. A "Back" gombbal visszatérünk a MainMenuState állapotba.

ScoreState: A játék végén jelenik meg egy JPanel, ahol lehetőségünk van újrajátszásra "Rematch", visszalépésre a menübe "QuitToMenu", és az eredmények kiírására egy fájlba. Ha már létezik az adott fájl akkor hozzáfűzéssel kerülnek rögzítésre az adatok (append).

HUD: A GamePlayState állapotban megjelenő "statikusan" elhelyezkedő objektumokat tartalmazza a képen (töltény ikon, mellette a töltények száma, aktuális idő, saját és ellenség pontszáma). Ez alól kivétel a karakterek neve és életerejük, mely mindig a mozgó játékosok felett helyezkednek el.

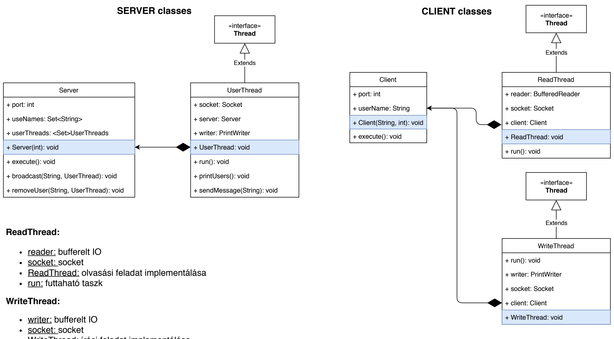
EscapeMenu: Lehetőséget nyújt a kilépésre, azzal hogy az "Esc" billentyűt lenyomva egy JPanel jelenik meg a képernyő közepén, melyen a "QuitToMenu" gombra kattintva a MainMenuState-be juthatunk vissza. Az "Esc" ismételt megnyomásával eltűnik az ablak.

Tile: Az egyes képek elhelyezése Tile típusú objektumokban.

TileMap: Tile-okat tartalmazó tömb feltöltése .txt fájlból. Beolvassuk a sort és oszlopot majd ezután az adott számoknak megfelelően a griden (*mapon*) elhelyezzük a megfelelő tile-okat.
A draw() végzi az elkészült map kirajzolását. A mapon csak statikus tile-ok lesznek.

GamePlayState: A tényleges játékmenet ebben az állapotban történik. TileMap és háttér betöltése, HUD és EscapeMenu objektum létrehozása, Timer indítása, Playerek spawnolása, Consumables (Ammo, Health) és erősebb Weapon spawnolása a pályán, lövedékek létrehozása a ShootWeapon() metóduson keresztül.
A játékosok élete csökken ha eltalálják őket, tüskéhez érnek és a pályáról leesve respawnolódnak (*pontlevonással (*ár, még kérdéses az implementáció*). A JProgressbar az életerő alapján frissül, a pontok pedig az okozott sebzés és ölés függvényében kerülnek meghatározásra.
Match leállítása a matchDuration eltelte után. Ekkor megjelenik a ScoreState és az ott megjelenő opciók közül választhatunk.

III. Networking



Konstruktorok

Server class:

- port: felhasznált port
- userNames: játékosok neve
- userThreads: kliens kiszolgálás párhuzamosan szálakban történik
- execute: init és listen() állapotig való eljutás
- broadcast: üzenet minden csatlakozott kliensnek
- removeUser: kliens szál terminálása

UserThread class:

- socket: socket
- server: server
- writer: bufferelt IO
- run: adott kliens kiszolgálása
- printUsers: csatlakozott kliensek printelése (debug funckió)
- sendMessage: üzenet küldés

Client class:

- port: felhasznált port
- userName: játékos neve
- execute: init és connect() állapotig eljutás

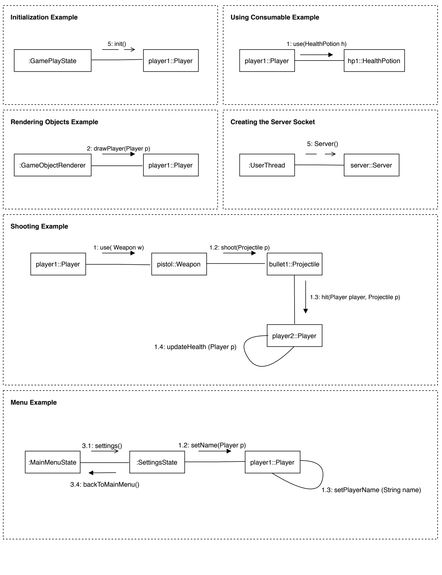
ReadThread:

- reader: bufferelt IO
- socket: socket
- ReadThread: olvasási feladat implementálása
- run: futtatható taszk

WriteThread:

- writer: bufferelt IO
- socket: socket
- WriteThread: írási feladat implementálása
- run: futtatható taszk

COMMUNICATION DIAGRAMS



SEQUENCE DIAGRAM