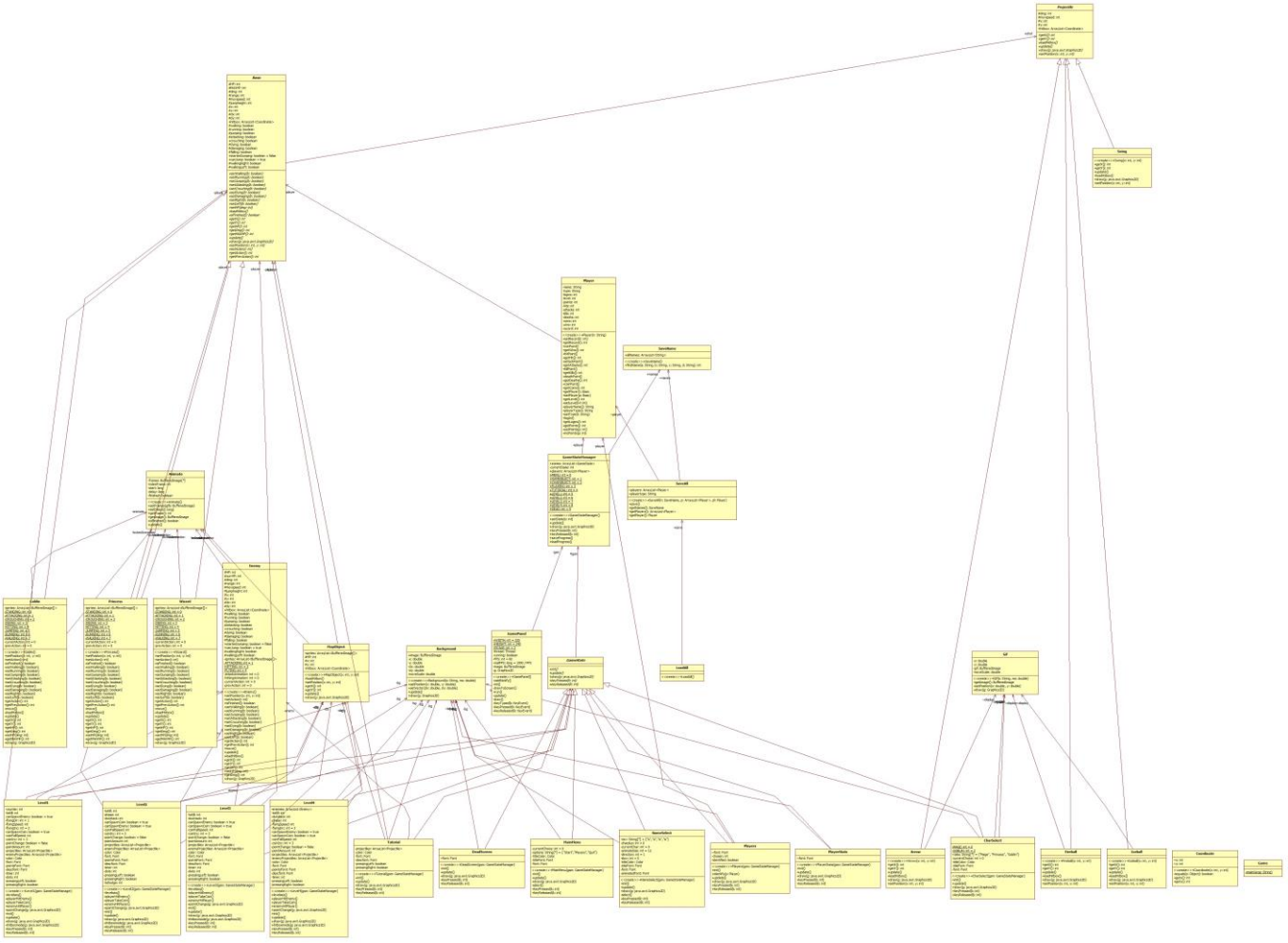
Dokumentáció

A házi feladat egy 2D játék, amiben egy választható karakterrel kell ellenségekkel harcolni. A játék java.awt és java.swing könyvtárat használ a megjelenítéshez és rajzoláshoz. Az adatok tárolásához miközben fut a program túlnyomó többségében ArrayList-et használ a tömbök tárolására. Ezenkívül a perzisztencia megoldására sima txt fájlformátumot használ. A játékablak egy JFramere épül, aminek egy saját ContentPane-t (GamePanel) csinálunk és erre rajzoljuk ki a játékot. Ez a GamePanel leszármazik a JPanel osztályból és megvalósítja a Runnable és KeyListener interface-t. Ez azért szükséges, hogy futhasson a játék és tudja a billentyűlenyomás eseményeket kezelni. A futást úgy biztosítjuk, hogy tartalmaz egy Thread-et a GamePanel-ünk amit elindítunk és miközben fut, egyfolytában vizsgáljuk az eltelt időt és ha lefutott az összes játékbeli folyamat, megnézzük, hogy mennyi idő telt el. Ezt időt levonjuk egy előre meghatározott célidőből ugyanis ahhoz, hogy konzisztensen fusson a játék, mindig azonos időintervallumokban kell lefuttatni a metódusokat. Tehát ami maradt idő, azt megvárjuk (Thread.sleep(maradt idő)). Mivel ezekkel már megvan az ablak, és ciklikusan futtatható a játék már csak a játékot kell megírni. A játékban lévő összes textúra leöltött, de ingyenesen használható licensz nélküli, nem általam csinált.

Az alábbi képen látható az egész osztálydiagram, WhiteStarUML-el legenerálva. Az egész kép kb. 3000x3000 pixeles, tehát lehet, hogy nem fog elférni itt a wordben de belinkelem imgur linken keresztül: https://imgur.com/a/HGHWf



Látható, hogy próbáltam objektumorientáltan megoldani a feladatot, így sok osztály alaposztályt biztosít a többinek. Kezdjük a megalkotásuk sorrendjében, tehát a már korábban említett GamePanel osztály ami középen helyezkedik el a képen, tartalmaz egy úgynevezett "GameStateManager" osztályt, ami a fő játékállapotkezelő osztály. A GamePanel osztálytól keresztül szinte az összes osztálynak van init(),update(),draw() metódusa, ami mindenhol ugyanazt a célt szolgálja:

- Init(): Egyszeri a lefutása általában, amikor először hozzáfordulunk az osztályhoz, akkor lefut és beállítja az alapértékeket vagy esetleg újratölti a textúrákat. A lényeg, hogy nem hívódik meg minden játékciklusban.

- Update(): Minden játékciklusban lefut, ha nincs más dolga az osztálynak, meghívja a tartalmazott osztályoknak az update() függvényét, amikor a végére ér az update()-lánc akkor ez általában egy-egy játékobjektumnál köt ki, amiknek az update() függvénye már tényleges frissítést kíván. Ilyen lehet például egy mozgás algoritmus, ha egy karakter halad egyenesen előre akkor minden frissítésben meg kell változatni a pozícióját. Ez a leggyakoribb használata az update()-nek de lehet még például egy-egy ütközés vizsgálata.
- Draw(): Minden játékciklusban lefut, szintén a rajzolási-lánc végéig megy, amint a nevéből következtetni lehet, kirajzol a képernyőre egy-egy képet. Például a feliratok kirajzolása, karakterek, animációk.

Tehát a GamePanel osztály tartalmez egy GameStateManager (gsm) osztályt, amelynek célja, hogy váltogassa a játék státuszát, illetve tárolja el a játékosok listáját és az éppeni játékost. Ez a gsm osztály eltárolja az összes GameState-et (játékállapotot). Tehát mindegyikből csinál egy példányt és ezek között váltakozik. A GameState egy absztakt osztály ami egy-egy játékállapotot reprezentál. Minden játékállapotnak van egy init(),update(),draw,keypressed,keyreleased függvénye amit meg kell valósítani. Az első három már meg lett magyarázva, annyi kiegészítéssel még, hogy a draw függvénynek van egy bemeneti paramétere: egy Graphics2D, azért hogy tudjuk mire kell rajzolni. A keypressed,keyreleased nyílván a user input kezelésére szolgál. Mint láthatjuk 6 konkrét játékállapot megvalósítás létezik, de mivel struktúráltan van megírva a program, ezért bármikor bővíthető további szintekkel a játék vagy más állapotokkal:

- MainMenu: A legelső játékállapot amit alapból betölt a játék, a főmenü. Itt látjuk az opciókat amiket választhatunk.
- NameSelect: Név választás. Azért fontos, mert ez alapján a 4 karakteres név alapján azonosítja be a játék az éppeni játékost. Ha nem találja a nevet a névnyilvántartásban, akkor új játékosként regisztrálja a nevével együtt.

- CharSelect: Karakterválasztás. Csak akkor választhat új karaktert a játékos ha ő egy új játékos. Egyébként a már kiválsztott karakterével kell játszania. Jelenleg 3 karakter közül választhat a játékos. Ezt már nehezebb bővíteni, mert teljesen más textúrájuk, játékmechanizmusuk van a karaktereknek. De lehetséges bővíteni. (Hiszen azt a 3-at is meg kellett valahogy csinálnom)
- Tutorial:Az úgymond legelső szint. De nincsenek ellenfelek, csak arra szolgál, hogy a játékos kitesztelje a mozgást és lássa hogy néz ki az egész játék.
- Level1: Itt már jönnek ellenségek, akik egy előre megírt algoritmus szerint mozognak. Illetve számolja a játék a pontokat is amiket a játékos pontjaihoz ad hozzá. (Vagy vonja le tőlük)
- Players: Az összes játékos listázása, kiválasztott játékosnak a részletes statisztikái.

Az utóbbi 2 játékállapotot csak nagyon alapszinten csináltam meg, mert azokat könnyű bővíteni és nem nagyon lényeges dolgok. Főleg a játékmenetet lehetne számtalan dologgal bővíteni, de nem az volt a célom, hogy egy teljesen élvezhető, kiadható játékot csináljak, hanem hogy megcsináljam a keretstruktúrát amivel működik az egész és könnyen bővíthető. Így is rengeteg befektetett munka van benne.

MainMenu: Ebben az állapotban a fel-le nyilak segítségével váltogathatunk a Start,Players,Quit között amik nevük szerint váltják a játékállapotot. Illeteve az 's' és 'l' billentyűzetgombokkal betöltöhetjük (load) és elmenthetjük (save) a játékbeli játékosokat. Folyamatosan vizsgáljuk a választást és ha ENTER-t nyom a játékos továbblépünk.

NameSelect: Hasonlóan egyfolytában vizsgáljuk a választott betűk értékeit miközben a fel-le, jobbra-balra nyílakkal állíthatja a felhasználó a választását. Az ENTER-el

továbblép az állapot de előtte megkeresi a karatereket összeillesztve név szerint a felhasználót (**Player**). Ha megtalálta, akkor betölti a gsm-be így majd hozzáférhetünk később.

CharSelect: Választhatunk a három karakter közül: **Wizard**, **Princess**, **Goblin**. Mind a három karaternek másak a tulajdonságai. Ezek alatt értendőek a mozgási sebesség, sebzés, életpont, támadási mód. Viszont nagyon sok közös van bennük, ezért mind a 3 egy közös anyaosztályból származik aminek a neve:

Basic: Ez a Basic osztály határozza meg meg az egyes karaktereknek a tulajdonságait, játéklogikáját. Mivel ez egy abstract osztály, ezért az összes metódust meg kell valósítani a konkrét altípusokban. Ilyenek például az egyes animációknak minden egyes kép betöltése, hitbox(**Coordinate**) számítása, mozgáslogika beállítása, stb. Ezenkívül ami az a sok változó az osztályban azok az egyes cselekvések beazonosítása: tehát azt mondja meg, hogy éppen mit csinál a karakter, ugyanis e-szerint kell az animációt(**Animate**) betölteni. Szintén azért fontosak ezek a változók, hogy a mozgáslogikát megcsináljuk amivel nagyon sok munkám volt. Ez azoknak a leírása és lekódolása, hogy miket csinálhat a karakter miközben éppen valamit már csinál. (Pl. Nem mozghat miközben támad, nem ugorhat miközben ugrás közben van, nem ugorhat amikor éppen esik lefele) Hozzá meg az animációk váltogatása, hogy viszonylag normálisan nézzen ki. (Ne töltse be újra az animációt ha lenyomva tartuk a jobbra nyílat, ugrás vagy esés közben ne legyen sétálási animáció de attól még tudjon mozogni esés és ugrás közben, stb..).

Wizard: A varázsló lassú mozgási sebesség, kevés élet, nagy sebzés jellemzi. Tűzbombát(**Fireball**) lő ki a támadás gombra. Korlátozott egyidejű lövések száma.

Princess: A hercegnő viszonylag gyors, sok élet, kis sebzés. Maga előtt Suhant(**Swing**) egyet a kardjával a támadás gombra. Korlátlan támadás szám.

Goblin: A goblin gyors, közepes élet, közepes sebzés. Egy nyílat(**Arrow**) lő ki a támadás gombra. Korlátozott lövések száma.

Projectile: Absztakt anyaosztály a lövedékek számára, ami lényeges, hogy ezeknek is van hitboxa, animációja nincs, csak egy kép és a **Swing** is lövedéknek számít, még ha logikailag nem túl egyértelmű. A lövedékeknek van sebessége, sebzése,x,y koordinátája és nagyjából más nem is kell a hitbox számításhoz. Azért származtatjuk a **Swing**-et is a lövedékből, mert így mindegy, hogy milyen karakterünk van, egy ArrayList<Projectile> tömbben el tudjuk tárolni és vizsgálni a lövedék állapotát.

Fireball: Lassú, sokat sebző lövedék. A pálya végéig repül.

Swing: Nagyon gyors, kis hatótáv, keveset sebző "lövedék".

Arrow: Nagyon gyors, pálya végéig megy, közepes sebzés.

Tehát ezek közül a karakterek közül lehet választani, ha új a játékos. Folyamatosan vizsgáljuk a választást a jobbra-balra nyílak alapján. Az ENTER-el továbblép a felhasználó.

Tutorial: Az első alkalom amikor a felhasználó láthatja a karakterét és mozoghat, támadhat vele. Itt már rendesen figyelni kell a billentyűlenyomásokat, elengedéseket hogy reagálni tudjon a program rá. Az ENTER-el kiléphetünk a MainMenu-be, a nyílakkal jobbra-balra sétálunk, ha lenyomva van a SHIFT, akkor sprintelünk, a felfele nyíllal ugrunk, a lefele nyíllal lehajolunk és a SPACE-el támad a karakterünk. El van tárolva az összes lövedék egy tömbben. Minden játékciklusban figyelni kell az ütközéseket. A korlátolt támadás számú karaktereknek megjelenik, hogy mennyi lőszert tudnak kilőni. A játékos életpontja is feltűnik, illetve egy érme(**MapObject**) és egy sárkány(**Enemy**). Ebből a játékállapotból úgy tud továbbjutni a játékos, ha eltalálja a sárkányt(**Enemy**) egy támadással.

Enemy: Az ellenséges sárkány, ezeket kell megölni a pályákon, azért nem származik a Basic-ből, mert nem játékos által irányított, így olyan függvényeket is meg kellett volna valósítani benne, amik egyáltalán nem szükségesek, sőt zavaróak is. (Ettől függetlenül maradtak benne feleseleges függvények meg változók, de nem merem őket kiszedni mert félek, hogy véletlen kitörlök egy szükséges függvényt és elrontom azt ami működik. Meg amúgy is ha valaki irányítani szeretné a sárkányt akkor az így megmaradt feleseleges függvényekkel könnyebben meg lehet írni azt is.) A sárkányt a gép irányítja egy megadott algoritmus alapján, ez most az hogy egyhelyben áll a Tutorialban és a Level1-en pedig fel-le repül és véletlenszerűen kilő egy-egy jéggolyót(**Iceball**). Tükrözött textúrák.

Iceball: Úgyanúgy a **Projectile**-ból származik mint a játékos által kilőtt lövedékek, viszont ő az ellenkező irányba repül és ezért tükrözve van.

MapObject: Ez a mi esetünkben egy sima érme, amit a játékos majd fel tud szedni, de lehetne bármi, például tüske vagy tényleg akármi. A lényeg, hogy nem csinál semmit vagy egyhelyben van, vagy esik lefele, de semmi mozgáslogika nincs benne. Hitboxa ugyanúgy van mint a többi objektumnak a pályán.

Level1: A fő játéklevel, jelenleg nincs több level, de kb. 5 perc alatt hozzá lehetne rakni egy másikat más mozgási algoritmusos ellenségekkel. Itt már számít a játékos életpontja, sima pontszáma és ebben az állapotban kalkulálódnak a statisztikák. A pálya abból áll, hogy sárkányok jönnek random időközönként, amikor éppen nincs folyamatban egy sárkány és ugyanúgy random időközönként esik egy-egy érme (bár más valószínűséggel). A játékosnak meg kell ölnie a sárkányokat, ami úgy lehetséges, hogy attól függően hogy milyen karakterrel va, eltalálja a saját lövedékével a sárkányt. De vigyáznia kell, mert a sárkány is lő random időpontokban egy-egy saját lövedéket amikeket ki kell kerülnie. Ha

bónusz pontokat szeretne szerezni a játékos, akkor el kell kapnia a leeső érméket a saját testével, nem tudja őket úgy felszedni, hogy lelövi őket. Minden játékciklusban vizsgálni kell:

- A játékos eltalálta-e az ellenséget
    - meghal-e az ellenség
- A játékos felszedte-e az érmét
- Az ellenség eltalálta-e a játékost
    - meghal-e a játékos
- Történt-e pontváltozás a játékos pontjában
    - Mínusz pont
    - Plussz pont
- El kell-e távolítani bármilyen lövedéket a pályáról
- Ha elért az ellenség a másik oldalra, el kell távolítani

A játék akkor ér véget, ha 5 sárkányt spawnolt a játék és mindegyik eltűnt. (Vagy megölte a játékos őket vagy átértek a pálya másik oldalára) Ekkor visszadob a főmenübe. Szintén visszadob a főmenübe és befejezi a játékot ha a játékos ENTER-t nyom.

Players: Az összes játékos megtekintése itt lehetséges, jelenleg nincsen megoldva, hogy ha túl sok játékos lenne nyílvántartva, akkor nem fér ki a képernyőre az összes, bár ahhoz elég sok játékos kéne kb 25-30 talán. Ha ezen belül van a játékosszám akkor kényelmesen böngészhetjük a statisztikákat. A fel-le nyílakkal választjuk ki a játékost a részletes statisztikáért, illetve onnan az ENTER gombbal lépünk vissza a nevekhez. Bármikor visszaléphetünk a főmenübe az ESC gombbal. Nyilván folyamatosan figyelni kell a választást és a választott játékost lekérdezni a gsm-től.

Egyéb fontos osztályok:

Player: A játékos osztály azonosítja be a játékost, tehát amit el kell tárolnia az benne van, ilyenek: Név,típus,bejelentkezések száma, melyik levelen van, pontok, támadások száma, találatok száma, halálok száma, ölések száma, maga a Karakter(**Basic**)

Ami nagyon fontos, hogy mentésnél, visszaöltésnél maga a Karakter ami(Basic) nem szerializálható mivel sok olyan osztály van benne ami nem szerializálható, ilyen például a BufferedImage[] tehát kénytelenek vagyunk ezt kihagyni a mentésből. Viszont minden játékosnak el kell tárolnia a karaktert különben hiányozni fognak az animációk, ezért van egy olyan String változója, hogy Type aminek az értéke "Mage","Princess" vagy "Goblin" tehát amint visszatöltjük a játékost ez alapján kell csinálni egy új Karaktert abból a típusból amilyen fajta a játékos. *Ezzel is nagyon sok időt dolgoztam mire rájöttem hogy oldjam meg.*

Coordinate: **int X és int Y párokat tárol.** Az egyes ütközéseket úgy detektálom, hogy minden dolognak van egy alapkoordinátája ahonnan a kép ki lesz rajzolva, viszont ez nagyon pontatlan lenne, hiszen például az egyes karakterek 64x64-es képből kb csak 30x30 tényleges pixelen vannak a többi teljesen átlátszó. Szóval minden egyes objektumnak kézileg be kellett állítani hogy mettől meddig számítson a hitbox-a. **Pl. A goblinnak: hitbox.x = goblin.x + 15; hitbox.y = goblin.y+20. Ez lesz az első koordináta-pár. Innen hozzá kell még adni azokat a koordinátákat amilyen széles és amilyen magas a goblin textúra. Ez kb. 30x30 tehát egy egymásbaágyazott for ciklussal belepakolom az összes ilyen koordináta-párt egyesével ami kb 900 koordináta párt fog eredményezni. Ez a goblin hitboxa, ha bármelyik koordináta egyezik a sárkány lövedékének bármelyik koordinátájával akkor az eltalálta a goblint. Ezeket az ütközéseket kell vizsgálni a játékban minden egyes ciklusban.**

Ezeken az osztályokon kívül vannak még segédosztályok, amik a fájlkezelésben segítenek legfőképpen, róluk néhány szó:

Animate: Az animációkat játsza le. Tartalmazza egy tömbben az összes képét egy-egy animációnak(BufferedImage[]). Be lehet állítani a lejátszásnak a gyorsaságát(Long). Az egész időzítő ugyanúgy működik, mint ahogy fut a program, tehát az eltelt időt egy-egy update között számolni kell és akkor kell váltani képet, ha az eltelt idő nagyobb mint a delay. Ha a végére értünk akkor pedig visszaállítani az indexet az elejére.

Background: Betölt egy egyszerű képet és meg lehet adni egy mozgási sebességet amivel el lesz tolva minden update()-ben a kép, így olyan érzést adva neki, mint ami mozog. Figyelni kell hogyha elfogy a kép akkor újrajzoljuk.

Gif: Ugyanaz mint az előző csak leegyszerűsítve mozgás nélkül.(Felesleges új osztálynak de ezzel akartam betölteni a .gif fájlokat ami nem sikerült viszont használom az osztályt másra ezért maradt)

SaveAll: Ez az osztály a leegyszerűsített gsm osztálynak az adatait tartalmazza. Azokat az adatokat amik szükségesek a visszatöltéshez. A konstruktorában megkapja a szükséges adatokat amiket beállít magának, majd egy saját save() metódussal kimenti őket a .txt fileba.

SaveName: Ebben az osztályban mentem le at összes nevet és csak nevet amik vannak a játékban. Azért, hogy ez alapján be tudjam azonosítani a játékost. Mivel akkor adjuk hozz a neveket amikor a játékost is a játékosokhoz, ezért ugyanolyan sorrendben lesznek eltárolva a nevek mint a játékosok, tehát elég ha visszaadjuk a keresett név indexét és tudjuk, hogy mi a játékosnak az indexe. Nem lenne fontos ez az osztály sem viszont így könnyebben tudjuk megkeresni a játékost.

LoadAll: Kicsit viccesen lett kialakítva a perzisztencia, mert ez a LoadAll osztály eltárol egy SaveAll osztályt amiben ugye benne vannak a szükséges adatok, tehát amikor egy

SaveAll-t lementünk, akkor egy SaveAll osztályt töltünk majd vissza, és ebből a SaveAll osztályból tudjuk majd kibányásznki az adatokat méghozzá úgy hivatkozva rá, hogy LoadAll.SaveAll.name, stb..

Tehát logikailag vannak benne fura dolgok, viszont ez az miatt van, mert még félév elején elkezdtem a házit és voltak dolgok amiket csak közben tudtam meg ezért át kellett volna írni nagyon sok mindent, ami iszonyat sok munka lett volna. Tehát inkább egy ad hoc módszerrel megoldottam sokmindent.

++++++++++++++++++++++++++++++++++Javítás++

A javított, hozzáadott verzijóa házi feladatnak tartalmazza: Use-case diagram, szekvenciadiagrammok, hozzáadott osztályok, minifixek. (+ JUnit a forrásfájloknál)

Minifixek: Több kis bugot fixeltem a játékban, például eredetileg ha nyomtad a jobbra nyílat és hirtelen elengedted és lenyomtad a balrát akkor kicsit megakadt a karakter, ez azért volt, mert a KeyRelease-re a játékos mozgása kikapcsolt, tökmindegy hogy a másik irány le volt nyomva, így csak pár ciklussal utána aktiválódott (ms) a mozgás. A hitboxoknak a javítása, hogy ne tűnjön úgy, mintha nem talált volna el valami, mégis eltalál + újított "hitboxmode" ami kirajzolja az összes objektumnak a teljes hitboxtartományát ha bekapcsoljuk (Eddig csak hozzávetőleges tartomány volt). Új feliratok, átmenet a levelek között, új statisztikák, rekordszámítás , stb...

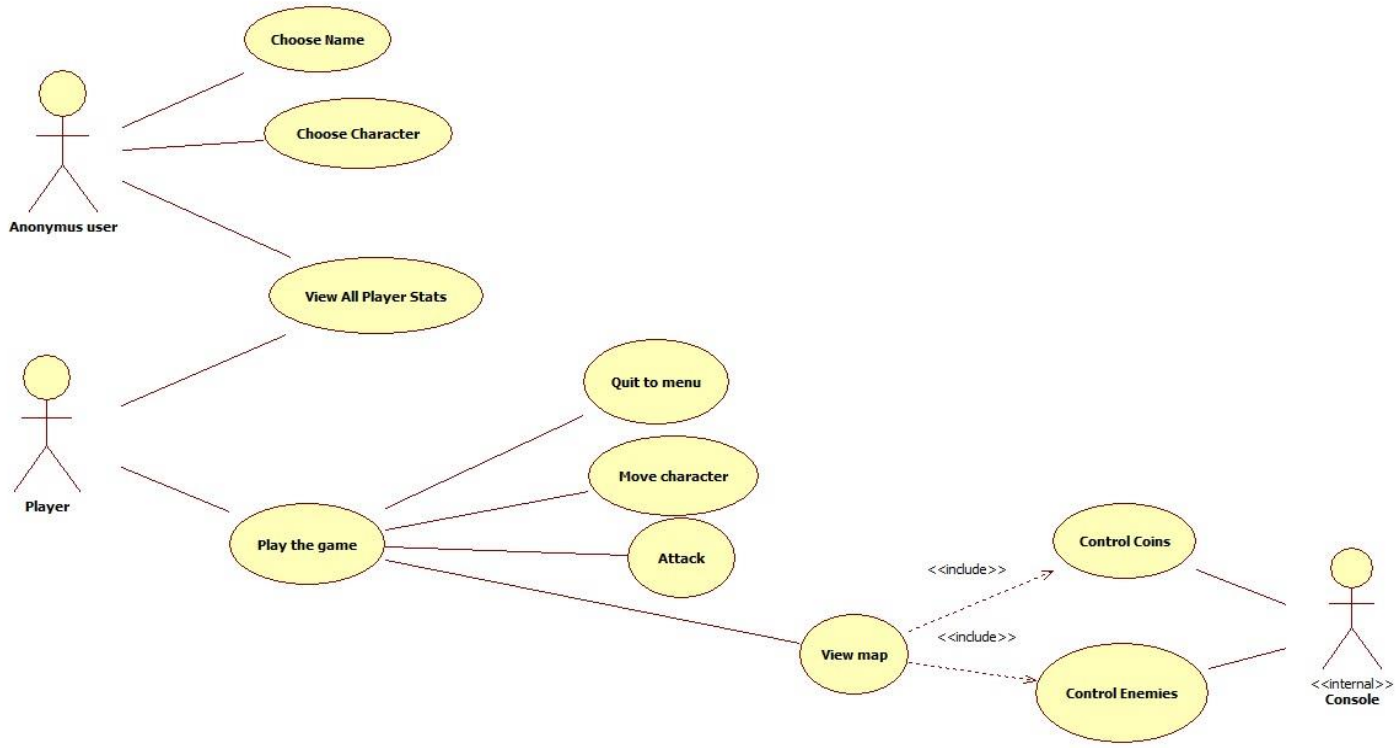
Level2: A hozzáadott level-eket csak nagyon tömören, mert az alapjuk ugyanaz mint a level1. A level2-n nincs más dolgunk mint leeső coinokat gyűjteni, miközben egyre nehezedő lövedékeket kell kikerülnünk.

Level3: A feladat a közeledő sárkányok megölése, vagy túlélés. Nincs globális range a lövedékeknek mert a sárkány a játékos útvonalát követi, így ha egyhelyben lőne a játékos egyszerűen megölhetné a sárkányt.

Level4: Utolsó szint, amin egyszerre nem 1, hanem 3 sárkányt kell megölnie a játékosnak, a sárkányok lassabbak mint az előző szinteken, viszont nem látni a hpjukat ezért sosem tudni mennyit kell még ütni beléjük. 3 fázis van ahol megnő a sárkányok lövésének a frekvenciája.

DeadScreen: A halotti képernyő, amin kiírjuk hogy melyik szinten halt meg a játékos és visszatérhetünk a főmenübe.

Use-case diagram:



Anon user: Az a felhasználó aki még nincs azonosítva a neve alapján

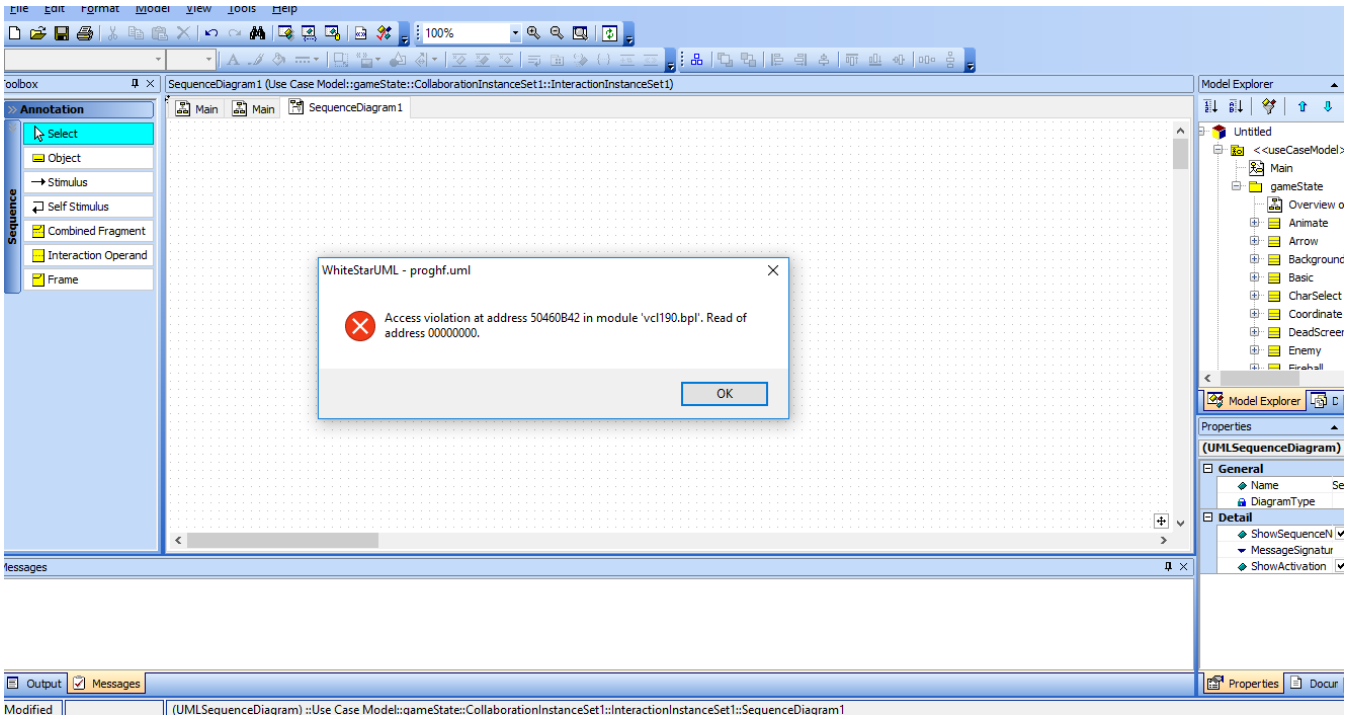
Player: Az a felhasználó aki be van azonosítva és játszhat.

Console: A belső logika szerint irányító. (Level)

A use-casek nevükből következendően adják magukat. A korábban említett játékállapotoknál részletesen le van írva hogyan lehet eljutni a különböző use-caseek-hez.

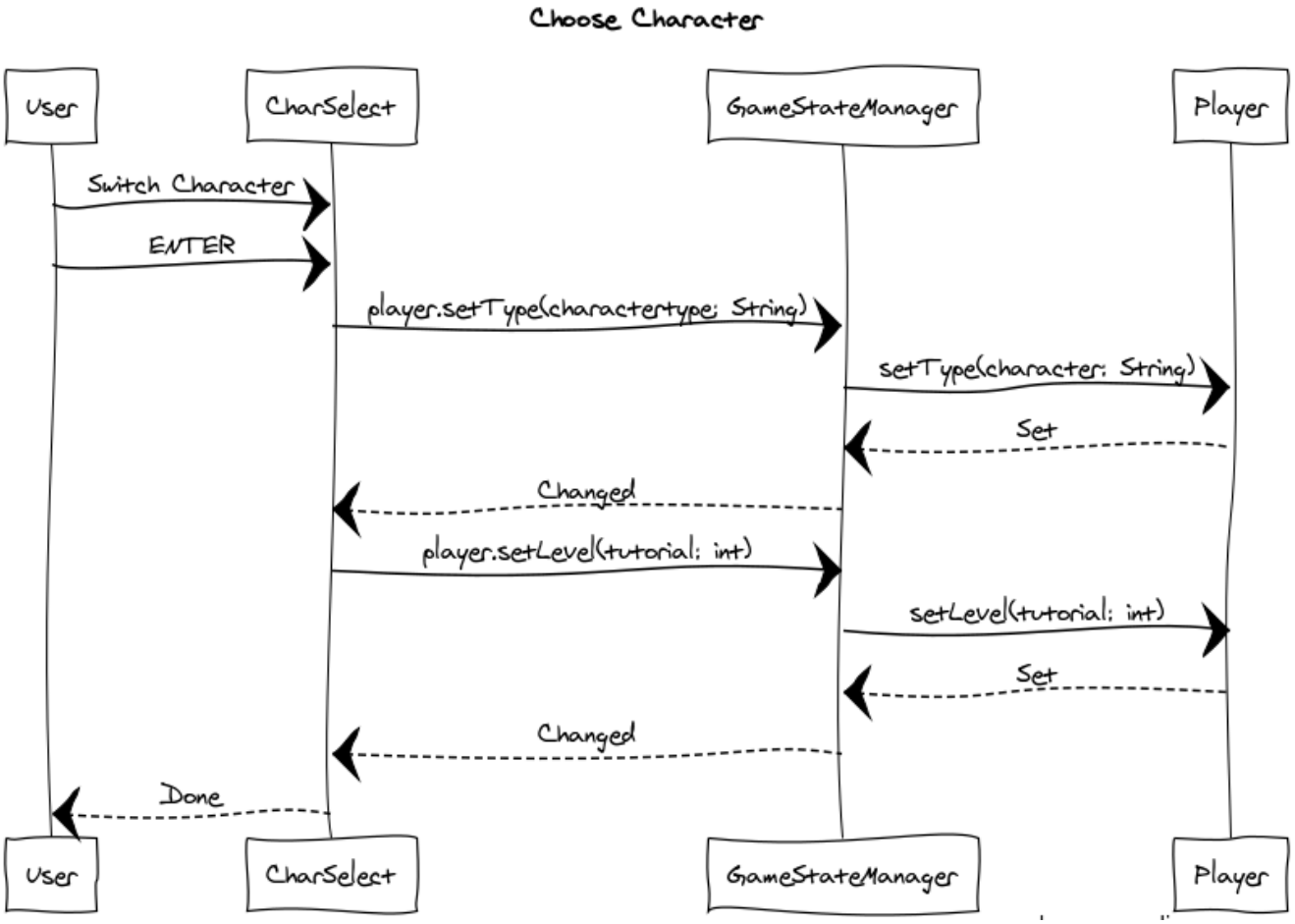
Szekvenciadiagrammok:

A WhiteStarUML-el:

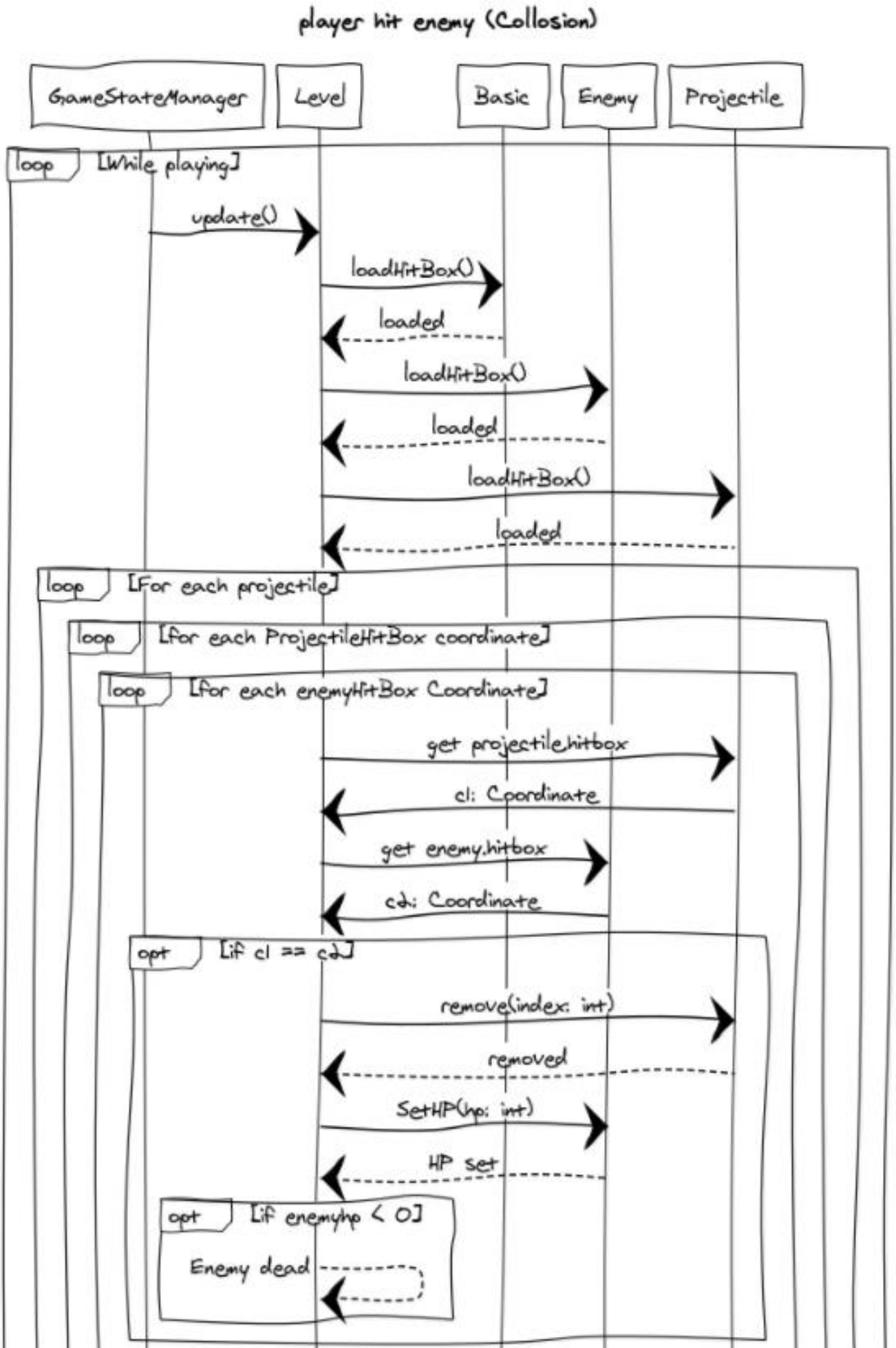


https://www.websequencediagrams.com/:

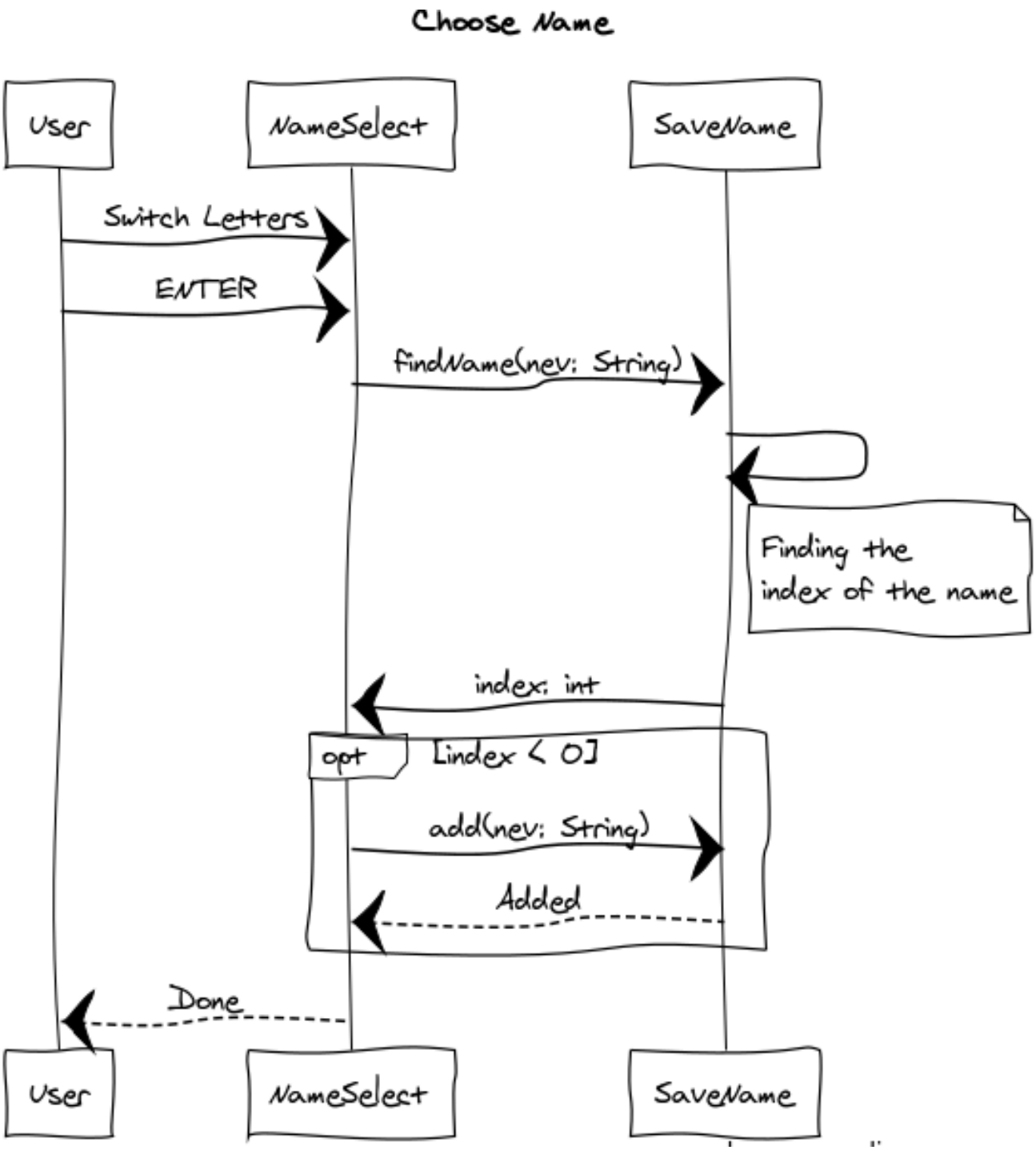
Choose Character



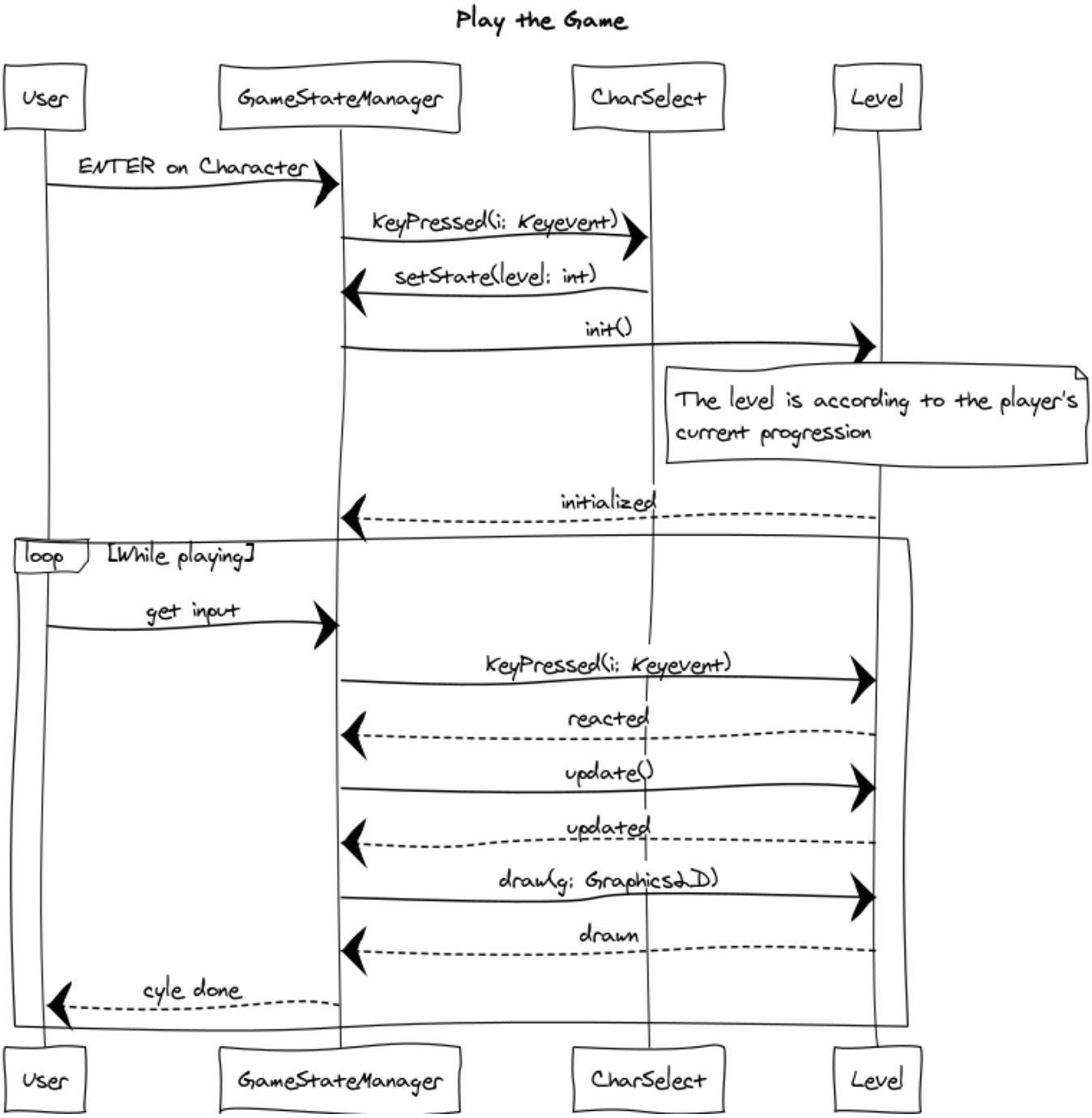
Player Hit Enemy


player hit enemy (Collosion)
GameStateManager
Level
Basic
Enemy
Projectile
loop [While playing]
update()
loadHitBox()
loaded
loadHitBox()
loaded
loadHitBox()
loaded
loop [For each projectile]
loop [for each ProjectileHitBox coordinate]
loop [for each enemyHitBox Coordinate]
get projectile.hitbox
c1: Coordinate
get enemy.hitbox
c2: Coordinate
opt [if c1 == c2]
remove(index: int)
removed
SetHP(hp: int)
HP set
opt [if enemyhp < 0]
Enemy dead

Choose Name

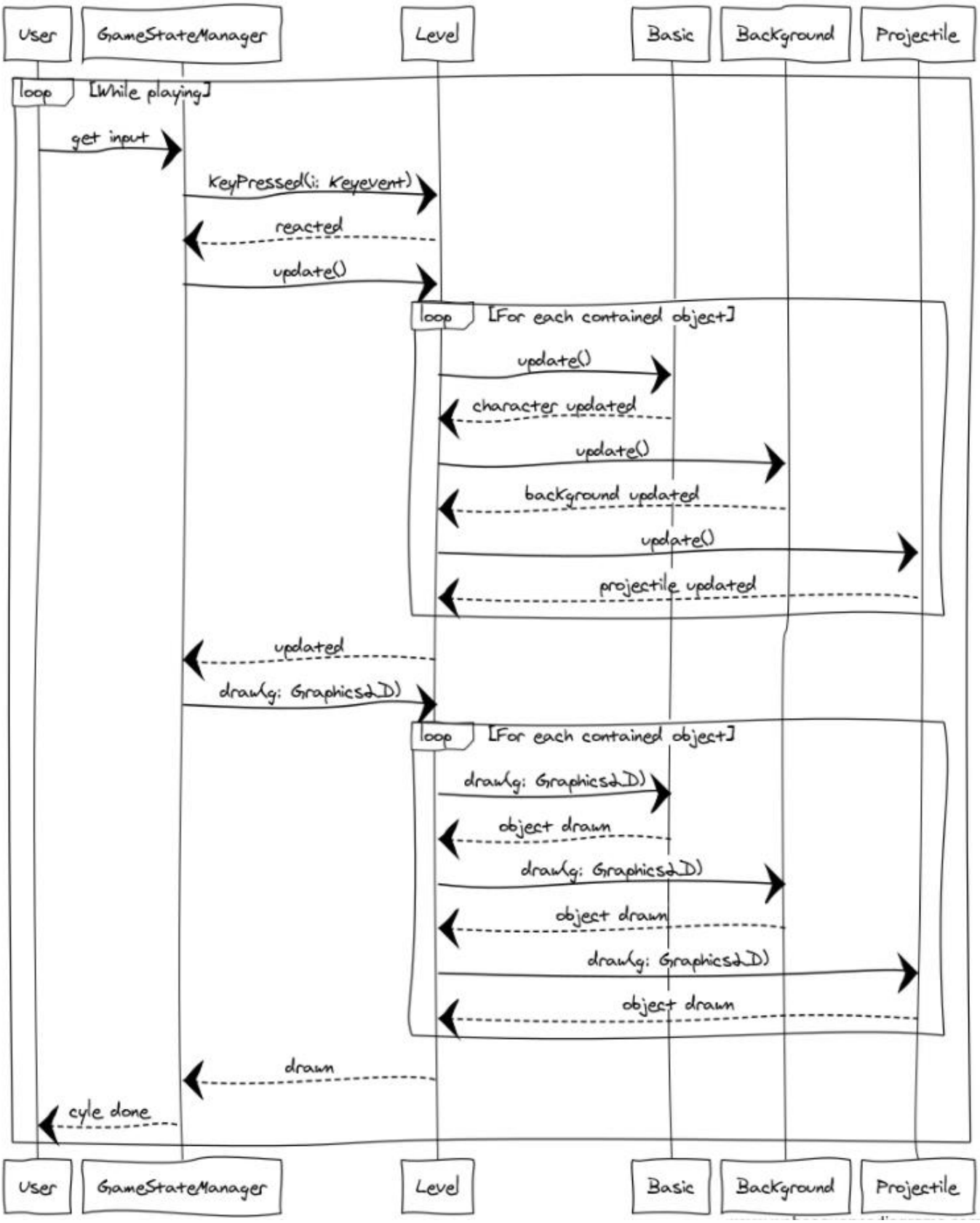


Play Game

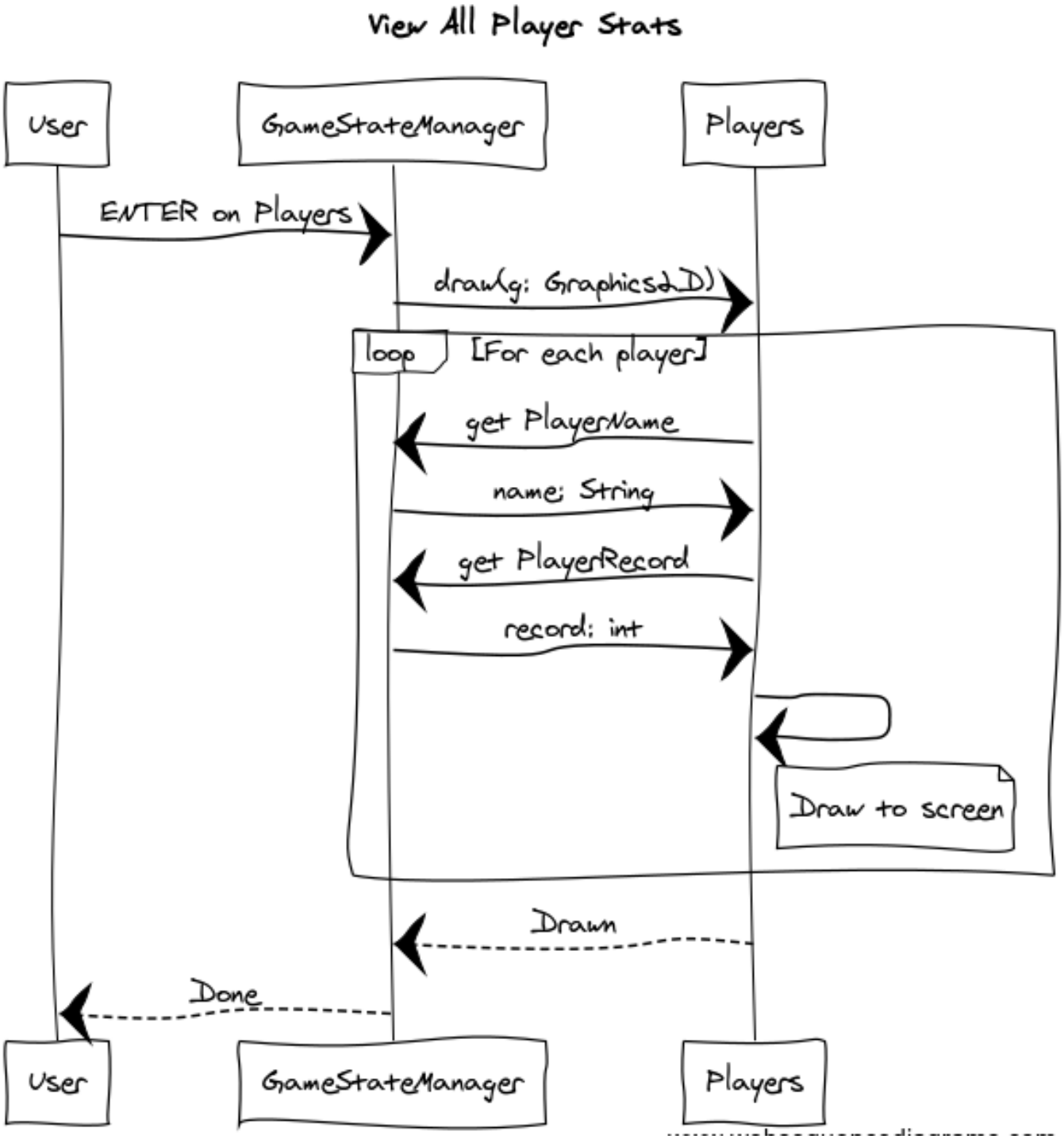

Play the Game
User
GameStateManager
CharSelect
Level
ENTER on Character
keyPressed(i: Keyevent)
setState(level: int)
init()
The level is according to the player's current progression
initialized
loop
[While playing]
get input
keyPressed(i: Keyevent)
reacted
update()
updated
draw(g: Graphics2D)
drawn
cyle done
User
GameStateManager
CharSelect
Level

View Map

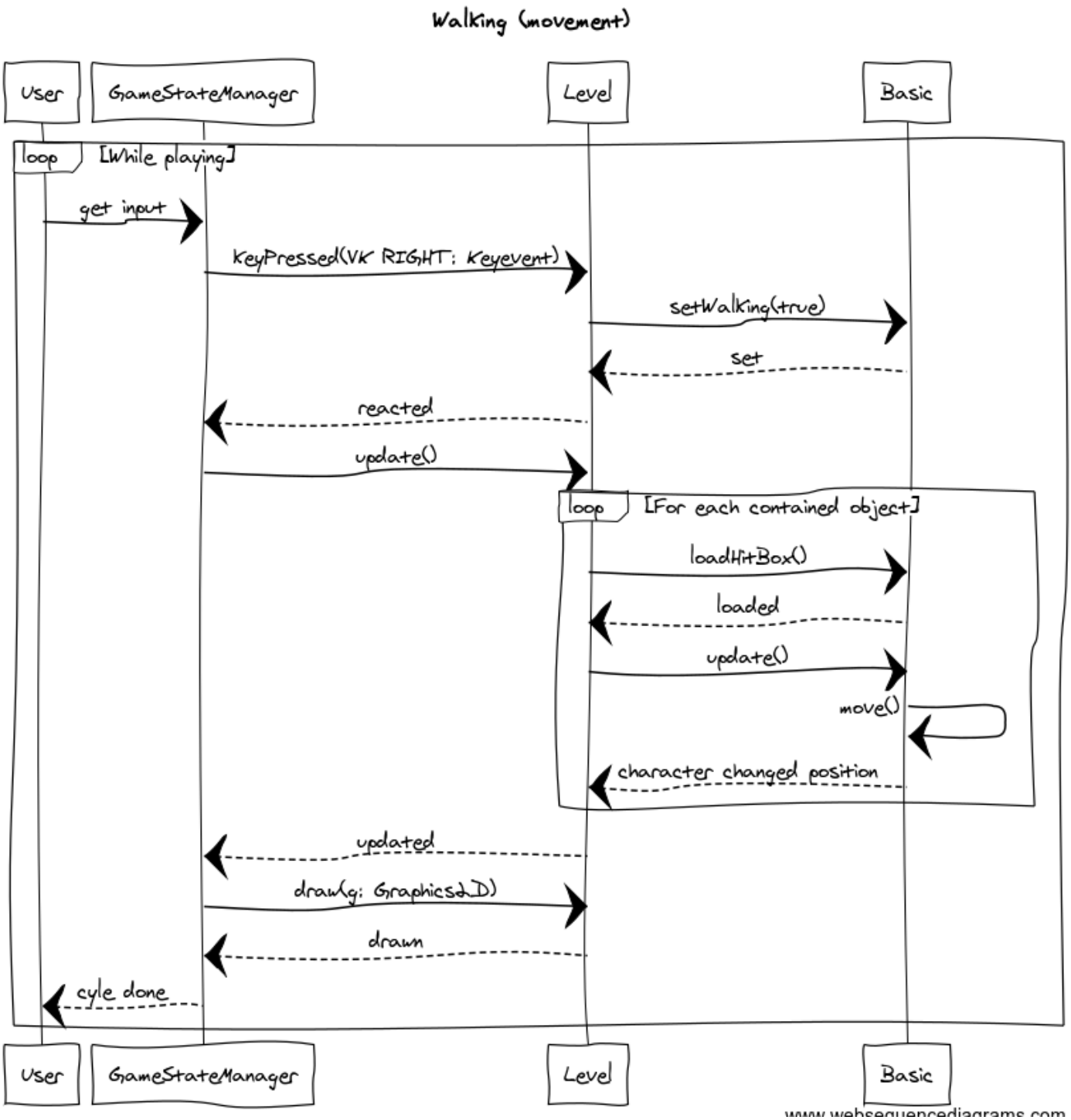
View map


User
GameStateManager
Level
Basic
Background
Projectile
loop
[While playing]
get input
keyPressed(i: Keyevent)
reacted
update()
loop
[For each contained object]
update()
character updated
update()
background updated
update()
projectile updated
updated
draw(g: Graphics2D)
loop
[For each contained object]
draw(g: Graphics2D)
object drawn
draw(g: Graphics2D)
object drawn
draw(g: Graphics2D)
object drawn
drawn
cyle done
User
GameStateManager
Level
Basic
Background
Projectile

View Other Player Stats



Player Moving


Walking (movement)
User
GameStateManager
Level
Basic
loop
[While playing]
get input
keyPressed(VK_RIGHT: Keyevent)
setWalking(true)
set
reacted
update()
loop
[For each contained object]
loadHitBox()
loaded
update()
move()
character changed position
updated
draw(g: Graphics2D)
drawn
cyle done
User
GameStateManager
Level
Basic
www.websequencediagrams.com